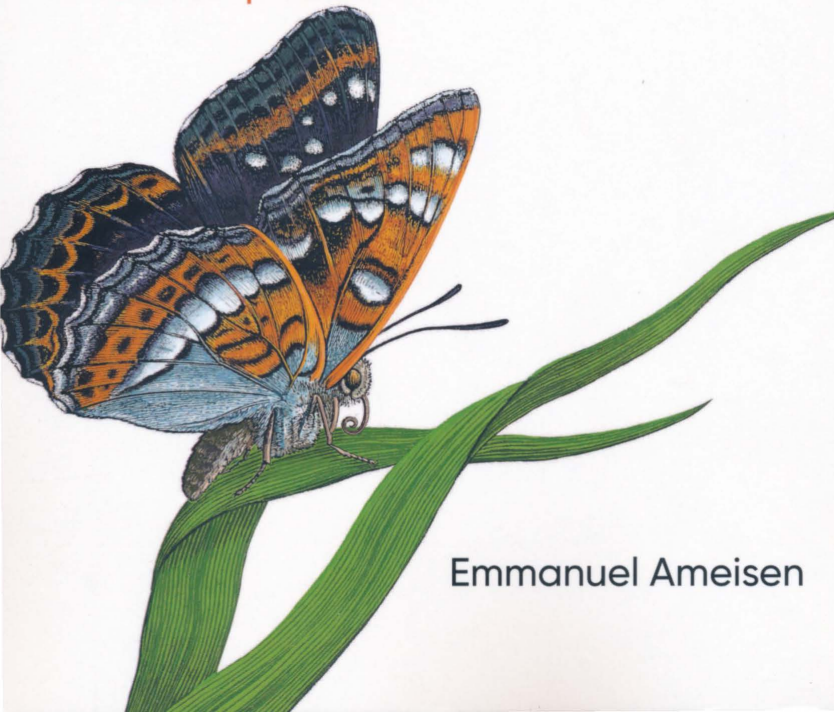
O'REILLY®

Développer des applications machine learning

De l'idée au produit fini



Emmanuel Ameisen

Développer des applications machine learning

Emmanuel Ameisen

Table des matières

Introduction

Pourquoi utiliser des applications propulsées par le Machine Learning

Au cours de la dernière décennie, l'apprentissage automatique, autrement dit le *machine learning* (ou ML, un abrégé que nous utiliserons couramment dans ce livre), a été de plus en plus utilisé pour alimenter une variété de produits tels que des systèmes d'assistance automatisés, des services de traduction, des moteurs de recommandation, des modèles de détection des fraudes, et bien d'autres champs d'application.

Étonnamment, il n'existe pas beaucoup de ressources pour enseigner aux ingénieurs et aux scientifiques comment construire de tels produits. De nombreux livres et cours expliquent comment entraîner des modèles de ML ou comment construire des projets logiciels, mais peu d'entre eux mélangent les deux mondes pour enseigner comment construire des applications pratiques qui sont propulsées par le ML.

Déployer le ML dans le cadre d'une application exige un mélange de créativité, de solides pratiques d'ingénierie et d'esprit analytique. Les produits ML sont notoirement difficiles à construire car ils nécessitent beaucoup plus que le simple entraînement d'un modèle sur un jeu de données. Choisir la bonne approche ML pour une caractéristique donnée, analyser les erreurs d'un modèle et les problèmes de qualité des données, et valider les résultats du modèle pour garantir la qualité du produit sont autant de problèmes difficiles qui sont au cœur du processus de construction d'applications d'apprentissage automatique.

Ce livre passe en revue toutes les étapes de ce processus et a pour but de vous aider à les réaliser en partageant un mélange de méthodes, d'exemples de codes et de conseils venant aussi bien de ma part que de celle d'autres praticiens expérimentés. Nous aborderons les compétences pratiques requises pour concevoir, construire et déployer des applications utilisant la technologie ML. L'objectif de ce livre est de vous aider à réussir chaque étape du processus de ML.

Utiliser le ML pour construire des applications pratiques

Si vous lisez régulièrement les articles sur l'apprentissage automatique ou encore des blogs d'ingénierie d'entreprise, vous vous sentirez peut-être dépassé par la combinaison d'équations d'algèbre linéaire et de termes d'ingénierie. La nature hybride de ce champ conduit de nombreux ingénieurs et scientifiques qui pourraient apporter leur expertise variée à se sentir intimidés par le domaine du ML. De même, les entrepreneurs et les responsables produit ont souvent du mal à faire le lien entre leurs idées pour l'entreprise et ce qu'il est possible de réaliser aujourd'hui avec le ML (et ce qui pourrait être possible demain).

Ce livre couvre les leçons que j'ai apprises en travaillant au sein d'équipes chargées des données dans plusieurs entreprises et en aidant des centaines de spécialistes des données, d'ingénieurs logiciel et de chefs de produit à construire des projets de ML appliqué grâce à mon travail de direction du programme d'intelligence artificielle chez Insight Data Science.

L'objectif de ce livre est de partager un guide pratique, étape par étape, pour la construction d'applications alimentées, propulsées par le ML. C'est un ouvrage pratique qui se concentre sur des conseils et des méthodes concrètes pour vous aider à prototyper, à itérer et à déployer des modèles. Comme il couvre un large éventail de sujets, nous n'entrerons dans les détails qu'autant qu'il est nécessaire à chaque étape. Dans la mesure du possible, je vous fournirai des ressources pour vous aider à approfondir les sujets abordés si vous le souhaitez.

Les concepts importants sont illustrés par des exemples pratiques, dont une étude de cas qui passera de l'idée au modèle déployé à la fin du livre. La plupart des exemples seront accompagnés d'illustrations, et beaucoup contiendront du code. Tout le code utilisé dans ce livre peut être trouvé dans le dépôt GitHub qui l'accompagne.

Comme ce livre se concentre sur la description du processus de ML, chaque chapitre s'appuie sur des concepts définis dans les précédents. C'est pourquoi je vous recommande de le lire dans l'ordre afin de comprendre comment chaque étape successive s'inscrit dans le processus global. Si vous cherchez à explorer un sous-ensemble du processus de ML, vous serez peut-être mieux servi par un livre plus spécialisé. Si c'est le cas, je vous ferai quelques recommandations.

Ressources supplémentaires

- Si vous souhaitez connaître suffisamment bien le ML pour écrire vos propres algorithmes à partir de zéro, je vous recommande Data Science from Scratch, de Joel Grus (`https://bit.ly/3b4C346`). Si c'est la théorie du deep learning qui vous intéresse, le manuel Deep Learning (MIT Press), de Ian Goodfellow, Yoshua Bengio et Aaron Courville, est une ressource de référence (`https://www.deeplearningbook.org/`).

- Si vous vous demandez comment entraîner des modèles de manière efficace et précise sur des jeux de données spécifiques, Kaggle (`https://www.kaggle.com/`) et fast.ai (`https://fast.ai/` ou bien `https://github.com/fastai`) sont d'excellents sites à visiter.
- Si vous souhaitez apprendre à construire des applications évolutives qui doivent traiter de grandes quantités de données, je vous recommande le livre Designing Data-Intensive Applications (O'Reilly), de Martin Kleppmann (`https://bit.ly/2Vm7os5`).

Si vous avez de l'expérience en matière de codage ainsi que quelques connaissances de base dans ce domaine et que vous souhaitiez construire des produits axés sur le ML, ce livre vous guidera tout au long du processus, en allant de l'idée du produit au prototype final. Si vous travaillez déjà en tant qu'informaticien ou ingénieur ML, ce livre vous permettra d'ajouter de nouvelles techniques à vos outils de développement. Si vous ne savez pas comment coder mais que vous collaborez avec des spécialistes des données, ce livre peut vous aider à comprendre les processus d'apprentissage automatique, à condition que vous soyez prêt à sauter certains des exemples de code les plus complexes.

Commençons par approfondir la signification du ML d'un point de vue pratique.

Le ML en pratique

Pour les besoins de cette introduction, considérez l'apprentissage automatique, le ML donc, comme le processus consistant à exploiter des motifs dans des données pour ajuster automatiquement les algorithmes. Il s'agit d'une définition générale, et vous ne serez donc pas surpris d'apprendre que de nombreuses applications, outils et services commencent à intégrer le ML au cœur de leur fonctionnement.

Certaines de ces tâches sont orientées vers l'utilisateur, comme les moteurs de recherche, les recommandations sur les plateformes sociales, les services de traduction ou les systèmes qui détectent automatiquement les visages familiers dans les photographies ; elles suivent les instructions émises *via* des commandes vocales ou encore tentent de fournir des suggestions utiles pour terminer une phrase dans un email.

Certaines fonctionnent de manière moins visible, en filtrant silencieusement les courriers électroniques non sollicités (les spams) et les comptes frauduleux, en diffusant des publicités, en prédisant les futurs modes d'utilisation afin d'allouer efficacement les ressources, ou en expérimentant la personnalisation de l'expérience sur un site Web pour chaque utilisateur.

De nombreux produits tirent actuellement parti du ML, et bien d'autres encore pourraient le faire. L'idée consiste à identifier les problèmes pratiques qui pourraient bénéficier du ML, et à trouver une solution efficace à ces problèmes. Passer d'un objectif de produit de haut niveau à des résultats obtenus grâce au ML est une tâche difficile que ce livre tente de vous aider à accomplir.

Certains cours de ML vont enseigner aux étudiants les méthodes du ML en leur fournissant un jeu de données et en leur faisant entraîner un modèle sur celles-ci, mais l'entraînement d'un algorithme sur un jeu de données n'est qu'une petite partie du processus de ML. Les produits convaincants ne sont pas seulement le résultat d'un score d'exactitude global, mais aussi le résultat d'un long processus. Ce livre commence par l'idéation et se poursuit jusqu'à la production, en illustrant chaque étape par un exemple d'application. Nous partagerons les outils, les meilleures pratiques et les pièges courants appris en travaillant avec des équipes qui déploient ce genre de systèmes chaque jour.

Ce que couvre ce livre

Pour traiter de la question de la construction d'applications alimentées par le ML, ce livre se concentre sur le concret et la pratique. En particulier, il vise à illustrer l'ensemble du processus de construction d'applications s'appuyant sur le ML.

Pour ce faire, je commencerai par décrire les méthodes permettant d'aborder chaque étape du processus. Ensuite, j'illustrerai ces méthodes à l'aide d'un exemple de projet servant d'étude de cas. Le livre contient également de nombreux exemples pratiques d'utilisation du ML dans l'industrie et propose des entretiens avec des professionnels qui ont construit et géré des modèles ML de production.

Globalité du processus de ML

Pour fournir avec succès un produit de ML aux utilisateurs, il faut faire plus qu'entraîner un modèle. Vous devez *traduire* de manière réfléchie les besoins du produit en un problème de ML, *recueillir* des données adéquates, *itérer* efficacement entre les modèles, *valider* vos résultats et les *déployer* de manière robuste.

La construction d'un modèle ne représente souvent qu'un dixième de la charge de travail totale d'un projet de ML. La maîtrise de l'ensemble du pipeline de ML est cruciale pour construire des projets avec succès, réussir les entretiens de ML, et être un contributeur de premier plan au sein des équipes de ML.

Une étude de cas technique et pratique

Nous ne réimplémenterons pas d'algorithmes en C à partir de zéro, mais nous resterons pratiques et techniques en utilisant des bibliothèques et des outils fournissant des abstractions de plus haut niveau. Dans ce livre, nous allons construire ensemble un exemple d'application de ML, de l'idée initiale au déploiement du produit.

J'illustrerai les concepts clés par des extraits de code, le cas échéant, ainsi que par des figures décrivant notre application. La meilleure façon d'apprendre le ML est de le pratiquer. Je vous

encourage donc à parcourir le livre en reproduisant les exemples et en les adaptant pour construire votre propre application utilisant le ML.

Des applications du mode réel

Tout au long de ce livre, j'inclurai des conversations et des conseils de responsables qui ont travaillé dans des équipes chargées des données au sein d'entreprises technologiques telles que StitchFix, Jawbone et FigureEight. Ces discussions porteront sur des conseils pratiques recueillis après avoir construit des applications de ML concernant des millions d'utilisateurs et corrigeront certaines idées fausses courantes sur ce qui fait le succès des *data scientists* et des équipes travaillant dans le champ de la science des données.

Prérequis

Ce livre suppose une certaine familiarité avec la programmation. J'utiliserai principalement Python pour les exemples techniques et je suppose que le lecteur est familier avec sa syntaxe. Si vous souhaitez rafraîchir vos connaissances en Python, je vous recommande *The Hitchhiker's Guide to Python* (O'Reilly), de Kenneth Reitz et Tanya Schlusser (`https://bit.ly/2RvjttV`).

En outre, si je vais définir la plupart des concepts de ML mentionnés dans le livre, je ne vais pas couvrir les rouages internes de tous les algorithmes utilisés. La plupart de ces algorithmes sont des méthodes standard qui sont couvertes dans de multiples ressources de ML, comme celles mentionnées plus haut à la rubrique « Ressources supplémentaires ».

Notre étude de cas : rédaction assistée par ML

Pour illustrer concrètement cette idée, nous allons construire ensemble une application de ML au fur et à mesure que nous progresserons dans ce livre.

Pour cette étude de cas, j'ai choisi une application qui peut illustrer avec précision la complexité de l'itération et du déploiement des modèles de ML. Je voulais également couvrir un produit qui puisse produire de la valeur. C'est pourquoi nous allons mettre en place un *assistant de rédaction propulsé par le ML*.

Notre objectif est de construire un système qui aidera les utilisateurs à mieux écrire. En particulier, nous visons à aider les gens à mieux rédiger leurs questions. Cet objectif peut sembler très vague, et je le définirai plus clairement au fur et à mesure que nous avancerons dans le projet, mais c'est un bon exemple pour quelques raisons essentielles.

Les données textuelles sont partout

> Les données textuelles sont abondamment disponibles pour la plupart des cas d'utilisation auxquels vous pouvez penser, et sont au cœur de nombreuses applications pratiques du ML. Que nous essayions de mieux comprendre les critiques de notre produit, de clas-

sifier avec exactitude les demandes d'assistance reçues ou encore d'adapter nos messages promotionnels à des publics potentiels, nous consommons et produisons des données textuelles.

Les assistants de rédaction sont utiles

De la fonction de prédiction de texte de Gmail au correcteur orthographique intelligent de Grammarly, les éditeurs basés sur le ML ont prouvé qu'ils pouvaient apporter de la valeur aux utilisateurs, et ce de diverses manières. Il est donc particulièrement intéressant pour nous d'explorer la manière de les construire en partant de zéro.

L'écriture assistée par ML se suffit à elle-même

De nombreuses applications de ML ne peuvent fonctionner que si elles sont étroitement intégrées dans un écosystème plus large, comme les prévisions d'ETA pour les entreprises de covoiturage (*Estimated Time of Arrival*, ou heure d'arrivée estimée), les systèmes de recherche et de recommandation pour les détaillants en ligne ou encore les modèles d'enchères. Un éditeur de texte, cependant, même s'il pourrait bénéficier d'une intégration dans un écosystème d'édition de documents, peut s'avérer précieux à lui seul et être exposé par le biais d'un simple site Web.

Tout au long du livre, ce projet nous permettra de mettre en évidence les défis et les solutions associées que nous proposons pour construire des applications basées sur le ML.

Le processus du ML

Le chemin qui mène d'une idée à une application de ML déployée est long et sinueux. Après avoir vu de nombreuses entreprises et personnes construire de tels projets, j'ai identifié quatre étapes clés successives, qui seront chacune couvertes dans une section de ce livre.

1. *Identifier la bonne approche en matière de ML* : le domaine du ML est vaste et propose souvent une multitude de moyens pour atteindre un objectif pour un produit donné. La meilleure approche pour un certain problème dépendra de nombreux facteurs tels que les critères de réussite, la disponibilité des données et la complexité des tâches. Les objectifs de cette étape sont de définir les bons critères de réussite et d'identifier un jeu de données initial adéquat ainsi que choisir un modèle.
2. *Construire un prototype initial* : commencez par construire un prototype de bout en bout avant de travailler sur un modèle. Ce prototype devrait viser à atteindre l'objectif correspondant au produit sans faire appel au ML, et il vous permettra de déterminer la meilleure façon d'appliquer les techniques de ML. Une fois le prototype construit, vous devriez avoir une idée quant à la nécessité de faire appel au ML, et vous devriez pouvoir commencer à rassembler un jeu de données pour entraîner un modèle.

3. *Itérer sur les modèles :* maintenant que vous disposez d'un jeu de données, vous pouvez entraîner un modèle et évaluer ses défauts. L'objectif de cette étape est d'alterner de manière répétée entre l'analyse des erreurs et l'implémentation. Augmenter la vitesse à laquelle cette boucle d'itérations se déroule est le meilleur moyen d'augmenter la vitesse de développement du ML.
4. *Déployer et assurer le suivi :* lorsqu'un modèle montre de bonnes performances, vous devez choisir une option de déploiement adéquate. Une fois déployés, les modèles échouent souvent de manière inattendue. Les deux derniers chapitres de cet ouvrage traitent des méthodes permettant d'atténuer et de surveiller les erreurs des modèles.

Il y a beaucoup de chemin à parcourir. Plongeons donc tout de suite et commençons par le Chapitre 1 !

Conventions utilisées dans ce livre

Les conventions typographiques suivantes sont utilisées dans ce livre :

Italique

Indique les nouveaux termes, les noms de fichiers et les extensions de fichiers, voire les adresses électroniques.

`Largeur constante`

Utilisée pour les listings des programmes, ainsi que dans le corps du texte pour faire référence aux éléments de programme tels que les noms de variables ou de fonctions, les bases de données, les types de données, les variables d'environnement, les déclarations et les mots-clés ainsi que pour les URL.

`Largeur constante en gras`

Affiche les commandes ou autres textes qui doivent être tapés littéralement par l'utilisateur.

`Largeur constante en italique`

Affiche le texte qui doit être remplacé par des valeurs fournies par l'utilisateur ou déterminées par le contexte.

Cet élément indique une note générale.

Utilisation des exemples de code

Les exemples de code pour ce livre peuvent être téléchargés à l'adresse :

```
https://github.com/hundredblocks/ml-powered-applications
```

Ce dépôt GitHub contient des instructions pour l'installation et la configuration du code et des outils utilisés dans ce livre.

N.D.T. : Notez bien que, puisque tout le code est accessible directement à partir du dépôt GitHub de l'auteur, seuls les commentaires y sont traduits. Cette remarque s'applique aussi aux bases de données publiques utilisées dans ce qui suit, et donc également aux illustrations correspondant à l'exécution du code. Par contre, ce code a été testé, en ajoutant si nécessaire des indications complémentaires.

Si vous devez, pour une raison ou pour une autre, vous resservir de parties de ce code pour votre propre usage, faites-le librement. En revanche, nous vous demandons d'en citer la source dans une éventuelle publication. C'est juste une question de respect du droit d'auteur en particulier, et du droit des auteurs en général.

PARTIE I

Trouver la bonne approche en matière de ML

La plupart des particuliers ou des entreprises savent bien quels problèmes ils souhaitent résoudre, par exemple prévoir quels clients quitteront une plateforme en ligne ou fabriquer un drone qui suivra un utilisateur pendant qu'il descend une montagne à ski. De même, la plupart des gens peuvent rapidement apprendre à entraîner un modèle afin de classifier les clients ou de détecter des objets avec une exactitude raisonnable compte tenu d'un certain jeu de données.

Ce qui est beaucoup plus rare, cependant, c'est la capacité à prendre un problème, à estimer la meilleure façon de le résoudre, à élaborer un plan pour l'aborder avec le ML, et finalement à exécuter ce plan en toute confiance. C'est souvent une compétence qui doit être acquise par l'expérience, après de multiples déboires liés à des projets trop ambitieux et à des délais non respectés.

Pour un produit donné, il existe de nombreuses solutions potentielles de ML. Sur la Figure I.1, vous pouvez voir une maquette d'un outil potentiel d'aide à la rédaction sur la gauche, avec une suggestion et une possibilité pour l'utilisateur de donner son avis. À droite de l'image, vous trouvez un diagramme représentant une approche potentielle avec le ML permettant de fournir de telles recommandations.

Cette section commence par couvrir ces différentes approches potentielles, ainsi que les méthodes permettant de choisir l'une d'entre elles plutôt que les autres. Elle s'intéresse ensuite aux méthodes servant à concilier les mesures de performance d'un modèle avec les exigences du produit.

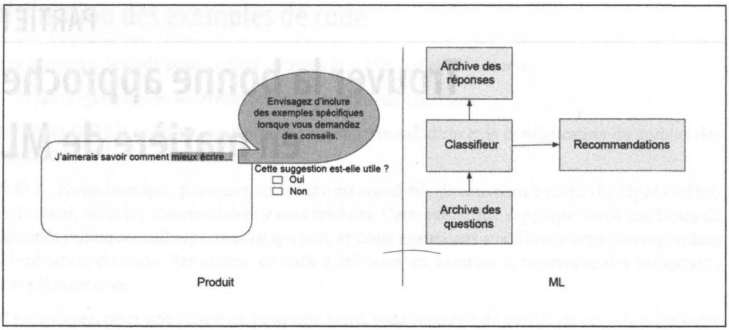


Figure I.1 : Du produit au ML.

Pour ce faire, nous aborderons successivement deux thèmes :

Chapitre 1

À la fin de ce chapitre, vous serez en mesure de formuler une idée pour une application, d'estimer s'il est possible de la résoudre, de déterminer si vous avez besoin du ML pour le faire, et de déterminer quel type de modèle serait le plus logique pour commencer.

Chapitre 2

Dans ce chapitre, nous verrons comment évaluer avec précision les performances de votre modèle dans le contexte des objectifs de votre application, et comment utiliser cette mesure pour faire des progrès réguliers.

CHAPITRE 1

Du produit au ML

L'apprentissage automatique, le *machine learning* ou *ML* en abrégé, permet aux machines d'apprendre à partir de données et de se comporter de manière probabiliste afin de résoudre des problèmes posés par un objectif donné selon des techniques d'optimisation. Cela s'oppose à la programmation traditionnelle, dans laquelle un programmeur écrit des instructions étape par étape pour décrire la manière de résoudre un problème. Cela rend le ML particulièrement utile pour *construire des systèmes pour lesquels nous sommes incapables de définir une solution heuristique.*

La Figure 1.1 décrit deux façons d'écrire un système permettant de détecter des chats. À gauche, un programme consiste en une procédure qui a été écrite manuellement. Sur la droite, une approche de ML exploite un ensemble de photos de chats et de chiens étiquetées avec le nom de l'animal correspondant pour permettre à un modèle d'apprendre à associer l'image à la catégorie voulue. Dans l'approche par ML, il n'y a pas de spécification quant à la manière dont le résultat doit être atteint, mais seulement un ensemble d'échantillons fournis en entrée et en sortie.

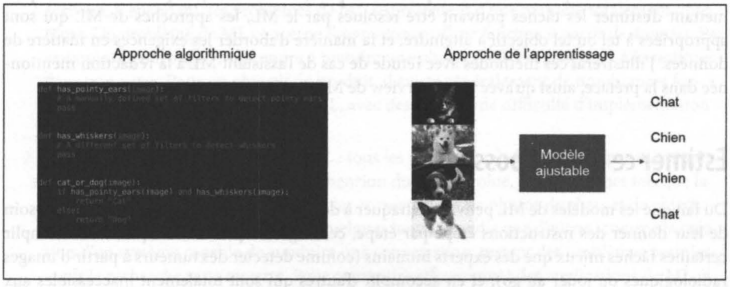


Figure 1.1 : De la définition de procédures à la présentation d'échantillons.

Les technologies basées sur le ML sont puissantes et peuvent donner accès à des produits entièrement nouveaux, mais le fait de se baser sur la reconnaissance des formes introduit un niveau d'incertitude. Il est important d'identifier les parties d'un produit qui pourraient bénéficier du ML et de définir un objectif d'apprentissage de manière à minimiser les risques que les utilisateurs aient une mauvaise expérience.

Par exemple, il est pratiquement impossible (et extrêmement long à essayer) pour les humains d'écrire des instructions étape par étape pour détecter automatiquement quel animal se trouve dans une image en fonction de la valeur des pixels. Cependant, en transmettant des milliers d'images de différents animaux à un réseau de neurones convolutif (CNN), nous pouvons construire un modèle qui effectue cette classification avec plus d'exactitude qu'un humain. C'est pourquoi il s'agit d'une tâche intéressante à traiter à l'aide du ML.

D'un autre côté, une application qui calcule automatiquement vos impôts doit s'appuyer sur les directives fournies par le gouvernement. Comme vous le savez, il est généralement mal vu d'avoir des erreurs dans votre déclaration d'impôts. C'est pourquoi l'utilisation du ML pour générer automatiquement des déclarations d'impôts est une proposition tout à fait discutable.

Vous ne voulez jamais utiliser le ML quand vous pouvez résoudre votre problème avec un ensemble gérable de règles déterministes. Par « gérable », j'entends un ensemble de règles que vous pourriez écrire à la main en toute confiance et dont la maintenance ne serait pas trop complexe.

Ainsi, bien que le ML ouvre un monde très large d'applications, il est important de réfléchir aux tâches qu'il *peut*, et *doit*, résoudre. Lorsque vous réfléchissez à la conception de produits, vous devriez partir d'un problème commercial concret, déterminer s'il nécessite de faire appel à l'apprentissage automatique, puis vous efforcer de trouver l'approche ML qui vous permettra d'itérer le plus rapidement possible.

Nous aborderons ce processus dans le présent chapitre, en commençant par les méthodes permettant d'estimer les tâches pouvant être résolues par le ML, les approches de ML qui sont appropriées à tel ou tel objectif à atteindre, et la manière d'aborder les exigences en matière de données. J'illustrerai ces méthodes avec l'étude de cas de l'assistant ML à la rédaction mentionnée dans la préface, ainsi qu'avec une interview de Monica Rogati.

Estimer ce qui est possible

Du fait que les modèles de ML peuvent s'attaquer à des tâches sans que les humains aient besoin de leur donner des instructions étape par étape, cela signifie qu'ils sont capables d'accomplir certaines tâches mieux que des experts humains (comme détecter des tumeurs à partir d'images radiologiques ou jouer au go), et en accomplir d'autres qui sont totalement inaccessibles aux

humains (comme recommander des articles parmi des millions d'autres ou changer la voix d'un locuteur pour qu'elle ressemble à celle d'une autre personne).

La capacité du ML à apprendre directement à partir des données le rend utile dans un large éventail d'applications, mais rend aussi plus difficile pour les humains de distinguer précisément les problèmes qu'il peut ou non résoudre. Pour chaque résultat positif publié dans un article de recherche ou dans un blog institutionnel, il y a des centaines d'idées raisonnables qui ont totalement échoué.

Bien qu'il n'y ait actuellement aucun moyen sûr de prédire le succès d'une approche basée sur le ML, il existe des lignes directrices qui peuvent vous aider à réduire les risques associés à la réalisation d'un projet de ce type. Le plus important est de toujours partir d'un « objectif produit » à atteindre pour ensuite décider de la meilleure façon de résoudre le problème posé.

À ce stade, soyez ouvert à toute approche, qu'elle nécessite ou non l'utilisation du ML. Lorsque vous envisagez des approches par apprentissage automatique, assurez-vous de les évaluer en fonction de leur adéquation avec les objectifs à atteindre pour le produit, et non pas simplement en fonction de l'intérêt des méthodes en soi.

La meilleure façon d'y parvenir est de suivre deux étapes successives : (1) formuler l'objectif de votre produit selon un paradigme de ML, et (2) évaluer la faisabilité de cette tâche de ML. En fonction de votre évaluation, vous pouvez réajuster votre formulation jusqu'à ce que vous soyez satisfait. Voyons ce que ces étapes signifient réellement.

1. *Formuler un objectif de produit selon un paradigme de ML* : lorsque nous construisons un produit, nous commençons par réfléchir au service que nous voulons offrir aux utilisateurs. Comme nous l'avons mentionné dans la préface, nous illustrerons les concepts développés dans ce livre en utilisant comme étude de cas un éditeur qui aide les utilisateurs à rédiger de meilleures questions. L'objectif de ce produit est clair : nous voulons que les utilisateurs reçoivent des conseils utiles et réalisables quant au contenu qu'ils écrivent. Cependant, les problèmes de ML se présentent d'une manière totalement différente. Un problème de ML concerne *l'apprentissage d'une fonction à partir de données*. Par exemple, il peut s'agir d'apprendre à prendre une phrase dans une langue et à la restituer dans une autre. Pour un objectif de produit, il existe généralement de nombreuses formulations différentes en termes de ML, avec des niveaux de difficulté d'implémentation variables.
2. *Évaluer la faisabilité de la tâche de ML* : tous les problèmes de ML ne sont pas identiques ! Au fur et à mesure que notre compréhension du ML a évolué, des problèmes tels que la construction d'un modèle pour classifier correctement les photos de chats et de chiens sont devenus des tâches résolues en quelques heures, tandis que d'autres, comme la création d'un système capable de conduire une conversation, restent des problèmes pour lesquels la recherche reste ouverte. Pour construire efficacement des applications de ML, il est important d'envisager plusieurs cadres de développement potentiels et de commencer

par ceux que nous jugeons les plus simples. L'une des meilleures façons d'évaluer la difficulté d'un problème de ML est d'examiner à la fois le type de données dont il a besoin et les modèles existants qui pourraient exploiter ces données.

Pour proposer différents cadres et évaluer leur faisabilité, nous devons examiner deux aspects essentiels d'un problème de ML : les données et les modèles.

Nous commencerons par les modèles.

Modèles

Il existe de nombreux modèles couramment utilisés en ML, et nous nous abstiendrons de donner ici un aperçu de chacun d'entre eux. N'hésitez pas à vous référer aux ouvrages énumérés dans la préface pour un aperçu plus complet. En plus des modèles courants, de nombreuses variantes de modèles, de nouvelles architectures et des stratégies d'optimisation sont publiées chaque semaine. Rien qu'en mai 2019, par exemple, plus de 13 000 articles ont été soumis à ArXiv, une archive électronique populaire de recherche où les articles sur les nouveaux modèles sont fréquemment soumis.

Il est toutefois utile de partager une vue d'ensemble des différentes catégories de modèles et de la manière dont elles peuvent être appliquées à différents problèmes. À cette fin, je propose ici une taxonomie simple des modèles basée sur la façon dont ils abordent un problème. Vous pouvez l'utiliser comme un guide pour choisir une approche servant à aborder un problème de ML particulier. Comme les modèles et les données sont étroitement liés dans une démarche de ML, vous remarquerez un certain chevauchement entre cette section et la section « Types de données », plus loin dans ce chapitre.

Les algorithmes de ML peuvent être catégorisés selon qu'ils nécessitent ou non des étiquettes. Ici, le terme *étiquette* fait référence à la présence dans les données d'une sortie idéale qu'un modèle devrait produire pour un échantillon donné. Les algorithmes supervisés exploitent des jeux de données qui contiennent des étiquettes pour les entrées, et ils visent à apprendre une mise en correspondance entre entrées et étiquettes. Les algorithmes non supervisés, en revanche, ne nécessitent pas d'étiquettes. Enfin, les algorithmes faiblement supervisés exploitent des étiquettes qui ne sont pas exactement la sortie souhaitée, mais qui lui ressemblent d'une certaine manière.

De nombreux objectifs peuvent être atteints par des algorithmes supervisés ou non. Par exemple, un système de détection des fraudes peut être construit en entraînant un modèle pour détecter les transactions qui diffèrent de la moyenne, sans qu'il soit nécessaire de les étiqueter. Un tel système pourrait également être construit en étiquetant manuellement les transactions comme étant frauduleuses ou légitimes, et en entraînant un modèle pour apprendre à partir de ces étiquettes.

Pour la plupart des applications, les approches supervisées sont plus faciles à valider car nous avons accès à des étiquettes permettant d'évaluer la qualité des prévisions d'un modèle. Il est

également plus facile d'entraîner les modèles, puisque nous avons accès aux résultats souhaités. Si la création d'un jeu de données étiquetées peut parfois prendre du temps au départ, elle facilite grandement la construction et la validation des modèles. Pour cette raison, ce livre couvrira principalement les approches supervisées.

Cela étant dit, le fait de déterminer le type d'intrants que votre modèle prendra en compte et les résultats qu'il produira vous aidera à réduire considérablement le champ des approches potentielles. Partant de là, n'importe laquelle des catégories suivantes d'approches de ML pourrait être une bonne solution :

- Classification et régression
- Extraction de connaissances
- Organisation en catalogues
- Modèles génératifs

Ces points vont être développés dans la section suivante. En explorant ces différentes approches de modélisation, je vous recommande de réfléchir au type de données dont vous disposez ou que vous pourriez collecter. Souvent, la disponibilité des données finit par être le facteur limitant dans le choix du modèle.

Classification et régression

Certains projets visent à classifier efficacement les points de données entre deux ou plusieurs catégories, ou à leur attribuer une valeur sur une échelle continue (on parle alors de *régression* au lieu de *classification*). La régression et la classification sont techniquement différentes, mais les méthodes utilisées pour les traiter se chevauchent souvent de manière significative, et c'est pourquoi nous les regroupons ici.

L'une des raisons pour lesquelles la classification et la régression sont similaires est que la plupart des modèles de classification produisent un score de probabilité pour qu'un modèle appartienne à une catégorie. L'aspect classification se résume alors à décider comment attribuer un objet à une catégorie sur la base de ces scores. À un niveau élevé, un modèle de classification peut donc être considéré comme une régression portant sur des valeurs de probabilité.

En général, nous classifions ou scorons des échantillons individuels, tels que les filtres de spams qui classifient chaque courriel comme étant valide ou indésirable, les systèmes de détection de fraudes qui classifient les utilisateurs comme étant frauduleux ou légitimes, ou encore les modèles radiologiques de vision par ordinateur qui classent les os comme étant fracturés ou sains.

Sur la Figure 1.2, vous pouvez voir un exemple de classification d'une phrase en fonction du sentiment (au sens d'opinion, de jugement) qu'elle exprime et du sujet qu'elle couvre.

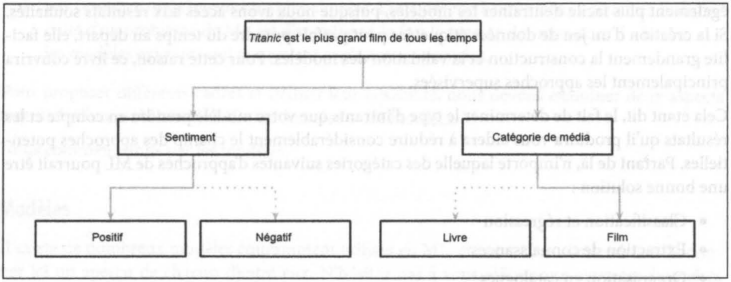


Figure 1.2 : Classement d'une phrase dans plusieurs catégories.

Dans les projets de régression, au lieu d'attribuer une classe à chaque échantillon, nous leur donnons une valeur. Prévoir le prix de vente d'une maison en fonction d'attributs tels que le nombre de pièces et l'endroit où elle se situe est un exemple de problème de régression.

Dans certains cas, nous avons accès à une série de points de données du passé (au lieu d'un instantané) pour prédire un événement futur. Ce type de données est souvent appelé *série chronologique*, et faire des prédictions à partir d'une telle série de points de données est appelé *prévision*. Les données de séries chronologiques peuvent représenter les antécédents médicaux d'un patient, ou encore une série de mesures de fréquentation de parcs ou de musées. Ces projets bénéficient souvent de modèles et de fonctionnalités qui peuvent exploiter cette dimension temporelle supplémentaire.

Dans d'autres cas, nous essayons de détecter des événements inhabituels à partir d'un jeu de données. C'est ce qu'on appelle la *détection d'anomalies*. Lorsqu'un problème de classification consiste à essayer de détecter des événements qui représentent une petite minorité dans les données, et sont donc difficiles à détecter avec exactitude, un ensemble de méthodes différentes est souvent nécessaire. Chercher une aiguille dans une botte de foin est ici une bonne analogie.

Un bon travail de classification et de régression nécessite le plus souvent une importante phase de sélection et d'ingénierie des caractéristiques. La sélection de caractéristiques consiste à identifier un sous-ensemble de ces caractéristiques qui ont la valeur prédictive la plus élevée. La génération de caractéristiques est la tâche qui consiste à identifier et à générer de bons prédicteurs pour une certaine cible en modifiant et en combinant les caractéristiques existantes dans un jeu de données. Nous aborderons ces deux sujets plus en détail dans la Partie III.

Récemment, le deep learning a montré une capacité prometteuse à générer automatiquement des caractéristiques utiles à partir d'images, de textes et de sons. À l'avenir, il pourrait jouer un rôle plus important dans la simplification de la génération et de la sélection de caractéristiques, mais pour l'instant, cela reste une partie intégrante du flux de travail du ML.

Enfin, nous pouvons souvent nous appuyer sur la classification ou la notation décrites plus haut pour donner des conseils utiles. Pour ce faire, il faut construire un modèle de classification interprétable et l'utiliser pour générer de tels conseils. Nous y reviendrons de manière détaillée plus tard !

Tous les problèmes ne visent pas à attribuer un ensemble de catégories ou de valeurs à un échantillon. Dans certains cas, nous aimerions opérer à un niveau plus granulaire et extraire des informations à partir de certaines parties d'une entrée, comme par exemple savoir où se trouve un objet dans une image.

Extraire des connaissances à partir de données non structurées

Les *données structurées* sont des données qui sont stockées sous forme tabulaire. Les tableaux de bases de données et les feuilles Excel sont de bons exemples de données structurées. Les *données non structurées* font référence à des jeux de données qui ne se présentent pas sous forme tabulaire. Il s'agit de textes (articles, critiques, Wikipédia, *etc.*), de musiques, de vidéos et de chansons.

Sur la Figure 1.3, vous pouvez voir un exemple de données structurées à gauche et de données non structurées à droite. Les modèles d'extraction des connaissances partent d'une source de données non structurées, et recherchent la structure de celle-ci en utilisant le ML.

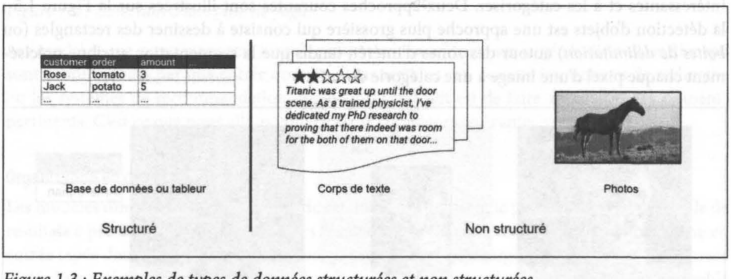


Figure 1.3 : Exemples de types de données structurées et non structurées.

Dans le cas d'un texte, l'extraction de connaissances peut être utilisée pour ajouter par exemple de la structure à des séries de critiques. Un modèle peut être entraîné pour extraire des aspects tels que la propreté, la qualité du service et le prix des examens. Les utilisateurs pourraient alors accéder facilement aux critiques qui mentionnent les sujets qui les intéressent.

Dans le domaine médical, un modèle d'extraction des connaissances pourrait être construit de manière à prendre en entrée le texte brut d'articles médicaux pour en extraire des informations telles que la maladie dont il est question dans l'article, ainsi que le diagnostic associé et ses per-

formances. Sur la Figure 1.4, un modèle prend une phrase en entrée et extrait quels mots font référence à un type de média et quels mots font référence au titre d'un média. L'application d'un tel modèle sur les commentaires d'un forum de fans, par exemple, nous permettrait de générer des résumés pour des films qui sont fréquemment discutés.

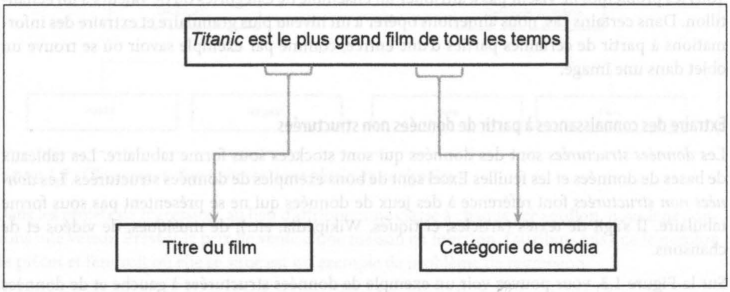


Figure 1.4 : Extraction du type de média et du titre à partir d'une phrase.

Pour les images, les tâches d'extraction de connaissances consistent souvent à trouver des zones intéressantes et à les catégoriser. Deux approches courantes sont illustrées sur la Figure 1.5 : la détection d'objets est une approche plus grossière qui consiste à dessiner des rectangles (ou *boîtes de délimitation*) autour des zones d'intérêt, tandis que la segmentation attribue précisément chaque pixel d'une image à une catégorie donnée.

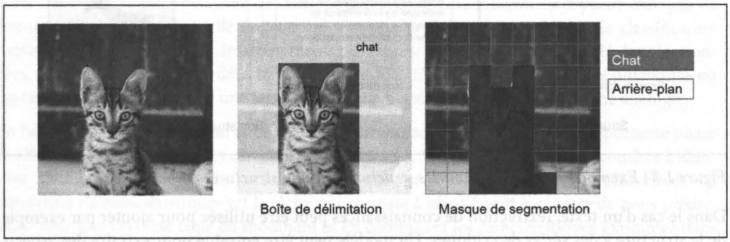


Figure 1.5 : Boîtes de délimitation et masques de segmentation.

Parfois, ces informations extraites peuvent être utilisées comme données d'entrée pour un autre modèle. Un exemple est l'utilisation d'un modèle de détection de poses pour extraire les points clés dans une vidéo d'un yogi, suivie de l'injection de ces points clés dans un second modèle qui classifie la pose comme étant correcte ou non sur la base de données étiquetées. La Figure 1.6

montre un exemple d'une succession de deux modèles qui pourraient faire exactement cela. Le premier modèle extrait des informations structurées (les coordonnées des articulations) à partir de données non structurées (une photo), et le second prend ces coordonnées et classifie la pose de yoga correspondante.

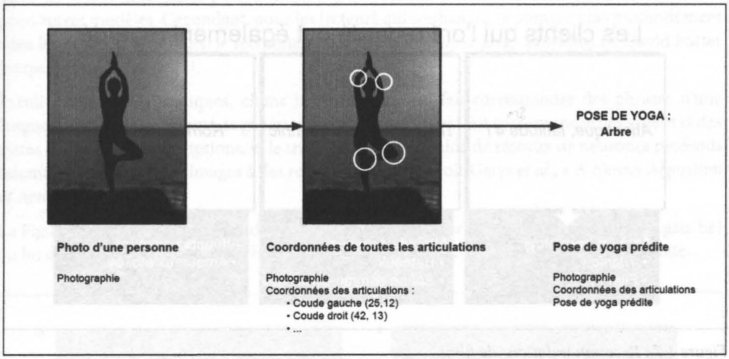


Figure 1.6 : Détection d'une pose de yoga.

Les modèles que nous avons vus jusqu'à présent se concentrent sur la génération de sorties qui sont conditionnées par une entrée donnée. Dans certains cas, comme les moteurs de recherche ou les systèmes de recommandation, l'objectif du produit est de faire apparaître des éléments pertinents. C'est ce que nous allons couvrir dans la catégorie suivante.

Organisation en catalogues

Les modèles d'organisation sous forme de catalogues produisent le plus souvent un ensemble de résultats à présenter aux utilisateurs. Ces résultats peuvent être conditionnés par une chaîne en entrée tapée dans une barre de recherche, par une image téléchargée ou encore par une phrase dictée à un assistant vocal. Dans de nombreux cas, tels que les services de streaming, cet ensemble de résultats peut également être présenté de manière proactive à l'utilisateur sous la forme d'un contenu qu'il est susceptible d'apprécier sans qu'il ait à formuler la moindre demande.

La Figure 1.7 montre un exemple d'un tel système qui propose de regarder des films réputés être des candidats potentiels en se basant sur un film que l'utilisateur vient de visionner, mais sans que celui-ci n'effectue aucune forme de recherche.

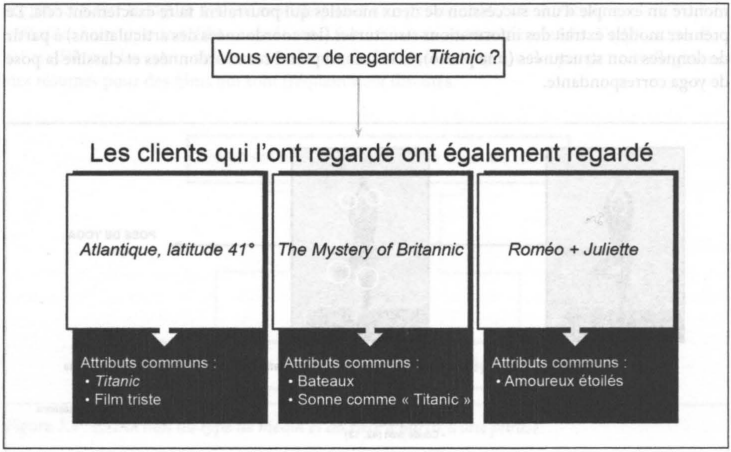


Figure 1.7 : Recommandations de films.

Ces modèles *recommandent* donc soit des articles qui sont liés à un article pour lequel l'utilisateur a déjà exprimé un intérêt (voir par exemple le fonctionnement d'Amazon), soit fournissent un moyen utile d'effectuer une recherche dans un catalogue (permettant aux utilisateurs de rechercher des articles en tapant du texte ou en soumettant leurs propres photos).

Ces recommandations sont le plus souvent basées sur l'apprentissage des habitudes des utilisateurs précédents, auquel cas on les appelle systèmes de recommandation *collaboratifs*. Parfois, elles sont basées sur des attributs particuliers d'éléments, auquel cas on les appelle systèmes de recommandation *basés sur le contenu*. Certains systèmes utilisent à la fois des approches collaboratives et des approches basées sur le contenu.

Enfin, le ML peut également être utilisé à des fins créatives. Les modèles peuvent apprendre à générer des images, des sons et même des textes amusants et esthétiques. Ces modèles sont dits *génératifs*.

Modèles génératifs

Les modèles génératifs se concentrent sur la génération de données, potentiellement dépendantes de l'entrée effectuée par l'utilisateur. Comme ces modèles ont pour objectif cette génération de données plutôt que leur classification en catégories, leur notation, l'extraction d'informations ou leur organisation, ils ont généralement un large éventail de sorties. Cela signifie que

les modèles génératifs sont particulièrement adaptés à des tâches telles que la traduction, où les résultats sont énormément variés.

D'autre part, les modèles génératifs sont souvent utilisés pour entraîner et avoir des résultats moins contraints, ce qui en fait un choix plus risqué pour la production. C'est pourquoi, à moins qu'ils ne soient nécessaires pour atteindre votre objectif, je vous recommande de commencer par d'autres modèles. Cependant, pour les lecteurs qui souhaitent se plonger plus profondément dans les modèles génératifs, je recommande le livre *Generative Deep Learning*, de David Foster (`https://bit.ly/2V8J6TR`).

Parmi les exemples pratiques, citons la traduction, qui fait correspondre des phrases d'une langue à une autre, le résumé, la génération de sous-titres, qui fait correspondre des vidéos et des pistes audio à des transcriptions, et le transfert de style à l'aide de réseaux de neurones profonds faisant correspondre des images à des rendus artistiques (voir Gatys *et al.*, « *A Neural Algorithm of Artistic Style* »[1]).

La Figure 1.8 montre un exemple de modèle génératif transformant une photographie (à gauche) en lui donnant un style similaire à celui d'une peinture présentée dans la vignette de droite.

Figure 1.8 : Exemple de transfert de style tiré de Gatys et al., « A Neural Algorithm of Artistic Style ».

Comme vous pouvez le constater, chaque type de modèle nécessite un type de données différent pour l'entraînement. En général, le choix d'un modèle dépend fortement des données que vous êtes en mesure d'obtenir – la disponibilité des données détermine souvent le choix du modèle.

Voyons maintenant quelques scénarios de données courants et les modèles associés.

1 Voir l'adresse `https://arxiv.org/abs/1508.06576`.

Données

Les modèles de ML supervisés exploitent des motifs, des structures communes, dans les données pour apprendre des relations utiles entre les entrées et les sorties. Si un jeu de données contient des caractéristiques qui sont prédictives quant à la sortie cible, il devrait être possible pour un modèle approprié d'en tirer des enseignements. Le plus souvent, cependant, nous ne disposons pas initialement des données adéquates pour entraîner un modèle afin de résoudre de bout en bout le problème concernant l'utilisation d'un produit.

Par exemple, supposons que nous entraînions un système de reconnaissance vocale qui sera chargé d'écouter les demandes des clients, de comprendre leur intention et d'effectuer des actions en fonction de cette intention. Lorsque nous commencerons à travailler sur ce projet, nous pourrions définir un ensemble d'intentions que nous voudrions comprendre, comme par exemple « jouer un film à la télévision ».

Pour entraîner un modèle de ML à cette tâche, il faudrait disposer d'un jeu de données contenant des clips audio d'utilisateurs venant d'horizons divers et demandant, en employant leurs propres termes, que le système leur passe un film. Il est crucial de disposer d'un ensemble d'entrées représentatif, car un modèle quelconque ne pourra tirer des enseignements que des données que nous lui présenterons. Si un jeu de données contient des échantillons provenant uniquement d'un certain sous-ensemble de la population, un produit ne sera utile qu'à ce sous-ensemble. Dans cette optique, en raison du domaine spécialisé que nous avons sélectionné, il est extrêmement improbable qu'un jeu de données contenant de tels échantillons existe déjà.

Pour la plupart des applications auxquelles nous voudrions nous attaquer, nous devrons rechercher, conserver et collecter des données supplémentaires. Le processus d'acquisition des données peut varier considérablement en termes de portée et de complexité en fonction des spécificités d'un projet, et il est essentiel d'estimer à l'avance le défi ainsi posé pour pouvoir réussir.

Pour commencer, définissons quelques situations différentes dans lesquelles vous pouvez vous retrouver lorsque vous recherchez un jeu de données. Cette situation initiale devrait être un facteur clé pour décider de la marche à suivre.

Types de données

Une fois que nous avons défini un problème comme étant une *mise en correspondance entre les entrées et les sorties*, nous pouvons rechercher les sources de données qui suivent ce *mappage*.

Dans le cas de la détection de fraudes, il pourrait s'agir d'échantillons d'utilisateurs frauduleux et innocents, ainsi que de caractéristiques de leur compte que nous pourrions utiliser pour prédire leur comportement. Dans le cas de la traduction, il s'agirait d'un corpus de paires de phrases dans les domaines source et cible. Pour l'organisation et la recherche de contenus, il pourrait s'agir d'un historique des recherches et des clics déjà effectués.

Il est rare que nous puissions trouver la correspondance exacte que nous recherchons. C'est pourquoi il est utile d'examiner quelques cas différents. Considérez cela comme une hiérarchie des besoins en données.

Disponibilité des données

Il existe en gros trois niveaux de disponibilité des données, du meilleur scénario au plus compliqué. Malheureusement, comme pour la plupart des autres tâches, vous pouvez généralement supposer que le type de données le plus utile sera aussi le plus difficile à trouver. Passons-les en revue, en nous basant sur l'illustration de la Figure 1.9.

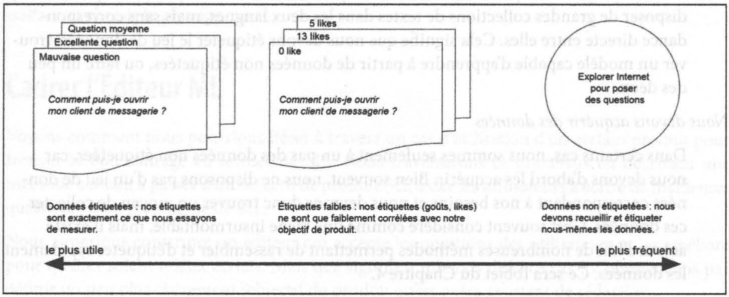


Figure 1.9 : Disponibilité des données versus utilité des données.

Il existe des données étiquetées

Il s'agit de la catégorie la plus à gauche sur la Figure 1.9. Lorsqu'on travaille sur un modèle supervisé, trouver un jeu de données étiquetées est le rêve de tout praticien. *Étiquetées* signifie ici que de nombreux points de données contiennent la valeur cible que le modèle essaie de prédire. Cela facilite grandement l'entraînement et l'évaluation de la qualité du modèle, car les étiquettes fournissent des réponses correspondant à la vérité de terrain. En pratique, il est rare de trouver un jeu de données étiquetées qui réponde à vos besoins et qui soit disponible gratuitement sur le Web. Il est cependant courant de confondre abusivement le jeu de données que vous trouvez avec le jeu de données dont vous avez besoin.

Il existe des données faiblement étiquetées

Il s'agit de la catégorie intermédiaire sur la Figure 1.9. Certains jeux de données contiennent des étiquettes qui ne sont pas exactement une cible de modélisation, mais qui sont plus ou moins corrélées avec elle. L'historique des titres lus et sautés pour un service de diffusion de musique en continu fournit un exemple de jeu de données faiblement

étiquetées pour prédire si un utilisateur aime ou n'aime pas une chanson. Un auditeur peut ne pas avoir marqué une chanson comme n'étant pas aimée. Mais s'il l'a sautée pendant la lecture, c'est une indication qu'il ne l'a peut-être pas aimée. Les étiquettes faibles sont par définition moins précises, mais souvent plus faciles à trouver que les étiquettes parfaites.

Il existe des données non étiquetées

Cette catégorie se trouve sur le côté droit de la Figure 1.9. Dans certains cas, bien que nous ne disposions pas d'un jeu de données étiquetées mettant en correspondance les entrées et les sorties souhaitées, nous avons au moins accès à un jeu de données contenant des échantillons pertinents. Pour l'exemple de traduction de texte, nous pourrions disposer de grandes collections de textes dans les deux langues, mais sans correspondance directe entre elles. Cela signifie que nous devons étiqueter le jeu de données, trouver un modèle capable d'apprendre à partir de données non étiquetées, ou faire un peu des deux.

Nous devons acquérir des données

Dans certains cas, nous sommes seulement à un pas des données non étiquetées, car nous devons d'abord les acquérir. Bien souvent, nous ne disposons pas d'un jeu de données correspondant à nos besoins et nous devrons donc trouver un moyen de collecter ces données. C'est souvent considéré comme une tâche insurmontable, mais il existe aujourd'hui de nombreuses méthodes permettant de rassembler et d'étiqueter rapidement les données. Ce sera l'objet du Chapitre 4.

Pour notre étude de cas, un jeu de données idéal serait un ensemble de questions tapées par l'utilisateur, ainsi qu'un ensemble de questions mieux formulées. Un jeu de données *faiblement étiqueté* serait composé de nombreuses questions avec quelques étiquettes faibles indiquant leur qualité, comme « aime » ou « vote pour ». Cela aiderait un modèle à apprendre ce qui fait les bonnes et les mauvaises questions, mais ne fournirait pas d'exemples côte à côte pour la même question. Vous pouvez voir ces deux situations sur la Figure 1.9.

En général, en ML, un jeu de données faiblement étiqueté fait référence à un jeu de données qui contient des informations qui aideront un modèle à apprendre, mais pas la vérité de terrain exacte. En pratique, la plupart des jeux de données que nous pouvons rassembler sont faiblement étiquetés.

Avoir un jeu de données imparfait est tout à fait normal et ne devrait pas vous bloquer. Le processus de ML est de nature itérative, donc commencer par un certain jeu de données et obtenir quelques résultats initiaux est la meilleure façon de progresser, quelle que soit la qualité des données.

Les jeux de données sont itératifs

Dans de nombreux cas, comme vous ne serez pas en mesure de trouver immédiatement un jeu de données contenant une correspondance directe entre les entrées et les sorties souhaitées, je suggère d'itérer progressivement la manière dont vous formulez le problème, ce qui facilitera la recherche d'un jeu de données adéquat pour commencer. Chaque jeu de données que vous explorez et utilisez vous fournira des informations précieuses qui pourront vous servir pour élaborer la prochaine version de votre jeu de données et générer des caractéristiques utiles pour vos modèles.

Plongeons maintenant dans notre étude de cas, et voyons comment nous pouvons utiliser ce que nous avons appris pour identifier différents modèles et jeux de données que nous pourrions utiliser, et choisir le plus approprié.

Cadrer l'Éditeur ML

Voyons comment nous pourrions itérer à travers un cas d'utilisation d'un certain produit pour trouver le bon cadrage en termes de ML. Nous traverserons ce processus en décrivant une méthode servant à passer d'un objectif de produit (ici, aider les utilisateurs à écrire de meilleures questions) à un paradigme de ML.

Nous aimerions construire un éditeur qui accepte les questions des utilisateurs et les améliore pour qu'elles soient mieux écrites. Mais que signifie « mieux » dans ce cas ? Commençons par définir un peu plus clairement l'objectif du produit qu'est notre assistant de rédaction.

De nombreuses personnes utilisent les forums, les réseaux sociaux et les sites Web tels que Stack Overflow[2] (du moins dans des domaines liés à la programmation) pour trouver des réponses à leurs questions. Cependant, la façon dont les gens posent leurs questions a un impact considérable sur l'utilité de la réponse qu'ils reçoivent. Cela est regrettable tant pour l'utilisateur qui cherche à obtenir une réponse à sa question que pour les futurs utilisateurs qui pourraient avoir le même problème et trouver utile une réponse existante. Notre objectif sera donc de construire un *assistant qui puisse aider les utilisateurs à rédiger de meilleures questions.*

Nous avons un objectif en termes de produit et nous devons maintenant décider de l'approche de modélisation à utiliser. Pour prendre cette décision, nous allons passer par la boucle d'itération de sélection du modèle et de validation des données mentionnée plus haut.

Tenter de tout faire avec le ML : un cadre de bout en bout

Dans ce contexte, *de bout en bout* signifie utiliser un modèle unique pour passer de l'entrée à la sortie sans aucune étape intermédiaire. Comme la plupart des objectifs fixés pour des produits

2 Voir l'adresse `https://stackoverflow.com/`.

sont très spécifiques, tenter de résoudre un cas d'utilisation entier en l'apprenant de bout en bout nécessite souvent de construire sur mesure des modèles ML de pointe. Cette solution peut être la bonne pour les équipes qui ont les ressources nécessaires pour développer et maintenir de tels modèles, mais il vaut souvent la peine de commencer par des modèles plus connus et mieux compris.

Dans notre cas, nous pourrions essayer de rassembler un ensemble de questions mal formulées, ainsi que leurs versions éditées par des professionnels. Nous pourrions alors utiliser un modèle génératif pour passer directement d'un texte à l'autre.

La Figure 1.10 illustre ce à quoi cela ressemblerait dans la pratique. Elle montre un diagramme simple avec l'entrée de l'utilisateur à gauche, la sortie souhaitée à droite, et un modèle entre les deux.

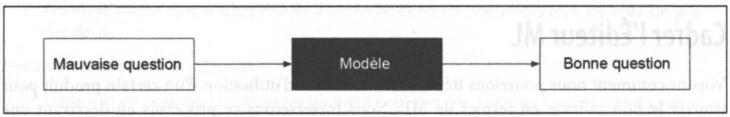


Figure 1.10 : Une approche de bout en bout.

Comme vous allez le voir, cette approche implique des défis importants :

Données

Pour acquérir un tel jeu de données, il faudrait trouver des paires de questions répondant à la même intention, mais une qualité de formulation différente. C'est un type de jeu de données assez rare à trouver tel quel. Le construire nous-mêmes serait également coûteux, car nous aurions besoin de l'aide d'éditeurs professionnels pour générer ces données.

Modèle

Les modèles permettant d'aller d'une séquence de texte à une autre, qui entrent dans la catégorie des modèles génératifs dont il a été question plus haut, ont énormément progressé ces dernières années. Les modèles de séquence à séquence (tels que décrits dans l'article de I. Sutskever *et al.*, « *Sequence to Sequence Learning with Neural Networks* »[3]) ont été proposés à l'origine en 2014 pour les tâches de traduction, et ils comblent le fossé entre traduction automatique et traduction humaine. Toutefois, ces modèles ont surtout fait leurs preuves pour les tâches se situant au niveau des phrases, et ils n'ont pas été fréquemment utilisés pour traiter des textes comprenant plus d'un paragraphe. En effet, jusqu'à présent, ils n'ont pas été en mesure de saisir le contexte à long terme s'étendant d'un paragraphe à l'autre. De plus, comme ils ont généralement un grand nombre

3 Voir l'adresse `https://arxiv.org/abs/1409.3215`.

de paramètres, ils font partie des modèles les plus lents à entraîner. Si un modèle n'est entraîné qu'une seule fois, cela ne pose pas nécessairement de problème. S'il doit être à nouveau entraîné toutes les heures ou tous les jours, cette lenteur peut devenir un facteur important.

Latence

Les modèles de séquence à séquence sont souvent des *modèles autorégressifs*, ce qui signifie qu'ils nécessitent la sortie produite pour le mot précédent pour commencer à travailler sur le suivant. Cela leur permet d'exploiter les informations fournies par des mots voisins, mais les rend plus lents à s'entraîner et à effectuer des inférences que les modèles plus simples. De tels modèles peuvent prendre quelques secondes pour produire une réponse lors de l'inférence, contrairement à la latence de moins d'une seconde pour les modèles plus simples. Bien qu'il soit possible d'optimiser un tel modèle pour qu'il s'exécute assez rapidement, cela nécessitera un travail d'ingénierie supplémentaire.

Facilité d'implémentation

L'entraînement de bout en bout de modèles complexes est un processus très délicat et sujet aux erreurs, car ils comportent de nombreuses parties mobiles. Cela signifie que nous devons prendre en considération le compromis entre la performance potentielle d'un modèle et la complexité qu'il ajoute à un pipeline. Cette complexité nous ralentira lors de la construction d'un pipeline, mais elle introduit de surcroît une charge de maintenance. Si nous prévoyons que d'autres membres de l'équipe devront peut-être itérer et améliorer votre modèle, il peut être intéressant de choisir un ensemble de modèles plus simples et mieux compris.

Cette approche de bout en bout pourrait fonctionner, mais elle nécessitera d'importantes collectes de données et d'efforts d'ingénierie en amont, sans garantie de succès. Il serait donc utile d'explorer d'autres alternatives, comme nous allons le voir ensuite.

L'approche la plus simple : être l'algorithme

Comme vous le verrez dans l'interview à la fin de cette section, c'est souvent une bonne idée pour les data scientists *d'être l'algorithme* avant qu'ils ne l'implémentent. En d'autres termes, pour comprendre comment automatiser au mieux un problème, il faut commencer par tenter de le résoudre manuellement. Donc, si nous modifions nous-mêmes les questions pour améliorer leur lisibilité et les chances d'obtenir une réponse, comment nous y prendrions-nous ?

Une première approche consisterait à ne pas du tout utiliser de données mais à s'appuyer sur l'état de l'art pour définir ce qui fait qu'une question ou un corps de texte est bien écrit. Pour obtenir des conseils généraux à ce sujet, nous pourrions faire appel à un rédacteur professionnel ou consulter des guides de style parus dans des articles pour en savoir plus.

En outre, nous devrions nous plonger dans un jeu de données pour examiner des exemples individuels et les tendances, et les laisser éclairer notre stratégie de modélisation. Nous passerons ce point pour l'instant, car nous verrons comment le faire de manière plus approfondie au Chapitre 4.

Pour commencer, nous pourrions examiner les recherches existantes afin d'identifier quelques attributs que nous pourrions utiliser pour aider les gens à écrire plus clairement. Ces caractéristiques pourraient inclure des facteurs tels que :

La simplicité de la prose

> Nous donnons souvent aux nouveaux auteurs le conseil d'utiliser des mots et des structures de phrases plus simples. Nous pourrions ainsi établir une série de critères sur la longueur appropriée des phrases et des mots, et recommander des modifications si nécessaire.

Ton

> Nous pourrions mesurer l'utilisation des adverbes, des superlatifs et de la ponctuation pour évaluer la polarité du texte. En fonction du contexte, les questions plus pointues ou plus tranchées peuvent recevoir moins de réponses.

Caractéristiques structurelles

> Enfin, nous pourrions essayer d'extraire la présence d'attributs structurels importants tels que l'utilisation de salutations ou de points d'interrogation.

Une fois que nous avons identifié et généré des caractéristiques utiles, nous pouvons construire une solution simple qui les utilise pour fournir des recommandations. Il n'y a pas de *machine learning* ici, mais cette phase est cruciale pour deux raisons : elle fournit une base de référence qui est très rapide à implémenter et qui de plus servira d'étalon pour mesurer les modèles.

Pour valider notre intuition sur la façon de détecter une bonne écriture, nous pouvons rassembler un jeu de données composé de « bons » et de « mauvais » textes et voir si nous pouvons distinguer le bon du mauvais en utilisant ces caractéristiques.

Un terrain d'entente : trier les leçons de notre expérience

Maintenant que nous disposons d'un ensemble de caractéristiques de base, nous pouvons essayer de les utiliser pour *apprendre un modèle de style à partir d'un corpus de données*. Pour ce faire, nous pouvons rassembler un jeu de données, en extraire les caractéristiques que nous avons décrites précédemment et entraîner un classifieur pour séparer les bons et les mauvais exemples.

Une fois que nous disposons d'un modèle capable de classifier un texte écrit, nous pouvons l'inspecter pour identifier les caractéristiques qui sont hautement prédictives et les utiliser comme recommandations. Nous verrons comment faire cela en pratique au Chapitre 7.

La Figure 1.11 décrit cette approche. Sur le côté gauche, un modèle est entraîné pour classifier une question comme étant bonne ou mauvaise. Sur le côté droit, le modèle entraîné reçoit une question et note les reformulations candidates pour cette question, ce qui lui permet d'obtenir un meilleur score. La reformulation ayant le score le plus élevé est recommandée à l'utilisateur.

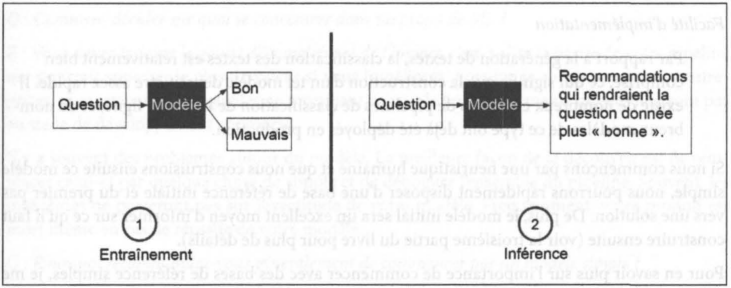


Figure 1.11 : Un juste milieu entre les approches manuelle et bout à bout.

Examinons les défis que nous avons exposés plus haut dans la section « Tenter de tout faire avec le ML : un cadre de bout en bout », et voyons si l'approche à l'aide d'un classifieur les rend plus faciles :

Jeu de données

Nous pourrions obtenir un jeu de données composé de bons et de mauvais échantillons en recueillant les questions d'un forum en ligne, ainsi qu'une certaine métrique quant à leur qualité, comme le nombre d'avis ou de votes positifs. Contrairement à l'approche de bout en bout, cela ne nous oblige pas à avoir accès aux mêmes questions reformulées. Nous avons simplement besoin d'un ensemble de bons et de mauvais échantillons dont nous pouvons espérer tirer des caractéristiques globales, ce qui représente un jeu de données plus facile à trouver.

Modèle

Nous devons considérer deux choses ici : à quel point un modèle est-il prédictif (peut-il séparer efficacement les bons et les mauvais articles ?), et à quel point est-il facile d'en extraire des caractéristiques (pouvons-nous voir quels attributs ont été utilisés pour classifier un échantillon ?). Il existe de nombreux modèles potentiels que nous pourrions utiliser ici, ainsi que différentes caractéristiques que nous pourrions extraire du texte pour le rendre plus explicable.

Latence

> La plupart des classifieurs de texte sont assez rapides. Nous pourrions commencer avec un modèle simple comme une forêt aléatoire, qui peut renvoyer des résultats en moins d'un dixième de seconde sur du matériel ordinaire, et passer à des architectures plus complexes si nécessaire.

Facilité d'implémentation

> Par rapport à la génération de textes, la classification des textes est relativement bien comprise, ce qui signifie que la construction d'un tel modèle devrait être assez rapide. Il existe de nombreux exemples de pipelines de classification de textes en ligne, et de nombreux modèles de ce type ont déjà été déployés en production.

Si nous commençons par une heuristique humaine et que nous construisions ensuite ce modèle simple, nous pourrons rapidement disposer d'une base de référence initiale et du premier pas vers une solution. De plus, le modèle initial sera un excellent moyen d'informer sur ce qu'il faut construire ensuite (voir la troisième partie du livre pour plus de détails).

Pour en savoir plus sur l'importance de commencer avec des bases de référence simples, je me suis entretenu avec Monica Rogati, qui partage ici certaines des leçons qu'elle a apprises en aidant des équipes chargées des données à fournir des produits.

Monica Rogati : Comment choisir et hiérarchiser les projets ML ?

Après avoir obtenu son doctorat en informatique, Monica Rogati a commencé sa carrière chez LinkedIn où elle a travaillé sur des produits de fond tels que l'intégration du ML dans l'algorithme People You May Know (personnes que vous pourriez connaître) et a construit la première version de l'appariement entre emplois et candidats. Elle est ensuite devenue vice-présidente des données chez Jawbone, où elle a mis sur pied et dirigé toute l'équipe chargée des données. Monica est aujourd'hui conseillère auprès de dizaines d'entreprises dont le nombre d'employés varie de 5 à 8 000. Elle a aimablement accepté de partager certains des conseils qu'elle donne souvent aux équipes lorsqu'il s'agit de concevoir et d'exécuter des produits basés sur le ML.

Q : Comment déterminez-vous la portée d'un produit de ML ?

R : Vous devez vous rappeler que vous essayez d'utiliser les meilleurs outils pour résoudre un problème, et donc de ne faire appel au ML que si cela a un sens.

Disons que vous vouliez prédire ce que fera un utilisateur d'une application et le lui montrer en tant que suggestion. Vous devriez commencer par combiner les discussions sur la modélisation et le produit. Cela implique, entre autres, de concevoir le produit en fonction de la façon dont les insuffisances du ML seront gérées.

Vous pourriez commencer par prendre en compte la confiance que notre modèle a dans ses prédictions. Nous pourrions ensuite formuler nos suggestions différemment en fonction du score de confiance. Si ce score de confiance est supérieur à 90 %, nous présentons la suggestion de manière très visible ; s'il est supérieur à 50 %, nous l'affichons toujours, mais avec moins d'emphase, et nous n'affichons rien si le score de confiance est inférieur à ce pourcentage.

Q : Comment décider sur quoi se concentrer dans un projet de ML ?

R : Vous devez trouver le *goulet d'étranglement de l'impact*, c'est-à-dire la partie de votre pipeline qui pourrait apporter le plus de valeur si vous l'améliorez. Lorsque je collabore avec des entreprises, je constate souvent qu'elles ne travaillent pas sur le bon problème ou qu'elles n'en sont pas au stade de développement approprié.

Il y a souvent des problèmes autour du modèle. La meilleure façon de le découvrir est de remplacer celui-ci par quelque chose de simple et de déboguer l'ensemble du pipeline. Souvent, les problèmes ne porteront pas sur l'exactitude de votre modèle. Fréquemment, votre produit est mort même en cas de réussite de votre modèle.

Q : Pourquoi recommandez-vous généralement de commencer par un modèle simple ?

R : Le but de notre plan devrait être de mettre notre modèle en péril d'une manière ou d'une autre. La meilleure façon d'y parvenir est de commencer par ce que l'on appelle une « proposition de l'homme de paille » afin d'évaluer les performances dans le pire des cas. Pour notre exemple précédent, cela pourrait simplement consister à suggérer une action quelconque que l'utilisateur aurait prise auparavant.

Si nous faisions cela, avec quelle fréquence notre prédiction serait-elle correcte, et jusqu'à quel point notre modèle serait-il ennuyeux pour l'utilisateur si nous nous trompions ? En supposant que notre modèle ne soit pas vraiment meilleur que cette base de référence, notre produit aurait-il encore de la valeur ?

Cela s'applique bien aux exemples de compréhension et de génération du langage naturel tels que les *chatbots*, la traduction, les questions-réponses et la synthèse. Souvent, dans des tâches de synthèse, par exemple, il suffit d'extraire les principaux mots-clés et catégories couvertes par un article pour répondre aux besoins de la plupart des utilisateurs.

Q : Une fois que vous disposez de l'ensemble de votre pipeline, comment identifiez-vous le goulet d'étranglement de l'impact ?

R : Vous devriez commencer par imaginer que le goulet d'étranglement de l'impact est résolu, et vous demander si l'effort que vous avez estimé pour cela en vaut la peine. J'encourage les scientifiques à composer un tweet et les entreprises à rédiger un communiqué de presse avant même de se lancer dans un projet. Cela leur permet d'éviter de travailler sur quelque chose juste parce qu'ils l'ont trouvé cool et de mettre en contexte l'impact des résultats en fonction de l'effort fourni.

L'idéal est que vous puissiez présenter les résultats indépendamment de la sortie finale : si vous n'obtenez pas la meilleure sortie, cela a-t-il quand même un impact ? Avez-vous appris quelque chose ou validé certaines hypothèses ? Un moyen d'y parvenir est de mettre en place une infrastructure permettant de réduire l'effort nécessaire pour le déploiement.

Chez LinkedIn, nous avions accès à un élément de conception très utile, une petite fenêtre avec quelques lignes de texte et des hyperliens, que nous pouvions personnaliser avec nos données. Il était ainsi plus facile de lancer des expériences pour des projets tels que des recommandations d'emploi, puisque le design était déjà approuvé. Comme l'investissement en ressources était faible, l'impact n'avait pas à être aussi important, ce qui a permis d'accélérer le cycle d'itération. La barrière se situe alors au niveau des préoccupations hors ingénierie, telles que l'éthique, l'équité et l'image de marque.

Q : Comment décidez-vous des techniques de modélisation à utiliser ?

R : La première ligne de défense consiste à examiner les données vous-même. Supposons que nous voulons construire un modèle pour recommander des groupes aux utilisateurs de LinkedIn. Une manière naïve serait de recommander le groupe le plus populaire contenant le nom de son entreprise dans l'intitulé du groupe. Après avoir examiné quelques échantillons, nous avons découvert que l'un des groupes les plus populaires dans le cas de la société Oracle était « Oracle ça craint ! », ce qui aurait été un groupe terrible à recommander aux employés d'Oracle.

Il est toujours utile de consacrer des efforts manuels à l'examen des entrées et des sorties de votre modèle. Faites défiler une série d'échantillons pour voir si quelque chose vous semble bizarre. Dans la direction de mon département chez IBM, on avait l'habitude de faire quelque chose manuellement pendant une heure avant de commencer à travailler.

L'examen de vos données vous aide à réfléchir à de bonnes heuristiques, à des modèles et à des moyens de recadrer le produit. Si vous classez les échantillons dans votre jeu de données par fréquence, vous pourriez même être en mesure d'identifier et d'étiqueter rapidement 80 % de vos cas d'utilisation.

Chez Jawbone, par exemple, les gens saisissaient des « phrases » pour consigner le contenu de leurs repas. Une fois étiqueté le « top 100 » de ces phrases à la main, nous en avions couvert 80 % et nous avions une idée précise des principaux problèmes que nous aurions à résoudre, comme les variétés de l'encodage de texte et des langues.

La dernière ligne de défense est d'avoir une main-d'œuvre diversifiée qui regarde les résultats. Cela vous permettra de repérer les cas où un modèle fait preuve de discrimination, par exemple en étiquetant vos amis comme étant des gorilles, ou se montrera insensible en évoquant des expériences passées douloureuses avec sa rétrospective intelligente « l'année dernière à la même époque ».

Conclusion

Comme nous l'avons vu, la construction d'une application basée sur le ML commence par l'évaluation de la faisabilité et le choix d'une approche. Le plus souvent, le choix d'une approche supervisée est la façon la plus simple de commencer. Parmi celles-ci, la classification, l'extraction de connaissances, l'organisation sous forme de catalogues ou les modèles génératifs sont les paradigmes les plus courants dans la pratique.

Lorsque vous sélectionnez une approche, vous devez déterminer la facilité avec laquelle vous pourrez accéder à des données fortement ou faiblement étiquetées, ou sinon à n'importe quelles données. Vous devez ensuite comparer les modèles et les jeux de données potentiels en définissant un objectif pour le produit et en choisissant l'approche de modélisation qui vous permet le mieux d'atteindre cet objectif.

Nous avons illustré ces étapes pour notre Éditeur ML, en choisissant de commencer par des heuristiques simples et une approche basée sur la classification. Enfin, nous avons montré comment des leaders comme Monica Rogati ont appliqué ces pratiques pour transmettre avec succès des modèles basés sur le ML aux utilisateurs.

Maintenant que nous avons choisi une approche initiale, il est temps de définir des indicateurs de réussite et de créer un plan d'action pour progresser régulièrement. Il s'agira de fixer des exigences minimales en matière de performances, de se plonger dans les ressources de modélisation et de données disponibles, et de construire un prototype simple.

Nous aborderons tous ces sujets au Chapitre 2.

CHAPITRE 2

Créer un plan

Dans le chapitre précédent, nous avons abordé la manière d'estimer si le passage par le ML est nécessaire, de trouver où il pourrait être utilisé de la manière la plus adéquate possible et de convertir un objectif de produit en un cadre de développement ML le plus approprié qui soit. Dans le présent chapitre, nous aborderons l'emploi de métriques pour suivre la progression du ML et du produit, et comparer différentes implémentations ML. Ensuite, nous identifierons des méthodes pour construire une base de référence et planifier des itérations de modélisation.

J'ai eu la malheureuse opportunité de voir de nombreux projets de ML condamnés dès le départ en raison d'un mauvais ajustement entre les métriques du produit et les métriques des modèles. Un nombre encore plus grand de projets échouent en produisant de bons modèles, mais qui ne sont pas utiles pour un produit, plutôt qu'en raison de difficultés de modélisation. C'est pourquoi j'ai voulu consacrer un chapitre aux métriques et à la planification.

Nous vous donnerons ici des conseils pour tirer parti des ressources existantes et des contraintes de votre problème afin d'élaborer un plan d'action, qui simplifiera considérablement tout projet de ML.

Commençons par définir plus en détail les indicateurs de performance.

Mesurer le succès

En matière de ML, le premier modèle que nous construisons devrait être le modèle le plus simple qui puisse répondre aux besoins d'un produit, car la génération et l'analyse des résultats sont le moyen le plus rapide de progresser en ML. Dans le chapitre précédent, nous avons abordé trois approches potentielles de complexité croissante pour l'Éditeur ML. Les voici en guise de rappel :

Base de référence : concevoir des heuristiques basées sur la connaissance du domaine

Nous pourrions commencer par définir simplement nous-mêmes les règles, en nous basant sur des connaissances préalables quant à ce qui fait qu'un contenu est bien écrit. Nous testerons ces règles en vérifiant si elles permettent de faire la différence entre un texte bien écrit et un texte mal écrit.

Modèle simple : classifier le texte comme étant bon ou mauvais, et utiliser le classifieur pour générer des recommandations

Nous pourrions alors entraîner un modèle simple pour différencier les bonnes et les mauvaises questions. Si le modèle fonctionne bien, nous pouvons ensuite l'inspecter pour voir quelles sont les caractéristiques qu'il a trouvées hautement prédictives d'une bonne question, et utiliser ces caractéristiques comme recommandations.

Modèle complexe : entraîner de bout en bout un modèle allant du mauvais texte au bon texte

C'est l'approche la plus complexe, tant en termes de modèle que de données, mais si nous disposions des ressources nécessaires pour collecter les données d'entraînement ainsi que pour construire et maintenir un modèle complexe, nous pourrions résoudre directement les exigences du produit.

Toutes ces approches sont différentes et peuvent évoluer au fur et à mesure que nous apprenons à partir des prototypes en cours de route, mais lorsque vous travaillez sur le ML, vous devriez définir un ensemble commun de métriques pour comparer le succès des pipelines de modélisation.

Vous n'avez pas toujours besoin du ML.

Vous avez peut-être remarqué que l'approche de base ne repose pas du tout sur le ML. Comme nous l'avons vu au Chapitre 1, certaines caractéristiques ne nécessitent pas de faire appel à l'apprentissage automatique. Il est également important de réaliser que même les caractéristiques qui pourraient bénéficier du ML peuvent souvent simplement utiliser une heuristique pour leur première version. Une fois que l'heuristique a été utilisée, vous pouvez même réaliser que vous n'avez pas du tout besoin du ML.

Construire une heuristique est aussi souvent le moyen le plus rapide de produire une caractéristique. Une fois celle-ci créée et utilisée, vous aurez une vision plus claire des besoins de vos utilisateurs. Cela vous permettra d'évaluer si vous avez besoin du ML, et de sélectionner une approche de modélisation.

Dans la plupart des cas, commencer sans ML est le moyen le plus rapide de construire un produit ML.

Dans cette optique, nous couvrirons quatre catégories de performances qui ont un grand impact sur l'utilité de tout produit de ML : les métriques produit, les métriques du modèle, l'état de fraîcheur et la rapidité. Une définition claire de ces métriques nous permettra de mesurer avec exactitude les performances de chaque itération.

Performance produit

Nous avons parlé de l'importance de commencer par un objectif clair pour le produit ou les caractéristiques. Une fois que cet objectif est clarifié, il convient de définir une métrique pour juger de son succès. Cette métrique doit être distincte de toute autre métrique liée au modèle, et ne doit être qu'un reflet du succès du produit. Les métriques produit peuvent être aussi simples que le nombre d'utilisateurs qu'une caractéristique attire, ou être plus nuancées, comme le taux de clics (TDC) sur les recommandations que nous fournissons.

Les métriques du produit sont en fin de compte les seules qui comptent, car elles représentent les objectifs poursuivis pour votre produit ou votre caractéristique. Toutes les autres métriques devraient être utilisées comme des outils pour améliorer celles du produit. Toutefois, les métriques du produit ne doivent pas nécessairement être uniques. Si la plupart des projets ont tendance à se focaliser sur l'amélioration d'une certaine métrique, leur impact est souvent mesuré en termes de métriques multiples, y compris celles qui servent de garde-fous, et qui ne doivent pas descendre en dessous d'un seuil fixé. Par exemple, un projet de ML peut viser à augmenter une métrique donnée, telle que le TDC, tout en en maintenant d'autres stables, comme la durée moyenne d'une session utilisateur, par exemple.

Pour notre Éditeur ML, nous choisirons une métrique qui mesure l'utilité d'une recommandation. Par exemple, nous pourrions prendre pour cela la proportion de fois où les utilisateurs suivent les suggestions. Afin de calculer une telle métrique, l'interface de l'Éditeur ML devrait capturer le fait qu'un utilisateur approuve une suggestion, par exemple en la superposant au-dessus de la saisie et en la rendant cliquable.

Nous avons vu que chaque produit se prête à de nombreuses approches ML potentielles. Pour mesurer l'efficacité de telle ou telle approche, vous devriez effectuer le suivi des performances du modèle.

Performance du modèle

Pour la plupart des produits en ligne, la métrique ultime du produit, celle qui détermine le succès d'un modèle, est la proportion de visiteurs qui utilisent les résultats de ce modèle par rapport à l'ensemble des visiteurs qui pourraient en bénéficier. Dans le cas d'un système de recommandation, par exemple, la performance est souvent jugée en mesurant le nombre de personnes qui cliquent sur les produits recommandés (voyez le Chapitre 8 pour les pièges potentiels de cette approche).

Lorsqu'un produit est toujours en cours de développement et n'est pas encore déployé, il n'est pas possible de mesurer les paramètres d'utilisation. Pour juger tout de même des progrès, il est important de définir une métrique de succès distincte, appelée *métrique hors ligne* ou encore *métrique de modèle*. Une bonne métrique hors ligne doit pouvoir être évaluée sans exposer un

modèle aux utilisateurs, et être aussi corrélée que possible avec les métriques et les objectifs du produit.

Différentes approches de modélisation utilisent différentes métriques de modèle, et le changement d'approche peut nettement faciliter l'atteinte d'un niveau de performance de modélisation suffisant pour atteindre les objectifs du produit.

Par exemple, supposons que vous essayez de proposer des suggestions utiles aux utilisateurs lorsqu'ils tapent une requête de recherche sur un site de commerce en ligne. Vous mesurerez le succès de cette caractéristique en mesurant le TDC, c'est-à-dire la fréquence à laquelle les utilisateurs cliquent sur les suggestions que vous faites.

Pour générer les suggestions, vous pourriez construire un modèle qui tente de deviner les mots qu'un utilisateur va taper et lui présenter la phrase complète prédite au fur et à mesure qu'il l'écrit. Vous pourriez mesurer la performance de ce modèle en calculant son exactitude au niveau des mots, en calculant la fréquence à laquelle il prédit la prochaine série de mots correcte. Un tel modèle devrait atteindre une exactitude extrêmement élevée pour aider à augmenter le TDC du produit, car il suffirait d'une erreur de prédiction d'un seul mot pour rendre une suggestion inutile. Cette approche est illustrée sur le côté gauche de la Figure 2.1.

Une autre approche consisterait à entraîner un modèle qui classifie les entrées des utilisateurs en catégories dans votre catalogue et suggère les trois catégories les plus probables prédites. Vous mesureriez la performance de votre modèle en utilisant l'exactitude sur toutes les catégories plutôt sur chaque mot saisi. Comme le nombre de catégories dans un catalogue est beaucoup plus petit que le vocabulaire de n'importe quelle langue, ce serait une métrique de modélisation nettement plus simple à optimiser. En outre, le modèle n'a besoin de prédire correctement qu'une seule catégorie pour générer un clic. Il est beaucoup plus facile pour ce modèle d'augmenter le TDC du produit. Vous pouvez voir une maquette de la façon dont cette approche fonctionnerait en pratique sur le côté droit de la Figure 2.1.

Comme vous pouvez le voir, de petites modifications de l'interaction entre le modèle et le produit peuvent permettre d'utiliser une approche de modélisation plus simple et de fournir des résultats plus fiables. Voici quelques autres exemples de mise à jour d'une application pour faciliter une tâche de modélisation :

- Modifier une interface afin que les résultats d'un modèle puissent être omis s'ils sont inférieurs à un seuil de confiance. Lors de la construction d'un modèle servant à compléter automatiquement une phrase tapée par l'utilisateur, par exemple, le modèle peut ne donner de bons résultats que sur un sous-ensemble de phrases. Nous pouvons implémenter une logique pour ne montrer une suggestion aux utilisateurs que si le score de confiance du modèle dépasse 90 %.

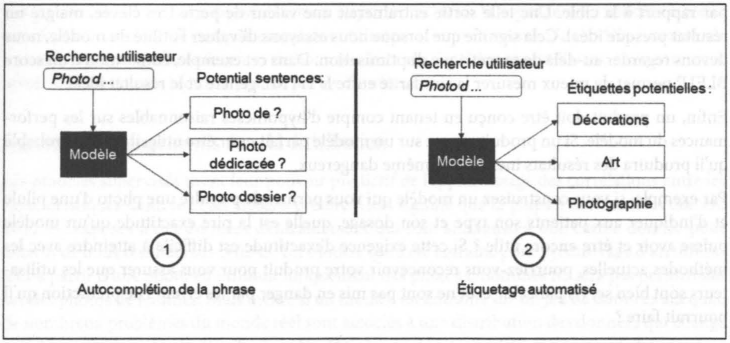


Figure 2.1 : Une légère modification d'un produit peut faciliter considérablement les tâches de modélisation.

- Présenter quelques autres prédictions ou heuristiques en plus de la prédiction principale d'un modèle. Par exemple, la plupart des sites Web affichent plus d'une recommandation suggérée par un modèle. En affichant cinq éléments candidats au lieu d'un seul, la probabilité pour que les suggestions soient utiles aux utilisateurs augmente, même si le modèle reste le même.
- Communiquer aux utilisateurs le fait qu'un modèle est encore en phase d'expérimentation et leur donner la possibilité de donner leur avis. Lorsqu'ils détectent automatiquement une langue qui n'est pas la langue maternelle de l'utilisateur et la traduisent pour lui, les sites Web ajoutent souvent un bouton permettant à l'utilisateur de faire savoir si la traduction est exacte et utile.

Même lorsqu'une approche de modélisation est appropriée pour un problème, il peut parfois être utile de générer des métriques de modèle supplémentaires présentant une meilleure corrélation avec les performances du produit.

J'ai travaillé une fois avec un spécialiste des données qui a construit un modèle pour générer du code HTML à partir de croquis dessinés à la main pour des sites Web simples[4]. La métrique d'optimisation du modèle compare chaque jeton HTML prédit au jeton correct en utilisant la perte d'entropie croisée. L'objectif du produit est toutefois que le HTML généré produise un site Web qui ressemble au croquis d'entrée, quel que soit l'ordre des jetons.

L'entropie croisée ne tient pas compte de l'alignement : si un modèle génère une séquence HTML correcte à l'exception d'un jeton supplémentaire au début, tous les jetons seront décalés de un

4 Voir l'article d'Ashwin Kumar, « *Automated front-end development using deep learning* » (`https://bit.ly/2Vc6HTm`).

par rapport à la cible. Une telle sortie entraînerait une valeur de perte très élevée, malgré un résultat presque idéal. Cela signifie que lorsque nous essayons d'évaluer l'utilité du modèle, nous devons regarder au-delà de sa métrique d'optimisation. Dans cet exemple, l'utilisation d'un score BLEU[5] permet de mieux mesurer la similarité entre le HTML généré et le résultat idéal.

Enfin, un produit doit être conçu en tenant compte d'hypothèses raisonnables sur les performances du modèle. Si un produit repose sur un modèle parfait pour être utile, il est très probable qu'il produira des résultats inexacts ou même dangereux.

Par exemple, si vous construisez un modèle qui vous permet de prendre une photo d'une pilule et d'indiquer aux patients son type et son dosage, quelle est la pire exactitude qu'un modèle puisse avoir et être encore utile ? Si cette exigence d'exactitude est difficile à atteindre avec les méthodes actuelles, pourriez-vous reconcevoir votre produit pour vous assurer que les utilisateurs sont bien servis par celui-ci et ne sont pas mis en danger par les erreurs de prédiction qu'il pourrait faire ?

Pour notre étude de cas, le produit que nous voulons construire fournira des conseils en matière de rédaction. La plupart des modèles de ML ont certaines entrées pour lesquelles ils excellent et d'autres avec lesquelles ils devront se battre. Du point de vue du produit, si nous ne sommes pas en mesure d'aider – nous devons nous assurer que nous ne faisons pas de mal – nous aimerions limiter la quantité de temps pendant lequel nous produisons un résultat qui est pire que l'entrée. Comment pourrions-nous exprimer cela dans des métriques de modèle ?

Supposons que nous construisions un modèle de classification qui tente de prédire si une question est bonne en fonction du nombre de votes positifs qu'elle a reçus. La précision du classifieur serait définie comme étant la proportion de questions qui sont vraiment bonnes par rapport à celles qu'il prédit comme telles. D'un autre côté, son rappel serait la proportion de questions qui sont prédites comme étant bonnes par rapport à toutes les bonnes questions du jeu de données.

Si nous voulons avoir des conseils toujours pertinents, nous voudrions privilégier la *précision* du modèle, car lorsqu'un modèle de haute précision classifie une question comme étant bonne (et formule donc une recommandation), il y a de fortes chances que cette question le soit effectivement. Une haute précision signifie que lorsque nous faisons une recommandation, elle aura tendance à être correcte. Pour en savoir plus sur les raisons pour lesquelles les modèles avec une haute précision sont plus utiles pour la rédaction de recommandations, n'hésitez pas à vous référer à l'interview de Chris Harland à la fin du Chapitre 8.

Nous mesurons ces métriques en examinant les résultats d'un modèle sur un jeu de validation représentatif. Nous nous pencherons sur ce que cela signifie dans la section « Évaluer votre modèle : aller au-delà de l'exactitude » du Chapitre 5, mais, pour l'instant, considérez un jeu de validation comme un ensemble de données issues de l'entraînement et utilisées pour estimer les performances de votre modèle sur des données qui n'ont pas encore été vues.

5 Voir l'adresse `https://fr.wikipedia.org/wiki/BLEU_(algorithme)`.

Les performances initiales du modèle sont importantes, mais la capacité d'un modèle à rester utile face à l'évolution du comportement des utilisateurs l'est tout autant. Un modèle entraîné sur un certain jeu de données fournira de bons résultats sur des données similaires, mais comment savoir si nous devons mettre à jour ce jeu de données ?

Fraîcheur et changements dans la distribution

Les modèles supervisés tirent leur pouvoir prédictif de l'apprentissage des corrélations entre les caractéristiques d'entrée et les cibles à prédire. Cela signifie que la plupart des modèles doivent avoir été exposés à des données d'entraînement qui sont similaires à une entrée donnée pour obtenir de bons résultats sur celle-ci. Un modèle qui a été entraîné pour prédire l'âge d'un utilisateur à partir d'une photo en utilisant uniquement des photos d'hommes ne sera pas performant sur des photos de femmes. Mais même si un modèle est entraîné sur un jeu de données adéquat, de nombreux problèmes du monde réel sont associés à une distribution des données qui change avec le temps. Lorsque la distribution des données évolue, le modèle doit souvent changer également afin de maintenir le même niveau de performance.

Imaginons qu'après avoir remarqué l'impact de la pluie sur la circulation à San Francisco, vous construisiez un modèle pour prédire les conditions de circulation en fonction de la quantité de pluie tombée la semaine dernière. Si vous avez construit votre modèle en octobre en utilisant les données des trois derniers mois, ce modèle a probablement été entraîné sur des données correspondant à des précipitations quotidiennes moyennes inférieures à un pouce (voir la Figure 2.2 pour un exemple d'une telle distribution). À l'approche de l'hiver, les précipitations moyennes se rapprocheront d'environ 3 pouces, ce qui est plus élevé que tout ce à quoi le modèle a été exposé pendant l'entraînement, comme vous pouvez le constater également sur la Figure 2.2. Si le modèle n'est pas entraîné sur des données plus récentes, il aura du mal à continuer à bien fonctionner.

En général, un modèle peut donner de bons résultats sur des données qu'il n'a jamais vues auparavant, à condition qu'elles soient suffisamment similaires aux données auxquelles il a été exposé pendant l'entraînement.

Tous les problèmes n'ont pas les mêmes exigences en matière de fraîcheur. Des services de traduction de langues anciennes peuvent s'attendre à ce que les données qu'ils exploitent restent relativement constantes, tandis que les moteurs de recherche doivent être construits en partant du principe qu'ils devront évoluer aussi vite que les utilisateurs changent leurs habitudes de recherche.

En fonction de votre problème spécifique, vous devez réfléchir à la difficulté de garder les modèles à jour, disons *frais*. À quelle fréquence devrez-vous recycler les modèles, et combien cela vous coûtera-t-il à chaque fois pour le faire ?

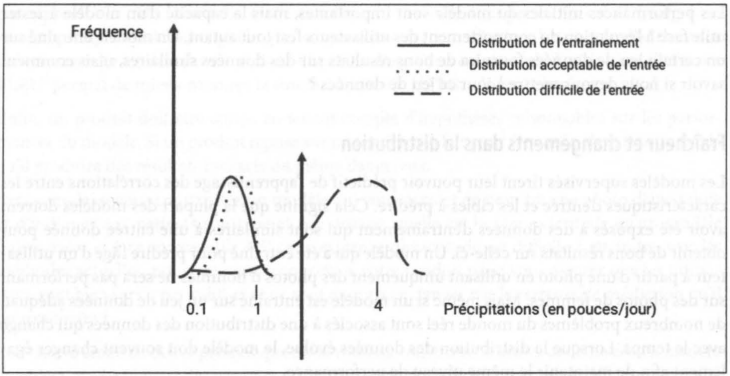


Figure 2.2 : Évolution de distributions.

Pour notre Éditeur ML, nous imaginons que la cadence à laquelle la définition d'une « prose bien formulée » change est relativement faible, peut-être de l'ordre d'une année. Les exigences en matière de fraîcheur seraient cependant modifiées si nous ciblions des domaines spécifiques. Par exemple, la bonne façon de poser une question concernant les mathématiques changera beaucoup plus lentement que la meilleure formulation de questions traitant des tendances musicales. Comme nous estimons que les modèles devront être recyclés chaque année, nous aurons besoin de données fraîches pour nous entraîner de manière annuelle.

Notre base de référence et notre modèle simple peuvent tous deux tirer des enseignements de données non appariées, ce qui simplifie le processus de collecte de données (il suffirait de trouver de nouvelles questions provenant de l'année dernière). Le modèle complexe nécessite des données appariées, ce qui signifie que nous devrons trouver des exemples de mêmes phrases, dites de manière « bonne » et « mauvaise », et ce chaque année. Par conséquent, il sera beaucoup plus difficile de satisfaire à l'exigence de fraîcheur que nous avons définie pour un modèle nécessitant des données appariées, car il est plus long d'acquérir un jeu de données actualisées.

Pour la plupart des applications, la popularité peut contribuer à alléger les exigences en matière de collecte de données. Si notre service de formulation de questions devient viral, nous pourrions ajouter un bouton permettant aux utilisateurs d'évaluer la qualité des résultats. Nous pourrions alors recueillir les entrées antérieures des utilisateurs ainsi que les prédictions du modèle et les évaluations associées, et utiliser ces données en tant que jeu d'entraînement.

Pour qu'une application soit populaire, elle doit cependant être utile. Souvent, il faut pour cela répondre aux demandes des utilisateurs en temps utile. La vitesse à laquelle un modèle peut fournir des prédictions est donc un facteur important à prendre en compte.

Vitesse

Dans l'idéal, un modèle devrait fournir une prédiction rapidement. Cela permet aux utilisateurs d'interagir plus facilement avec lui et simplifie la diffusion d'un modèle de manière simultanée à de nombreux utilisateurs. Quelle doit donc être la rapidité d'un modèle ? Pour certains cas d'utilisation, comme la traduction d'une courte phrase, les utilisateurs s'attendront à une réponse immédiate. Pour d'autres, comme un diagnostic médical, les patients seraient heureux d'attendre 24 heures si cela signifiait qu'ils obtiendraient les résultats les plus exacts.

Dans notre cas, nous envisagerons deux façons possibles de faire des suggestions : par le biais d'une boîte de soumission où l'utilisateur écrit, clique sur Soumettre et obtient un résultat, ou bien par une mise à jour dynamique à chaque fois que l'utilisateur saisit une nouvelle lettre. Bien que nous puissions privilégier cette dernière solution, car elle permettrait de rendre l'outil beaucoup plus interactif, elle nécessiterait des modèles fonctionnant beaucoup plus rapidement.

On pourrait imaginer qu'un utilisateur attende quelques petites secondes pour obtenir un résultat une fois qu'il a cliqué sur un bouton de soumission, mais, pour qu'un modèle s'exécute au fur et à mesure de la saisie du texte par un utilisateur, il faudrait qu'il fonctionne en nettement moins d'une seconde. Les modèles les plus puissants prennent plus de temps pour traiter les données, aussi, lorsque nous itérerons dans les modèles, nous garderons cette exigence à l'esprit. Tout modèle que nous utilisons devrait être capable de traiter l'intégralité d'un pipeline pour un échantillon en moins de deux secondes.

Les durées d'exécution des inférences des modèles augmentent à mesure que ces derniers deviennent plus complexes. La différence est significative, même dans un domaine où chaque point de données individuel peut être relativement petit comme en TLN (par opposition aux tâches sur les vidéos en direct, par exemple). Sur les données textuelles utilisées pour l'étude de cas de ce livre, par exemple, un LSTM est environ trois fois plus lent qu'une forêt aléatoire (environ 22 ms pour un LSTM, alors que la forêt aléatoire ne prend que 7 ms). Sur un point de données individuel, ces différences sont faibles, mais elles peuvent rapidement se cumuler lorsqu'il faut exécuter des inférences sur des dizaines de milliers d'échantillons à la fois.

Pour les applications complexes où un appel d'inférence sera associé à de multiples appels réseau ou à des requêtes sur des bases de données, le temps d'exécution du modèle peut devenir court comparé au reste de la logique de l'application. Dans ces cas, la vitesse desdits modèles devient moins problématique.

En fonction de votre problème, vous pourriez envisager d'autres catégories, telles que les contraintes matérielles, le temps de développement et la maintenabilité. Il est important de bien comprendre vos besoins avant de sélectionner un modèle afin de vous assurer que vous le choisissez en connaissance de cause.

Une fois que vous avez identifié les exigences et les métriques associées, il est temps de commencer à élaborer un plan. Il faut pour cela estimer les défis à relever. Dans la section suivante, je

vais traiter de méthodes permettant de tirer parti des travaux antérieurs et d'explorer un jeu de données pour décider de la suite des opérations.

Estimer la portée et les défis

Comme nous l'avons vu, les performances en matière de ML sont souvent rapportées en termes de métriques des modèles. Bien que ces métriques soient utiles, elles devraient être utilisées pour améliorer les métriques du produit telles que nous les avons définies, et qui représentent la tâche réelle que nous essayons de résoudre. Lorsque nous itérons sur un pipeline, nous devrions garder à l'esprit les métriques du produit et chercher à les améliorer.

Les outils que nous avons couverts jusqu'à présent nous aideront à déterminer si un projet vaut réellement la peine d'être abordé, et à mesurer notre performance actuelle. La prochaine étape logique consiste à esquisser un plan d'attaque pour estimer la portée et la durée d'un projet et anticiper les éventuels obstacles.

En matière de ML, le succès exige généralement de bien comprendre le contexte de la tâche, d'acquérir un bon jeu de données et de construire un modèle approprié.

Nous allons aborder chacune de ces catégories dans la section suivante.

Exploiter l'expertise dans le domaine

Le modèle le plus simple par lequel nous pouvons commencer est un modèle heuristique : une bonne règle empirique basée sur la connaissance du problème et des données. La meilleure façon de concevoir une heuristique consister à observer ce que les experts font actuellement. La plupart des applications pratiques ne sont pas entièrement nouvelles. Comment les gens résolvent-ils actuellement le problème que vous essayez vous-même de résoudre ?

La deuxième meilleure façon de concevoir l'heuristique est d'examiner vos données. En vous basant sur votre jeu de données, comment résoudre cette tâche si vous faites le travail manuellement ?

Pour identifier les bonnes heuristiques, je recommande soit d'apprendre des experts dans le domaine concerné, soit de se familiariser avec les données. Je vais décrire les deux de manière un peu plus détaillée dans ce qui suit.

Apprendre des experts

Pour de nombreux domaines que nous pourrions vouloir automatiser, apprendre auprès d'experts dans ces domaines peut nous faire gagner des dizaines d'heures de travail. Si nous essayons de construire un système de maintenance prédictive pour les équipements d'une entreprise, par exemple, nous devrions commencer par contacter un directeur d'usine pour comprendre quelles

hypothèses nous pouvons raisonnablement faire. Il pourrait s'agir de comprendre la fréquence actuelle des interventions de maintenance, les symptômes qui indiquent généralement qu'une machine aura bientôt besoin d'un entretien, et les exigences légales concernant la maintenance.

Il existe bien sûr des domaines où il sera probablement difficile de trouver des experts, comme dans le cas de données propriétaires pour un cas d'utilisation inédit, par exemple pour prédire l'utilisation d'une caractéristique unique d'un site Web. Dans ces cas, cependant, nous pouvons souvent trouver des professionnels qui ont dû faire face à des problèmes similaires et tirer des enseignements de leurs expériences.

Cela nous permettra d'apprendre les caractéristiques utiles dont nous pouvons tirer parti, de trouver les pièges à éviter et, surtout, de nous empêcher de réinventer la roue, chose pour laquelle de nombreux spécialistes des données ont une mauvaise réputation.

Examiner les données

Comme le mentionnent dans leurs entretiens Monica Rogati (Chapitre 1) et Robert Munro (Chapitre 4), il est essentiel d'examiner les données avant de commencer la modélisation.

L'analyse exploratoire des données (AED) est le processus de visualisation et d'exploration d'un jeu de données, souvent pour avoir une intuition sur un problème d'entreprise donné. L'AED est un élément essentiel de la construction de tout produit basé sur des données. En plus de l'AED, il est crucial d'étiqueter individuellement les échantillons de la manière dont vous espérez qu'un modèle le fasse. Cela permet de valider les hypothèses et de confirmer que vous avez choisi des modèles qui peuvent exploiter votre jeu de données de manière appropriée.

Le processus d'AED vous permettra de comprendre les tendances dans vos données, et le fait de les étiqueter vous-même vous obligera à construire un ensemble d'heuristiques pour résoudre votre problème. Après avoir effectué les deux étapes précédentes, vous devriez avoir une idée plus précise du type de modèles qui vous servira le mieux, ainsi que des stratégies supplémentaires de collecte et d'étiquetage des données dont vous pourriez avoir besoin.

La prochaine étape logique consiste à voir comment d'autres ont abordé des problèmes de modélisation similaires.

Se tenir sur les épaules des géants

D'autres personnes ont-elles résolu des problèmes similaires ? Si oui, la meilleure façon de commencer est de comprendre et de reproduire les résultats existants. Recherchez des implémentations publiques, soit avec des modèles similaires, soit avec des jeux de données similaires, soit les deux.

Dans l'idéal, cela impliquerait de trouver du code source ouvert et un jeu de données disponibles, mais ceux-ci ne sont pas toujours faciles à obtenir, surtout pour des produits très spéci-

fiques. Néanmoins, le moyen le plus rapide de démarrer un projet de ML est de reproduire des résultats existants et de s'appuyer sur eux.

Dans un domaine qui comporte autant de pièces mobiles que le ML, il est crucial de se tenir sur les épaules des géants.

Si vous envisagez d'utiliser des codes ou des jeux de données open source dans votre travail, veuillez vous assurer que vous êtes autorisé à le faire. La plupart des dépôts et des jeux de données seront accompagnés d'une licence qui définit les usages acceptables. En outre, mentionnez toute source que vous utilisez, idéalement en faisant référence au travail original correspondant.

Il peut souvent être judicieux d'établir une preuve de concept convaincante avant d'engager des ressources importantes dans un projet. Avant d'utiliser du temps et de l'argent pour étiqueter des données, par exemple, nous devons nous convaincre que nous pouvons construire un modèle qui tirera des enseignements de ces données.

Donc, comment trouver un moyen efficace de commencer ? Comme la plupart des sujets que nous aborderons dans ce livre, celui-ci comprend deux parties principales : les données et le code.

Données libres de droits

Il se peut que vous ne puissiez pas toujours trouver un jeu de données qui corresponde à vos désirs, mais vous pouvez souvent trouver un jeu de données ayant une nature suffisamment similaire pour être utile. Que signifie un jeu de données similaire dans ce contexte ? Il est utile ici de réfléchir aux modèles de ML en tant que mappage entre une entrée et une sortie. Dans cette optique, cela signifie simplement un jeu de données avec des types d'entrée et de sortie similaires (mais pas nécessairement dans le même domaine).

Fréquemment, des modèles utilisant des entrées et des sorties similaires peuvent être appliqués à des contextes entièrement différents. Sur le côté gauche de la Figure 2.3 se trouvent deux modèles qui prédisent chacun une séquence de texte à partir d'une image en entrée. L'un est utilisé pour décrire des photos, tandis que l'autre génère un code HTML pour un site Web à partir d'une capture d'écran dudit site. De même, le côté droit de la Figure 2.3 montre un modèle qui prédit un type d'aliment à partir d'une description textuelle, et un autre qui prédit un genre de musique à partir d'une transcription de partition (selon la notation anglo-saxonne).

Par exemple, disons que nous essayons de construire un modèle pour prédire l'audience d'articles de presse, mais que nous avons du mal à trouver un jeu de données concernant les articles de presse et le nombre de lecteurs correspondant. Nous pourrions commencer avec comme jeu de données des statistiques librement accessibles du trafic sur les pages Wikipédia et y entraîner un modèle prédictif. Si nous sommes satisfaits de ses performances, il est raisonnable de penser que, compte tenu d'un jeu de données sur le nombre de consultations d'un article de presse, notre modèle pourrait fonctionner raisonnablement bien. Trouver un jeu de données similaire

peut aider à prouver la validité d'une approche, et rend plus raisonnable le fait de dépenser des ressources pour acquérir des données.

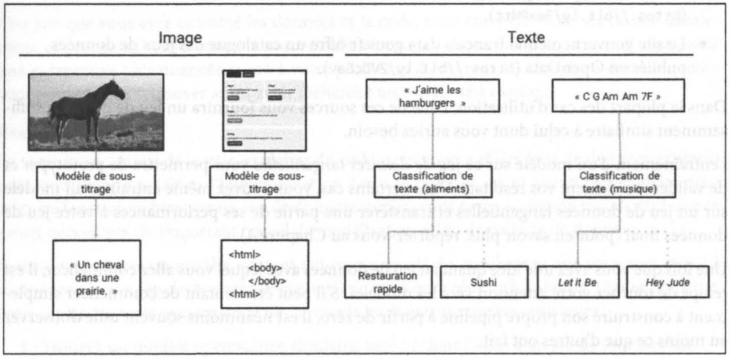


Figure 2.3 : Différents modèles avec des entrées et des sorties similaires

Cette méthode fonctionne également lorsque l'on travaille sur des données propriétaires. Souvent, le jeu de données dont vous avez besoin pour une tâche de prédiction n'est pas facile d'accès. Dans certains cas, les données dont vous auriez besoin ne sont actuellement pas collectées. Dans une telle situation, la construction d'un modèle performant sur un jeu de données similaire peut souvent être le meilleur moyen de convaincre les parties prenantes de l'utilité de construire un nouveau pipeline de collecte de données, ou encore de faciliter l'accès à un pipeline existant.

Lorsqu'il s'agit de données accessibles au public, de nouvelles sources et collections de données apparaissent régulièrement. En voici quelques-unes que j'ai trouvées utiles :

- Le site Internet Archive conserve un ensemble de jeux de données, notamment des données concernant des sites Web, des vidéos et des livres (`https://archive.org/details/datasets`, en anglais).
- La section r/datasets de Reddit est consacrée au partage de jeux de données (`https://www.reddit.com/r/datasets/`).
- La page d'accès aux jeux de données de Kaggle offre une large sélection dans une grande variété de domaines (`https://www.kaggle.com/datasets`).
- Le dépôt dédié à l'apprentissage automatique de l'UCI est une vaste ressource de jeux de données ML (`https://archive.ics.uci.edu/ml/index.php`).
- La recherche de jeux de données sur Google couvre un large éventail de données accessibles (`https://datasetsearch.research.google.com/`).

- Le site Common Crawl parcourt et archive les données provenant d'une large partie du Web et met les résultats à la disposition du public (`https://commoncrawl.org/`).
- Wikipédia dispose également d'une liste évolutive de jeux de données concernant le ML (`https://bit.ly/3exPWtt`).
- Le site gouvernemental français data.gouv.fr offre un catalogue des jeux de données publiés en OpenData (`https://bit.ly/2VDcEaV`).

Dans la plupart des cas d'utilisation, l'une de ces sources vous fournira un jeu de données suffisamment similaire à celui dont vous auriez besoin.

L'entraînement d'un modèle sur ce *jeu de données tangentielles* vous permettra de prototyper et de valider rapidement vos résultats. Dans certains cas, vous pouvez même entraîner un modèle sur un jeu de données tangentielles et transférer une partie de ses performances à votre jeu de données final (pour en savoir plus, reportez-vous au Chapitre 4).

Une fois que vous avez une idée quant au jeu de données avec lequel vous allez commencer, il est temps de tourner votre attention vers les modèles. S'il peut être tentant de commencer simplement à construire son propre pipeline à partir de zéro, il est néanmoins souvent utile d'observer au moins ce que d'autres ont fait.

Code open source

La recherche de code existant peut permettre d'atteindre deux objectifs de haut niveau : voir les défis auxquels d'autres ont dû faire face lors de modélisations similaires, et faire apparaître les problèmes potentiels liés au jeu de données choisi. Pour cette raison, je recommande de rechercher à la fois des pipelines qui s'attaquent à votre objectif produit et du code qui fonctionne avec le jeu de données que vous avez sélectionné. Si vous trouvez un exemple, la première étape consisterait à reproduire vous-même ses résultats.

J'ai vu de nombreux spécialistes des données tenter d'exploiter le code ML qu'ils avaient trouvé en ligne pour finalement constater qu'ils étaient incapables d'entraîner les modèles donnés avec un niveau d'exactitude similaire à celui revendiqué par les auteurs. Comme les nouvelles approches ne sont pas toujours accompagnées d'un code bien documenté et fonctionnel, les résultats de ML sont souvent difficiles à reproduire et devraient donc toujours être vérifiés.

Tout comme pour la recherche de données, un bon moyen de trouver des codes base similaires est d'abstraire votre problème en fonction de ses types d'entrée et de sortie, et de trouver des codes base traitant de problèmes ayant des types similaires.

Par exemple, en essayant de générer du code HTML à partir de captures d'écran d'un site Web, Tony Beltramelli, l'auteur de l'article « pix2code» : Generating Code from a Graphical User Interface Screenshot »[6], s'est rendu compte que son problème se résumait à traduire une image en une séquence. Il a tiré parti des architectures existantes et des meilleures pratiques d'un domaine

6 Voir l'adresse `https://arxiv.org/abs/1705.07962`.

plus mature et a également généré des séquences à partir d'images, c'est-à-dire du sous-titrage d'images ! Cela lui a permis d'obtenir d'excellents résultats sur une tâche entièrement nouvelle et de tirer parti d'années de travail dans une application adjacente.

Une fois que vous avez examiné les données et le code, vous êtes prêt à aller de l'avant. Idéalement, ce processus vous a donné quelques indications pour commencer votre travail et acquérir une perspective plus nuancée quant à votre problème. Résumons les situations dans lesquelles vous pouvez vous retrouver après avoir recherché un travail déjà existant.

Rassembler les deux

Comme nous venons de le voir, exploiter du code open source et des jeux de données existants peut contribuer à accélérer l'implémentation. Dans le pire des cas, si aucun des modèles existants ne fonctionne bien sur un jeu de données public, vous savez au moins maintenant que ce projet nécessitera un important travail de modélisation et/ou de collecte de données.

Si vous avez trouvé un modèle existant qui résout une tâche similaire et que vous ayez réussi à l'entraîner sur son jeu de données d'origine, il ne vous reste plus qu'à l'adapter à votre domaine. Pour ce faire, je vous recommande de parcourir les étapes successives suivantes :

1. Trouvez un modèle open source similaire, associé dans l'idéal à un jeu de données sur lequel il a été entraîné, et essayez de reproduire vous-même les résultats de cet entraînement.
2. Une fois que vous avez reproduit les résultats, trouvez un jeu de données qui se rapproche le plus de votre cas d'utilisation, et essayez d'entraîner le modèle précédent sur ce jeu de données.
3. Une fois que vous avez intégré le jeu de données au code ayant servi à l'entraînement, il est temps de juger de la performance de votre modèle à l'aide des métriques que vous avez définies et de commencer à itérer.

Nous explorerons les pièges de chacune de ces étapes et la manière de les surmonter à partir de la deuxième partie du livre. Pour l'instant, revenons à notre étude de cas et passons en revue le processus que nous venons de décrire.

Planifier l'Éditeur ML

Examinons les conseils de rédaction courants et recherchons des jeux de données et des modèles candidats pour l'Éditeur ML.

Plan initial pour un éditeur

Nous devrions commencer par mettre en œuvre des heuristiques basées sur des directives d'écriture courantes. Nous rassemblerons ces règles en recherchant les guides existants pour l'écriture et l'édition, comme ceux décrits dans la section « L'approche la plus simple : être l'algorithme » du Chapitre 1.

Notre jeu de données parfait serait constitué de questions et de leur qualité associée. Tout d'abord, nous devrions rapidement trouver un jeu de données similaire qui soit plus facile à acquérir. Sur la base des performances observées sur ce jeu de données, nous élargirons et approfondirons ensuite notre recherche si nécessaire.

Les messages postés sur les réseaux sociaux et les forums en ligne sont de bons exemples de textes associés à une métrique de qualité. Comme la plupart de ces métriques existent pour favoriser les contenus utiles, elles comprennent souvent des évaluations de qualité telles que « aime » (*like*) ou « vote positif ».

Stack Exchange (`https://stackexchange.com/`), un réseau de communautés de questions-réponses, est un site populaire pour les questions et réponses. Il existe également un dépôt de données entièrement anonymisées de Stack Exchange sur Internet Archive (`https://archive.org/details/stackexchange`), une des sources de données que nous avons mentionnées précédemment. C'est un excellent jeu de données pour commencer.

Nous pouvons construire un modèle initial en utilisant les questions de Stack Exchange et en essayant de prédire le score des votes positifs pour une question à partir de son contenu. Nous profiterons également de cette occasion pour examiner le jeu de données et l'étiqueter, en essayant de trouver des motifs.

Le modèle que nous voulons construire tente de classifier avec exactitude la qualité des textes, pour ensuite fournir des recommandations de rédaction. Il existe de nombreux modèles open source pour la classification des textes. Consultez par exemple sur ce sujet ce tutoriel d'apprentissage de la bibliothèque ML de Python scikit-learn : `https://bit.ly/2VxhJS4`.

Une fois que nous aurons un classifieur fonctionnel, nous verrons comment l'exploiter au Chapitre 7 pour formuler des recommandations.

Maintenant que nous disposons d'un jeu de données initial potentiel, passons aux modèles et décidons par quoi nous devons commencer.

Toujours commencer par un modèle simple

Un point important de ce chapitre est que l'objectif de la construction d'un modèle et d'un jeu de données initial est de produire des résultats informatifs qui guideront les travaux de modélisation et de collecte de données ultérieurs vers un produit plus utile.

En commençant par un modèle simple et en extrayant les tendances de ce qui fait le succès d'une question sur Stack Overflow, nous pouvons rapidement mesurer les performances et itérer sur elles.

L'approche inverse, qui consiste à essayer de construire un modèle parfait à partir de zéro, ne fonctionne pas dans la pratique. La raison en est que le ML est un processus itératif dans lequel le moyen le plus rapide de progresser est de voir comment un modèle échoue. Plus votre modèle échoue rapidement, plus vous progresserez. Nous nous pencherons sur ce processus itératif de manière beaucoup plus détaillée dans la troisième partie.

Nous devons toutefois garder à l'esprit les mises en garde qu'il convient d'émettre pour chaque approche. Par exemple, l'intérêt manifesté pour une question dépend de beaucoup plus de facteurs que de la seule qualité de sa formulation. Le contexte du post, la communauté dans laquelle il a été posté, sa popularité, l'heure à laquelle il a été publié, et bien d'autres détails que le modèle initial ignorera, ont également une grande importance. Pour tenir compte de ces facteurs, nous limiterons notre jeu de données à un sous-ensemble de communautés. Notre premier modèle ignorera toutes les métadonnées relatives à un message, mais nous envisagerons de les intégrer si cela semble nécessaire.

En tant que tel, notre modèle utilise ce que l'on appelle souvent une *étiquette faible*, c'est-à-dire une étiquette qui n'est que faiblement corrélée avec le résultat souhaité. En analysant les performances du modèle, nous déterminerons si cette étiquette contient suffisamment d'informations pour être utile.

Nous avons un point de départ, et nous pouvons maintenant décider de la manière dont nous allons progresser. Faire des progrès réguliers en matière de ML peut souvent sembler difficile en raison de l'aspect imprévisible de la modélisation. Il est difficile de savoir à l'avance dans quelle mesure une approche de modélisation donnée sera couronnée de succès. C'est pourquoi j'aimerais partager avec vous quelques conseils pour progresser de manière régulière.

Pour faire des progrès réguliers : commencer simplement

Il convient de répéter qu'une grande partie du défi dans le ML est similaire à l'un des plus grands défis dans le domaine du logiciel : résister à l'envie de construire des pièces qui ne sont pas encore nécessaires. De nombreux projets de ML échouent parce qu'ils s'appuient sur un plan initial d'acquisition de données et de construction de modèles, et qu'ils n'évaluent pas et ne mettent pas régulièrement à jour ce plan. En raison de la nature stochastique du ML, il est extrêmement difficile de prédire jusqu'où un jeu de données ou un certain modèle nous mènera.

C'est pourquoi il est *vital* de commencer par le modèle le plus simple qui pourrait répondre à vos besoins, de construire un prototype de bout en bout incluant ce modèle, et de juger de ses performances non seulement en termes de métriques d'optimisation, mais aussi en fonction de l'objectif de votre produit.

Commencer par un pipeline simple

Dans la grande majorité des cas, l'examen des performances d'un modèle simple sur un jeu de données initial est le meilleur moyen de décider de la tâche à entreprendre. L'objectif est alors de répéter cette approche pour chacune des étapes suivantes, en apportant de petites améliorations progressives faciles à suivre, plutôt que de tenter de construire le modèle parfait en une seule fois.

Pour cela, nous devrons construire un pipeline qui puisse recevoir des données et renvoyer des résultats. Pour la plupart des problèmes de ML, il faut en fait envisager deux pipelines distincts.

Entraînement

Pour que votre modèle soit capable de faire des prévisions exactes, vous devez d'abord l'entraîner.

Un pipeline d'entraînement ingère toutes les données étiquetées sur lesquelles vous souhaitez effectuer l'apprentissage (pour certaines tâches, les jeux de données peuvent être si volumineux qu'une seule machine ne suffit pas pour tous les faire tenir !) et les transmet à un modèle. Il entraîne ensuite ce modèle sur le jeu de données jusqu'à ce qu'il atteigne des performances satisfaisantes. Le plus souvent, un pipeline d'entraînement est utilisé avec plusieurs modèles pour ensuite comparer leurs performances sur un jeu de données de validation.

Inférence

C'est votre pipeline mis en production. Il sert les résultats d'un modèle entraîné à votre utilisateur.

À un niveau élevé, un pipeline d'inférence commence par accepter les données en entrée et les prétraiter. La phase de prétraitement comprend généralement plusieurs étapes. Le plus souvent, ces étapes comprennent le nettoyage et la validation de l'entrée, la génération des caractéristiques dont un modèle a besoin, et le formatage des données en une représentation numérique appropriée pour un modèle de ML. Dans les systèmes plus complexes, les pipelines doivent aussi souvent aller chercher les informations supplémentaires dont le modèle a besoin, comme des caractéristiques utilisateur stockées dans une base de données, par exemple. Le pipeline fait ensuite passer l'échantillon dans le modèle, applique toutes les logiques de post-traitement et renvoie un résultat.

La Figure 2.4 montre un organigramme d'un pipeline d'inférence et d'entraînement typique. Dans l'idéal, les étapes de nettoyage et de prétraitement devraient être les mêmes pour les pipelines d'entraînement et d'inférence, et ce afin de garantir qu'un modèle entraîné reçoit des données ayant le même format et les mêmes caractéristiques au moment de l'inférence.

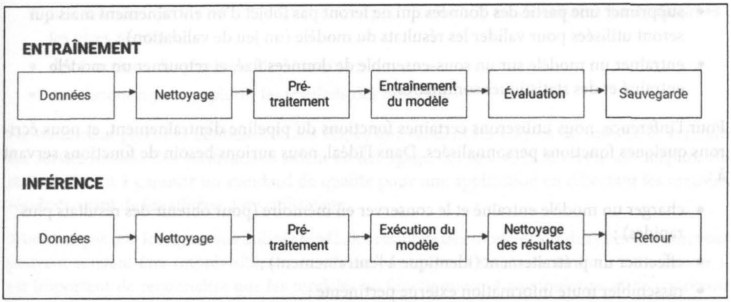


Figure 2.4 : Les pipelines d'entraînement et d'inférence sont complémentaires.

Des pipelines de différents modèles seront construits en tenant compte de préoccupations différentes, mais en général, l'infrastructure de haut niveau reste relativement stable. C'est pourquoi il est utile de commencer par construire de bout en bout votre pipeline d'entraînement et d'inférence pour évaluer rapidement le goulet d'étranglement de l'impact que Monica Rogati a évoqué dans son entretien au Chapitre 1.

La plupart des pipelines ont une structure de haut niveau similaire, mais en raison des différences dans la structure des jeux de données, les fonctions elles-mêmes n'ont souvent rien en commun. Illustrons cela en examinant le pipeline pour l'éditeur.

Pipeline pour l'Éditeur ML

Pour l'éditeur, nous allons construire des pipelines d'entraînement et d'inférence en utilisant Python, qui est un langage courant de choix en ML. L'objectif de ce premier prototype est de construire un pipeline de bout en bout sans trop se soucier de sa perfection.

Comme cela devrait être fait dans tout travail qui prend du temps, nous pouvons, et nous allons, en revoir certaines parties pour les améliorer. Pour l'entraînement, nous allons écrire un pipeline assez standard, largement applicable à de nombreux problèmes de ML et qui n'a que quelques fonctions, principalement celles servant à :

- charger des enregistrements de données ;
- nettoyer les données en supprimant les enregistrements incomplets et en saisissant les valeurs manquantes si nécessaire ;
- prétraiter et formater les données de manière à ce qu'elles puissent être comprises par un modèle ;

- supprimer une partie des données qui ne feront pas l'objet d'un entraînement mais qui seront utilisées pour valider les résultats du modèle (un jeu de validation) ;
- entraîner un modèle sur un sous-ensemble de données fixé, et retourner un modèle entraîné et des statistiques sommaires.

Pour l'inférence, nous utiliserons certaines fonctions du pipeline d'entraînement, et nous écrirons quelques fonctions personnalisées. Dans l'idéal, nous aurions besoin de fonctions servant à :

- charger un modèle entraîné et le conserver en mémoire (pour obtenir des résultats plus rapides) ;
- effectuer un prétraitement (identique à l'entraînement) ;
- rassembler toute information externe pertinente ;
- faire passer un échantillon au travers d'un modèle (une fonction d'inférence) ;
- effectuer un post-traitement, pour nettoyer les résultats avant de les fournir aux utilisateurs.

Il est souvent plus facile de visualiser un pipeline sous la forme d'un organigramme, comme celui représenté sur la Figure 2.5.

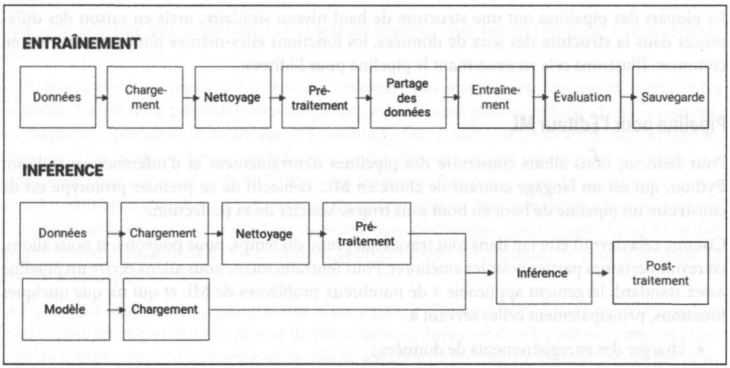


Figure 2.5 : Pipelines pour l'éditeur.

En outre, nous allons écrire diverses fonctions d'analyse et d'exploration pour nous aider à diagnostiquer les problèmes, comme par exemple :

- une fonction qui visualise les échantillons pour lesquels le modèle donne les meilleurs et les pires résultats ;
- des fonctions d'exploration des données ;
- une fonction pour explorer les résultats des modèles.

De nombreux pipelines contiennent des étapes qui valident les entrées du modèle et vérifient ses sorties finales. Ces vérifications aident au débogage, comme vous le verrez au Chapitre 10, et contribuent à garantir un standard de qualité pour une application en détectant les mauvais résultats avant de les afficher à un utilisateur.

N'oubliez pas que lorsque vous utilisez le ML, les résultats des modèles sur des données non vues peuvent souvent être imprévisibles et ne seront pas toujours satisfaisants. Pour cette raison, il est important de reconnaître que les modèles ne fonctionneront pas toujours, et de concevoir l'architecture des systèmes en fonction de ce potentiel d'erreurs.

Conclusion

Nous avons maintenant vu comment définir des métriques de base qui nous permettent de comparer des modèles entièrement différents et de comprendre les compromis entre chacun d'eux. Nous avons abordé les ressources et les méthodes à utiliser pour accélérer le processus de construction de vos tout premiers pipelines. Nous avons ensuite donné un aperçu de ce que nous devrons construire pour chaque pipeline afin d'obtenir une première série de résultats.

Nous avons maintenant une idée conçue en tant que problème de ML, une manière de mesurer les progrès et un plan initial. Il est temps de se plonger dans l'implémentation.

Dans la deuxième partie, nous nous pencherons sur la manière de construire un premier pipeline, et d'explorer et de visualiser un jeu de données initial.

PARTIE II

Construire un pipeline fonctionnel

La recherche, l'entraînement et l'évaluation des modèles étant un processus long, aller dans la mauvaise direction peut s'avérer très coûteux en matière de ML. C'est pourquoi ce livre se concentre sur la réduction des risques et sur l'identification de la plus haute priorité sur laquelle travailler.

Alors que la première partie était axée sur la planification afin de maximiser notre vitesse et nos chances de succès, ce chapitre se penchera sur les questions d'implémentation. Comme l'illustre la Figure II.1, en ML comme dans la plupart des domaines du génie logiciel, il faut parvenir à un produit minimum viable (MVP) le plus rapidement possible. C'est précisément ce que nous allons aborder ici : la manière la plus rapide de mettre en place un pipeline et de l'évaluer.

L'amélioration de ce dit modèle fera l'objet de la troisième partie de ce livre.

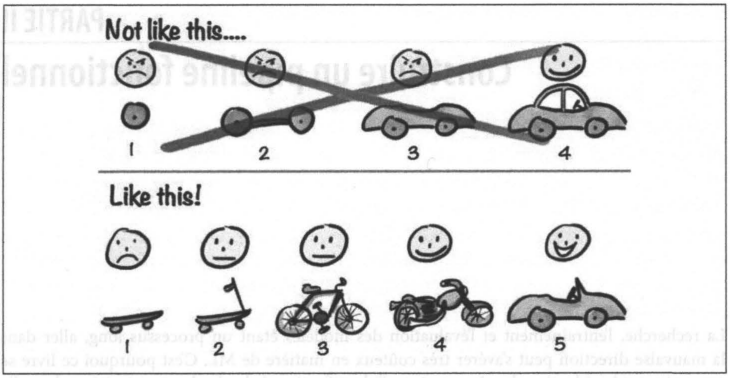


Figure II.1 : La bonne façon de construire votre premier pipeline (reproduit avec l'autorisation d'Henrik Kniberg).

Nous allons construire notre modèle initial en deux étapes :

Chapitre 3

Dans ce chapitre, nous allons construire la structure et l'échafaudage de notre application. Cela impliquera la construction d'un pipeline pour recueillir les entrées des utilisateurs et les suggestions faites en retour, ainsi qu'un pipeline distinct pour entraîner nos modèles avant de les utiliser.

Chapitre 4

Dans ce chapitre, nous nous concentrerons sur la collecte et l'inspection d'un jeu de données initial. L'objectif est ici d'identifier rapidement des motifs dans nos données et de prévoir lesquels d'entre eux seront prédictifs et utiles pour notre modèle.

CHAPITRE 3

Construire votre premier pipeline de bout en bout

Dans la première partie, nous avons commencé par aborder la manière de passer des exigences relatives aux produits aux approches de modélisation candidates. Puis, nous sommes passés à la phase de planification et nous avons décrit comment trouver des ressources pertinentes et les exploiter pour établir un plan initial de ce qu'il faut construire. Enfin, nous avons discuté de la manière dont la construction d'un premier prototype de système fonctionnel était la meilleure façon de progresser. C'est ce que nous allons traiter dans ce chapitre.

Cette première itération sera peu brillante de par sa conception. Son but est simplement de nous permettre de disposer de tous les éléments d'un pipeline afin de pouvoir établir des priorités concernant les prochaines améliorations. Avoir un prototype complet est le moyen le plus simple d'identifier le goulet d'étranglement que Monica Rogati a décrit dans le Chapitre 1.

Commençons par construire le pipeline le plus simple qui puisse produire des prédictions à partir d'une entrée.

L'échafaudage le plus simple

Dans la section « Commencer par un pipeline simple » du Chapitre 2, nous avons décrit comment la plupart des modèles de ML consistent en deux pipelines, l'un pour l'entraînement et l'autre pour l'inférence. L'entraînement nous permet de générer un modèle de haute qualité, et l'inférence consiste à servir les résultats aux utilisateurs. Pour en savoir plus sur la différence entre entraînement et inférence, reportez-vous à la section du Chapitre 2 indiquée au début du paragraphe.

Pour le premier prototype d'une application, nous nous concentrerons sur le fait d'être capable de fournir des résultats aux utilisateurs. Cela signifie que, sur les deux pipelines que nous avons décrits au Chapitre 2, nous commencerons par le pipeline d'inférence. Cela nous permettra d'examiner rapidement comment les utilisateurs peuvent interagir avec les résultats d'un modèle, et donc de recueillir des informations utiles pour faciliter l'entraînement d'un modèle.

Si nous nous concentrons uniquement sur l'inférence, nous ignorerons pour l'instant ce qui concerne l'entraînement. Et comme nous n'entraînons pas un modèle, nous pouvons à la place écrire quelques règles simples. L'écriture de telles règles ou heuristiques est souvent un excellent moyen de débuter. C'est la façon la plus rapide de construire un prototype, et cela nous permet de voir immédiatement une version simplifiée de l'application complète.

Bien que cela puisse sembler superflu si nous voulons de toute façon mettre en œuvre une solution de ML (comme nous le ferons plus tard dans le livre), il s'agit d'une fonction critique pour nous forcer à affronter notre problème et à concevoir un premier ensemble d'hypothèses quant à la meilleure façon de le résoudre.

L'élaboration, la validation et la mise à jour des hypothèses sur la meilleure façon de modéliser les données sont des éléments essentiels du processus itératif de construction de modèles, qui commence avant même que nous ayons construit notre premier modèle !

Voici quelques exemples d'excellentes heuristiques tirées de projets que j'ai vus utilisés par des stagiaires que j'ai encadrés chez Insight Data Science.

- Estimation de la qualité du code : lors de la construction d'un modèle visant à prédire si un codeur a bien fonctionné sur HackerRank (un site Web de codage compétitif) à partir d'un échantillon de code, Daniel a commencé par compter le nombre de parenthèses ouvertes et fermées, de crochets et d'accolades.

  Dans la plupart des codes qui fonctionnent correctement, le nombre de parenthèses ouvrantes et fermées se correspond, et cette règle s'est donc avérée être une base de référence assez solide. En outre, elle lui a donné l'intuition de concentrer sa modélisation sur l'utilisation d'un arbre de syntaxe abstraite[7] pour capturer encore plus d'informations structurelles sur le code.

- Comptage des arbres : lorsqu'il a essayé de compter les arbres d'une ville à partir d'images satellite, après avoir examiné certaines données, Mike a commencé par concevoir une règle d'estimation de la densité des arbres basée sur le dénombrement de la proportion de pixels verts dans une image donnée.

  Il s'avère que cette approche a fonctionné pour les arbres qui étaient dispersés, mais a échoué lorsqu'il s'agissait de bosquets d'arbres. Là encore, cela a permis de définir les étapes de modélisation suivantes, qui se sont concentrées sur la construction d'un pipeline capable de traiter des arbres densément groupés.

7 Voir l'adresse `https://fr.wikipedia.org/wiki/Arbre_de_la_syntaxe_abstraite`.

La grande majorité des projets de ML devraient commencer par une heuristique similaire. La clé est de se souvenir qu'il faut la concevoir en se basant sur les connaissances d'experts ainsi que l'exploration de données, et de l'utiliser pour confirmer les hypothèses initiales et accélérer l'itération.

Une fois que vous avez une heuristique, il est temps de créer un pipeline qui peut recueillir les données, les prétraiter, leur appliquer vos règles et servir des résultats. Cela peut être aussi simple qu'un script Python que vous pouvez appeler depuis une fenêtre de terminal, ou encore une application Web qui rassemble les images de la caméra d'un utilisateur pour ensuite délivrer les résultats en direct.

Il s'agit ici de faire pour votre produit la même chose que pour votre approche ML, c'est-à-dire de le simplifier autant que possible et de le construire de manière à ce que vous disposiez d'une version fonctionnelle simple. C'est ce qu'on appelle souvent un MVP (*minimum viable product*, soit produit minimum viable) et c'est une méthode éprouvée pour obtenir des résultats utiles le plus rapidement possible.

Prototype pour un Éditeur ML

Pour notre Éditeur ML, nous nous appuierons sur des recommandations d'édition courantes afin d'élaborer quelques règles sur ce qui fait de bonnes ou de mauvaises questions et afficher les résultats de ces règles aux utilisateurs.

Pour une version minimale de notre projet qui prend une entrée utilisateur depuis la ligne de commande et renvoie des suggestions, nous n'avons besoin d'écrire que quatre fonctions, montrées ici :

```
input_text = parse_arguments()
processed = clean_input(input_text)
tokenized_sentences = preprocess_input(processed)
suggestions = get_suggestions(tokenized_sentences)
```

Plongeons dans chacune d'elles ! Nous allons garder l'analyseur d'arguments simple et nous commencerons par récupérer une chaîne de texte provenant de l'utilisateur, sans aucune option. Vous pouvez trouver le code source de l'exemple, comme de tous les autres exemples de code, dans le dépôt GitHub de ce livre.

Analyser et nettoyer les données

Tout d'abord, nous analysons simplement les données entrantes provenant de la ligne de commande. Cette fonction est relativement simple à écrire en Python.

```
def parse_arguments():
    """

    :retourne : Le texte à éditer
    """

    parser = argparse.ArgumentParser(
    description="Receive text to be edited"
    )
    parser.add_argument(
        'text',
        metavar='input text',
        type=str
    )
    args = parser.parse_args()
    return args.text
```

Chaque fois qu'un modèle s'exécute sur une entrée utilisateur, vous devez commencer par le valider et le vérifier ! Dans notre cas, les utilisateurs taperont des données, nous nous assurerons donc que leur saisie contient des caractères que nous pourrons analyser. Pour nettoyer nos entrées, nous supprimerons les caractères non ASCII. Cela ne devrait pas trop restreindre la créativité de nos utilisateurs, et nous permettre de faire des suppositions raisonnables sur le contenu du texte.

```
def clean_input(text):
    """

    :paramètre texte : Entrée utilisateur
    :retourne : Texte traité, sans caractères non ascii
    """

    # Pour simplifier les choses au départ, on ne garde que les caractères ASCII
    return str(text.encode().decode('ascii', errors='ignore'))
```

Nous devons maintenant prétraiter notre entrée et formuler des recommandations. Pour commencer, nous nous appuierons sur certaines des recherches existantes sur la classification des textes que nous avons mentionnées dans la section « L'approche la plus simple» : être l'algorithme » du Chapitre 1. Il s'agira de compter des mots (anglais) tels que « told » et « said » (*dit*) et de calculer des statistiques résumées de syllabes, de mots et de phrases pour estimer la complexité des phrases.

Pour calculer les statistiques au niveau des mots, nous devons être capables d'identifier les mots à partir des phrases. Dans le monde du traitement du langage naturel, on appelle cela la *tokenisation*.

Tokeniser le texte

La tokenisation n'est pas simple, et la plupart des méthodes « naïves » auxquelles vous pouvez penser, comme la division de notre entrée en mots basés sur des espaces ou des points, échoueront sur un texte réaliste en raison de la diversité des façons de séparer les mots. Prenez cette phrase, fournie à titre d'exemple pour un cours sur le TLN de l'université de Stanford[8] :

> *"Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing."*

(soit : « M. O'Neill pense que les histoires des garçons sur la capitale du Chili ne sont pas amusantes. »)

La plupart des méthodes simples échoueront sur cette phrase en raison de la présence de points et d'apostrophes qui véhiculent des significations diverses. Au lieu de construire notre propre tokeniseur, nous utiliserons nltk (`https://www.nltk.org/`), une bibliothèque open source populaire, qui nous permet de le faire facilement en deux étapes, comme suit :

```
def preprocess_input(text):
    """

    :paramètre texte : Texte traité
    :retourne : Texte prêt à être analysé,
                avec les phrases et les mots tokenisés
    """

    sentences = nltk.sent_tokenize(text)
    tokens = [nltk.word_tokenize(sentence) for sentence in sentences]
    return tokens
```

Une fois que notre résultat est prétraité, nous pouvons l'utiliser pour générer des caractéristiques qui nous aideront à juger de la qualité d'une question.

Générer des caractéristiques

La dernière étape consiste à rédiger quelques règles que nous pourrions utiliser pour donner des conseils à nos utilisateurs. Pour ce prototype simple, nous commencerons par calculer la fréquence de quelques verbes et connecteurs courants, puis nous compterons l'utilisation des adverbes et nous déterminerons le score de lisibilité de Flesch[9]. Nous renverrons ensuite un rapport de ces mesures à nos utilisateurs :

```
def get_suggestions(sentence_list):
    """
    Retourne une chaîne contenant nos suggestions
    :paramètre sentence_list : une liste de phrases,
                               chacune étant une liste de mots
    :retourne : des suggestions pour améliorer l'entrée
```

8 Voir l'adresse `https://stanford.io/2ywXyfy`.

9 Voir l'adresse `https://bit.ly/2RMRiqH` en anglais, ou `https://bit.ly/34NEHJ6` en français.

```
    """

    told_said_usage = sum(
        (count_word_usage(tokens, ["told", "said"]) for tokens in sentence_list)
    )
    but_and_usage = sum(
        (count_word_usage(tokens, ["but", "and"]) for tokens in sentence_list)
    )
    wh_adverbs_usage = sum(
        (
            count_word_usage(
                tokens,
                [
                    "when",
                    "where",
                    "why",
                    "whence",
                    "whereby",
                    "wherein",
                    "whereupon",
                ],
            )
            for tokens in sentence_list
        )
    )
    result_str = ""
    adverb_usage = "Adverb usage: %s told/said, %s but/and, %s wh adverbs" % (
        told_said_usage,
        but_and_usage,
        wh_adverbs_usage,
    )
    result_str += adverb_usage
    average_word_length = compute_total_average_word_length(sentence_list)
    unique_words_fraction = compute_total_unique_words_fraction(sentence_list)

    word_stats = "Average word length %.2f, fraction of unique words %.2f" % (
        average_word_length,
        unique_words_fraction,
    )
    # Utilisation du break HTML pour affichage ultérieur dans une webapp
    result_str += "<br/>"
    result_str += word_stats

    number_of_syllables = count_total_syllables(sentence_list)
    number_of_words = count_total_words(sentence_list)
    number_of_sentences = len(sentence_list)

    syllable_counts = "%d syllables, %d words, %d sentences" % (
        number_of_syllables,
        number_of_words,
        number_of_sentences,
    )
    result_str += "<br/>"
```

```
    result_str += syllable_counts

    flesch_score = compute_flesch_reading_ease(
        number_of_syllables, number_of_words, number_of_sentences
    )

    flesch = "%d syllables, %.2f flesch score: %s" % (
        number_of_syllables,
        flesch_score,
        get_reading_level_from_flesch(flesch_score),
    )

    result_str += "<br/>"
    result_str += flesch

    return result_str
```

Voilà, nous pouvons maintenant appeler notre application depuis la ligne de commande et voir ses résultats en direct. Ce n'est pas encore très utile, mais nous avons un point de départ qu'il est possible de tester et à partir duquel nous pouvons itérer, ce que nous ferons ensuite.

Tester votre flux de travail

Maintenant que nous avons construit ce prototype, nous pouvons tester nos hypothèses sur la façon dont nous avons formulé notre problème et sur l'utilité de la solution proposée. Dans cette section, nous examinerons à la fois la qualité objective de nos règles initiales et l'utilité de nos résultats.

Comme Monica Rogati l'a expliqué dans le Chapitre 1 : « Fréquemment, votre produit est mort même en cas de réussite de votre modèle ». Si la méthode que nous avons choisie excelle à mesurer la qualité des questions mais que notre produit ne fournit aucun conseil aux utilisateurs pour améliorer leur rédaction, notre produit ne sera pas utile malgré la qualité de notre méthode. En examinant notre pipeline complet, évaluons à la fois l'utilité de l'expérience utilisateur actuelle et les résultats de notre modèle artisanal.

Expérience utilisateur

Examinons d'abord le niveau de satisfaction quant à l'utilisation de notre produit, indépendamment de la qualité de notre modèle. En d'autres termes, si nous imaginons que nous finirons par obtenir un modèle suffisamment performant, est-ce la façon la plus utile de présenter les résultats à nos utilisateurs ?

Si nous réalisons un recensement des arbres, par exemple, nous pouvons présenter nos résultats comme un résumé d'une longue analyse portant sur une ville entière. Nous pourrions inclure

le nombre d'arbres recensés, ainsi que des statistiques ventilées par quartier, et une mesure de l'erreur sur un jeu de test faisant office d'étalon-or.

En d'autres termes, nous voudrions nous assurer que les résultats que nous présentons sont utiles (ou qu'ils le seront si nous améliorons notre modèle). D'un autre côté, bien sûr, nous aimerions également que notre modèle soit performant. C'est le prochain aspect que nous évaluerons.

Modélisation des résultats

Nous avons mentionné l'intérêt de se concentrer sur la bonne métrique dans la section « Mesurer le succès » du Chapitre 2. Le fait de disposer d'un prototype fonctionnel dès le début nous permettra d'identifier et d'itérer les métriques que nous avons choisies pour nous assurer qu'elles représentent le succès du produit.

Par exemple, si nous construisions un système pour aider les utilisateurs à rechercher des voitures de location à proximité, nous pourrions utiliser une mesure telle que le gain cumulatif actualisé (DCG). Le DCG mesure la qualité du classement en produisant un score qui est le plus élevé lorsque les éléments les plus pertinents sont retournés plus tôt que les autres[10]. Lors de la construction initiale de notre outil, nous avons peut-être supposé que nous voulions qu'au moins une suggestion utile apparaisse dans nos cinq premiers résultats. Nous avons donc utilisé le DCG à 5 pour noter notre modèle. Cependant, lorsque les utilisateurs essaient l'outil, nous pouvons remarquer qu'ils ne considèrent jamais que les trois premiers résultats affichés. Dans ce cas, nous devrions changer notre métrique de succès pour le DCG de 5 à 3.

Le but de considérer à la fois l'expérience utilisateur et la performance du modèle est de s'assurer que nous travaillons sur l'aspect le plus impactant. Si votre expérience utilisateur est mauvaise, améliorer votre modèle n'aidera en rien. En fait, vous pouvez même réaliser que vous seriez mieux servi avec un modèle entièrement différent ! Examinons deux exemples.

Trouver le goulet d'étranglement de l'impact

L'objectif de l'examen à la fois des résultats de la modélisation et de la présentation courante du produit est d'identifier le prochain défi à relever. La plupart du temps, il s'agira d'itérer sur la manière dont nous présentons les résultats à nos utilisateurs (ce qui pourrait signifier changer la façon dont nous entraînons nos modèles), ou d'améliorer les performances du modèle en identifiant les principaux points d'échec.

Nous nous plongerons davantage dans l'analyse des erreurs dans la troisième partie du livre, mais nous devrions identifier les modes de défaillance et les moyens appropriés pour les résoudre. Il est important de déterminer si la tâche la plus impactante à accomplir se situe dans le domaine de la modélisation ou dans celui du produit, car cela nécessite des mesures correctives différentes. Voyons un exemple de chaque cas :

10 Voir par exemple l'adresse `https://en.wikipedia.org/wiki/Discounted_cumulative_gain`.

Du côté du produit

Supposons que vous avez construit un modèle qui examine des images dans des articles de recherche et qui prédit s'ils seront acceptés dans des conférences de haut niveau (voir le document de Jia-Bin Huang « *Deep Paper Gestalt* », qui aborde cette question[11]). Cependant, vous avez remarqué que le fait de ne renvoyer à un utilisateur qu'une probabilité de rejet n'est pas le résultat le plus satisfaisant. Dans ce cas, il ne serait *pas* utile d'améliorer votre modèle. Il serait logique de se concentrer sur l'extraction de conseils à partir du modèle afin de pouvoir aider nos utilisateurs à améliorer leurs documents et à augmenter leurs chances d'être acceptés.

Du côté du modèle

Vous avez construit un modèle de notation de crédit et vous remarquez que, tous les autres facteurs étant égaux, il attribue des risques plus élevés de défaillance à un certain groupe ethnique. Cela est probablement dû à un biais dans les données d'entraînement que vous avez utilisées. Vous devriez donc recueillir des données plus représentatives et construire un nouveau pipeline de nettoyage et d'augmentation de vos sources pour tenter de remédier à ce problème. Dans ce cas, quelle que soit la manière dont vous présentez les résultats, *le modèle doit être corrigé*. Des exemples comme celui-ci sont courants et c'est pourquoi vous devriez toujours aller bien au-delà d'une métrique agrégée et examiner l'impact de votre modèle sur différentes tranches de vos données. C'est ce que nous allons faire au Chapitre 5.

Pour illustrer davantage ce point, voyons comment cela se présente pour notre Éditeur ML.

Évaluation du prototype de l'Éditeur ML

Voyons comment notre pipeline initial se comporte à la fois en termes d'expérience utilisateur et de performance du modèle. Commençons par ajouter quelques entrées dans notre application. Nous testerons d'abord une question simple, puis une question plus compliquée et enfin un paragraphe complet. Tous ces exemples sont bien entendu en anglais.

Comme nous utilisons un score de facilité de lecture, nous aimerions idéalement que notre flux de travail donne une note élevée pour la phrase simple, une note faible pour la phrase complexe, et des suggestions pour améliorer notre paragraphe. Examinons quelques exemples à l'aide de notre prototype.

Question simple :

```
$ python ml_editor.py "Is this workflow any good?"
Adverb usage: 0 told/said, 0 but/and, 0 wh adverbs
Average word length 3.67, fraction of unique words 1.00
```

11 Voir l'adresse `https://arxiv.org/abs/1812.08775`.

```
6 syllables, 5 words, 1 sentences
6 syllables, 100.26 flesch score: Very easy to read
```

Question complexe :

```
$ python ml_editor.py "Here is a needlessly obscure question, that"\
"does not provide clearly which information it would"\
"like to acquire, does it?"

Adverb usage: 0 told/said, 0 but/and, 0 wh adverbs
Average word length 4.86, fraction of unique words 0.90
30 syllables, 18 words, 1 sentences
30 syllables, 47.58 flesch score: Difficult to read
```

Paragraphe entier :

```
$ python ml_editor.py "Ideally, we would like our workflow to return a positive"\ " score
  for the simple sentence, a negative score for the convoluted one, and "\ "suggestions for
  improving our paragraph. Is that the case already?"
Adverb usage: 0 told/said, 1 but/and, 0 wh adverbs
Average word length 4.03, fraction of unique words 0.76
52 syllables, 33 words, 2 sentences
52 syllables, 56.79 flesch score: Fairly difficult to read
```

Examinons ces résultats en utilisant les deux aspects que nous venons de définir.

Modèle

Il n'est pas certain que nos résultats correspondent bien à ce que nous considérons comme une écriture de qualité. La phrase complexe et le paragraphe entier reçoivent un score de lisibilité similaire. Je serais le premier à admettre que ma prose peut parfois être difficile à lire, mais le paragraphe précédent est plus compréhensible que la phrase alambiquée que nous avons testée avant lui.

Les attributs que nous extrayons du texte ne sont pas nécessairement les plus corrélés à une notion de « bonne écriture ». Cela est généralement dû au fait que le succès n'est pas assez clairement défini : face à deux questions, comment pouvons-nous dire que l'une est meilleure que l'autre ? Lorsque nous construirons notre jeu de données dans le prochain chapitre, nous définirons ce point plus clairement.

Comme prévu, nous avons un certain travail de modélisation à faire, mais est-ce que nous présentons les résultats de manière utile ?

Expérience utilisateur

D'après les résultats présentés précédemment, deux problèmes apparaissent immédiatement. Les informations que nous renvoyons ont un côté écrasant et sont en même temps non pertinentes. L'objectif de notre produit est de fournir des recommandations concrètes à nos utilisa-

teurs. Les caractéristiques et le score de lisibilité sont une métrique de qualité, mais cela n'aidera pas l'utilisateur à décider comment améliorer la formulation de sa question. Nous pourrions vouloir réduire nos recommandations à un seul score, avec des recommandations concrètes pour l'améliorer.

Par exemple, nous pourrions suggérer des modifications générales telles que l'utilisation de moins d'adverbes, ou travailler à un niveau plus granulaire en proposant des changements au niveau des mots et des phrases. Dans l'idéal, nous pourrions présenter les résultats en mettant en évidence ou en soulignant les parties de l'entrée qui requièrent l'attention des utilisateurs. J'ai ajouté une maquette de ce que cela pourrait donner sur la Figure 3.1.

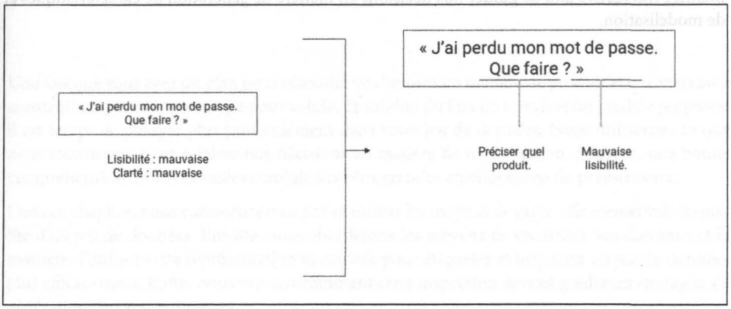


Figure 3.1 : Quelques suggestions de rédaction plus exploitables.

Même si nous n'étions pas en mesure de mettre directement en évidence les recommandations dans la chaîne en entrée, notre produit pourrait bénéficier de recommandations similaires à celles qui figurent sur le côté droit de la Figure 3.1, et qui sont plus faciles à mettre en œuvre qu'une liste de scores.

Conclusion

Nous avons construit un premier prototype d'inférence et nous l'avons utilisé pour évaluer la qualité de nos heuristiques et le flux de travail de notre produit. Cela nous a permis de limiter nos critères de performance et d'itérer sur la manière dont nous aimerions présenter les résultats à nos utilisateurs.

Pour notre Éditeur ML, nous avons appris que nous devrions à la fois nous concentrer sur l'amélioration de l'expérience utilisateur en fournissant des recommandations concrètes, et améliorer

notre approche de modélisation en examinant les données pour définir plus clairement ce qui fait qu'une question est bonne.

Dans les trois premiers chapitres, nous avons utilisé les objectifs de notre produit pour définir l'approche initiale à adopter, nous avons exploré des ressources existantes pour établir un plan pour notre approche, et nous avons construit un prototype initial pour valider notre plan et nos hypothèses.

Il est maintenant temps de se plonger dans ce qui est souvent la partie la plus négligée d'un projet ML : l'exploration de notre jeu de données. Dans le Chapitre 4, nous verrons comment rassembler un premier jeu de données, évaluer sa qualité et en étiqueter des sous-ensembles de manière interactive afin de guider nos décisions en matière de génération de caractéristiques et de modélisation.

CHAPITRE 4

Acquérir un jeu de données initial

Une fois que vous avez un plan pour résoudre vos besoins en matière de produit et que vous avez construit un premier prototype pour valider la solidité du flux de travail et du modèle proposés, il est temps de plonger plus profondément dans votre jeu de données. Nous utiliserons ce que nous trouverons pour éclairer nos décisions en matière de modélisation. Souvent, une bonne compréhension de vos données conduit aux plus grandes améliorations de performance.

Dans ce chapitre, nous commencerons par examiner les moyens de juger efficacement de la qualité d'un jeu de données. Ensuite, nous aborderons les moyens de vectoriser vos données, et la manière d'utiliser cette représentation vectorisée pour étiqueter et inspecter un jeu de données plus efficacement. Enfin, nous verrons comment cette inspection devrait guider les stratégies de génération de caractéristiques.

Commençons par découvrir un jeu de données et par juger de sa qualité.

Itérer sur des jeux de données

La façon la plus rapide de construire un produit ML est de construire, évaluer et itérer rapidement sur des modèles. Les jeux de données eux-mêmes sont une partie essentielle du succès des modèles. C'est pourquoi la collecte, la préparation et l'étiquetage des données doivent être considérés comme un *processus itératif*, tout comme la modélisation. Commencez par un jeu de données simple que vous pouvez collecter immédiatement, et soyez ouvert aux améliorations en fonction de ce que vous apprenez.

Cette approche itérative des données peut sembler déroutante au premier abord. Dans la recherche en ML, les performances sont souvent rapportées sur des jeux de données standard que la communauté utilise en tant que référence, et qui sont donc immuables. Dans le génie logiciel traditionnel, nous écrivons des règles déterministes pour nos programmes, de sorte que nous traitons les données comme quelque chose à recevoir, traiter et enregistrer.

L'ingénierie ML combine les deux termes (ingénierie et ML) afin de construire des produits. Notre jeu de données n'est donc qu'un outil de plus pour nous permettre de réaliser nos objectifs. En ingénierie ML, le choix d'un jeu de données initial, sa mise à jour régulière et son enrichissement constituent souvent la *majeure partie du travail*. Cette différence de flux de travail entre recherche et industrie est illustrée sur la Figure 4.1.

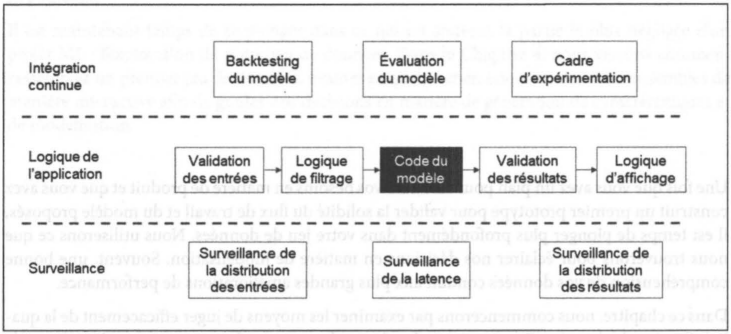


Figure 4.1 : Les jeux de données sont fixes dans la recherche, mais font partie du produit dans l'industrie.

Traiter les données comme étant une partie de votre produit sur laquelle vous pouvez (et devriez) itérer, changer et améliorer est souvent un grand changement de paradigme pour les nouveaux venus dans l'industrie. Cependant, une fois que vous vous y serez habitué, les données deviendront votre meilleure source d'inspiration pour développer de nouveaux modèles et le premier endroit où vous chercherez des réponses lorsque les choses tourneront mal.

Science des données

J'ai vu de multiples fois le processus de conservation d'un jeu de données être le principal obstacle à la fabrication de produits ML. Cela s'explique en partie par le manque relatif d'éducation sur le sujet (la plupart des cours en ligne fournissent un jeu de données prédéfini et se concentrent sur les modèles), ce qui fait que de nombreux praticiens craignent cette partie du travail.

Il est facile de penser que le travail sur les données est une corvée à laquelle il faut s'attaquer avant de jouer avec des modèles amusants, mais les modèles ne servent qu'à extraire des tendances et des motifs à partir des données existantes. S'assurer que les données que nous utilisons présentent des motifs suffisamment prédictifs pour qu'un modèle puisse en tirer parti (et vérifier

s'il contient des biais évidents) est donc une partie fondamentale du travail d'un spécialiste des données (en fait, vous avez peut-être remarqué que l'on ne parle pas de spécialiste des modèles).

Ce chapitre se concentre sur ce processus, depuis la collecte d'un jeu de données initial jusqu'à l'inspection et la validation de son applicabilité au ML. Commençons par explorer un jeu de données de manière efficace pour juger de sa qualité.

Explorer votre premier jeu de données

Comment explorer un jeu de données initial ? La première étape consiste bien sûr à collecter un jeu de données. C'est là que je vois le plus souvent les praticiens se retrouver bloqués dans leur recherche d'un jeu de données parfait. N'oubliez pas que notre objectif est d'obtenir un jeu de données simple pour en extraire des résultats préliminaires. Comme pour d'autres choses en ML, il faut commencer simplement, et construire à partir de là.

Soyez efficace, commencez petit

Pour la plupart des problèmes de ML, plus de données peuvent conduire à un meilleur modèle, mais cela ne signifie pas que vous devez commencer par le plus grand jeu de données possible. Au début d'un projet, un petit jeu de données vous permet d'inspecter et de comprendre facilement vos données et de voir comment mieux les modéliser. Vous devez donc viser un premier jeu de données facile à utiliser. Ce n'est qu'une fois que vous avez établi une stratégie qu'il est logique de passer à une plus grande échelle.

Si vous travaillez dans une entreprise où des téraoctets de données sont stockés dans un cluster, vous pouvez commencer par extraire un sous-ensemble échantillonné de manière uniforme qui pourra tenir dans la mémoire de votre machine locale. Si vous souhaitez commencer à travailler sur un projet secondaire visant à identifier les marques des voitures qui circulent devant votre maison, par exemple, commencez par quelques dizaines d'images de voitures prises dans la rue.

Une fois que vous aurez vu comment votre modèle initial fonctionne et où il se débat, vous serez capable d'itérer sur votre jeu de données en toute connaissance de cause !

Vous pouvez trouver de nombreux jeux de données existants en ligne sur des plateformes telles que Kaggle (`https://www.kaggle.com/`) ou Reddit (`https://www.reddit.com/r/datasets/`), ou en rassembler quelques exemples vous-même, soit en « grattant » le Web, soit en tirant parti de grands jeux de données open source, tels que ceux trouvés sur le site Common Crawl (`https://commoncrawl.org/`), soit en générant vos propres données ! Pour plus d'informations, reportez-vous à la section « Données libres de droits » du Chapitre 2.

La collecte et l'analyse de données sont non seulement nécessaires, mais elles vous permettront d'accélérer le processus, surtout au début du développement d'un projet. Examiner votre jeu de

données et en connaître les caractéristiques est le moyen le plus simple d'obtenir un bon pipeline de modélisation et de génération de caractéristiques.

La plupart des praticiens surestiment l'impact du travail sur le modèle et sous-estiment la valeur du travail sur les données. C'est pourquoi je recommande de toujours faire un effort pour corriger cette tendance et de s'orienter vers l'examen des données.

Lorsqu'on examine des données, il est bon d'identifier les tendances de manière exploratoire, mais il ne faut pas s'arrêter là. Si votre objectif est de créer des produits ML, vous devez vous demander quelle est la meilleure façon de tirer parti de ces tendances de manière automatisée. Comment ces tendances peuvent-elles vous aider à faire fonctionner un produit automatisé ?

Données : connaissances et produits

Une fois que vous disposez d'un jeu de données, il est temps de vous y plonger et d'en explorer le contenu. Ce faisant, gardons à l'esprit la distinction entre l'exploration des données à des fins d'analyse et l'exploration des données à des fins de création de produits. Alors que les deux visent à extraire et à comprendre les tendances dans les données, la première se préoccupe de créer des champs de connaissance à partir de ces tendances (en apprenant par exemple que la plupart des connexions frauduleuses à un site Web ont lieu le jeudi et proviennent de la région de Seattle), tandis que la seconde consiste à utiliser ces tendances pour construire des caractéristiques (en utilisant le moment d'une tentative de connexion et l'adresse IP correspondante pour construire un service qui empêche les connexions frauduleuses à des comptes).

Bien que la différence puisse sembler subtile, elle entraîne un niveau de complexité supplémentaire dans le cas de la construction d'un produit. Nous devons avoir confiance dans le fait que les motifs que nous détectons dans les données s'appliqueront à celles que nous recevrons à l'avenir, et quantifier les différences entre les données sur lesquelles nous nous entraînons et celles que nous nous attendons à recevoir en production.

Pour la prédiction des fraudes, la première étape consiste à remarquer un aspect saisonnier des connexions frauduleuses. Nous devons ensuite utiliser cette tendance saisonnière observée pour estimer la fréquence à laquelle nous devons entraîner nos modèles sur les données récemment recueillies. Nous nous plongerons dans d'autres exemples au fur et à mesure que nous explorerons nos données plus en profondeur, plus loin dans ce chapitre.

Avant de remarquer des tendances prédictives, nous devrions commencer par nous pencher sur les problèmes de qualité. Si le jeu de données choisi ne répond pas aux normes de qualité, nous devons l'améliorer avant de passer à la modélisation.

Une rubrique sur la qualité des données

Dans cette section, nous aborderons certains aspects à examiner lors de la première utilisation d'un nouveau jeu de données. Chaque jeu de données comporte ses propres biais et bizarreries,

qui nécessitent des outils différents pour être compris. La rédaction d'une rubrique complète couvrant tout ce que vous pouvez souhaiter rechercher dans un jeu de données dépasse évidemment le cadre de cet ouvrage. Cependant, il existe quelques catégories auxquelles il est utile de prêter attention lors d'une première approche d'un jeu de données. Commençons par le formatage.

Format des données

Le jeu de données est-il déjà formaté de manière à ce que les entrées et les sorties soient claires, ou nécessite-t-il un prétraitement et un étiquetage supplémentaires ?

Lors de la construction d'un modèle qui tente de prévoir si un utilisateur cliquera sur une annonce, par exemple, un jeu de données classique consistera en un journal mémorisant l'historique de tous les clics pour une période donnée. Vous devriez transformer ce jeu de données de manière à ce qu'il contienne de multiples instances d'une publicité présentée à un utilisateur ainsi que le fait que celui-ci a (ou non) cliqué. Vous voudrez également inclure toutes les caractéristiques de l'utilisateur ou de la publicité que vous pensez exploitables par votre modèle.

Si on vous donne un jeu de données qui a déjà été traité ou agrégé pour vous, vous devriez valider le fait que vous comprenez la manière dont les données ont été traitées. Si l'une des colonnes qui vous ont été fournies contient un taux de conversion moyen, par exemple, pouvez-vous calculer ce taux vous-même et vérifier qu'il correspond bien à la valeur indiquée ?

Parfois, vous n'aurez pas accès aux informations nécessaires pour reproduire et valider les étapes de prétraitement. Dans ces cas, l'examen de la qualité des données vous aidera à déterminer les caractéristiques auxquelles vous faites confiance, et celles qu'il serait préférable de ne pas prendre en compte.

Qualité des données

Il est crucial d'examiner la qualité d'un jeu de données avant de commencer à le modéliser. Si vous savez qu'il manque la moitié des valeurs d'une caractéristique essentielle, vous n'allez pas passer des heures à déboguer un modèle pour essayer de comprendre pourquoi il ne fonctionne pas bien.

Les données peuvent être de mauvaise qualité de plusieurs façons. Elles peuvent être manquantes, imprécises ou même corrompues. Obtenir une image exacte de leur qualité vous permettra non seulement d'estimer quel niveau de performance est raisonnable, mais aussi de sélectionner plus facilement les caractéristiques et les modèles potentiels à utiliser.

Si vous travaillez avec les journaux d'activité des utilisateurs pour prévoir l'utilisation d'un produit en ligne, pouvez-vous estimer le nombre d'événements enregistrés qui manquent ? Pour les événements dont vous disposez, combien ne contiennent qu'un sous-ensemble d'informations sur l'utilisateur ?

Si vous travaillez sur un texte en langage naturel, comment évaluez-vous la qualité de ce texte ? Par exemple, y a-t-il beaucoup de caractères incompréhensibles ? L'orthographe est-elle très erronée ou incohérente ?

Si vous travaillez sur des images, sont-elles suffisamment claires pour que vous puissiez effectuer la tâche vous-même ? S'il vous est difficile de détecter un objet dans une image, pensez-vous que votre modèle aura du mal à le faire ?

De manière générale, quelle proportion de vos données semble bruitée ou incorrecte ? Combien d'entrées sont difficiles à interpréter ou à comprendre ? Si les données ont des étiquettes, avez-vous tendance à les approuver ou trouvez-vous souvent à mettre en doute leur exactitude ?

J'ai travaillé sur quelques projets visant par exemple à extraire des informations d'une imagerie satellitaire. Dans le meilleur des cas, ces projets ont accès à un ensemble d'images avec des annotations correspondantes indiquant les objets intéressants tels que des champs ou des plans. Dans certains cas, cependant, ces annotations peuvent être inexactes ou même manquantes. De telles erreurs ont un impact significatif sur toute approche de modélisation, et il est donc vital de les découvrir rapidement. Nous pouvons travailler avec les étiquettes manquantes en étiquetant nous-mêmes un jeu de données initial, ou encore en trouvant une étiquette faible que nous sommes en mesure d'utiliser, mais nous ne pouvons le faire que si nous constatons les problèmes de qualité *à l'avance*.

Après avoir vérifié le format et la qualité des données, une étape supplémentaire peut aider à faire émerger les problèmes de manière proactive : l'examen de la quantité de données et de la distribution des caractéristiques.

Quantité de données et distribution des caractéristiques

Estimons si nous disposons de suffisamment de données et si les valeurs des caractéristiques semblent se situer dans une fourchette raisonnable.

Combien de données avons-nous ? Si nous disposons d'un vaste jeu de données, nous devons en sélectionner un sous-ensemble pour commencer notre analyse. En revanche, si notre jeu de données est trop petit, ou si certaines classes sont sous-représentées, les modèles que nous entraînons risquent d'être tout aussi biaisés que nos données. La meilleure façon d'éviter de tels biais est d'accroître la diversité de nos données grâce à des procédures de collecte et d'augmentation. La façon dont vous mesurez la qualité de vos données dépend de votre jeu de données, mais le Tableau 4.1 couvre quelques questions pour vous aider à démarrer.

Tableau 4.1 : Une rubrique sur la qualité des données.

Qualité	Format	Quantité et distribution
Certains champs pertinents sont-ils parfois vides ?	Combien d'étapes de prétraitement vos données nécessitent-elles ?	Combien d'échantillons avez-vous ?
Y a-t-il des erreurs de mesure potentielles ?	Pourrez-vous effectuer un prétraitement de la même façon en production ?	Combien d'échantillons par classe ? Y a-t-il des absents ?

Pour donner un exemple pratique, lors de l'élaboration d'un modèle permettant de classer automatiquement les courriers électroniques du support client en différents domaines d'expertise, un spécialiste des données avec lequel je travaillais, Alex Wahl, s'est vu attribuer neuf catégories distinctes, avec un seul échantillon par catégorie. Un tel jeu de données est trop petit pour qu'un modèle puisse en tirer des enseignements, c'est pourquoi il a concentré l'essentiel de ses efforts sur une stratégie de génération de données (`https://bit.ly/2XYad5E`). Il a utilisé des modèles de formulations courantes pour chacune des neuf catégories afin de produire des milliers d'échantillons supplémentaires sur lesquels un modèle pourrait ensuite s'appuyer. Grâce à cette stratégie, il a réussi à amener un pipeline à un niveau de précision bien plus élevé qu'il ne l'aurait fait en essayant de construire un modèle suffisamment complexe pour apprendre à partir de seulement neuf échantillons.

Appliquons ce processus d'exploration au jeu de données que nous avons choisi pour notre Éditeur ML et estimons sa qualité !

Inspection des données pour l'Éditeur ML

Pour notre Éditeur ML, nous avons d'abord décidé d'utiliser le jeu de données anonymisé Stack Exchange Data Dump (`https://archive.org/details/stackexchange`). Stack Exchange est un réseau de sites Web de questions-réponses, chacun axé sur un thème tel que la philosophie ou les jeux. L'archive contient de nombreux documents, un pour chacun des sites Web du réseau Stack Exchange.

Pour notre jeu de données initial, nous choisirons un site Web qui semble contenir des questions suffisamment larges pour permettre de construire des heuristiques utiles. À première vue, la communauté des écrivains semble convenir (`https://writing.stackexchange.com/`).

Chaque archive de site Web est fournie sous la forme d'un fichier XML. Nous devons construire un pipeline pour ingérer ces fichiers et les transformer en texte dont nous pourrons ensuite extraire les caractéristiques. L'exemple suivant montre à quoi ressemble le contenu du fichier Posts.xml pour le site `datascience.stackexchange.com` :

```
<?xml version="1.0" encoding="utf-8"?>
<posts>
<row Id="5" PostTypeId="1" CreationDate="2014-05-13T23:58:30.457"
Score="9" ViewCount="516" Body="&lt;p&gt; &quot;Hello World&quot; example? "
OwnerUserId="5" LastActivityDate="2014-05-14T00:36:31.077"
Title="How can I do simple machine learning without hard-coding behavior?"
Tags="&lt;machine-learning&gt;" AnswerCount="1" CommentCount="1" />
  <row Id="7" PostTypeId="1" AcceptedAnswerId="10" ... />
```

Pour pouvoir exploiter ces données, nous devrons pouvoir charger le fichier XML, décoder les balises HTML dans le texte et représenter les questions et les données associées dans un format qui serait plus facile à analyser, comme un DataFrame pandas. C'est précisément ce que fait la

fonction suivante. Pour rappel, le code de cette fonction, ainsi que tous les autres codes de ce livre, se trouvent dans le dépôt GitHub de ce livre[12].

```
import xml.etree.ElementTree as ElT

def parse_xml_to_csv(path, save_path=None):
    """
    Ouvre le dump des posts .xml et convertit le texte en csv, en le
    tokenisant lors du processus
    :paramètre path : chemin d'accès au document xml contenant les posts
    :retourne : un dataframe du texte traité
    """

    # Utilise la bibliothèque standard de python pour parcourir le fichier XML
    doc = ElT.parse(path)
    root = doc.getroot()

    # Chaque ligne est une question
    all_rows = [row.attrib for row in root.findall("row")]

    # Utilise tdqm pour afficher l'état d'avancement
    # car le prétraitement prend du temps
    for item in tqdm(all_rows):
        # Décode le texte depuis le HTML
        soup = BeautifulSoup(item["Body"], features="html.parser")
        item["body_text"] = soup.get_text()

    # Crée un dataframe depuis notre liste de dictionnaires
    df = pd.DataFrame.from_dict(all_rows)
    if save_path:
        df.to_csv(save_path)
    return df
```

Même pour un jeu de données relativement petit ne contenant à ce stade que 30 000 questions, ce processus prend plus d'une minute. Nous sérialisons donc le fichier traité sur disque pour n'avoir à le faire qu'une seule fois. Pour cela, nous pouvons simplement utiliser la fonction to_csv de pandas, comme le montre la dernière ligne de l'extrait de code.

Cette pratique est généralement recommandée pour tout prétraitement nécessaire à l'entraînement d'un modèle. Le code de prétraitement qui s'exécute juste avant le processus d'optimisation du modèle peut ralentir considérablement l'expérimentation. Dans la mesure du possible, prétraitez toujours les données à l'avance et sérialisez-les sur disque.

Une fois que nous disposons de nos données dans ce format, nous pouvons examiner les aspects que nous avons décrits précédemment. L'ensemble du processus d'exploration que nous détaillons ensuite se trouve dans le notebook d'exploration du jeu de données dans le dépôt GitHub de ce livre.

12 Rappelons l'adresse du dépôt : `https://github.com/hundredblocks/ml-powered-applications`.

Pour commencer, nous utilisons `df.info()` pour afficher des informations sommaires sur notre DataFrame, ainsi que toute valeur vide. Voici ce qu'elle renvoie :

```
>>>> df.info()

AcceptedAnswerId        4124 non-null float64
AnswerCount             33650 non-null int64
Body                    33650 non-null object
ClosedDate              969 non-null object
CommentCount            33650 non-null int64
CommunityOwnedDate      186 non-null object
CreationDate            33650 non-null object
FavoriteCount           3307 non-null float64
Id                      33650 non-null int64
LastActivityDate        33650 non-null object
LastEditDate            10521 non-null object
LastEditorDisplayName   606 non-null object
LastEditorUserId        9975 non-null float64
OwnerDisplayName        1971 non-null object
OwnerUserId             32117 non-null float64
ParentId                25679 non-null float64
PostTypeId              33650 non-null int64
Score                   33650 non-null int64
Tags                    7971 non-null object
Title                   7971 non-null object
ViewCount               7971 non-null float64
body_text               33650 non-null object
full_text               33650 non-null object
text_len                33650 non-null int64
is_question             33650 non-null bool
```

Nous pouvons constater que nous avons un peu plus de 33 000 posts, dont seulement environ 4 000 ont une réponse acceptée. De plus, nous pouvons également remarquer que certaines des valeurs de `Body`, qui représente le contenu d'un post, sont nulles, ce qui semble suspect. Nous nous attendons à ce que tous les messages contiennent du texte. En regardant les lignes avec un `Body` nul, on s'aperçoit rapidement qu'elles appartiennent à un type de post qui n'a pas de référence dans la documentation fournie avec le jeu de données, donc nous les supprimons.

Plongeons rapidement dans le format et voyons si nous le comprenons. Chaque post a une valeur PostTypeId de 1 pour une question, ou de 2 pour une réponse. Nous voudrions voir quels types de questions reçoivent des scores élevés, car nous aimerions utiliser le score d'une question comme un label faible pour notre véritable étiquette, la qualité d'une question.

Tout d'abord, associons les questions avec les réponses correspondantes. Le code suivant sélectionne toutes les questions qui ont une réponse acceptée et les joint au texte de ladite réponse. Nous pouvons ensuite examiner les premières lignes et valider le fait que les réponses correspondent bien aux questions. Cela nous permettra également de parcourir rapidement le texte et de juger de sa qualité.

```
questions_with_accepted_answers = df[
    df["is_question"] & ~(df["AcceptedAnswerId"].isna())
]
q_and_a = questions_with_accepted_answers.join(
    df[["Text"]], on="AcceptedAnswerId", how="left", rsuffix="_answer"
)

pd.options.display.max_colwidth = 500
q_and_a[["Text", "Text_answer"]][:5]
```

Dans le Tableau 4.2, on peut constater que les questions et les réponses (en anglais évidemment) semblent correspondre et que le texte paraît pour l'essentiel correct. Nous sommes maintenant convaincus que nous pouvons faire correspondre les questions et les réponses qui leur sont associées.

Tableau 4.2 : Questions et réponses correspondantes.

Id	body_text	body_text_answer
1	I've always wanted to start writing (in a totally amateur way), but whenever I want to start something I instantly get blocked having a lot of questions and doubts.\nAre there some resources on how to start becoming a writer?\nI'm thinking something with tips and easy exercises to get the ball rolling.\n	When I'm thinking about where I learned most how to write, I think that reading was the most important guide to me. This may sound silly, but by reading good written newspaper articles (facts, opinions, scientific articles, and most of all, criticisms of films and music), I learned how others did the job, what works and what doesn't. In my own writing, I try to mimic other people's styles that I liked. Moreover, I learn new things by reading, giving me a broader background that I need when re…
2	What kind of story is better suited for each point of view? Are there advantages or disadvantages inherent to them?\nFor example, writing in the first person you are always following a character, while in the third person you can "jump" between story lines.\n	With a story in first person, you are intending the reader to become much more attached to the main character. Since the reader sees what that character sees and feels what that character feels, the reader will have an emotional investment in that character. Third person does not have this close tie; a reader can become emotionally invested but it will not be as strong as it will be in first person.\nContrarily, you cannot have multiple point characters when you use first person without ex…
3	I finished my novel, and everyone I've talked to says I need an agent. How do I find one?\n	Try to find a list of agents who write in your genre, check out their websites!\nFind out if they are accepting new clients. If they aren't, then check out another agent. But if they are, try sending them a few chapters from your story, a brief, and a short cover letter asking them to represent you.\nIn the cover letter mention your previous publication credits. If sent via post, then I suggest you give them a means of reply, whether it be an email or a stamped, addressed envelope.\nAgents…

Comme dernier test sanitaire, voyons combien de questions n'ont pas reçu de réponse, combien en ont reçu au moins une et combien ont reçu une réponse qui a été acceptée.

```
has_accepted_answer = df[df["is_question"] & ~(df["AcceptedAnswerId"].isna())]
no_accepted_answers = df[
    df["is_question"]
    & (df["AcceptedAnswerId"].isna())
    & (df["AnswerCount"] != 0)
```

```
]
no_answers = df[
    df["is_question"]
    & (df["AcceptedAnswerId"].isna())
    & (df["AnswerCount"] == 0)
]

print(
    "%s questions with no answers, %s with answers, %s with an accepted answer"
    % (len(no_answers), len(no_accepted_answers), len(has_accepted_answer))
)
```

```
3584 questions with no answers, 5933 with answers, 4964 with an accepted answer.
```

Nous avons une répartition relativement comparable entre les questions auxquelles on a répondu, celles auxquelles on a partiellement répondu et celles auxquelles on n'a pas répondu. Cela semble raisonnable, de sorte que nous pouvons nous sentir suffisamment confiants pour poursuivre notre exploration.

Nous comprenons le format de nos données et nous en avons suffisamment pour commencer. Si vous travaillez sur un projet et que votre jeu de données actuel soit trop petit, ou contienne une majorité de caractéristiques trop difficiles à interpréter, vous devriez rassembler des données supplémentaires ou bien essayer un jeu de données entièrement différent.

Notre jeu de données est d'une qualité suffisante pour continuer. Il est maintenant temps de l'explorer plus en profondeur, dans le but d'éclairer notre stratégie de modélisation.

Étiqueter pour trouver les tendances des données

L'identification des tendances dans notre jeu de données n'est pas seulement une question de qualité. Cette partie du travail consiste à nous mettre à la place de notre modèle et à essayer de prédire le type de structure qu'il adoptera. Nous y parviendrons en séparant les données en différents clusters (j'expliquerai la notion de cluster plus loin dans ce chapitre) et en essayant d'extraire les points communs dans chaque cluster.

Voyons une liste des étapes à suivre pour faire ce travail en pratique. Nous commencerons par générer des statistiques sommaires de notre jeu de données, puis nous verrons comment l'explorer rapidement en tirant parti des techniques de vectorisation. Grâce à cette vectorisation et au clustering, nous explorerons notre jeu de données de manière efficace.

Statistiques de synthèse

Lorsque vous commencez à examiner un jeu de données, il est généralement judicieux de consulter quelques statistiques sommaires pour chacune des caractéristiques dont vous disposez. Cela

vous permet à la fois d'avoir une idée générale des caractéristiques de votre jeu de données et de trouver un moyen facile de séparer vos classes.

L'identification précoce des différences de distribution entre les classes de données est utile en ML, car elle facilitera notre tâche de modélisation ou nous évitera de surestimer les performances d'un modèle qui ne fait que tirer parti d'une caractéristique particulièrement informative.

Par exemple, si vous essayez de prédire si des tweets expriment une opinion positive ou négative, vous pouvez commencer par compter le nombre moyen de mots dans chaque tweet. Vous pourriez ensuite tracer un histogramme de cette caractéristique pour connaître sa distribution.

Un histogramme vous permettrait de remarquer si tous les tweets positifs sont plus courts que les négatifs. Cela pourrait vous amener à ajouter la longueur des mots comme prédicteur pour vous faciliter la tâche, ou au contraire collecter des données supplémentaires pour que votre modèle puisse connaître le contenu des tweets et pas seulement leur longueur.

Pour illustrer ce point, voici quelques statistiques sommaires concernant notre Éditeur ML.

Statistiques sommaires pour l'Éditeur ML

Pour notre exemple, nous pouvons tracer un histogramme de la longueur des questions dans notre jeu de données, en mettant en évidence les différentes tendances entre les questions à haut et à bas score. Voici comment nous procédons en utilisant pandas :

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

"""
df contient des questions et leur réponse obtenues à partir de
writers.stackexchange.com
Nous dessinons deux histogrammes : un pour les questions dont le score
est inférieur au score médian et un pour les questions dont le score
est supérieur au score médian
Dans les deux cas, nous éliminons les valeurs aberrantes
pour simplifier notre visualisation
"""

high_score = df["Score"] > df["Score"].median()
# Nous filtrons les questions réellement longues
normal_length = df["text_len"] < 2000

ax = df[df["is_question"] & high_score & normal_length]["text_len"].hist(
    bins=60,
    density=True,
    histtype="step",
    color="orange",
    linewidth=3,
    grid=False,
    figsize=(16, 10),
)
```

```
df[df["is_question"] & ~high_score & normal_length]["text_len"].hist(
    bins=60,
    density=True,
    histtype="step",
    color="purple",
    linewidth=3,
    grid=False,
)

handles = [
Rectangle((0, 0), 1, 1, color=c, ec="k") for c in ["orange", "purple"]
]
labels = ["High score", "Low score"]
plt.legend(handles, labels)
ax.set_xlabel("Sentence length (characters)")
ax.set_ylabel("Percentage of sentences")
```

La Figure 4.2 montre que les distributions sont en grande partie similaires, les questions ayant un score élevé ayant tendance à être légèrement plus longues (cette tendance est particulièrement visible autour de 800 caractères). Cela indique que cette caractéristique de longueur peut être utile pour un modèle permettant de prédire le score d'une question.

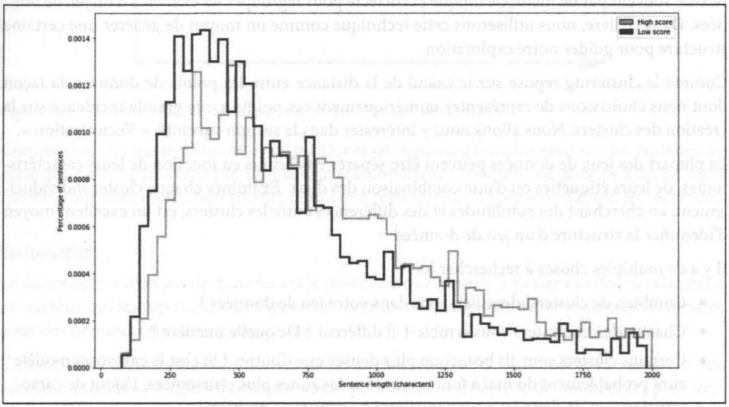


Figure 4.2 : Histogramme de la longueur du texte pour les questions avec un score élevé et faible.

Nous pouvons tracer d'autres variables de la même manière pour identifier davantage de caractéristiques potentielles. Une fois que nous avons identifié quelques caractéristiques, examinons de plus près notre jeu de données afin de pouvoir identifier des tendances plus granulaires.

Explorer et étiqueter efficacement

Vous ne pouvez pas aller plus loin en examinant des statistiques descriptives comme des moyennes et des graphiques tels que des histogrammes. Pour développer une intuition pour vos données, vous devriez passer un certain temps à examiner des points de données individuels. Cependant, il est assez inefficace de passer en revue les points d'un jeu de données en les prenant au hasard. Dans cette section, je vais vous expliquer comment maximiser votre efficacité lorsque vous visualisez des points de données individuels.

Le *clustering* (ou regroupement) est une méthode utile ici. Il consiste à regrouper un ensemble d'objets de telle sorte que les objets d'un même groupe (appelé un *cluster*) se ressemblent davantage (dans un certain sens) les uns les autres que ceux d'autres groupes (*clusters*). Nous utiliserons le clustering à la fois pour explorer nos données et par la suite pour les prédictions de notre modèle (voir la section « Réduction de la dimensionnalité », plus loin dans ce chapitre).

De nombreux algorithmes de clustering regroupent les points de données en mesurant la distance entre ces points et en attribuant à un même groupe ceux qui sont proches les uns des autres. La Figure 4.3 montre un exemple d'algorithme de clustering qui sépare un jeu de données en trois groupes/clusters différents. Le clustering est une méthode non supervisée, et donc il n'existe souvent pas de méthode unique et correcte pour regrouper les éléments d'un jeu de données. Dans ce livre, nous utiliserons cette technique comme un moyen de générer une certaine structure pour guider notre exploration.

Comme le clustering repose sur le calcul de la distance entre les points de données, la façon dont nous choisissons de représenter numériquement ces points a une grande incidence sur la création des clusters. Nous allons nous y intéresser dans la section suivante, « Vectorisation ».

La plupart des jeux de données peuvent être séparés en clusters en fonction de leurs caractéristiques, de leurs étiquettes ou d'une combinaison des deux. Examiner chaque cluster individuellement, en cherchant des similitudes et des différences entre les clusters, est un excellent moyen d'identifier la structure d'un jeu de données.

Il y a de multiples choses à rechercher ici :

- Combien de clusters identifiez-vous dans votre jeu de données ?
- Chacun de ces clusters vous semble-t-il différent ? De quelle manière ?
- Certains clusters sont-ils beaucoup plus denses que d'autres ? Si c'est le cas, votre modèle aura probablement du mal à fonctionner sur les zones plus clairsemées. L'ajout de caractéristiques et de données peut contribuer à atténuer ce problème.
- Tous les clusters représentent-ils des données qui semblent aussi « difficiles » à modéliser ? Si certains clusters semblent représenter des points de données plus complexes, notez-les afin de pouvoir les réexaminer lorsque nous évaluerons les performances de notre modèle.

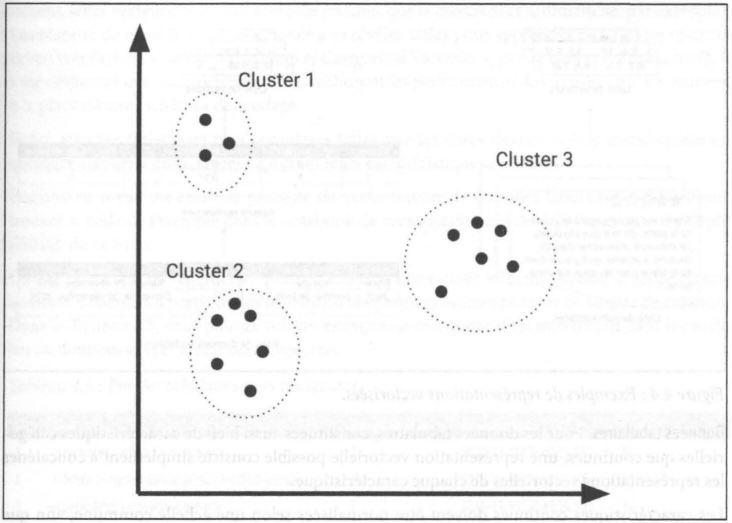


Figure 4.3 : Génération de trois clusters à partir d'un jeu de données.

Comme nous l'avons mentionné, les algorithmes de clustering fonctionnent sur des vecteurs, de sorte que nous ne pouvons pas simplement transmettre un ensemble de phrases à un tel algorithme. Pour que nos données soient prêtes à être regroupées, nous devons d'abord les vectoriser.

Vectorisation

La *vectorisation* d'un jeu de données est le processus qui consiste à passer des données brutes à un vecteur qui les représente. La Figure 4.4 montre un exemple de représentations vectorisées pour des données textuelles et tabulaires.

Il existe de nombreuses façons de vectoriser les données. Nous nous concentrerons donc sur quelques méthodes simples qui fonctionnent pour certains des types de données les plus courants, comme les données tabulaires, le texte et les images.

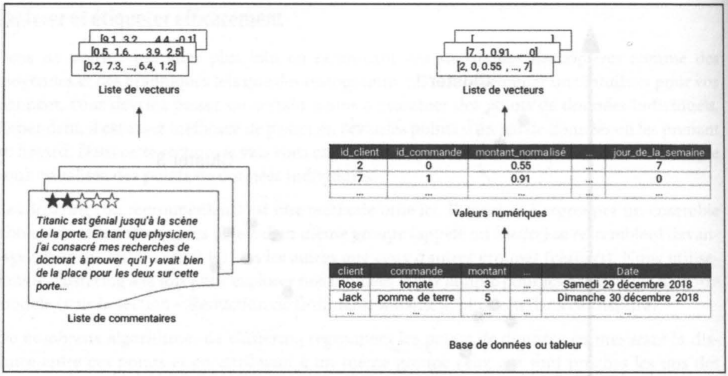


Figure 4.4 : Exemples de représentations vectorisées.

Données tabulaires. Pour les données tabulaires, constituées aussi bien de caractéristiques catégorielles que continues, une représentation vectorielle possible consiste simplement à concaténer les représentations vectorielles de chaque caractéristique.

Les caractéristiques continues doivent être normalisées selon une échelle commune, afin que celles ayant une plus grande échelle n'entraînent pas l'ignorance totale par les modèles des caractéristiques plus petites. Il existe différentes façons de normaliser les données, mais commencer par transformer chaque caractéristique de telle sorte que sa moyenne soit nulle et sa variance de un est souvent une bonne première étape. Cette *normalisation* est aussi appelée *score standard* (ou *standardization* en anglais).

Les caractéristiques catégorielles, telles que les couleurs, peuvent être converties en un encodage one-hot : une liste aussi longue que le nombre de valeurs distinctes de la caractéristique ne comprenant que des zéros et un seul un, dont l'indice correspond à la valeur actuelle (par exemple, dans un ensemble de données contenant quatre couleurs distinctes, nous pourrions encoder le rouge comme valant [1, 0, 0, 0] et le bleu comme valant [0, 0, 1, 0]). Vous vous demandez peut-être pourquoi nous n'assignerions pas simplement un nombre à chaque valeur potentielle, comme 1 pour le rouge et 3 pour le bleu. C'est parce qu'un tel schéma de codage impliquerait un ordre entre les valeurs (le bleu serait plus grand que le rouge), ce qui est souvent incorrect pour les variables catégorielles.

Une propriété de l'encodage one-hot est que la distance entre deux valeurs de caractéristiques données est toujours égale à un. Cela fournit souvent une bonne représentation pour un modèle, mais dans certains cas, comme les jours de la semaine, certaines valeurs peuvent être plus similaires que d'autres (le samedi et le dimanche sont tous deux des jours de week-end, et donc, idéa-

lement, leurs vecteurs devraient être plus proches que le mercredi et le dimanche, par exemple). Les réseaux de neurones ont commencé à se révéler utiles pour apprendre de telles représentations (voir l'article « *Entity Embeddings of Categorical Variables* », par C. Guo et F. Berkhahn[13]). Il a été démontré que ces représentations améliorent les performances des modèles qui les utilisent à la place d'autres schémas d'encodage.

Enfin, les caractéristiques plus complexes telles que les dates devraient être transformées en quelques éléments numériques capturant leurs caractéristiques saillantes.

Passons en revue un exemple pratique de vectorisation de données tabulaires. Vous pouvez trouver le code de l'exemple dans le notebook de vectorisation des données tabulaires du dépôt GitHub de ce livre.

Disons qu'au lieu de regarder le contenu des questions, nous voulons prédire le score qu'une question obtiendra à partir de ses balises, du nombre de commentaires et de sa date de création. Dans le Tableau 4.3, vous pouvez voir un exemple de ce à quoi cela ressemblerait dans le cas du jeu de données `writers.stackexchange.com`.

Tableau 4.3 : Entrées tabulaires sans traitement.

Id	Tags	CommentCount	CreationDate	Score
1	<resources><first-time-author>	7	2010-11-18T20:40:32.857	32
2	<fiction><grammatical-person><third-person>	0	2010-11-18T20:42:31.513	20
3	<publishing><novel><agent>	1	2010-11-18T20:43:28.903	34
5	<plot><short-story><planning><brainstorming>	0	2010-11-18T20:43:59.693	28
7	<fiction><genre><categories>	1	2010-11-18T20:45:44.067	21

Chaque question comporte plusieurs balises, ainsi qu'une date et un certain nombre de commentaires. Prétraitons chacune d'entre elles. Tout d'abord, nous normalisons les champs numériques :

```
def get_norm(df, col):
    return (df[col] - df[col].mean()) / df[col].std()

tabular_df["NormComment"]= get_norm(tabular_df, "CommentCount") tabular_df["NormScore"]= get_
    norm(tabular_df, "Score")
```

Ensuite, nous extrayons les informations pertinentes à partir de la date. Nous pourrions, par exemple, choisir l'année, le mois, le jour et l'heure de l'affichage. Chacun de ces éléments est une valeur numérique que notre modèle peut utiliser.

```
# Convertit nos dates en un datetime pandas
tabular_df["date"] = pd.to_datetime(tabular_df["CreationDate"])
```

13 Voir l'adresse `https://arxiv.org/pdf/1604.06737.pdf`.

```
# Extrait des caractéristiques significatives de l'objet datetime
tabular_df["year"] = tabular_df["date"].dt.year
tabular_df["month"] = tabular_df["date"]
dt.month tabular_df["day"] = tabular_df["date"]
dt.day tabular_df["hour"] = tabular_df["date"].dt.hour
```

Nos balises sont des caractéristiques catégorielles, chaque question pouvant recevoir un nombre quelconque de balises. Comme nous l'avons vu précédemment, la manière la plus simple de représenter les entrées catégorielles est de faire appel à un encodage one-hot, en transformant chaque balise en sa propre colonne, chaque question ayant une valeur de 1 pour une caractéristique de balise donnée, et ce uniquement si cette balise est associée à cette question.

Comme nous avons plus de trois cents balises dans notre jeu de données, nous avons choisi de ne créer ici qu'une colonne pour les cinq balises les plus populaires qui sont utilisées dans plus de cinq cents questions. Nous pourrions ajouter chacune des balises, mais comme la majorité d'entre elles n'apparaissent qu'une seule fois, cela ne serait pas utile pour identifier des motifs dans les données.

```
# Sélectionne nos balises, représentées sous forme de chaînes,
# et les transforme en tableaux de balises
tags = tabular_df["Tags"]
clean_tags = tags.str.split("><").apply(
    lambda x: [a.strip("<").strip(">") for a in x])

# Utilise get_dummies de pandas pour obtenir des valeurs indicateur
# sélectionne uniquement les balises qui apparaissent plus de 500 fois
tag_columns = pd.get_dummies(clean_tags.apply(pd.Series).stack()).sum(level=0)
all_tags = tag_columns.astype(bool).sum(axis=0).sort_values(ascending=False) top_tags = all_
   tags[all_tags > 500]
top_tag_columns = tag_columns[top_tags.index]

# Ajoute nos balises dans notre DataFrame initial
final = pd.concat([tabular_df, top_tag_columns], axis=1)

# On ne conserve que les éléments vectorisés
col_to_keep = ["year", "month", "day", "hour", "NormComment",
               "NormScore"] + list(top_tags.index)
final_features = final[col_to_keep]
```

Dans le Tableau 4.4, vous pouvez voir que nos données sont maintenant entièrement vectorisées, chaque ligne ne comportant que des valeurs numériques. Nous pouvons injecter ces données dans un algorithme de clustering, ou dans un modèle de ML supervisé.

Tableau 4.4 : Entrées tabulaires vectorisées.

Id	Year	Month	Day	Hour	Norm-Comment	Norm-Score	Creative writing	Fiction	Style	Characters	Technique	Novel	Publishing
1	2010	11	18	20	0.165706	0.140501	0	0	0	0	0	0	0
2	2010	11	18	20	-0.103524	0.077674	0	1	0	0	0	0	0
3	2010	11	18	20	-0.065063	0.150972	0	0	0	0	0	1	1
5	2010	11	18	20	-0.103524	0.119558	0	0	0	0	0	0	0
7	2010	11	18	20	-0.065063	0.082909	0	1	0	0	0	0	0

Vectorisation et fuites de données

Vous utiliserez généralement les mêmes techniques pour vectoriser les données afin de les visualiser et pour les injecter dans un modèle. Il existe toutefois une distinction importante. Lorsque vous vectorisez des données pour alimenter un modèle, vous devez vectoriser vos données d'entraînement et sauvegarder les paramètres que vous avez utilisés pour obtenir les vecteurs servant à cet entraînement. Vous devez ensuite utiliser les mêmes paramètres pour vos jeux de validation et de test.

Lors de la normalisation des données, par exemple, vous devez calculer des statistiques sommaires, telles que la moyenne et l'écart-type uniquement sur votre jeu d'entraînement (en utilisant les mêmes valeurs pour normaliser vos données de validation), et lors de l'inférence en production.

L'utilisation de vos données de validation et d'entraînement pour la normalisation, ou pour décider quelles catégories conserver dans votre encodage one-hot, provoquerait des fuites de données, comme ce serait le cas en vous appuyant sur des informations provenant de l'extérieur de votre jeu d'entraînement afin de créer des caractéristiques pour celui-ci. Cela gonflerait artificiellement les performances de votre modèle, mais le rendrait moins performant en production. Nous aborderons ce point plus en détail dans la section « Fuite de données » dans le Chapitre 5.

Différents types de données exigent différentes méthodes de vectorisation. En particulier, les données textuelles nécessitent souvent des approches plus créatives.

Données textuelles. La façon la plus simple de vectoriser du texte consiste à utiliser un vecteur de comptage, qui est l'équivalent pour les mots d'un encodage one-hot. Commencez par construire un vocabulaire constitué de la liste des mots uniques de votre jeu de données. Associez chaque mot de ce vocabulaire à un indice (de 0 à la taille du vocabulaire). Vous pouvez ensuite représenter chaque phrase ou paragraphe par une liste aussi longue que votre vocabulaire. Pour chaque phrase, la valeur de chaque indice représente le nombre d'occurrences du mot associé dans la phrase donnée.

Cette méthode ne tient pas compte de l'ordre des mots dans une phrase et est donc appelée *sac de mots*. La Figure 4.5 montre deux phrases et leur représentation par sac de mots. Les deux phrases sont transformées en vecteurs qui contiennent des informations sur le nombre de fois où un mot apparaît dans une phrase, mais pas sur l'ordre dans lequel les mots sont présents dans cette phrase.

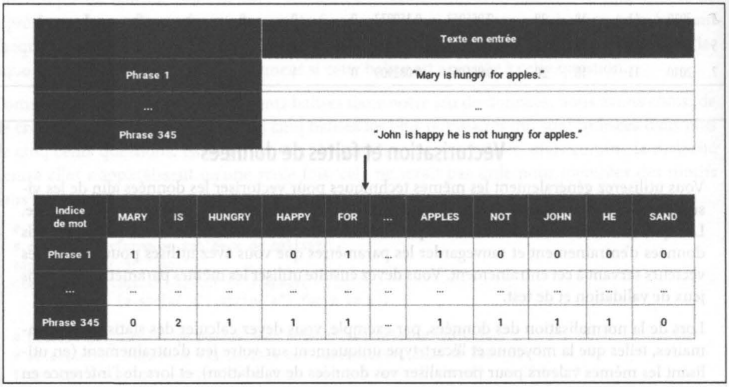


Figure 4.5 : Obtention de vecteurs « sac de mots » à partir de phrases.

L'utilisation d'une représentation par sac de mots ou de sa version normalisée TF-IDF (abréviation de Term Frequency-Inverse Document Frequency) est simple en utilisant scikit-learn, comme vous pouvez le voir ici :

```
# Crée une instance d'un vectoriseur tfidf
# Nous pourrions utiliser CountVectorizer pour une version non normalisée
vectorizer = TfidfVectorizer()

# Remplit notre vectoriseur avec les questions de notre base de données
# Retourne un tableau de texte vectorisé
bag_of_words = vectorizer.fit_transform(df[df["is_question"]]["Text"])
```

De nombreuses méthodes novatrices de vectorisation de texte ont été développées au fil des ans, à commencer par Word2Vec en 2013 (voir l'article « *Efficient Estimation of Word Representations in Vector Space* » de Mikolov *et al.*[14]) et des approches plus récentes telles que fastText (voir l'article « *Bag of Tricks for Efficient Text Classification* » de Joulin *et al.*[15]) Ces techniques de vecto-

14 Voir l'adresse `https://arxiv.org/abs/1301.3781`.

15 Voir l'adresse `https://arxiv.org/abs/1607.01759`.

risation produisent des vecteurs de mots qui tentent d'apprendre une représentation capturant mieux les similitudes entre les concepts qu'un codage TF-IDF. Elles y parviennent en apprenant quels mots tendent à apparaître dans des contextes similaires dans de grands corpus de texte tels que Wikipédia. Cette approche est basée sur l'hypothèse de distribution, qui prétend que les éléments linguistiques ayant des distributions similaires ont des significations similaires.

Concrètement, cela se fait en apprenant un vecteur pour chaque mot et en entraînant un modèle pour prédire un mot manquant dans une phrase en utilisant à cet effet les vecteurs de mots qui l'entourent. Le nombre de mots voisins à prendre en compte est appelé la *taille de la fenêtre*. Sur la Figure 4.6, vous pouvez voir une représentation de cette tâche pour une taille de fenêtre de deux. Sur la gauche, les vecteurs de mots pour les deux mots avant et après la cible sont injectés dans un modèle simple. Ce modèle simple et les valeurs des vecteurs de mots sont ensuite optimisés pour que la sortie corresponde au vecteur de mots du mot manquant.

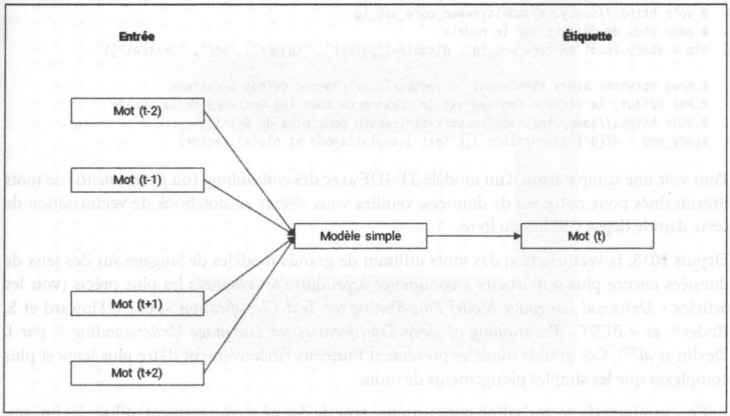


Figure 4.6 : Apprentissage de vecteurs de mots, d'après l'article de Word2Vec « Efficient Estimation of Word Representations in Vector Space » *de Mikolov* et al.

Il existe de nombreux modèles de vectorisation de mots préentraînés en open source. L'utilisation de vecteurs produits par un modèle qui a été préentraîné sur un grand corpus (souvent Wikipédia ou une archive de nouvelles) peut aider nos modèles à mieux exploiter le sens sémantique des mots courants.

Par exemple, les vecteurs de mots mentionnés dans l'article fastText de Joulin *et al.* sont disponibles en ligne dans un outil indépendant. Pour une approche plus personnalisée, spaCy est une boîte à outils de TLN qui fournit des modèles préentraînés pour une variété de tâches, ainsi que des moyens faciles de construire les vôtres (`https://spacy.io/`).

Voici un exemple d'utilisation de spaCy pour charger des vecteurs de mots préentraînés et les utiliser afin d'obtenir un vecteur de phrase sémantiquement significatif. En arrière-plan, spaCy récupère la valeur préentraînée pour chaque mot de notre jeu de données (ou l'ignore s'il ne faisait pas partie de sa tâche de préentraînement) et fait la moyenne de tous les vecteurs d'une question pour obtenir une représentation de celle-ci.

```
import spacy

# Nous chargeons un grand modèle, et nous désactivons
# les pièces inutiles du pipeline pour notre tâche
# Cela accélère considérablement le processus de vectorisation
# Voir https://spacy.io/models/en#en_core_web_lg
# pour plus de détails sur le modèle
nlp = spacy.load('en_core_web_lg', disable=["parser", "tagger", "ner", "textcat"])

# Nous obtenons alors simplement le vecteur pour chacune de nos questions
# Par défaut, le vecteur renvoyé est la moyenne de tous les vecteurs de la phrase
# Voir https://spacy.io/usage/vectors-similarity pour plus de détails
spacy_emb = df[df["is_question"]]["Text"].apply(lambda x: nlp(x).vector)
```

Pour voir une comparaison d'un modèle TF-IDF avec des *embeddings* (ou *plongements*) de mots préentraînés pour notre jeu de données, veuillez vous référer au notebook de vectorisation de texte dans le dépôt GitHub du livre.

Depuis 2018, la vectorisation des mots utilisant de grands modèles de langage sur des jeux de données encore plus importants a commencé à produire les résultats les plus précis (voir les articles « *Universal Language Model Fine-Tuning for Text Classification* », par J. Howard et S. Ruder[16], et « *BERT : Pre-training of Deep Transformers for Language Understanding* », par J. Devlin *et al.*[17]). Ces grands modèles présentent toutefois l'inconvénient d'être plus lents et plus complexes que les simples plongements de mots.

Enfin, examinons la vectorisation pour un autre type de données couramment utilisé : les images.

Données d'images. Les données d'images sont déjà vectorisées, car une image n'est rien d'autre qu'un tableau multidimensionnel de nombres, souvent appelés *tenseurs*[18] dans la communauté ML. La plupart des images RVB standard à trois canaux, par exemple, sont simplement stockées sous la forme d'une liste de nombres de longueur égale à la hauteur de l'image en pixels, multipliée par sa largeur, multipliée par trois (pour les canaux rouge, vert et bleu). La Figure 4.7

16 Voir l'adresse `https://arxiv.org/abs/1801.06146`.
17 Voir l'adresse `https://arxiv.org/abs/1810.04805`.
18 Voir l'adresse `https://fr.wikipedia.org/wiki/Tenseur`.

illustre comment nous pouvons représenter une image comme un tenseur de nombres, représentant l'intensité de chacune des trois couleurs primaires.

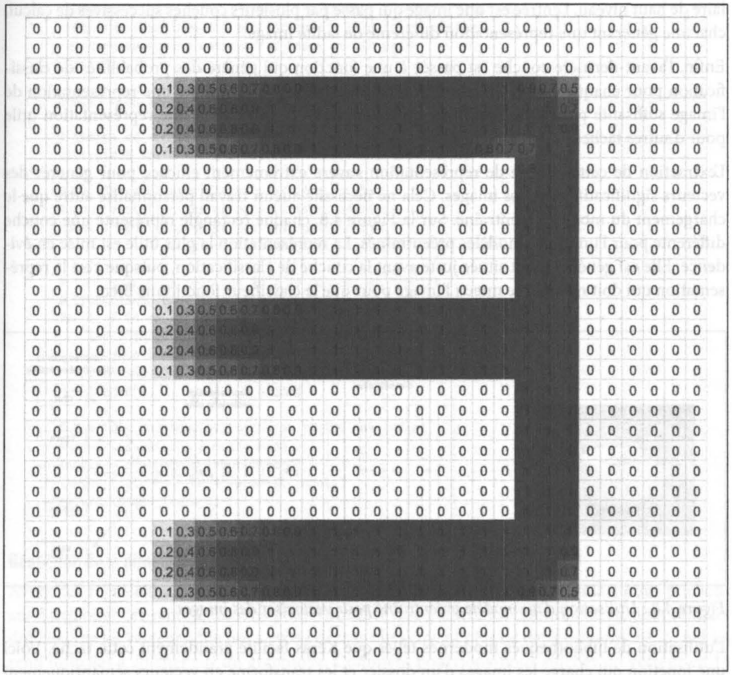


Figure 4.7 : Représentation d'un 3 sous forme de matrice de valeurs de 0 à 1 (seul le canal rouge est montré ici).

Bien que nous puissions utiliser cette représentation telle quelle, nous aimerions que nos tenseurs saisissent un peu plus le sens sémantique de nos images. Pour ce faire, nous pouvons utiliser une approche similaire à celle du texte et exploiter de grands réseaux de neurones préentraînés.

Les modèles qui ont été entraînés sur des jeux de données de classification massifs tels que VGG (voir l'article de A. Simonyan et A. Zimmerman, « *Very Deep Convolutional Networks for Large-*

Scale Image Recognition »[19]) ou Inception (voir l'article de C. Szegedy *et al.*, « *Going Deeper with Convolutions* »[20]) sur le jeu de données ImageNet[21] finissent par apprendre des représentations très expressives afin de bien classifier. Ces modèles suivent pour la plupart une structure similaire de haut niveau. L'entrée est une image qui passe par plusieurs couches successives de calcul, chacune générant une représentation différente de ladite image.

Enfin, l'avant-dernière couche est passée à une fonction qui génère des probabilités de classification pour chaque classe. Cette avant-dernière couche contient donc une représentation de l'image suffisante pour classifier l'objet qu'elle contient, ce qui en fait une représentation utile pour d'autres tâches.

L'extraction de cette couche de représentation s'avère extrêmement efficace pour générer des vecteurs significatifs pour les images. Cela ne nécessite aucun travail personnalisé autre que le chargement du modèle préentraîné. Sur la Figure 4.8, chaque rectangle représente une couche différente pour l'un de ces modèles préentraînés. La représentation la plus utile est mise en évidence. Elle est généralement située juste avant la couche de classification, puisque c'est la représentation qui doit résumer le mieux l'image pour que le classifieur fonctionne bien.

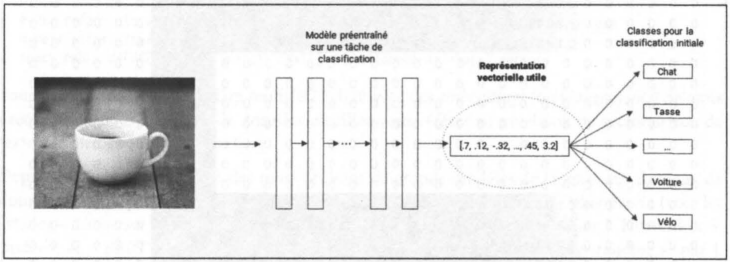


Figure 4.8 : Utilisation d'un modèle préentraîné pour vectoriser des images.

L'utilisation de bibliothèques modernes telles que Keras facilite grandement cette tâche. Voici une fonction qui charge les images d'un dossier et les transforme en vecteurs sémantiquement significatifs pour une analyse en aval, en utilisant un réseau préentraîné disponible dans Keras :

```
import numpy as np

from keras.preprocessing import image
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
```

19 Voir l'adresse `https://arxiv.org/abs/1409.1556`.
20 Voir l'adresse `https://arxiv.org/abs/1409.4842`.
21 Voir l'adresse `http://www.image-net.org/`.

```
def generate_features(image_paths):
    """
    Prend un tableau de chemins d'accès à des images
    Renvoie des caractéristiques préentrainées pour chaque image
    :paramètre image_paths : tableau de chemins d'accès aux images
    :retourne : tableau des activations de la dernière couche,
     et correspondance entre array_index et file_path
    """

    images = np.zeros(shape=(len(image_paths), 224, 224, 3))

    # Charge un modèle préentrainé
    pretrained_vgg16 = VGG16(weights='imagenet', include_top=True)

    # Utilise seulement l'avant-dernière couche,
    # pour exploiter les caractéristiques apprises
    model = Model(inputs=pretrained_vgg16.input,
                  outputs=pretrained_vgg16.get_layer('fc2').output)

    # Nous chargeons tous nos jeux de données en mémoire
    # (fonctionne pour de petits jeux de données)
    for i, f in enumerate(image_paths):
        img = image.load_img(f, target_size=(224, 224))
        x_raw = image.img_to_array(img)
        x_expand = np.expand_dims(x_raw, axis=0)
        images[i, :, :, :] = x_expand

    # Une fois que nous avons chargé toutes nos images,
    # nous les passons à notre modèle
    inputs = preprocess_input(images)
    images_features = model.predict(inputs)
    return images_features
```

Réduction de la dimensionnalité

Avoir des représentations vectorielles est nécessaire pour les algorithmes, mais nous pouvons aussi exploiter ces représentations pour visualiser directement les données ! Cela peut sembler difficile, car les vecteurs que nous avons décrits ont souvent plus de deux dimensions, ce qui les rend difficiles à afficher sur un graphique. Comment pourrions-nous afficher un vecteur en 14 dimensions ?

Une fois que vous avez une représentation vectorisée, vous pouvez la transformer en clusters ou transmettre vos données à un modèle, mais vous pouvez aussi l'utiliser pour inspecter plus efficacement votre jeu de données. En regroupant des points de données ayant des représentations similaires, il est possible d'examiner plus rapidement les tendances dans votre jeu de données. Nous allons voir comment procéder.

Apprentissage par transfert

Les modèles préentraînés sont utiles pour vectoriser nos données, mais ils peuvent aussi parfois être entièrement adaptés à notre tâche. L'apprentissage par transfert est le processus consistant à utiliser un modèle qui a été préalablement entraîné sur un jeu de données, ou une tâche, pour l'appliquer à un autre jeu de données, ou à une autre tâche. Plus que la simple réutilisation de la même architecture ou du même pipeline, l'apprentissage par transfert se sert des poids précédemment appris d'un modèle entraîné comme point de départ pour une nouvelle tâche.

L'apprentissage par transfert peut en théorie fonctionner pour passer de n'importe quelle tâche à n'importe quelle autre, mais il est couramment utilisé afin d'améliorer les performances de petits jeux de données, en transférant des poids de grands jeux de données tels que ImageNet pour la vision par ordinateur ou WikiText pour le TLN[22].

Si l'apprentissage par transfert améliore souvent les performances, il peut également introduire une source supplémentaire de biais non désirés. Même si vous nettoyez soigneusement votre jeu de données actuel, si vous utilisez un modèle qui a été préentraîné sur la totalité de Wikipédia, par exemple, il pourrait véhiculer le préjugé sexiste dont il a été montré qu'il y était présent (voir l'article « *Gender Bias in Neural Natural Language Processing* » par K. Lu *et al.*[23]).

Geoffrey Hinton, qui a remporté un prix Turing pour son travail sur le deep learning, reconnaît ce problème dans une conférence avec le conseil suivant : « Pour traiter des hyperplans dans un espace à 14 dimensions, visualisez un espace en 3D et dites-vous *quatorze* très fort. Tout le monde le fait ». (Voir ici la diapositive 16 de la conférence de G. Hinton *et al.* « *An Overview of the Main Types of Neural Network Architecture* »[24]). Si cela vous semble difficile, vous serez ravi d'entendre parler de la *réduction de la dimensionnalité*, qui est la technique consistant à représenter les vecteurs avec un nombre réduit de dimensions tout en préservant autant que possible leur structure.

Les techniques de réduction de la dimensionnalité telles que t-SNE (voir l'article de L. van der Maaten et G. Hinton, PCA, « *Visualizing Data Using t-SNE* »[25]), et UMAP (voir l'article de L. McInnes *et al.*, « *UMAP : Uniform Manifold Approximation and Projection for Dimension Reduction* »[26]) vous permettent de projeter des données avec de hautes dimensions, telles que des vecteurs représentant des phrases, des images ou d'autres caractéristiques, sur un plan 2D.

Ces projections sont utiles pour remarquer des tendances dans les données que vous pouvez ensuite étudier. Il s'agit toutefois de représentations approximatives des données réelles. Vous devez donc valider toute hypothèse que vous faites en examinant un tel tracé en faisant appel à d'autres méthodes. Si vous voyez des groupes de points appartenant tous à une même classe et

22 Voir l'adresse `https://sforce.co/2Y1I38X`.
23 Voir l'adresse `https://arxiv.org/abs/1807.11714`.
24 Voir l'adresse `https://bit.ly/353o5gH`.
25 Voir l'adresse `https://bit.ly/2XYN4jD`.
26 Voir l'adresse `https://arxiv.org/abs/1802.03426`.

qui semblent avoir une caractéristique commune, vérifiez que votre modèle exploite effectivement cette caractéristique, par exemple.

Pour commencer, tracez vos données à l'aide d'une technique de réduction de la dimensionnalité, et colorez chaque point à l'aide d'un attribut que vous cherchez à inspecter. Pour les tâches de classification, commencez par colorer chaque point en vous basant sur son étiquette. Pour les tâches non supervisées, vous pouvez colorier les points en fonction par exemple des valeurs des caractéristiques fournies qui vous intéressent. Cela vous permet de voir si certaines régions semblent faciles, ou au contraire plus difficiles, à séparer pour votre modèle.

Voici comment le faire facilement en utilisant UMAP, en lui transmettant les résultats que nous avons générés plus haut dans la section « Vectorisation » :

```
import umap

# Ajuste UMAP à nos données, et renvoie les données transformées
umap_emb = umap.UMAP().fit_transform(embeddings)

fig = plt.figure(figsize=(16, 10))
color_map = {
    True: '#ff7f0e',
    False:'#1f77b4'
}
plt.scatter(umap_emb[:, 0], umap_emb[:, 1],
            c=[color_map[x] for x in sent_labels],
            s=40, alpha=0.4)
```

Pour rappel, nous avons décidé de commencer par utiliser uniquement les données de la communauté des écrivains de Stack Exchange. Le résultat pour ce jeu de données est affiché sur la Figure 4.9. À première vue, nous pouvons voir quelques régions que nous devrions explorer, comme la région dense des questions sans réponse, en haut à gauche. Si nous pouvons identifier les caractéristiques qu'elles ont en commun, nous serons à même de découvrir une caractéristique de classification utile.

Une fois les données vectorisées et tracées, il est généralement bon de commencer à identifier systématiquement des groupes de points de données similaires et de les explorer. Nous pourrions le faire simplement en examinant les tracés UMAP, mais nous pouvons également tirer parti du clustering.

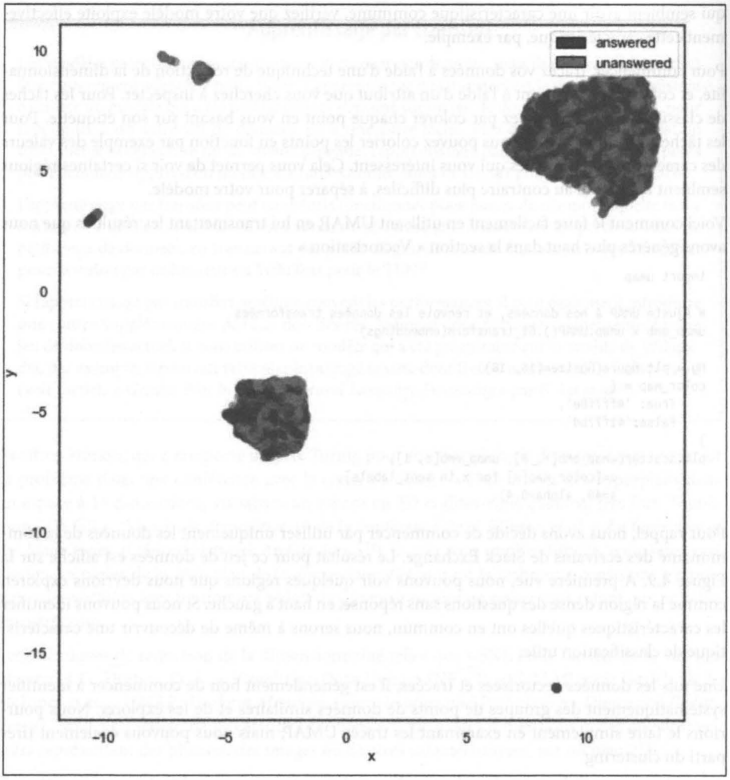


Figure 4.9 : Graphique UMAP coloré selon qu'une question donnée a reçu ou non une réponse satisfaisante.

Clustering

Nous avons déjà mentionné le clustering comme méthode pour extraire une structure à partir des données. Que le but soit d'inspecter un jeu de données ou d'analyser les performances d'un modèle, comme nous le ferons au Chapitre 5, le clustering est un outil essentiel à avoir dans

votre arsenal. J'utilise le clustering de la même manière que la réduction de la dimensionnalité, comme un moyen supplémentaire de faire apparaître les problèmes et les points de données intéressants.

Une méthode de base pour regrouper des données en pratique consiste à commencer par essayer quelques algorithmes simples, tels que les k-moyennes[27], et à ajuster leurs hyperparamètres, comme le nombre de clusters, jusqu'à obtenir une performance satisfaisante.

Les performances du clustering sont difficiles à quantifier. En pratique, l'utilisation d'une combinaison de visualisation de données et de méthodes telles que la méthode dite *elbow method*[28] ou du *coefficient de silhouette*[29] est suffisante pour notre besoin, qui n'est pas de séparer parfaitement nos données mais d'identifier les régions où notre modèle peut présenter des problèmes.

Voici un exemple de code permettant de regrouper en clusters les données de notre jeu, et de visualiser nos groupes à l'aide d'une technique de dimensionnalité que nous avons décrite précédemment, UMAP.

```
from sklearn.cluster import KMeans
import matplotlib.cm as cm

# Choisit le nombre de clusters et la carte de couleurs
n_clusters=3
cmap = plt.get_cmap("Set2")

# Ajuste l'algorithme de clustering à nos caractéristiques vectorisées
clus = KMeans(n_clusters=n_clusters, random_state=10)
clusters = clus.fit_predict(vectorized_features)

# Trace les caractéristiques réduites de la dimensionnalité sur un plan 2D
plt.scatter(umap_features[:, 0], umap_features[:, 1],
            c=[cmap(x/n_clusters) for x in clusters], s=40, alpha=.4)
plt.title('UMAP projection of questions, colored by clusters', fontsize=14)
```

Comme vous pouvez le voir sur la Figure 4.10, la façon dont nous regrouperions instinctivement la représentation 2D ne correspond pas toujours aux clusters que notre algorithme trouve à partir des données vectorisées. Cela peut être dû à des artefacts dans notre algorithme de réduction de la dimensionnalité ou à une topologie de données complexe. En fait, l'ajout d'un cluster assigné à un point comme caractéristique peut parfois améliorer les performances d'un modèle en lui permettant d'exploiter ladite topologie.

Une fois que vous avez des clusters, examinez chacun d'entre eux pour essayer d'y identifier les tendances dans vos données. Pour ce faire, vous devez sélectionner quelques points par cluster et agir comme si vous étiez le modèle, en étiquetant donc ces points avec ce que vous pensez que la bonne réponse devrait être. Dans la prochaine section, je décrirai comment effectuer ce travail d'étiquetage.

27 Voir l'adresse `https://fr.wikipedia.org/wiki/K-moyennes`.
28 Voir l'adresse `https://en.wikipedia.org/wiki/Elbow_method_(clustering)`.
29 Voir l'adresse `https://fr.wikipedia.org/wiki/Silhouette_(clustering)`.

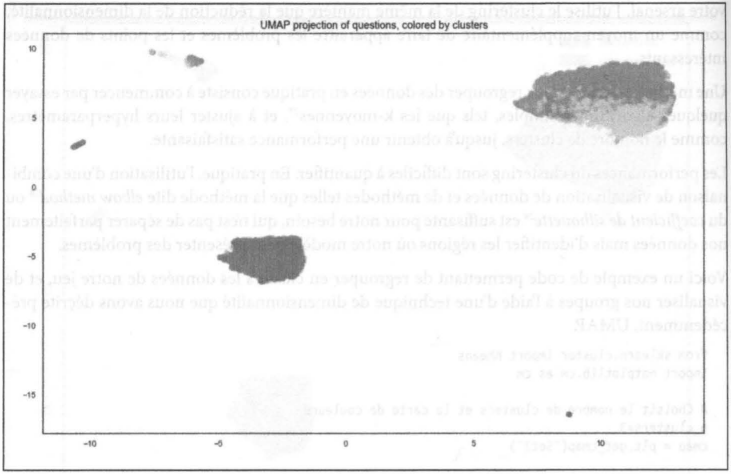


Figure 4.10 : Visualisation de nos questions, colorées par cluster.

Soyez l'algorithme

Une fois que vous aurez examiné les métriques agrégées et les informations sur les clusters, je vous encourage à suivre les conseils donnés par Monica Rogati dans le Chapitre 1 (voir la section « Comment choisir et hiérarchiser les projets ML ? ») et à essayer de faire le travail de votre modèle en étiquetant quelques points de données dans chaque cluster avec les résultats que vous aimeriez voir produise par un modèle.

Si vous n'avez jamais essayé de faire le travail de votre algorithme, il sera difficile de juger de la qualité de ses résultats. D'un autre côté, si vous passez un certain temps à étiqueter vous-même les données, vous remarquerez souvent des tendances qui vous faciliteront grandement la tâche de modélisation.

Vous vous souvenez peut-être avoir vu ce conseil dans notre précédente section sur l'heuristique, et cela ne devrait pas vous surprendre. Le choix d'une approche de modélisation implique de faire presque autant d'hypothèses sur nos données que de construire des heuristiques, et il est donc logique que ces hypothèses soient basées sur les données.

Vous devriez étiqueter les données même si votre jeu contient déjà des étiquettes. Cela vous permet de valider le fait que vos étiquettes saisissent les bonnes informations et qu'elles sont

correctes. Dans notre étude de cas, nous utilisons le score d'une question comme mesure de sa qualité, ce qui est une étiquette faible. En étiquetant nous-mêmes quelques échantillons, nous pourrons valider l'hypothèse selon laquelle cette étiquette est appropriée.

Une fois que vous aurez étiqueté quelques échantillons, n'hésitez pas à mettre à jour votre stratégie de vectorisation en ajoutant toutes les caractéristiques que vous découvrez pour vous aider à rendre la représentation de vos données aussi informative que possible, et revenez à l'étiquetage. Il s'agit d'un processus itératif, comme l'illustre la Figure 4.11.

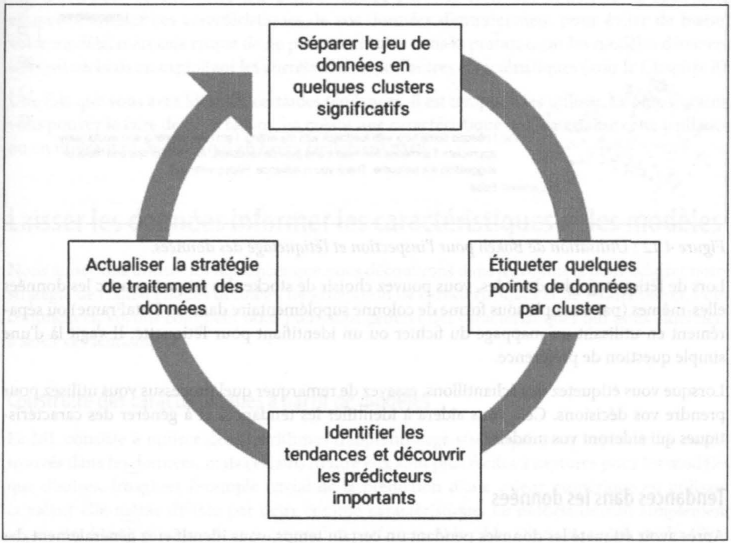


Figure 4.11 : Le processus d'étiquetage des données.

Pour accélérer votre étiquetage, assurez-vous de tirer parti de votre analyse préalable en étiquetant quelques points de données dans chaque cluster que vous avez identifié et pour chaque valeur courante dans votre distribution de caractéristiques.

Une façon d'y parvenir est de tirer parti des bibliothèques de visualisation pour explorer vos données de manière interactive. Bokeh (`https://docs.bokeh.org/en/latest/`) offre la possibilité de créer des tracés interactifs. Une façon rapide d'étiqueter les données consiste à parcourir un tracé de nos échantillons vectorisés, en étiquetant quelques échantillons pour chaque cluster.

La Figure 4.12 montre un exemple représentatif d'un cluster de questions restées pour la plupart sans réponse. Les questions de ce cluster étaient généralement assez vagues, et il était donc difficile d'y répondre objectivement. C'est pourquoi elles n'ont pas reçu de réponses. Elles sont qualifiées avec exactitude de mauvaises questions. Pour voir le code source de ce graphique et un exemple de son utilisation pour notre Éditeur ML, naviguez vers le notebook d'exploration de données pour générer des caractéristiques dans le dépôt GitHub de ce livre[30].

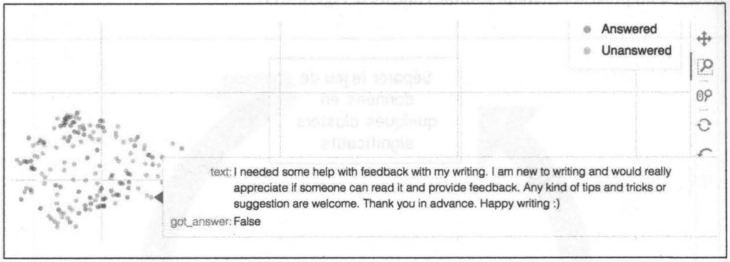


Figure 4.12 : Utilisation de Bokeh pour l'inspection et l'étiquetage des données.

Lors de l'étiquetage des données, vous pouvez choisir de stocker les étiquettes avec les données elles-mêmes (par exemple sous forme de colonne supplémentaire dans un DataFrame) ou séparément en utilisant un mappage du fichier ou un identifiant pour l'étiquette. Il s'agit là d'une simple question de préférence.

Lorsque vous étiquetez des échantillons, essayez de remarquer quel processus vous utilisez pour prendre vos décisions. Cela vous aidera à identifier les tendances et à générer des caractéristiques qui aideront vos modèles.

Tendances dans les données

Après avoir étiqueté les données pendant un certain temps, vous identifierez généralement des tendances. Certaines peuvent être instructives (les courts tweets sont généralement plus simples à classer comme positifs ou négatifs) et vous guider pour générer des caractéristiques utiles pour vos modèles. D'autres peuvent être des corrélations non pertinentes en raison de la manière dont les données ont été recueillies.

Peut-être que tous les tweets que nous avons collectés et qui sont en français se trouvent être négatifs, ce qui conduirait probablement un modèle à classifier automatiquement les tweets

30 En cas d'erreur d'exécution liée à l'attribut UMAP non reconnu, voyez par exemple la page `https://github.com/lmcinnes/umap/issues/24` qui propose des solutions à ce problème.

français comme négatifs. Je vous laisse le soin de décider dans quelle mesure cela pourrait être inexact sur un échantillon plus large et plus représentatif.

Si vous remarquez quelque chose de ce genre, ne désespérez pas ! Ce genre de tendances est crucial à identifier *avant* de commencer à construire des modèles, car elles gonfleraient artificiellement l'exactitude des données d'entraînement, ce qui pourrait vous amener à mettre en production un modèle qui ne fonctionne pas bien.

La meilleure façon de traiter de tels échantillons biaisés est de collecter des données supplémentaires pour rendre votre jeu d'entraînement plus représentatif. Vous pouvez également essayer d'éliminer ces caractéristiques de vos données d'entraînement pour éviter de biaiser votre modèle, mais cela risque de ne pas être efficace dans la pratique, car les modèles détectent souvent un biais en exploitant les corrélations avec d'autres caractéristiques (voir le Chapitre 8).

Une fois que vous avez identifié certaines tendances, il est temps de les utiliser. Le plus souvent, vous pouvez le faire de deux façons, en créant une caractéristique qui matérialise cette tendance ou en utilisant un modèle qui en tirera facilement parti.

Laisser les données informer les caractéristiques et les modèles

Nous aimerions utiliser les tendances que nous découvrons dans les données pour éclairer notre stratégie de traitement des données, de génération de caractéristiques et de modélisation. Pour commencer, voyons comment nous pourrions générer des caractéristiques qui nous aideraient à saisir ces tendances.

Construire des caractéristiques à partir de modèles

Le ML consiste à utiliser des algorithmes d'apprentissage statistique pour exploiter les motifs trouvés dans les données, mais certains d'entre eux sont plus faciles à capturer pour les modèles que d'autres. Imaginez l'exemple trivial de la prédiction d'une valeur numérique en utilisant la valeur elle-même divisée par deux comme caractéristique. Le modèle devrait simplement apprendre à multiplier par deux pour prédire parfaitement la cible. D'un autre côté, prédire l'état de marchés boursiers à partir de données d'historique est un problème qui nécessite de faire appel à des modèles beaucoup plus complexes.

C'est pourquoi une grande partie des avantages pratiques du ML provient de la génération de *caractéristiques* supplémentaires qui aideront nos modèles à identifier des motifs utiles dans les données. La facilité avec laquelle un modèle identifie de tels motifs dépend de la façon dont nous représentons les données ainsi que de la quantité de données dont nous disposons. Plus vous avez de données et moins elles comportent de bruit, moins vous avez de travail d'ingénierie des caractéristiques à effectuer.

Cependant, il est souvent valable de commencer par générer des caractéristiques, tout d'abord parce que nous débutons généralement avec un petit ensemble de données, et ensuite parce que cela permet d'encoder nos croyances sur les données et de déboguer nos modèles.

La saisonnalité est une tendance courante qui bénéficie de la génération de caractéristiques spécifiques. Disons qu'un détaillant en ligne a remarqué que la plupart de ses ventes ont lieu les deux derniers week-ends du mois. Lorsqu'il élabore un modèle pour prévoir les ventes futures, il veut s'assurer qu'il a le potentiel pour saisir ce schéma.

Comme vous le verrez, selon la façon dont les dates sont représentées, la tâche pourrait s'avérer assez difficile pour les modèles. La plupart des modèles ne sont en effet capables de prendre que des entrées numériques (voir la section « Vectorisation », plus haut dans ce chapitre, pour les méthodes de transformation du texte et des images en entrées numériques). Examinons donc quelques façons de représenter les dates.

Date et heure brutes

La façon la plus simple de représenter le temps est l'heure Unix, qui représente « le nombre de secondes écoulées depuis 00:00:00 le jeudi 1er janvier 1970 »[31].

Bien que cette représentation soit simple, notre modèle devrait apprendre quelques schémas assez complexes pour identifier les deux derniers week-ends du mois. Le dernier week-end de 2020, par exemple (de 00:00:00 le 26 à 23:59:59 le 30 décembre), est représenté en temps Unix comme la plage allant de 1608937200 à 1609109999 (vous pouvez le vérifier si vous prenez la différence entre les deux nombres, qui représente un intervalle de 23 heures, 59 minutes et 59 secondes mesuré en secondes).

Rien dans cette fourchette ne permet de faire facilement le lien avec d'autres week-ends d'autres mois, de sorte qu'il sera assez difficile pour un modèle de séparer les week-ends pertinents des autres lorsqu'il utilisera le temps Unix en entrée. Nous pouvons faciliter la tâche d'un modèle en générant des caractéristiques.

Extraction du jour de la semaine et du jour du mois

Une façon de rendre notre représentation des dates plus claire serait d'extraire le jour de la semaine et le jour du mois dans deux attributs distincts.

La façon dont nous représenterions 23:59:59 le 27 décembre 2020, par exemple, utiliserait le même nombre que précédemment, et deux valeurs supplémentaires représentant le jour de la semaine (0 pour le dimanche, par exemple) et le jour du mois (27).

Cette représentation permettra à notre modèle d'apprendre plus facilement que les valeurs relatives aux week-ends (0 et 6 pour le dimanche et le samedi) et aux dates ultérieures du mois correspondent à une activité plus élevée.

31 Voir l'adresse `https://fr.wikipedia.org/wiki/Heure_Unix`.

Il est également important de noter que les représentations introduiront souvent un biais dans notre modèle. Par exemple, en encodant le jour de la semaine sous forme de nombre, l'encodage pour le vendredi (égal à cinq) sera cinq fois supérieur à celui du lundi (égal à un). Cette échelle numérique est un artefact de notre mode de calcul et ne représente pas quelque chose que nous souhaitons que notre modèle apprenne.

Croisement de caractéristiques

Si la représentation précédente facilite la tâche de nos modèles, il leur faudrait encore apprendre une relation complexe entre le jour de la semaine et le jour du mois : un trafic élevé ne se produit pas le week-end en début de mois ni les jours de semaine en fin de mois.

Certains modèles, tels que les réseaux de neurones profonds, exploitent des combinaisons non linéaires de caractéristiques et peuvent donc capter ces relations, mais ils ont souvent besoin d'une quantité importante de données. Une façon courante de résoudre ce problème consiste à faciliter encore plus la tâche et à introduire des *croisements de caractéristiques*.

Un croisement de caractéristiques est une caractéristique générée simplement en multipliant (croisant) deux ou plusieurs caractéristiques entre elles. L'introduction d'une combinaison non linéaire de caractéristiques permet à notre modèle d'effectuer plus facilement une discrimination basée sur une combinaison de valeurs de plusieurs caractéristiques.

Dans le Tableau 4.5, vous pouvez voir comment chacune des représentations que nous avons décrites se présenterait pour quelques exemples de points de données (en l'occurrence, pour la fin de l'année 2018).

Sur la Figure 4.13, vous pouvez voir comment ces valeurs de caractéristiques changent avec le temps, et lesquelles rendent plus facile pour un modèle de séparer des points de données spécifiques des autres.

Tableau 4.5 : Représenter vos données plus clairement facilitera fortement le bon fonctionnement de vos algorithmes.

Représentation humaine	Données brutes (datetime Unix)	Jour de la semaine (DoW)	Jour du mois (DoM)	Croisement (DoW / DoM)
Samedi, 29 décembre, 2018, 00:00:00	1546041600	7	29	174
Samedi, 29 décembre, 2018, 01:00:00	1546045200	7	29	174
...	...	...	...	...
Dimanche, 30 décembre, 2018, 23:59:59	1546214399	1	30	210

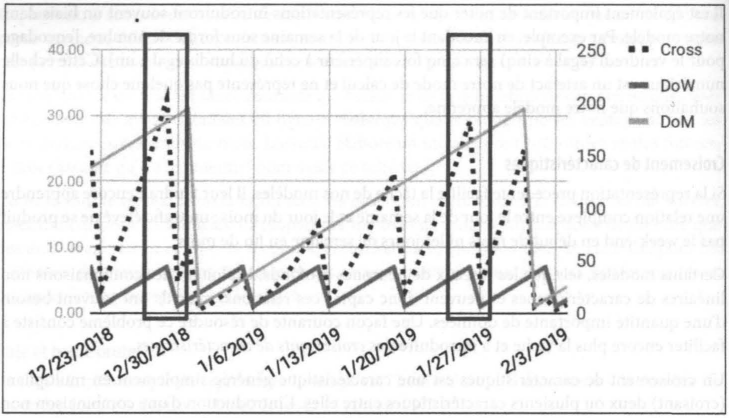


Figure 4.13 : Les derniers week-ends du mois sont plus faciles à séparer en utilisant des croisements de caractéristiques et des caractéristiques extraites.

Il existe une dernière façon de représenter nos données qui permettra à notre modèle d'apprendre encore plus facilement la valeur prédictive des deux derniers week-ends du mois.

Donner la réponse à votre modèle

Cela peut sembler être de la triche, mais si vous savez pertinemment qu'une certaine combinaison de valeurs de caractéristiques est particulièrement prédictive, vous pouvez créer une nouvelle caractéristique binaire qui est non nulle uniquement lorsque la combinaison pertinente de valeurs est vérifiée. Dans notre cas, cela signifierait ajouter une fonction appelée par exemple « deux_derniers_weekends », qui ne sera réglée à la valeur un que pendant les deux derniers week-ends du mois.

Si les deux derniers week-ends sont aussi prédictifs que nous l'avions supposé, le modèle apprendra simplement à exploiter cette caractéristique et sera beaucoup plus exact. Lorsque vous construisez des produits ML, n'hésitez jamais à faciliter la tâche de votre modèle. Il vaut mieux avoir un modèle qui fonctionne sur une tâche plus simple qu'un modèle qui se débat sur une tâche complexe.

La génération de caractéristiques est un domaine très vaste, et des méthodes existent pour la plupart des types de données. L'examen de chaque caractéristique qu'il est utile de générer pour différents types de données dépasse le cadre de cet ouvrage. Si vous souhaitez voir des exemples

et des méthodes plus pratiques, je vous recommande de consulter le livre *Feature Engineering for Machine Learning* (O'Reilly), d'Alice Zheng et Amanda Casari[32].

En général, la meilleure façon de générer des caractéristiques utiles est d'examiner vos données à l'aide des méthodes que nous avons décrites et de vous demander quelle est la façon la plus simple de les représenter de manière à ce que votre modèle en apprenne les schémas. Dans la section suivante, je vais décrire quelques exemples de caractéristiques que j'ai générées en utilisant ce processus pour l'Éditeur ML.

Caractéristiques pour l'Éditeur ML

Pour notre Éditeur ML, en utilisant les techniques décrites plus haut pour inspecter notre jeu de données (voir les détails dans le notebook d'exploration des données pour générer des caractéristiques, dans le dépôt GitHub de ce livre), nous avons généré les caractéristiques suivantes :

- Les verbes d'action (en anglais) tels que can et should sont prédictifs de la réponse à une question. Nous avons donc ajouté une valeur binaire qui vérifie s'ils sont présents dans chaque question.
- Les points d'interrogation sont également de bons prédicteurs, c'est pourquoi nous avons créé une caractéristique has_question.
- Les questions sur l'utilisation correcte de la langue anglaise tendaient généralement à ne pas obtenir de réponse, c'est pourquoi nous avons ajouté une caractéristique is_language_question.
- La longueur du texte de la question est un autre facteur, les questions très courtes ayant tendance à rester sans réponse. Cela a conduit à l'ajout d'une caractéristique de normalisation de la longueur des questions.
- Dans notre jeu de données, le titre de la question contient également des informations cruciales, et le fait de regarder les titres lors de l'étiquetage a rendu la tâche beaucoup plus facile. Cela a conduit à inclure le texte du titre dans tous les calculs de caractéristiques précédents.

Une fois que nous avons un premier ensemble de caractéristiques, nous pouvons commencer à construire un modèle. La construction de ce premier modèle est le sujet du chapitre suivant, le Chapitre 5.

Avant de passer aux modèles, j'ai voulu approfondir le sujet de la collecte et de la mise à jour d'un jeu de données. Pour ce faire, je me suis entretenu avec Robert Munro, un expert dans ce domaine.

32 Voir l'adresse `http://shop.oreilly.com/product/0636920049081.do`.

J'espère que vous apprécierez le résumé de notre discussion proposé dans la section suivante, et cela vous communiquera suffisamment d'enthousiasme pour aborder notre prochaine partie : la construction de notre premier modèle !

Robert Munro : Comment trouver, étiqueter et exploiter les données ?

Robert Munro a fondé plusieurs sociétés d'intelligence artificielle, constituant certaines des meilleures équipes dans ce domaine. Il a été directeur de la technologie chez Figure Eight, une entreprise leader dans le domaine de l'étiquetage des données pendant leur plus forte période de croissance. Avant cela, Robert a dirigé les produits des premiers services de traitement du langage naturel et de traduction automatique de AWS. Au cours de notre conversation, Robert partage quelques leçons qu'il a apprises en construisant des jeux de données pour le ML.

Q : Comment démarrer un projet de ML ?

R : La meilleure façon de procéder est de commencer par le problème de l'entreprise, car cela vous donnera des limites avec lesquelles travailler. Dans l'exemple de votre étude de cas sur l'Éditeur ML, est-ce que vous éditez un texte que quelqu'un d'autre a écrit après l'avoir soumis, ou est-ce que vous suggérez des modifications en direct au fur et à mesure que quelqu'un écrit ? Dans le premier cas, vous pourriez traiter les demandes par lots avec un modèle plus lent, tandis que dans le second, vous aurez besoin de quelque chose de plus rapide.

En termes de modèles, la seconde approche invaliderait les modèles de type séquence à séquence, car ils seraient trop lents. En outre, ces modèles ne fonctionnent pas aujourd'hui au-delà des recommandations au niveau des phrases et nécessitent beaucoup de texte en parallèle pour être entraînés. Une solution plus rapide consisterait à exploiter un classifieur et à utiliser les caractéristiques importantes qu'il extrait comme suggestions. Ce que vous attendez de ce modèle initial, c'est une mise en œuvre facile et des résultats auxquels vous pouvez faire confiance, en commençant par des bayésiens naïfs sur les caractéristiques des sacs de mots, par exemple.

Enfin, vous devez passer du temps à examiner certaines données et à les étiqueter vous-même. Cela vous donnera une idée de la difficulté du problème et des solutions qui pourraient convenir.

Q : De combien de données avez-vous besoin pour commencer ?

R : Lorsque vous collectez des données, vous cherchez à garantir que vous disposez d'un jeu représentatif et diversifié. Commencez par examiner les données dont vous disposez, et voyez si certains types de données sont sous-représentés afin de pouvoir en recueillir davantage. Le clustering sur votre jeu de données et la recherche de valeurs aberrantes peuvent vous aider à accélérer ce processus.

Pour l'étiquetage des données, dans le cas courant de la classification, nous avons constaté qu'un étiquetage de l'ordre de 1 000 échantillons de votre catégorie la plus rare fonctionne bien en pratique. Vous obtiendrez au moins un signal suffisant pour vous dire si vous devez continuer à utiliser votre approche actuelle de modélisation. À partir d'environ 10 000 échantillons, vous pouvez commencer à avoir confiance dans les modèles que vous construisez.

Au fur et à mesure que vous obtiendrez des données, l'exactitude de votre modèle s'accroîtra lentement, ce qui vous donnera une courbe montrant la façon dont vos performances évoluent avec ces données. À chaque instant, vous ne vous intéressez qu'à la dernière partie de la courbe, ce qui devrait vous donner une estimation de la valeur actuelle que vous donneront davantage de données. Dans la grande majorité des cas, l'amélioration que vous obtiendrez en étiquetant une plus grande quantité de données sera plus significative que si vous itériez sur le modèle.

Q : Quel processus utilisez-vous pour recueillir et étiqueter les données ?

R : Vous pouvez regarder votre meilleur modèle actuel et voir ce qui le fait trébucher. L'échantillonnage des incertitudes est une approche courante : identifiez les échantillons sur lesquels votre modèle est le plus incertain (ceux qui sont les plus proches de sa limite de décision), et trouvez des échantillons similaires de manière à les ajouter au jeu d'entraînement.

Vous pouvez également entraîner un « modèle d'erreur » pour trouver plus de données avec lesquelles votre modèle actuel se débat. Utilisez les erreurs que votre modèle commet comme étiquettes (en étiquetant chaque point de données comme étant « prédit correctement » ou « prédit incorrectement »). Une fois que vous avez entraîné un « modèle d'erreur » sur ces échantillons, vous pouvez l'utiliser sur vos données non étiquetées, et étiqueter les échantillons sur lesquels il prédit que votre modèle échouera.

D'un autre côté, vous pouvez également entraîner un « modèle d'étiquetage » pour trouver les meilleurs échantillons à étiqueter ensuite. Disons que vous avez un million d'échantillons, dont vous n'en avez étiqueté que 1 000. Vous pouvez créer un jeu d'entraînement avec 1 000 images étiquetées échantillonnées au hasard et 1 000 images non étiquetées, et entraîner un classifieur binaire pour prédire quelles images vous avez étiquetées. Vous pouvez ensuite utiliser ce « modèle d'étiquetage » pour identifier les points de données qui sont les plus différents de ceux que vous avez déjà étiquetés, et les étiqueter.

Q : Comment validez-vous le fait que vos modèles apprennent quelque chose d'utile ?

R : Un piège courant est de finir par concentrer les efforts d'étiquetage sur une petite partie du jeu contenant des données pertinentes. Il se peut que votre modèle se débatte avec des articles qui traitent de basket-ball. Si vous continuez à annoter encore plus d'articles sur le basket, votre modèle peut devenir excellent en ce qui concerne ce sport, mais mauvais pour tout le reste. C'est pourquoi, même si vous devriez utiliser des stratégies pour recueillir des données, vous devez toujours procéder à un échantillonnage aléatoire de votre jeu de test pour valider votre modèle.

Enfin, la meilleure façon de le faire est de suivre la dérive des performances de votre modèle déployé. Vous pourriez surveiller l'incertitude du modèle ou, idéalement, le ramener aux métriques associées au produit : vos métriques d'utilisation diminuent-elles progressivement ? Cela peut être dû à d'autres facteurs, mais c'est un bon signal de déclenchement pour investiguer, et éventuellement mettre à jour votre jeu de données d'entraînement.

Conclusion

Dans ce chapitre, nous avons abordé des conseils importants pour examiner un jeu de données de manière efficace et efficiente.

Nous avons commencé par examiner la qualité des données et la manière de décider si elles sont suffisantes pour nos besoins. Ensuite, nous avons abordé la meilleure façon de se familiariser avec le type de données dont vous disposez : en commençant par des statistiques sommaires, puis en passant à des clusters de points similaires pour identifier les grandes tendances.

Nous avons ensuite expliqué pourquoi il est utile de consacrer un temps important à l'étiquetage des données afin d'identifier les tendances que nous pouvons ensuite exploiter pour créer des caractéristiques utiles. Enfin, nous avons profité de l'expérience de Robert Munro, qui a aidé de nombreuses équipes à construire des jeux de données de pointe pour le ML.

Maintenant que nous avons examiné un jeu de données et généré des caractéristiques que nous espérons être prédictives, nous sommes prêts à construire notre premier modèle, ce que nous ferons au Chapitre 5.

PARTIE III

Itérer sur les modèles

La première partie portait sur les meilleures pratiques pour mettre en place un projet de ML et en suivre l'évolution. Dans la deuxième partie, nous avons vu l'intérêt de construire un pipeline de bout en bout aussi rapidement que possible, tout en explorant un jeu de données initiales.

En raison de sa nature expérimentale, le ML est pour une bonne partie un processus itératif. Vous devriez prévoir d'itérer de manière répétée sur des modèles et des données, en suivant une boucle expérimentale telle que l'illustre la Figure III.1.

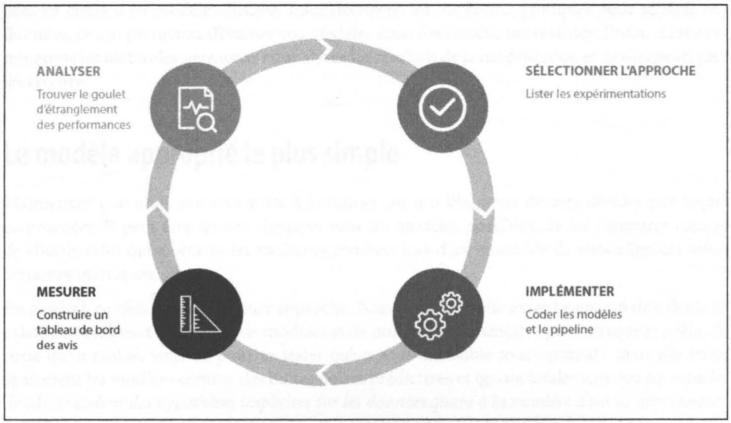


Figure III.1 : La boucle du ML.

Cette troisième partie décrira une itération de la boucle. Lorsque vous travaillez sur des projets de ML, vous devriez prévoir de passer par de multiples itérations de ce type avant d'espérer atteindre des performances satisfaisantes. Voici un aperçu des chapitres de cette partie du livre :

Chapitre 5

Dans ce chapitre, nous allons entraîner un premier modèle et l'évaluer. Ensuite, nous analyserons en profondeur ses performances et nous identifierons les moyens de l'améliorer.

Chapitre 6

Ce chapitre traite des techniques permettant de construire et de déboguer rapidement les modèles et d'éviter les erreurs qui font perdre du temps.

Chapitre 7

Dans ce chapitre, nous utiliserons l'Éditeur ML comme étude de cas pour montrer comment tirer parti d'un classifieur entraîné pour fournir des suggestions aux utilisateurs et pour construire un modèle de suggestion pleinement fonctionnel.

CHAPITRE 5

Entraîner et évaluer votre modèle

Dans les chapitres précédents, nous avons abordé la manière d'identifier le problème à résoudre, d'élaborer un plan pour y faire face, de construire un pipeline simple, d'explorer un jeu de données et de générer un ensemble initial de caractéristiques. Ces étapes nous ont permis de rassembler suffisamment d'informations pour commencer à entraîner un modèle adéquat. Un modèle « adéquat » signifie ici un modèle qui correspond bien à la tâche à accomplir et qui a de bonnes chances de bien fonctionner.

Dans ce chapitre, nous commencerons par passer brièvement en revue certaines préoccupations liées au choix d'un modèle. Ensuite, nous décrirons les meilleures pratiques pour séparer vos données, ce qui permettra d'évaluer vos modèles dans des conditions réalistes. Enfin, nous examinerons les méthodes permettant d'analyser les résultats de la modélisation et de diagnostiquer les erreurs.

Le modèle approprié le plus simple

Maintenant que nous sommes prêts à entraîner un modèle, nous devons décider par lequel commencer. Il peut être tentant d'essayer tous les modèles possibles, de les comparer tous et de choisir celui qui a obtenu les meilleurs résultats lors d'un ensemble de tests effectués selon certaines métriques.

En général, ce n'est pas la meilleure approche. Non seulement elle exige beaucoup de calculs (il existe de nombreux ensembles de modèles et de nombreux paramètres pour chaque modèle, de sorte qu'en réalité, vous ne pourrez tester qu'un sous-ensemble sous-optimal), mais elle traite également les modèles comme des boîtes noires prédictives et ignore totalement que *les modèles de ML encodent des hypothèses implicites sur les données quant à la manière dont ils apprennent.*

Différents modèles font des hypothèses différentes sur les données, et sont donc adaptés à des tâches différentes. En outre, comme le ML est un domaine itératif, vous voudrez choisir des modèles que vous pourrez construire et évaluer rapidement.

Commençons par voir comment identifier des modèles simples. Ensuite, nous aborderons quelques exemples de motifs dans les données et de modèles appropriés pour les exploiter.

Modèles simples

Un modèle simple doit être rapide à implémenter, compréhensible et déployable : rapide à implémenter parce que votre premier modèle ne sera probablement pas le dernier, compréhensible parce qu'il vous permettra de le déboguer plus facilement, et déployable parce que c'est une exigence fondamentale pour une application utilisant le ML. Commençons par explorer ce que j'entends par « rapide à implémenter ».

Rapidité d'implémentation

Choisissez un modèle que vous pourrez implémenter facilement. En général, cela signifie qu'il faut choisir un modèle bien compris, sur lequel de nombreux tutoriels ont été écrits et pour lequel des gens pourront vous aider (surtout si vous posez des questions bien formulées à l'aide de notre Éditeur ML !). Pour une nouvelle application basée sur le ML, vous aurez suffisamment de défis à relever en termes de traitement des données et de déploiement d'un résultat fiable. Vous devriez donc d'abord faire de votre mieux pour éviter tous les maux de tête liés aux modèles.

Si possible, commencez par utiliser des modèles provenant de bibliothèques populaires comme Keras ou scikit-learn, et attendez avant de vous plonger dans un dépôt GitHub expérimental qui n'a pas de documentation et n'a pas été mis à jour au cours des neuf derniers mois.

Une fois votre modèle implémenté, vous voudrez l'inspecter et comprendre comment il exploite votre jeu de données. Pour ce faire, vous avez besoin d'un modèle qui soit compréhensible.

Compréhensibilité

L'*explicabilité* et l'*interprétabilité* du modèle décrivent la capacité d'un modèle à exposer les raisons (telles qu'une combinaison donnée de prédicteurs) qui l'ont amené à faire des prédictions. L'explicabilité peut être utile pour diverses raisons, comme vérifier que nos modèles ne sont pas biaisés de manière indésirable ou expliquer à un utilisateur ce qu'il pourrait faire pour améliorer les résultats des prédictions. Elle facilite également l'itération et le débogage.

Si vous pouvez extraire les caractéristiques sur lesquelles un modèle s'appuie pour prendre des décisions, vous aurez une vision plus claire des caractéristiques à ajouter, à modifier ou à supprimer, ou du genre de modèle qui pourrait faire de meilleurs choix.

Malheureusement, l'interprétation des modèles est souvent complexe, même pour les modèles simples, et parfois difficile pour les plus grands. Dans la section « Évaluer l'importance des caractéristiques », plus loin dans ce chapitre, nous verrons comment relever ce défi et nous vous aiderons à identifier les points d'amélioration de votre modèle. Nous utiliserons entre autres des systèmes d'exploration sous forme de boîtes noires qui tentent de fournir des explications sur les prédictions d'un modèle, quel que soit son fonctionnement interne.

Les modèles plus simples, tels que la régression logistique ou les arbres de décision, sont généralement plus faciles à expliquer car ils donnent une certaine mesure de l'importance des caractéristiques, ce qui est une autre raison pour laquelle ils sont généralement de bons modèles à essayer en premier.

Déploiement

Pour rappel, l'objectif final de votre modèle est de fournir un service précieux aux personnes qui l'utiliseront. Cela signifie que lorsque vous réfléchissez au modèle à entraîner, vous devez toujours vous demander si vous serez en mesure de le déployer.

Nous aborderons cette question de déploiement dans la quatrième partie du livre, mais vous devriez déjà réfléchir à des questions telles que les suivantes :

- Combien de temps faut-il à un modèle entraîné pour faire une prédiction pour un utilisateur ? Lorsque vous pensez à la latence des prédictions, vous devez inclure non seulement le temps nécessaire à un modèle pour produire un résultat, mais aussi le délai entre le moment où un utilisateur soumet une demande de prédiction et celui où il reçoit le résultat. Cela inclut toutes les étapes de prétraitement, telles que la génération de caractéristiques, les appels réseau et les étapes de post-traitement qui se produisent entre la sortie d'un modèle et les données présentées à un utilisateur.
- Ce pipeline d'inférence est-il assez rapide pour notre cas d'utilisation si nous prenons en compte le nombre d'utilisateurs simultanés que nous attendons ?
- Combien de temps faut-il pour entraîner le modèle, et à quelle fréquence faut-il le faire ? Si l'entraînement de votre modèle dure 12 heures et que vous deviez le relancer toutes les 4 heures pour qu'il soit à jour, non seulement vos factures risquent d'être élevées, mais en plus votre modèle sera toujours dépassé.

Nous pouvons comparer la simplicité des modèles en utilisant un tableau tel que celui de la Figure 5.1. À mesure que le domaine du ML évolue et que de nouveaux outils sont construits, des modèles qui peuvent être complexes à déployer ou difficiles à interpréter aujourd'hui seront peut-être plus simples à utiliser à l'avenir, et ce tableau devra alors être mis à jour. Pour cette raison, je vous suggère d'en construire votre propre version en fonction de votre domaine de problème particulier.

Nom du modèle	Facilité d'implémentation		Compréhensibilité		Déploiement		« Score de simplicité » total
	Modèle bien compris	Implémentation contrôlée	Facilité d'extraction de l'importance des caractéristiques	Facilité de débogage	Durée d'inférence	Durée d'entraînement	
Arbre de décision (de scikit-learn)	5/5	5/5	4/5	4/5	5/5	5/5	28/30
CNN CNN (de Keras)	4/5	5/5	3/5	3/5	3/5	2/5	20/30
Transformation (depuis un dépôt github personnel)	2/5	1/5	0/5	0/5	2/5	1/5	6/30

Figure 5.1 : Notation de modèles en fonction de leur simplicité.

Même parmi les modèles qui sont simples, interprétables et déployables, il y a encore de nombreux candidats potentiels. Pour faire votre choix, vous devez également tenir compte des modèles que vous avez identifiés au Chapitre 4.

Des motifs détectés dans les données aux modèles

Les motifs, ou schémas, que nous avons identifiés et les caractéristiques que nous avons générées devraient guider notre choix de modèle. Voyons quelques exemples de schémas dans les données et les modèles appropriés pour les exploiter.

Nous voulons ignorer l'échelle des caractéristiques

De nombreux modèles tirent davantage parti des plus grandes caractéristiques que des plus petites. Cela peut être bien dans certains cas, mais indésirable dans d'autres. Pour les modèles utilisant des procédures d'optimisation telles que la descente de gradient comme les réseaux de neurones, les différences d'échelle des caractéristiques peuvent parfois conduire à une instabilité dans la procédure d'apprentissage.

Si vous souhaitez utiliser à la fois l'âge en années (de un à cent ans) et le revenu en dollars (disons que nos données atteignent jusqu'à neuf chiffres) comme constituant deux prédicteurs, vous devez vous assurer que votre modèle est capable de tirer parti des caractéristiques les plus prédictives, quelle que soit leur échelle.

Vous pouvez vous en assurer en prétraitant les caractéristiques pour normaliser leur échelle afin d'avoir une moyenne nulle et une variance unitaire. Si toutes les caractéristiques sont norma-

lisées selon la même échelle, un modèle considérera chacune d'entre elles de manière égale (au moins au début).

Une autre solution consiste à se tourner vers des modèles qui ne sont pas affectés par les différences d'échelle des caractéristiques. Les exemples pratiques les plus courants sont les arbres de décision, les forêts aléatoires et les arbres de décision avec boosting de gradient. XGBoost (`https://xgboost.readthedocs.io/en/latest/`) est une implémentation d'arbres avec boosting de gradient couramment utilisée en production en raison de sa robustesse, ainsi que de sa rapidité.

Notre variable prédite est une combinaison linéaire de prédicteurs

Parfois, il y a de bonnes raisons de croire que nous pouvons faire de bonnes prédictions en utilisant seulement une combinaison linéaire de nos caractéristiques. Dans ces cas, nous devrions utiliser un modèle linéaire, tel qu'une régression linéaire pour les problèmes continus ou une régression logistique ou un classifieur bayésien naïf pour les problèmes de classification.

Ces modèles sont simples, efficaces et permettent souvent une interprétation directe de leurs poids qui peut nous aider à identifier des caractéristiques importantes. Si nous pensons que les relations entre nos caractéristiques et notre variable prédite sont plus complexes, l'utilisation d'un modèle non linéaire tel qu'un réseau de neurones multicouche ou la génération de croisements de caractéristiques peut aider (voir le début de la section « Laisser les données informer les caractéristiques et les modèles » du Chapitre 4).

Nos données ont un aspect temporel

Si nous avons affaire à des séries chronologiques de points de données dont la valeur à un certain moment dépend de valeurs antérieures, nous voudrions exploiter des modèles qui encodent explicitement ces informations. À titre d'exemples, on peut citer les modèles statistiques tels que la moyenne mobile intégrée autorégressive (ARIMA) ou les réseaux de neurones récurrents (RNN).

Chaque point de données est une combinaison de motifs

Pour résoudre les problèmes dans le domaine de l'imagerie, par exemple, les réseaux de neurones convolutifs (CNN) se sont avérés utiles grâce à leur capacité à apprendre des *filtres invariants par translation*. Cela signifie qu'ils sont capables d'extraire des motifs locaux dans une image, quelle que soit leur position. Une fois qu'un CNN apprend à détecter un œil, il peut le détecter n'importe où dans une image, et pas seulement aux endroits où il est apparu dans le jeu d'entraînement.

Les filtres de convolution se sont avérés utiles dans d'autres domaines contenant des schémas locaux, comme la reconnaissance vocale ou la classification de textes, les CNN y ayant été utili-

sés avec succès pour la classification de phrases. Pour un exemple, voir par exemple l'implémentation réalisée par Yoon Kim dans le document « *Convolutional Neural Networks for Sentence Classification* »[33].

Il y a de nombreux autres points à prendre en compte pour réfléchir aux bons modèles à utiliser. Pour la plupart des problèmes classiques de ML, je recommande d'utiliser l'organigramme pratique que l'équipe de scikit-learn fournit utilement[34]. Il fournit des suggestions de modèles pour de nombreux cas d'utilisation courants.

Modèle pour l'Éditeur ML

Pour l'Éditeur ML, nous aimerions que notre premier modèle soit rapide et raisonnablement facile à déboguer. En outre, nos données sont constituées d'échantillons individuels, sans qu'il soit nécessaire de prendre en compte un aspect temporel (comme une série de questions, par exemple). C'est pourquoi nous commencerons par une base de référence populaire et résiliente, en l'occurrence une forêt aléatoire.

Une fois que vous avez identifié un modèle qui semble raisonnable, il est temps de l'entraîner. En règle générale, vous ne devez pas entraîner votre modèle sur la totalité du jeu de données que vous avez collecté au Chapitre 4. Vous devrez commencer par présenter certaines données de votre jeu d'entraînement. Voyons pourquoi et comment vous devriez le faire.

Partager votre jeu de données

Le but principal de notre modèle est de fournir des prédictions valables pour les données que nos utilisateurs soumettront. Cela signifie que notre modèle devra éventuellement être performant sur des données qu'il n'a jamais vues auparavant.

Lorsque vous entraînez un modèle sur un jeu de données, la mesure de ses performances sur le même jeu vous indique seulement sa capacité à faire des prédictions sur des données qu'il a déjà vues. Si vous entraînez un modèle uniquement sur un sous-ensemble de vos données, vous pouvez alors utiliser la partie du jeu sur lequel le modèle n'a pas été entraîné pour estimer ses performances sur des données qui n'ont pas encore été vues.

Sur la Figure 5.2, vous pouvez voir un exemple de division en trois jeux distincts (entraînement, validation et test) basée sur un attribut de notre jeu de données (l'auteur d'une question). Dans ce chapitre, nous verrons ce que signifie chacun de ces jeux et comment y réfléchir.

33 Voir l'adresse `https://arxiv.org/abs/1408.5882`.
34 Voir l'adresse `https://bit.ly/3eYcdkl`.

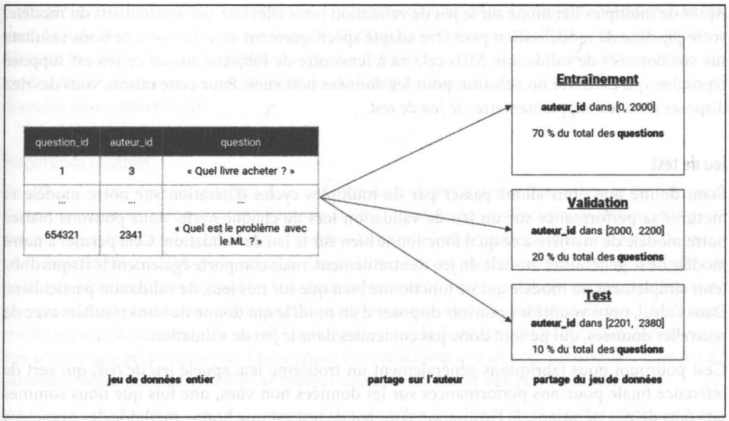


Figure 5.2 : Fractionnement des données sur l'auteur tout en attribuant la bonne proportion de questions à chaque partage.

Le premier cas à prendre en considération est celui du jeu de validation.

Jeu de validation

Pour estimer les performances de notre modèle sur des données non vues, nous retirons délibérément une partie de notre jeu d'entraînement, puis nous utilisons les performances de ce nouveau jeu comme une approximation des performances de notre modèle en production. Ce jeu de données nous permet de valider le fait que notre modèle peut se généraliser à des données qui n'ont pas encore été vues. On l'appelle donc généralement un *jeu de validation*.

Vous pouvez choisir différentes sections de vos données à conserver comme jeu de validation pour évaluer votre modèle et l'entraîner sur les données restantes. Effectuer de multiples cycles de ce processus permet de contrôler tout écart dû à un choix particulier du jeu de validation. C'est ce que l'on s'appelle une *validation croisée*.

À mesure que vous modifiez votre stratégie de prétraitement des données et le type de modèle que vous utilisez ou ses hyperparamètres, les performances de votre modèle sur le jeu de validation changeront (et idéalement s'amélioreront). L'utilisation du jeu de validation vous permet d'ajuster les hyperparamètres, de la même manière que l'utilisation d'un jeu d'entraînement permet à un modèle d'ajuster ses paramètres.

Après de multiples itérations sur le jeu de validation pour effectuer des ajustements du modèle, votre pipeline de modélisation peut être adapté spécifiquement afin d'obtenir de bons résultats sur vos données de validation. Mais cela va à l'encontre de l'objectif auquel ce jeu est supposé répondre, qui est d'être un substitut pour les données non vues. Pour cette raison, vous devriez disposer d'un jeu supplémentaire : le *jeu de test*.

Jeu de test

Étant donné que nous allons passer par de multiples cycles d'itération sur notre modèle et mesurer sa performance sur un jeu de validation lors de chaque cycle, nous pouvons biaiser notre modèle de manière à ce qu'il fonctionne bien sur le jeu de validation. Cela permet à notre modèle de se généraliser au-delà du jeu d'entraînement, mais comporte également le risque d'obtenir simplement un modèle qui ne fonctionne bien que sur nos jeux de validation particuliers. Dans l'idéal, nous voudrions pouvoir disposer d'un modèle qui donne de bons résultats avec de nouvelles données, qui ne sont donc pas contenues dans le jeu de validation.

C'est pourquoi nous fabriquons généralement un troisième jeu, appelé *jeu de test*, qui sert de référence finale pour nos performances sur les données non vues, une fois que nous sommes satisfaits de nos itérations. Si l'utilisation d'un jeu de test est une bonne méthode, les praticiens utilisent parfois le jeu de validation comme jeu de test. Cela augmente le risque de créer des biais dans le modèle, mais peut être approprié lorsque l'on ne réalise que quelques expérimentations.

Il est important d'éviter d'utiliser les performances du jeu de test pour éclairer les décisions de modélisation, car ce jeu est censé représenter les données non vues auxquelles nous serons confrontés en production. Adapter une approche de modélisation pour obtenir de bonnes performances sur le jeu de test risque donc d'entraîner une surestimation des performances réelles du modèle.

Pour qu'un modèle soit performant en production, les données sur lesquelles vous effectuez l'entraînement devraient ressembler à celles produites par les utilisateurs qui interagiront avec votre produit. Dans l'idéal, tous les types de données que vous pourriez recevoir des utilisateurs devraient être représentés dans votre jeu de données. Si ce n'est pas le cas, alors gardez présent à l'esprit le fait que les performances de votre jeu de données ne sont indicatives que pour un sous-ensemble seulement de vos utilisateurs.

Pour notre Éditeur ML, cela signifie que les utilisateurs qui ne rentrent pas dans le cadre des données démographiques de *writers.stackoverflow.com* pourraient ne pas être aussi bien servis par nos recommandations. Si nous voulions résoudre ce problème, nous devrions élargir le jeu de données pour y inclure des questions plus représentatives de ces utilisateurs. Nous pourrions commencer par intégrer des questions provenant d'autres sites Web de Stack Exchange afin de couvrir un ensemble plus large de sujets, ou plus largement de différents sites Web de questions et réponses.

Corriger un jeu de données de cette manière peut être difficile pour un projet secondaire. Cependant, lors de la construction de produits destinés aux consommateurs, il est nécessaire d'aider à détecter les faiblesses du modèle avant que les utilisateurs n'y soient exposés. Bon nombre des modes de défaillance que nous aborderons au Chapitre 8 auraient pu être évités avec un jeu de données plus représentatif.

Proportions relatives

En général, vous devriez maximiser la quantité de données à partir desquelles le modèle peut s'entraîner, tout en conservant des jeux de validation et de test suffisamment importants pour fournir des métriques de performance exactes. Les praticiens utilisent souvent 70 % des données pour l'entraînement, 20 % pour la validation et 10 % pour les tests, mais cela dépend entièrement de la quantité de données. Pour les très grands jeux, vous pouvez vous permettre d'utiliser une plus grande proportion des données pour l'entraînement, tout en disposant d'assez de données restantes pour la validation des modèles. Pour des jeux de données plus petits, vous devrez peut-être utiliser une proportion plus limitée pour l'entraînement afin de pouvoir disposer d'un jeu de validation suffisamment grand pour fournir une mesure exacte des performances.

Vous savez maintenant pourquoi vous voulez partager les données, et quelles divisions envisager. Mais comment décider quel point de données devrait être pris en compte dans chaque fractionnement ? Les méthodes de partage que vous utilisez ont un impact significatif sur les performances de la modélisation et devraient dépendre des caractéristiques particulières de votre jeu de données.

Fuite de données

La méthode que vous utilisez pour partager vos données est un élément crucial de la validation. Vous devez vous efforcer de faire en sorte que votre jeu de validation/test soit proche de ce que vous attendez de la part des données non encore vues.

Le plus souvent, les jeux d'entraînement, de validation et de test sont séparés par des points de données échantillonnés de manière aléatoire. Dans certains cas, cela peut entraîner des *fuites de données*. Une fuite de données se produit lorsque (en raison de notre procédure d'entraînement) un modèle reçoit pendant l'entraînement des informations auxquelles il n'aura pas accès lorsqu'il sera utilisé en production devant de vrais utilisateurs.

Les fuites de données devraient être évitées à tout prix, car elles conduisent à une vision exagérée des performances de notre modèle. Un modèle entraîné sur un jeu présentant des fuites de données est capable d'exploiter des informations pour faire des prédictions, alors qu'il n'en sera pas capable lorsqu'il rencontrera des données différentes. Cela facilite artificiellement la tâche du modèle, mais uniquement en raison des fuites d'informations. La performance du modèle semble élevée pour les données du jeu, mais elle sera bien pire en production.

Sur la Figure 5.3, j'ai dégagé quelques causes courantes où le fractionnement aléatoire de votre jeu entraînera des fuites de données. Les causes potentielles des fuites de données sont nombreuses, et nous allons en explorer dans ce qui suit deux qui sont fréquentes.

Pour commencer notre exploration, abordons l'exemple en haut de la Figure 5.3, la fuite de données temporelles. Ensuite, nous passerons à la contamination des échantillons, une catégorie qui englobe les deux exemples du bas de la Figure 5.3.

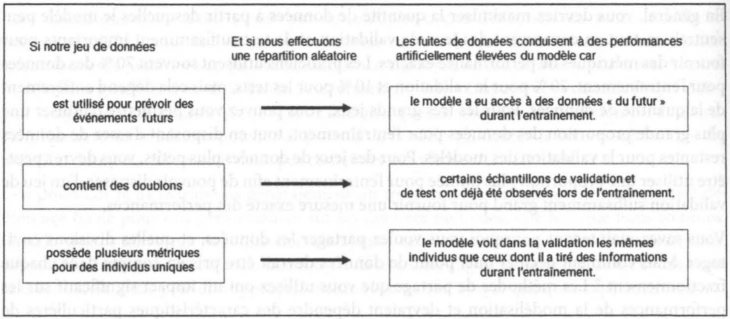


Figure 5.3 : Un fractionnement aléatoire peut souvent entraîner des fuites de données.

Fuite de données temporelles. En matière de prévision pour des séries chronologiques, un modèle doit tirer des enseignements des points de données du passé pour prévoir des événements qui ne se sont pas encore produits. Si nous effectuons une répartition aléatoire sur un jeu de données devant servir à effectuer des prévisions, nous introduirons une fuite de données : un modèle qui est entraîné sur un ensemble de points aléatoires et qui est évalué sur le reste aura accès à des données d'entraînement qui se produisent *après* les événements qu'il essaie de prédire.

Le modèle fonctionnera artificiellement bien sur les jeux de validation et de test, mais échouera en production, car tout ce qu'il a appris est de tirer parti des informations futures, qui ne sont pas disponibles dans le monde réel.

Une fois que vous en êtes conscient, une fuite de données temporelles est généralement facile à détecter. D'autres types de fuites de données peuvent donner à un modèle accès à des informations qu'il ne devrait pas connaître pendant l'entraînement, et gonfler ainsi artificiellement ses performances en « contaminant » ses données d'origine. Ces fuites peuvent souvent être beaucoup plus difficiles à détecter.

Contamination des échantillons. Une source courante de fuite de données réside dans le niveau auquel le caractère aléatoire se produit. Lors de la construction d'un modèle pour prédire la note que les élèves recevraient dans leurs dissertations, un spécialiste des données que j'assistais a découvert que son modèle fonctionnait presque parfaitement sur un jeu de test avec des données extérieures.

Pour une tâche aussi difficile, un modèle qui fonctionne aussi bien devrait être examiné de près car cela indique fréquemment la présence d'un bogue ou d'une *fuite de données*. Certains diraient que l'équivalent en ML de la loi de Murphy est que plus vous êtes agréablement surpris par la performance de votre modèle sur vos données de test, et plus vous risquez d'avoir une erreur dans votre pipeline.

Dans cet exemple, comme la plupart des étudiants avaient rédigé plusieurs essais, le fractionnement des données a conduit à ce que les travaux des mêmes étudiants soient présents à la fois dans les jeux d'entraînement et dans les jeux de test. Cela a permis au modèle de récupérer les caractéristiques qui identifiaient les étudiants et d'utiliser ces informations pour faire des prédictions précises (les étudiants de ce jeu de données avaient tendance à avoir des notes similaires dans tous leurs travaux).

Si nous devions déployer ce prédicteur de notes pour un usage futur, il ne serait pas en mesure de proposer des notes utiles pour des étudiants qu'il n'a pas vus auparavant, et il se contenterait de prédire des notes historiques pour les étudiants dont les dissertations ont servi à son entraînement. Cela ne serait absolument pas utile.

Pour résoudre la fuite de données de cet exemple, une nouvelle répartition a été faite au niveau des étudiants plutôt qu'au niveau des dissertations. Cela signifie que chaque étudiant apparaît soit uniquement dans le jeu d'entraînement, soit uniquement dans le jeu de validation. Comme la tâche est devenue beaucoup plus difficile, cela a entraîné une diminution de l'exactitude du modèle. Cependant, comme le processus d'entraînement était désormais beaucoup plus proche de ce qu'il serait en production, ce nouveau modèle était beaucoup plus valable.

La contamination des échantillons peut se produire de manière nuancée dans des tâches courantes. Prenons l'exemple d'un site Web de réservation de location d'appartements. Ce site intègre un modèle de prédiction des clics qui, en fonction d'une requête de l'utilisateur et d'un certain article, prédit si l'utilisateur cliquera sur cet article. Ce modèle est utilisé pour décider quelles listes doivent être affichées aux utilisateurs.

Pour entraîner un tel modèle, ce site Web pourrait utiliser un jeu de données sur les caractéristiques des utilisateurs telles que leur nombre de réservations précédentes, jumelées avec les appartements qui leur ont été présentés et s'ils ont cliqué dessus. Ces données sont généralement stockées dans une base de données de production qui peut être interrogée pour produire de telles paires d'informations. Si les ingénieurs de ce site Web devaient simplement interroger la base de données pour constituer un tel jeu de données, ils seraient probablement confrontés à un cas de fuite de données. Pouvez-vous comprendre pourquoi ?

Sur la Figure 5.4, j'ai esquissé une illustration de ce qui peut mal tourner en décrivant une prédiction pour un utilisateur spécifique. En haut, vous pouvez voir les caractéristiques qu'un modèle pourrait utiliser en production pour fournir une prédiction de clic. Ici, un nouvel utilisateur sans réservation préalable se voit présenter un certain appartement. En bas, vous pouvez voir l'état des caractéristiques quelques jours plus tard lorsque les ingénieurs extraient les données de la base de données.

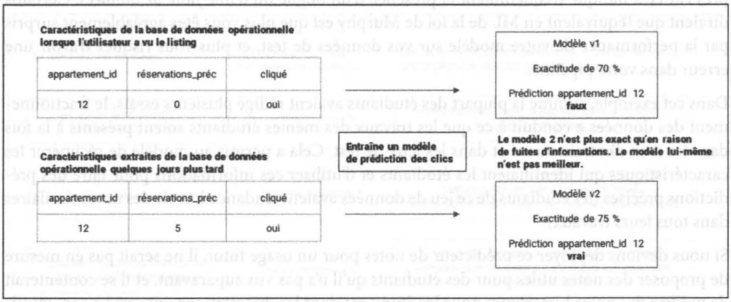


Figure 5.4 : Les fuites de données peuvent se produire pour des raisons subtiles, telles que l'absence de version dans les données.

Remarquez la différence dans `réservations_préc`, qui est due à l'activité des utilisateurs *après* que la liste leur a été présentée initialement. En utilisant un instantané d'une base de données, des informations sur les actions futures de l'utilisateur ont été introduites dans le jeu d'entraînement. Nous savons maintenant que l'utilisateur va finalement réserver cinq appartements ! Une telle fuite peut amener un modèle entraîné avec les informations du bas de la figure à produire une prédiction correcte à partir de données d'entraînement incorrectes. L'exactitude du modèle sur le jeu de données généré sera élevée, car il exploitera alors des données auxquelles il n'aura pas accès en production. Lorsque le modèle sera déployé, ses performances seront plus mauvaises que prévu.

Si vous devez retirer quelque chose de cette anecdote, c'est de toujours étudier les résultats d'un modèle, surtout s'il montre des performances étonnamment bonnes.

Partage des données pour l'Éditeur ML

Le jeu de données que nous utilisons pour entraîner notre Éditeur ML contient des questions posées sur Stack Overflow, ainsi que leurs réponses. À première vue, une répartition aléatoire

peut sembler suffisante et est assez simple à mettre en œuvre dans scikit-learn. Nous pourrions, par exemple, écrire une fonction comme celle qui est présentée ici :

```
from sklearn.model_selection import train_test_split

def get_random_train_test_split(posts, test_size=0.3, random_state=40):
    """
    Obtient le partage entraînement/test à partir du DataFrame
    Suppose que le DataFrame comporte une ligne par échantillon de question
    :paramètre posts: tous les posts, avec leurs étiquettes
    :paramètre test_size: la proportion à allouer au test
    :paramètre random_state: une semence aléatoire
    """
    return train_test_split(
        posts, test_size=test_size, random_state=random_state
    )
```

Une telle approche peut entraîner des fuites ; pouvez-vous les identifier ?

Si nous repensons à notre cas d'application, nous savons que nous aimerions que notre modèle travaille sur des questions qu'il n'a jamais vues auparavant, en ne regardant que leur contenu. Sur un site Web de questions-réponses, cependant, de nombreux autres facteurs peuvent jouer un rôle dans le succès de la réponse à une question. L'un de ces facteurs est l'identité de l'auteur.

Si nous divisons nos données de manière aléatoire, un auteur donné pourrait apparaître à la fois dans nos jeux d'entraînement et de validation. Si certains auteurs populaires ont un style particulier, notre modèle pourrait s'adapter à ce style et atteindre des performances artificiellement élevées dans notre jeu de validation en raison de fuites de données. Pour éviter cela, il serait plus sûr pour nous de veiller à ce que chaque auteur n'apparaisse que lors de l'entraînement ou que lors de la validation. C'est le même type de fuite que nous avons décrit un peu plus tôt dans l'exemple de notation des étudiants.

En utilisant la classe `GroupShuffleSplit` de scikit-learn et en passant la caractéristique représentant l'identifiant unique d'un auteur à sa méthode `split`, nous pouvons garantir qu'un auteur donné n'apparaît que dans un seul des fractionnements.

```
from sklearn.model_selection import GroupShuffleSplit

def get_split_by_author(
    posts, author_id_column="OwnerUserId", test_size=0.3, random_state=40
    ):
    """
    Obtient le partage entraînement/test
    Garantit que chaque auteur n'apparaît que dans un des fractionnements
    : paramètre posts: tous les posts, avec leurs étiquettes
    :paramètre author_id_column: nom de la colonne contenant author_id
    :paramètre test_size: la proportion à allouer au test
    :paramètre random_state: une semence aléatoire
    """
```

```
    splitter = GroupShuffleSplit(
        n_splits=1, test_size=test_size, random_state=random_state
    )
    splits = splitter.split(posts, groups=posts[author_id_column])
    return next(splits)
```

Pour voir une comparaison entre les deux méthodes de fractionnement, consultez le notebook correspondant dans le dépôt GitHub de ce livre (voir `splitting_data.ipynb`).

Une fois les données partagées, un modèle peut être ajusté au jeu d'entraînement. Nous avons abordé les parties nécessaires à un pipeline d'entraînement dans la section « Commencer par un pipeline simple » du Chapitre 2. Pour l'entraînement d'un notebook de modèles simples dans le dépôt GitHub de ce livre, je montre un exemple de pipeline d'entraînement de bout en bout pour l'Éditeur ML (voir `train_simple_model.ipynb`). Nous allons analyser les résultats de ce pipeline.

Nous avons couvert les principaux risques que nous voulons garder à l'esprit lors du partage des données, mais que devons-nous faire une fois que notre jeu de données est divisé et que nous avons entraîné un modèle sur les données ainsi formées ? Dans la prochaine section, nous traiterons des différentes manières pratiques d'évaluer les modèles entraînés et de la meilleure façon de les exploiter.

Juger des performances

Maintenant que nous avons partagé nos données, nous pouvons entraîner notre modèle et juger de ses performances. La plupart des modèles sont entraînés pour minimiser une fonction de coût, qui représente la distance entre les prédictions d'un modèle et les véritables étiquettes. Plus la valeur de la fonction de coût est faible, et mieux le modèle est adapté aux données. La fonction que vous minimisez dépend de votre modèle et de votre problème, mais il est généralement judicieux d'examiner sa valeur à la fois sur le jeu d'entraînement et sur le jeu de validation.

Cela aide généralement à estimer le *compromis biais-variance* de notre modèle, qui évalue la mesure dans laquelle notre modèle a appris à partir des données des informations précieuses, généralisables, sans mémoriser les détails de notre jeu d'entraînement.

Je suppose ici que vous êtes familier avec les métriques de classification standard, mais voici un bref rappel, juste au cas où. Pour les problèmes de classification, l'exactitude (*accuracy*) représente la proportion d'échantillons qu'un modèle prédit correctement. En d'autres termes, c'est la proportion de résultats vrais, qui sont aussi bien de vrais positifs que de vrais négatifs. Dans les cas de fort déséquilibre, une grande exactitude peut masquer un mauvais modèle. Si 99 % des cas sont positifs, un modèle qui prédit toujours la classe positive aura une exactitude de 99 %, mais il ne sera pas très utile. La précision, le rappel et le score f1 permettent de remédier à cette limitation. La précision est la proportion de vrais positifs parmi les échantillons prédits comme tels. Le rappel (*recall*) est la proportion de vrais positifs parmi les échantillons qui avaient une étiquette positive. Le score f1 est la moyenne harmonique de la précision et du rappel.

Dans le notebook d'entraînement d'un modèle simple du dépôt GitHub de ce livre (voir `train_simple_model.ipynb`), nous entraînons une première version d'une forêt aléatoire en utilisant les vecteurs TF-IDF et les caractéristiques que nous avons identifiées dans la section « Caractéristiques pour l'Éditeur ML » du Chapitre 4.

Voici les valeurs d'exactitude, de précision, de rappel et de score f1 pour notre jeu d'entraînement et notre jeu de validation.

```
Training accuracy = 0.585, precision = 0.582, recall = 0.585, f1 = 0.581
Validation accuracy = 0.614, precision = 0.615, recall = 0.614, f1 = 0.612
```

Un rapide coup d'œil sur ces métriques nous permet de constater deux choses :

- Comme nous disposons d'un jeu de données équilibré composé de deux classes, le fait de choisir une classe au hasard pour chaque échantillon nous donnerait une exactitude d'environ 50 %. L'exactitude de notre modèle atteint 61 %, ce qui est mieux qu'une base de référence aléatoire.
- Notre exactitude sur le jeu de validation est plus élevée que sur le jeu d'entraînement. Il semble que notre modèle fonctionne bien sur les données non vues.

Plongeons plus profondément pour en apprendre plus sur les performances du modèle.

Compromis entre biais et variance

Une faible performance sur le jeu d'entraînement est le symptôme d'un biais élevé, également appelé *sous-apprentissage*, qui signifie qu'un modèle n'a pas réussi à saisir des informations utiles : il n'est même pas capable de bien fonctionner sur les points de données dont il a déjà obtenu les étiquettes.

De fortes performances sur le jeu d'entraînement, mais de faibles performances sur le jeu de validation, sont le symptôme d'une variance élevée, également appelée *surapprentissage*, ce qui signifie qu'un modèle a trouvé des moyens d'apprendre les correspondances entrée/sortie pour les données sur lesquelles il a été entraîné, mais que ce qu'il a appris ne se généralise pas aux données non vues.

Le sous-apprentissage et le surapprentissage sont deux cas extrêmes du compromis entre biais et variance, qui décrit la manière dont les types d'erreurs dans un modèle changent en fonction de l'augmentation de sa complexité. À mesure que la complexité du modèle s'accroît, la variance augmente et le biais diminue, et donc le modèle passe d'un sous-apprentissage à un surapprentissage. Vous pouvez voir cela sur la Figure 5.5.

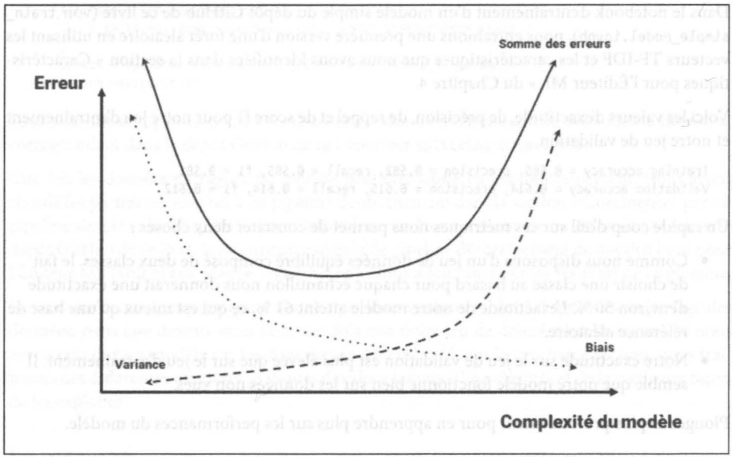


Figure 5.5 : Plus la complexité augmente, plus le biais diminue, mais plus la variance augmente également.

Dans notre cas, comme notre performance de validation est meilleure que notre performance d'entraînement, nous pouvons voir que notre modèle ne subit pas de surapprentissage. Nous pouvons probablement augmenter la complexité de notre modèle ou de ses caractéristiques pour améliorer les performances. Pour agir sur le compromis biais-variance, il faut trouver un point optimal entre la réduction du biais, qui augmente la performance du modèle sur le jeu d'entraînement, et la réduction de la variance, qui augmente sa performance sur le jeu de validation (ce qui entraîne souvent comme sous-produit une détérioration de la performance de l'entraînement).

Les métriques de performance aident à générer une perspective globale quant aux performances d'un modèle. Cela aide à deviner comment un modèle se comporte, mais ne donne pas beaucoup d'intuition pour ce qui concerne la réussite ou l'échec de celui-ci. Pour améliorer notre modèle, nous devons plonger encore plus profondément.

Aller au-delà des métriques agrégées

Une métrique de performance permet de déterminer si un modèle a appris correctement à partir d'un jeu de données, ou s'il a besoin d'être amélioré. L'étape suivante consiste à examiner les

résultats de manière plus approfondie afin de comprendre en quoi un modèle échoue ou réussit. Cette étape est cruciale pour deux raisons :

Validation des performances

> Les métriques de performance peuvent être très trompeuses. Lorsque l'on travaille sur un problème de classification avec des données très déséquilibrées, comme la prédiction d'une maladie rare qui apparaît chez moins de 1 % des patients, tout modèle qui prédit toujours qu'un patient est en bonne santé atteindra une exactitude de 99 %, bien qu'il n'ait aucun pouvoir prédictif. Il existe des métriques de performance adaptées à la plupart des problèmes (le score f1[35] serait plus efficace pour le problème précédent), mais l'essentiel est de se rappeler qu'il s'agit de métriques agrégées et qu'elles donnent une image incomplète de la situation. Pour se fier aux performances d'un modèle, il faut examiner les résultats à un niveau plus granulaire.

Itération

> La construction de modèles est un processus itératif, et la meilleure façon d'entamer une boucle d'itération est d'identifier à la fois ce qu'il faut améliorer et comment l'améliorer. Les métriques de performance n'aident pas à identifier les points faibles d'un modèle et la partie du pipeline qui doit être améliorée. Trop souvent, j'ai vu des spécialistes des données tenter d'améliorer les performances des modèles en essayant simplement de nombreux autres modèles ou hyperparamètres, ou encore en construisant des caractéristiques supplémentaires au hasard. Cette approche revient à lancer des fléchettes sur le mur les yeux bandés. La clé pour construire rapidement des modèles efficaces est d'identifier et de traiter les raisons spécifiques pour lesquelles ils échouent.

En gardant ces deux motivations à l'esprit, nous allons couvrir quelques moyens de plonger toujours plus profondément dans les performances d'un modèle.

Évaluer votre modèle : aller au-delà de l'exactitude

Il existe une multitude de façons d'inspecter la performance d'un modèle, et nous ne couvrirons pas toutes les méthodes d'évaluation possibles. Nous nous concentrerons sur quelques-unes de ces méthodes qui sont souvent utiles pour déceler ce qui pourrait se passer sous la surface.

Lorsqu'il s'agit d'enquêter sur les performances d'un modèle, considérez-vous comme un détective et chacune des méthodes abordées ci-après comme des moyens différents de révéler les indices. Nous commencerons par couvrir plusieurs techniques qui mettent en contraste les prédictions d'un modèle avec les données afin de découvrir des schémas intéressants.

35 Voir l'adresse `https://bit.ly/2yTKYaa`.

Contraste entre données et prédictions

La première étape pour évaluer un modèle en profondeur consiste à trouver des moyens plus granulaires que les métriques agrégées pour révéler le contraste entre données et prédictions. Nous aimerions décomposer les métriques de performance agrégées, telles que l'exactitude, la précision ou le rappel, sur différents sous-ensembles de nos données. Voyons comment faire pour relever le défi courant qu'est la classification en matière de ML.

Vous pouvez trouver tous les exemples de code dans le notebook de comparaison des données et des prédictions dans le dépôt GitHub de ce livre (voir `comparing_data_to_predictions.ipynb`).

Pour les problèmes de classification, je recommande généralement de commencer par regarder une matrice de confusion, comme celle qui est illustrée sur la Figure 5.6, dont les lignes représentent chaque classe vraie, et les colonnes représentent les prédictions de notre modèle. Un modèle avec des prédictions parfaites aura une matrice de confusion avec des zéros partout sauf dans la diagonale allant du haut à gauche au bas à droite. En réalité, c'est rarement le cas. Voyons pourquoi une matrice de confusion est souvent très utile.

Matrice de confusion

Une matrice de confusion nous permet de voir d'un coup d'œil si notre modèle est particulièrement efficace sur certaines classes et s'il est en difficulté sur d'autres. Cela est particulièrement utile pour les jeux de données comportant de nombreuses classes différentes, ou des classes qui sont déséquilibrées.

Souvent, j'ai vu des modèles d'une exactitude impressionnante montrer une matrice de confusion avec une colonne entièrement vide, ce qui signifie qu'il y a une classe que le modèle ne prédit jamais. Cela arrive souvent pour des classes rares et peut donc parfois être inoffensif. Cependant, si la classe rare représente un résultat important, comme par exemple un emprunteur en défaut de paiement, une matrice de confusion nous aidera à remarquer le problème. Nous pouvons alors le corriger en accordant par exemple une plus grande importance à la classe rare dans la fonction de perte de notre modèle.

La ligne du haut sur la Figure 5.6 montre que le modèle initial que nous avons entraîné fonctionne bien lorsqu'il s'agit de prévoir des questions de faible qualité. La ligne du bas montre que le modèle a du mal à détecter toutes les questions ayant une bonne qualité. En effet, parmi toutes les questions qui ont reçu un score élevé, notre modèle ne prédit correctement leur classe que la moitié du temps. En revanche, en regardant la colonne de droite, on peut voir que lorsque le modèle prédit qu'une question est de qualité élevée, sa prédiction tend à être exacte.

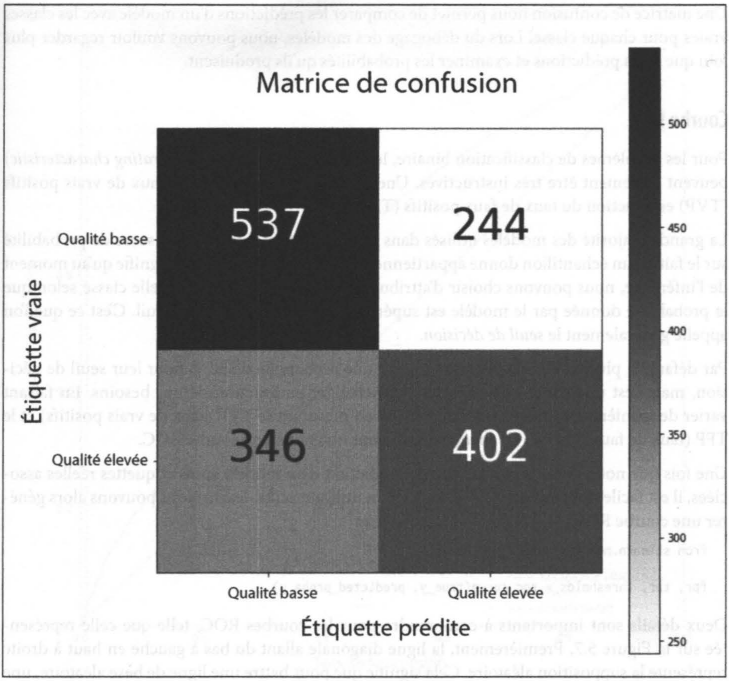


Figure 5.6 : Matrice de confusion pour une première base de référence sur notre tâche de classification des questions.

Les matrices de confusion peuvent être encore plus utiles lorsque l'on travaille sur des problèmes concernant plus de deux classes. Par exemple, j'ai travaillé une fois avec un ingénieur qui essayait de classifier des mots à partir d'énoncés de discours, et qui avait tracé une matrice de confusion pour son dernier modèle. Il a immédiatement remarqué deux valeurs symétriques, hors diagonale, qui étaient anormalement élevées. Ces deux classes (qui représentaient chacune un mot) confondaient le modèle et étaient la cause de la majorité de ses erreurs. Après un examen plus approfondi, il s'est avéré que les mots qui confondaient le modèle étaient *when* (quand) et *where* (où). La collecte de données supplémentaires pour ces deux termes a suffi pour aider le modèle à mieux différencier ces mots de consonance similaire.

Une matrice de confusion nous permet de comparer les prédictions d'un modèle avec les classes vraies pour chaque classe. Lors du débogage des modèles, nous pouvons vouloir regarder plus loin que leurs prédictions et examiner les probabilités qu'ils produisent.

Courbe ROC

Pour les problèmes de classification binaire, les courbes ROC (*receiver operating characteristic*) peuvent également être très instructives. Une courbe ROC représente le taux de vrais positifs (TVP) en fonction du taux de faux positifs (TFP).

La grande majorité des modèles utilisés dans la classification renvoient un score de probabilité sur le fait qu'un échantillon donné appartienne à une certaine classe. Cela signifie qu'au moment de l'inférence, nous pouvons choisir d'attribuer un échantillon à telle ou telle classe selon que la probabilité donnée par le modèle est supérieure ou non à un certain seuil. C'est ce que l'on appelle généralement le *seuil de décision*.

Par défaut, la plupart des classifieurs utilisent une probabilité de 50 % pour leur seuil de décision, mais c'est une chose que nous pouvons changer en fonction de nos besoins. En faisant varier de manière continue le seuil de 0 à 1 et en mesurant le TVP (taux de vrais positifs) et le TFP (taux de faux positifs) en chaque point, nous obtenons une courbe ROC.

Une fois que nous avons la probabilité de prédiction d'un modèle et les étiquettes réelles associées, il est facile d'obtenir les TVP et les TFP en utilisant scikit-learn. Nous pouvons alors générer une courbe ROC.

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(true_y, predicted_proba_y)
```

Deux détails sont importants à comprendre pour les courbes ROC, telle que celle représentée sur la Figure 5.7. Premièrement, la ligne diagonale allant du bas à gauche en haut à droite représente la supposition aléatoire. Cela signifie que pour battre une ligne de base aléatoire, une paire classifieur/seuil doit se trouver au-dessus de cette ligne. En outre, le modèle parfait serait représenté par la ligne pointillée qui longe les bords en haut et à gauche.

En raison de ces deux détails, les modèles de classification utilisent souvent l'aire sous la courbe (AUC) pour représenter la performance. Plus l'AUC est grande, plus notre classifieur peut se rapprocher d'un modèle « parfait ». Un modèle aléatoire aura une AUC de 0,5, tandis qu'un modèle parfait aura une AUC de 1. Toutefois, lorsque nous nous intéressons à une application pratique, nous devons choisir un seuil spécifique qui nous donne le rapport TVP/TFP le plus utile pour notre cas d'utilisation.

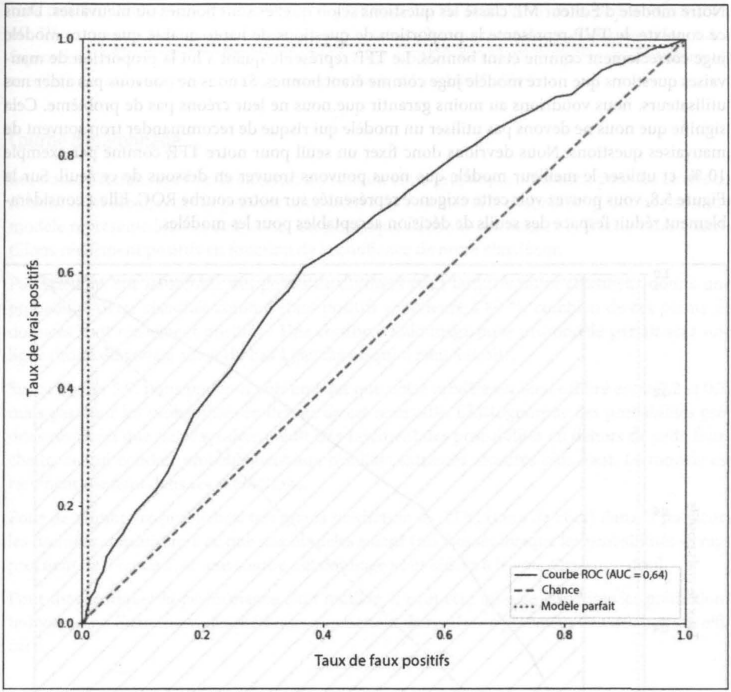


Figure 5.7 : Courbe ROC pour un modèle initial.

C'est pourquoi je recommande d'ajouter des lignes verticales ou horizontales à une courbe ROC afin de représenter nos besoins en matière de produits. Lors de la construction d'un système qui achemine les demandes des clients vers le personnel si cela est jugé suffisamment urgent, le TFP que vous pouvez vous permettre est alors entièrement déterminé par la capacité de votre équipe d'assistance et le nombre d'utilisateurs que vous avez. Cela signifie que tout modèle dont le TFP est supérieur à cette limite ne doit même pas être pris en considération.

Le fait de tracer un seuil sur une courbe ROC vous permet d'avoir un objectif plus concret que de simplement obtenir le plus grand score possible pour l'aire AUC. Assurez-vous que vos efforts sont adaptés à votre objectif !

Notre modèle d'Éditeur ML classe les questions selon qu'elles sont bonnes ou mauvaises. Dans ce contexte, le TVP représente la proportion de questions de haute qualité que notre modèle juge correctement comme étant bonnes. Le TFP représente quant à lui la proportion de mauvaises questions que notre modèle juge comme étant bonnes. Si nous ne pouvons pas aider nos utilisateurs, nous voudrions au moins garantir que nous ne leur créons pas de problème. Cela signifie que nous ne devons pas utiliser un modèle qui risque de recommander trop souvent de mauvaises questions. Nous devrions donc fixer un seuil pour notre TFP, comme par exemple 10 %, et utiliser le meilleur modèle que nous pouvons trouver en dessous de ce seuil. Sur la Figure 5.8, vous pouvez voir cette exigence représentée sur notre courbe ROC. Elle a considérablement réduit l'espace des seuils de décision acceptables pour les modèles.

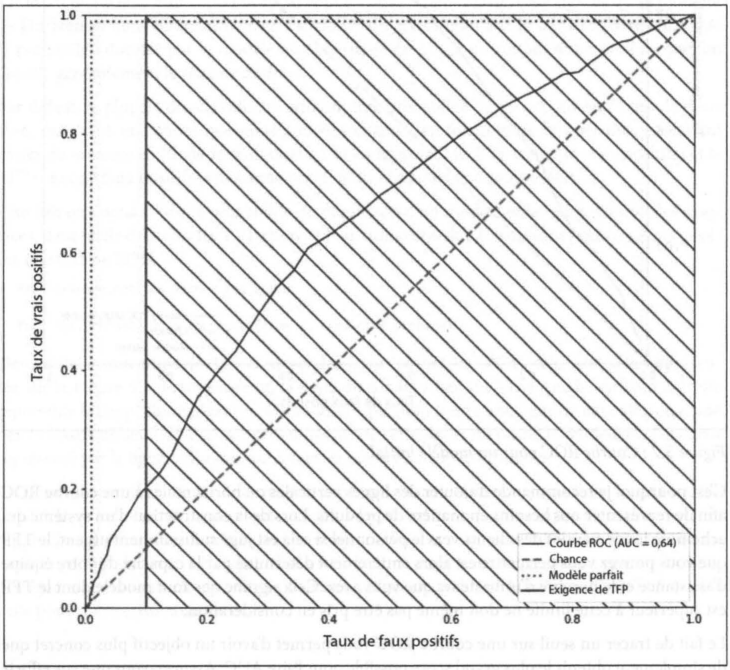


Figure 5.8 : Ajout de lignes ROC représentant les besoins de notre produit.

Une courbe ROC nous donne une vision plus nuancée de l'évolution de la performance d'un modèle à mesure que nous rendons ses prédictions plus ou moins conservatrices. Une autre façon d'examiner la probabilité de prédiction d'un modèle consiste à comparer ses distributions avec les véritables distributions de classe pour voir s'il est bien calibré.

Courbe d'étalonnage

Les tracés d'étalonnage (ou de calibration) sont un autre graphique informatif pour les tâches de classification binaire, car ils peuvent nous aider à déterminer si la probabilité produite par notre modèle représente bien sa confiance. Un graphique d'étalonnage montre la fraction des échantillons réellement positifs en fonction de la confiance de notre classifieur.

Par exemple, sur l'ensemble des points de données pour lesquels notre classifieur donne une probabilité d'être classifiés comme étant positifs supérieure à 80 %, combien de ces points de données sont réellement positifs ? Une courbe d'étalonnage pour un modèle parfait sera une ligne droite diagonale allant du bas à gauche jusqu'en haut à droite.

Sur la Figure 5.9, nous pouvons voir en haut que notre modèle est bien calibré entre 0,2 et 0,7, mais pas pour les probabilités en dehors de cet intervalle. L'histogramme des probabilités prédites révèle ici que notre modèle prédit très rarement des probabilités en dehors de cette fourchette, ce qui conduit probablement aux résultats extrêmes montrés plus haut. Le modèle est rarement confiant dans ses prédictions.

Pour de nombreux problèmes, tels que la prédiction de TDC (taux de clics) dans la publicité, les données conduiront à ce que nos modèles soient très faussés lorsque les probabilités se rapprochent de 0 ou de 1, et une courbe d'étalonnage nous aidera à le voir d'un coup d'œil.

Pour diagnostiquer la performance d'un modèle, il peut être utile de visualiser les prédictions individuelles. Examinons des méthodes permettant de rendre ce processus de visualisation efficace.

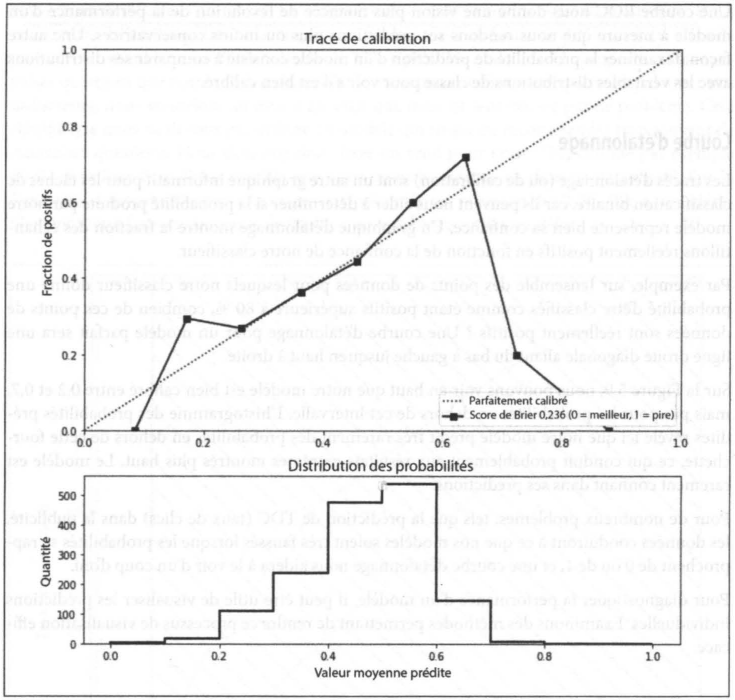


Figure 5.9 : Une courbe d'étalonnage : en haut, la ligne diagonale représente un modèle parfait, en bas, un histogramme des valeurs prédites.

Réduction de la dimensionnalité pour les erreurs

Nous avons décrit les techniques de vectorisation et de réduction de la dimensionnalité pour l'exploration des données dans les sections « Vectorisation » et « Réduction de la dimensionnalité » du Chapitre 4. Voyons comment ces mêmes techniques peuvent être utilisées pour rendre l'analyse des erreurs plus efficace.

Lorsque nous avons abordé initialement la manière d'utiliser les méthodes de réduction de la dimensionnalité pour visualiser les données, nous avons coloré chaque point d'un jeu de données selon sa classe pour observer la topologie des étiquettes. Lors de l'analyse des erreurs dans les modèles, nous pouvons utiliser différents schémas de couleur pour identifier ces erreurs.

Pour identifier les tendances en ce qui concerne les erreurs, colorez chaque point de données selon que la prédiction d'un modèle était correcte ou non. Cela vous permettra d'identifier les types de points de données similaires sur lesquels un modèle est peu performant. Une fois que vous avez identifié une région dans laquelle un modèle fonctionne mal, visualisez quelques points de données dans cette région. La visualisation d'échantillons concrets est un excellent moyen de générer les caractéristiques qui y sont représentées afin d'aider un modèle à mieux s'y adapter.

Pour aider à faire ressortir les tendances dans des échantillons concrets, vous pouvez également utiliser les méthodes de clustering de la section « Regroupement » du Chapitre 4. Après avoir regroupé les données, mesurez la performance du modèle sur chaque cluster, et identifiez les clusters où le modèle est le moins performant. Inspectez les points de données dans ces clusters pour vous aider à générer davantage de caractéristiques.

Les techniques de réduction de la dimensionnalité sont une façon de faire apparaître des échantillons stimulants. Pour ce faire, nous pouvons également utiliser directement le score de confiance d'un modèle.

La méthode Top-k

La recherche de régions d'erreur denses permet d'identifier les modes de défaillance d'un modèle. Auparavant, nous avons utilisé la réduction de la dimensionnalité pour nous aider à trouver de telles régions, mais nous pouvons aussi utiliser directement un modèle lui-même. En exploitant les probabilités de prédiction, nous pouvons identifier les points de données qui étaient les plus délicats, ou pour lesquels un modèle était le plus incertain. Appelons cette approche la *méthode Top-k*.

La méthode Top-k est simple. Tout d'abord, choisissez un nombre gérable d'échantillons à visualiser, que nous appellerons k. Pour un projet personnel où une seule personne visualisera les résultats, commencez par dix à quinze échantillons. Pour chaque classe ou groupe que vous avez trouvé précédemment, visualisez :

- les k échantillons les plus performants ;
- les k pires échantillons en termes de performances ;
- les k échantillons les plus incertains.

La visualisation de ces échantillons vous aidera à identifier ceux qui sont faciles, difficiles ou qui prêtent à confusion pour votre modèle. Voyons chaque catégorie plus en détail.

Les k échantillons les plus performants

Tout d'abord, affichez les k échantillons que votre modèle a prédits correctement et pour lesquels il était le plus confiant. Lorsque vous visualisez ces échantillons, essayez d'identifier les points communs entre eux en termes de valeur des caractéristiques qui pourraient expliquer les performances du modèle. Cela vous aidera à identifier les caractéristiques qui sont exploitées avec succès par un modèle.

Après avoir visualisé des échantillons réussis pour identifier les caractéristiques exploitées par un modèle, tracez les échantillons qui ont échoué pour identifier les caractéristiques qu'il ne parvient pas à saisir.

Les k pires échantillons en termes de performances

Affichez les k échantillons que votre modèle a prédits de manière incorrecte et dont il était le plus sûr. Commencez par k échantillons dans les données d'entraînement, puis de validation.

Tout comme la visualisation des clusters d'erreurs, la visualisation de k échantillons de points de données sur lesquels un modèle fonctionne le moins bien dans un jeu d'entraînement peut aider à identifier les tendances des points de données sur lesquels le modèle échoue. L'affichage de ces points de données vous aide à identifier les caractéristiques additionnelles qui faciliteraient la tâche d'un modèle.

En explorant l'erreur d'un modèle initial pour l'Éditeur ML, par exemple, j'ai constaté que certaines questions postées recevaient un score faible parce qu'elles ne contenaient pas de réelle question. Le modèle n'était pas dans ce cas en mesure de prédire un faible score, et j'ai donc ajouté une fonction permettant de compter les points d'interrogation dans le corps du texte. L'ajout de cette fonction a permis au modèle de faire des prédictions exactes pour ces questions « sans question ».

La visualisation des k pires échantillons dans les données de validation peut aider à identifier ceux qui diffèrent de manière significative des données d'entraînement. Si vous identifiez des échantillons trop « difficiles » dans le jeu de validation, reportez-vous aux conseils de la section « Partager votre jeu de données », plus haut dans ce chapitre, pour mettre à jour votre stratégie de fractionnement des données.

Enfin, les modèles ne sont pas toujours sûrs d'avoir raison ou tort : ils peuvent aussi produire des prédictions incertaines. C'est ce que nous allons voir maintenant.

Les k échantillons les plus incertains

La visualisation des k échantillons les plus incertains consiste à afficher ceux pour lesquels un modèle était le moins confiant dans ses prédictions. Dans le cas d'une classification, sur laquelle ce livre se concentre principalement, les échantillons incertains sont ceux où un modèle produit une probabilité aussi égale possible pour chaque classe.

Si un modèle est bien étalonné (ou calibré, voir la section « Courbe d'étalonnage » plus haut dans ce chapitre pour des explications sur cette notion), il produira des probabilités uniformes pour des échantillons qu'un étiqueteur humain considérerait également comme étant incertains. Par exemple, pour un classifieur de chats par rapport à des chiens, une image contenant à la fois un chien et un chat entrerait dans cette catégorie.

Des échantillons incertains dans le jeu d'entraînement sont souvent le symptôme d'étiquettes contradictoires. En effet, si un jeu d'entraînement contient deux échantillons en double, ou similaires, qui sont chacun étiquetés comme appartenant à une classe différente, un modèle minimisera sa perte pendant l'entraînement en produisant une probabilité égale pour chaque classe lorsque cet échantillon lui est présenté. Des étiquettes contradictoires conduisent donc à des prédictions incertaines, et vous pouvez utiliser la méthode Top-k pour tenter de trouver ces échantillons.

Le tracé des k échantillons les plus incertains dans votre jeu de validation peut vous aider à trouver des lacunes dans vos données d'entraînement. Les échantillons de validation sur lesquels un modèle est incertain, mais qui sont clairs pour un étiqueteur humain, sont souvent le signe que le modèle n'a pas été exposé à ce type de données dans son jeu d'entraînement. Tracer les k échantillons les plus incertains pour un jeu de validation peut aider à identifier les types de données qui devraient être présents dans le jeu d'entraînement.

L'évaluation Top-k peut être implémentée de manière simple. Dans la prochaine section, je vous montrerai un exemple pratique.

Conseils pour l'implémentation de la méthode Top-k

Ce qui suit est une implémentation simple de la méthode Top-k qui s'appuie sur les DataFrames de pandas. La fonction prend en entrée un DataFrame contenant les probabilités prédites et les étiquettes, et renvoie chacun des Top-k définis ci-dessus. Elle se trouve dans le dépôt GitHub de ce livre (voir le notebook `top_k.ipynb`).

```
def get_top_k(df, proba_col, true_label_col, k=5, decision_threshold=0.5):
    """
    Pour les problèmes de classification binaire
    Retourne les k échantillons les plus corrects et les plus incorrects
    pour chaque classe.
    Retourne également les k échantillons les plus incertains
    :paramètre df: DataFrame contenant les prédictions et les étiquettes vraies
    :paramètre proba_col: nom de colonne pour les probabilités prédites
    :paramètre true_label_col: nom de colonne pour les étiquettes vraies
    :paramètre k: nombre d'échantillons à montrer pour chaque catégorie
    :paramètre decision_threshold: limite de décision du classifieur pour
                                   classifier comme positif
    :Retourne: correct_pos, correct_neg, incorrect_pos, incorrect_neg, uncertain
    """
```

```
    # Obtient les prédictions correctes et incorrectes
    correct = df[
        (df[proba_col] > decision_threshold) == df[true_label_col]
    ].copy()
    incorrect = df[
        (df[proba_col] > decision_threshold) != df[true_label_col]
    ].copy()

    top_correct_positive = correct[correct[true_label_col]].nlargest( k,
        proba_col
    )
    top_correct_negative = correct[~correct[true_label_col]].nsmallest( k,
        proba_col
    )
    top_incorrect_positive = incorrect[incorrect[true_label_col]].nsmallest( k,
        proba_col
    )
    top_incorrect_negative = incorrect[~incorrect[true_label_col]].nlargest( k,
        proba_col
    )

    # Obtient les échantillons les plus proches du seuil de décision
    most_uncertain = df.iloc[
        (df[proba_col] - decision_threshold).abs().argsort()[:k]
    ]

    return (
        top_correct_positive,
        top_correct_negative,
        top_incorrect_positive,
        top_incorrect_negative,
        most_uncertain,
)
```

Illustrons la méthode Top-k en l'utilisant pour l'Éditeur ML.

Méthode Top-k pour l'Éditeur ML

Nous allons appliquer la méthode Top-k au premier classifieur que nous avons entraîné. Un notebook contenant des exemples d'utilisation de la méthode Top-k est disponible dans le dépôt GitHub de ce livre.

La Figure 5.10 montre les deux échantillons les plus corrects pour chaque classe pour notre premier modèle d'Éditeur ML. La caractéristique qui diffère le plus entre les deux classes est `text_len`, qui représente la longueur du texte. Le classifieur a appris que les bonnes questions ont tendance à être longues, et les mauvaises courtes. Il s'appuie donc fortement sur la longueur du texte pour distinguer les différentes classes.

```
In [166]: # Most confident correct positive predictions
          top_pos(to_display)
```

Out[166]:

Id	predicted_probs	true_label	Title	body_text	text_len	action_verb_full	question_mark_full	language_question
38388	0.84	True	Punctuation when using inline dialogue	I am a bit crazy about punctuation and I have a question that I'm struggling to find a consensus...	277	False	True	False
7602	0.81	True	Is it unusual for a flashback to have a very long dialogue?	This flashback is from a short story I'm writing (unedited first draft):\r\n\r\n met Limrel last sum...	870	True	True	False

```
In [167]: # Most confident correct negative predictions
          top_neg(to_display)
```

Out[167]:

Id	predicted_probs	true_label	Title	body_text	text_len	action_verb_full	question_mark_full	language_question
7878	0.20	False	When quoting a person's informal speech, how much liberty do you have to make changes to what th...	Even during a formal interview for a news article, people speak informally. They say "uhm", they...	116	True	True	False
19463	0.21	False	Printing by the Publisher	My first book was published through Xlibris. They have reported no sales from numerous authors, ...	131	True	True	False

Figure 5.10 : Top-k des échantillons les plus corrects.

La Figure 5.11 confirme cette hypothèse. Les questions sans réponse que notre classifieur prédit comme étant les plus susceptibles de recevoir une réponse sont les plus longues, et *vice versa*. Cette observation corrobore également ce que nous allons voir un peu plus loin dans la section « Évaluer l'importance des caractéristiques », où nous montrerons que `text_len` est effectivement la caractéristique la plus importante.

```
In [168]: # Most confident incorrect negative predictions
          worst_pos(to_display)
```

Out[168]:

Id	predicted_probs	true_label	Title	body_text	text_len	action_verb_full	question_mark_full	language_question
18735	0.23	True	What do I need to know about publishing a book which I have illustrated, not written?	I recently went to the market to buy books for my 2 year old kid and found many expensive books ...	184	True	True	False
19909	0.25	True	How to copyright a book without lawyer and outside USA?	I would like to publish an ebook with amazon and i dont have time/money to keep copyrights with ...	56	True	True	False

```
In [169]: # Most confident incorrect positive predictions
          worst_neg(to_display)
```

Out[169]:

Id	predicted_probs	true_label	Title	body_text	text_len	action_verb_full	question_mark_full	language_question
12574	0.86	False	When does repetition start becoming tedious (especially metaphors)?	I always find myself using CTRL + F to remove the words/phrases I think I'm repeating too much (...	361	False	True	False
42039	0.78	False	How do I write a MODERN combat/violence scene without being dry?	Warning: I have ADHD and this might be a little ramble-y, sorry.\n\nI'm completely stumped. I'm tr...	770	True	True	False

Figure 5.11 : Top-k des échantillons les plus incorrects.

Nous avons établi que le classifieur utilise `text_len` pour identifier facilement les questions avec ou sans réponse, mais que cette caractéristique n'est pas suffisante et entraîne des erreurs de classification. Nous devrions ajouter d'autres caractéristiques pour améliorer notre modèle. Visualiser plus de deux échantillons aiderait à identifier davantage de caractéristiques candidates.

L'utilisation de la méthode Top-k sur les données d'entraînement et de validation permet d'identifier les limites de notre modèle et de notre jeu de données. Nous avons vu comment elle peut aider à déterminer si un modèle a la capacité de représenter des données, si un jeu de données est suffisamment équilibré et s'il contient suffisamment d'échantillons représentatifs.

Nous avons surtout abordé les méthodes d'évaluation des modèles de classification, car ces modèles sont applicables à de nombreux problèmes concrets. Examinons brièvement les moyens de contrôler les performances lorsque l'on ne fait pas de classification.

Autres modèles

De nombreux modèles peuvent être évalués à l'aide d'un framework de classification. Pour la détection d'objets, par exemple, où l'objectif est qu'un modèle produise des boîtes de délimitation autour des objets recherchés dans une image, l'exactitude est une métrique courante. Comme chaque image peut avoir plusieurs boîtes de délimitation représentant des objets et des prédictions, le calcul de l'exactitude nécessite une étape supplémentaire. Tout d'abord, le calcul du chevauchement entre les prédictions et les étiquettes (souvent à l'aide de l'indice Jaccard[36]) permet de marquer chaque prédiction comme étant correcte ou incorrecte. Ensuite, on peut calculer l'exactitude et utiliser toutes les méthodes précédentes de ce chapitre.

De même, lors de la construction de modèles visant à recommander un contenu, la meilleure façon d'itérer est souvent de tester le modèle sur une variété de catégories et de rapporter ses performances. L'évaluation devient alors similaire à un problème de classification, où chaque catégorie représente une classe.

Pour les types de problèmes où ces méthodes peuvent s'avérer délicates, comme dans le cas des modèles génératifs, vous pouvez toujours utiliser votre exploration précédente des données pour scinder un jeu de données en de multiples catégories, et générer des métriques de performance pour chaque catégorie.

Lorsque j'ai travaillé avec un spécialiste des données pour construire un modèle de simplification des phrases, l'examen de la performance du modèle conditionné par la longueur des phrases a montré que des phrases plus longues s'avéraient beaucoup plus difficiles à traiter par le modèle. Cela a nécessité une inspection et un étiquetage à la main, mais a conduit à l'étape suivante, qui consistait clairement à compléter les données d'entraînement par des phrases plus longues, ce qui a permis d'améliorer les performances de manière significative.

36 Voir l'adresse `https://fr.wikipedia.org/wiki/Indice_et_distance_de_Jaccard`.

Nous avons couvert de nombreuses façons d'inspecter les performances d'un modèle en comparant ses prédictions avec des étiquettes, mais nous pouvons également inspecter le modèle lui-même. Si un modèle ne fonctionne pas du tout bien, il peut être utile d'essayer d'interpréter ses prédictions.

Évaluer l'importance des caractéristiques

Une autre façon d'analyser les performances d'un modèle est d'examiner quelles caractéristiques des données il utilise pour faire des prédictions. C'est ce qu'on appelle *l'analyse de l'importance des caractéristiques*. L'évaluation de l'importance des caractéristiques est utile pour éliminer ou itérer sur les caractéristiques qui n'aident pas actuellement le modèle. L'importance des caractéristiques peut également aider à identifier celles qui sont suspicieusement prédictives, ce qui est souvent un signe de fuite de données. Nous commencerons par générer l'importance des caractéristiques dans le cas des modèles où il est facile de le faire, puis nous couvrirons les situations où ces caractéristiques peuvent être difficiles à extraire directement.

Évaluation directe à partir d'un classifieur

Pour valider le fait qu'un modèle fonctionne correctement, visualisez les caractéristiques que le modèle utilise ou ignore. Pour les modèles simples, tels que les arbres de régression ou de décision, il est facile d'extraire l'importance des caractéristiques en examinant les paramètres appris du modèle.

Pour le premier modèle que nous avons utilisé dans l'étude de cas de l'Éditeur ML, qui est une forêt aléatoire, nous pouvons simplement utiliser l'API de scikit-learn pour obtenir un classement de l'importance de toutes les caractéristiques. Le code de l'importance des caractéristiques et ses utilisations peut être trouvé dans le notebook correspondant du dépôt GitHub de ce livre (voir `feature_importance.ipynb`).

```
def get_feature_importance(clf, feature_names):
    importances = clf.feature_importances_
    indices_sorted_by_importance = np.argsort(importances)[::-1]
    return list(
        zip(
            feature_names[indices_sorted_by_importance],
            importances[indices_sorted_by_importance],
        )
    )
```

Si nous utilisons la fonction ci-dessus sur notre modèle entraîné, avec un traitement de liste simple, nous pouvons obtenir une liste des dix caractéristiques les plus informatives :

```
Top 10 importances:

text_len: 0.0091
are: 0.006
what: 0.0051
writing: 0.0048
can: 0.0043
ve: 0.0041
on: 0.0039
not: 0.0039
story: 0.0039
as: 0.0038
```

Il y a quelques points à noter ici :

- La longueur du texte est la caractéristique la plus informative.
- Les autres caractéristiques que nous avons générées n'apparaissent pas du tout, avec des importances très largement inférieures aux autres. Le modèle n'a pas été en mesure de les exploiter pour séparer les classes de manière significative.
- Les autres caractéristiques représentent soit des mots très courants, soit des noms en rapport avec le sujet de l'écriture.

Comme notre modèle et nos caractéristiques sont simples, ces résultats sont en fait susceptibles de nous donner des idées pour de nouvelles caractéristiques à construire. Nous pourrions, par exemple, ajouter une caractéristique qui compte l'utilisation de mots courants et rares pour voir s'ils sont prédictifs d'une réponse recevant un score élevé.

Si les caractéristiques ou les modèles deviennent complexes, la génération de l'importances des caractéristiques nécessite l'utilisation d'outils d'explicabilité de modèles.

Explicateurs de boîte noire

Lorsque la complexité des caractéristiques augmente, leur importance peut devenir plus difficile à interpréter. Certains modèles plus complexes, tels que les réseaux de neurones, peuvent même ne pas être en mesure d'exposer l'importance de leurs caractéristiques apprises. Dans de telles situations, il peut être utile de recourir à des explicateurs dits de *boîte noire*, qui tentent d'expliquer les prédictions d'un modèle indépendamment de son fonctionnement interne.

Généralement, ces explicateurs identifient des caractéristiques prédictives pour un modèle sur un point de données donné plutôt que globalement. Ils le font en modifiant la valeur de chaque caractéristique pour un échantillon donné et en observant comment les prévisions du modèle changent en conséquence. LIME et SHAP sont deux explicateurs populaires.

Pour un exemple complet d'utilisation, reportez-vous au notebook `black_box_explainer.ipynb` dans le dépôt GitHub du livre.

La Figure 5.12 montre une explication fournie par LIME concernant les mots qui ont été les plus importants pour décider de classifier cet exemple de question comme étant de haute qualité. LIME a généré ces explications en retirant de manière répétée des mots de la question passée en entrée, et en observant quels mots font que notre modèle penche davantage vers une classe ou vers une autre.

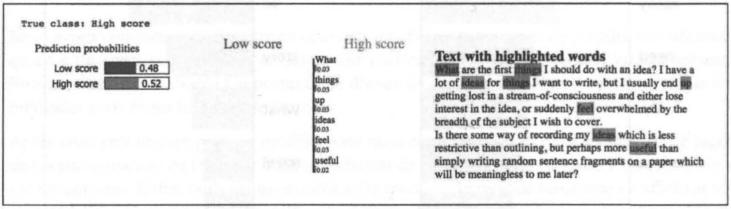


Figure 5.12 : Explication pour un échantillon particulier.

Nous pouvons voir que le modèle a correctement prédit que la question recevrait un score élevé. Cependant, le modèle n'était pas particulièrement confiant dans ce résultat, la probabilité ne dépassant pas 52 %. Le côté droit de la Figure 5.12 montre les mots qui ont eu le plus d'impact sur la prédiction. Ces mots ne semblent pas être particulièrement pertinents pour une question de haute qualité, et donc examinons d'autres échantillons pour voir si le modèle tire parti de motifs plus utiles.

Pour avoir une idée rapide des tendances, nous pouvons utiliser LIME sur un échantillon plus large de questions. L'exécution de LIME sur chaque question et l'agrégation des résultats peuvent nous donner une idée du mot que notre modèle trouve globalement prédictif pour prendre ses décisions.

Sur la Figure 5.13, nous indiquons les prédictions les plus importantes pour 500 questions de notre jeu de données. Nous pouvons voir que la tendance de notre modèle à se baser sur des mots courants est également apparente dans cet échantillon plus large. Il semble que le modèle ait du mal à généraliser au-delà de l'emploi de mots fréquents. Les caractéristiques du sac de mots représentant les mots rares ont le plus souvent une valeur de zéro. Pour améliorer la situation, nous pourrions soit collecter un jeu de données plus important pour exposer nos modèles à un vocabulaire plus varié, soit créer des caractéristiques qui seront moins rares.

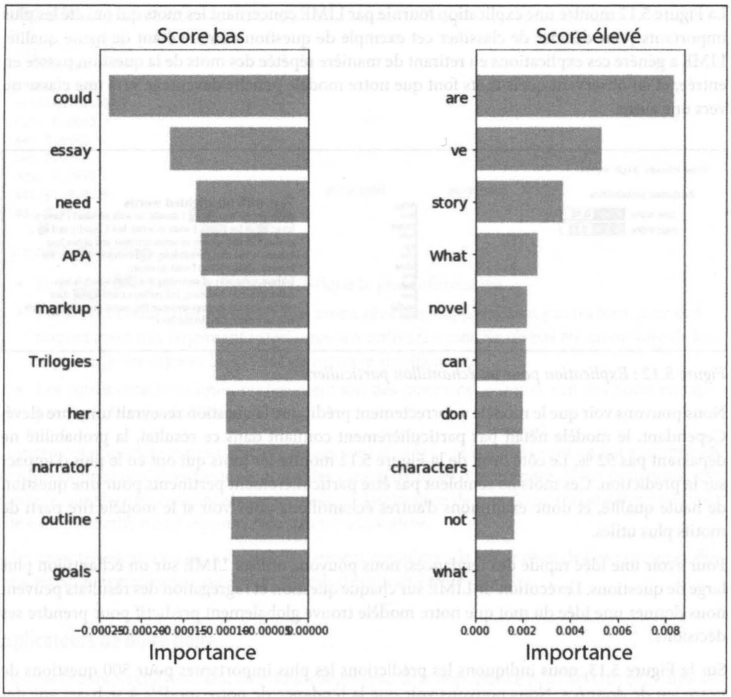


Figure 5.13 : Explication sur de multiples échantillons.

Vous serez souvent surpris par les prédicteurs que votre modèle finit par utiliser. Si certaines caractéristiques sont plus prédictives pour le modèle que vous ne le pensez, essayez de trouver des échantillons contenant ces caractéristiques dans vos données d'entraînement et examinez-les. Profitez de cette occasion pour vérifier la manière dont vous avez divisé votre jeu de données, et surveillez les fuites de données.

Lors de la construction d'un modèle permettant de classifier automatiquement les e-mails en différents sujets en fonction de leur contenu, par exemple, un ingénieur ML que j'encadrais a découvert un jour que le meilleur prédicteur était un code de trois lettres en haut des messages. Il s'est avéré que c'était un code interne au jeu de données qui correspondait presque parfaite-

ment aux étiquettes. Le modèle ignorait totalement le contenu du courriel et mémorisait une étiquette préexistante. Il s'agissait d'une exemple clair d'une fuite de données, qui n'a pu être détectée qu'en examinant l'importance des caractéristiques.

Conclusion

Nous avons commencé ce chapitre en couvrant les critères permettant de prendre une décision quant à un premier modèle en se basant sur tout ce que nous avons appris jusqu'à présent. Ensuite, nous avons abordé l'importance de diviser les données en plusieurs jeux, ainsi que les méthodes pour éviter les fuites de données.

Après avoir entraîné un premier modèle, nous nous sommes plongés dans les moyens de juger de ses performances en trouvant différentes façons de comparer et de contraster ses prédictions sur les données. Enfin, nous avons procédé à l'inspection du modèle lui-même en affichant les caractéristiques importantes et en utilisant un explicateur de boîte noire pour avoir une idée de la caractéristique que le modèle utilise pour faire des prédictions.

Vous devriez maintenant avoir une certaine intuition des améliorations que vous pourriez apporter à votre modélisation. Cela nous amène au sujet du Chapitre 6, où nous nous pencherons plus en profondeur sur les méthodes permettant de résoudre les problèmes que nous avons abordés ici en surface, en déboguant et en dépannant un pipeline ML.

CHAPITRE 6

Déboguer vos problèmes de ML

Dans le chapitre précédent, nous avons entraîné et évalué notre premier modèle.

Amener un pipeline à un niveau de performance satisfaisant est difficile et nécessite de multiples itérations. L'objectif de ce chapitre est de vous guider à travers un de ces cycles d'itération. Dans ce chapitre, je traiterai des outils de débogage des pipelines de modélisation et des moyens d'écrire des tests pour s'assurer qu'ils continuent à fonctionner une fois que nous commençons à les modifier.

Les meilleures pratiques en matière de logiciels encouragent les praticiens à tester, valider et inspecter régulièrement leur code, en particulier pour les étapes sensibles telles que la sécurité ou l'analyse des entrées. Cela ne devrait pas être différent pour le ML, où les erreurs dans un modèle peuvent être beaucoup plus difficiles à détecter que dans les logiciels traditionnels.

Nous allons vous donner quelques conseils qui vous aideront à vous assurer que votre pipeline est robuste et que vous pouvez l'essayer sans provoquer la défaillance de tout votre système. Mais commençons par les meilleures pratiques en matière de logiciels !

Meilleures pratiques en matière de logiciels

Pour la plupart des projets de ML, vous répéterez le processus de construction d'un modèle, d'analyse de ses lacunes et vous y remédierez à de multiples reprises. Vous êtes également susceptible de modifier chaque partie de votre infrastructure plus d'une fois, et il est donc crucial de trouver des méthodes pour augmenter la vitesse d'itération.

En ML, comme pour tout autre projet informatique, vous devriez suivre les meilleures pratiques éprouvées en matière de logiciels. La plupart d'entre elles peuvent être appliquées aux projets d'apprentissage automatique sans modification, par exemple en ne construisant que ce dont

vous avez besoin, ce que l'on appelle souvent le principe KISS (Keep It Stupid Simple, ou encore Keep It Simple, Stupid)[37].

Les projets ML sont itératifs par nature, et ils passent par de nombreuses itérations différentes d'algorithmes de nettoyage des données et de génération de caractéristiques, ainsi que de choix de modèles. Même en suivant ces meilleures pratiques, deux domaines finissent souvent par ralentir la vitesse d'itération : le débogage et les tests. L'accélération du débogage et de l'écriture des tests peut avoir un impact significatif sur n'importe quel projet, mais elle est encore plus cruciale pour les projets de ML, où la nature stochastique des modèles transforme souvent une simple erreur en une enquête qui dure de longues journées.

De nombreuses ressources existent pour vous aider à apprendre comment déboguer les programmes en général, comme le guide de débogage concis de l'université de Chicago[38]. Si, comme la plupart des praticiens du ML, votre langage de prédilection est Python, je vous recommande de consulter la documentation Python pour pdb, la bibliothèque standard de débogage de Python[39].

Plus que pour la plupart des logiciels classiques, cependant, le code ML peut souvent s'exécuter apparemment correctement, mais produire des résultats totalement absurdes. Cela signifie que si ces outils et conseils s'appliquent tels quels à la plupart du code ML, ils ne sont pas suffisants pour diagnostiquer des problèmes courants. C'est ce que j'illustre sur la Figure 6.1 : alors que, dans la plupart des applications logicielles, le fait d'avoir une forte réussite dans les tests peut nous donner un niveau élevé de confiance dans le bon fonctionnement de notre application, les pipelines ML peuvent passer de nombreux tests avec succès mais donner des résultats totalement incorrects. Un programme ML ne doit pas seulement s'exécuter – il devrait produire des résultats prédictifs précis.

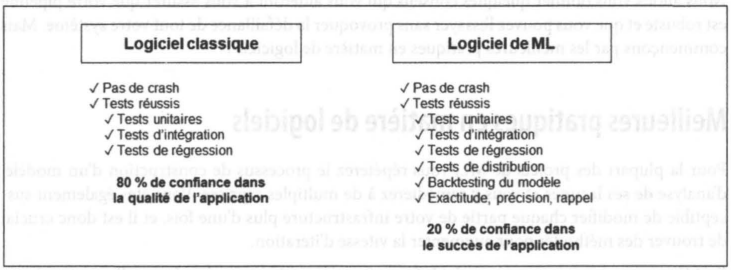


Figure 6.1 : Un pipeline de ML peut s'exécuter sans erreur et pour autant être erroné.

37 Voir l'adresse `https://people.apache.org/~fhanik/kiss.html`.
38 Voir l'adresse (en anglais) `https://uchicago-cs.github.io/debugging-guide/`.
39 Voir l'adresse `https://docs.python.org/fr/3/library/pdb.html`.

Comme le ML présente un ensemble de défis supplémentaires en matière de débogage, nous allons aborder quelques méthodes spécifiques qui peuvent se révéler utiles.

Meilleures pratiques spécifiques au ML

Lorsqu'il s'agit de ML, et plus que pour tout autre type de logiciel, il ne suffit pas d'exécuter un programme de bout en bout pour être convaincu de sa justesse. Un pipeline peut fonctionner entièrement sans erreur et produire un modèle totalement inutile.

Supposons que votre programme charge des données et les transmet à un modèle. Votre modèle prend en compte ces données et optimise ses paramètres sur la base d'un algorithme d'apprentissage. Finalement, votre modèle entraîné produit des sorties à partir d'un jeu de données différent. Votre programme s'est exécuté sans aucun bogue visible. Le problème est qu'en faisant simplement tourner votre programme, vous n'avez aucune garantie que les prédictions de votre modèle sont correctes.

La plupart des modèles prennent simplement une entrée numérique ayant une certaine forme (par exemple une matrice représentant une image), et produisent des données d'une forme différente (une liste de coordonnées de points clés dans l'image d'entrée, par exemple). Cela signifie que la plupart des modèles continueront à fonctionner, même si une étape de traitement des données a corrompu celles-ci avant de les transmettre au modèle, et ce tant que les données sont encore numériques et qu'elles ont une forme que le modèle peut recevoir en entrée.

Si votre pipeline de modélisation est peu performant, comment pouvez-vous savoir si cela est dû à la qualité d'un modèle ou à la présence d'un bogue, plus tôt dans le processus ?

La meilleure façon de s'attaquer à ces problèmes en matière de ML est de suivre une approche progressive. Commencez par valider le flux de données, puis la capacité d'apprentissage, et enfin la généralisation et l'inférence. La Figure 6.2 donne un aperçu du processus que nous allons couvrir dans ce chapitre.

Ce chapitre vous fera passer par chacune de ces trois étapes, en expliquant chacune d'entre elles en détail. Il peut être tentant de sauter des étapes de ce plan lorsque l'on est confronté à un bogue déroutant, mais la grande majorité des fois, j'ai constaté que suivre cette approche de principe est le moyen le plus rapide d'identifier et de corriger les erreurs.

Commençons par valider le flux de données. La façon la plus simple de procéder consiste à prendre un très petit sous-ensemble de données et à vérifier qu'elles peuvent circuler tout au long de votre pipeline.

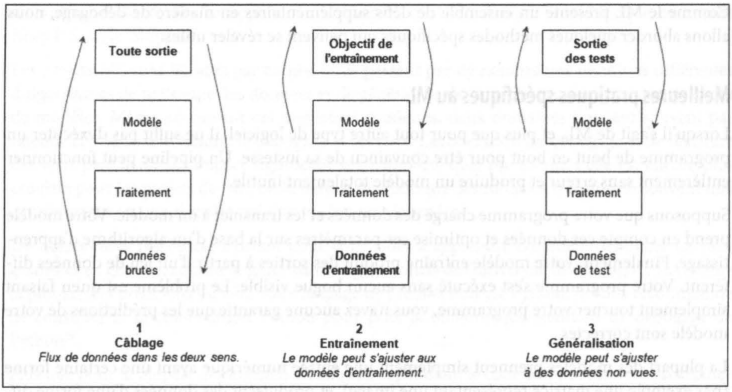


Figure 6.2 : L'ordre dans lequel il faut déboguer un pipeline.

Câbler le débogage : visualisation et test

Cette première étape n'a rien de compliqué et elle vous simplifiera considérablement la vie une fois que vous l'aurez adoptée : commencez par faire fonctionner vos pipelines pour un petit sous-ensemble d'échantillons pris dans votre jeu de données. Cela correspond à l'étape de gauche sur la Figure 6.2. Une fois que vous vous serez assuré que votre pipeline fonctionne pour quelques échantillons, vous pourrez faire des tests pour vérifier qu'il continue d'en être de même au fur et à mesure des modifications que vous y apporterez.

Commencer par un petit échantillonnage

Le but de cette première étape est de vérifier que vous êtes capable d'ingérer des données, de les transformer dans le bon format, de les transmettre à un modèle et de faire en sorte que ce modèle produise quelque chose de correct. À ce stade, vous ne jugez pas si votre modèle peut apprendre quelque chose, mais seulement si le pipeline peut laisser passer les données.

Concrètement, cela signifie :

- Sélectionner quelques échantillons dans votre jeu de données.
- Obtenir que votre modèle produise une prédiction pour ces échantillons.

- Faire en sorte que votre modèle mette à jour ses paramètres pour produire des prévisions correctes pour ces échantillons.

Les deux premiers points visent à vérifier que notre modèle peut ingérer des données en entrée et produire un résultat d'apparence raisonnable. Ce premier résultat sera très probablement erroné du point de vue de la modélisation, mais il nous permettra de vérifier que les données circulent bien.

Le dernier point a pour but de s'assurer que notre modèle a la capacité d'apprendre une correspondance à partir d'une certaine entrée vers la sortie associée. L'ajustement de quelques points de données ne produira pas un modèle utile et conduira probablement à un surajustement. Ce processus nous permet simplement de valider le fait que le modèle peut mettre à jour ses paramètres pour s'adapter à un ensemble d'entrées et de sorties.

Voici à quoi ressemblerait cette première étape en pratique : si vous entraînez un modèle pour prédire si des campagnes Kickstarter seront couronnées de succès, vous envisagez peut-être de vous servir de toutes les campagnes menées ces dernières années. En suivant ce conseil, vous devriez commencer par vérifier si votre modèle est capable de produire une prédiction pour deux campagnes. Ensuite, utilisez l'étiquette de ces campagnes (qu'elles aient été réussies ou non) pour optimiser les paramètres du modèle jusqu'à ce qu'il prédise le bon résultat.

Si nous avons choisi notre modèle de manière appropriée, il devrait avoir la capacité de tirer des enseignements à partir de notre jeu de données. Et si notre modèle peut tirer des enseignements du jeu de données entier, il devrait être capable de mémoriser un point de données. La capacité d'apprendre à partir de quelques échantillons est une condition nécessaire pour qu'un modèle puisse apprendre d'un jeu de données complet. C'est également beaucoup plus facile à valider que l'ensemble du processus d'apprentissage, de sorte que le fait de commencer par un échantillon nous permet de réduire rapidement les éventuels problèmes futurs.

La grande majorité des erreurs qui peuvent survenir à ce stade initial sont liées à des problèmes de non-concordance des données : les données que vous chargez et prétraitez sont injectées dans votre modèle dans un format qu'il ne peut pas accepter. Comme la plupart des modèles n'acceptent que des valeurs numériques, par exemple, ils peuvent échouer lorsqu'une valeur donnée est laissée vide et a une valeur nulle.

Certains cas d'inadéquation peuvent être plus insaisissables et conduire à un échec silencieux. Un pipeline alimenté par des valeurs qui ne sont pas dans la bonne plage ou forme peut quand même fonctionner, mais produirait un modèle peu performant. Les modèles qui nécessitent des données normalisées s'entraîneront quand même souvent sur des données non normalisées : ils ne pourront alors tout simplement pas les ajuster de manière utile. De même, le fait d'alimenter un modèle avec une matrice de forme incorrecte peut entraîner une mauvaise interprétation des données passées en entrée et produire des sorties incorrectes.

Il est plus difficile de détecter de telles erreurs, car elles se manifesteront plus tard dans le processus, une fois que nous aurons évalué les performances d'un modèle. La meilleure façon de les détecter de manière proactive est de visualiser les données au fur et à mesure que vous construisez votre pipeline et d'élaborer des tests pour encoder les suppositions. Nous allons voir ensuite comment faire.

Étapes de visualisation

Comme nous l'avons vu dans les chapitres précédents, si les métriques sont un élément crucial du travail de modélisation, il est tout aussi important d'inspecter et d'examiner régulièrement nos données. En observant quelques échantillons pour commencer, il est plus facile de remarquer les changements ou les incohérences.

L'objectif de ce processus est d'inspecter les changements à intervalles réguliers. Si vous considérez un pipeline de données comme une chaîne de montage, vous voudrez inspecter le produit *après chaque changement significatif*. Cela signifie que vérifier la valeur de votre point de données à chaque ligne est probablement un rythme trop fréquent, et que le fait de ne regarder que les valeurs d'entrée et de sortie n'est définitivement pas assez instructif.

Sur la Figure 6.3, j'illustre quelques exemples de points d'inspection que vous pourriez utiliser pour examiner un pipeline de données. Dans ce schéma, nous inspectons les données lors de plusieurs étapes, en partant des données brutes jusqu'aux résultats produits par le modèle.

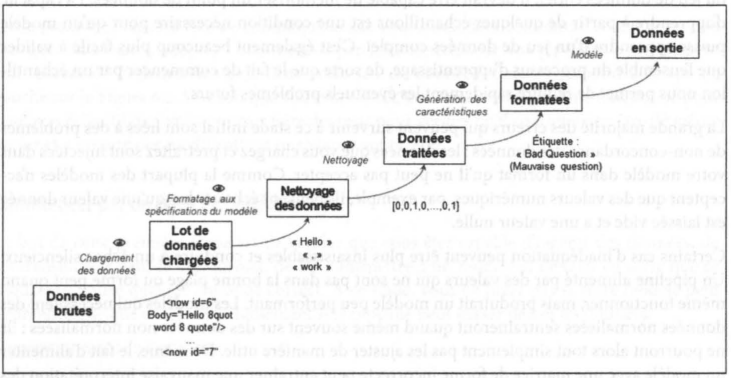


Figure 6.3 : Points d'inspection potentiels.

Nous allons ensuite aborder quelques étapes clés qui valent souvent la peine d'être inspectées. Nous commencerons par le chargement des données, puis nous passerons au nettoyage, à la génération de caractéristiques, au formatage et aux sorties des modèles.

Chargement des données

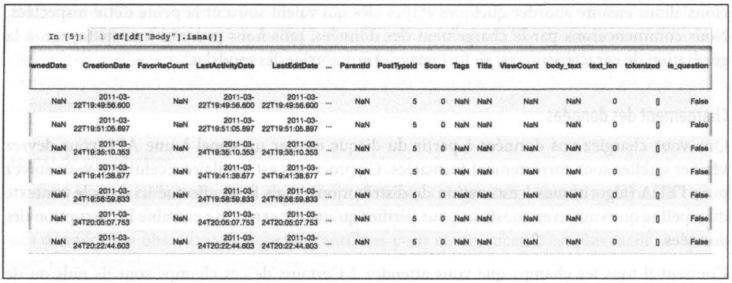
Que vous chargiez vos données à partir du disque ou par un appel à une API, vous devrez vérifier qu'elles sont correctement formatées. Ce processus est similaire à celui que vous suivez pour l'EDA (algorithmes à estimation de distribution), mais il est effectué ici dans le contexte du pipeline que vous avez construit pour vérifier qu'aucune erreur n'a entraîné la corruption des données.

Contient-il tous les champs que vous attendez ? Certains de ces champs sont-ils nuls ou de valeur constante ? Certaines des valeurs se trouvent-elles dans une fourchette qui semble incorrecte, par exemple une variable d'âge qui est parfois négative ? Si vous travaillez avec du texte, de la parole ou des images, les exemples correspondent-ils à vos attentes quant à l'aspect, au son ou à la lecture ?

La plupart de nos étapes de traitement reposent sur des hypothèses que nous faisons sur la structure de nos données d'entrée, il est donc crucial de valider cet aspect.

Comme l'objectif est ici d'identifier les incohérences entre nos attentes et la réalité, vous voudrez peut-être visualiser plus d'un ou deux points de données. La visualisation d'un échantillon représentatif nous permettra de nous assurer que nous n'observons pas seulement un échantillon « chanceux » et que nous supposons alors à tort que tous les points de données sont de la même qualité.

La Figure 6.4 montre un exemple pour notre étude de cas à partir du notebook d'exploration du jeu de données que vous trouvez dans le dépôt GitHub de ce livre (voir `dataset_exploration.ipynb`). Ici, des centaines de messages dans nos archives sont de type non documenté et doivent donc être filtrés. Sur la figure, vous pouvez voir les lignes avec un champ PostTypeId de 5, qui n'est pas référencé dans la documentation du jeu de données, et que nous retirons donc des données d'entraînement.



```
In [5]: df[df["Body"].isna()]
```

wnedDate	CreationDate	FavoriteCount	LastActivityDate	LastEditDate	...	ParentId	PostTypeId	Score	Tags	Title	ViewCount	body_text	text_len	tokenized	is_question
NaN	2011-03-22T19:49:56.600	NaN	2011-03-22T19:49:56.600	2011-03-22T19:49:56.600	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-22T19:51:05.897	NaN	2011-03-22T19:51:05.897	2011-03-22T19:51:05.897	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T19:35:10.353	NaN	2011-03-24T19:35:10.353	2011-03-24T19:35:10.353	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T19:41:38.677	NaN	2011-03-24T19:41:38.677	2011-03-24T19:41:38.677	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T19:58:59.833	NaN	2011-03-24T19:58:59.833	2011-03-24T19:58:59.833	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T20:05:07.753	NaN	2011-03-24T20:05:07.753	2011-03-24T20:05:07.753	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T20:22:44.603	NaN	2011-03-24T20:22:44.603	2011-03-24T20:22:44.603	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False

Figure 6.4 : Visualisation de quelques lignes de données.

Une fois que vous avez vérifié que les données sont conformes aux attentes énoncées dans la documentation fournie avec le jeu de données, il est temps de commencer à les traiter à des fins de modélisation. Cela commence par un bon nettoyage.

Nettoyage et sélection des caractéristiques

L'étape suivante dans la plupart des pipelines consiste à supprimer toute information inutile. Il peut s'agir de champs ou de valeurs qui ne seront pas utilisés par le modèle, mais aussi de tous les champs pouvant contenir des informations sur notre étiquette, mais auxquelles notre modèle n'aurait pas accès en production (voir la section « Partager votre jeu de données » du Chapitre 5).

N'oubliez pas que chaque caractéristique que vous supprimez est un prédicteur potentiel pour votre modèle. La tâche consistant à décider quelles caractéristiques conserver et lesquelles supprimer est appelée la *sélection de caractéristiques* et fait partie intégrante de l'itération sur les modèles.

Vous devriez vérifier qu'aucune information cruciale n'est perdue, que toutes les valeurs inutiles sont supprimées, et que vous n'avez pas laissé dans le jeu de données des informations superflues qui augmenteraient artificiellement les performances de notre modèle à cause de fuites d'informations (voir la section « Fuites de données » dans le Chapitre 5).

Une fois les données nettoyées, vous voudrez générer quelques caractéristiques que votre modèle pourra utiliser.

Génération de caractéristiques

Lors de la création d'une nouvelle caractéristique, comme par exemple l'ajout à un nom de produit de la fréquence correspondante des références dans la description d'une campagne Kickstarter, il est important de vérifier ses valeurs. Vous devez vous assurer que les valeurs de la carac-

téristique sont renseignées, et que ces valeurs semblent raisonnables. C'est une tâche difficile, car il faut non seulement identifier toutes les caractéristiques, mais aussi estimer des valeurs raisonnables pour chacune d'entre elles.

À ce stade, il n'est pas nécessaire de procéder à une analyse plus en profondeur, car cette étape se concentre sur la validation des hypothèses concernant les données qui circulent dans le modèle, et non pour l'instant sur l'utilité de ces données ou du modèle.

Une fois que les caractéristiques ont été générées, vous devriez vous assurer qu'elles peuvent être transmises au modèle dans un format qu'il peut comprendre.

Formatage des données

Comme nous l'avons vu dans les chapitres précédents, avant de passer des points de données à un modèle, vous devrez les transformer dans un format qu'il peut comprendre. Cela peut inclure la normalisation des valeurs d'entrée, la vectorisation du texte en le représentant de manière numérique, ou le formatage d'une vidéo en noir et blanc en tant que tenseur 3D (voir la section « Vectorisation » du Chapitre 4).

Si vous travaillez sur un problème supervisé, vous utiliserez une étiquette en plus de l'entrée, comme les noms de classes dans la classification, ou une carte de segmentation dans la segmentation d'images. Il faudra également les transformer en un format compréhensible par le modèle.

D'après mon expérience de travail sur les problèmes de segmentation d'images, par exemple, le décalage de données entre les étiquettes et les prédictions des modèles est l'une des causes d'erreurs les plus fréquentes. Ces modèles utilisent des masques de segmentation comme étiquettes. Ces masques sont de la même taille que l'image d'entrée, mais, au lieu de valeurs de pixels, ils contiennent des étiquettes de classe pour chaque pixel. Malheureusement, différentes bibliothèques utilisent des conventions différentes pour représenter ces masques, de sorte que les étiquettes se retrouvent souvent dans un format erroné, ce qui empêche le modèle d'apprendre.

J'ai illustré ce piège courant sur la Figure 6.5. Supposons qu'un modèle s'attende à ce que les masques de segmentation soient passés avec une valeur de 255 pour les pixels qui sont d'une certaine classe, et 0 sinon. Si un utilisateur suppose de son côté que les pixels contenus dans le masque doivent avoir une valeur de 1 au lieu de 255, il peut transmettre ses masques étiquetés dans le format indiqué dans la représentation « fourni » de la figure. Cela conduirait à considérer le masque comme étant presque entièrement vide, et le modèle produirait alors des prédictions inexactes.

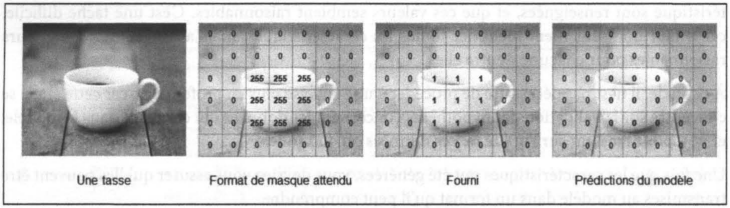


Figure 6.5 : Des étiquettes mal formatées empêcheront un modèle d'apprendre.

De même, les étiquettes de classification sont souvent représentées sous la forme d'une liste de zéros avec un unique un à la position de l'indice de la classe vraie. Une simple erreur de décalage d'une seule position peut conduire à répercuter ce décalage sur toutes les étiquettes. Dans ce cas, un modèle apprendrait à toujours prédire l'étiquette ainsi décalée. Ce type d'erreur peut être difficile à résoudre si vous ne prenez pas le temps d'examiner vos données.

Comme les modèles de ML parviennent à s'adapter à la plupart des sorties numériques, qu'elles aient une structure ou un contenu précis, c'est à ce stade que se produisent de nombreux bogues délicats et que cette méthode est utile pour les trouver.

Voici un exemple de ce à quoi ressemble une telle fonction de formatage pour notre étude de cas. Je génère une représentation vectorisée de notre texte de question. Ensuite, j'ajoute des caractéristiques supplémentaires à cette représentation. Comme la fonction consiste en de multiples transformations et opérations vectorielles, visualiser sa valeur de retour me permettra de vérifier qu'elle formate bien les données comme nous le souhaitons.

```
def get_feature_vector_and_label(df, feature_names):
    """
    Génère des vecteurs d'entrée et de sortie en utilisant
    la caractéristique vectors et les noms de caractéristiques donnés
    :paramètre df: DataFrame en entrée
    :paramètre feature_names: noms des colonnes de caractéristiques
                              (autres que vectors)
    :retourne: tableau de caractéristiques et tableau d'étiquettes
    """
    vec_features = vstack(df["vectors"])
    num_features = df[feature_names].astype(float)
    features = hstack([vec_features, num_features])
    labels = df["Score"] > df["Score"].median() return features, labels

    features = [
        "action_verb_full",
        "question_mark_full",
        "text_len", "language_question",
    ]

    X_train, y_train = get_feature_vector_and_label(train_df, features)
```

Lorsque l'on travaille avec des données textuelles en particulier, il y a généralement plusieurs étapes à franchir avant que ces données ne soient correctement formatées pour un modèle. Passer d'une chaîne de texte à une liste de tokens et à une représentation vectorisée incluant d'éventuelles caractéristiques supplémentaires est un processus sujet aux erreurs. Une simple inspection de la forme des objets à chaque étape peut aider à détecter de nombreuses erreurs simples.

Une fois que les données sont dans le format approprié, vous pouvez les transmettre à un modèle. La dernière étape consiste à visualiser et à valider les sorties du modèle.

Sorties du modèle

Dans un premier temps, l'examen des sorties nous aide à voir si les prédictions de notre modèle sont du bon type ou ont la forme requise (si nous prédisons le prix d'une maison et la durée de l'annonce sur le marché, notre modèle produit-il un tableau de deux chiffres ?).

En outre, lorsqu'on ajuste un modèle sur seulement quelques points de données, on devrait voir ses résultats commencer à correspondre à l'étiquette vraie. Si le modèle ne correspond pas aux points de données, cela peut indiquer que les données sont mal formatées ou qu'elles sont corrompues.

Si les résultats du modèle ne changent pas du tout pendant l'entraînement, cela peut signifier que notre modèle n'exploite pas réellement les données qui lui sont passées en entrée. Dans un tel cas, je recommande de se référer à la section « Se tenir sur les épaules des géants », dans le Chapitre 2, pour valider que le modèle est utilisé correctement.

Une fois que nous aurons parcouru tout le pipeline pour une première série d'échantillons, il sera temps d'écrire quelques tests pour automatiser une partie de ce travail de visualisation.

Systématiser notre validation visuelle

Le travail de visualisation décrit plus haut permet de détecter une quantité importante de bogues et constitue un bon investissement en temps pour chaque nouveau pipeline. La validation des hypothèses sur la façon dont les données circulent dans le modèle permet de gagner un temps considérable en aval, temps qui peut maintenant être consacré à l'entraînement et à la généralisation.

Cependant, les pipelines changent souvent. Alors que vous mettez à jour différents aspects de manière itérative pour améliorer votre modèle et modifier une partie de la logique de traitement, comment pouvez-vous garantir que tout fonctionne toujours comme prévu ? Traverser le pipeline et visualiser un échantillon à toutes les étapes, et ce à chaque fois que vous effectuez un changement, deviendrait rapidement fatigant.

C'est là que les meilleures pratiques en matière de génie logiciel dont nous avons parlé plus tôt entrent en jeu. Il est temps d'isoler chaque partie de ce pipeline, et d'encoder nos observations sous forme de tests que nous pourrons exécuter au fur et à mesure de l'évolution de notre pipeline pour le valider.

Scinder vos préoccupations

Tout comme les logiciels classiques, le ML bénéficie grandement d'une organisation modulaire. Pour faciliter le débogage actuel et futur, scindez chaque fonction afin de pouvoir vérifier qu'elle fonctionne individuellement avant d'examiner le pipeline plus large.

Une fois qu'un pipeline est décomposé en différentes fonctions, vous serez à même d'écrire des tests pour celles-ci.

Tester votre code ML

Il est difficile de tester le comportement d'un modèle. Cependant, la majorité du code d'un pipeline ML ne concerne pas le pipeline d'entraînement ou le modèle lui-même. Si vous revenez à notre exemple de pipeline dans la section « Commencer par un pipeline simple » du Chapitre 2, vous remarquerez que la plupart des fonctions se comportent de manière déterministe et peuvent donc être testées.

Selon mon expérience, j'ai appris, en aidant des ingénieurs et des spécialistes des données à déboguer leurs modèles, que la grande majorité des erreurs provient de la façon dont les données sont acquises, traitées ou injectées dans le modèle. Tester la logique de traitement des données est donc crucial pour construire un produit ML réussi.

Pour encore plus d'informations sur les tests potentiels d'un système de ML, je vous recommande l'article de E. Breck *et al.* intitulé « *The ML Test Score» : A Rubric for ML Production Readiness and Technical Debt Reduction* »[40], qui contient de nombreux autres exemples et leçons tirées du déploiement de tels systèmes chez Google.

Dans la prochaine section, nous allons décrire des tests utiles à écrire dans trois domaines clés. Sur la Figure 6.6, vous pouvez voir chacun de ces domaines, ainsi que quelques exemples des tests que nous allons décrire ensuite.

Les pipelines commencent par ingérer des données, et nous allons donc vouloir tester cette partie en premier.

Tester l'ingestion des données

Les données existent généralement sous une forme sérialisée sur un disque ou dans une base de données. Lorsque nous transférons des données du stockage vers notre pipeline, nous devons nous assurer de leur intégrité et de leur correction. Nous pouvons commencer par écrire des tests qui vérifient que les points de données que nous chargeons possèdent toutes les caractéristiques dont nous aurons besoin.

40 Voir l'adresse `https://research.google/pubs/pub46555/`.

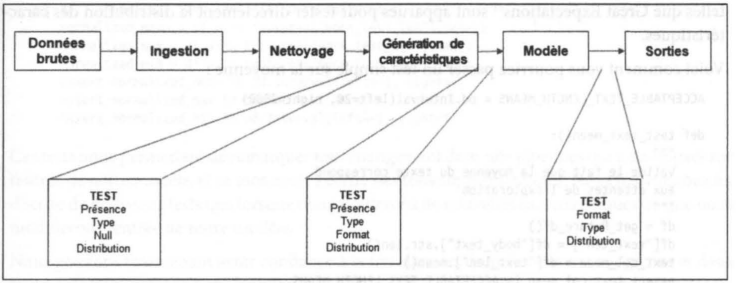


Figure 6.6 : Les trois domaines clés à tester.

Voici trois tests servant à vérifier que notre analyseur renvoie le bon type (un DataFrame), que toutes les colonnes importantes sont définies, et que les caractéristiques ne sont pas toutes nulles. Vous pouvez trouver les tests que nous allons couvrir dans ce chapitre (et d'autres) dans le dossier `tests` du dépôt GitHub de ce livre.

```
def test_parser_returns_dataframe():
    """
    Teste que notre analyseur fonctionne et renvoie un DataFrame
    """
    df = get_fixture_df()
    assert isinstance(df, pd.DataFrame)


def test_feature_columns_exist():
    """
    Valide que toutes les colonnes requises sont présentes
    """
    df = get_fixture_df()
    for col in REQUIRED_COLUMNS:
        assert col in df.columns


def test_features_not_all_null():
    """
    Valide qu'aucune caractéristique n'est entièrement vide
    """
    df = get_fixture_df()
    for col in REQUIRED_COLUMNS:
        assert not df[col].isnull().all()
```

Nous pouvons également tester chaque caractéristique pour son type et valider qu'elle n'est pas nulle. Enfin, nous pouvons encoder nos hypothèses sur la distribution et les plages de ces valeurs en testant leurs valeurs moyennes, minimales et maximales. Récemment, des bibliothèques

telles que Great Expectations[41] sont apparues pour tester directement la distribution des caractéristiques.

Voici comment vous pourriez passer un test simple sur la moyenne :

```
ACCEPTABLE_TEXT_LENGTH_MEANS = pd.Interval(left=20, right=2000)

def test_text_mean():
    """
    Valide le fait que la moyenne du texte correspond
    aux attentes de l'exploration
    """
    df = get_fixture_df()
    df["text_len"] = df["body_text"].str.len()
    text_col_mean = df["text_len"].mean()
    assert text_col_mean in ACCEPTABLE_TEXT_LENGTH_MEANS
```

Ces tests nous permettent de vérifier que, quelles que soient les modifications apportées au niveau du stockage ou de l'API de notre source de données, nous pouvons savoir que notre modèle a accès au même type de données que celui sur lequel il a été entraîné en premier. Une fois que nous sommes sûrs de la cohérence des données que nous ingérons, examinons l'étape suivante, le traitement des données.

Tester le traitement des données

Après avoir vérifié que les données qui parviennent à l'entrée de notre pipeline sont conformes à nos attentes, nous devrions vérifier que nos étapes de nettoyage et de génération de caractéristiques font ce que nous en attendons. Nous pouvons commencer par écrire des tests pour la fonction de prétraitement dont nous disposons, en vérifiant qu'elle fait effectivement ce que nous attendons d'elle. Nous pouvons également écrire des tests similaires à ceux de l'ingestion de données et nous concentrer sur le fait de garantir que nos hypothèses sur l'état des données entrant dans notre modèle sont valides.

Cela signifie qu'il faut vérifier la présence, le type et les caractéristiques des points de données après notre pipeline de traitement. Voici des exemples de tests pour la présence des caractéristiques générées, leur type et les valeurs minimales, maximales et moyennes :

```
def test_feature_presence(df_with_features):
    for feat in REQUIRED_FEATURES:
        assert feat in df_with_features.columns

def test_feature_type(df_with_features):
    assert df_with_features["is_question"].dtype == bool assert
    df_with_features["action_verb_full"].dtype == bool assert
    df_with_features["language_question"].dtype == bool
    assert df_with_features["question_mark_full"].dtype == bool assert
    df_with_features["norm_text_len"].dtype == float assert
    df_with_features["vectors"].dtype == list
```

41 Voir l'adresse https://github.com/great-expectations/great_expectations.

```
def test_normalized_text_length(df_with_features):
    normalized_mean = df_with_features["norm_text_len"].mean()
    normalized_max = df_with_features["norm_text_len"].max()
    normalized_min = df_with_features["norm_text_len"].min()
    assert normalized_mean in pd.Interval(left=-1, right=1)
    assert normalized_max in pd.Interval(left=-1, right=1)
    assert normalized_min in pd.Interval(left=-1, right=1)
```

Ces tests nous permettent de remarquer tout changement dans nos pipelines qui a un impact sur l'entrée de notre modèle, et ce sans avoir à écrire des tests supplémentaires. Nous n'aurons besoin d'écrire de nouveaux tests que lorsque nous ajouterons de nouvelles caractéristiques ou que nous modifierons l'entrée de notre modèle.

Nous pouvons maintenant avoir confiance à la fois dans les données que nous ingérons et dans les transformations que nous leur appliquons. Il est donc temps de tester la prochaine partie du pipeline, le modèle.

Tester les sorties du modèle

Comme pour les deux catégories précédentes, nous allons écrire des tests pour valider le fait que les valeurs produites par le modèle ont les dimensions et les plages correctes. Nous testerons également les prédictions pour des entrées spécifiques. Cela permet de détecter de manière proactive les régressions dans la qualité des prédictions pour les nouveaux modèles, et de garantir que tout modèle que nous utilisons produit toujours les résultats attendus sur ces entrées d'échantillons. Lorsqu'un nouveau modèle montre de meilleures performances agrégées, il peut être difficile de remarquer si cette performance s'est en fait détériorée sur des types d'entrées spécifiques. La réalisation de tels tests aide à détecter plus facilement ce genre de problèmes.

Dans les exemples suivants, je commence par tester la forme des prédictions de notre modèle, ainsi que leurs valeurs. Le troisième test vise à prévenir les régressions en garantissant que le modèle classe une question en entrée spécifique qui est mal formulée comme étant de faible qualité.

```
def test_model_prediction_dimensions(
    df_with_features, trained_v1_vectorizer, trained_v1_model
):
    df_with_features["vectors"] = get_vectorized_series(
        df_with_features["full_text"].copy(), trained_v1_vectorizer
    )

    features, labels = get_feature_vector_and_label(
        df_with_features, FEATURE_NAMES
    )

    probas = trained_v1_model.predict_proba(features)
    # le modèle fait une prédiction par échantillon d'entrée
    assert probas.shape[0] == features.shape[0]
```

```
    # le modèle prédit des probabilités pour deux classes
    assert probas.shape[1] == 2


def test_model_proba_values(
    df_with_features, trained_v1_vectorizer, trained_v1_model
):
    df_with_features["vectors"] = get_vectorized_series(
        df_with_features["full_text"].copy(), trained_v1_vectorizer
    )

    features, labels = get_feature_vector_and_label(
        df_with_features, FEATURE_NAMES
    )

    probas = trained_v1_model.predict_proba(features)
    # les probabilités du modèle sont comprises entre 0 et 1
    assert (probas >= 0).all() and (probas <= 1).all()


    def test_model_predicts_no_on_bad_question():
    input_text = "This isn't even a question. We should score it poorly"
    is_question_good = get_model_predictions_for_input_texts([input_text])
    # Le modèle classifie la question comme étant mauvaise
    assert not is_question_good[0]
```

Nous avons d'abord procédé à une inspection visuelle des données pour vérifier qu'elles restaient utiles et utilisables tout au long de notre pipeline. Ensuite, nous avons rédigé des tests pour garantir que ces hypothèses restent correctes au fur et à mesure de l'évolution de notre stratégie de traitement. Il est maintenant temps d'aborder la deuxième partie de la Figure 6.2, le débogage de la procédure d'entraînement.

Débogage de l'entraînement : faire apprendre votre modèle

Une fois que vous avez testé votre pipeline et validé qu'il fonctionne pour un échantillon, vous savez certaines choses. Votre pipeline reçoit des données et les transforme avec succès. Il transmet ensuite ces données à un modèle dans le bon format. Enfin, le modèle peut à son tour recevoir quelques points de données et en tirer des enseignements, ce qui lui permet de produire des résultats corrects.

Il est maintenant temps de voir si votre modèle peut fonctionner sur plus que quelques points de données et apprendre à partir de votre jeu d'entraînement. L'objectif de la section suivante est de vous permettre d'entraîner votre modèle sur de nombreux échantillons et de *l'ajuster à toutes vos données d'entraînement*.

Pour cela, vous pouvez désormais faire passer la totalité de votre jeu d'entraînement dans votre modèle et mesurer ses performances. Si vous disposez d'un jeu de grande taille, vous pouvez

également augmenter progressivement la quantité de données que vous introduisez dans votre modèle tout en gardant un œil sur ses performances globales.

L'un des avantages de l'augmentation progressive de la taille de votre jeu de données d'entraînement est que vous pourrez mesurer l'effet des données supplémentaires sur les performances de votre modèle. Commencez par quelques centaines d'échantillons, puis augmentez cette quantité à quelques milliers, avant de passer à l'ensemble de votre jeu de données (si celui-ci contient moins de mille échantillons, n'hésitez pas à passer directement à l'utilisation de la totalité du jeu).

À chaque étape, ajustez votre modèle en fonction des données et évaluez sa performance sur ces *mêmes données*. Si votre modèle a la capacité d'apprendre à partir des données que vous utilisez, sa performance sur les données d'entraînement devrait rester relativement stable.

Pour contextualiser la performance du modèle, je recommande de générer une estimation du niveau d'erreur acceptable dans le cas de votre tâche en étiquetant par exemple vous-même quelques échantillons, et en comparant vos prédictions à l'étiquette vraie. La plupart des tâches s'accompagnent également d'une fraction d'erreur irréductible, représentant la meilleure performance compte tenu de la complexité de ladite tâche. La Figure 6.7 propose une illustration de la performance habituelle de l'entraînement comparée à ces métriques.

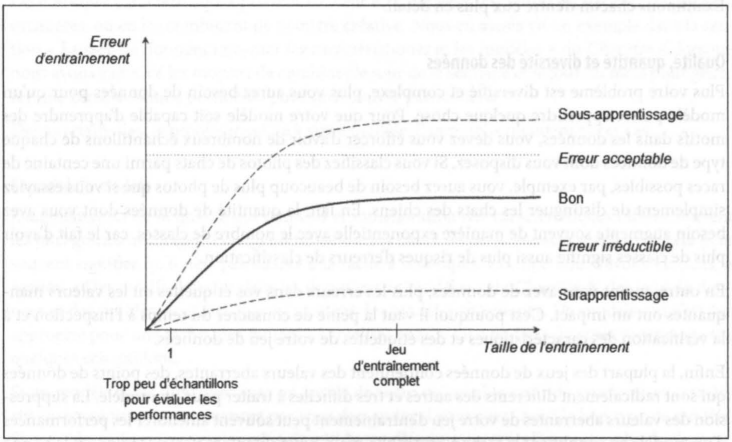


Figure 6.7 : Exactitude de l'entraînement en fonction de la taille du jeu de données.

Les performances d'un modèle sur l'ensemble du jeu de données devraient être pires que lorsqu'on utilise un seul échantillon, car mémoriser un jeu d'entraînement complet est plus difficile qu'avec un seul échantillon, mais elles devraient tout de même rester dans les limites définies précédemment.

Si vous êtes en mesure d'injecter tout votre jeu d'entraînement et que les performances de votre modèle atteignent l'exigence que vous avez définie lors de l'examen de l'objectif poursuivi pour votre produit, n'hésitez pas à passer à la section suivante ! Si ce n'est pas le cas, j'ai exposé dans ce qui suit quelques raisons courantes pour lesquelles un modèle peut avoir des difficultés avec un jeu d'entraînement.

Difficultés de la tâche

Si les performances d'un modèle sont nettement inférieures aux attentes, il se peut que la tâche soit trop difficile. Pour évaluer la difficulté d'une tâche, considérez les points suivants :

- la quantité et la diversité des données dont vous disposez ;
- si les caractéristiques que vous avez générées sont prédictives ;
- la complexité de votre modèle.

Examinons chacun d'entre eux plus en détail.

Qualité, quantité et diversité des données

Plus votre problème est diversifié et complexe, plus vous aurez besoin de données pour qu'un modèle puisse apprendre quelque chose. Pour que votre modèle soit capable d'apprendre des motifs dans les données, vous devez vous efforcer d'avoir de nombreux échantillons de chaque type de données dont vous disposez. Si vous classifiez des photos de chats parmi une centaine de races possibles, par exemple, vous aurez besoin de beaucoup plus de photos que si vous essayiez simplement de distinguer les chats des chiens. En fait, la quantité de données dont vous avez besoin augmente souvent de manière exponentielle avec le nombre de classes, car le fait d'avoir plus de classes signifie aussi plus de risques d'erreurs de classification.

En outre, moins vous avez de données, plus les erreurs dans vos étiquettes ou les valeurs manquantes ont un impact. C'est pourquoi il vaut la peine de consacrer du temps à l'inspection et à la vérification des caractéristiques et des étiquettes de votre jeu de données.

Enfin, la plupart des jeux de données contiennent des valeurs aberrantes, des points de données qui sont radicalement différents des autres et très difficiles à traiter pour un modèle. La suppression des valeurs aberrantes de votre jeu d'entraînement peut souvent améliorer les performances d'un modèle en simplifiant la tâche à accomplir, mais, pour autant, ce n'est pas toujours la bonne approche : si vous pensez que votre modèle peut rencontrer des points de données similaires en

production, vous devriez conserver les valeurs aberrantes et vous concentrer sur *l'amélioration de vos données et de votre modèle* afin que celui-ci puisse s'y adapter avec succès.

Plus un jeu de données est complexe, plus il peut être utile de travailler sur les moyens de représenter vos données de manière à permettre à un modèle d'en tirer plus facilement des enseignements. Voyons ce que cela signifie.

Représentation des données

Est-il facile de détecter les motifs qui vous intéressent dans les données en utilisant uniquement la représentation que vous donnez à votre modèle ? Si un modèle a du mal à bien fonctionner sur les données d'entraînement, vous devriez ajouter des caractéristiques qui rendent ces données plus expressives, et aident ainsi le modèle à mieux apprendre.

Il peut s'agir de caractéristiques nouvelles que nous avions décidé d'ignorer auparavant, mais qui peuvent être prédictives. Dans l'exemple de notre Éditeur ML, une première itération du modèle ne prenait en compte que le texte dans le corps d'une question. Après avoir exploré plus avant le jeu de données, j'ai remarqué que les titres des questions sont souvent très instructifs pour savoir si elles sont bonnes ou non. Le fait de réintégrer cette caractéristique dans le jeu de données a permis au modèle d'être plus performant.

De nouvelles caractéristiques peuvent souvent être générées en réutilisant des caractéristiques existantes, ou en les combinant de manière créative. Nous en avons vu un exemple dans la section « Laisser les données informer les caractéristiques et les modèles » du Chapitre 4, lorsque nous avons examiné les moyens de combiner le jour de la semaine et le jour du mois pour générer une caractéristique pertinente pour une analyse particulière.

Dans certains cas, le problème est lié à votre modèle. Examinons maintenant ces cas.

Capacité du modèle

L'augmentation de la qualité des données et l'amélioration des caractéristiques offrent souvent les plus grands avantages. Lorsqu'un modèle est la cause de performances médiocres, cela peut souvent signifier qu'il n'est pas adapté à la tâche à accomplir. Comme nous l'avons vu dans la section « Des motifs détectés dans les données aux modèles » du Chapitre 5, des jeux de données et des problèmes spécifiques nécessitent des modèles spécifiques. Un modèle qui n'est pas approprié pour une tâche aura du mal à la réaliser, même s'il a pu faire du surapprentissage sur quelques échantillons.

Si un modèle a des difficultés avec un jeu de données qui semble avoir de nombreuses caractéristiques prédictives, commencez par vous demander si vous avez choisi le bon type de modèle. Si possible, utilisez une version plus simple du modèle pour pouvoir l'inspecter plus facilement. Par exemple, si un modèle de forêt aléatoire ne fonctionne pas du tout, essayez un arbre de

décision sur la même tâche et visualisez ses partages de données pour voir s'il utilise les caractéristiques que vous pensiez prédictives.

D'autre part, le modèle que vous utilisez est peut-être trop simple. Il est bon de commencer par le modèle le plus simple pour pouvoir itérer rapidement, mais il y a des tâches qui sont totalement hors de portée de certains modèles. Pour les effectuer, vous devrez peut-être ajouter de la complexité à votre modèle. Pour vérifier qu'un modèle est bien adapté à une tâche, je vous recommande de consulter l'état de la technique, comme nous l'avons décrit dans la section « Se tenir sur les épaules des géants » du Chapitre 2. Trouvez des exemples de tâches similaires, et examinez quels modèles ont été utilisés pour les réaliser. L'utilisation d'un de ces modèles devrait être un bon point de départ.

Si le modèle semble approprié à la tâche, son manque de performance pourrait être dû à la procédure d'entraînement.

Problèmes d'optimisation

Commencer par valider le fait qu'un modèle peut s'adapter à un petit ensemble d'échantillons nous rend confiants que les données peuvent circuler dans les deux sens. Nous ne savons cependant pas si notre procédure d'entraînement peut ajuster correctement un modèle sur l'ensemble du jeu de données. La méthode que notre modèle utilise pour mettre à jour ses poids peut être inadéquate pour notre jeu de données actuel. De tels problèmes surviennent souvent dans le cas de modèles plus complexes tels que les réseaux de neurones, où le choix des hyperparamètres peut avoir un impact significatif sur les performances de l'entraînement.

Lorsqu'il s'agit de modèles qui sont ajustés à l'aide de techniques de descente de gradient, comme les réseaux de neurones, l'utilisation d'outils de visualisation tels que TensorBoard peut aider à faire ressortir les problèmes d'entraînement. Lorsque vous tracez la perte au cours de votre processus d'optimisation, vous devriez voir celle-ci diminuer fortement dans un premier temps, puis de manière progressive. La Figure 6.8 présente un exemple de tableau de bord TensorBoard illustrant une fonction de perte (en l'occurrence, une entropie croisée) au fur et à mesure que l'entraînement progresse.

Une telle courbe peut montrer que la perte diminue très faiblement au fil du temps, ce qui indique qu'un modèle apprend peut-être trop lentement. Dans un tel cas, vous pourriez augmenter le taux d'apprentissage et retracer la même courbe pour voir si la perte diminue plus rapidement. Si une courbe de perte semble très instable, en revanche, elle peut être due à un taux d'apprentissage trop élevé.

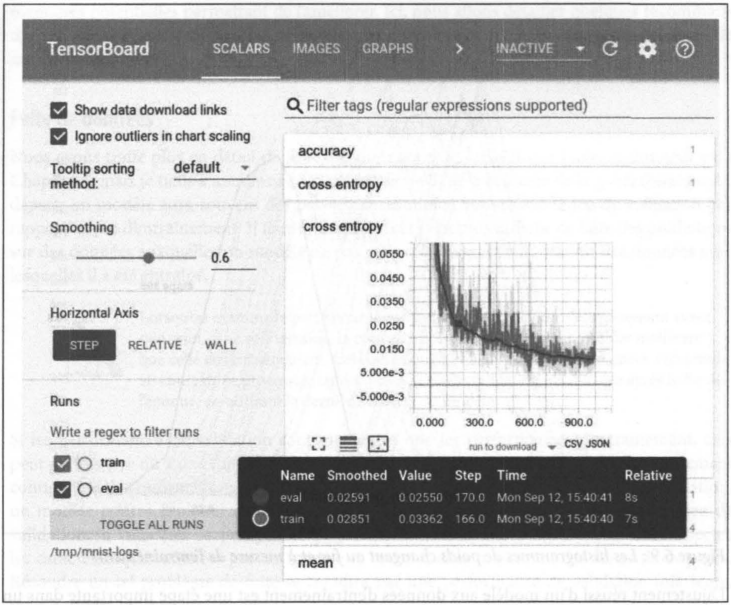


Figure 6.8 : Capture d'écran du tableau de bord TensorBoard (extrait de la documentation de TensorBoard).

En plus de la perte, la visualisation des valeurs de poids et des activations peut vous aider à identifier si un réseau n'apprend pas correctement. Sur la Figure 6.9, vous pouvez voir un changement dans la distribution des poids au fur et à mesure que l'entraînement progresse. Si vous voyez que les distributions restent stables pendant quelques époques, cela peut être un signe que vous devriez augmenter le taux d'apprentissage. Si elles varient trop, il faut plutôt le diminuer.

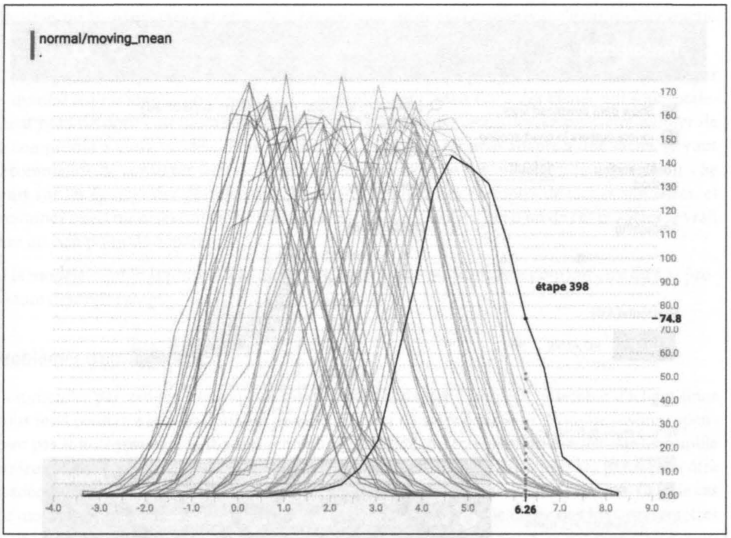


Figure 6.9 : Les histogrammes de poids changent au fur et à mesure de l'entraînement.

L'ajustement réussi d'un modèle aux données d'entraînement est une étape importante dans un projet de ML, mais ce n'est pas la dernière. L'objectif final de la construction d'un produit de ML est d'obtenir un modèle qui puisse fonctionner correctement sur des échantillons qu'il n'a jamais vus auparavant. Pour cela, nous avons besoin d'un modèle capable de bien se généraliser à des échantillons inconnus, sujet que je vais maintenant aborder.

Déboguer la généralisation : rendre votre modèle utile

La généralisation est la troisième et dernière partie de la Figure 6.2. Elle vise à faire en sorte qu'un modèle de ML fonctionne bien sur des données qu'il n'a jamais vues auparavant. Dans la section « Partager votre jeu de données » du Chapitre 5, nous avons vu l'importance de la création de jeux d'entraînement, de validation et de test séparés pour évaluer la capacité d'un modèle à se généraliser à partir d'échantillons non encore vus. Dans la section « Évaluer votre modèle : aller au-delà de l'exactitude », toujours dans le Chapitre 5, nous avons abordé les méthodes permettant d'analyser les performances d'un modèle et d'identifier les caractéristiques supplé-

mentaires potentielles permettant de l'améliorer. Ici, nous allons détailler quelques recommandations dans le cas où un modèle ne fonctionne toujours pas sur le jeu de validation après de multiples itérations.

Fuite de données

Nous avons traité plus en détail de fuite des données dans la section « Fuite de données » du Chapitre 5, mais je tiens à mentionner ici ce problème dans le contexte de la généralisation. Au départ, un modèle aura souvent des performances moins bonnes sur le jeu de validation par rapport au jeu d'entraînement. Il faut s'y attendre, car il est plus difficile de faire des prédictions sur des données auxquelles un modèle n'a pas été exposé auparavant que sur des données avec lesquelles il a été entraîné.

Lorsqu'on examine la perte pour la validation et la perte pour l'entraînement avant que celui-ci ne soit terminé, la performance de validation peut sembler meilleure que celle de l'entraînement. Cela est dû au fait que la perte d'entraînement s'accumule au cours de ce processus, tandis que la perte de validation est calculée après la fin de l'époque, en utilisant la dernière version du modèle.

Si les performances de validation sont meilleures que les performances d'entraînement, cela peut parfois être dû à une fuite de données. Si les échantillons dans les données d'entraînement contiennent des informations sur d'autres échantillons présents dans les données de validation, un modèle pourra exploiter ces informations, et obtenir ainsi de bons résultats sur le jeu de validation. Si vous êtes surpris par les performances réalisées lors de la validation, inspectez les caractéristiques utilisées par un modèle et voyez si elles présentent des fuites de données. Résoudre un tel problème de fuite se traduira par une performance de validation inférieure, mais donnera un meilleur modèle.

Une fuite de données peut nous amener à croire qu'un modèle se généralise alors que ce n'est pas vraiment le cas. Dans d'autres situations, il est clair, à l'examen des performances d'un jeu de validation, que le modèle ne fonctionne bien que pour l'entraînement. Cela peut signifier que le modèle est victime de surapprentissage.

Surapprentissage

Dans la section « Compromis entre biais et variance » du Chapitre 5, nous avons vu que, lorsqu'un modèle a du mal à s'ajuster aux données d'entraînement, nous disons que le modèle est victime de sous-apprentissage. Nous avons également vu que l'opposé du *sous-apprentissage* est le *surapprentissage*, qui se produit lorsque notre modèle s'ajuste trop bien à nos données d'entraînement.

Que signifient des « données trop bien ajustées » ? Cela veut dire qu'au lieu d'apprendre des tendances généralisables, qui sont par exemple en corrélation avec une bonne ou une mauvaise

écriture, un modèle peut mettre en avant des schémas spécifiques qui sont présents dans des échantillons individuels d'un jeu d'entraînement, mais qu'on ne retrouve pas dans des données différentes. Ces schémas l'aident à obtenir un score élevé pour le jeu d'entraînement, mais ils ne sont pas utiles pour classifier d'autres échantillons.

La Figure 6.10 montre un exemple pratique de surapprentissage et de sous-apprentissage pour un jeu de données jouet. Le modèle avec surapprentissage s'ajuste parfaitement aux données d'entraînement, mais il n'approxime pas exactement la tendance sous-jacente. Il ne permet donc pas de prédire avec exactitude ce qui se passe pour les points non vus. Le modèle avec sous-apprentissage ne saisit pas du tout la tendance dans les données. Le modèle qualifié d'ajustement raisonnable est moins performant sur les données d'entraînement que le modèle avec surapprentissage, mais il est plus performant sur les données non vues.

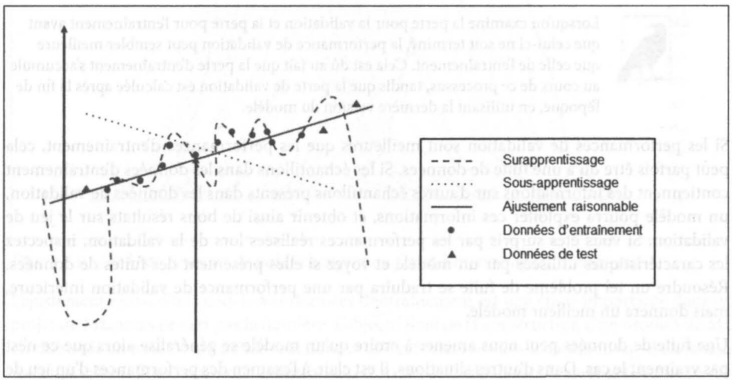


Figure 6.10 : Surapprentissage ou sous-apprentissage.

Lorsqu'un modèle fonctionne beaucoup mieux sur le jeu d'entraînement que sur le jeu de test, cela signifie généralement qu'il y a surapprentissage. Le modèle a appris les détails spécifiques des données d'entraînement, mais il n'est pas capable de fonctionner sur des données non vues.

Comme le surapprentissage est dû au fait qu'un modèle apprend trop sur les données d'entraînement, nous pouvons l'empêcher en réduisant la capacité d'un modèle à apprendre à partir d'un jeu de données. Il existe plusieurs façons de le faire, que nous allons aborder maintenant.

Régularisation

La *régularisation* ajoute une pénalité sur la capacité d'un modèle à représenter l'information. Elle vise à limiter la capacité d'un modèle à se concentrer sur de nombreux schémas non pertinents, et à l'encourager à choisir des caractéristiques moins nombreuses mais plus prédictives.

Une façon courante de régulariser un modèle est d'imposer une pénalité sur la valeur absolue de ses poids. Pour des modèles tels que la régression linéaire et logistique, par exemple, les régularisations L1 et L2 ajoutent un terme supplémentaire à la fonction de perte qui pénalise les poids importants. Dans le cas de L1, ce terme est la somme de la valeur absolue des poids. Pour L2, il est la somme des valeurs au carré des poids.

Les différentes méthodes de régularisation ont des effets différents. La régularisation L1 peut aider à sélectionner des caractéristiques informatives en mettant à zéro les caractéristiques non informatives (pour en savoir plus, consultez la page Wikipédia « Lasso (statistiques) »[42]). La régularisation L1 est également utile lorsque certaines caractéristiques sont corrélées en encourageant le modèle à n'exploiter qu'une seule d'entre elles.

Les méthodes de régularisation peuvent également être spécifiques à un modèle. Les réseaux de neurones utilisent souvent *dropout* comme méthode de régularisation. Dropout ignore de manière aléatoire une certaine proportion des neurones d'un réseau pendant l'entraînement. Cela permet d'éviter qu'un seul neurone ne devienne trop influent, ce qui pourrait permettre au réseau de mémoriser certains aspects des données d'entraînement.

Pour les modèles basés sur les arbres tels que les forêts aléatoires, la réduction de la profondeur maximale des arbres diminue le risque de surapprentissage des données, et permet ainsi de régulariser l'ensemble de la forêt. Augmenter le nombre d'arbres utilisés dans une forêt permet également de la régulariser.

Une autre façon d'éviter le surapprentissage sur les données d'entraînement est de rendre les données elles-mêmes plus difficiles à ajuster. Nous pouvons y parvenir grâce à un processus appelé *augmentation des données*.

Augmentation des données

L'augmentation des données est le processus de création de nouvelles données d'entraînement obtenues en modifiant légèrement les points de données existants. L'objectif est de produire artificiellement des points de données différents de ceux qui existent déjà, afin d'exposer un modèle à un type d'entrée plus varié. Les stratégies d'augmentation dépendent du type des données. Sur la Figure 6.11, vous pouvez voir quelques stratégies potentielles d'augmentation des données dans le cas d'images.

42 Voir l'adresse https://fr.wikipedia.org/wiki/Lasso_(statistiques).

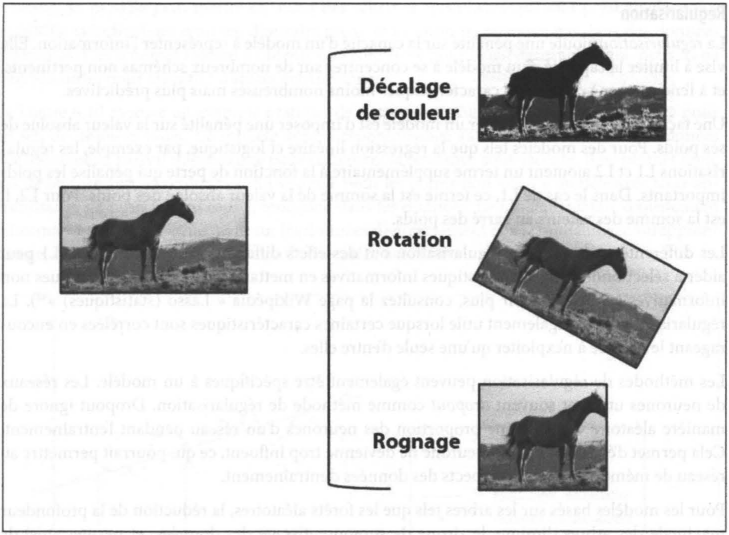


Figure 6.11 : Quelques exemples d'augmentation des données pour les images.

L'augmentation des données rend un jeu d'entraînement moins homogène, et donc plus complexe. L'ajustement des données d'entraînement devient ainsi plus difficile, mais cela expose un modèle à un plus grand nombre d'entrées pendant l'entraînement. L'augmentation des données se traduit souvent par des performances plus faibles sur le jeu d'entraînement, mais conduit à des performances plus élevées sur les données non vues, telles que celles d'un jeu de validation et des échantillons en production. Cette stratégie est particulièrement efficace si nous pouvons utiliser l'augmentation pour rendre notre jeu d'entraînement plus similaire aux échantillons du monde réel.

J'ai une fois aidé un ingénieur à utiliser l'imagerie satellite pour détecter les routes inondées après un ouragan. Le projet était difficile, car il n'avait accès qu'à des données étiquetées pour les villes non inondées. Pour améliorer les performances de son modèle sur l'imagerie des ouragans, qui est nettement plus sombre et de moindre qualité, ils ont construit des pipelines d'augmentation qui ont rendu les images d'entraînement plus sombres et plus floues. Cela a réduit les performances de l'entraînement, puisque les routes étaient désormais plus difficiles à détecter. Mais, d'un autre côté, cela a augmenté la performance du modèle sur le jeu de validation car le processus d'augmentation a exposé le modèle à des images plus similaires à celles qu'il rencon-

trerait dans le jeu de validation. L'augmentation des données a contribué à rendre le jeu d'entraînement plus représentatif, et a donc rendu le modèle plus robuste.

Si, après avoir utilisé les méthodes décrites précédemment, un modèle donne toujours de mauvais résultats sur un jeu de validation, vous devez itérer sur le jeu de données lui-même.

Refondre l'ensemble des données

Dans certains cas, un partage difficile entre entraînement et validation peut conduire à un modèle avec sous-apprentissage et à des problèmes sur le jeu de validation. Si un modèle n'est exposé qu'à des échantillons faciles dans son jeu d'entraînement, mais à des échantillons difficiles dans son jeu de validation, il sera incapable d'apprendre à partir de points de données également difficiles. De même, certaines catégories d'échantillons peuvent être sous-représentées dans le jeu d'entraînement, ce qui empêche un modèle d'en tirer des enseignements. Si un modèle est entraîné pour minimiser une métrique agrégée, il risque de s'ajuster essentiellement à la majorité des classes, en ignorant les classes minoritaires.

Si les stratégies d'augmentation peuvent aider, la meilleure solution est souvent de revoir le partage des données d'entraînement pour rendre celles-ci plus représentatives. Pour ce faire, nous devons contrôler soigneusement les fuites de données et faire en sorte que les partages soient aussi équilibrés que possible en termes de difficulté. Si la nouvelle répartition des données attribue tous les échantillons faciles au jeu de validation, la performance du modèle sur celui-ci sera artificiellement élevée, mais elle ne se traduira pas par des résultats en production. Pour éviter que les partages de données ne soient de qualité inégale, nous pouvons utiliser la validation croisée à k plis[43], dans laquelle nous effectuons k divisions différentes successives, et nous mesurons la performance du modèle sur chaque division.

Une fois que nous aurons équilibré nos jeux d'entraînement et de validation pour nous assurer qu'ils sont d'une complexité similaire, les performances de notre modèle devraient s'améliorer. Si les performances ne sont toujours pas satisfaisantes, il se peut que nous nous attaquions tout simplement à un problème vraiment difficile.

Analyser la tâche à accomplir

Un modèle peut avoir du mal à se généraliser car la tâche est trop complexe. Les entrées que nous utilisons peuvent ne pas être prédictives de la cible, par exemple. Pour vous assurer que la tâche à laquelle vous vous attelez est d'une difficulté appropriée à l'état actuel du ML, je vous suggère de vous référer une fois de plus à la section « Se tenir sur les épaules des géants » du Chapitre 2, où j'ai décrit comment explorer et évaluer l'état actuel de la technique.

En outre, le fait de disposer d'un jeu de données ne signifie pas qu'une tâche est résoluble. Considérez par exemple la tâche impossible qui consisterait à prédire avec exactitude des sorties aléa-

43 Voir l'adresse `https://fr.wikipedia.org/wiki/Validation_croisée`.

toires à partir d'entrées aléatoires. Vous pourriez construire un modèle qui fonctionne bien sur un jeu d'entraînement en le mémorisant, mais ce modèle ne serait pas capable de prédire avec exactitude d'autres sorties aléatoires à partir d'entrées aléatoires.

Si vos modèles ne se généralisent pas, votre tâche risque d'être trop difficile. Il se peut que vos échantillons d'entraînement ne contiennent pas assez d'informations pour vous permettre d'apprendre des *caractéristiques significatives*, et qui seront instructives pour les futurs points de données. Si c'est le cas, alors votre problème n'est pas bien adapté au ML, et je vous invite à revoir le Chapitre 1 pour trouver un meilleur cadrage.

Conclusion

Dans ce chapitre, nous avons abordé les trois étapes successives que vous devez suivre pour faire fonctionner un modèle. Tout d'abord, déboguez l'enchaînement de votre pipeline en inspectant les données et en effectuant des tests. Ensuite, faites en sorte qu'un modèle fonctionne bien lors d'un test d'entraînement pour valider le fait qu'il a la capacité à apprendre. Enfin, vérifiez qu'il est capable de généraliser et de produire des résultats utiles sur des données non encore vues.

Ce processus vous aidera à déboguer les modèles, à les construire plus rapidement et à les rendre plus robustes. Une fois que vous avez construit, entraîné et débogué votre premier modèle, l'étape suivante consiste à évaluer ses performances, et soit à l'itérer soit à le déployer.

Dans le Chapitre 7, nous verrons comment utiliser un classifieur entraîné pour fournir des recommandations pratiques aux utilisateurs. Nous comparerons ensuite les modèles candidats pour l'Éditeur ML et nous déciderons lequel doit être retenu pour obtenir les meilleures recommandations possibles.

CHAPITRE 7

Utilisation des classifieurs pour des recommandations de rédaction

La meilleure façon de progresser en matière de ML consiste à suivre de manière répétée la boucle itérative décrite dans la Figure 7.1, que nous avons déjà vue dans l'introduction de la troisième partie. Commencez par établir une hypothèse de modélisation, itérez sur un pipeline de modélisation, et effectuez une analyse d'erreur détaillée pour étayer votre prochaine hypothèse.

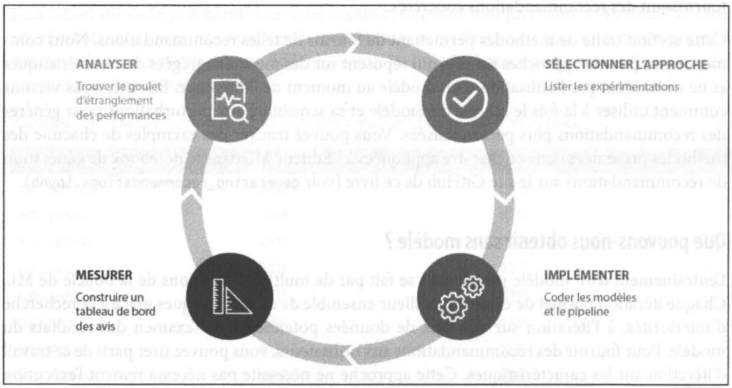


Figure 7.1 : La boucle du ML.

Les chapitres précédents ont décrit les multiples étapes de cette boucle. Au Chapitre 5, nous avons abordé la manière d'entraîner et de scorer un modèle. Dans le Chapitre 6, nous avons partagé des conseils sur la façon de construire des modèles plus rapidement et de dépanner les erreurs liées au ML. Ce chapitre clôt une itération de la boucle en présentant d'abord des méthodes d'utilisation de classifieurs entraînés afin de fournir des suggestions aux utilisateurs, puis en sélectionnant un modèle à utiliser pour l'Éditeur ML, et enfin en combinant les deux pour construire un Éditeur ML fonctionnel.

Dans la section « Planifier l'Éditeur ML » du Chapitre 2, nous avons présenté notre plan pour l'Éditeur ML, qui consiste à entraîner un modèle qui classifie des questions dans des catégories avec des scores élevés et faibles, et à utiliser ce modèle ainsi entraîné pour guider les utilisateurs à rédiger de meilleures questions. Voyons comment nous pouvons utiliser un tel modèle pour fournir des conseils d'écriture aux utilisateurs.

Extraire les recommandations des modèles

L'objectif de l'Éditeur ML est de fournir des recommandations de rédaction. La classification d'une question comme étant bonne ou mauvaise est un premier pas dans cette direction, puisqu'elle permet d'afficher pour un utilisateur la qualité actuelle d'une question. Nous aimerions aller plus loin et aider les utilisateurs à améliorer la formulation de leurs questions en leur fournissant des recommandations concrètes.

Cette section traite de méthodes permettant de fournir de telles recommandations. Nous commencerons par des approches simples qui reposent sur des mesures agrégées de caractéristiques et ne nécessitent pas l'utilisation d'un modèle au moment de l'inférence. Ensuite, nous verrons comment utiliser à la fois le score d'un modèle et sa sensibilité aux perturbations pour générer des recommandations plus personnalisées. Vous pouvez trouver des exemples de chacune des méthodes présentées dans ce chapitre appliquées à l'Éditeur ML dans le notebook de génération de recommandations sur le site GitHub de ce livre (voir `generating_recommendations.ipynb`).

Que pouvons-nous obtenir sans modèle ?

L'entraînement d'un modèle performant se fait par de multiples itérations de la boucle de ML. Chaque itération permet de créer un meilleur ensemble de caractéristiques grâce à la recherche d'antériorités, à l'itération sur des jeux de données potentiels et à l'examen des résultats du modèle. Pour fournir des recommandations aux utilisateurs, vous pouvez tirer parti de ce travail d'itération sur les caractéristiques. Cette approche ne nécessite pas nécessairement l'exécution d'un modèle pour chaque question soumise par un utilisateur, et se concentre plutôt sur la formulation de recommandations générales.

Vous pouvez le faire soit en utilisant directement les caractéristiques, soit en incorporant un modèle entraîné pour vous aider à sélectionner les caractéristiques pertinentes.

Utilisation de statistiques sur les caractéristiques

Une fois les caractéristiques prédictives identifiées, elles peuvent être communiquées directement à un utilisateur sans utiliser de modèle. Si la valeur moyenne d'une caractéristique est sensiblement différente pour chaque classe, vous pouvez partager cette information directement pour aider les utilisateurs à orienter leurs exemples dans la direction de la classe cible.

L'une des caractéristiques que nous avons identifiées très tôt pour notre Éditeur ML était la présence de points d'interrogation. L'examen des données a montré que les questions ayant obtenu un score élevé ont tendance à avoir moins de points d'interrogation. Pour utiliser cette information afin de générer des recommandations, nous pouvons écrire une règle qui avertit l'utilisateur si la proportion de points d'interrogation dans sa question est beaucoup plus importante que dans les questions ayant obtenu un score élevé.

La visualisation des valeurs moyennes des caractéristiques de chaque étiquette peut se faire en quelques lignes de code à l'aide de pandas.

```
class_feature_values = feats_labels.groupby("label").mean()
class_feature_values = class_feature_values.round(3)
class_feature_values.transpose()
```

L'exécution du code précédent donne les résultats qui sont indiqués dans le Tableau7.1. Dans ces résultats, nous pouvons voir que de nombreuses caractéristiques que nous avons générées ont des valeurs significativement différentes pour les questions ayant un score élevé ou un faible score, étiquetées ici Vrai et Faux.

Tableau 7.1 : Différences dans les valeurs des caractéristiques entre les classes.

Étiquette	Faux	Vrai
num_questions	0,432	0,409
num_periods	0,814	0,754
num_commas	0,673	0,728
num_exclam	0,019	0,015
num_quotes	0,216	0,199
num_colon	0,094	0,081
num_stops	10,537	10,610
num_semicolon	0,013	0,014
num_words	21,638	21,480
num_chars	822,104	967,032

L'utilisation de statistiques sur les caractéristiques est un moyen simple de fournir des recommandations solides. Elle est, à bien des égards, similaire à l'approche heuristique que nous avons d'abord élaborée dans le Chapitre 1, à la section « L'approche la plus simple : être l'algorithme ».

Lorsque l'on compare les valeurs des caractéristiques entre les classes, il peut être difficile d'identifier les caractéristiques qui contribuent le plus à ce qu'une question soit classée d'une certaine manière. Pour mieux l'estimer, nous pouvons nous servir de l'importance des caractéristiques.

Extraire l'importance globale des caractéristiques

Nous avons montré des exemples de génération de l'importance des caractéristiques dans le contexte de l'évaluation de modèles dans la section « Évaluer l'importance des caractéristiques » du Chapitre 5. L'importance des caractéristiques peut également être utilisée pour hiérarchiser les recommandations basées sur les caractéristiques. Lors de l'affichage des recommandations aux utilisateurs, les caractéristiques qui sont les plus prédictives pour un classifieur entraîné devraient être classées par ordre de priorité.

Ensuite, j'ai affiché les résultats d'une analyse de l'importance des caractéristiques pour un modèle de classification des questions qui utilise un total de 30 caractéristiques. Chacune des caractéristiques du haut a une importance beaucoup plus grande que celles du bas. En guidant les utilisateurs à agir en premier lieu en fonction de ces caractéristiques supérieures, on les aidera à améliorer leurs questions plus rapidement selon le modèle.

```
Importances (5 plus hautes) :

num_chars: 0.053
num_questions: 0.051
num_periods: 0.051
ADV: 0.049
ADJ: 0.049

Importances (5 plus basses):

X: 0.011
num_semicolon: 0.0076
num_exclam: 0.0072
CONJ: 0
SCONJ: 0
```

En combinant les statistiques sur les caractéristiques et l'importance des caractéristiques, les recommandations peuvent être plus faciles à exploiter et plus ciblées. La première approche fournit des valeurs cibles pour chaque caractéristique, tandis que la seconde donne la priorité à un sous-ensemble plus restreint des caractéristiques les plus importantes à afficher. Ces approches fournissent également des recommandations de manière rapide, car elles ne nécessitent pas d'exécuter un modèle au moment de l'inférence, mais seulement de comparer une entrée aux statistiques des caractéristiques les plus importantes.

Comme nous l'avons vu dans la section « Évaluer l'importance des caractéristiques » du Chapitre 5, l'extraction de l'importance des caractéristiques peut être plus difficile pour les modèles complexes. Si vous utilisez un modèle qui n'expose pas les importances des caractéristiques, vous pouvez utiliser un explicateur de boîte noire sur un grand jeu d'échantillons pour tenter d'en déduire les valeurs.

L'importance des caractéristiques et les statistiques sur les caractéristiques présentent un autre inconvénient, à savoir qu'elles ne fournissent pas toujours des recommandations précises. Étant donné que les recommandations sont basées sur des statistiques agrégées sur l'ensemble du jeu de données, elles ne seront pas applicables à chaque échantillon individuel. Les statistiques sur les caractéristiques ne fournissent que des recommandations générales, telles que « les questions qui contiennent plus d'adverbes ont tendance à recevoir des notes plus élevées ». Cependant, il existe des exemples de questions ayant une proportion d'adverbes inférieure à la moyenne et qui reçoivent tout de même un score élevé. De telles recommandations ne sont donc pas utiles pour ces questions.

Dans les deux sections suivantes, nous aborderons les méthodes permettant de fournir des recommandations plus granulaires qui fonctionnent au niveau des échantillons individuels.

Utiliser un score de modèle

Le Chapitre 5 a décrit comment les classifieurs produisent un score pour chaque échantillon. L'échantillon se voit ensuite attribuer à une classe selon que ce score est supérieur ou non à un certain seuil. Si le score d'un modèle est bien étalonné (voir la section « Courbe d'étalonnage » du Chapitre 5 pour plus d'informations à ce sujet), il peut alors être utilisé comme une estimation de la probabilité qu'un échantillon d'entrée appartienne à la classe donnée.

Pour afficher un score au lieu d'une classe pour un modèle scikit-learn, utilisez la fonction `predict_proba` et sélectionnez la classe pour laquelle vous souhaitez afficher un score.

```
# probabilities est un tableau contenant une probabilité par classe
probabilities = clf.predict_proba(features)

# Positive_probs ne contient que le score de la classe positive
positive_probs = clf[:,1]
```

S'il est bien étalonné, la présentation d'un score aux utilisateurs leur permet de suivre les améliorations apportées à leur question au fur et à mesure qu'ils suivent les recommandations de modification, ce qui leur permet d'obtenir un score plus élevé. Des mécanismes de retour d'information rapide tels qu'un score aident les utilisateurs à avoir un sentiment de confiance accru dans les recommandations fournies par un modèle.

En plus d'un score calibré, un modèle entraîné peut également être utilisé pour fournir des recommandations afin d'améliorer un échantillon spécifique.

Extraire l'importance locale des caractéristiques

Des recommandations peuvent être générées pour un échantillon individuel en utilisant un explicateur de boîte noire en plus d'un modèle entraîné. Dans la section « Évaluer l'importance des caractéristiques » du Chapitre 5, nous avons vu comment les explicateurs de boîte noire estiment l'importance des valeurs des caractéristiques pour un échantillon spécifique en appliquant de manière répétée de légères perturbations aux caractéristiques d'entrée, et en observant les changements qui se produisent dans le score prédit par le modèle. Cela fait de ces explicateurs un excellent outil pour fournir des recommandations.

Démontons cela en utilisant le paquet LIME[44] pour générer des explications pour un échantillon. Dans l'exemple de code qui suit, nous instancions d'abord un explicateur tabulaire, puis nous choisissons un échantillon à expliquer dans nos données de test. Nous montrons les explications dans le notebook de génération de recommandations du dépôt GitHub de ce livre (voir `generating_recommendations.ipynb`), et nous les affichons sous forme de tableau.

```
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(
    train_df[features].values,
    feature_names=features,
    class_names=["low", "high"],
    discretize_continuous=True,
)

idx = 8
exp = explainer.explain_instance(
    test_df[features].iloc[idx, :],
    clf.predict_proba,
    num_features=10,
    labels=(1,),
)

print(exp_array)
exp.show_in_notebook(show_table=True, show_all=False)
exp_array = exp.as_list()
```

L'exécution du code précédent produit le graphique illustré sur la Figure 7.2 ainsi que le tableau des importances des caractéristiques indiqué dans le code suivant. Les probabilités prédites du modèle sont affichées sur le côté gauche de la figure. Au milieu de la figure, les valeurs des caractéristiques sont classées en fonction de leur contribution à la prédiction.

44 Voir l'adresse `https://github.com/marcotcr/lime`.

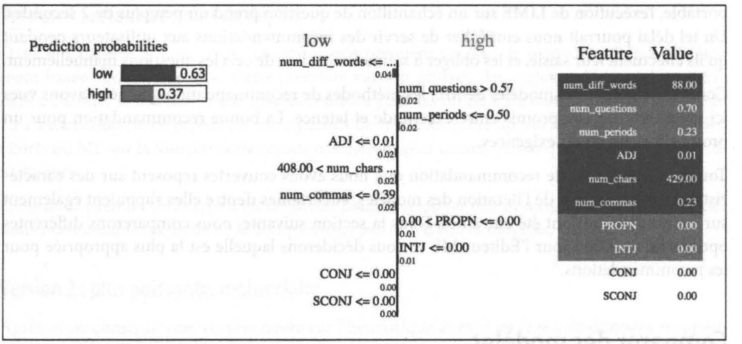


Figure 7.2 : Explications sous forme de recommandations.

Ces valeurs sont identiques à celles de la sortie plus lisible de la console affichée ci-dessous. Chaque ligne de cette sortie représente une valeur de caractéristique et son impact sur le score du modèle. Par exemple, le fait que la caractéristique `num_diff_words` ait une valeur inférieure à 88,00 a fait baisser le score du modèle d'environ 0,038. Selon ce modèle, l'augmentation de la longueur de la question en entrée au-delà de ce nombre en améliorerait la qualité.

```
[('num_diff_words <= 88.00', -0.038175093133182826),
('num_questions > 0.57', 0.022220445063244717),
('num_periods <= 0.50', 0.018064270196074716),
('ADJ <= 0.01', -0.01753028452563776),
('408.00 < num_chars <= 655.00', -0.01573650444507041),
('num_commas <= 0.39', -0.015551364531963608),
('0.00 < PROPN <= 0.00', 0.011826217792851488),
('INTJ <= 0.00', 0.011302327527387477),
('CONJ <= 0.00', 0.0),
('SCONJ <= 0.00', 0.0)]
```

Pour plus d'exemples d'utilisation, veuillez vous référer au notebook de génération de recommandations dans le dépôt GitHub du livre (voir `generating_recommendations.ipynb`).

Les explicateurs de boîte noire peuvent générer des recommandations exactes pour un modèle individuel, mais elles présentent un inconvénient. Ils produisent des estimations en perturbant les caractéristiques des entrées et en exécutant un modèle sur chaque entrée ainsi perturbée, de sorte que leur utilisation pour générer des recommandations est plus lente que les méthodes déjà discutées. Par exemple, le nombre de perturbations par défaut que LIME utilise pour évaluer l'importance des caractéristiques est de 500. Cette méthode est donc de deux ordres de grandeur plus lente que celles qui ne doivent exécuter un modèle qu'une seule fois, et même plus lente que celles qui n'ont pas besoin d'exécuter un modèle du tout. Sur mon ordinateur

portable, l'exécution de LIME sur un échantillon de question prend un peu plus de 2 secondes. Un tel délai pourrait nous empêcher de servir des recommandations aux utilisateurs pendant qu'ils effectuent leur saisie, et les obliger à soumettre au lieu de cela les questions manuellement.

Comme de nombreux modèles de ML, les méthodes de recommandation que nous avons vues ici présentent un compromis entre exactitude et latence. La bonne recommandation pour un produit dépend de ses exigences.

Toutes les méthodes de recommandation que nous avons couvertes reposent sur des caractéristiques générées lors de l'itération des modèles, et certaines d'entre elles s'appuient également sur les modèles qui ont été entraînés. Dans la section suivante, nous comparerons différentes options de modèles pour l'Éditeur ML et nous déciderons laquelle est la plus appropriée pour les recommandations.

Comparer des modèles

Dans la section « Mesurer le succès » du Chapitre 2, nous avons abordé des mesures importantes pour juger du succès d'un produit. La section « Juger des performances » du Chapitre 5 décrivait des méthodes d'évaluation des modèles. Ces méthodes peuvent également être utilisées pour comparer des itérations successives de modèles et de caractéristiques afin d'identifier les plus performants.

Dans cette section, nous allons choisir un sous-ensemble de métriques clés et les utiliser pour évaluer trois itérations successives de l'Éditeur ML en termes de performance du modèle et d'utilité des recommandations.

L'objectif de l'Éditeur ML est de fournir des recommandations en utilisant les techniques qui ont été mentionnées. Pour alimenter ces recommandations, un modèle doit répondre aux exigences suivantes. Il doit être bien calibré afin que ses probabilités prédites représentent une estimation significative de la qualité d'une question. Comme nous l'avons vu dans la section « Mesurer le succès » du Chapitre 2, il devrait avoir une haute précision afin que les recommandations qu'il formule soient exactes. Les caractéristiques qu'il utilise devraient être compréhensibles pour un utilisateur, car elles serviront de base aux recommandations. Enfin, il devrait également être suffisamment rapide pour nous permettre d'utiliser un explicateur de boîte noire pour fournir des recommandations.

Décrivons quelques approches de modélisation successives pour l'Éditeur ML et comparons leurs performances. Le code pour ces comparaisons de performances se trouve dans le notebook de comparaison des modèles que vous trouvez dans le dépôt GitHub de ce livre (voir `comparing_models.ipynb`).

Version 1 : retour au point de départ

Dans le Chapitre 3, nous avons construit une première version de l'éditeur qui était entièrement basée sur l'heuristique. Cette première version utilisait des règles codées en dur, destinées à encoder la lisibilité et affichait les résultats pour les utilisateurs dans un format structuré. La construction de ce pipeline nous a permis de modifier notre approche et de concentrer les efforts du ML sur la fourniture de recommandations plus claires, plutôt que sur un ensemble de métriques.

Comme ce premier prototype a été construit pour développer une intuition du problème auquel nous nous attaquions, nous ne le comparerons pas ici à d'autres modèles.

Version 2 : plus puissante, moins claire

Après avoir construit une version basée sur l'heuristique et exploré le jeu de données récupéré sur Stack Overflow, nous avons opté pour une première approche de modélisation. Le modèle simple que nous avons formé se trouve dans le notebook correspondant du dépôt GitHub de ce livre (voir `train_simple_model.ipynb`).

Ce modèle a utilisé une combinaison de caractéristiques générées par la vectorisation du texte à l'aide des méthodes décrites dans la section « Vectorisation » du Chapitre 4, et de caractéristiques créées manuellement qui ont été mises en évidence au cours de l'exploration des données. Lors de la première exploration du jeu de données, j'ai remarqué quelques schémas :

- Les questions plus longues ont obtenu des scores plus élevés.
- Les questions qui portaient spécifiquement sur l'utilisation de la langue courante (l'anglais en l'occurrence) ont reçu des notes plus faibles.
- Les questions qui contenaient au moins un point d'interrogation ont reçu des notes plus élevées.

J'ai créé des caractéristiques pour encoder ces hypothèses en comptant la longueur du texte, la présence de ponctuation et d'abréviations, et la fréquence des points d'interrogation.

En plus de ces caractéristiques, j'ai vectorisé les questions d'entrée en utilisant TF-IDF. L'utilisation d'un schéma de vectorisation simple me permet de lier les importances des caractéristiques d'un modèle à des mots individuels, ce qui peut permettre des recommandations au niveau des mots en utilisant les méthodes décrites précédemment.

Cette première approche a montré une performance globale acceptable, avec une précision de 0,62. Son étalonnage a cependant laissé beaucoup à désirer, comme vous pouvez le voir sur la Figure 7.3.

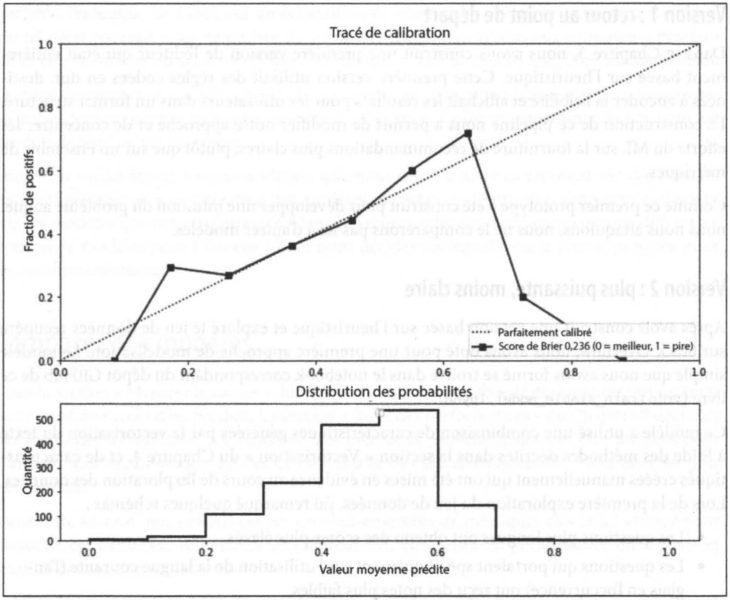


Figure 7.3 : Calibrage du modèle (version 2).

Après avoir inspecté l'importance des caractéristiques de ce modèle, j'ai réalisé que la seule caractéristique prédictive créée manuellement était la longueur des questions. Les autres caractéristiques générées n'avaient aucun pouvoir prédictif. L'exploration du jeu de données a révélé quelques autres caractéristiques qui paraissaient prédictives :

- Un usage modéré de la ponctuation semblait être prédictif de scores élevés.
- Les questions plus chargées émotionnellement semblaient recevoir un score plus faible.
- Les questions qui étaient descriptives et utilisaient plus d'adjectifs semblaient recevoir un score plus élevé.

Pour encoder ces nouvelles hypothèses, j'ai généré un nouvel ensemble de caractéristiques. J'ai créé des décomptes pour chaque élément de ponctuation possible. J'ai ensuite créé des comptages qui, pour chaque catégorie d'élément dans le discours, comme un verbe ou un adjectif, mesuraient combien de mots dans une question appartenaient à cette catégorie. Enfin, j'ai ajouté

une caractéristique pour coder le sentiment émotionnel d'une question. Pour plus de détails à ce sujet, reportez-vous au notebook du deuxième modèle dans le dépôt GitHub de ce livre (voir `second_model.ipynb`).

Cette version mise à jour du modèle a donné des résultats légèrement meilleurs dans l'ensemble, avec une précision de 0,63. Son calibrage n'a pas été amélioré par rapport au modèle précédent. L'affichage des caractéristiques importantes pour ce modèle a révélé que ce dernier repose exclusivement sur les caractéristiques créées manuellement, montrant que celles-ci ont un certain pouvoir prédictif.

Le fait qu'un modèle repose sur des caractéristiques aussi compréhensibles permet d'expliquer plus facilement les recommandations à un utilisateur que lorsqu'on utilise des caractéristiques vectorisées au niveau des mots. Par exemple, les caractéristiques les plus importantes au niveau des mots pour ce modèle sont *are* (sont) et *what* (quoi). Nous pouvons deviner pourquoi ces mots peuvent être corrélés à la qualité de la question, mais conseiller à un utilisateur de réduire ou d'augmenter l'occurrence de mots arbitraires dans sa question ne permet pas de formuler des recommandations claires.

Pour remédier à cette limitation de la représentation vectorisée et en reconnaissant que les caractéristiques créées manuellement étaient prédictives, j'ai tenté de construire un modèle plus simple qui n'utilise aucune caractéristique liée à la vectorisation.

Version 3 : des recommandations compréhensibles

Le troisième modèle ne contient que les caractéristiques décrites précédemment (nombre de signes de ponctuation et de parties du discours, expression de sentiment dans la question et longueur de celle-ci). Le modèle n'utilise donc que 30 caractéristiques, contre plus de 7 000 avec des représentations vectorisées. Pour plus de détails, reportez-vous au notebook du troisième modèle dans le dépôt GitHub de ce livre (voir `third_model.ipynb`). La suppression des caractéristiques vectorisées et la conservation des caractéristiques manuelles permettent à notre Éditeur ML de n'utiliser que celles qui sont explicables à un utilisateur. Cependant, cela peut conduire à une performance plus faible du modèle.

En termes de performance globale, ce modèle est moins performant que les précédents avec une précision de 0,597. Toutefois, il est nettement mieux étalonné (ou calibré) que les modèles précédents. Sur la Figure 7.4, vous pouvez voir que le modèle numéro 3 est bien calibré pour la plupart des probabilités, même celles qui sont au-dessus de à 0,7 et avec lesquelles les autres modèles ont du mal à composer. L'histogramme montre que cela est dû au fait que ce modèle prédit également ces probabilités plus souvent que les autres.

En raison de l'éventail plus large des scores qu'il produit et de l'amélioration de l'étalonnage de ceux-ci, ce modèle est le meilleur choix lorsqu'il s'agit d'afficher un score afin de guider les utilisateurs. Ce modèle est également le meilleur choix pour formuler des recommandations claires,

car il ne fait appel qu'à des caractéristiques explicables. Enfin, comme il repose sur moins de caractéristiques que les autres modèles, il est également le plus rapide à exécuter.

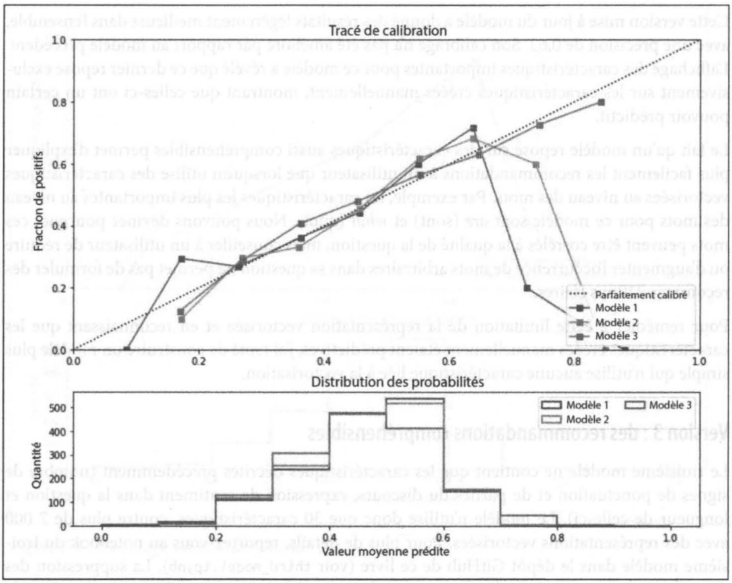


Figure 7.4 : Comparaison entre étalonnages.

Le modèle 3 est le meilleur choix pour l'Éditeur ML, et il est donc le modèle que nous devrions déployer pour une version initiale. Dans la section suivante, nous allons brièvement couvrir la manière d'utiliser ce modèle avec les techniques de recommandation pour fournir aux utilisateurs des recommandations pour l'édition de leurs questions.

Générer des recommandations d'édition

L'Éditeur ML peut bénéficier de l'une des quatre méthodes que nous avons décrites pour générer des recommandations. En fait, toutes ces méthodes sont présentées dans le notebook de recommandations dans le dépôt GitHub du livre (voir `generating_recommendations.ipynb`). Comme

le modèle que nous utilisons est rapide, nous illustrerons ici l'approche la plus élaborée, à l'aide d'explications en boîte noire.

Commençons par examiner l'ensemble de la fonction de recommandation qui prend en charge une question et fournit des conseils de rédaction sur la base d'un modèle entraîné. Voici à quoi ressemble cette fonction :

```
def get_recommendation_and_prediction_from_text(input_text, num_feats=10):
    global clf, explainer
    feats = get_features_from_input_text(input_text)
    pos_score = clf.predict_proba([feats])[0][1]

    exp = explainer.explain_instance(
        feats, clf.predict_proba, num_features=num_feats, labels=(1,)
    )
    parsed_exps = parse_explanations(exp.as_list())
    recs = get_recommendation_string_from_parsed_exps(parsed_exps)
    return recs, pos_score
```

En appelant cette fonction sur un exemple d'entrée et en imprimant proprement ses résultats, on obtient des recommandations telles que les suivantes. Nous pouvons ensuite proposer celles-ci aux utilisateurs pour leur permettre d'itérer sur leur question.

```
>> recos, score = get_recommendation_and_prediction_from_text(example_question)
>> print("%s score" % score)
0.4 score
>> print(*recos, sep="\n")
Increase question length
Increase vocabulary diversity
Increase frequency of question marks
No need to increase frequency of periods
Decrease question length
Decrease frequency of determiners
Increase frequency of commas
No need to decrease frequency of adverbs
Increase frequency of coordinating conjunctions
Increase frequency of subordinating conjunctions
```

Décomposons cette fonction. En commençant par sa signature, la fonction prend comme arguments une chaîne de saisie représentant une question, ainsi qu'un argument optionnel déterminant combien de caractéristiques parmi les plus importantes doivent faire l'objet de recommandations. Elle renvoie des recommandations, ainsi qu'un score représentant la qualité actuelle de la question.

En plongeant dans le corps de la question, la première ligne fait référence à deux variables définies globalement, le modèle entraîné et une instance d'un explicatif LIME comme celui que nous avons défini dans la section « Extraire l'importance locale des caractéristiques », plus haut dans ce chapitre. Les deux lignes suivantes génèrent des caractéristiques à partir du texte d'entrée

et transmettent ces caractéristiques au classifieur pour qu'il les prédise. Ensuite, exp est défini en utilisant LIME pour générer des explications.

Les deux derniers appels de fonction transforment ces explications en recommandations lisibles par un être humain. Voyons comment cela se passe en examinant les définitions de ces fonctions. Commençons par parse_explanations :

```
def parse_explanations(exp_list):
    global FEATURE_DISPLAY_NAMES
    parsed_exps = []
    for feat_bound, impact in exp_list:
        conditions = feat_bound.split(" ")

    # Nous ignorons les conditions doublement délimitées,
    # par exemple 1 <= a < 3, car elles sont plus difficiles
    # à formuler sous forme de recommandation
    if len(conditions) == 3:
        feat_name, order, threshold = conditions

        simple_order = simplify_order_sign(order)
        recommended_mod = get_recommended_modification(simple_order, impact)

        parsed_exps.append(
            {
                "feature": feat_name,
                "feature_display_name": FEATURE_DISPLAY_NAMES[feat_name],
                "order": simple_order,
                "threshold": threshold,
                "impact": impact,
                "recommendation": recommended_mod,
            }
        )
return parsed_exps
```

Cette fonction est longue, mais elle permet d'atteindre un objectif relativement simple. Elle prend l'ensemble des importances des caractéristiques renvoyées par LIME et produit un dictionnaire plus structuré qui peut être utilisé dans des recommandations. Voici un exemple de cette transformation :

```
# exps est dans le format des explications de LIME
>> exps = [('num_chars <= 408.00', -0.03908691525058592),
    ('DET > 0.03', -0.014685507408497802)]

>> parse_explanations(exps)

[{'feature': 'num_chars',
    'feature_display_name': 'question length',
    'order': '<',
    'threshold': '408.00',
    'impact': -0.03908691525058592,
    'recommendation': 'Increase'},
```

```
{'feature': 'DET',
    'feature_display_name': 'frequency of determiners',
    'order': '>',
    'threshold': '0.03',
    'impact': -0.014685507408497802,
    'recommendation': 'Decrease'}]
```

Notez que l'appel de fonction a converti la valeur seuil affichée par LIME en une recommandation d'augmentation ou de diminution de la valeur d'une caractéristique. Ceci est réalisé en utilisant la fonction suivante, `get_recommended_modification` :

```
def get_recommended_modification(simple_order, impact):
    bigger_than_threshold = simple_order == ">"
    has_positive_impact = impact > 0

    if bigger_than_threshold and has_positive_impact:
        return "No need to decrease"
    if not bigger_than_threshold and not has_positive_impact:
        return "Increase"
    if bigger_than_threshold and not has_positive_impact:
        return "Decrease"
    if not bigger_than_threshold and has_positive_impact:
        return "No need to increase"
```

Une fois que les explications sont analysées pour en faire des recommandations, il ne reste plus qu'à les présenter dans un format approprié. Cela est réalisé par le dernier appel de fonction dans `get_recommendation_and_prediction_from_text` :

```
def get_recommendation_string_from_parsed_exps(exp_list):
    recommendations = []
    for feature_exp in exp_list:
        recommendation = "%s %s" % (
            feature_exp["recommendation"],
            feature_exp["feature_display_name"],
        )
        recommendations.append(recommendation)
    return recommendations
```

Si vous souhaitez expérimenter cet éditeur et itérer sur lui, n'hésitez pas à vous référer au notebook de génération de recommandations dans le dépôt GitHub de ce livre (voir `generating_recommendations.ipynb`). À la fin du notebook, j'ai inclus un exemple d'utilisation du modèle de recommandations pour reformuler une question plusieurs fois et augmenter son score. Je reproduis cet exemple ici pour montrer comment de telles recommandations peuvent être utilisées pour guider les utilisateurs dans l'édition de leurs questions.

```
// Premier essai de question
>> get_recommendation_and_prediction_from_text(
    """
I want to learn how models are made
"""
```

```
)

0.39 score
Increase question length
Increase vocabulary diversity
Increase frequency of question marks
No need to increase frequency of periods
No need to decrease frequency of stop words

// On suit les trois premières recommendations
>> get_recommendation_and_prediction_from_text(
    """
I'd like to learn about building machine learning products.
Are there any good product focused resources?
Would you be able to recommend educational books?
"""
)

0.48 score
Increase question length
Increase vocabulary diversity
Increase frequency of adverbs
No need to decrease frequency of question marks
Increase frequency of commas

// On suit les recommendations encore une fois
>> get_recommendation_and_prediction_from_text(
    """
I'd like to learn more about ML, specifically how to build ML products.
When I attempt to build such products, I always face the same challenge:
how do you go beyond a model?
What are the best practices to use a model in a concrete application?
Are there any good product focused resources?
Would you be able to recommend educational books?
"""
)

0.53 score
```

Voilà, nous disposons ainsi d'un pipeline qui peut prendre en charge une question et fournir des recommandations concrètes aux utilisateurs. Ce pipeline est loin d'être parfait, mais nous avons maintenant un éditeur alimenté par ML qui fonctionne de bout en bout. Si vous souhaitez essayer de l'améliorer, je vous encourage à interagir avec cette version actuelle et à identifier les modes de défaillance à traiter. Il est intéressant de noter que si les modèles peuvent toujours être réutilisés, je dirais que l'aspect le plus prometteur à améliorer pour cet éditeur serait de générer de nouvelles caractéristiques encore plus claires pour les utilisateurs.

Conclusion

Dans ce chapitre, nous avons abordé différentes méthodes pour générer des suggestions à partir d'un modèle de classification entraîné. En gardant ces méthodes à l'esprit, nous avons comparé différentes approches de modélisation pour notre Éditeur ML, et nous avons choisi celle qui optimiserait l'objectif de notre produit, à savoir aider les utilisateurs à poser de meilleures questions. Nous avons ensuite construit un pipeline de bout en bout pour l'Éditeur ML et nous l'avons utilisé pour fournir des recommandations.

Le modèle que nous avons choisi peut encore être amélioré et bénéficier d'un plus grand nombre de cycles d'itération. Si vous souhaitez vous entraîner à utiliser les concepts que nous avons exposés dans cette partie du livre, je vous encourage à parcourir vous-même ces cycles. Dans l'ensemble, chaque chapitre de cette partie représente un aspect de la boucle d'itération du ML. Pour progresser dans de tels projets, suivez les étapes décrites dans cette section jusqu'à ce que vous estimiez qu'un modèle est prêt à être déployé.

Dans la quatrième partie, nous aborderons les risques liés au déploiement des modèles, la manière de les atténuer et les méthodes de surveillance et de réaction à la variabilité dans les performances des modèles.

PARTIE IV

Déployer et surveiller

Une fois que nous avons construit un modèle et que nous l'avons validé, nous voudrions que les utilisateurs y aient accès. Il existe de nombreuses méthodes différentes pour publier les modèles de ML. Le cas le plus simple consiste à construire une petite API. Mais, pour garantir le bon fonctionnement de vos modèles pour tous vos utilisateurs, vous aurez besoin de plus que cela.

La Figure IV.1 illustre certains des systèmes que nous allons couvrir dans les prochains chapitres et qui accompagnent généralement un modèle en production.

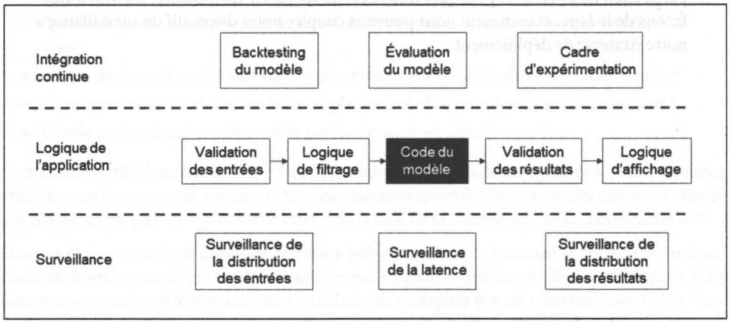


Figure IV.1 : Pipeline typique de modélisation en production.

En production, les pipelines de ML doivent être capables de détecter les données et de modéliser les défaillances et de les traiter efficacement, mais avec grâce. Dans l'idéal, vous devriez également viser à prédire de manière proactive toute défaillance et disposer d'une stratégie pour

déployer des modèles mis à jour. Si tout cela vous semble difficile, ne vous inquiétez pas ! C'est ce que nous allons aborder dans cette quatrième partie.

Chapitre 8

Avant le déploiement, nous devrions toujours effectuer un dernier cycle de validation. L'objectif est d'examiner en profondeur les abus potentiels ainsi que les utilisations négatives de nos modèles, et de faire de notre mieux pour anticiper et mettre en place des garanties autour de tels problèmes.

Chapitre 9

Nous aborderons ici les différentes méthodes et plateformes de déploiement des modèles, ainsi que la manière de choisir l'un ou l'autre.

Chapitre 10

Dans ce chapitre, nous apprendrons comment bâtir un environnement de production robuste pouvant supporter des modèles. Cela inclut la détection et le traitement des défaillances des modèles, l'optimisation des performances de ceux-ci, et la systématisation du réentraînement.

Chapitre 11

Dans ce dernier chapitre, nous aborderons l'étape cruciale du suivi des opérations. En particulier, nous verrons pourquoi nous devons surveiller les modèles, les meilleures façons de le faire, et comment nous pouvons coupler notre dispositif de surveillance à notre stratégie de déploiement.

CHAPITRE 8

Considérations lors du déploiement des modèles

Les chapitres précédents portaient sur l'entraînement des modèles et les performances en termes de généralisation. Ce sont des étapes nécessaires pour déployer un modèle, mais elles ne sont pas suffisantes pour garantir le succès d'un produit propulsé par le ML.

Le déploiement d'un modèle nécessite une plongée plus profonde dans les problèmes de défaillance qui pourraient avoir un impact sur les utilisateurs. Lorsque vous construisez des produits qui apprennent à partir des données, voici quelques questions auxquelles vous devez répondre :

- Comment les données que vous utilisez ont-elles été collectées ?
- Quelles hypothèses votre modèle fait-il en tirant des enseignements de ce jeu de données ?
- Ce jeu de données est-il suffisamment représentatif pour produire un modèle utile ?
- Comment les résultats de vos travaux pourraient-ils être utilisés à mauvais escient ?
- Quelle est l'utilisation prévue et la portée de votre modèle ?

Le domaine de l'éthique des données vise à répondre à certaines de ces questions, et les méthodes utilisées sont en constante évolution. Si vous souhaitez approfondir le sujet, les éditions O'Reilly ont rédigé un rapport complet sur le sujet, *Ethics and Data Science*, par Mike Loukides *et al*[45].

Dans ce chapitre, nous aborderons certaines préoccupations concernant la collecte et l'utilisation des données, ainsi que les défis à relever pour s'assurer que les modèles continuent à bien fonctionner pour tout le monde. Nous conclurons ce chapitre par un entretien avec Chris Harland, où nous aborderons des conseils pratiques pour traduire les prédictions des modèles en fonction des retours des utilisateurs.

Commençons par examiner les données, d'abord en couvrant les questions de propriété, puis en passant aux biais.

45 Voir l'adresse `https://bit.ly/3fJX0Uz`.

Les données et leurs problèmes

Dans cette section, nous commencerons par présenter des conseils à garder présents à l'esprit lorsque vous stockez, utilisez et générez des données. Nous traiterons d'abord de la propriété des données et des responsabilités qui accompagnent leur stockage. Ensuite, nous aborderons les sources courantes de biais dans les jeux de données et les méthodes permettant de prendre en compte ce biais lors de la construction de modèles. Enfin, nous couvrirons des exemples des conséquences négatives de ces biais et des raisons pour lesquelles il est important de les atténuer.

Propriété des données

La propriété des données fait référence aux exigences liées à la collecte et à l'utilisation des données. Voici quelques aspects importants à prendre en compte en ce qui concerne la propriété des données :

- Collecte de données : êtes-vous légalement autorisé à collecter et à utiliser le jeu de données sur lequel vous souhaitez entraîner votre modèle ?
- Utilisation des données et autorisation : avez-vous clairement expliqué à vos utilisateurs pourquoi vous aviez besoin de leurs données et comment vous vouliez les utiliser, et ont-ils accepté ?
- Stockage des données : comment stockez-vous vos données, qui y a accès, et quand les supprimerez-vous ?

La collecte de données auprès des utilisateurs peut aider à personnaliser et à adapter les expériences quant aux produits. Elle implique également des responsabilités à la fois morales et juridiques. S'il y a toujours eu une obligation morale de conserver les données fournies par les utilisateurs, les nouvelles réglementations en font de plus en plus une obligation légale. En Europe, par exemple, le règlement RGPD fixe désormais des directives strictes concernant la collecte et le traitement des données.

Pour les organisations qui stockent de grandes quantités de données, des violations de ces données représentent un risque de responsabilité important. Ces violations érodent la confiance des utilisateurs dans l'organisation et entraînent souvent des poursuites judiciaires. La limitation de la quantité de données collectées réduit donc l'exposition juridique.

Pour notre Éditeur ML, nous commencerons par utiliser des jeux de données accessibles publiquement, donc qui ont été recueillis avec l'accord des utilisateurs et qui sont stockés en ligne. Si nous voulions enregistrer des données supplémentaires, telles que des enregistrements de la manière dont notre service est utilisé afin de l'améliorer, nous devrions définir clairement une politique de collecte des données et la partager avec les utilisateurs.

Outre la collecte et le stockage des données, il est important de se demander si l'utilisation des données ainsi collectées peut conduire à de mauvaises performances. Un certain jeu de données peut être approprié dans certains cas, mais pas dans d'autres. Examinons pourquoi.

Biais dans les données

Les jeux de données sont le résultat de décisions spécifiques quant à la collecte de données. Ces décisions conduisent à des jeux qui présentent une vision biaisée du monde. Du fait que les modèles de ML tirent des enseignements des jeux de données, ils reproduisent donc ces préjugés.

Par exemple, supposons qu'un modèle est entraîné sur des données historiques pour prédire les compétences requises afin qu'une personne devienne P.-D.G. d'une grande entreprise sur la base d'informations incluant son sexe. Historiquement, selon la fiche d'information de 2018 « *The Data on Women Leaders* » compilée par le Pew Research Center[46], la plupart des P.-D.G. des 500 plus grandes entreprises du célèbre classement Fortune sont des hommes. L'utilisation de ces données pour entraîner un modèle permettra à celui-ci d'apprendre qu'être un homme est un précieux prédicteur de leadership. Le fait d'être un homme *et* d'être un P.-D.G. est corrélé dans le jeu de données choisi pour des raisons sociétales, ce qui a conduit à réduire les possibilités pour les femmes d'être même envisagées pour occuper de tels rôles. En entraînant aveuglément un modèle sur ces données et en l'utilisant pour faire des prédictions, nous ne ferions que renforcer les préjugés et les biais du passé.

Il peut être tentant de considérer les données comme une vérité de terrain. En réalité, la plupart des jeux de données sont une collection de mesures approximatives qui ignorent un contexte plus large. Nous devrions commencer par supposer que tout jeu de données est biaisé, et estimer comment ce biais affectera notre modèle. Nous pouvons ensuite prendre des mesures pour améliorer un jeu de données en le rendant plus représentatif, et ajuster les modèles afin de limiter leur capacité à propager le biais existant.

Voici quelques exemples de sources courantes d'erreurs et de biais dans les jeux de données :

- *Erreurs de mesure ou données corrompues* : chaque point de données comporte une incertitude due à la méthode utilisée pour le produire. La plupart des modèles ignorent cette incertitude et peuvent donc propager des erreurs de mesure systématiques.
- *Représentation* : la plupart des jeux de données présentent une vision non représentative d'une population. De nombreux jeux de données, aux premiers temps de la reconnaissance faciale, contenaient surtout des images d'hommes blancs. Cela a conduit à des modèles performants pour cette population mais défaillants pour d'autres.
- *Accès* : certains jeux de données peuvent être plus difficiles à trouver que d'autres. Par exemple, des textes en anglais sont plus faciles à rassembler en ligne que pour d'autres langues. Cette facilité d'accès fait que la plupart des modèles linguistiques de pointe sont

46 Voir l'adresse `https://pewrsr.ch/2xXkjcn`.

entraînés exclusivement à partir de données en anglais. Par conséquent, les anglophones auront accès à des services plus performants en matière de ML que les non-anglophones. Cette disparité se renforce souvent d'elle-même, car le volume supplémentaire d'utilisateurs pour les produits en anglais contribue à rendre ces modèles encore meilleurs que ceux des autres langues.

Des jeux de tests sont utilisés pour évaluer la performance des modèles. C'est pourquoi vous devez veiller tout particulièrement à ce que votre jeu de tests soit aussi exact et représentatif que possible.

Jeux de test

La question de la représentation apparaît dans chaque problème de ML. Dans la section « Partager votre jeu de données » du Chapitre 5, nous avons traité de l'intérêt de séparer les données en différents jeux pour évaluer la performance d'un modèle. Pour ce faire, vous devriez essayer de construire un jeu de test qui soit inclusif, représentatif et réaliste. En effet, un jeu de test sert d'indicateur de la performance en production.

Pour ce faire, lors de la conception de votre jeu de test, pensez à chaque utilisateur qui pourrait interagir avec votre modèle. Pour augmenter les chances que vos utilisateurs aient une expérience tout aussi positive les uns que les autres, essayez d'inclure dans votre jeu de test des échantillons représentatifs de chaque type d'utilisateur.

Concevez votre jeu de test pour encoder les objectifs du produit. Lorsque vous construisez un modèle de diagnostic, vous devez par exemple vous assurer qu'il fonctionne correctement pour tous les sexes. Pour évaluer si c'est le cas, vous devez faire en sorte qu'ils soient tous représentés dans votre jeu de test. La collecte d'un ensemble diversifié de points de vue peut vous aider dans cette entreprise. Si vous le pouvez, avant de déployer un modèle, donnez à un panel diversifié d'utilisateurs l'occasion d'examiner, d'interagir avec lui et de partager leurs commentaires.

Je voudrais faire une dernière remarque en ce qui concerne les biais. Les modèles sont souvent entraînés à partir de données historiques, qui représentent l'état du monde dans le passé. De ce fait, les biais affectent le plus souvent des populations déjà privées de droits. Travailler à l'élimination de ces préjugés est donc une entreprise qui peut contribuer à rendre les systèmes plus équitables pour les personnes qui en ont le plus besoin.

Biais systémique

Les biais systémiques font référence aux politiques institutionnelles et structurelles qui ont conduit à une discrimination injuste de certaines populations. En raison de cette discrimination, ces populations sont souvent sur- ou sous-représentées dans les jeux de données historiques. Par exemple, si des facteurs sociétaux ont contribué à ce que certaines populations soient historiquement surreprésentées dans les bases de données portant sur les arrestations crimi-

nelles, un modèle de ML entraîné à partir de ces données encodera ce biais et le transposera dans des prédictions modernes.

Cela peut avoir des conséquences désastreuses et conduire à la marginalisation de sous-ensembles de la population. Pour un exemple concret, vous pouvez consulter le rapport de J. Angwin *et al.* intitulé *Machine Bias* sur les biais dans le ML pour les prédictions sur la criminalité[47].

Il est difficile de supprimer ou de limiter les biais dans un jeu de données. Lorsque l'on tente d'empêcher qu'un modèle ne soit biaisé par rapport à certaines caractéristiques, telles que l'appartenance ethnique ou le sexe, certains ont essayé de supprimer l'attribut en question de la liste des caractéristiques qu'un modèle utilise pour faire des prédictions.

En pratique, le simple fait de supprimer une caractéristique n'empêche pas un modèle d'être biaisé à son encontre, car la plupart des jeux de données contiennent de nombreuses autres caractéristiques qui sont fortement corrélées avec elle. Par exemple, le code postal et les revenus sont fortement corrélés avec l'appartenance ethnique aux États-Unis. Si vous ne supprimez qu'une seule caractéristique, un modèle peut être tout aussi biaisé, bien que de manière plus difficile à détecter.

Vous devriez plutôt être explicite quant aux contraintes en matière d'équité (*fairness constraints*) que vous essayez de faire respecter. Par exemple, vous pourriez suivre l'approche décrite dans le document de M. B. Zafar *et al.* intitulé *Fairness Constraints : Mechanisms for Fair Classification*[48], où l'équité d'un modèle est mesurée à l'aide de la règle dite du p%. Cette règle du p% est définie comme étant « le rapport entre le pourcentage de sujets ayant une certaine valeur d'attribut sensible recevant un résultat positif, et le pourcentage de sujets n'ayant pas cette valeur mais recevant le même résultat, ne devrait pas être inférieur à p:100 ». L'utilisation d'une telle règle nous permet de quantifier le biais et donc de mieux l'aborder, mais elle exige de garder la trace de la caractéristique pour laquelle nous aimerions qu'un modèle ne soit pas biaisé.

Outre l'évaluation des risques, des biais et des erreurs dans un jeu de données, le ML nécessite l'évaluation des modèles.

Préoccupations en matière de modélisation

Comment minimiser le risque qu'un modèle introduise un biais indésirable ?

Les modèles peuvent avoir des impacts négatifs sur les utilisateurs de multiples façons. Tout d'abord, nous nous attaquerons aux boucles de rétroaction fugitives, puis nous explorerons les risques qu'un modèle échoue de manière discrète sur une petite partie de la population. Nous discuterons ensuite de l'importance de contextualiser les prédictions du ML de manière appro-

47 Voir l'article sur le site de ProPublica, à l'adresse `https://bit.ly/2Z1Dnl8`.

48 Voir l'adresse `https://arxiv.org/abs/1507.05259`.

priée pour les utilisateurs, et nous terminerons cette section en abordant le risque de voir des acteurs malveillants se servir de manière abusive des modèles.

Boucles de rétroaction

Dans la plupart des systèmes alimentés par le ML, le fait qu'un utilisateur suive la recommandation d'un modèle augmentera la probabilité pour que les futurs modèles fassent la même recommandation. Si ce phénomène n'est pas contrôlé, il peut conduire les modèles à entrer dans une boucle de rétroaction autorenforcée.

Par exemple, si nous entraînons un modèle à recommander des vidéos aux utilisateurs, et que notre première version de ce modèle soit légèrement plus susceptible de recommander des vidéos de chats que de chiens, alors les utilisateurs regarderont en moyenne plus de vidéos de chats que de chiens. Si nous entraînons une deuxième version du modèle en utilisant un jeu de données basé sur des historiques de recommandations et de clics, nous intégrerons le biais du premier modèle dans notre jeu de données. Notre deuxième modèle favorisera donc beaucoup plus les chats.

Comme les modèles de recommandation de contenu sont souvent actualisés plusieurs fois par jour, il ne faudrait pas longtemps pour que notre dernière version en date du modèle recommande exclusivement des vidéos de chats. Vous pouvez en voir un exemple sur la Figure 8.1. En raison de la popularité initiale d'une vidéo de chat, le modèle apprend progressivement à recommander de plus en plus de vidéos de cet animal, jusqu'à ce qu'il atteigne l'état à droite, ne recommandant alors que des vidéos de chats.

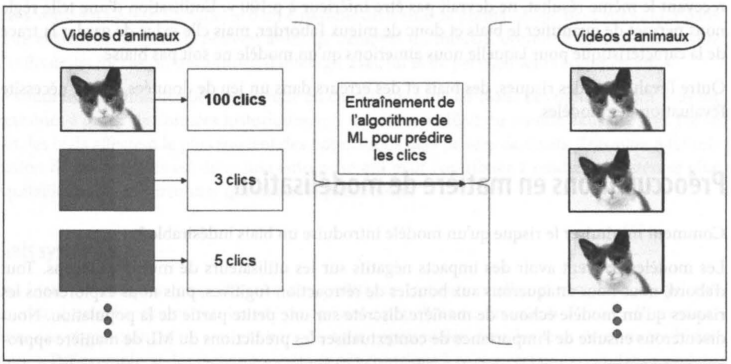


Figure 8.1 : Exemple de boucle de rétroaction.

Remplir l'Internet de vidéos de chats ne semble peut-être pas représenter une tragédie, mais vous pouvez imaginer comment de tels mécanismes peuvent rapidement renforcer des préjugés négatifs et recommander des contenus inappropriés ou dangereux à des utilisateurs peu méfiants. En fait, les modèles qui tentent de maximiser la probabilité qu'un utilisateur clique apprennent à recommander des contenus appelés *pièges à clics* (*clickbait*), c'est-à-dire des contenus sur lesquels il peut être très tentant de cliquer, mais qui n'apportent aucune valeur à l'utilisateur.

Les boucles de rétroaction ont également tendance à introduire un biais pour favoriser une minorité d'utilisateurs très actifs. Si une plateforme vidéo utilise le nombre de clics sur chaque vidéo pour entraîner son algorithme de recommandation, le risque sera de surreprésenter ses utilisateurs les plus actifs, donc ceux qui représentent la grande majorité des clics. Tous les autres utilisateurs de la plateforme seraient alors exposés aux mêmes vidéos, quelles que soient leurs préférences individuelles.

Pour limiter les effets négatifs de tels modèles, choisissez un type de mesure qui soit moins enclin à créer une telle boucle. Les clics indiquent seulement si un utilisateur ouvre une vidéo, et non s'il l'apprécie. Utiliser les clics comme objectif d'optimisation conduit à recommander un contenu plus accrocheur, sans se soucier de sa pertinence. Remplacer la métrique cible par le temps de visionnage, qui est davantage corrélé à la satisfaction de l'utilisateur, permettrait d'atténuer une telle boucle de rétroaction.

Même dans ce cas, les algorithmes de recommandation qui sont optimisés pour un engagement quelconque comportent toujours le risque de dégénérer en une boucle de rétroaction, puisque leur seul objectif est de maximiser une métrique pratiquement sans limites. Par exemple, même si un algorithme optimise le temps de visionnage pour encourager un contenu plus engageant, l'état du monde qui maximiserait cette métrique serait celui dans lequel chaque utilisateur passerait sa journée entière à regarder des vidéos ! L'utilisation de telles métriques d'engagement peut contribuer à accroître l'usage, mais cela soulève la question de savoir si c'est toujours un objectif qui vaut la peine d'être optimisé.

Outre le risque de créer des boucles de rétroaction, les modèles peuvent également présenter des performances inférieures à celles attendues en production malgré des scores convaincants sur les métriques de validation hors ligne.

Réfléchir à la problématique des performances

Dans la section « Évaluer votre modèle : aller au-delà de l'exactitude », du Chapitre 5, nous avons abordé une série de métriques d'évaluation qui tentent de juger des performances sur différents sous-ensembles d'un jeu de données. Ce type d'analyse est utile pour s'assurer qu'un modèle fonctionne également bien pour différents types d'utilisateurs.

C'est particulièrement important lorsqu'il s'agit d'entraîner de nouvelles versions de modèles existants et de prendre des décisions quant à leur déploiement. Si vous ne comparez que les

performances globales, vous pourriez ne pas remarquer une dégradation significative des performances sur un segment des données.

Le fait de ne pas remarquer une telle dégradation des performances a entraîné des défaillances catastrophiques de produits. En 2015, un système automatisé de marquage de clichés a classifié les photos des utilisateurs afro-américains dans la catégorie des gorilles[49]. C'est un échec épouvantable, et une conséquence de la non-validation d'un modèle sur un ensemble représentatif d'entrées.

Ce genre de problème peut se poser lors de la mise à jour d'un modèle existant. Supposons par exemple que vous mettiez à jour un modèle de reconnaissance faciale. Le modèle précédent avait une exactitude de 90 %, et le nouveau a une exactitude de 92 %. Avant de déployer ce nouveau modèle, vous devez évaluer ses performances sur quelques sous-ensembles d'utilisateurs différents. Vous constaterez peut-être que si les performances se sont légèrement améliorées dans l'ensemble, l'exactitude du nouveau modèle est très faible pour les photos de femmes de plus de 40 ans, et vous devriez donc vous abstenir de le déployer. Vous devriez plutôt modifier les données d'entraînement pour ajouter des échantillons plus représentatifs, et réentraîner un modèle capable d'être performant pour chaque catégorie.

L'omission de ces points de repère peut conduire à ce que les modèles échouent pour une partie importante de leur public cible. La plupart des modèles ne fonctionneront jamais pour la totalité de toutes les entrées possibles, mais il est important de valider le fait qu'ils fonctionnent pour toutes celles qui sont attendues.

Considérer le contexte

Les utilisateurs ne seront pas toujours conscients qu'une information donnée provient d'une prédiction d'un modèle de ML. Dans la mesure du possible, vous devriez partager le contexte d'une prédiction avec un utilisateur, afin qu'il puisse prendre une décision éclairée sur la manière de l'exploiter. Pour ce faire, vous pouvez commencer par lui décrire comment le modèle a été entraîné.

Il n'existe pas encore de format standard de « modèle de clause de non-responsabilité », mais des recherches actives dans ce domaine ont montré des avancées prometteuses, telles que les *model cards* (voir l'article de M. Mitchell *et al.*, « *Model Cards for Model Reporting* »[50]), un système de documentation pour un rapport transparent sur les modèles. Dans l'approche proposée, un modèle est accompagné de métadonnées sur la manière dont il a été entraîné, sur les données sur lesquelles il a été testé, sur l'utilisation à laquelle il est destiné, *etc.*

Dans notre étude de cas, l'Éditeur ML fournit un retour d'information basé sur un ensemble de questions spécifiques. Si nous devions le déployer en tant que produit, nous inclurions une

49 Voir par exemple l'article de la BBC de 2015 à l'adresse `https://bbc.in/35WHqjQ`.
50 Voir l'adresse `https://arxiv.org/abs/1810.03993`.

clause de non-responsabilité concernant les types d'entrées sur lesquelles le modèle est censé être performant. Cet avertissement pourrait être aussi simple que : « Ce produit essaie de recommander de meilleures façons de formuler une question. Il a été entraîné sur les questions posées sur le site de Stack Exchange et peut donc refléter les préférences particulières de cette communauté. »

Il est important de tenir informés les utilisateurs, et ce de manière claire. Examinons maintenant les problèmes potentiels qui peuvent provenir d'utilisateurs moins *sympathiques*.

Se défendre contre les agressions

Certains projets de ML doivent prendre en compte le risque de voir des modèles battus en brèche par des fraudeurs. Ils peuvent par exemple tenter de tromper un modèle chargé de détecter les transactions suspectes par carte de crédit. Des adversaires peuvent aussi vouloir sonder un modèle entraîné pour glaner sur les données sous-jacentes des informations auxquelles ils ne devraient pas être autorisés à accéder, telles que des informations sensibles sur les utilisateurs.

Vaincre un modèle

De nombreux modèles de ML sont déployés pour protéger les comptes et les transactions à l'égard des fraudeurs. En retour, ces fraudeurs tentent de déjouer ces modèles en les trompant et en leur faisant croire qu'ils sont des utilisateurs légitimes.

Si vous essayez d'empêcher les connexions frauduleuses à une plateforme en ligne, par exemple, vous pouvez envisager des ensembles de caractéristiques qui incluraient le pays d'origine de l'utilisateur (de nombreuses attaques à grande échelle utilisent plusieurs serveurs de la même région). Si vous entraînez un modèle sur de telles caractéristiques, vous risquez cependant d'introduire un biais contre les utilisateurs non frauduleux dans les pays où vivent les fraudeurs. De plus, en ne s'appuyant que sur une telle caractéristique, il sera facile pour les acteurs malveillants de tromper vos systèmes en falsifiant leur localisation.

Pour se défendre contre des adversaires, il est important de mettre régulièrement à jour les modèles. À mesure que les attaquants apprennent les modèles de défense existants et adaptent leur comportement pour les vaincre, actualisez vos modèles afin qu'ils puissent rapidement classifier ce nouveau comportement comme frauduleux. Cela nécessite des systèmes de surveillance afin que nous puissions détecter les changements dans la nature des activités. Nous aborderons cette question plus en détail au Chapitre 11. Dans de nombreux cas, pour se défendre contre les agresseurs, il faut générer de nouvelles caractéristiques pour mieux détecter leur comportement. N'hésitez pas à vous reporter à la section « Laisser les données informer les caractéristiques et les modèles » du Chapitre 4 pour un rappel sur la génération de caractéristiques.

Le type d'attaque le plus courant sur les modèles vise à les tromper en leur faisant faire de fausses prédictions, mais d'autres types d'attaques existent. Certaines attaques visent à utiliser un modèle entraîné pour connaître les données sur lesquelles il a précisément été entraîné.

Exploitation d'un modèle

Plus que de tromper un modèle, les attaquants pourraient l'utiliser pour apprendre des informations privées. Un modèle reflète les données sur lesquelles il a été entraîné, de sorte que l'on peut utiliser ses prédictions pour déduire des schémas dans le jeu de données original. Pour illustrer cette idée, prenons l'exemple d'un modèle de classification entraîné sur un jeu de données contenant deux échantillons. Chaque échantillon est d'une classe différente, et les deux échantillons ne diffèrent que par une seule valeur de caractéristique. Si vous donniez à un attaquant l'accès à un modèle entraîné sur ce jeu de données et lui permettiez d'observer les prédictions de ce modèle sur des entrées arbitraires, il pourrait éventuellement en déduire que cette caractéristique est la seule prédictive dans le jeu de données. De même, un attaquant pourrait déduire la distribution des caractéristiques dans les données d'entraînement. Ces distributions reçoivent souvent des informations sensibles ou privées.

Dans l'exemple de détection de connexion frauduleuse, imaginons que le code postal soit l'un des champs obligatoires lors de cette connexion. Un attaquant pourrait tenter de se connecter avec de nombreux comptes différents, en testant différents codes postaux pour voir quelles valeurs conduisent à une connexion réussie. Ce faisant, il pourrait estimer la répartition des codes postaux dans le jeu d'entraînement, et donc la répartition géographique des clients de ce site Web.

Le moyen le plus simple de contrecarrer l'efficacité de ces attaques est de limiter le nombre de requêtes qu'un utilisateur donné peut faire, réduisant ainsi sa capacité à explorer les valeurs des caractéristiques. Il ne s'agit pas d'une solution miracle, car des attaquants sophistiqués peuvent être en mesure de créer de multiples comptes pour contourner une telle limite.

Les adversaires décrits dans cette section ne sont pas les seuls utilisateurs malveillants dont vous devriez vous préoccuper. Si vous choisissez de partager votre travail avec la communauté au sens large, vous devriez également vous demander s'il ne pourrait pas être utilisé pour des applications dangereuses.

Préoccupations en matière d'abus et de double usage

Le terme « double usage » décrit des technologies qui sont développées dans un objectif précis mais qui peuvent être utilisées dans d'autres buts. En raison de la capacité du ML à donner de bons résultats sur des jeux de données de type similaire (voir la Figure 2.3), les modèles de ML présentent souvent un problème de double usage.

Si vous construisez un modèle qui permet aux gens de changer leur voix pour ressembler à celle de leurs amis, pourrait-il être utilisé à mauvais escient pour se faire passer pour d'autres personnes sans leur consentement ? Si vous choisissez de le construire, comment pourriez-vous inclure les conseils et les ressources nécessaires pour vous assurer que les utilisateurs comprennent l'utilisation correcte de votre modèle ?

De même, tout modèle capable de classifier avec précision les visages a des implications à double usage pour la surveillance. Bien qu'un tel modèle puisse à l'origine être construit pour mettre en place un système d'alerte intelligent, il pourrait ensuite être utilisé pour suivre automatiquement les individus à travers un réseau de caméras à l'échelle d'une ville. Les modèles sont construits à partir d'un certain jeu de données, mais ils peuvent présenter des risques lorsqu'ils sont réentraînés sur d'autres jeux de données similaires.

Il n'existe actuellement pas de meilleures pratiques qui soient claires sur la prise en compte du double usage. Si vous pensez que votre travail pourrait être exploité dans des buts non éthiques, je vous encourage à envisager d'en rendre plus difficile la reproduction à cette fin, ou à engager une discussion réfléchie avec la communauté. En 2019, le laboratoire de recherche OpenAI a décidé de ne pas publier son modèle linguistique le plus puissant, craignant qu'il ne facilite grandement la diffusion de la désinformation en ligne (voir à ce sujet le billet d'annonce sur le site d'OpenAI, « *Better Language Models and Their Implications* »[51]). Bien que cette décision soit relativement nouvelle, je ne serais pas surpris que de telles préoccupations soient soulevées plus souvent à l'avenir.

Pour conclure ce chapitre, je vous propose de partager une discussion avec Chris Harland, actuellement directeur de l'ingénierie chez Textio, qui a une grande expérience dans le déploiement de modèles pour les utilisateurs et dans la présentation des résultats avec un contexte suffisant pour les rendre utiles.

Chris Harland : Expériences de déploiement

Chris est titulaire d'un doctorat en physique et a travaillé sur diverses tâches de ML, notamment la vision par ordinateur pour extraire des informations structurées à partir des justificatifs de logiciels de gestion. Il a travaillé dans l'équipe de recherche de Microsoft, où il a réalisé tout le potentiel de l'ingénierie ML. Chris a ensuite rejoint Textio, une entreprise qui développe des produits d'écriture augmentée pour aider les utilisateurs à rédiger des descriptions de poste plus convaincantes. Chris et moi avons discuté de son expérience de déploiement de produits utilisant la technologie ML et de la façon dont il aborde la validation des résultats au-delà des métriques d'exactitude.

Q : Textio utilise le ML pour guider directement les utilisateurs. En quoi est-ce différent des autres tâches de ML ?

R : Lorsque vous vous concentrez uniquement sur des prédictions, comme par exemple quand acheter de l'or ou qui suivre sur Twitter, vous pouvez tolérer un certain degré de variance. Lorsque vous donnez des conseils pour l'écriture, ce n'est pas le cas, car vos recommandations comportent beaucoup de sous-texte, c'est-à-dire de contenu implicite.

51 Voir l'adresse `https://openai.com/blog/better-language-models/`.

Si vous me dites d'écrire 200 mots de plus, votre modèle devrait être cohérent et permettre à l'utilisateur de suivre ses conseils. Une fois que l'utilisateur a écrit 150 mots, le modèle ne peut plus changer d'avis et recommander de réduire le nombre de mots.

Les orientations doivent également être claires : « supprimer les mots d'arrêt de 50 % » est une instruction qui prête à confusion, mais « réduire la longueur de ces 3 phrases » peut aider les utilisateurs de manière plus concrète. Le défi consiste alors à maintenir les performances tout en utilisant des caractéristiques plus compréhensibles pour l'utilisateur.

Pour l'essentiel, les assistants d'écriture basés sur le ML guident l'utilisateur dans notre espace de caractéristiques, pour aller d'un point initial à un meilleur point en accord avec notre modèle. Parfois, cela peut impliquer de passer par des points moins bons, ce qui peut être une expérience frustrante pour l'utilisateur. Le produit doit être construit en tenant compte de ces contraintes.

Q : Quels sont les bons moyens pour mettre en œuvre cette assistance ?

R : À titre indicatif, la précision est beaucoup plus intéressante que le rappel. Si vous envisagez de donner des conseils à une personne, le rappel serait la capacité de donner de tels conseils dans tous les domaines potentiellement pertinents et dans certains domaines non pertinents (qui sont nombreux), tandis que la précision consisterait à donner des conseils dans quelques domaines prometteurs en ignorant d'autres domaines potentiels.

Lorsqu'on donne des conseils, le coût de l'erreur est très élevé, et c'est pourquoi la précision est la plus utile. Les utilisateurs tireront également des enseignements des recommandations que votre modèle a fournies précédemment et les appliqueront sans autre suggestion aux données futures, ce qui rend la précision de ces recommandations encore plus importante.

En outre, comme nous faisons apparaître différents facteurs, nous mesurons si les utilisateurs en tirent effectivement parti. Si ce n'est pas le cas, nous devrions comprendre pourquoi. Un exemple pratique est notre fonction « ratio actif/passif », qui était sous-utilisée. Nous avons réalisé que cela était dû au fait que la recommandation n'était pas suffisamment exploitable. Nous l'avons donc améliorée en mettant en évidence les mots eux-mêmes que nous recommandons de modifier.

Q : Comment trouvez-vous de nouvelles façons de guider vos utilisateurs ou de proposer de nouvelles caractéristiques ?

R : Les deux approches, descendante et ascendante, sont précieuses.

La recherche d'hypothèses en allant de haut en bas est axée sur la connaissance du domaine et consiste essentiellement en un appariement de caractéristiques à partir d'une expérience antérieure. Cela peut provenir des équipes chargées des produits ou de la vente, par exemple. Une hypothèse descendante peut ressembler à quelque chose comme « nous croyons qu'il y a quelque chose dans l'aspect mystérieux des e-mails de recrutement qui contribue à stimuler l'engagement ». Le défi du *top-down* est généralement de trouver un moyen pratique d'extraire cette caractéristique. Ce n'est qu'alors que l'on peut valider le fait que la caractéristique est prédictive.

La démarche ascendante vise à introspecter un pipeline de classification pour comprendre ce qu'il trouve prédictif. Si nous disposons d'une représentation générale du texte telle que des vecteurs de mots, des tokens et des parties d'annotations vocales que nous injectons ensuite dans des ensembles de modèles pour les classifier comme étant du bon ou du mauvais texte, quelles sont alors les caractéristiques les plus prédictives de notre classification ? Les experts du domaine seront souvent les mieux équipés pour identifier ces caractéristiques à partir des prédictions d'un modèle. Le défi consiste alors à trouver un moyen de rendre ces caractéristiques compréhensibles pour l'homme.

Q : Comment décidez-vous si un modèle est suffisamment bon ?

R : Vous ne devriez pas sous-estimer l'impact d'un petit jeu de données textuelles dans une langue donnée. Il s'avère que le fait de n'utiliser qu'un millier de documents dans votre domaine suffit pour de nombreux cas d'utilisation. Avoir la possibilité d'étiqueter ce petit jeu de données en vaut la peine. Vous pouvez alors commencer par tester votre modèle sur des données hors échantillon.

Vous devriez faciliter l'exécution d'expériences. Une écrasante majorité des idées que vous avez pour changer votre produit finissent par avoir un effet net nul, ce qui devrait vous permettre d'être un peu moins préoccupé par les nouvelles caractéristiques.

Enfin, construire un mauvais modèle, c'est bien et c'est par là qu'il faut commencer. La correction des mauvais modèles rendra votre produit beaucoup plus résistant aux problèmes et l'aidera à évoluer plus rapidement.

Q : Comment voyez-vous le fonctionnement d'un modèle une fois qu'il est en production ?

R : En cours de production, exposez clairement les prédictions de votre modèle aux utilisateurs et laissez-les les passer en revue. Enregistrez les valeurs des caractéristiques, les prédictions et les remplacements afin de pouvoir les observer et les analyser plus tard. Si votre modèle produit un score, trouver des moyens de comparer ce score à l'utilisation de vos recommandations peut être un signal supplémentaire. Si vous prédisez qu'un courrier électronique sera ouvert, par exemple, il peut être extrêmement utile d'avoir accès aux données de vérité de terrain de vos utilisateurs afin de pouvoir améliorer votre modèle.

La mesure ultime du succès est la réussite du client, qui est la plus tardive et qui est influencée par de nombreux autres facteurs.

Conclusion

Nous avons commencé par couvrir les préoccupations liées à l'utilisation et au stockage des données. Ensuite, nous avons examiné les causes des biais dans les jeux de données et proposé les conseils pour les identifier et les réduire. Nous avons également examiné les défis auxquels

les modèles sont confrontés dans le monde réel, et la manière de réduire les risques liés à leur exposition aux utilisateurs. Enfin, nous avons examiné comment architecturer les systèmes de manière à ce qu'ils soient conçus pour être résistants aux erreurs.

Il s'agit de questions complexes, et le domaine du ML a encore beaucoup à faire pour lutter contre toutes les formes potentielles d'abus. La première étape consiste pour tous les praticiens à être conscients de ces préoccupations et à en tenir compte dans leurs propres projets.

Nous sommes maintenant prêts à déployer des modèles. Pour commencer, nous allons explorer dans le Chapitre 9 les compromis entre différentes options de déploiement. Ensuite, nous aborderons au Chapitre 10 les méthodes permettant d'atténuer certains des risques associés au déploiement des modèles.

CHAPITRE 9

Choisir votre option de déploiement

Les chapitres précédents ont couvert le processus de passage d'une idée de produit à une implémentation ML, ainsi que les méthodes pour itérer sur cette application jusqu'à ce que vous soyez prêt à la déployer.

Ce chapitre couvre les diverses options de déploiement et les compromis entre chacune d'entre elles. Différentes approches de déploiement sont adaptées à différents ensembles d'exigences. Lorsque vous choisirez l'une d'entre elles, vous devrez tenir compte de plusieurs facteurs, tels que la latence, les besoins en matière de matériel et de réseau, ainsi que les questions de confidentialité, de coût et de complexité.

L'objectif du déploiement d'un modèle est de permettre aux utilisateurs d'interagir avec lui. Nous traiterons des approches courantes pour atteindre cet objectif, ainsi que des conseils pour choisir entre différentes approches lors du déploiement des modèles.

Nous débuterons par la manière la plus simple de commencer pour le déploiement des modèles en faisant tourner un serveur Web pour servir des prédictions.

Déploiement côté serveur

Le déploiement côté serveur consiste à mettre en place un serveur Web qui peut accepter les requêtes des clients, les faire passer par un pipeline d'inférence et renvoyer les résultats. Cette solution s'inscrit dans un paradigme de développement Web, car elle traite les modèles comme un autre point de terminaison dans une application. Les utilisateurs ont des requêtes qu'ils envoient à ce point, et ils attendent des résultats.

Il existe deux charges de travail courantes pour les modèles déployés côté serveur : *streaming* et *batch*. Les flux de travail en streaming acceptent les requêtes au fur et à mesure qu'elles arrivent et les traitent immédiatement. Les flux de travail batch (par lots) sont exécutés moins fréquem-

ment et traitent un grand nombre de requêtes en même temps. Examinons tout d'abord les flux de travail en streaming.

Application ou API de streaming

L'approche par streaming considère un modèle comme un point de terminaison auquel les utilisateurs peuvent envoyer des requêtes. Dans ce contexte, il peut s'agir des utilisateurs finaux d'une application, ou bien d'un service interne qui s'appuie sur les prédictions d'un modèle. Par exemple, un modèle qui prédit le trafic d'un site Web pourrait être employé par un service interne chargé d'ajuster le nombre de serveurs disponibles pour qu'il corresponde au nombre d'utilisateurs prévu.

Dans une application de streaming, le chemin de code d'une requête passe par un ensemble d'étapes qui sont les mêmes que le pipeline d'inférence que nous avons traité dans la section « Commencer par un pipeline simple » du Chapitre 2. Pour rappel, ces étapes sont les suivantes :

1. Valider la requête. Vérifiez les valeurs des paramètres passés, et éventuellement vérifiez si l'utilisateur a les autorisations correctes pour l'exécution de ce modèle.
2. Rassembler des données supplémentaires. Interroger d'autres sources de données pour obtenir les données supplémentaires dont nous pourrions avoir besoin, comme par exemple des informations relatives à un utilisateur.
3. Prétraiter les données.
4. Exécuter le modèle.
5. Post-traiter les résultats. Vérifiez que les résultats se situent dans des limites acceptables. Ajoutez un contexte pour le rendre compréhensible à l'utilisateur, par exemple en expliquant le degré de confiance d'un modèle.
6. Renvoyer un résultat.

Vous pouvez voir cette séquence d'étapes illustrée sur la Figure 9.1.

L'approche par point de terminaison est rapide à implémenter, mais elle nécessite une infrastructure qui s'adapte de manière linéaire au nombre courant d'utilisateurs, puisque chaque utilisateur entraîne un appel d'inférence distinct. Si le trafic augmente au-delà de la capacité d'un serveur à traiter les demandes, celles-ci commenceront à être retardées, voire à échouer. L'adaptation d'un tel pipeline à la structure du trafic exige donc de pouvoir lancer et arrêter facilement de nouveaux serveurs, ce qui nécessitera un certain niveau d'automatisation.

Toutefois, pour une simple démonstration telle que l'Éditeur ML, qui n'est destiné à être visité que par quelques utilisateurs à la fois, une approche en streaming est généralement un bon choix. Pour déployer notre Éditeur ML, nous utilisons une application Web légère en Python telle que Flask, qui permet de mettre en place facilement une API pour servir un modèle avec quelques lignes de code.

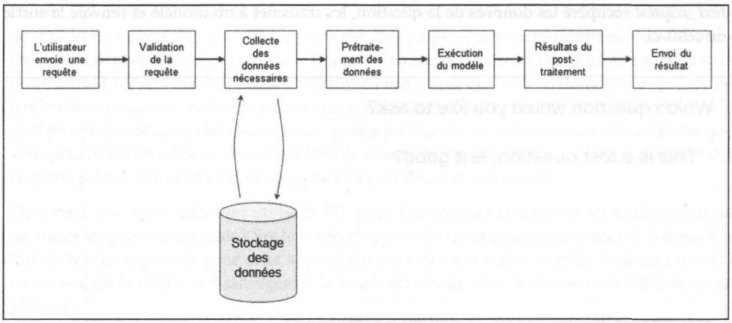


Figure 9.1 : Flux de travail de l'API de diffusion en streaming.

Vous pouvez trouver le code de déploiement du prototype dans le dépôt GitHub du livre, mais je vais vous donner un aperçu de haut niveau ici. L'application Flask se compose de deux parties, une API qui prend en charge les requêtes et les envoie à un modèle pour qu'elles soient traitées à l'aide de Flask, et un simple site Web construit en HTML pour que les utilisateurs puissent saisir leur texte et afficher les résultats. La définition d'une telle API ne nécessite pas beaucoup de code. Ici, vous pouvez voir deux fonctions qui prennent en charge la majeure partie du travail pour servir la version 3 de l'Éditeur ML :

```
from flask import Flask, render_template, request

@app.route("/v3", methods=["POST", "GET"])
def v3():
    return handle_text_request(request, "v3.html")

def handle_text_request(request, template_name):
    if request.method == "POST":
        question = request.form.get("question")
        suggestions = get_recommendations_from_input(question)
        payload = {"input": question, "suggestions": suggestions}
        return render_template("results.html", ml_result=payload)
    else:
        return render_template(template_name)
```

La fonction `v3` définit un routage, ce qui lui permet de déterminer le code HTML à afficher lorsqu'un utilisateur accède à la page `/v3`. Elle utilise la fonction `handle_text_request` pour décider ce qui doit être affiché. Lorsqu'un utilisateur accède à la page pour la première fois, le type de requête est GET et la fonction affiche donc un modèle HTML. Une capture d'écran de cette page HTML est présentée sur la Figure 9.2. Si un utilisateur clique sur le bouton « Get recommandation » (Obtenir une recommandation), le type de requête devient POST, donc *handle_*

text_request récupère les données de la question, les transmet à un modèle et renvoie la sortie de celui-ci.

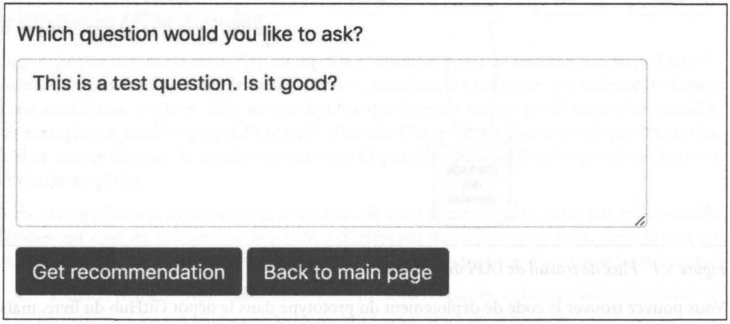


Figure 9.2 : Page Web simple pour utiliser un modèle.

Une application de streaming est nécessaire lorsqu'il existe de fortes contraintes de latence. Si les informations dont un modèle a besoin ne sont disponibles qu'au moment de la prédiction, et que la prédiction du modèle doive être immédiate, vous aurez besoin d'une approche de streaming. Par exemple, un modèle qui prédit le prix d'un trajet spécifique dans une application de VTC nécessite des informations sur la localisation de l'utilisateur et la disponibilité actuelle des conducteurs afin de faire une prédiction, qui n'est disponible qu'au moment de la requête. Un tel modèle doit également produire une prédiction immédiatement, puisqu'elle doit être affichée à l'utilisateur pour qu'il puisse décider s'il veut utiliser le service.

Dans certains autres cas, les informations nécessaires au calcul des prédictions sont disponibles à l'avance. Il peut alors être plus facile de traiter un grand nombre de requêtes à la fois plutôt qu'au fur et à mesure de leur arrivée. C'est ce que l'on appelle la *prédiction par lots*, ou *batch*, que nous allons aborder maintenant.

Prédictions par lots

L'approche par lots considère le pipeline d'inférence comme un travail qui peut être exécuté sur plusieurs échantillons à la fois. Un travail par lots exécute un modèle sur de nombreux échantillons et stocke les prédictions afin qu'elles puissent être utilisées lorsque c'est nécessaire. Ce mode de fonctionnement est approprié lorsque vous avez accès aux caractéristiques nécessaires pour un modèle avant que la prédiction du modèle ne soit requise.

Par exemple, supposons que vous souhaitez construire un modèle pour fournir à chaque vendeur de votre équipe une liste d'entreprises qui sont les prospects les plus intéressants à contacter. Il s'agit d'un problème courant de ML appelé *lead scoring*. Pour entraîner un tel modèle, vous pourriez utiliser des caractéristiques telles que l'historique des conversations par e-mail et les tendances du marché. De telles caractéristiques sont disponibles avant qu'un vendeur ne décide quel prospect contacter, c'est-à-dire avant qu'une prédiction ne soit nécessaire. Cela signifie que vous pourriez faire calculer la nuit une liste de prospects dans un travail par lots, et disposer des résultats prêts à être affichés le matin, quand les vendeurs en ont besoin.

De même, une application qui utilise le ML pour hiérarchiser et classifier les notifications de messages les plus importantes à lire le matin n'a pas de fortes exigences en termes de latence. Un flux de travail approprié pour cette application consisterait à traiter tous les messages non lus en un seul lot le matin, et à sauvegarder la liste hiérarchisée pour le moment où l'utilisateur en a besoin.

Une approche par lots nécessite autant de cycles d'inférence qu'une approche de streaming, mais elle peut être plus efficace en termes de ressources. Comme les prédictions sont faites à un moment prédéterminé, et que le nombre de prédictions est connu au début d'un lot, il est plus facile d'allouer et de paralléliser les ressources. En outre, une approche par lots peut être plus rapide au moment de l'inférence, puisque les résultats ont été précalculés et qu'ils ont simplement besoin d'être récupérés. Cette méthode offre des avantages similaires à ceux d'une mise en cache.

La Figure 9.3 montre les deux côtés de ce flux de travail. Au moment du traitement par lots, nous calculons des prédictions pour tous les points de données et nous stockons les résultats que nous produisons ainsi. Au moment de l'inférence, nous récupérons les résultats précalculés.

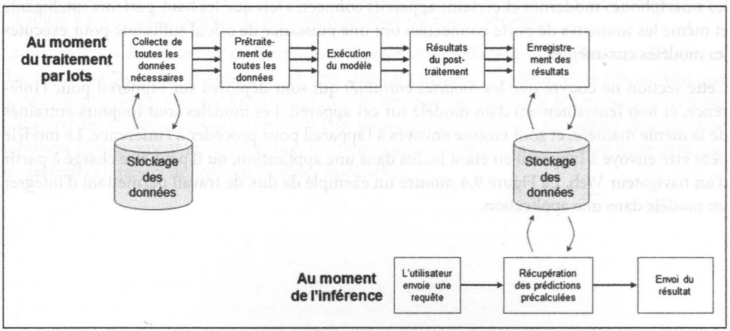


Figure 9.3 : Exemple de flux de travail par lots.

Il est également possible d'utiliser une approche hybride. Dans le plus grand nombre de cas possibles, il faut effectuer un calcul préalable et, au moment de l'inférence, soit récupérer les résultats précalculés, soit les calculer sur place s'ils ne sont pas disponibles ou s'ils sont dépassés. Une telle approche permet d'obtenir des résultats aussi rapidement que possible, puisque tout ce qui peut être calculé à l'avance le sera. Elle s'accompagne du coût de la maintenance à la fois d'un pipeline par lots et d'un pipeline de streaming, ce qui accroît de manière significative la complexité d'un système.

Nous avons abordé deux méthodes courantes de déploiement d'applications sur un serveur : streaming et batch. Ces deux approches exigent que les serveurs d'hébergement exécutent des inférences pour les clients, ce qui peut rapidement devenir coûteux si un produit devient populaire. En outre, ces serveurs représentent un point de défaillance central pour votre application. Si la demande de prédictions augmente soudainement, vos serveurs peuvent ne pas être en mesure de répondre à toutes les requêtes.

Vous pouvez également traiter les requêtes directement sur les appareils des clients qui les formulent. L'exécution des modèles sur les appareils des utilisateurs réduit les coûts d'inférence et vous permet de maintenir un niveau de service constant, quelle que soit la popularité de votre application, puisque les clients fournissent les ressources informatiques nécessaires. C'est ce qu'on appelle le *déploiement côté client*.

Déploiement côté client

L'objectif du déploiement de modèles du côté client est d'exécuter tous les calculs sur le client, ce qui élimine la nécessité d'un serveur pour exécuter les modèles. Les ordinateurs, les tablettes, les smartphones modernes et certains appareils connectés tels que les haut-parleurs intelligents et même les sonnettes de porte connectées ont une puissance de calcul suffisante pour exécuter les modèles eux-mêmes.

Cette section ne couvre que les *modèles entraînés* qui sont déployés sur l'appareil pour l'inférence, et non l'entraînement d'un modèle sur cet appareil. Les modèles sont toujours entraînés de la même manière, et sont ensuite envoyés à l'appareil pour procéder à l'inférence. Le modèle peut être envoyé à l'appareil en étant inclus dans une application, ou il peut être chargé à partir d'un navigateur Web. La Figure 9.4 montre un exemple de flux de travail permettant d'intégrer un modèle dans une application.

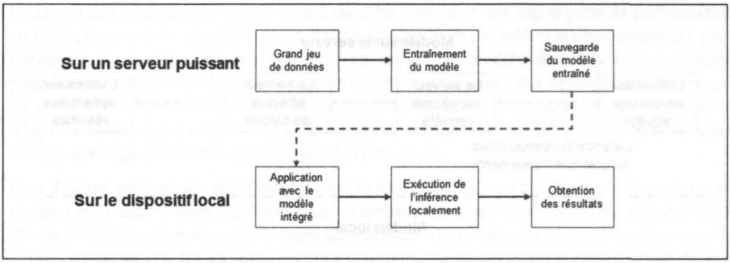


Figure 9.4 : Un modèle exécutant une inférence sur un dispositif (on peut toujours effectuer l'entraînement sur un serveur).

Les appareils portables offrent une puissance de calcul plus limitée que les serveurs puissants. Cette approche limite donc la complexité des modèles qui peuvent être utilisés, mais le fait de faire tourner les modèles sur l'appareil peut offrir de multiples avantages.

Premièrement, cela réduit la nécessité de construire une infrastructure capable d'effectuer des inférences pour chaque utilisateur. De plus, l'exécution de modèles sur des appareils réduit la quantité de données à transférer entre ces dispositifs et le serveur. Cela réduit la latence du réseau, et peut même permettre à une application de s'exécuter sans aucun accès au réseau.

Enfin, si les données nécessaires à l'inférence contiennent des informations sensibles, le fait de faire fonctionner un modèle sur un appareil supprime la nécessité de transférer ces données vers un serveur distant. Le fait de ne pas avoir de données sensibles sur les serveurs réduit le risque qu'un tiers non autorisé accède à celles-ci (voir la section « Les données et leurs problèmes » du Chapitre 8 pour savoir pourquoi cela peut constituer un risque sérieux).

La Figure 9.5 compare le flux de travail pour fournir une prédiction à un utilisateur dans le cas des modèles côté serveur d'une part, et côté client d'autre part. En haut, vous pouvez voir que le délai le plus long pour un flux de travail côté serveur est souvent le temps nécessaire pour transférer les données vers le serveur. En bas, vous constatez que, si les modèles côté client ne subissent pratiquement aucune latence, ils traitent souvent les échantillons plus lentement que les serveurs en raison de leurs contraintes matérielles.

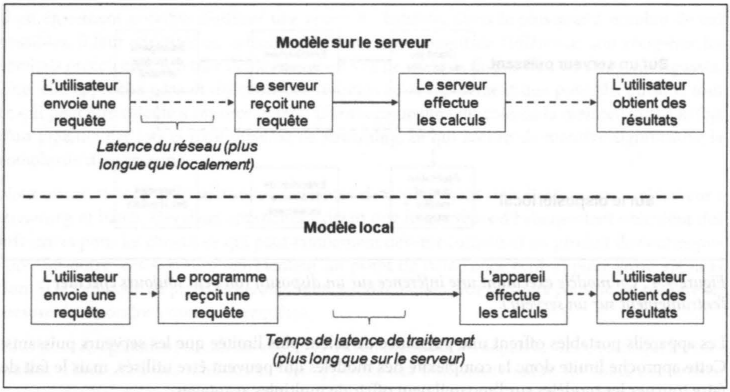


Figure 9.5 : Exécution sur un serveur, ou localement.

Tout comme pour le déploiement côté serveur, il existe de multiples façons de déployer des applications côté client. Dans les sections suivantes, nous couvrirons deux méthodes, en déployant les modèles de manière native et en les exécutant par le biais du navigateur. Ces approches sont pertinentes pour les smartphones et les tablettes, qui ont accès à une boutique d'applications et à un navigateur Web, mais pas pour d'autres dispositifs connectés tels que les microcontrôleurs, que nous n'aborderons pas ici.

Sur l'appareil

Les processeurs des ordinateurs portables et des smartphones ne sont généralement pas optimisés pour exécuter les modèles de ML, et ils exécuteront donc un pipeline d'inférence plus lentement. Pour qu'un modèle côté client fonctionne rapidement et sans trop de puissance, il doit être aussi petit que possible.

La réduction de la taille du modèle peut se faire en utilisant un modèle plus simple, en réduisant son nombre de paramètres ou encore en limitant la précision des calculs. Dans les réseaux de neurones, par exemple, les poids sont souvent élagués (notamment en éliminant ceux dont les valeurs sont proches de zéro) et quantifiés (en diminuant la précision des poids). Vous pouvez également réduire le nombre de caractéristiques utilisées par votre modèle pour accroître encore son efficacité. Ces dernières années, des bibliothèques telles que Tensorflow Lite (`https://www.tensorflow.org/lite`) ont commencé à fournir des outils pratiques pour réduire la taille des modèles et contribuer à les rendre plus facilement déployables sur les appareils mobiles.

En raison de ces exigences, la plupart des modèles vont subir une légère perte de performance en étant portés sur un appareil. Les produits qui ne peuvent pas tolérer la dégradation des performances des modèles, tels que ceux qui reposent sur des modèles de pointe trop complexes pour être exécutés sur un dispositif tel qu'un smartphone, doivent être déployés sur un serveur. En général, si le temps nécessaire pour exécuter l'inférence sur un appareil est plus long que le temps nécessaire pour transmettre les données au serveur afin qu'elles y soient traitées, vous devriez envisager d'exécuter votre modèle dans un cloud.

Pour d'autres applications, telles que les claviers prédictifs des smartphones qui offrent des suggestions pour aider à taper plus rapidement, l'intérêt d'avoir un modèle local qui n'a pas besoin d'accès à l'Internet l'emporte sur la perte de précision. De même, une application pour smartphone conçue pour aider les randonneurs à identifier des plantes en les photographiant devrait fonctionner hors ligne pour pouvoir être utilisée lors d'une sortie. Une telle application nécessiterait le déploiement d'un modèle sur l'appareil, quitte à sacrifier l'exactitude des prédictions.

Une application de traduction est un autre exemple de produit alimenté par ML qui bénéficie d'un fonctionnement local. Une telle application est susceptible d'être utilisée dans un pays étranger, où les utilisateurs n'ont pas forcément accès au réseau. Avoir un modèle de traduction capable de fonctionner localement devient une exigence, même s'il n'est pas aussi précis qu'un modèle plus complexe qui ne pourrait s'exécuter que sur un serveur.

Outre les problèmes de réseau, exécuter des modèles dans le cloud ajoute un risque pour la vie privée. L'envoi de données des utilisateurs dans le cloud et leur stockage dans celui-ci, même temporaire, augmente les risques pour qu'un attaquant y ait accès. Prenons le cas d'une application en apparence aussi bénigne que la superposition de filtres sur des photos. De nombreux utilisateurs peuvent ne pas se sentir à l'aise avec le fait que leurs photos soient transmises à un serveur pour être traitées et stockées indéfiniment. Pouvoir garantir aux utilisateurs que leurs photos ne quittent jamais l'appareil est un élément de différenciation important dans un monde de plus en plus soucieux du respect de la vie privée. Comme nous l'avons vu dans la section « Les données et leurs problèmes » du Chapitre 8, la meilleure façon d'éviter de mettre en danger les données sensibles est de s'assurer qu'elles ne quittent jamais l'appareil, ou qu'elles ne sont jamais stockées sur vos serveurs.

D'autre part, la quantification de l'élagage et de la simplification d'un modèle est un processus qui prend du temps. Le déploiement sur des appareils ne vaut la peine que si les avantages en termes de latence, d'infrastructure et de respect de la vie privée sont suffisamment importants pour investir dans l'effort d'ingénierie. Pour l'Éditeur ML, nous nous limiterons à une API de streaming basée sur le Web.

Enfin, l'optimisation des modèles spécifiquement pour qu'ils fonctionnent sur un certain type de dispositif risque de prendre beaucoup de temps, car le processus d'optimisation peut différer d'un appareil à un autre. Il existe d'autres options pour exécuter les modèles localement, notamment celles qui exploitent les points communs entre les appareils afin de réduire le travail d'ingénierie nécessaire. Un domaine passionnant à ce sujet est l'exécution du ML dans le navigateur.

Côté navigateur

La plupart des appareils intelligents ont accès à un navigateur. Ces navigateurs ont souvent été optimisés pour permettre des calculs graphiques rapides. Cela a conduit à un intérêt croissant pour les bibliothèques qui utilisent des navigateurs afin que le client effectue des tâches de ML.

Le plus populaire de ces *frameworks* est Tensorflow.js (`https://www.tensorflow.org/js`), qui permet d'entraîner et d'exécuter l'inférence en JavaScript dans le navigateur pour la plupart des modèles différenciables, même ceux qui ont été entraînés dans des langages différents comme Python.

Cela permet aux utilisateurs d'interagir avec les modèles par le biais du navigateur sans avoir à installer d'applications supplémentaires. En outre, comme les modèles s'exécutent dans le navigateur en utilisant JavaScript, les calculs sont effectués sur l'appareil de l'utilisateur. Votre infrastructure n'a besoin de servir que la page Web qui contient les poids des modèles. Enfin, Tensorflow.js prend en charge WebGL, ce qui lui permet d'exploiter les GPU sur l'appareil du client, s'ils sont disponibles bien sûr, pour effectuer des calculs plus rapides.

L'utilisation d'un framework JavaScript facilite le déploiement d'un modèle côté client sans exiger autant de travail spécifique au dispositif que l'approche précédente. *A contrario*, cela présente l'inconvénient d'augmenter les coûts en termes de bande passante, puisque le modèle devra être téléchargé par les clients à chaque fois qu'ils ouvrent la page, au lieu d'être téléchargé une seule fois lorsqu'ils installent l'application.

Tant que les modèles que vous utilisez ne pèsent que quelques mégaoctets ou moins et qu'ils peuvent être téléchargés rapidement, l'utilisation de JavaScript pour les exécuter sur le client peut être un moyen utile de réduire les coûts du serveur. Si un tel coût devenait un jour un problème pour l'Éditeur ML, le déploiement du modèle à l'aide d'un framework comme Tensorflow.js serait l'une des premières méthodes que je recommanderais d'explorer.

Jusqu'à présent, nous avons considéré les clients uniquement pour déployer des modèles déjà entraînés, mais nous pourrions aussi décider d'entraîner des modèles sur des postes client. Dans la prochaine partie, nous examinerons quand cela pourrait être utile.

Apprentissage fédéré : une approche hybride

Nous avons surtout abordé les différentes façons de déployer des modèles que nous avons déjà entraînés (idéalement en suivant les directives des chapitres précédents), après quoi nous choisissons la méthode de déploiement. Nous avons examiné différentes solutions pour obtenir un modèle unique pour tous nos utilisateurs, mais que se passerait-il si nous voulions que chaque utilisateur ait un modèle différent ?

La Figure 9.6 montre la différence entre un système piloté par le haut, qui a un modèle entraîné commun à tous les utilisateurs, et un système piloté par le bas, où chaque utilisateur a une version légèrement différente du modèle.

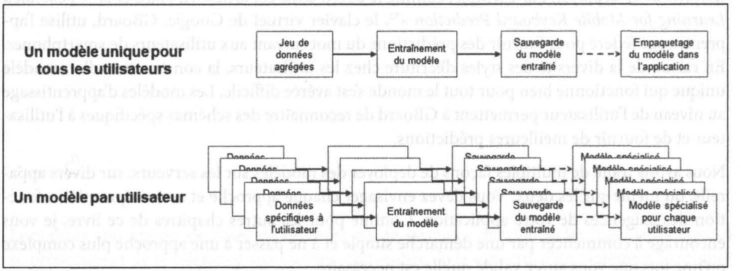


Figure 9.6 : Un grand modèle ou de multiples modèles individuels.

Pour de nombreuses applications, telles que la recommandation de contenu, la rédaction de conseils ou de soins de santé, la source d'information la plus importante d'un modèle est les données dont il dispose sur l'utilisateur. Nous pouvons exploiter ce fait en générant des caractéristiques pour un modèle qui soient spécifiques à l'utilisateur, ou nous pouvons décider que chaque utilisateur doit avoir son propre modèle. Ces modèles peuvent tous partager la même architecture, mais ils auront des valeurs de paramètres différentes qui reflètent les données individuelles de chaque utilisateur.

Cette idée est au cœur de l'*apprentissage fédéré*, un domaine de l'apprentissage profond qui a fait l'objet d'une attention croissante ces derniers temps avec des projets tels que OpenMined (`https://www.openmined.org/`). Dans l'apprentissage fédéré, chaque client a son propre modèle. Chaque modèle apprend à partir des données de son utilisateur, et envoie des mises à jour agrégées (et potentiellement anonymisées) au serveur. Le serveur exploite toutes les mises à jour pour améliorer son modèle, et il transmet ce nouveau modèle aux clients individuels.

Chaque utilisateur reçoit un modèle qui est personnalisé en fonction de ses besoins, tout en bénéficiant d'informations agrégées sur les autres utilisateurs. L'apprentissage fédéré améliore la confidentialité des utilisateurs, car leurs données ne sont jamais transférées au serveur, qui ne reçoit de son côté que des mises à jour agrégées des modèles. Cela contraste avec l'apprentissage d'un modèle par la méthode traditionnelle qui consiste à collecter des données sur chaque utilisateur et à les stocker toutes sur un serveur.

L'apprentissage fédéré est une direction passionnante pour le ML, mais il ajoute une couche de complexité supplémentaire. S'assurer que chaque modèle individuel fonctionne bien et que les

données transmises au serveur sont correctement anonymisées est plus compliqué que d'entraîner un seul modèle.

L'apprentissage fédéré est déjà utilisé dans des applications pratiques par des équipes qui ont les moyens de le déployer. Par exemple, comme le décrit dans cet article A. Hard *et al.* « *Federated Learning for Mobile Keyboard Prediction* »[52], le clavier virtuel de Google, GBoard, utilise l'apprentissage fédéré pour fournir des prédictions du mot suivant aux utilisateurs de smartphones. En raison de la diversité des styles d'écriture chez les utilisateurs, la construction d'un modèle unique qui fonctionne bien pour tout le monde s'est avérée difficile. Les modèles d'apprentissage au niveau de l'utilisateur permettent à GBoard de reconnaître des schémas spécifiques à l'utilisateur et de fournir de meilleures prédictions.

Nous avons traité de multiples façons de déployer des modèles sur les serveurs, sur divers appareils, ou même sur les deux. Vous devez envisager chaque approche et ses compromis en fonction des exigences de votre application. Comme pour les autres chapitres de ce livre, je vous encourage à commencer par une démarche simple et à ne passer à une approche plus complexe qu'une fois que vous aurez validé qu'elle est nécessaire.

Conclusion

Il existe de multiples façons de servir une application alimentée par le ML. Vous pouvez mettre en place une API de streaming pour permettre à un modèle de traiter les échantillons au fur et à mesure de leur arrivée. Vous pouvez aussi utiliser un flux de travail par lots qui traitera plusieurs points de données à la fois selon une programmation régulière. Vous pouvez également choisir de déployer vos modèles côté client en les regroupant dans une application ou en les diffusant par le biais d'un navigateur Web. Cela réduirait vos coûts d'inférence et vos besoins d'infrastructure, mais rendrait votre processus de déploiement plus complexe.

La bonne approche dépend des besoins de votre application, tels que les exigences en matière de latence, les problèmes de matériel, de réseau et de confidentialité, ainsi que les coûts d'inférence. Pour un prototype simple comme notre Éditeur ML, commencez par un point de terminaison ou un simple flux de travail par lots, et itérez à partir de là.

Mais le déploiement d'un modèle ne se limite pas à son exposition aux utilisateurs. Dans le Chapitre 10, nous aborderons les méthodes permettant d'établir des garanties autour des modèles afin d'atténuer les erreurs, les outils d'ingénierie aidant à rendre le processus de déploiement plus efficace et les approches servant à valider le fait que les modèles fonctionnent comme ils le devraient.

52 Voir l'adresse `https://arxiv.org/abs/1811.03604`.

CHAPITRE 10

Protéger les modèles

Lors de la conception de bases de données ou de systèmes distribués, les ingénieurs logiciel se préoccupent de la tolérance aux pannes, c'est-à-dire de la capacité d'un système à continuer à fonctionner lorsque certains de ses composants tombent en panne. Dans le domaine des logiciels, la question n'est pas de savoir si une partie donnée du système va tomber en panne, mais quand. Les mêmes principes peuvent être appliqués au ML. Quelle que soit la qualité d'un modèle, il échouera sur certains échantillons, et vous devez donc concevoir un système capable de gérer ces défaillances avec grâce.

Dans ce chapitre, nous aborderons différentes manières de contribuer à prévenir ou à atténuer les échecs. Tout d'abord, nous verrons comment vérifier la qualité des données que nous recevons et produisons, et utiliser cette vérification pour décider de la manière d'afficher les résultats aux utilisateurs. Ensuite, nous examinerons les moyens de rendre un pipeline de modélisation plus robuste pour pouvoir servir efficacement de nombreux utilisateurs. Puis nous aborderons les options permettant de tirer parti des commentaires des utilisateurs et de juger de la performance d'un modèle. Nous terminerons ce chapitre par une interview de Chris Moody sur les meilleures pratiques de déploiement.

Ingénierie autour des défaillances

Examinons quelques-unes des causes les plus probables de défaillance d'un pipeline de ML. Le lecteur attentif remarquera que ces cas de défaillance sont quelque peu similaires aux conseils de débogage que nous avons vus dans la section « Câbler le débogage : visualisation et test » du Chapitre 6. En effet, l'exposition d'un modèle aux utilisateurs en production s'accompagne d'un ensemble de défis qui reflètent ceux qui accompagnent le débogage d'un modèle.

Les bogues et les erreurs peuvent apparaître n'importe où, mais trois domaines en particulier sont les plus importants à vérifier : les entrées d'un pipeline, la confiance d'un modèle, et les sorties qu'il produit. Examinons chacun d'entre eux dans l'ordre.

Contrôles des entrées et des sorties

Tout modèle a été entraîné sur un jeu de données spécifique présentant des caractéristiques particulières. Les données d'entraînement présentaient un certain nombre de caractéristiques, et chacune d'entre elles était d'un certain type. En outre, chaque caractéristique suivait une certaine distribution que le modèle apprenait afin de fonctionner correctement.

Comme nous l'avons vu dans la section « Fraîcheur et changements dans la distribution » du Chapitre 2, si les données de production sont différentes des données sur lesquelles un modèle a été entraîné, celui-ci peut avoir du mal à fonctionner. Pour y remédier, vous devez vérifier les données d'entrée de votre pipeline.

Vérifier les entrées

Certains modèles peuvent encore donner de bons résultats lorsqu'ils sont confrontés à de petites différences dans la distribution des données. Toutefois, si un modèle reçoit des données très différentes de celles qu'il a vues lors de l'entraînement ou si certaines caractéristiques sont manquantes ou d'un type inattendu, il aura du mal à fonctionner.

Comme nous l'avons vu précédemment, les modèles de ML sont capables de fonctionner même lorsqu'ils reçoivent des entrées incorrectes (à condition que ces entrées soient de la bonne forme et du bon type). Les modèles produiront des sorties, mais ces sorties peuvent être largement incorrectes. Prenons l'exemple illustré sur la Figure 10.1. Un pipeline classifie une phrase dans l'un des deux sujets en la vectorisant d'abord, et en appliquant un modèle de classification sur la représentation ainsi vectorisée. Si le pipeline reçoit une chaîne de caractères aléatoires, il la transformera quand même en un vecteur, et le modèle fera une prédiction. Cette prédiction est absurde, mais il n'y a aucun moyen de le savoir uniquement en regardant les résultats du modèle.

Pour éviter qu'un modèle ne débouche sur des sorties incorrectes, nous devons détecter que ces entrées sont incorrectes avant de les passer au modèle.

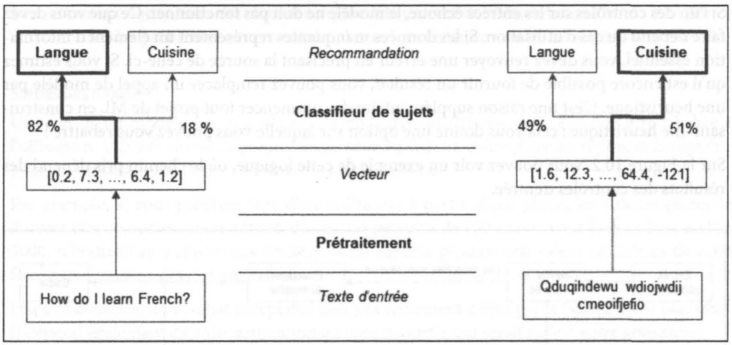


Figure 10.1 : Les modèles produiront toujours une prédiction pour des entrées aléatoires.

Contrôles et tests

Dans cette section, nous parlons des contrôles d'entrée, par opposition aux tests d'entrée que nous avons vus dans la section « Tester votre code ML » du Chapitre 6. La différence est subtile mais importante. Les tests valident le fait que le code se comporte comme prévu compte tenu d'une entrée connue et prédéterminée. Ils sont exécutés généralement à chaque changement de code ou de modèle, pour valider qu'un pipeline fonctionne toujours correctement. Les contrôles d'entrée abordés dans cette section font partie du pipeline lui-même et modifient le flux de contrôle d'un programme en fonction de la qualité des intrants. Les contrôles sur les intrants qui échouent peuvent entraîner l'utilisation d'un modèle différent, ou la non-exécution totale d'un modèle.

Les contrôles portent sur des domaines similaires à ceux des tests présentés dans la section « Tester votre code de ML » du Chapitre 6. Par ordre d'importance, ils vont :

1. Vérifier que toutes les caractéristiques nécessaires sont présentes.
2. Contrôler chaque type d'élément.
3. Valider les valeurs des caractéristiques.

Il peut être difficile de vérifier les valeurs des caractéristiques de manière isolée, car la distribution de ces caractéristiques peut être complexe. Un moyen simple d'effectuer une telle validation consiste à définir une fourchette raisonnable de valeurs qu'une caractéristique pourrait prendre, et à confirmer qu'elle se situe dans cette fourchette.

Si l'un des contrôles sur les entrées échoue, le modèle ne doit pas fonctionner. Ce que vous devez faire dépend du cas d'utilisation. Si les données manquantes représentent un élément d'information essentiel, vous devez renvoyer une erreur en précisant la source de celle-ci. Si vous estimez qu'il est encore possible de fournir un résultat, vous pouvez remplacer un appel de modèle par une heuristique. C'est une raison supplémentaire de commencer tout projet de ML en construisant une heuristique : cela vous donne une option sur laquelle vous pouvez vous rabattre !

Sur la Figure 10.2, vous pouvez voir un exemple de cette logique, où le chemin pris dépend des résultats des contrôles d'entrée.

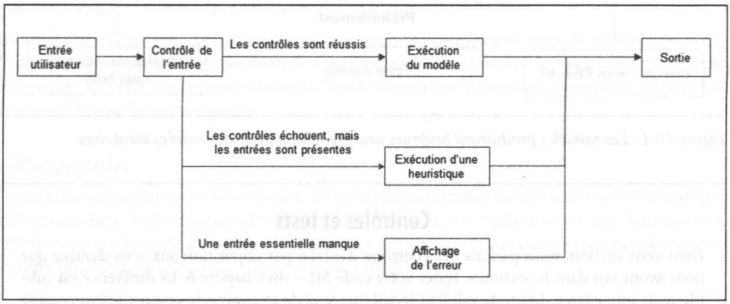


Figure 10.2 : Exemple de logique de branchement pour les contrôles d'entrée.

Voici un exemple de logique de flux de contrôle de l'Éditeur ML qui vérifie les caractéristiques manquantes et les types des caractéristiques. Selon la qualité de l'entrée, le code génère une erreur ou exécute une heuristique. J'ai copié l'exemple ici, mais vous pouvez également le trouver dans le dépôt GitHub de ce livre avec le reste du code de l'Éditeur ML.

```
def validate_and_handle_request(question_data):
    missing = find_absent_features(question_data)
    if len(missing) > 0:
        raise ValueError("Missing feature(s) %s" % missing)

    wrong_types = check_feature_types(question_data)
    if len(wrong_types) > 0:
        # Si les données sont erronées mais que nous avons
        # la longueur de la question, exécuter une heuristique
        if "text_len" in question_data.keys():
            if isinstance(question_data["text_len"], float):
                return run_heuristic(question_data["text_len"])
        raise ValueError("Incorrect type(s) %s" % wrong_types)

    return run_model(question_data)
```

La vérification des entrées des données du modèle permet de réduire les modes de défaillance et d'identifier les problèmes sur ces entrées. Vous devez ensuite valider les sorties d'un modèle.

Sorties des modèles

Une fois qu'un modèle fait une prédiction, vous devriez déterminer si elle doit être affichée à l'utilisateur. Dans le cas où la prédiction se situe en dehors d'une plage de réponses acceptable pour un modèle, vous devez envisager de ne pas l'afficher.

Par exemple, si vous prédisez l'âge d'un utilisateur à partir d'une photo, les valeurs en sortie doivent être comprises entre zéro et, disons, un peu plus de 100 ans (si vous lisez ce livre en l'an 3000, n'hésitez pas à ajuster ces limites). Si un modèle produit une valeur en dehors de cette fourchette, vous ne devriez pas l'afficher.

Dans ce contexte, un résultat acceptable n'est pas seulement défini par le fait qu'il soit plausible. Il dépend également de votre estimation du type de sortie qui serait *utile à notre utilisateur*.

Pour notre Éditeur ML, nous voulons seulement fournir des recommandations qui soient réalisables. Si un modèle prévoit que tout ce qu'un utilisateur a écrit doit être entièrement supprimé, il s'agirait d'une recommandation plutôt inutile (et même insultante). Voici un exemple d'extrait validant les sorties du modèle et revenant à une heuristique si nécessaire :

```
def validate_and_correct_output(question_data, model_output):
    # Vérifie le type et la plage et signale les erreurs en conséquence
    try:
        # Lève une valeur d'erreur si la sortie du modèle est incorrecte
        verify_output_type_and_range(model_output)
    except ValueError:
        # Nous exécutons une heuristique, mais nous pourrions utiliser
        # ici un modèle différent
        run_heuristic(question_data["text_len"])

    # Si nous n'avons pas levé d'erreur, nous renvoyons
    # le résultat de notre modèle
    return model_output
```

Lorsqu'un modèle échoue, vous pouvez revenir à une heuristique, comme nous l'avons vu précédemment, ou à un modèle plus simple que vous avez peut-être construit plus tôt. Il est souvent utile d'essayer un type de modèle antérieur, car différents modèles peuvent présenter des erreurs non corrélées.

J'ai illustré cela sur un exemple jouet dans la Figure 10.3. Sur la gauche, vous pouvez voir un modèle plus performant avec une frontière de décision plus complexe. Sur la droite, vous pouvez voir un modèle moins performant et plus simple. Le modèle le plus mauvais fait plus d'erreurs, mais ses erreurs sont différentes de celles du modèle complexe en raison de la forme différente de sa frontière de décision. De ce fait, le modèle le plus simple donne quelques bons échantillons que le modèle complexe donne comme étant mauvais. C'est l'intuition qui explique pourquoi il

est raisonnable d'utiliser un modèle simple comme solution de secours lorsqu'un modèle primaire échoue.

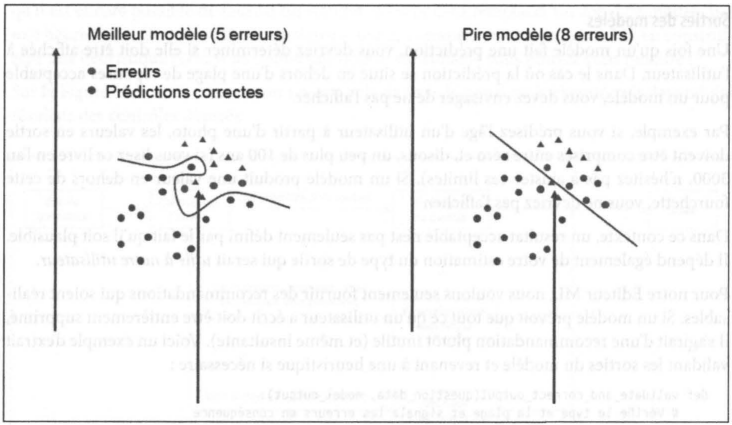


Figure 10.3 : Un modèle plus simple fait souvent des erreurs différentes.

Si vous utilisez un modèle plus simple comme sauvegarde, vous devriez également valider ses sorties de la même manière, et revenir à une heuristique ou afficher une erreur si elles ne passent pas vos contrôles.

Valider le fait que les sorties d'un modèle se situent dans une fourchette raisonnable est un bon début, mais ce n'est pas suffisant. Dans la prochaine section, nous aborderons les protections supplémentaires que nous pouvons mettre en place autour d'un modèle.

Défaillances des modèles

Nous avons mis en place des dispositifs de sécurité pour détecter et corriger les entrées et sorties erronées. Dans certains cas, cependant, l'entrée de notre modèle peut être correcte, et la sortie du même modèle peut être raisonnable tout en étant totalement erronée.

Pour revenir à l'exemple de la prédiction de l'âge d'un utilisateur à partir d'une photo, garantir que l'âge prédit par le modèle est un âge humain plausible est un bon début, mais dans l'idéal, nous aimerions prédire l'âge *correct* pour cet utilisateur spécifique.

Aucun modèle ne sera correct à 100 % et de légères erreurs peuvent souvent être acceptables, mais, dans la mesure du possible, vous devriez chercher à détecter quand un modèle se trompe. Cela vous permet de signaler éventuellement un certain cas comme étant trop difficile, et donc d'encourager les utilisateurs à fournir une meilleure contribution (sous la forme d'une photo bien éclairée, par exemple).

Il existe deux approches principales pour détecter les erreurs. La plus simple consiste à suivre la confiance d'un modèle pour estimer si un résultat sera exact. La seconde consiste à construire un modèle supplémentaire chargé de détecter les échantillons sur lesquels un modèle principal est susceptible d'échouer.

Pour la première méthode, les modèles de classification peuvent produire une probabilité susceptible d'être utilisée comme une estimation de la confiance du modèle dans ses résultats. Si ces probabilités sont bien étalonnées (voir la section « Courbe d'étalonnage » dans le Chapitre 5), elles peuvent servir à détecter les cas où un modèle est incertain, et à décider de ne pas afficher les résultats à un utilisateur.

Parfois, les modèles se trompent, même s'ils attribuent une forte probabilité à un échantillon. C'est là qu'intervient la deuxième approche : utiliser un modèle pour filtrer les entrées les plus difficiles.

Modèle de filtrage

En plus de ne pas toujours être fiable, l'utilisation du score de confiance d'un modèle présente un autre inconvénient de taille. Pour obtenir ce score, l'ensemble du pipeline d'inférence doit être exécuté, que ses prédictions soient utilisées ou non. C'est un gaspillage particulièrement important lorsque l'on utilise des modèles plus complexes qui doivent être exécutés sur un GPU, par exemple. L'idéal serait d'estimer les performances d'un modèle sur un échantillon sans l'exécuter sur celui-ci.

C'est l'idée qui sous-tend les modèles de filtrage. Comme vous savez que certaines entrées seront difficiles à gérer pour un modèle, vous devez les détecter à l'avance et ne pas vous donner du tout la peine d'exécuter un modèle sur elles. Un modèle de filtrage est la version ML des tests d'entrée. Il s'agit d'un classifieur binaire qui est entraîné pour prédire si un modèle fonctionnera bien sur un échantillon donné. L'hypothèse de base d'un tel modèle est qu'il existe dans le type des points de données des tendances qui sont difficiles à traiter pour le modèle principal. Si ces échantillons difficiles ont suffisamment de points communs, le modèle de filtrage peut apprendre à les séparer des entrées plus faciles.

Voici quelques types d'entrées que vous souhaiterez peut-être capter avec un modèle de filtrage :

- Des entrées qui sont qualitativement différentes de celles sur lesquelles le modèle principal fonctionne bien.

- Des entrées sur lesquelles le modèle a été entraîné, mais avec lesquelles il a eu des difficultés.
- Des entrées « pirates » qui visent à tromper le modèle principal.

Sur la Figure 10.4, vous pouvez voir un exemple actualisé de la logique de la Figure 10.2, avec maintenant un modèle de filtrage. Comme vous pouvez le voir, le modèle de filtrage n'est exécuté que si les contrôles d'entrée passent, car il suffit de filtrer les entrées qui auraient pu se retrouver dans la case « Exécuter le modèle ».

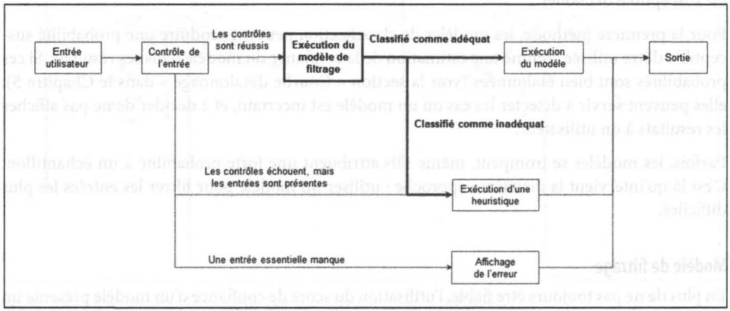


Figure 10.4 : Ajout d'une étape de filtrage à nos contrôles d'entrée (en gras).

Pour entraîner un modèle de filtrage, il vous suffit de rassembler un jeu de données contenant deux catégories d'échantillons : les catégories sur lesquelles votre modèle principal a réussi, et celles sur lesquelles il a échoué. Cela peut être fait en utilisant nos données d'entraînement et ne nécessite aucune collecte de données supplémentaire !

Sur la Figure 10.5, je montre comment procéder en utilisant un modèle entraîné et son résultat sur un jeu de données, comme le montre le graphique de gauche. On échantillonne certains points de données que le modèle a correctement prédit et d'autres sur lesquels le modèle a échoué. Vous pouvez ensuite entraîner un modèle de filtrage pour prédire lesquels de ces points de données sont ceux sur lesquels le modèle original a échoué.

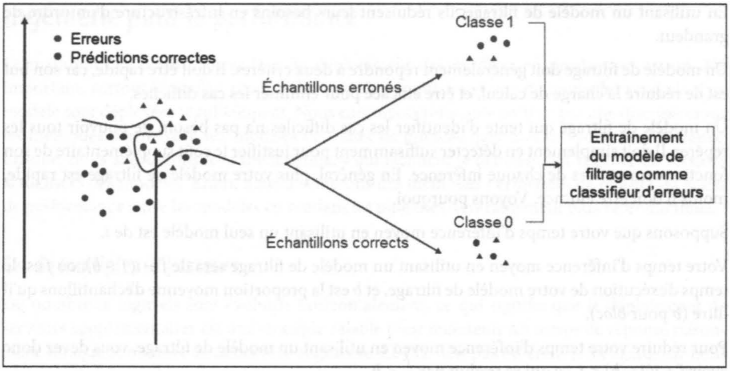


Figure 10.5 : Obtenir des données d'entraînement pour un modèle de filtrage.

Une fois que vous avez un classifieur entraîné, l'entraînement d'un modèle de filtrage peut être relativement simple. Avec un jeu de test et un classificateur entraîné, c'est ce que vous permettra de faire la fonction suivante.

```
def get_filtering_model(classifier, features, labels):
    """
    Obtient l'erreur de prédiction pour un jeu de données
    de classification binaire
    :paramètre classifier: classifieur entraîné
    :paramètre features: caractéristiques d'entrée
    :paramètre labels: étiquettes vraies
    """
    predictions = classifier.predict(features)
    # Crée des étiquettes où les erreurs sont à 1,
    " et les suppositions correctes sont à 0
    is_error = [pred != truth for pred, truth in zip(predictions, labels)]

    filtering_model = RandomForestClassifier()
    filtering_model.fit(features, is_error)
    return filtering_model
```

Cette approche est utilisée par Google pour sa fonction Smart Reply, qui suggère quelques réponses courtes à un courriel entrant (voir cet article de A. Kanan *et al.* intitulé « *Smart Reply : Automated Response Suggestion for Email* »[53]). Ils utilisent ce qu'ils appellent un modèle de déclenchement (*triggering model*), chargé de décider s'il faut exécuter le modèle principal qui suggère des réponses. Dans leur cas, seuls environ 11 % des e-mails conviennent à ce modèle.

53 Voir l'adresse `https://bit.ly/2WPvGgh`.

En utilisant un modèle de filtrage, ils réduisent leurs besoins en infrastructure d'un ordre de grandeur.

Un modèle de filtrage doit généralement répondre à deux critères. Il doit être rapide, car son but est de réduire la charge de calcul, et être efficace pour éliminer les cas difficiles.

Un modèle de filtrage qui tente d'identifier les cas difficiles n'a pas besoin de pouvoir tous les repérer. Il doit simplement en détecter suffisamment pour justifier le coût supplémentaire de son fonctionnement lors de chaque inférence. En général, plus votre modèle de filtrage est rapide, moins il doit être efficace. Voyons pourquoi.

Supposons que votre temps d'inférence moyen en utilisant un seul modèle est de i.

Votre temps d'inférence moyen en utilisant un modèle de filtrage sera de $f + i(1 - b)$, où f est le temps d'exécution de votre modèle de filtrage, et b est la proportion moyenne d'échantillons qu'il filtre (b pour *bloc*).

Pour réduire votre temps d'inférence moyen en utilisant un modèle de filtrage, vous devez donc avoir $f + i(1 - b) < i$, ce qui se traduit par $\frac{f}{i} < b$.

Cela signifie que la proportion de cas que votre modèle filtre doit être supérieure au rapport entre sa vitesse d'inférence et la vitesse de votre plus grand modèle.

Par exemple, si votre modèle de filtrage est 20 fois plus rapide que votre modèle ordinaire ($\frac{f}{i} = 5\ \%$), il devrait bloquer plus de 5 % des cas ($5\ \% < b$) pour être utile en production.

Bien sûr, vous voudriez également vous assurer que la précision de votre modèle de filtrage est bonne, ce qui signifie que la majorité des entrées qu'il bloque sont effectivement trop difficiles pour votre modèle principal.

Une façon d'y parvenir serait de laisser régulièrement passer quelques échantillons que votre modèle de filtrage aurait bloqués et d'examiner comment votre modèle principal s'en sort avec eux. Nous aborderons cette question plus en détail dans la section « Choisir ce que vous voulez surveiller » du Chapitre 11.

Comme le modèle de filtrage est différent du modèle d'inférence et qu'il est entraîné spécifiquement pour prédire les cas difficiles, il peut détecter ces cas avec plus d'exactitude qu'en se basant sur la sortie de probabilité du modèle principal. L'utilisation d'un modèle de filtrage permet donc à la fois de réduire la probabilité de mauvais résultats et d'améliorer l'utilisation des ressources.

Pour ces raisons, l'ajout de modèles de filtrage aux contrôles d'entrée et de sortie existants peut accroître considérablement la robustesse d'un pipeline de production. Dans la prochaine section, nous aborderons d'autres moyens de rendre les pipelines robustes en examinant comment adapter les applications de ML à un plus grand nombre d'utilisateurs et comment organiser des processus d'entraînement complexes.

Ingénierie pour la performance

Le maintien des performances lors du déploiement des modèles en production est un défi important, surtout lorsqu'un produit devient plus populaire et que de nouvelles versions d'un modèle sont déployées régulièrement. Nous commencerons cette section par une discussion sur les méthodes permettant aux modèles de traiter de grandes quantités de requêtes d'inférence. Ensuite, nous aborderons les caractéristiques qui facilitent le déploiement régulier de versions actualisées des modèles. Enfin, nous discuterons des méthodes permettant de réduire les écarts de performance entre les modèles en rendant les pipelines d'entraînement plus reproductibles.

Cas de multiples utilisateurs

De nombreux logiciels sont évolutifs horizontalement, ce qui signifie que le déploiement de serveurs supplémentaires est une stratégie valable pour maintenir un temps de réponse raisonnable lorsque le nombre de demandes augmente. Le ML n'est pas différent à cet égard, car nous pouvons simplement ajouter de nouveaux serveurs pour faire fonctionner nos modèles et gérer la capacité supplémentaire.

Si vous utilisez un modèle de deep learning, vous aurez peut-être besoin d'un GPU pour fournir les résultats dans un délai acceptable. Si c'est le cas, et que vous vous attendiez à avoir suffisamment de requêtes pour nécessiter plus qu'une seule machine équipée d'un GPU, vous devriez exécuter votre logique applicative et votre modèle d'inférence sur deux serveurs différents.

Comme les instances de GPU sont souvent d'un ordre de grandeur plus cher que les instances ordinaires pour la plupart des fournisseurs de services dans le cloud, le fait de disposer d'une instance moins chère pour faire évoluer votre application et d'instances de GPU s'occupant uniquement de l'inférence réduira considérablement vos coûts de calcul. Lorsque vous utilisez cette stratégie, vous devez garder à l'esprit que vous introduisez certains frais de communication supplémentaires et vous assurer que cela n'est pas trop préjudiciable à votre cas d'utilisation.

En plus d'augmenter l'allocation des ressources, le ML se prête à des méthodes efficaces pour gérer le trafic supplémentaire, comme la mise en cache.

Mise en cache pour le ML

La mise en cache est la pratique qui consiste à stocker les résultats d'appels de fonction afin que les futurs appels à cette même fonction et avec les mêmes paramètres puissent être exécutés plus rapidement en récupérant simplement les résultats qui ont été enregistrés. La mise en cache est une pratique courante pour accélérer les pipelines d'ingénierie et est très utile pour le ML.

Mise en cache des résultats d'inférence. La méthode LRU (*Least Recently Used*) est une approche simple, qui consiste à garder une trace des entrées les plus récentes d'un modèle et de leurs

sorties correspondantes. Avant d'exécuter le modèle sur une nouvelle entrée, il faut consulter le cache. Si une entrée correspondante y est trouvée, on sert les résultats directement depuis le cache. La Figure 10.6 présente un exemple de ce type de flux de travail. La première ligne représente l'étape de mise en cache lorsqu'une entrée est initialement rencontrée. La deuxième ligne représente l'étape de récupération une fois que la même entrée est vue à nouveau.

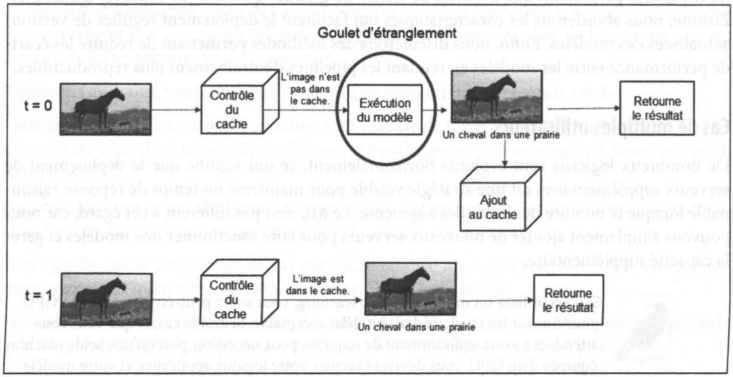


Figure 10.6 : Mise en cache pour un modèle de sous-titrage d'images.

Ce type de stratégie de mise en cache fonctionne bien pour les applications où les utilisateurs fournissent le même type de données. Elle n'est pas appropriée si chaque entrée est unique. Si une application prend des photos d'empreintes de pattes pour prédire à quel animal elles appartiennent, elle devrait rarement recevoir deux photos identiques, de sorte qu'un cache LRU n'aiderait pas.

Lorsque vous utilisez la mise en cache, vous ne devriez le faire que pour des fonctions sans effets de bord. Si une fonction `run_model` stocke également les résultats dans une base de données, par exemple, l'utilisation d'un cache LRU se traduira par le fait que des appels de fonction en double ne seront pas sauvegardés, ce qui peut ne pas être le comportement escompté.

En Python, le module `functools`[54] propose une implémentation par défaut d'un cache LRU que vous pouvez utiliser avec un simple décorateur, comme montré ici :

```
from functools import lru_cache

@lru_cache(maxsize=128)
def run_model(question_data):
```

54 Voir l'adresse `https://docs.python.org/fr/3/library/functools.html`.

```
    # Insérer une inférence de modèle lent ci-dessous
    pass
```

La mise en cache est particulièrement utile lorsque la récupération des caractéristiques, leur traitement et l'exécution de l'inférence sont plus lents que l'accès à un cache. Selon votre approche de la mise en cache (en mémoire ou sur disque, par exemple) et la complexité du modèle que vous utilisez, cette méthode aura différents degrés d'utilité.

Mise en cache par indexation. Bien que la méthode de mise en cache décrite ci-dessus ne soit pas appropriée lors de la réception d'entrées uniques, nous pouvons mettre en cache d'autres aspects du pipeline qui peuvent être précalculés. C'est plus facile si un modèle ne repose pas uniquement sur les entrées de l'utilisateur.

Supposons que nous mettons en place un système qui permet aux utilisateurs de rechercher un contenu lié soit à une requête textuelle, soit à une image qu'ils fournissent. Il est peu probable que la mise en cache des requêtes des utilisateurs améliore les performances de manière significative si l'on s'attend à ce que ces requêtes varient considérablement. Cependant, puisque nous construisons un système de recherche, nous avons accès à une liste d'articles potentiels dans notre catalogue que nous pourrions retourner. Cette liste nous est connue à l'avance, que nous soyons un détaillant en ligne ou encore une plateforme d'indexation de documents.

Cela signifie que nous pourrions calculer à l'avance des aspects de modélisation qui ne dépendent que des articles de notre catalogue. Si nous choisissons une approche de modélisation qui nous permet de faire ce calcul au préalable, nous pouvons effectuer des inférences beaucoup plus rapidement.

Pour cette raison, une approche courante lors de la construction d'un système de recherche consiste à intégrer d'abord tous les documents indexés dans un vecteur pertinent (voir la section « Vectorisation » du Chapitre 4 pour plus d'informations sur les méthodes de vectorisation). Une fois que les *embeddings* sont créés, ils peuvent être stockés dans une base de données. Ceci est illustré sur la ligne supérieure de la Figure 10.7. Lorsqu'un utilisateur soumet une requête de recherche, celle-ci est incorporée au moment de l'inférence, et une recherche est effectuée dans la base de données pour trouver les embeddings les plus similaires et renvoyer les produits correspondants. Vous pouvez voir cela illustré sur la ligne inférieure de la Figure 10.7.

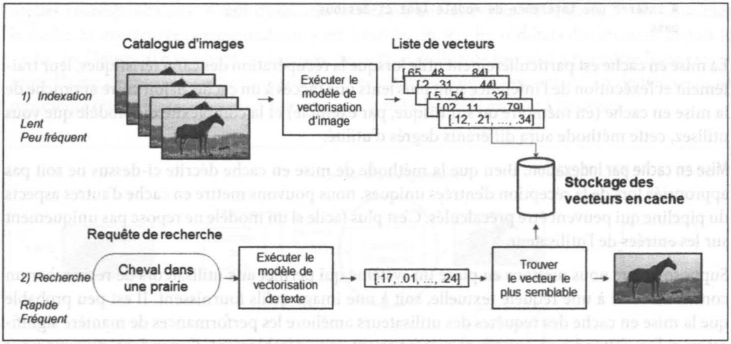


Figure 10.7 : Une requête de recherche avec des embeddings en cache.

Cette approche accélère significativement l'inférence puisque la plupart des calculs ont été effectués à l'avance. La technique d'embeddings a été utilisée avec succès dans des pipelines de production à grande échelle dans des entreprises telles que Twitter[55] et Airbnb (voir cet article de M. Haldar *et al.*, « *Applying Deep Learning To Airbnb Search* »[56].

La mise en cache peut améliorer les performances, mais elle ajoute une couche de complexité. La taille du cache devient un hyperparamètre supplémentaire à ajuster en fonction de la charge de travail de votre application. En outre, chaque fois qu'un modèle ou les données sous-jacentes sont mis à jour, le cache doit être vidé afin d'éviter qu'il ne serve des résultats obsolètes. Plus généralement, la mise à jour d'un modèle en production vers une nouvelle version nécessite souvent une attention particulière. Dans la section suivante, nous allons aborder quelques domaines qui peuvent faciliter ces mises à jour.

Gestion du cycle de vie des modèles et des données

Il peut être difficile de tenir à jour les caches et les modèles. De nombreux modèles nécessitent un réentraînement régulier pour maintenir leur niveau de performance. Nous verrons au Chapitre 11 quand il faut réentraîner vos modèles, mais j'aimerais parler brièvement de la manière de déployer des modèles mis à jour auprès des utilisateurs.

Un modèle entraîné est généralement stocké sous forme de fichier binaire contenant des informations sur son type et son architecture, ainsi que sur ses paramètres appris. La plupart des applications de production chargent un modèle entraîné en mémoire lorsqu'elles se lancent et

55 Voir le post sur le blog de Twitter à l'adresse `https://bit.ly/2WQZyZw`.
56 Voir l'adresse `https://arxiv.org/abs/1810.09591`.

appellent celui-ci pour servir des résultats. Une façon simple de remplacer un modèle par une version plus récente consiste à remplacer le fichier binaire chargé par l'application. Ceci est illustré sur la Figure 10.8, où le seul aspect du pipeline qui est impacté par un nouveau modèle est la case mise en gras.

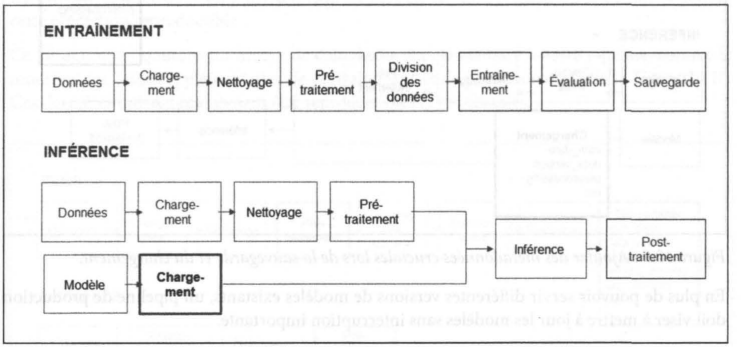


Figure 10.8 : Déployer une version mise à jour du même modèle peut sembler n'être qu'un simple changement.

Dans la pratique, cependant, ce processus est souvent nettement plus élaboré. Dans l'idéal, une application de ML fournit des résultats reproductibles, résiste aux mises à jour des modèles et est suffisamment souple pour gérer des modifications importantes de la modélisation et du traitement des données. Garantir cela implique quelques étapes supplémentaires que nous allons aborder dans ce qui suit.

Reproductibilité

Pour repérer et reproduire les erreurs, vous devez savoir quel modèle s'exécute en production. Pour ce faire, il faut conserver des archives des modèles entraînés, et des jeux de données sur lesquels ils ont été entraînés. Chaque paire modèle/jeu de données devrait se voir attribuer un identifiant unique. Cet identifiant devrait être enregistré chaque fois qu'un modèle est utilisé en production.

Sur la Figure 10.9, j'ai ajouté ces exigences aux boîtes de chargement et d'enregistrement pour représenter la complexité que cela ajoute à un pipeline de ML.

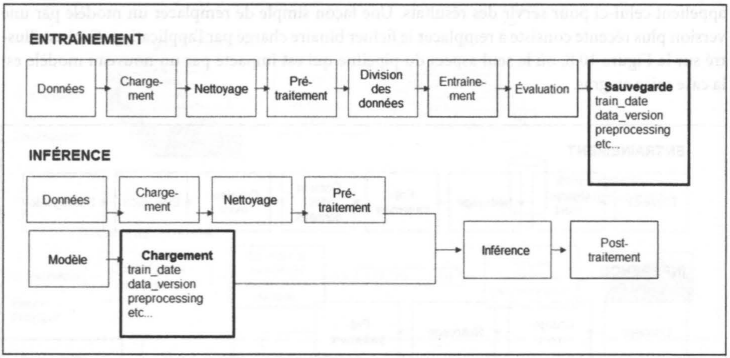


Figure 10.9 : Ajouter des métadonnées cruciales lors de la sauvegarde et du chargement.

En plus de pouvoir servir différentes versions de modèles existants, un pipeline de production doit viser à mettre à jour les modèles sans interruption importante.

Résilience

Pour permettre à une application de charger un nouveau modèle une fois qu'elle est mise à jour, il faut mettre en place un processus permettant de charger un modèle plus récent, idéalement sans perturber le service fourni aux utilisateurs. Cela peut consister à lancer un nouveau serveur prenant en charge le modèle mis à jour et à effectuer vers lui une transition en douceur du trafic, mais cela devient rapidement plus complexe pour des systèmes plus importants. Si un nouveau modèle fonctionne mal, nous aimerions pouvoir revenir au modèle précédent. Effectuer ces deux tâches correctement est un défi et serait traditionnellement classé dans le domaine du DevOps[57]. Bien que nous ne couvrions pas ce domaine en profondeur, le Chapitre 11 propose une introduction aux questions de monitoring.

Les changements en production peuvent être plus complexes que la mise à jour d'un modèle. Ils peuvent inclure des modifications importantes dans le traitement des données, qui doivent également pouvoir être déployées.

Flexibilité du pipeline

Nous avons vu précédemment que la meilleure façon d'améliorer un modèle est souvent d'itérer sur le traitement des données et la génération de caractéristiques. Cela signifie que les nouvelles

57 Voir par exemple l'adresse `https://fr.wikipedia.org/wiki/Devops`.

versions d'un modèle nécessiteront souvent des étapes de prétraitement supplémentaires ou des caractéristiques différentes.

Ce type de changement ne se reflète pas seulement dans le modèle binaire et serait souvent lié à une nouvelle version de votre application. C'est pourquoi la version de l'application doit également être stockée dans un journal lorsqu'un modèle fait une prédiction, et ce afin de rendre cette prédiction reproductible.

Ce faisant, nous ajoutons un niveau de complexité supplémentaire à notre pipeline, comme le montrent les cases supplémentaires de prétraitement et de post-traitement de la Figure 10.10. Ces dernières doivent maintenant être reproductibles et modifiables.

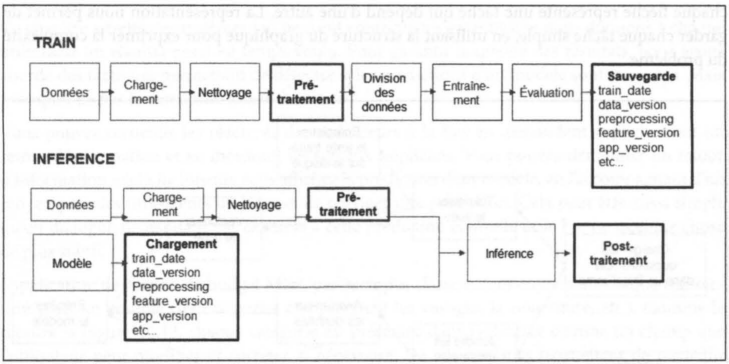


Figure 10.10 : Ajout d'une version du modèle et de l'application.

Le déploiement et la mise à jour des modèles est un défi. Lors de la construction d'une infrastructure de service, l'aspect le plus important est de pouvoir reproduire les résultats d'un modèle exécuté en production. Cela signifie qu'il faut lier chaque appel d'inférence au modèle exécuté, au jeu de données sur lequel le modèle a été entraîné et à la version du pipeline de données qui a servi ce modèle.

Traitement des données et DAG

Pour produire des résultats reproductibles comme décrit précédemment, un pipeline d'entraînement devrait également être reproductible et déterministe. Pour une combinaison donnée de jeu de données, d'étapes de prétraitement et de modèle, un pipeline d'entraînement devrait donc produire le même modèle entraîné, et ce à chaque cycle de cet entraînement.

De nombreuses étapes de transformation successives sont nécessaires pour construire un modèle, de sorte que les pipelines se « rompent » souvent à différents endroits. Il faut donc s'assurer que chaque partie a été exécutée avec succès, et que toutes les parties ont été exécutées dans le bon ordre.

Une façon de simplifier ce défi est de représenter notre processus de passage des données brutes au modèle entraîné sous la forme d'un *graphe orienté acyclique* (ou DAG), chaque nœud représentant une étape de traitement, et chaque étape représentant une dépendance entre deux nœuds. Cette idée est au cœur de la programmation des flux de données, un paradigme de programmation sur lequel repose la populaire bibliothèque de ML TensorFlow.

Les DAG peuvent être un moyen naturel de visualiser le prétraitement. Sur la Figure 10.11, chaque flèche représente une tâche qui dépend d'une autre. La représentation nous permet de garder chaque tâche simple, en utilisant la structure du graphique pour exprimer la complexité du problème.

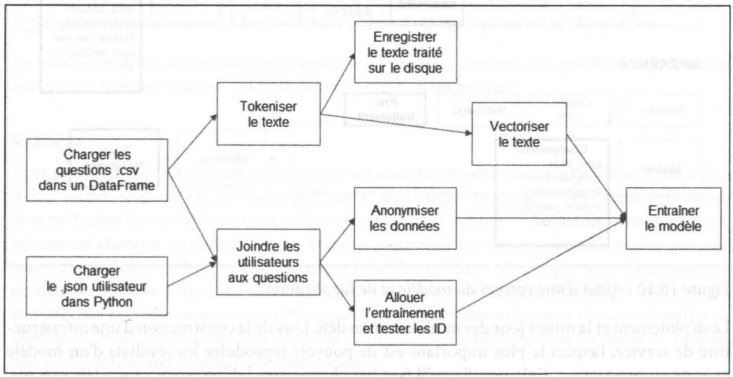


Figure 10.11 : Un exemple de DAG pour notre application.

Une fois que nous avons un DAG, nous pouvons alors garantir que nous suivons le même schéma d'opérations pour chaque modèle que nous produisons. Il existe de multiples solutions pour définir les DAG pour le ML, y compris des projets open source actifs comme Apache Airflow[58] ou Luigi[59] de Spotify. Les deux packages vous permettent de définir des DAG et fournissent un ensemble de tableaux de bord pour vous permettre de suivre l'évolution de vos DAG et de tous les journaux associés.

58 Voir l'adresse `https://airflow.apache.org/`.
59 Voir l'adresse `https://github.com/spotify/luigi`.

Lors de la construction initiale d'un pipeline de ML, l'utilisation d'un DAG peut être inutilement lourde, mais une fois qu'un modèle devient une partie essentielle d'un système en production, les exigences de reproductibilité rendent les DAG très convaincants. Une fois que les modèles sont régulièrement réentraînés et déployés, tout outil qui aide à systématiser, à déboguer et à versionner un pipeline offre un gain de temps crucial.

Pour conclure ce chapitre, j'aborderai un moyen supplémentaire et direct de garantir qu'un modèle fonctionne bien pour les utilisateurs.

Susciter des retours

Ce chapitre a traité des systèmes qui peuvent contribuer à garantir que nous donnons à chaque utilisateur un résultat exact en temps voulu. Pour garantir la qualité des résultats, nous avons abordé des tactiques permettant de détecter si les prédictions d'un modèle sont inexactes. Mais pourquoi ne pas le demander aux utilisateurs ?

Vous pouvez recueillir les réactions des utilisateurs à la fois en demandant explicitement un retour d'information et en mesurant les signaux implicites. Vous pouvez demander un retour d'information explicite lorsque vous affichez la prédiction d'un modèle, en l'accompagnant d'un moyen pour les utilisateurs de juger et de corriger une prédiction. Cela peut être aussi simple qu'un dialogue demandant par exemple « cette prédiction était-elle utile ? », ou quelque chose de plus subtil.

L'application de gestion de budget Mint, par exemple, classe automatiquement chaque transaction dans un compte (les catégories comprennent les voyages, la nourriture, *etc.*). Comme le montre la Figure 10.12, chaque catégorie est présentée dans l'interface comme un champ que l'utilisateur peut modifier et corriger si nécessaire. De tels systèmes permettent de recueillir des informations précieuses pour améliorer continuellement les modèles d'une manière moins intrusive qu'une enquête de satisfaction, par exemple.

Les utilisateurs ne peuvent pas fournir un retour d'information pour chaque prédiction faite par un modèle, c'est pourquoi la collecte de retours implicites est un moyen important de juger des performances du ML. La collecte de ce *feedback* consiste à examiner les actions effectuées par les utilisateurs afin d'en déduire si un modèle a fourni des résultats utiles.

Les signaux implicites sont utiles mais plus difficiles à interpréter. Il ne faut pas espérer trouver un signal implicite qui soit toujours en corrélation avec la qualité du modèle, mais seulement un qui le fasse de manière agrégée. Par exemple, dans un système de recommandation, si un utilisateur clique sur un des éléments proposés, vous pouvez raisonnablement supposer que cette recommandation était valable. Cela ne sera pas vrai dans tous les cas (les gens cliquent parfois sur les mauvaises choses !), mais tant que c'est vrai le plus souvent, il s'agit d'un signal implicite raisonnable.

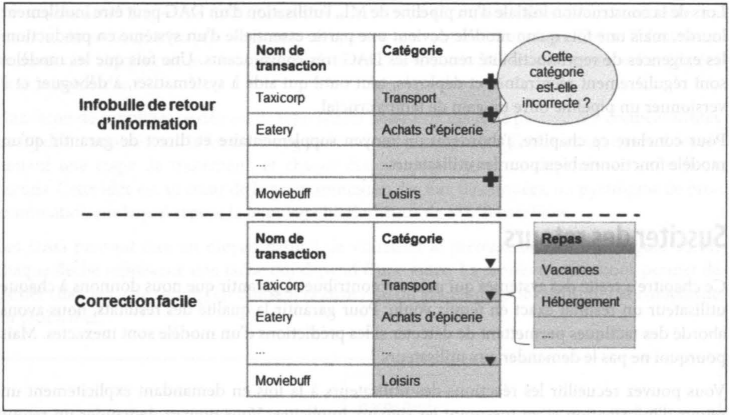


Figure 10.12 : Mint permet de laisser les utilisateurs corriger directement les erreurs.

En recueillant ces informations, comme le montre la Figure 10.13, vous pouvez ensuite estimer la fréquence à laquelle les utilisateurs ont trouvé les résultats utiles. La collecte de ces signaux implicites est utile, mais elle s'accompagne du risque supplémentaire de collecter et de stocker ces données, et d'introduire potentiellement des boucles de rétroaction négative, comme nous l'avons vu au Chapitre 8.

La mise en place de mécanismes de retour d'information implicites dans votre produit peut être un moyen précieux de recueillir des données supplémentaires. De nombreuses actions peuvent être considérées comme un mélange de retour d'information implicite et explicite.

Supposons que nous avons ajouté un bouton « Poser la question sur Stack Overflow » aux recommandations de notre Éditeur ML. En analysant quelles prédictions ont conduit les utilisateurs à cliquer sur ce bouton, nous pourrions mesurer la proportion de recommandations qui étaient suffisamment bonnes pour être affichées sous forme de questions. En ajoutant ce bouton, nous ne demandons pas directement aux utilisateurs si la suggestion est bonne, mais nous leur permettons d'y donner suite, ce qui nous donne une « étiquette faible » (voir la section « Types de données » dans le Chapitre 1 pour un rappel sur les données faiblement étiquetées) pour la qualité de la question.

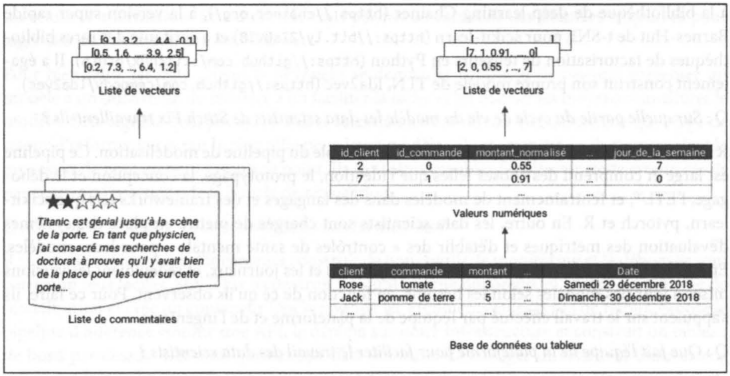


Figure 10.13 : Les actions des utilisateurs comme source de retour d'information.

En plus d'être une bonne source de données d'entraînement, les commentaires implicites et explicites des utilisateurs peuvent être le premier moyen de constater une dégradation des performances d'un produit de ML. Bien que, dans l'idéal, les erreurs devraient être détectées avant d'être affichées aux utilisateurs, le suivi de ces commentaires permet de détecter et de corriger les bogues plus rapidement. Nous reviendrons plus en détail sur ce point au Chapitre 11.

Les stratégies de déploiement et de mise à jour des modèles varient énormément en fonction de la taille d'une équipe et de son expérience en matière de ML. Certaines des solutions présentées dans ce chapitre sont excessivement complexes pour un prototype tel que notre Éditeur ML. D'autre part, certaines équipes qui ont investi une quantité importante de ressources dans le ML ont construit des systèmes complexes qui leur permettent de simplifier leur processus de déploiement et de garantir un haut niveau de qualité aux utilisateurs. Pour terminer ce chapitre, je vous propose une interview de Chris Moody, qui dirige l'équipe AI Instruments de Stitch Fix et va nous faire découvrir leur philosophie en matière de déploiement de modèles ML.

Chris Moody : Permettre aux spécialistes des données de déployer des modèles

Chris Moody, qui a fait des études de physique à Caltech et à l'UCSC, dirige aujourd'hui l'équipe AI Instruments de Stitch Fix. Il s'intéresse vivement au traitement du langage naturel ainsi qu'au deep learning, aux méthodes variationnelles et aux processus gaussiens. Il a contribué

à la bibliothèque de deep learning Chainer (`https://chainer.org/`), à la version super rapide Barnes-Hut de t-SNE pour scikit-learn (`https://bit.ly/2ZsBVIB`) et a écrit une des rares bibliothèques de factorisation de tenseurs en Python (`https://github.com/stitchfix/ntflib`). Il a également construit son propre modèle de TLN, lda2vec (`https://github.com/cemoody/lda2vec`).

Q : Sur quelle partie du cycle de vie du modèle les data scientists de Stitch Fix travaillent-ils ?

R : Chez Stitch Fix, les data scientists traitent l'ensemble du pipeline de modélisation. Ce pipeline est large et comprend des choses telles que l'idéation, le prototypage, la conception et le débogage, l'ETL[60], et l'entraînement de modèles dans des langages et des frameworks tels que scikit-learn, pytorch et R. En outre, les data scientists sont chargés de mettre en place des systèmes d'évaluation des métriques et d'établir des « contrôles de santé mentale » pour leurs modèles. Enfin, ils effectuent le test A/B, surveillent les erreurs et les journaux, et redéploient les versions mises à jour des modèles selon les besoins, en fonction de ce qu'ils observent. Pour ce faire, ils s'appuient sur le travail effectué par l'équipe de la plateforme et de l'ingénierie.

Q : Que fait l'équipe de la plateforme pour faciliter le travail des data scientists ?

R : L'objectif des ingénieurs de l'équipe de la plateforme est de trouver les bonnes abstractions pour la modélisation. Cela signifie qu'ils doivent comprendre comment un spécialiste des données travaille. Les ingénieurs ne construisent pas des pipelines de données individuels pour les data scientists travaillant sur un projet donné. Ils réalisent des solutions qui permettent à ceux-ci de le faire eux-mêmes. Plus généralement, ils construisent des outils qui permettent aux data scientists de s'approprier l'ensemble du flux de travail. Cela permet aux ingénieurs de consacrer plus de temps à l'amélioration de la plateforme et moins de temps à l'élaboration de solutions ponctuelles.

Q : Comment jugez-vous les performances des modèles une fois qu'ils sont déployés ?

R : Une grande partie de la force de Stitch Fix est de faire travailler ensemble les humains et les algorithmes. Par exemple, Stitch Fix passe beaucoup de temps à réfléchir à la bonne façon de présenter l'information à ses stylistes. Fondamentalement, si vous avez une API qui expose votre modèle d'un côté, et un utilisateur tel qu'un styliste ou un acheteur de marchandises de l'autre, comment devriez-vous concevoir les interactions entre eux ?

À première vue, vous pourriez être tenté de construire un *frontend* pour présenter simplement les résultats de votre algorithme aux utilisateurs. Malheureusement, cela peut leur donner l'impression qu'ils n'ont aucun contrôle sur l'algorithme et sur le système global, et peut entraîner une frustration lorsque celui-ci ne fonctionne pas bien. Vous devriez plutôt considérer cette interaction comme une boucle de rétroaction, permettant aux utilisateurs de corriger et d'ajuster les résultats. Agir de cette manière permet aux utilisateurs d'entraîner les algorithmes et d'avoir un impact beaucoup plus important sur l'ensemble du processus, en étant en mesure de fournir

60 Voir par exemple l'adresse `https://fr.wikipedia.org/wiki/Extract-transform-load`.

un retour d'information. En outre, cela vous permet de recueillir des données étiquetées pour juger des performances de vos modèles.

Pour bien faire, les data scientists devraient se demander comment ils peuvent présenter un modèle à un utilisateur, de manière à lui faciliter la tâche et lui donner les moyens d'améliorer le modèle. Cela signifie que, puisque les data scientists savent mieux que quiconque quel type de retour d'information serait le plus utile pour leurs modèles, il est essentiel qu'ils s'approprient le processus de bout en bout jusqu'à ce point. Ils sont ainsi capables de détecter toute erreur, car ils peuvent voir l'ensemble de la boucle de rétroaction.

Q : Comment contrôlez-vous et déboguez-vous les modèles ?

R : Lorsque votre équipe d'ingénieurs construit de bons outils, le monitoring et le débogage sont beaucoup plus faciles. Stitch Fix a construit un outil interne qui prend en charge un pipeline de modélisation et crée un conteneur Docker, valide les arguments et les types de retour, expose le pipeline d'inférence comme une API, le déploie sur notre infrastructure et construit un tableau de bord par-dessus. Cet outil permet aux data scientists de corriger directement toute erreur qui se produit pendant ou après le déploiement. Comme ces spécialistes sont désormais chargés de dépanner les modèles, nous avons également constaté que cette configuration incite à faire appel à des modèles simples et robustes, qui ont tendance à se « briser » plus rarement. Le fait de posséder l'ensemble du pipeline conduit les individus à optimiser l'impact et la fiabilité des modèles, plutôt qu'à s'occuper de leur complexité.

Q : Comment déployer les nouvelles versions des modèles ?

R : En plus, les data scientists mènent des expériences en utilisant un service de test A/B construit sur mesure qui leur permet de définir des paramètres granulaires. Ils analysent ensuite les résultats des tests, et, s'ils sont jugés concluants par l'équipe, ils déploient eux-mêmes la nouvelle version.

En ce qui concerne le déploiement, nous utilisons un système similaire à celui du développement de type canari, en commençant par déployer la nouvelle version sur une instance puis par mettre progressivement à jour les instances tout en surveillant les performances. Les data scientists ont accès à un tableau de bord qui montre le nombre d'instances sous chaque version ainsi que des mesures en continu des performances au fur et à mesure du déploiement.

Conclusion

Dans ce chapitre, nous avons abordé les moyens de rendre nos réponses plus résistantes en détectant de manière proactive les échecs potentiels de notre modèle et en trouvant des moyens de les atténuer. Cela a inclus à la fois des stratégies de validation déterministes et l'utilisation de modèles de filtrage. Nous avons également abordé quelques-uns des défis qui se posent lorsqu'on veut maintenir un modèle de production à jour. Ensuite, nous avons discuté de certaines des

façons dont nous pouvons estimer la performance d'un modèle. Enfin, nous avons jeté un coup d'œil sur un exemple pratique d'une entreprise qui déploie fréquemment et à grande échelle le ML, ainsi que les processus qu'elle a mis en place pour ce faire.

Dans le Chapitre 11, nous aborderons d'autres méthodes permettant de surveiller les performances des modèles et de tirer parti de diverses métriques pour diagnostiquer l'état de santé d'une application basée sur le ML.

CHAPITRE 11

Monitoring et mise à jour des modèles

Une fois qu'un modèle est déployé, ses performances doivent être contrôlées comme pour tout autre logiciel. Comme nous l'avons vu dans la section « Tester votre code ML » du Chapitre 6, les meilleures pratiques logicielles habituelles s'appliquent tout aussi bien au ML. Et tout comme aussi nous l'avons vu dans cette même section, il y a d'autres éléments à prendre en compte lorsqu'on traite des modèles de ML.

Dans ce chapitre, nous décrirons les principaux aspects à garder à l'esprit lors du suivi des modèles de ML. Plus précisément, nous répondrons à trois questions :

1. Pourquoi devrions-nous effectuer le monitoring de nos modèles ?
2. Comment assurer le monitoring de nos modèles ?
3. Quelles sont les actions que nous devrions mener dans le cadre de notre stratégie de monitoring ?

Commençons par expliquer comment les modèles de monitoring peuvent aider à décider quand déployer une nouvelle version, ou à traiter des problèmes en production.

Le monitoring sauve des vies

L'objectif du monitoring est de surveiller l'état de santé d'un système. Pour les modèles, cela signifie surveiller leurs performances et la qualité de leurs prédictions.

Si un changement dans les habitudes des utilisateurs fait soudainement qu'un modèle produit des résultats inférieurs à la moyenne, un bon système de surveillance vous permettra de le remarquer et de réagir le plus rapidement possible. Couvrons quelques questions clés que le monitoring peut nous aider à saisir.

Monitoring pour le taux de rafraîchissement

Nous avons vu dans la section « Fraîcheur et changements dans la distribution » du Chapitre 2 que la plupart des modèles doivent être régulièrement mis à jour pour maintenir un certain niveau de performance. Le monitoring peut être utilisé pour détecter quand un modèle n'est plus « frais » et doit être réentraîné.

Par exemple, supposons que nous utilisons les retours implicites que nous recevons de nos utilisateurs (quand ils cliquent sur les recommandations, par exemple) pour estimer l'exactitude d'un modèle. Si nous surveillons en permanence cette exactitude, nous pouvons entraîner un nouveau modèle dès que l'exactitude descend en dessous d'un seuil défini. La Figure 11.1 présente une chronologie de ce processus, avec des événements de réentraînement se produisant lorsque l'exactitude tombe en dessous d'un seuil.

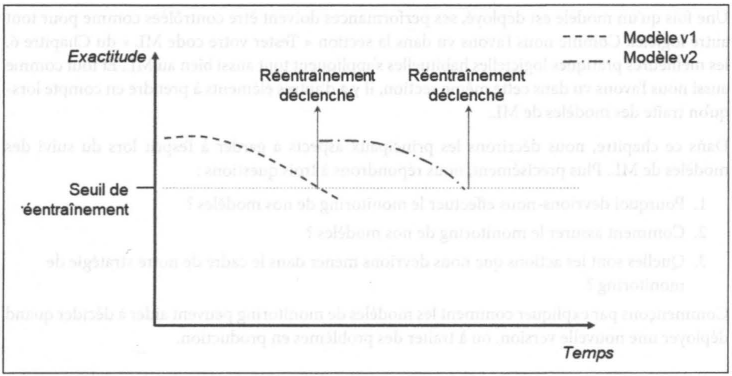


Figure 11.1 : Monitoring pour déclencher le redéploiement.

Avant de redéployer un modèle mis à jour, vous devriez vérifier que le nouveau modèle est meilleur. Nous verrons comment procéder un peu plus loin à la section « CI/CD pour le ML ». Tout d'abord, abordons d'autres aspects à surveiller, comme les abus potentiels.

Monitoring pour détecter les abus

Dans certains cas, par exemple lors de la mise en place de systèmes de prévention des abus ou de détection des fraudes, une fraction des utilisateurs s'emploie activement à faire échouer les

modèles. Dans ces cas, le monitoring devient un moyen essentiel pour détecter les attaques et estimer leur taux de réussite.

Un système de monitoring peut utiliser la détection d'anomalies pour détecter les attaques. Lors du suivi de chaque tentative de connexion au portail en ligne d'une banque, par exemple, un système de surveillance pourrait déclencher une alerte si le nombre de tentatives de connexion était soudainement multiplié par dix, ce qui pourrait être le signe d'une attaque.

Cette surveillance pourrait déclencher une alerte en cas de dépassement d'une valeur seuil, comme le montre la Figure 11.2, ou inclure des métriques plus nuancées telles que le taux d'augmentation des tentatives de connexion. En fonction de la complexité des attaques, il peut être utile de construire un modèle permettant de détecter ces anomalies avec plus de nuances qu'un simple seuil ne le pourrait.

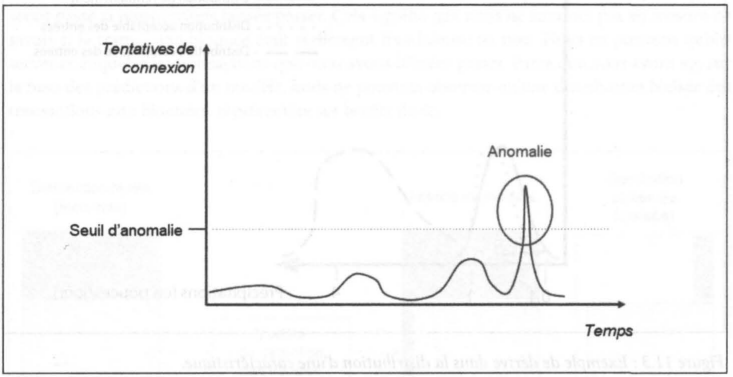


Figure 11.2 : Une anomalie évidente sur un tableau de bord de monitoring. Vous pourriez construire un modèle de ML supplémentaire pour la détecter automatiquement.

Outre la surveillance de la fraîcheur et la détection des anomalies, quels autres paramètres devrions-nous surveiller ?

Choisir ce que vous voulez surveiller

Les applications logicielles contrôlent généralement des paramètres tels que le temps moyen de traitement d'une requête, la proportion de requêtes dont le traitement a échoué, ou encore la quantité de ressources disponibles. Ces données sont utiles pour assurer un suivi dans tout ser-

vice de production et permettent de prendre des mesures correctives proactives avant que trop d'utilisateurs ne soient touchés.

Ensuite, nous aborderons d'autres métriques à surveiller pour détecter quand les performances d'un modèle commencent à décliner.

Métriques de performances

Un modèle peut devenir obsolète si la distribution des données commence à changer. Vous pouvez le voir sur la Figure 11.3.

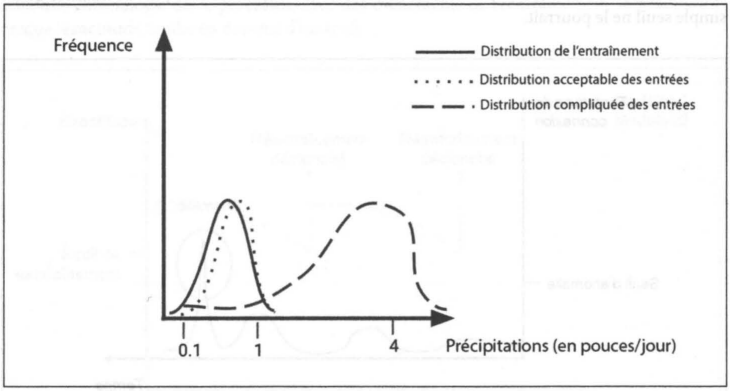


Figure 11.3 : Exemple de dérive dans la distribution d'une caractéristique.

Dans le cas de changements dans la distribution, celle-ci peut se produire aussi bien en entrée qu'en sortie. Prenons l'exemple d'un modèle qui tente de deviner quel film un utilisateur va regarder ensuite. Étant donné l'historique de l'utilisateur en tant qu'entrée, la prédiction du modèle devrait changer en fonction des nouvelles entrées dans un catalogue de films disponibles.

- Il est plus facile de suivre les changements dans la distribution des entrées (également appelée dérive des caractéristiques) que de suivre la distribution des sorties, car il peut être difficile d'accéder à la valeur idéale des sorties pour satisfaire les utilisateurs.
- Le suivi de la distribution des entrées peut être aussi simple que le suivi de statistiques sommaires telles que la moyenne et la variance des caractéristiques clés et le déclenchement d'une alerte si ces statistiques s'écartent des valeurs des données d'entraînement au-delà d'un seuil fixé.

- Le suivi des changements de distribution peut être plus difficile. Une première approche consiste à surveiller la distribution des résultats des modèles. Comme pour les entrées, un changement significatif dans la distribution des sorties peut être un signe que la performance du modèle s'est dégradée. La distribution des résultats que les utilisateurs auraient souhaité voir, cependant, peut être plus difficile à estimer.

L'une des raisons pour lesquelles l'estimation de la vérité de terrain peut être difficile est que les actions d'un modèle risquent souvent de nous empêcher de l'observer. Pour voir pourquoi cela peut être le cas, examinez l'illustration d'un modèle de détection de fraude par carte de crédit proposé sur la Figure 11.4. La distribution des données que le modèle va recevoir se trouve sur le côté gauche. Comme le modèle fait des prédictions sur les données, le code d'application agit sur ces prédictions en bloquant toute transaction prédite comme étant frauduleuse.

Une fois qu'une transaction est bloquée, nous sommes donc incapables d'observer ce qui se serait passé si nous l'avions laissée passer. Cela signifie que nous ne sommes pas en mesure de savoir si la transaction bloquée était réellement frauduleuse ou non. Nous ne pouvons qu'observer et étiqueter les transactions que nous avons laissées passer. Parce que nous avons agi sur la base des prédictions d'un modèle, nous ne pouvons observer qu'une distribution biaisée des transactions non bloquées, représentées sur le côté droit.

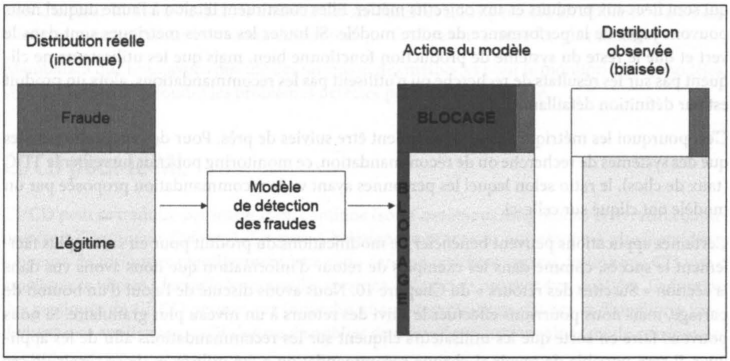


Figure 11.4 : Agir sur la base des prédictions d'un modèle peut biaiser la distribution observée des données.

Le fait de n'avoir accès qu'à un échantillon biaisé de la distribution réelle rend impossible l'évaluation correcte des performances d'un modèle. C'est l'objet de l'*évaluation contrefactuelle*, qui vise à évaluer ce qui se serait passé si nous n'avions pas mis en œuvre un modèle. Pour effectuer

en pratique une telle évaluation, vous pouvez vous abstenir d'exécuter un modèle sur un petit sous-ensemble d'échantillons (voir l'article de Lihong Li *et al.*, « *Counterfactual Estimation and Optimization of Click Metrics for Search Engines* »[61]). Ne pas agir sur un sous-ensemble aléatoire d'échantillons nous permettra alors d'observer une distribution impartiale des transactions frauduleuses. En comparant les prédictions du modèle aux résultats réels pour ces données aléatoires, nous pouvons commencer à estimer la précision et le rappel d'un modèle.

Cette approche fournit un moyen d'évaluer les modèles, mais elle a pour conséquence de laisser passer une partie des transactions frauduleuses. Dans de nombreux cas, ce compromis peut être favorable puisqu'il permet d'évaluer et de comparer les modèles. Dans certains cas, comme dans les domaines médicaux où il n'est pas acceptable de faire une prédiction aléatoire, cette approche ne doit pas être utilisée.

Dans la section « CI/CD pour le ML », un peu plus loin dans ce chapitre, nous allons aborder d'autres stratégies pour comparer les modèles et décider lesquels déployer, mais d'abord, nous allons aborder les autres types de métriques clés à pister.

Métriques d'entreprise

Comme nous l'avons vu tout au long de ce livre, les métriques les plus importantes sont celles qui sont liées aux produits et aux objectifs métier. Elles constituent l'étalon à l'aune duquel nous pouvons juger de la performance de notre modèle. Si toutes les autres métriques sont dans le vert et que le reste du système de production fonctionne bien, mais que les utilisateurs ne cliquent pas sur les résultats de recherche ou n'utilisent pas les recommandations, alors un produit est par définition défaillant.

C'est pourquoi les métriques produit devraient être suivies de près. Pour des applications telles que des systèmes de recherche ou de recommandation, ce monitoring pourrait surveiller le TDC (taux de clics), le ratio selon lequel les personnes ayant vu la recommandation proposée par un modèle ont cliqué sur celle-ci.

Certaines applications peuvent bénéficier de modifications du produit pour en suivre plus facilement le succès, comme dans les exemples de retour d'information que nous avons vus dans la section « Susciter des retours » du Chapitre 10. Nous avons discuté de l'ajout d'un bouton de partage, mais nous pourrions effectuer le suivi des retours à un niveau plus granulaire. Si nous pouvons faire en sorte que les utilisateurs cliquent sur les recommandations afin de les appliquer, il sera possible de savoir si chaque recommandation a été utilisée et de se servir de ces données pour entraîner une nouvelle version du modèle. La Figure 11.5 montre une comparaison illustrée entre l'approche agrégée sur le côté gauche et l'approche granulaire sur le côté droit.

61 Voir l'adresse `https://arxiv.org/abs/1403.1891`.

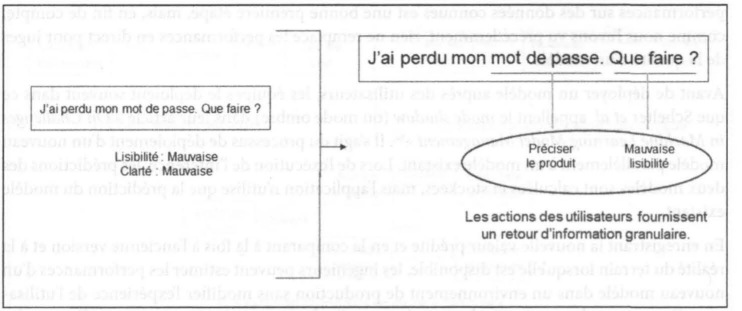


Figure 11.5 : Proposer des suggestions au niveau des mots nous donne plus d'opportunités de collecter les réactions des utilisateurs.

Comme je ne m'attends pas à ce que le prototype de l'Éditeur ML soit utilisé assez fréquemment pour que la méthode décrite fournisse un jeu de données suffisamment important, nous nous abstiendrons de l'implémenter ici. Si nous construisions un produit dont nous aurions l'intention d'assurer la maintenance, la collecte de ces données nous permettrait d'obtenir un retour d'information précis sur les recommandations que l'utilisateur trouve les plus utiles.

Maintenant que nous avons discuté des raisons et des méthodes pour le monitoring des modèles, voyons comment résoudre les problèmes détectés par cette surveillance.

CI/CD pour le ML

CI/CD peut se traduire par intégration continue (soit *Continuous Integration*) et livraison continue (soit *Continuous Delivery*). En gros, CI désigne le processus qui permet à plusieurs développeurs de fusionner régulièrement leur code dans un *codebase* central, tandis que CD vise à améliorer la vitesse à laquelle les nouvelles versions de logiciels peuvent être publiées. L'adoption des pratiques de CI/CD permet aux individus et aux organisations d'itérer et d'améliorer rapidement une application, qu'il s'agisse de publier de nouvelles fonctionnalités ou de corriger des bogues existants.

CI/CD pour le ML vise donc à faciliter le déploiement de nouveaux modèles ou la mise à jour des modèles existants. Il est facile de diffuser rapidement des mises à jour. Le défi consiste à garantir leur qualité.

En ce qui concerne le ML, nous avons vu qu'il ne suffit pas d'avoir une suite de tests pour garantir qu'un nouveau modèle améliore un précédent. Entraîner un nouveau modèle et tester ses

performances sur des données connues est une bonne première étape, mais, en fin de compte, comme nous l'avons vu précédemment, rien ne remplace les performances en direct pour juger de la qualité d'un modèle.

Avant de déployer un modèle auprès des utilisateurs, les équipes le déploient souvent dans ce que Schelter *et al.* appellent le *mode shadow* (ou mode ombre) dans leur article « *On Challenges in Machine Learning Model Management* »[62]. Il s'agit du processus de déploiement d'un nouveau modèle parallèlement à un modèle existant. Lors de l'exécution de l'inférence, les prédictions des deux modèles sont calculées et stockées, mais l'application n'utilise que la prédiction du modèle existant.

En enregistrant la nouvelle valeur prédite et en la comparant à la fois à l'ancienne version et à la réalité du terrain lorsqu'elle est disponible, les ingénieurs peuvent estimer les performances d'un nouveau modèle dans un environnement de production sans modifier l'expérience de l'utilisateur. Cette approche permet également de tester l'infrastructure nécessaire pour effectuer des inférences pour un nouveau modèle susceptible d'être plus complexe que le modèle existant. La seule chose que le mode d'inférence n'offre pas est la possibilité d'observer la réponse de l'utilisateur au nouveau modèle. La seule façon d'y parvenir est de le déployer réellement.

Une fois qu'un modèle a été testé, il est candidat au déploiement. Cependant, le déploiement d'un nouveau modèle comporte le risque d'exposer les utilisateurs à une dégradation des performances. La réduction de ce risque nécessite une certaine attention et est au centre du champ d'expérimentation.

La Figure 11.6 montre une visualisation de chacune des trois approches que nous avons couvertes ici, de la plus sûre, qui consiste à évaluer un mode sur un jeu de test, à la plus informative et la plus dangereuse, qui consiste à déployer un modèle en direct en production. Notez que si le mode shadow nécessite un effort d'ingénierie pour pouvoir exécuter deux modèles pour chaque étape d'inférence, il permet d'évaluer un modèle de manière presque aussi sûre qu'en utilisant un jeu de test et fournit pratiquement autant d'informations que si on l'exécutait en production.

Comme le déploiement de modèles en production peut être un processus risqué, les équipes d'ingénieurs ont développé des méthodes pour déployer les changements de manière progressive, en commençant par montrer les nouveaux résultats à un sous-ensemble d'utilisateurs seulement. Nous aborderons ce point dans la prochaine partie.

62 Voir l'adresse `https://bit.ly/3guTMob`.

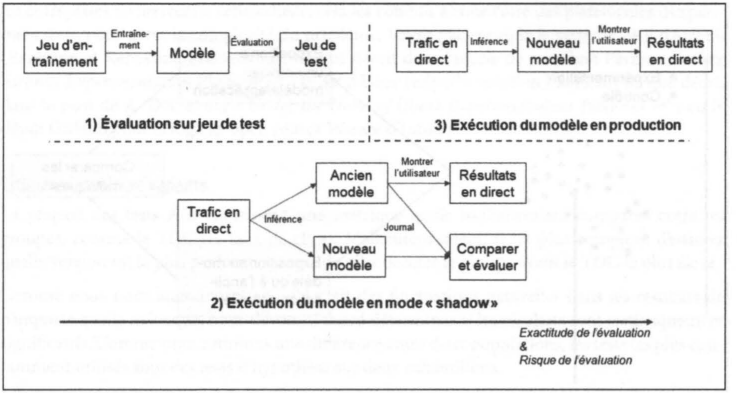


Figure 11.6 : Différentes modes d'évaluation d'un modèle, du plus sûr et le moins exact au plus risqué et le plus exact.

Test A/B et expérimentation

Dans le domaine du ML, l'objectif de l'expérimentation est de maximiser les chances d'utiliser le meilleur modèle, tout en minimisant le coût de l'essai de modèles sous-optimaux. Il existe de nombreuses approches d'expérimentation, la plus populaire étant le test A/B.

Le principe derrière le test A/B est simple : exposer un échantillon d'utilisateurs à un nouveau modèle, et le reste à un autre. Cela se fait généralement en proposant le modèle actuel à un groupe « témoin » plus important, et une nouvelle version que nous voulons tester à un groupe de « traitement » plus restreint. Une fois que nous avons effectué une expérimentation pendant une durée suffisante, nous comparons les résultats pour les deux groupes et nous choisissons le meilleur modèle.

Sur la Figure 11.7, vous pouvez voir comment échantillonner aléatoirement les utilisateurs à partir d'une population totale pour les affecter à un jeu de tests. Au moment de l'inférence, le modèle utilisé pour un certain utilisateur est déterminé par le groupe qui lui a été attribué.

L'idée qui sous-tend les tests A/B est simple, mais les questions qui en découlent, telles que choisir le groupe témoin et le groupe de traitement, décider d'une durée suffisante pour l'expérimentation, et évaluer le modèle le plus performant, sont autant de défis à relever.

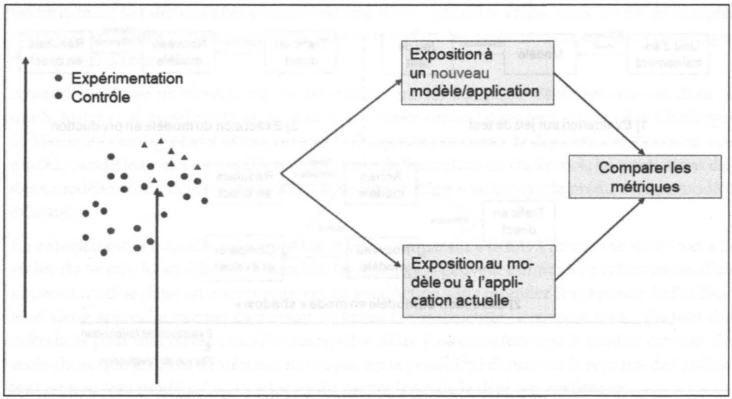


Figure 11.7 : Un exemple de test A/B.

En outre, un test A/B nécessite la construction d'une infrastructure supplémentaire pour permettre de servir différents modèles à différents utilisateurs. Examinons plus en détail chacun de ces défis.

Choix des groupes et de la durée

Décider quels utilisateurs devraient être servis avec quel modèle est assorti de quelques exigences. Les utilisateurs des deux groupes doivent être aussi semblables que possible, de sorte que toute différence observée dans les résultats puisse être attribuée à notre modèle et non au public choisi. Si tous les utilisateurs du groupe A sont des utilisateurs réguliers alors que le groupe B ne contient que des utilisateurs occasionnels, les résultats de l'expérimentation ne seront pas concluants.

En outre, le groupe de traitement B doit être suffisamment important pour fournir une conclusion statistiquement significative, mais aussi réduit que possible pour limiter l'exposition à un modèle potentiellement pire. La durée du test présente un compromis similaire : trop courte et nous risquons de ne pas avoir assez d'informations, trop longue et nous risquons de perdre des utilisateurs.

Ces deux contraintes sont déjà suffisamment difficiles, mais considérez un instant le cas de grandes entreprises qui disposent de centaines de data scientists qui effectuent des dizaines de tests A/B en parallèle. Plusieurs d'entre eux peuvent tester le même aspect du pipeline en même temps, ce qui rend plus difficile de déterminer avec exactitude l'effet d'un test individuel. Lorsque

les entreprises en arrivent à cette échelle, cela les conduit à construire des plateformes d'expérimentation pour gérer la complexité du problème. Voyez par exemple le système ERF d'Airbnb (Experiment Reporting Framework), tel que décrit dans l'article de Jonathan Parks, « *Scaling Airbnb's Experimentation Platform* »[63] ; le XP d'Uber (eXperimentation Platform) tel que décrit dans le post de A. Deb *et al.*, « *Under the Hood of Uber's Experimentation Platform* »[64] ; ou le dépôt GitHub pour le logiciel open source Wasabi d'Intuit[65].

Estimer la meilleure variante

La plupart des tests A/B choisissent une métrique qu'ils souhaiteraient comparer entre les groupes, comme le TDC (le taux de clics). Malheureusement, il est plus complexe d'estimer quelle version est la plus performante que de sélectionner le groupe ayant le TDC le plus élevé.

Comme nous nous attendons à ce qu'il y ait des fluctuations naturelles dans les résultats de n'importe quelle métrique, nous devons d'abord déterminer si les résultats sont statistiquement significatifs. Comme nous estimons une différence entre deux populations, les tests les plus couramment utilisés sont des tests d'hypothèse sur deux échantillons.

Pour qu'une expérimentation soit concluante, elle doit être menée sur un nombre suffisant de données. La quantité exacte dépend de la valeur de la variable que nous mesurons et de l'ampleur du changement que nous visons à détecter. Pour un exemple pratique, voyez par exemple le calculateur de la taille de l'échantillon d'Evan Miller[66].

Il est également important de décider de la taille de chaque groupe et de la durée de l'expérimentation avant de mener celle-ci. Si, au contraire, vous effectuez un test de signification en continu alors qu'un test A/B est en cours, et que vous déclariez le test réussi dès que vous voyez un résultat significatif, vous commettrez une erreur consistant à surestimer fortement la signification d'une expérimentation en recherchant la signification statistique de manière opportuniste (une fois encore, Evan Miller a une excellente explication à ce sujet[67]).

Bien que la plupart des expériences se concentrent sur la comparaison de la valeur d'une seule métrique, il est important de prendre également en compte d'autres impacts. Si le TDC moyen augmente mais que le nombre d'utilisateurs qui cessent d'utiliser le produit double, nous ne devrions probablement pas considérer qu'un modèle est meilleur.

De même, les tests A/B devraient tenir compte des résultats pour les différents segments d'utilisateurs. Si le TDC moyen augmente, mais qu'il s'effondre pour un segment donné, il peut être préférable de ne pas déployer le nouveau modèle.

63 Voir l'adresse `https://bit.ly/2XE2RT0`.
64 Voir l'adresse `https://eng.uber.com/xp/`.
65 Voir l'adresse `https://github.com/intuit/wasabi`.
66 Voir l'adresse `https://www.evanmiller.org/ab-testing/sample-size.html`.
67 Voir l'adresse `https://www.evanmiller.org/how-not-to-run-an-ab-test.html`.

L'implémentation d'une expérimentation nécessite la capacité d'affecter des utilisateurs à un groupe, d'assurer le suivi de l'affectation de chaque utilisateur et de présenter différents résultats en fonction de celle-ci. Cela nécessite la mise en place d'une infrastructure supplémentaire, que nous abordons ci-dessous.

Construire l'infrastructure

Les expérimentations s'accompagnent également d'exigences en matière d'infrastructure. La façon la plus simple d'effectuer un test A/B consiste à enregistrer le groupe associé à chaque utilisateur avec le reste des informations relatives à celui-ci, par exemple dans une base de données.

L'application peut alors s'appuyer sur une logique de branchement qui décide du modèle à exécuter en fonction de la valeur du champ voulu. Cette approche simple fonctionne bien pour les systèmes où les utilisateurs sont connectés, mais devient beaucoup plus difficile si un modèle est accessible aux utilisateurs qui ne sont pas connectés.

En effet, les expérimentations supposent généralement que chaque groupe est indépendant et exposé à une seule variante. Lorsque l'on sert des modèles à des utilisateurs qui ne sont pas connectés, il devient plus difficile de garantir qu'un utilisateur donné a toujours reçu la même variante à chaque session. Si la plupart des utilisateurs sont exposés à plusieurs variantes, cela pourrait invalider les résultats d'une expérimentation.

D'autres informations, telles que les cookies de navigation et les adresses IP, peuvent être utilisées pour identifier les utilisateurs. Une fois de plus, cependant, de telles approches nécessitent la construction de nouvelles infrastructures, ce qui peut être difficile pour de petites équipes aux ressources limitées.

Autres approches

Le test A/B est une méthode d'expérimentation populaire, mais il existe d'autres approches qui tentent de remédier à certaines de ses limites.

Les bandits à plusieurs bras (*multiarmed bandits*) constituent une approche plus souple qui permet de tester des variantes en continu et sur plus de deux alternatives. Ils mettent à jour de manière dynamique le modèle à utiliser en fonction des performances de chaque option. J'ai illustré le fonctionnement de ces « bandits » (dits aussi à K bras) sur la Figure 11.8. Ces bandits tiennent continuellement un compte de la performance de chaque option en fonction du succès de chaque requête qu'ils acheminent. La plupart des requêtes sont simplement dirigées vers la meilleure alternative actuelle, comme montré sur la partie gauche de la figure. Un petit sous-ensemble de requêtes est acheminé vers une alternative aléatoire, comme vous pouvez le voir sur la droite de la figure. Cela permet aux bandits de mettre à jour leur estimation du meilleur modèle, et de détecter si un modèle qui n'est actuellement pas desservi commence à donner de meilleurs résultats.

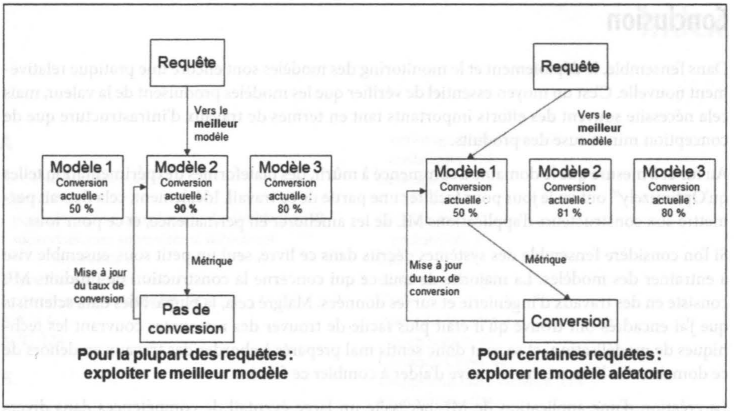


Figure 11.8 : Les bandits à plusieurs bras en pratique.

Les bandits à plusieurs bras contextuels poussent ce processus encore plus loin, en apprenant quel modèle constitue la meilleure option pour chaque utilisateur particulier. Pour plus d'informations, je recommande de consulter la présentation de l'équipe de Stitch Fix[68].

Bien que cette section traite de l'utilisation de l'expérimentation pour valider les modèles, les entreprises utilisent de plus en plus des méthodes d'expérimentation pour valider tout changement significatif qu'elles apportent à leurs applications. Cela leur permet d'évaluer en permanence les fonctionnalités que les utilisateurs trouvent utiles, ainsi que les performances des nouvelles fonctionnalités.

L'expérimentation étant un processus très difficile et sujet aux erreurs, de nombreuses start-up ont commencé à offrir des « services d'optimisation » permettant aux clients d'intégrer leurs applications sur une plateforme d'expérimentation hébergée pour décider quelles variantes sont les plus performantes. Pour les organisations qui ne disposent pas d'une équipe d'expérimentation dédiée, ces solutions peuvent être le moyen le plus simple de tester de nouvelles versions de modèles.

68 Voir l'adresse `https://bit.ly/36B1AQZ`.

Conclusion

Dans l'ensemble, le déploiement et le monitoring des modèles sont encore une pratique relativement nouvelle. C'est un moyen essentiel de vérifier que les modèles produisent de la valeur, mais cela nécessite souvent des efforts importants tant en termes de travaux d'infrastructure que de conception minutieuse des produits.

Au fur et à mesure que le domaine a commencé à mûrir, des plateformes d'expérimentation telles qu'Optimizely[69] ont vu le jour pour faciliter une partie de ce travail. Idéalement, cela devrait permettre aux constructeurs d'applications ML de les améliorer en permanence, et ce pour tous.

Si l'on considère l'ensemble des systèmes décrits dans ce livre, seul un petit sous-ensemble vise à entraîner des modèles. La majorité de tout ce qui concerne la construction de produits ML consiste en des travaux d'ingénierie et sur les données. Malgré cela, la plupart des data scientists que j'ai encadrés ont trouvé qu'il était plus facile de trouver des ressources couvrant les techniques de modélisation, et se sont donc sentis mal préparés à aborder des travaux en dehors de ce domaine. Ce livre est ma tentative d'aider à combler ce fossé.

La création d'une application de ML nécessite un large éventail de compétences dans divers domaines tels que les statistiques, le génie logiciel et la gestion de produits. Chaque partie du processus est suffisamment complexe pour justifier la rédaction de plusieurs ouvrages à ce sujet. L'objectif de ce livre est de vous fournir un large éventail d'outils pour vous aider à construire de telles applications et vous permettre de décider quels sujets approfondir en suivant par exemple les recommandations présentées dans la section « Ressources supplémentaires » de la préface.

Dans cette optique, j'espère que ce livre vous a donné des outils pour aborder avec plus de confiance la majorité des travaux liés à la construction de produits ML performants. Nous avons couvert chaque partie du cycle de vie des produits ML, en commençant par traduire un objectif produit en une approche ML, puis en trouvant et préparant des données et en itérant sur des modèles, avant de valider leurs performances et de les déployer.

Que vous ayez lu ce livre de bout en bout ou que vous vous soyez plongé dans les sections spécifiques les plus pertinentes pour votre travail, vous devriez maintenant avoir les connaissances requises pour commencer à construire vos propres applications ML.

Si ce livre vous a aidé à construire quelque chose ou si vous avez des questions ou des commentaires sur son contenu, n'hésitez pas à me contacter à l'adresse `mlpoweredapplications@gmail.com`. J'ai hâte d'avoir de vos nouvelles et de découvrir vos développements en ML.

69 Voir l'adresse `https://www.optimizely.com/`.

Index

E

F

G

V

W

O'REILLY®

Développer des applications machine learning

Acquérez les compétences nécessaires pour concevoir, construire et déployer des applications fondées sur l'apprentissage automatique (ou ML, pour *machine learning*). Au cours de ce livre pratique, vous construirez un exemple d'application, en allant de l'idée initiale au produit déployé. Que vous soyez spécialiste des données, ingénieur logiciel ou encore chef de produit, aussi bien professionnel expérimenté que novice, vous apprendrez pas à pas les outils, les meilleures pratiques dans ce domaine, ainsi que les défis liés à la création d'une application ML du monde réel.

L'auteur, data scientist expérimenté qui a dirigé un programme d'éducation sur le ML, explique les concepts du ML par la pratique en utilisant des extraits de code, des illustrations, des captures d'écran et des entretiens avec des experts. La première partie vous explique comment planifier une application ML et en mesurer la réussite. La deuxième partie apprend à construire un modèle de ML fonctionnel. La troisième partie vise à améliorer le modèle jusqu'à ce qu'il corresponde totalement à votre vision initiale. La quatrième partie couvre les stratégies de déploiement et de monitoring.

- Définissez l'objectif fixé pour votre produit et configurez un problème d'apprentissage automatique.
- Construisez rapidement votre premier pipeline de bout en bout et acquérez un jeu de données initial.
- Entraînez et évaluez vos modèles de ML, et attaquez-vous aux goulets d'étranglement des performances.
- Déployez et monitorez vos modèles dans un environnement de production.

« Les ouvrages sur l'apprentissage automatique font souvent l'impasse sur les sujet les plus difficiles : affiner le problème, déboguer les modèles et effectuer le déploiement chez les clients. Ce livre se concentre sur ces questions afin que vous puissiez faire passer vos projets de l'idée initiale à leur impact dans le monde réel. »

Alexander Gude
Data scientist chez Intuit

Emmanuel Ameisen, ingénieur en apprentissage automatique chez Stripe, a mis en œuvre et déployé des solutions d'analyse prédictive et de ML pour Local Motion et Zipcar. Récemment, il a piloté le programme d'IA chez Insight Data Science, en dirigeant plus d'une centaine de projets d'apprentissage automatique. Emmanuel est diplômé en intelligence artificielle, en ingénierie informatique et en management.

736551 X-20 **29,95 €**

MIXTE
Papier issu de sources responsables
FSC® C003309

editionsfirst.fr

FIRST
INTERACTIVE

9 782412 058022