Projet tutoré : Data Science

Compétition Kaggle : “Home Depot Product Search Relevance”

Réalisé par : Xavier Nabet et Victor Yon
Encardé par : Couceiro Miguel, Raïssi Chedy et Galbrun Esther

Master 1 - Sciences de la Cognition et Applications
2016

Compétition Kaggle : Home Depot Product Search Relevance

Réalisé par : Xavier Nabet et Victor Yon
Encardé par : Couceiro Miguel, Raïssi Chedy et Galbrun Esther

Année 2015-2016

Table des matières

1 Présentation du sujet

1.1 Science des données

Le sujet de ce projet tutoré s'inscrit dans le domaine très vaste de la science des données, en anglais *data science*. Cette discipline récente s'appuie sur des outils mathématiques, statistiques et informatiques afin de traiter et d'exploiter au mieux la grande quantité d'informations dont la société moderne est submergée. Plus précisément nous avons été placés face à des problèmes d'analyse de données, d'apprentissage automatique et de prédiction, le tout avec l'aide de l'outil informatique. Nous allons détailler dans ce rapport comment nous avons fait face à ces défis et quelles méthodes nous avons employées.

1.2 Kaggle

Kaggle[7] est un site internet communautaire basé sur le *data science*. Il est possible de créer des compétitions avec récompenses à la clé (de l'argent ou des contrats d'embauche principalement). Le commanditaire de la compétition (souvent une entreprise) soumet un problème, composé d'une description et d'un ou plusieurs jeux de données, sur Kaggle, et le site s'occupe d'organiser une compétition autour de ce problème.

Les participants choisissent donc les challenges auxquels ils veulent participer et tentent de résoudre le problème soumis de la meilleure façon possible, tout cela en compétition avec les autres participants. Pour cela, on a la possibilité de soumettre un fichier afin d'obtenir un score qui nous positionne dans un classement publique (qui regroupe tous les participants). Il est donc possible, à tout moment, de savoir qui sont les premiers, et de comparer son propre score à celui des autres.

Le site tient à avoir un aspect communautaire et encourage vivement tous les participants à se rendre sur le forum (système de discussion asynchrone) associé à la compétition. On peut trouver sur celui-ci de nombreux utilisateurs détaillant leurs problèmes rencontrés, leurs solutions ou tout simplement des informations qui permettent d'aider à avancer dans la résolution du problème.

A la fin de la compétition, le classement est bloqué et les récompenses sont distribuées (souvent au trio du podium). Il est tout de même possible de continuer à soumettre des résultats et d'être noté pour une compétition finie si on désire encore travailler et améliorer sa solution.

1.3 Sujet choisi

Le commanditaire ayant soumis le problème que nous avons choisi sur Kaggle est la célèbre compagnie américaine de distribution pour l'équipement de la maison : Home Depot[2]. En France, la comparaison pourrait être faite avec Leroy Merlin ou Castorama.

Pour faire ceci, Home Depot et Kaggle se sont associés pour proposer une compétition aux utilisateurs du site. Afin de récompenser et de montrer l'importance de leur problème, Home Depot a proposé 40 000$ au trio gagnant répartis de la façon suivante : 20 000$ pour le premier, 12 000$ pour le deuxième et 8 000$ pour le troisième.

Concernant le sujet en lui même, il s'agit d'une compétition qui a pour objectif d'améliorer leur système de pertinence de résultats lors qu'un utilisateur effectue une recherche. En effet, si un utilisateur fait une recherche en utilisant les mots clés "chaise blanche", il serait très inconfortable de lui présenter un robinet rouge dans la liste des résultats générés par sa recherche. Home Depot a donc fourni plusieurs jeux de données (voir figure 1.1) tels que la liste des descriptions des produits et la liste de leurs attributs. Un jeu de données dit de "train" est présent afin de permettre aux algorithmes d'apprendre et un jeu de "test" permet aux algorithmes d'appliquer leur apprentissage. La différence entre ces deux jeux de données réside dans le fait que le second (test) ne contient pas les niveaux de pertinance, seul Kaggle en à la connaissance. Le role des compétiteurs est de prédire la valeur de ce champ manquant et de fournir un fichier de soumission (sample_submission voir figure 1.1) afin d'être noté. Cette évaluation reflète alors la performance du programme pour l'attribution de la pertinence des associations "recherche-produit".La note est calculée sur le serveur du site en utilisant une formule $RMSE$[6] (Root Mean Squared Error), dont la formule est la suivante :

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2}$$

product_title : Prepac Elite 32 in. Wood Laminate Cabinet in White product_uid : 100189	
id : 1104 search_term : kitchen cupboards hinges	id : 1107 search_term : storage cabinets
Pertinence faible relevance : 1,33	Pertinence forte relevance : 3

TABLE 1.1 – Exemple d'un produits type associé à deux recherches d'après les données de *train*, sans la description ni les attributs (produit complet en annexe 1)

La nuance entre les notations est mis en évidence dans le tableau1.1, l'une étant notée à 1,33 et l'autre à 3, les notes possibles allant de minimum 1 à maximum 3.

Dans la liste des compétitions disponibles sur Kaggle, nous avons choisi celle-ci car, de notre point de vue, ce sujet regroupait beaucoup de matières présentes dans notre cursus. C'est un sujet complet, puisque nous sommes partis de rien et que nous avons dû établir toute notre stratégie de résolution.

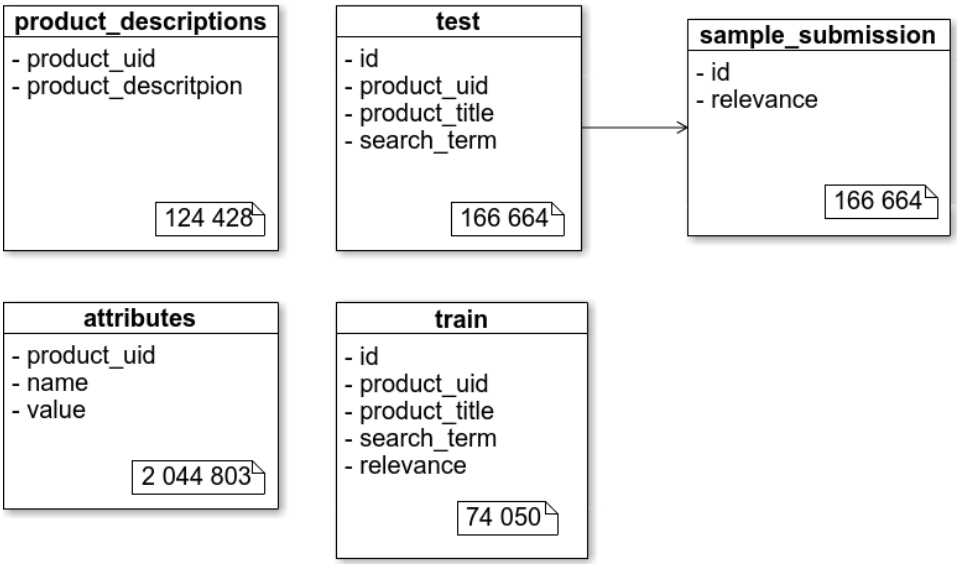

FIGURE 1.1 – Détails des fichiers fournis par Home Depot ainsi que le nombre d'entités

1.4 Contraintes

Chaque compétition a ses propres règles, mais certaines sont liées à Kaggle et doivent être acceptées pour participer aux compétitions. Parmi elles, nous pouvons retrouver les plus importantes :

— le fait que chaque participant n'a le droit qu'à un seul compte.

— le partage de codes entre participants (et équipes) n'est autorisé que s'il est fait publiquement sur le forum de la compétition.

— le fait que le nombre de soumissions soit limité à 5 par jour.

Concernant les règles propres à la compétition, on y trouve :

— les dates de celle-ci, à savoir : début le 18 janvier 2016 et fin le 25 avril 2016.

— le fait que les participants n'aient pas le droit d'utiliser des informations présentes sur le site de Home Depot hors de ce qui a été fourni pour la compétition.

— l'autorisation d'utiliser toutes données externes tant que celles-ci ne sont pas liées à Home Depot ni à tout autre entreprise dans le même secteur d'activités.

— le fait que toutes données externes utilisées doivent être postées sur le forum de la compétition.

1.5 Étapes de résolution

Les données fournies par Home Depot étant brutes, nous avons dû les comprendre et les analyser. Ceci ayant permis de mettre en oeuvre les compétences apprises lors du cours "Analyse de données". Lors de cette analyse, nous nous sommes rendus compte que les données contenaient de nombreuses petites erreurs telles que des fautes d'orthographe ou plusieurs abréviations différentes pour nommer la même unité (comme par exemple "in", "inch", "inches"). Grâce aux cours de "Traitement automatique des

langues" nous avons pu élaborer des expressions régulières permettant de régler ce problème.

Étant novices dans l'environnement de Kaggle, nous nous sommes renseignés (sur le forum de la compétition ainsi que sur celui d'autres compétitions) sur les méthodes utilisées pour résoudre les challenges soumis. En trouvant nos réponses, nous avons également trouvé un lien avec le cours "Mémoire et apprentissage numérique" puisque les méthodes utilisées par les participants sont souvent des algorithmes d'apprentissage. Nous avons donc utilisé les réseaux de neurones (vus en cours) ainsi qu'une méthode nouvelle pour nous et que nous avons apprise (*random forests*).

Nous avons appliqué ces deux méthodes et avons obtenu différents résultats. Nous allons maintenant vous détailler le travail que nous avons réalisé lors de ce projet tutoré.

2 Travail réalisé

2.1 Environnement de travail

En voyant le sujet, nous pensions utiliser le langage de programmation : Python. Nous étions libres de choisir le langage que nous désirions, mais nos tuteurs nous ont confirmé que l'utilisation de Python serait sûrement plus judicieuse grâce à la multitude de bibliothèques présentes pour le *data science*. Nous avons décidé d'utiliser la version 2.7 et non la 3.5 (plus récente) afin que les scriptes soit compatible avec l'environnement de travail de chacun. L'utilisation de Python était un défi pour nous (même si nous possédons tous deux un diplôme en informatique) car nous n'avions pas d'expérience avec ce langage, ce qui a constitué une compétence supplémentaire que nous avons dû acquérir.

Afin de travailler plus efficacement, de permettre un partage des fichiers efficace (plus que de s'échanger les fichiers par clés USB), nous avons utilisé Git[4] avec un dépôt sur le site BitBucket[1]. Ceci nous a également permis de partager notre travail avec nos tuteurs qui, par conséquent, pouvaient à tout moment regarder où nous en étions dans notre travail et ainsi leur permettre de nous aiguiller au mieux dans la marche à suivre.

Nous avons réaliser l'analyse du problème et des données ensemble, puis nous avons pris la décision de travailler chacun sur une méthode afin d'explorer davantage de pistes de recherche sur la résolution de notre problème. Nous avons alors choisi d'approfondir les réseaux de neurones (détails en partie 2.5) ainsi que les forêts aléatoires (détails en partie 2.4).

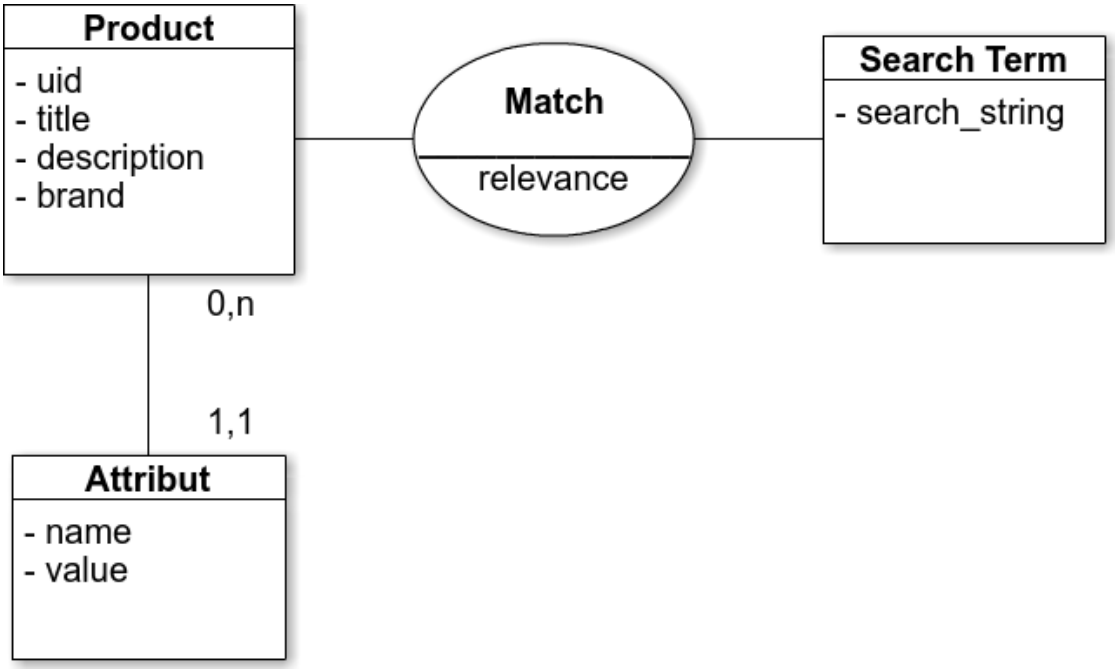

FIGURE 2.1 – Modélisation UML de la structure de données

2.2 Les données

2.2.1 La structure

La première étape de la modélisation du problème a été de formaliser la structure de données. La façon dont les fichiers ont été découpés n'étant pas très intuitive nous avons divisé les données en quatre parties plus lisibles et plus facilement exploitables (voir figure 2.1). La cinquième partie *sample_submission* étant le fichier de résultats.

— *Product* : représente un produit, identifié de façon unique par son *uid*. Chaque produit possède un titre, une description, une marque et un nombre variable d'attributs

— *Attribut* : représente un attribut d'un produit, défini par un nom et une valeur (doublon possible)

— *Search Term* : représente une recherche sous forme de texte, réalisée par un utilisateur (doublon possible et recherche vide possible)

— *Match* : représente le lien entre un produit et une recherche, la pertinence de ce lien est quantifiée dans l'intervalle $[0, 3]$. Pour une recherche présente dans le lot de *train* cette information est initialisée avec une valeur, pour les recherches du lot de *test* la valeur est initialement vide puis est estimée d'après les différentes méthodes de prédictions

2.2.2 Analyse

Une fois la structure bien établie nous nous sommes penchés sur le contenu des données. Afin d'orienter notre approche du problème nous avons analysé les caractéristiques

des données. Pour cela nous avons utilisé des tableurs, des scripts python et les données récoltées de façon collaborative sur le forum dédié à la compétition.

La première question que nous nous sommes posée est la suivante : est-il pertinent d'utiliser les données d'entraînement (*train*) pour essayer de prédire les données manquantes dans celles de *test* ? Cette question, volontairement naïve, se doit de trouver une réponse le plus explicitement possible pour légitimer l'utilisation des méthodes détaillées par la suite.

Les produits étant communs aux deux jeux de données, le problème est vite résolu pour ce point, il est naturellement possible de faire l'analogie entre deux champs identiques.

Le champ le plus important est probablement celui des termes de recherche (*search_string*) qui conditionnent directement la note de pertinence (*relevance*) que l'on doit prédire. On est en droit de se demander si les champs de recherche fournis pour l'apprentissage sont bien équivalents à ceux de test.

Le nombre de recherches uniques dans les 166664 données de test est de 22457 (soit 86.5% de doublon). Et le nombre de recherches uniques dans les 74050 données d'entraînement est de 11795 (soit 84.1% de doublon).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
test	1286	5813	7618	4387	2139	750	257	94	41	20	11	9	1	1
train	644	2884	3977	2432	1180	427	164	50	18	5	9	4	0	1

TABLE 2.1 – Fréquence des recherches (sans doublon) en fonction du nombre de mots (d'après [3])

En observant la table 2.1, on peut supposer que la taille des recherches entre le groupe d'entraînement et de test semble varier de la même façon,. Pour s'en assurer nous avons comparé statistiquement ces deux groupes. Pour les 14 entrées le coefficient de corrélation est de 0.99891 Ce qui signifie que la fréquence de nombre de mots dans les deux ensembles sont statistiquement équivalents et donc, nous pouvons rejeter l'hypothèse que cette ressemblance soit liée au hasard.

Une autre observation qui nous conforte dans l'idée que ces deux groupes sont semblables et que sur les 22427 données de test (sans doublon) 42.9% des termes sont présents dans les données d'entraînement.

Ces résultats nous ont permis de conclure qu'il était pertinent d'utiliser les données d'entraînement afin de prédire la pertinence (*relevance*) des données de test.

La seconde question que nous nous sommes posée face à ces données est : quelle importance donner aux attributs et comment les exploiter au mieux ? En effet le nombre d'attributs présents dans les données de la compétition est relativement élevé : plus de deux millions (soit en moyenne 16 par produit). Certains nous ont semblé plus importants que d'autres, par exemple on peut supposer qu'il est plus intéressant de considérer les attributs de taille et de couleur que le type de finitions.

On peut constater (d'après la figure 2.3) que la majorité des produits possèdent entre 10 et 30 attributs mais certains sont très sensibles au contexte. Par exemple des attributs tels que la marque sont présents pour tous les produits alors que les voltages ne sont

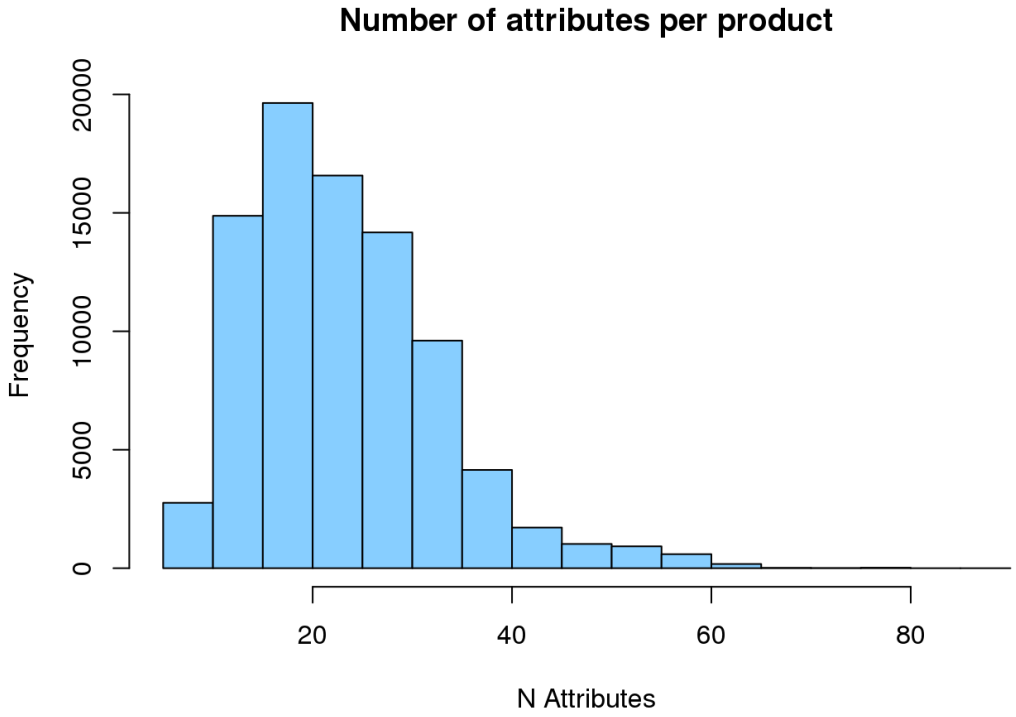

FIGURE 2.2 – Nombre d'attributs par produit (d'après [3])

indiqués que pour des produits spécifiques. Nous avons dénombré plus de 5000 attributs différents mais seulement les 30 premiers sont présents dans plus de 5% des produits. Ce qui signifie qu'une large majorité d'entre eux sont anecdotiques, certains attributs ne sont même présents que chez un seul produit.

Afin de traiter les attributs de façon plus fine nous avons décidé d'extraire deux groupes dont la sémantique est légèrement différente :

— les points ("bullets") : ce sont des attributs génériques qui se présentent sur la fiche de produits comme une simple liste. Presque tous les produits en possèdent entre 1 et 15 et nous avons estimé qu'il n'était pas cohérent de les considérer de la même façon que les autres attributs qui possèdent eux un nom associé à une valeur.

— la marque : cet attribut est le plus souvent représenté et son poids dans les recherches est tel que nous avons décidé de le considérer à part. C'est pour cela que dans la figure 2.1 la marque est considérée comme une propriété du produit.

La dernière question notable que nous nous sommes posés durant l'analyse des données concerne le type de mots présents dans les recherches.

Nous avons constaté que les champs de recherche étaient dépourvus de mots de liaisons et autres mots fiables en sémantique dans ce contexte ("the", "a", "cheap", ...). Probablement car le site Home Depot effectue ce traitement au préalable, ce qui nous a permis de ne pas nous soucier de ce point lors du nettoyage futur.

De part la nature du catalogue du site certaines catégories de mots sont plus courantes que d'autres et nécessitent une attention particulière. Par exemple les adjectifs de cou-

FIGURE 2.3 – Mots clés les plus présents dans les recherches du groupe d'entraînement

leurs et les noms de matériaux reviennent très souvent. Pour la même raison nous avons constaté que des mesures sont régulièrement spécifiées dans les recherches, dans diverses unités de mesure, mais ce sont toujours les unités anglophones qui sont utilisées (pouces, pieds ...), aussi bien dans les recherches que dans les descriptions et attributs des produits.

2.2.3 Les erreurs

Durant cette analyse nous avons constaté diverses erreurs de syntaxe et de sémantique dans les données fournies par le site. Certaines phrases liées à des promotions du site qui n'apportaient aucune information sur le produit étaient parfois présentes à la fin des descriptions. Les attributs de type "points" étaient toujours présents en double, une fois dans la liste des attributs et une fois dans la description du produit. De la même façon la marque du produit était spécifiée deux fois : une fois dans la liste des attributs et une fois au début du titre du produit. Afin de ne pas fausser les résultats en comptabilisant plusieurs fois les mêmes mots nous avons traité ces problèmes dans la phase de nettoyage des données (détails à la partie 2.3.2).

Des fautes de frappes sont aussi assez fréquentes dans les termes de recherche, par exemple "toliet" pour "toilet" ou "skil" pour "skill". Et certains noms d'attributs ayant le même sens étaient parfois orthographiés de deux façons différentes dans la liste des attributs : "Indoor/Outdoor" et "Indoor,Outdoor" sont tous les deux présents sur différents produits.

2.3 Modélisation du problème

Une fois le problème bien posé et la structure de données clairement définie nous avons pu commencer la pratique. Dans cette étape de modélisation l'objectif était de coder un ensemble d'outils qui devait être à la fois suffisamment générique pour permettre d'exploiter les données avec différentes méthodes de data science. Et suffisamment performants pour obtenir des résultats en un temps raisonnablement court.

Pour ce faire nous avons donc utilisé python (pour les raisons exposées dans la partie 2.1), ainsi que certaines de ses nombreuses bibliothèques. Les plus notables sont les suivantes :

— *Pandas*[13] : Fournit des structures de données haute performance et des outils d'analyse. Ce qui nous a été utile pour charger les données depuis les fichiers CSV.

— *NumPy*[12] : Propose des outils semblables à Pandas mais est nécessaire pour certaines dépendances

— *Pickle*[14] : Permet de sérialiser et dé-sérialiser facilement des instances d'objets python. Ce qui nous a permis de gagner du temps sur de nombreuses exécutions.

— *Keras*[8] : Cette bibliothèque est l'une des nombreuses disponibles pour travailler avec les réseaux de neurones sur python. Nous avons choisi celle-ci aux autres (PyBrain, Lasagne, ...) pour sa documentation riche et sa modularité relativement facile à utiliser.

2.3.1 Problèmes rencontrés

Nous avons rencontré divers problèmes lors de la prise en main de python et des dépendances listées précédemment.

Python 2.7 ne gérant pas nativement l'encodage unicode[18], le chargement des fichiers fournis par Kaggle nous a donné du fil à retordre. Certains caractères mal interprétés provoquaient des erreurs durant la phase de nettoyage (le symbole degré par exemple : °). Pour pallier à ce problème nous avons dû remplacer explicitement les caractères incriminés durant le chargement de données.

Le second problème auquel nous avons dû faire face est le temps de traitement des données (lecture depuis les fichiers CSV et nettoyage) qui pouvait prendre plus de 30 minutes. Ce délai devenait vraiment handicapant lors des phases de test où l'on voulait comparer plusieurs configurations. La solution à été de mettre en place un système de cache, grâce à *Pickle*. De cette façon les données ne sont pas traitées systématiquement, mais uniquement lorsque on le demande explicitement (après une modification de la procédure de nettoyage par exemple). Avec cette méthode le chargement des données est plus fluide et peut être effectué en moins de 30 secondes.

2.3.2 Nettoyage des données

Une partie cruciale, lorsque l'on s'adonne à la science des données, est de bien nettoyer celles-ci. C'est pourquoi nous avons pris soin de mettre en place une procédure de nettoyage rigoureuse et complète en se basant sur l'analyse réalisée précédemment.

L'ordre des différentes étapes est important car certaines peuvent en perturber d'autres. Les termes des recherches sont nettoyés de la même façon que tout le texte qui définit un produit pour que les chaînes de caractères qui en résultent soient comparables.

La première étape consiste à supprimer des erreurs inhérentes aux données, que nous avons listées dans la partie 2.2.3. Ensuite pour limiter l'impact de la syntaxe lors de la mise en relation entre un produit et une recherche nous avons remplacé toutes les majuscules par des minuscules (de cette façon le mot "Door" devient strictement égal au mot "door"). Pour la même raison toutes les suites de caractères blancs (espaces, tabulations, sauts de ligne, ...) sont remplacés par un simple espace. Nous avons ensuite listé d'autres règles de remplacements plus spécifiques pour le format des nombres (séparateurs des milliers et des décimales) et sur le formalisme de certains termes techniques.

A l'aide de nos connaissances en traitement automatique des langues et des expressions régulières nous avons pu détecter, unifier les différentes notations des 32 unités de mesures les plus récurrentes. Par exemple les notations "5 inches", "5 in.", "5 inch" seront toujours remplacées par "5_in", de cette façon, il est aisé de faire correspondre une recherche de taille avec les spécifications d'un produit et ce, même si l'unité n'est pas écrite de la même façon.

Ensuite toute la ponctuation est supprimée et finalement un traitement de racinisation (stemming en anglais) est effectué sur tous les mots. Ce traitement consiste à ne garder que la racine des mots, par exemple les mots "fishing", "fished", "fish" et "fisher" donnent "fish". Le but de cette manœuvre est de rendre les comparaisons entre chaînes de caractères plus permissives et plus cohérentes dans la majorité des cas. Pour réaliser ce traitement nous avons utilisé la bibliothèque *SnowballStemmer* qui utilise l'algorithme *Snowball* pour associer des mots avec leurs racines.

2.3.3 Les variables

Les méthodes de prédiction que nous avons planifié d'utiliser n'étant pas capables d'interpréter et de traiter directement du texte nous avons dû faire nous même le travail d'extraction des variables (*features* dans le vocabulaire du *data science*). Ces variables doivent exprimer au mieux la relation entre un produit et les mots d'une recherche.

Voici la liste des variables que nous avons décidé d'extraire :

Description	Quantitatif / Qualitatif	Valeurs	Notes
* nombre de mots correspondants dans le titre	quantitatif	0,n	
* nombre de mots correspondants dans la description	quantitatif	0,n	
* nombre de mots correspondants dans les points	quantitatif	0,n	
* la marque est présente dans la recherche	qualitatif	booléen	-1 si la marque est trop petite ou vide
* la marque est partiellement présente dans la recherche ?	qualitatif	booléen	-1 si la marque est trop petite ou vide
* la recherche se trouve exactement dans le titre	qualitatif	booléen	Mêmes mots dans le même ordre
* la recherche se trouve exactement dans la description	qualitatif	booléen	Mêmes mots dans le même ordre
* la recherche se trouve exactement dans les points	qualitatif	booléen	Mêmes mots dans le même ordre
indice de différence entre les mesures ayant les mêmes unités	quantitatif	[0,1]	1 = exactement les mêmes mesures 0 = aucune similarité dans les mesures
proximité des couleurs	quantitatif	[0,1]	d'après : code RGB[17] / attributs de couleurs / description
proximité des matériaux	quantitatif	[0,1]	
correspondance commerciale / résidentielle	qualitatif	booléen	-1 s'il n'y pas d'indication dans la recherche
correspondance intérieure / extérieure	qualitatif	booléen	-1 s'il n'y pas d'indication dans la recherche
* nombre de mots dans la recherche	quantitatif	0,n	
rapport entre le nombre de correspondances dans le titre et le nombre de mots	quantitatif	[0,1]	0 = aucun mot qui match dans le titre 1 = chaque mot match au moins une fois dans le titre
rapport entre le nombre de correspondances dans la description et le nombre de mots	quantitatif	[0,1]	0 = aucun mot qui match dans la description 1 = chaque mot match au moins une fois dans la description

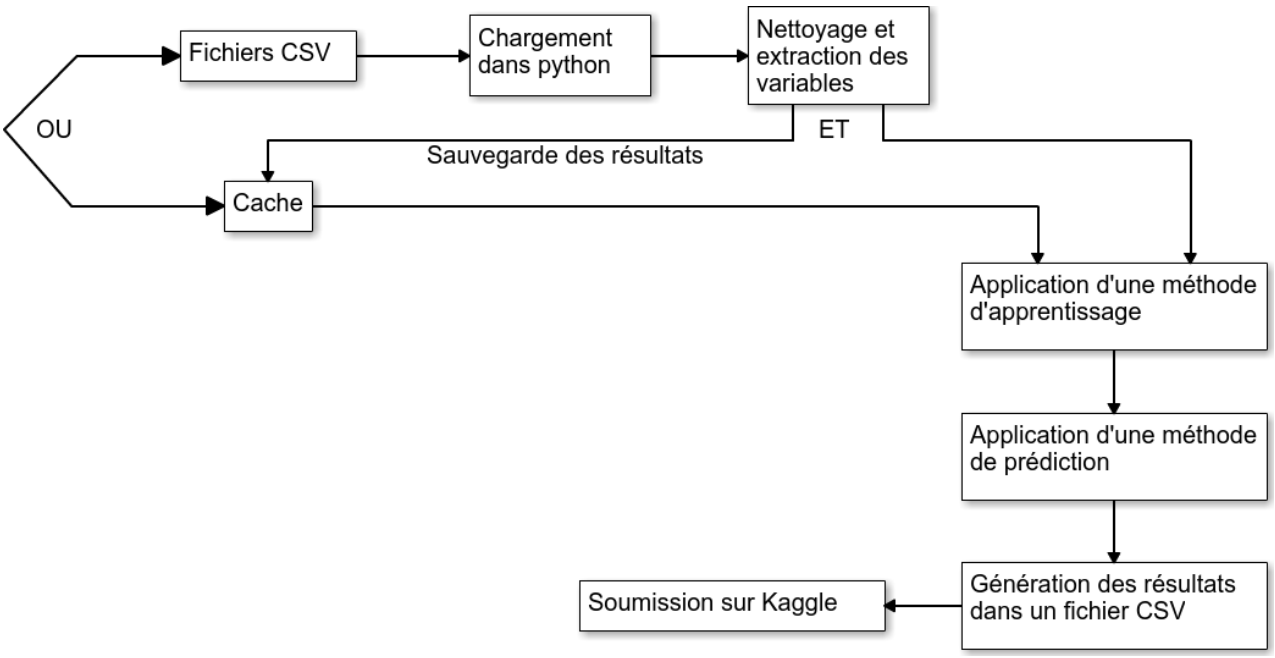

FIGURE 2.4 – Procédure de traitement des données issue de l'analyse

Les descriptions commençant par une étoiles sont les variables que l'on a implémenté et que l'on a utilisé par la suite.

L'extraction des variables se fait en parallèle du nettoyage car certains traitements peuvent fausser des résultats. Par exemple les variables qui exploitent la marque du produit sont traitées avant l'étape de racinisation alors que les autres correspondances de mots se font après.

La figure 2.4 résume la procédure de travaille que nous avons mis en place, en partant des données brutes, pour arriver a l'obtention de résultats. Et ce, quelles que soient les méthodes d'apprentissage et de prédiction choisies.

2.4 Forêt d'arbres décisionnels

Les "random forests"[10] (ou forêts aléatoires en français), sont des méta-estimateurs qui correspondent à un certain nombre de classifications des arbres de décision sur les différents sous-ensembles du jeu de données, ainsi que sur l'utilisation moyenne pour améliorer la prédiction. La taille des sous-ensembles est toujours la même que la taille originale du jeu de données entré en paramètre, mais les ensembles sont tirés de manière aléatoire avec remise.

Comme dit précédemment dans le rapport, nous nous sommes renseignés sur les algorithmes les plus utilisés dans les compétitions de Kaggle et en *data science*. Dans nos recherches, les forêts aléatoires ont été largement vantées de mérites tant pour le peu de puissance de calcul nécessaire, tant pour les résultats vraiment satisfaisants que cette méthode fournie, et étant inconnue pour nous, nous avons été encouragés par nos tuteurs à développer cette méthode afin d'en approfondir la découverte.

La principale difficulté de cette méthode réside dans le fait d'utiliser un format de données compatibles. En effet, lors des débuts du projet, nous avons été confrontés à un problème d'encodage des données, ce qui nous a conduit à développer des fonctions pour y remédier. Or, dès que nous avons commencé à travailler sur les réseaux de neurones et les forêts aléatoires nous avons du repenser ses fonctions pour que le format des données corresponde aux critères des algorithmes.

Pour un algorithme de forêts aléatoires que nous allons définir comme "basique", les besoins en puissance de calculs sont faibles si l'on compare à l'efficacité et à la qualité des résultats fournis (comme précisé plus haut). Dans cette configuration, le temps d'apprentissage n'excède pas quelques minutes (le plus long dans notre exécution étant le traitement de données, ce qui nous mena à utiliser une mémoire cache afin de pouvoir recharger les données de façon plus rapide). De plus, comme toute méthode d'apprentissage, plus celle-ci est poussée dans le nombre ou la complexité de ses paramètres, plus le temps d'exécution et d'apprentissage seront élevés (on peut parler de plusieurs jours voire plusieurs semaines).

Grâce à Python et sa richesse en terme de bibliothèques, nous avons pu facilement mettre en œuvre les forêts aléatoires. En effet, dans la bibliothèque *sklearn* (*scikit-learn*), il y a un package *ensemble* contenant tout le nécessaire afin d'utiliser les forêts aléatoires tant pour la classification que pour la régression (nous avons bien évidemment utilisé la régression pour tenter de résoudre notre problème).

Concernant les paramètres de cette méthode, ils sont nombreux[16] (nous n'allons donc pas tous les lister), mais nous allons exposer ceux que nous avons utilisé :

— *n_estimators* : correspond au nombre d'arbres que nous souhaitons utiliser dans notre forêt.

— *n_jobs* : correspond au nombre de tâches à effectuer en parallèle pour l'apprentissage et la prédiction. Il est également possible de lui donner tous les coeurs disponibles du processeur afin que l'algorithme profite de toute la puissance de l'ordinateur sur lequel il fonctionne.

— *random_state* : ce paramètre permet d'implanter une graine utilisée par le générateur de nombres aléatoires. Ceci est utile afin de contrôler l'efficacité des autres paramètres de sorte à ne pas avoir des améliorations (ou baisses) de la note obtenue lors de la soumission qui peuvent être dues justement au hasard.

— *verbose* : correspond au contrôle de la verbosité du processus de construction des arbres (cela permet d'avoir plus de détails sur la création des arbres).

Les prédictions résultantes de l'application de cette méthode sont plutôt satisfaisantes. Nous avons obtenus une note de 0.50 lors des soumissions (les premiers de la compétition étant arrivés à 0.43).

Pour approfondir la question des arbres nous avons pris connaissance d'une méthode nommée *XGBoost* qui utilise les arbres décisionnels et qui est une implémentations de l'algorithme de *boosting du gradient*[19]. Nous n'avons, par contre, pas eu l'occasion de la mettre en application dans ce projet.

2.5 Réseau de neurones artificiels et apprentissage profond

2.5.1 Description de la méthode

Un réseau de neurones artificiels est le parfait exemple de ce que rend possible le travail conjoint des sciences cognitives et de l'informatique. Notre cerveau excelle dans la détection de schémas, la classification et dans la prédiction (de trajectoires, d'événements répétitifs, ...). Et ce sont ces compétences qu'il est intéressant de reproduire artificiellement pour résoudre notre problème. Plus précisément, le type d'apprentissage employé ici est dit "profond" (en anglais *deep learning*). Il permet d'exploiter correctement plusieurs niveaux de couches de neurones[11].

Cette méthode nous a semblé parfaitement adéquate pour traiter notre problème car nous avons besoin d'une capacité de prédiction et généralisation que les réseaux de neurones sont capables de fournir. L'idée est "d'habituer" le réseau à faire le lien entre un ensemble de variables et une note de pertinence grâce aux données d'entraînement. Puis de lui demander de généraliser cette "connaissance" sur les données de test. Concrètement nous soumettons les résultats des variables (voir partie 2.3.3) sous la forme d'un vecteur à la couche d'entrée du réseaux de neurone. Le vecteur étant sous cette forme : $[1, 12, 4, 0, 1, 0, 1, 6]$ et la sortie : 1.33 (valeurs arbitraires).

2.5.2 Implémentation

Dans un premier temps nous avons essayé d'implémenter notre modèle avec la bibliothèque PyBrain[15] mais elle s'est avérée trop limitée car elle ne permet pas la gestion des différentes couches avec précision. De plus les calculs sont réalisés sur le CPU (processeur) exclusivement ce qui rend les phases d'apprentissages assez longues. Après s'être plus amplement documentés sur la question, nous avons essayé la bibliothèque Lasagne[9] qui promettait plus de libertés. Cette fois le problème opposé s'est posé à nous, les options sont très nombreuses et la documentation cible un public de professionnels de l'apprentissage automatique difficilement accessible pour nous. Ce qui nous a finalement amené à nous tourner vers Keras[8] qui s'est avéré être un juste milieu entre les deux bibliothèques évoquées précédemment. Il offre en plus la possibilité de configurer l'exécution pour que le plus gros des calculs s'effectuent sur le GPU (carte graphique) via un pilote fourni par *Nvidia*, ce qui nous a fait gagner un temps précieux. Nous avons donc implémenté notre modèle en nous basant sur les outils proposés par Keras.

Afin de trouver la configuration la plus adaptée au problème nous avons essayé de faire varier les paramètres suivants :

— Le nombre de couches cachées du modèle (de 3 à 6)

— le nombre de neurones présents sur chaque couches (de 256 à 1024)

— La proportion de *dropout* (de 0% à 20%). Ce qui ajoute un facteur aléatoire entre chaque couches de façon à limiter les effet de "surapprentissage".

Les résultats sont semblables à ceux obtenus avec la méthode des forêts aléatoires : le score est de 0.50 d'après le système d'évaluation de Kaggle. Bien que les pertes cal-

culées à la fin de l'apprentissage sont moitié moins élevées (environ 0.25). On peut expliquer cette différence par un effet de "surapprentissage" des données d'entraînements, mais nous n'avons pas réussi à régler ce problème, même avec la variation de la proportion de *dropout*.

N'ayant pas réussi à améliorer le score en faisant varier les paramètres du modèle nous avons fait l'hypothèse que notre résultat était limité par la qualité des variables et du nettoyage des données, non par le modèle en lui même. Cette théorie est confortée par le fait que nous obtenons la même limite avec la méthode de forêt d'arbres décisionnels qui utilise les mêmes données ayant subi les mêmes traitements.

3 Conclusion

Le contexte de la compétition nous a plongé dans un environnement aussi rude que passionnant où l'on a été amenés à concrétiser et approfondir des domaines que nous n'avions exploré que d'un point de vue théorique jusqu'à ce jour. Le fait de pouvoir comparer notre travail avec ceux de plusieurs centaines de professionnels du domaine fut très stimulant. En plus d'orienter certaines de nos méthodes cela nous a donné un aperçu de ce qu'il était possible d'accomplir pour des personnes qui travaillent dans le domaine du *data science* au quotidien.

Nous avons mis a profit les connaissances acquises durant les cours de "fouille de données" pour la partie d'analyse ainsi que celui de "mémoire et apprentissage numérique" pour des réseaux de neurones. Le cours de TAL (traitement automatique des langues) nous à aussi aidé à mener à bien faire le nettoyage des données. Et bien évidement nos compétences de programmations ont été mises à rude épreuve durant l'utilisation de Python et de ses nombreuses bibliothèques.

Ce projet étant complet, dans le sens où nous partions de rien et avions pour objectif de terminer avec des résultats, nous avons dû passer par toutes les étapes nécessaires au bon déroulement de celui-ci. Nous avons du réaliser l'analyse du problème et des données, se documenter sur les méthodes à employer, appliquer ces méthodes et finalement traiter les résultats. Nous nous sommes donc organisés pour répartir toutes ces étapes sur toute la durée du projet en prenant en compte les contraintes de la compétition et celles du projet tutoré. Les gagnants de la compétition ont récemment mis en ligne les détails de la méthode qu'ils ont utilisé pour arracher la première place[5]. Il est intéressant de comparer leur approche à la notre pour constater que l'on était sur la bonne voie et qu'il nous manquait surtout de l'expérience et du recul.

Nos résultats ne nous ont, certes, pas permis d'atteindre le haut du classement sur Kaggle mais nous avons appris à exploiter des méthodes de traitements de données : le stemming, l'extraction de variables, analyse de données. Mais aussi des méthodes de prédiction : les réseaux de neurones et les forêts aléatoires. Et surtout à savoir sous quelles conditions utiliser ces dernières. Nous ne manquerons pas de les utiliser à nouveau lorsque l'occasion se présentera pendant la suite de nos études ou de notre vie professionnelle.

Bibliographie

[1] BITBUCKET. Git solution for professional teams. www.bitbucket.org, 2016. [En ligne ; Page disponible le 31-mai-2016].

[2] DEPOT, H. Home improvement. www.homedepot.com, 2016. [En ligne ; Page disponible le 31-mai-2016].

[3] FORUM, K. Beginner data analysis. https://www.kaggle.com/dsoreo/home-depot-product-search-relevance/testing-r, 2016. [En ligne ; Page disponible le 31-mai-2016].

[4] GIT. Home page. www.git-scm.com, 2016. [En ligne ; Page disponible le 31-mai-2016].

[5] HOMEDEPOT, G. K. Solution for home depot product search relevance competition on kaggle. https://github.com/ChenglongChen/Kaggle_HomeDepot, 2016. [En ligne ; Page disponible le 31-mai-2016].

[6] KAGGLE. Root mean squared error. www.kaggle.com/wiki/RootMeanSquaredError, 2016. [En ligne ; Page disponible le 31-mai-2016].

[7] KAGGLE. Your home for data science. www.kaggle.com, 2016. [En ligne ; Page disponible le 31-mai-2016].

[8] KERAS. Deep learning library for theano and tensorflow. www.keras.io/, 2016. [En ligne ; Page disponible le 31-mai-2016].

[9] LASAGNE. Github de lasagne. https://github.com/Lasagne/Lasagne, 2016. [En ligne ; Page disponible le 31-mai-2016].

[10] LIAW, A. Classification and regression by randomforest. ftp://131.252.97.79/Transfer/Treg/WFRE_Articles/Liaw_02_Classification%20and%20regression%20by%20randomForest.pdf, 2002. [En ligne ; Page disponible le 31-mai-2016].

[11] NIELSEN, M. Neural networks and deep learning. http://neuralnetworksanddeeplearning.com, 2016. [En ligne ; Page disponible le 31-mai-2016].

[12] NUMPY. Fundamental package for scientific computing with python. www.numpy.org, 2016. [En ligne ; Page disponible le 31-mai-2016].

[13] PANDAS. Python data analysis library. www.pandas.pydata.org, 2016. [En ligne ; Page disponible le 31-mai-2016].

[14] PICKLE. Python object serialization. www.docs.python.org/2/library/pickle.html, 2016. [En ligne ; Page disponible le 31-mai-2016].

[15] PYBRAIN. Modular machine learning library for python. http://pybrain.org, 2010. [En ligne ; Page disponible le 31-mai-2016].

[16] SKLEARN. Random forest regressor. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html, 2016. [En ligne ; Page disponible le 31-mai-2016].

[17] WIKIPÉDIA. Rouge vert bleu — wikipédia, l'encyclopédie libre. http://fr.wikipedia.org/w/index.php?title=Rouge_vert_bleu&oldid=125700728, 2016. [En ligne ; Page disponible le 31-mai-2016].

[18] WIKIPÉDIA. Unicode — wikipédia, l'encyclopédie libre. http://fr.wikipedia.org/w/index.php?title=Unicode&oldid=126494328, 2016. [En ligne ; Page disponible le 31-mai-2016].

[19] XGBOOST. Introduction to boosted trees. http://xgboost.readthedocs.io/en/latest/model.html, 2016. [En ligne ; Page disponible le 31-mai-2016].

4 Annexes

Annexe 1 : Exemple complet d'un produit (uid 100189)

Titre :

Prepac Elite 32 in. Wood Laminate Cabinet in White

Description :

With four more inches in depth added to the standard 12, the Elite 32 in. Storage Cabinet offers you even more storage potential for your laundry room, workshop or garage. It has one fixed and two adjustable shelves, allowing you to fit a wide variety of items. Use it alone or add the optional 32 in. Stackable Wall Cabinet on top for a total of 89 in. vertical of storage.Stylish brushed metal handles2 adjustable shelves and 1 fixed shelfDoors feature high quality, European style 6 way adjustable hingesMDF doors with profiled (rounded) edges add a level of sophisticationFinished in durable fresh white laminateConstructed from CARB-compliant, laminated composite woods with a sturdy MDF backerShips ready to assemble, includes an instruction booklet for easy assembly and has a 5 year manufacturer's limited warranty on partsAt 16 in. it's deeper than standard cabinetsAssembled dimensions : 32 in. W x 65 in. H x 16 in. D

Attributs :

“Adjustable Shelves”	“No”
“Assembled Depth (in.)”	“16.0 in”
“Assembled Height (in.)”	“65 in”
“Assembled Width (in.)”	“32 in”
“Assembly Required”	“Yes”
“Bullet01”	“Stylish brushed metal handles”
“Bullet02”	“2 adjustable shelves and 1 fixed shelf”
“Bullet03”	“Doors feature high quality & European style 6 way adjustable hinges”
“Bullet04”	“MDF doors with profiled (rounded) edges add a level of sophistication”
“Bullet05”	“Finished in durable fresh white laminate”
“Bullet06”	“Constructed from CARB-compliant & laminated composite woods with a sturdy MDF backer”
“Bullet07”	“Ships ready to assemble & includes an instruction booklet for easy assembly and has a 5 year manufacturer’s limited warranty on parts”
“Bullet08”	“At 16 in. it’s deeper than standard cabinets”
“Bullet09”	“Assembled dimensions : 32 in. W x 65 in. H x 16 in. D”
“Collection Name”	“None”
“Color/Finish”	“White”
“Color/Finish Family”	“White”
“Finish”	“White”
“Hardware Included”	“Yes”
“Material”	“MDF / Composite Wood”
“MFG Brand Name”	“Prepac”
“Number of Doors”	“2”
“Number of Drawers”	“0”
“Number of Shelves”	“3”
“Product Depth (in.)”	“16”
“Product Height (in.)”	“65”
“Product Weight (lb.)”	“121”
“Product Width (in.)”	“32”
“Removable Shelves”	“Yes”
“Storage Capacity (cu. ft.)”	“18”
“Style”	“Classic”

Annexe 2 : exemples de termes de recherche

id	uid produit	terme de recherche
13	100004	“solar panel”
14	100005	“1 handle shower delta trim kit”
19	100006	“cooking stove”
24	100008	“self tapping screws”
25	100009	“3 1/20door casing”
26	100009	“door trim”
28	100010	“anchor stakes”
31	100010	“metal stakes”
32	100010	“stakes”
33	100010	“steel landscape edging”
36	100011	“electric start gas mower”
47	100012	“10 roll up outside blinds”
53	100013	“garbage disposal air switc”