Data Challenge : Priceminister

Arnaud ZANNI, Emmanuel SOYER, Raphaël LARSEN

25 Mars 2017

1 Introduction

1.1 Présentation du projet

L'objectif de ce projet est de prédire si le commentaire d'un utilisateur sur un produit donné peut se révéler utile pour d'autres utilisateurs. L'algorithme final sera utilisé pour ranger et filtrer les millions de commentaires sur PriceMinister pour améliorer la qualité de l'expérience de l'utilisateur.

Ce projet soulève un problème qui n'a pas encore été résolu de façon optimale par la communauté scientifique :

Ce n'est pas une analyse de sentiments comme le Kaggle d'IMDB qui consiste à savoir si le commentaire d'un film est positif ou négatif. Ce dernier est plus simple à résoudre car l'information recherchée pour la classification se trouve directement dans le vocabulaire ; on recherche des mots à connotation péjorative, neutre ou positive.

Il ne s'agit pas non plus de trouver des thèmes sous-jacents aux commentaires pour ensuite classer des documents selon leur sujet politique, scientifique, religieux, etc.

Il s'agit d'une classification binaire fondée sur le jugement que les gens peuvent avoir sur un texte. Il y a de facto une prise de distance avec le jeu de données. Une classification du commentaire objectif/subjectif n'est pas suffisante ; c'est plus subtile que les problématiques précédentes (pour lesquelles la littérature a déjà produit un grand nombre d'articles à la pointe de la performance).

1.2 Présentation du jeu de données

Le jeu de données sur lequel on a mené notre expérimentation contient trois matrices :

- une matrice `data_train` 80000×5 représentant les données d'apprentissage
- un vecteur d'éléments binaires `data_output` 80000×1 représentant la classification (0 = inutile | 1 = utile) des commentaires de `data_train`
- une matrice `data_test` 36000×5 représentant les données de test.

Les 5 colonnes des matrices sont organisées de la façon suivante :

- `'ID'` : identifiant du commentaire

	ID	review_content	review_title	review_stars	product
0	1	Ca serait plutot un film tres bien si ils ont ...	les bons acteurs avec un citron	4	e51597abac17b408ffc23010522f89c2e6c4ac0c8ac9cf...
1	2	Un univers unique, une mélodie immortelle, une...	Un univers unique !!	5	6187ea4f95c43a913061506bd165f13a24772a2fd758ec...
2	3	Je pense que compte-tenu de la rareté de ce fi...	C'est donné...	5	5494110a6f34bff078cd5facc4ee1852633135d8741ecf...
3	4	un film super top, une excellente surprise, dr...	garde a vous	5	5494110a6f34bff078cd5facc4ee1852633135d8741ecf...

Figure 1: les 4 premières lignes de **data_train**

- `'review_content'` : commentaire de l'utilisateur
- `'review_title'` : titre du commentaire
- `'stars'` : une note de 1 à 5 sur le produit
- `'product'` : identifiant du produit sur lequel le commentaire est attaché.

Dans la résolution de notre problème, il s'agit de travailler sur du texte et on s'est appuyé sur plusieurs références : Feature Engeeninering [**FE**] qui a été crucial avec l'application de la librairie NLTK [**nltk**] et la représentation Bag of Words, les méthodes de classification Machine Learning ([**skl**], [**mm**]), et enfin l'exploration des techniques Deep Learning ([**keras**], [**dl**], [**dl2**]) utilisant la représentation Words Embedding (librairie Gensim [**gensim**]) où des algorithmes adaptés aux textes ont été conduits : Perceptron, CNN [**CN**], [**DCNN**] et LSTM [**lstm**].

2 Text Preprocessing

2.1 À la recherche des bons tokens

Pour nettoyer le texte, on a d'abord appliqué le **stemming** en retirant les mots trop récurrents pour apporter de l'information et ensuite la **tokenization** : chaque mot a été tronqué selon sa racine. De cette façon, un mot écrit au pluriel et au singulier, ou un adjectif au masculin et au féminin, sera considéré comme identique. La librairie **NTLK** est utilisée pour parvenir à cette fin.

Étant donné qu'il faut juger si un commentaire est utile ou non, on a considéré que certains éléments textuels pouvaient fortement influencer ce jugement comme le respect de la grammaire ou l'excès de caractères spéciaux.

Tout d'abord, on regroupe tous les nombres, tous les chiffres en un token '100'. De cette façon, on regarde si la précision qu'apporte les chiffres a un impact sur l'appréciation du commentaire.

Puisqu'il n'y a plus de chiffres, hormis '100' dans tous les documents, on va ajouter de l'information dans les textes avec l'introduction d'autres chiffres, qui auront leur propre signification. Le souci de la tokenization, c'est qu'après son passage, un mot mal écrit (que ce soit faute d'orthographe ou intrusion de chiffres et autres symboles) et un mot bien écrit ne sont plus distinguables.

```
#commentaire initial
X_train[0]
```

```
'beau cuir, épais et souple - trés bel esthétique - bon rapport qualité
prix - grande contenance( 1 trés grande poche centrale ) + 1 pochette fe
rmée par fermeture éclair. attache métal.doublure intérieur médiocre'
```

```
#commentaire après preprocessing
ZOU_train[0]
```

```
'cuir 3 épais soupl 23 tré bel esthet 23 bon rapport qualit prix 23 gran
d conten 16 100 tré grand poch central 14 19 100 pochet ferm fermetur éc
lair 1 attach métal 1 doublur intérieur médiocr'
```

Figure 2: Exemple de preprocessing d'un commentaire de **data_train**

Dans notre cas, on perd de l'information. Avant tokenization, s'il y a une erreur dans un mot, on cherche donc à laisser une trace dans le texte en ajoutant des string chiffrés (par exemple, '!' est remplacé par '88').

Ainsi, on remplace tous les caractères spéciaux et toute la ponctuation par des chiffres isolés pour les mettre en valeur.

Différentes approches ont été expérimentées en incluant les éléments de `'review_title'` dans `'review_content'`, comme l'ajout d'une phrase à part entière, ou en traitant le titre séparément. Nos meilleurs résultats ont été obtenus en utilisant la première méthode.

Enfin, la colonne `'product'` n'a pas été utilisée car aucun produit de `data_train` ne se trouve dans `data_test`. L'identifiant du produit est trop compliqué et aléatoire pour en retirer quoique ce soit (par exemple, on aurait pu regrouper les produits électroniques ensembles si les quatre premiers chiffres de `'product'` permettaient d'identifier cette catégorie d'objets).

2.2 Clustering sur les documents

Dans le natural language processing, **Latent Dirichlet allocation (LDA)** est un modèle statistique génératif non supervisé qui permet à des ensembles d'observations d'expliquer pourquoi certaines parties des données sont similaires. C'est un clustering sur du texte pour espérer isoler des thématiques ou bien des groupes de mots plus influents que d'autres. On l'a fait aussi sur ce qu'on obtenait après Feature Extraction (expliquée après) mais rien d'utilisable n'en est sorti. On a interprété ce résultat et déduit que les thèmes associés aux commentaires avaient peu d'impact sur la classification.

3 Feature Extraction

3.1 Bag of Words

L'algorithme ne peut traiter que des représentations numériques des données et l'on doit par conséquent transformer le texte brut. Pour cela, on a eu recours à

la représentation "**Bag of Words**" pour extraire des features numériques :

- On tokenize les mots et on associe un identifiant integer à chaque token obtenu.
- On compte l'occurence des tokens dans chaque document.
- On normalise et on diminue l'importance des tokens peu représentés dans l'ensemble des documents.

Chaque nouvelle feature donnée représente la fréquence d'un token donné dans un commentaire. On obtient alors une nouvelle matrice dont les lignes sont les commentaires et les colonnes ces nouvelles features.

On a utilisé: `HashingVectorizer`, `CountVectorizer` et `TfidfVectorizer`.

`HashingVectorizer` ne permet pas de manipulation de features. On a donc cherché à privilégier les deux autre méthodes. Empiriquement, `CountVectorizer` nous a fourni les meilleurs résultats (à 1 % près) avec `ngramrange=(1,3)`, `maxdf=0.8`, `mindf=5`, suivi d'une normalisation avec la norme ℓ_2, et on l'a donc conservé pour la suite. Ici, `CountVectorizer` nous fournit environ 40000 features.

On a essayé d'ajouter au Bag of Words, le Bag of letters, mais on n'a pas su utiliser ces informations dû à des problèmes de mémoire.

3.2 Word Embedding

Une autre représentation du texte est possible : **Word Embedding**. Chaque token est projeté dans un espace de $\mathbf{R}^d$ dont la dimension d est un hyperparamètre. Soit on utilise un modèle pré-entraîné, disponible sur Internet, soit on entraîne soi-même le modèle avec nos tokens. Dans tous les cas, la librairie **Gensim** a été utilisée.

Pour les modèles pré-entraînés, le premier souci a été de trouver des modèles qui traitent les mots français. Le second, c'est que ces modèles ont appris à projeter dans l'espace des mots bien écrits ou des mots tronqués différemment des nôtres. Nous, on souhaite conserver les mots mal écrits et certains ne trouvaient pas leur image dans l'espace entraîné.

Les modèles pré-entraînés ont été pris sur la page web de Jean-Philippe Fauconnier [**JPF**], dont un avec 1.6 milliards de mots. Outre les problèmes de mémoire pour importer ces modèles, il fallait aussi conserver les mots représentés dans des espaces de dimension 30, ce qui est une modélisation plus coûteuse en mémoire. Cela a été très problématique pour le Deep Learning approfondi ultérieurement.

Pour les modèles que l'on a conçus, on n'a pas réussi à obtenir des modèles satisfaisants dans le cadre de l'apprentissage Deep Learning (étayé plus tard).

4 Feature Selection

Seul CLiPS, propose une librairie, nommé pattern.fr, permettant d'identifier la subjectivité et la positivité de phrases écites en français. Nous en avons profité

pour ajouter ces deux features dans les différents modèles, mais les résultats n'en ont pas été améliorés. Après l'obtention d'un jeu de données préparé, plusieurs techniques pour réduire le bruit ont été introduites :

- `SelectFromModel` : on a introduit le `linearSVC` avec la norme ℓ_1 pour réduire la dimension en contrôlant le paramètre C de régularisation.
- `SelectKBest` : ne conserve que les K features les plus performantes dans la classification. On utilise le χ^2 qui réalise un test statistique sur des données qui ne doivent contenir que des features positives. Ici, ce sont des fréquences donc c'est respecté. Cette méthode est bien adaptée pour les matrices creuses, ce qui est le cas ici.

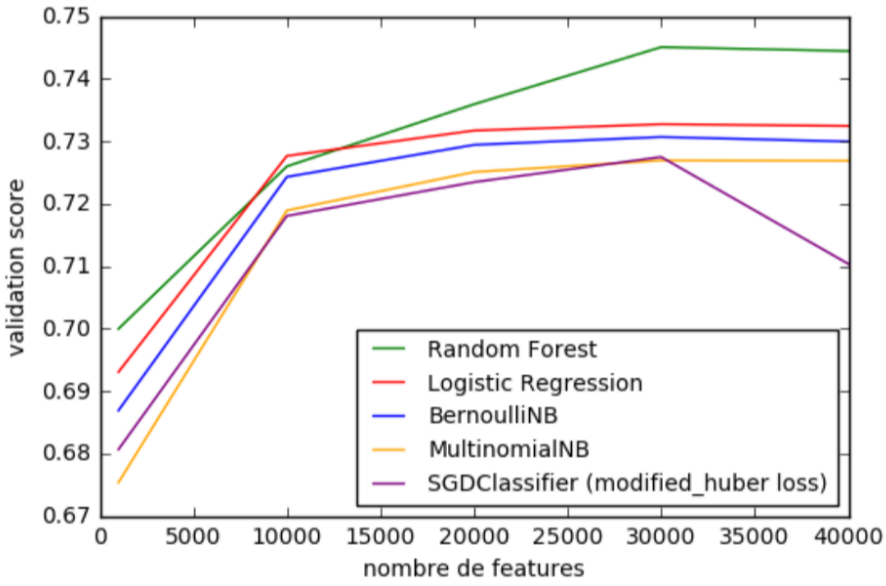
  On a testé aussi le `mutual_info_classif` au lieu du χ^2 mais il donne de moins bons résultats.
- `SelectKBest` suivi de `SelectFromModel` : Grossièrement, on retire $\frac{1}{4}$ des features existantes avec `SelectKBest` puis on réalise une autre dimension de réduction sur ces features utiles.
- **Recursive feature elimination** : cette méthode cherche à sélectionner des features récursivement en considérant des sous-ensembles de features de plus en plus petits. C'est un peu comme la méthode précédente mais généralisée.

La feature selection qui a donné les meilleurs résultats de classification a été obtenue avec `SelectKBest` où K représente environ $\frac{3}{4}$ du nombre de features initial. La figure 3 souligne malgré tout que `CountVectorizer` supprime déjà de nombreux éléments parasites et que le bruit éliminé par la Feature Selection est peu significatif.

5 Classification

On a utilisé la fonction `train_test_split()` avec 30 % de données utilisés pour le validation dataset. Sur ces jeux de données, on a appliqué toutes sortes de classifieurs et on va en illustrer que quelques uns :

- **Naive Bayes** utilise bien moins de puissance de calculs que les autres méthodes. Donc on s'en est servi comme méthode de référence ; on obtenait 72 % sur le validation dataset.
- Le **Support Vector Machine** est connu pour être une méthode efficace pour la classification de texte. Le linearSVC nous a donné des résultats moyens (69 %) et il nous a été impossible d'utiliser des kernels différents car le temps de calculs explosait.
- Même si apprendre un optimal decision tree est un problème NP-difficile, on a tenté **Random Forest Classifier (RFC)**. Nous n'avons pas eu le temps de bien ajuster les paramètres et RFC overfit totalement le training set comme l'illustre la figure 4. Néanmoins, c'est avec RFC que nous avons obtenu les meilleurs résultats sur le validation dataset (74.8 %) et sur le challenge data (71.5 %).



6 Deep Learning

Un traitement des données particulier est requis pour entraîner un réseau de neurones via la librarie Keras. La représentation **Word Embedding** est de mise et il faut que les données fassent toutes la même longueur avant passage dans la première couche Embedding du réseau de neurones qui réalise la projection.

On récupère les 5000 features les plus importantes et chaque token est remplacé par un integer. Puisque les commentaires comprennent un nombre de tokens variables, on tronque tout à 50 tokens par commentaire, quitte à remplir avec des 0 s'il en manque.

Plusieurs algorithmes ont été proposés en lien avec la classification de textes: **Multilayer Perceptron**, **Convolutionnal Neural Network** et **Long-Short Term Memory**.

Pour éviter l'overfitting, une fonction `callbacks()` a été introduite ; on s'arrête environ après 5 epochs où l'on obtient 70 % d'accuracy sur le training set et 66 % sur le validation set. L'overfit a donc lieu très rapidement, et ce, peu importe les méthodes utilisées. Quand on essaie d'augmenter la dimension de l'espace sur lequel on projette pour que le réseau de neurones saisissent davantage d'informations, la mémoire explose. On a vite été limité dans nos manipulations.

La profondeur, le nombre de features gardés, les optimizers, le nombre de couches, tous les hyperparamètres ont été testés mais sans succès remarquable.

Nous avons testé plusieurs configurations de réseaux de neurones, allant même jusqu'à ajouter les caractères subjectif/objectif et positif/négatif grâce à la fonction `sentiment` de `pattern.fr` en parallèle de la dernière couche convolutionnelle de neurones, mais les résultats n'ont jamais été époustouflants.

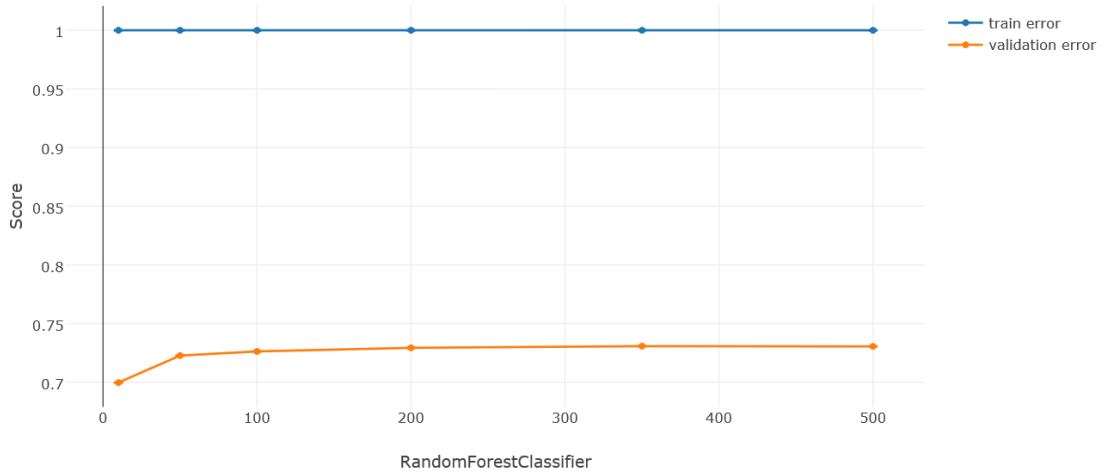


Figure 4: Scores obtenus avec Random Forest en fonction du nombre d'estimateurs. On avait $max features = log_2(nombre\ de\ features)$ comme hyperparamètre ajusté. Gros souci d'overfitting non résolu.

6.1 Crawling de twitter

Nous pensions que les mauvaises performances de nos réseaux de neurones sur le dataset de Price Minister étaient liées à la petite taille de celui-ci, nous avons donc décider d'augmenter le dataset en "crawlant" twitter.

6.1.1 Labeliser les tweets ?

Un tweet comporte un nombre important d'attributs concernant soit l'utilisateur soit le tweet lui-même[1]. Nous avons pensé à labeliser les tweets suivant des attributs tels que `favorite_count`[2] ou `retweet_count`[3].

Ces informations étant fortement biaisées par essence, il nous fallait les associer aux attributs de l'utilisateur ayant retweeté et à ceux de l'utilisateur à l'origine du tweet. Pour cela, on peut utiliser l'attribut `retweeted_status`[4]

Nous avons donc dû abandonner l'idée de labeliser les tweets.

6.1.2 Utiliser twitter de façon non supervisée

L'intérêt de twitter est que l'on trouve le même type de phrase que dans les commentaires des clients de PriceMinister. Il était donc envisageable d'entraîner notre première couche d'embedding de façon non supervisé sur des tweets pour ensuite fixer les poids et entraîner de façon supervisé les couches suivantes sur le dataset de PriceMinister. Nous avons pu récolter un peu plus de 333000 tweets

[1] https://dev.twitter.com/overview/api/users, https://dev.twitter.com/overview/api/tweets
[2] Indicates approximately how many times this Tweet has been liked by Twitter users.
[3] Number of times this Tweet has been retweeted.
[4] This attribute contains a representation of the original Tweet that was retweeted. Note that retweets of retweets do not show representations of the intermediary retweet, but only the original Tweet.

différents à partir d'une recherche par pair de mot-clé sur 100 mots-clés tirés du dataset de PriceMinister.

Malheureusement, cela n'a pas augmenté l'accuracy de façon significative.

6.2 Correcteur orthographique

Dans la section Text Preprocessing, nous sommes convaincus qu'il y a encore de quoi améliorer l'information et nous savons comment. Un bémol de notre approche est qu'elle ne repère pas en revanche les fautes d'orthographes. On a voulu utiliser la librairie **PyEnchant** qui contient un correcteur orthographique. On a fait face à toutes sortes de problèmes informatiques. On aurait pu créer notre propre fonction, avec importation d'un dictionnaire, etc, mais le temps nous manquant, on n'a pas su résoudre ce problème.

Avec les étapes illustrées de notre text preprocessing, on a amélioré la classification de 67 % d'accuracy à 71 % d'accuracy. On pense donc que l'information supplémentaire que pourrait apporter le correcteur orthographique est significative. On peut espérer gagner un ou deux points d'accuracy au moins avec son traitement.