Movie Reviews

Rapport

Valentin Fontanger

&

Julie Lascar

Master 1 DAC, Traitement automatique de la langue

28/02/2022

1. Introduction

L'objet de ce projet est d'utiliser le Machine Learning afin de détecter les sentiments d'un document. Nous avons utilisé le jeu de données ***movies1000,*** comportant 4000 avis d'utilisateurs sur des films. L'objectif principal est d'évaluer plusieurs modèles de classification, reposant sur la méthodologie ***BagOfWord (BOW).*** Les BagOfWords (sac de mots en français), permettent de constituer une matrice dont les lignes sont les documents et les colonnes les différents mots du vocabulaire. Pour chaque vecteur ligne, les coefficients traduisent une information sur la présence d'un mot dans le document. Nous utiliserons d'une part le nombre d'occurrences du mot dans le document, puis le tfidf du mot pour ce document.

2. Problématique et Algorithmes

Nous souhaitions trouver les ***meilleurs hyperparamètres*** afin de construire un BOW permettant d'obtenir une ***accuracy*** convenable pour notre problème de classification de documents. Dès ces hyperparamètres trouvés, nous avons entraîné un modèle de classification, pour lequel nous avons déterminé les meilleurs hyperparamètres. Voici les algorithmes entraînés : Regression Logistique, Linear Support Vector Machine, Naïve Bayes, Perceptron et Réseau de neurones. Ces algorithmes ont fait leurs preuves pour de nombreux problèmes de classifications.

3. Expériences et évaluations

3.1. Méthodologie

Notre approche a été de déterminer, dans un premier temps, quels étaient les éventuels challenges que ce dataset pouvait présenter. Nous avons trouvé un ***dataset parfaitement équilibré*** pour les deux classes.

Distribution des classes

Dès lors, il est bon de se poser la question suivante : l'une des deux classes pose-t-elle plus de difficultés que l'autre ? Pour le savoir, nous avons naïvement évalué les modèles de classifications, et avons regardé leur ***accuracy en validation-croisée*** pour un BOW TFIDF paramétré de manière naïve (max_feature = 50000, min_df = 0 , max_df = 1.0). La quantité TFIDF est obtenue de la manière suivante :

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

t : un terme, d : un document, D : le corpus, N : la taille du corpus

La quantité TFIDF est le produit du TF et de l'IDF.

Une étude de chaque modèle a été effectuée afin d'identifier les meilleurs hyperparamètres des BOWs et des modèles. Au cours de ce travail, nous avons prêté une attention particulière à l'analyse des résultats obtenus en étudiant l'évolution de l'accuracy selon les hyperparamètres.

3.2 Résultats

L'évaluation naïve des classifieurs (aucune optimisation) par validation croisée nous a dirigé vers le modèle ***Linear Support Vector Machine (LinearSVC)***, pour lequel nous obtenons 0.85 en accuracy. Il est important de préciser qu'à ce stade nous faisions l'hypothèse qu'aucune des deux classes ne soit plus complexe à identifier que l'autre. Afin de vérifier ce second point, nous avons regardé des mesures telles que la ***précision*** et le ***rappel*** pour chaque classe.

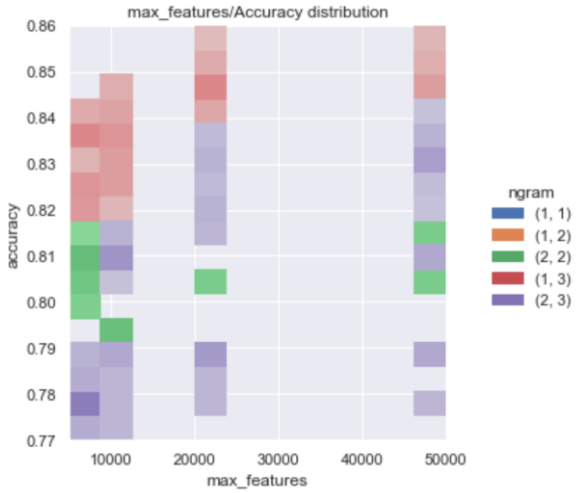
Précision : La précision est intuitivement la capacité du classifieur à ne pas prédire comme positif un échantillon négatif

Rappel : Le rappel est intuitivement la capacité du classifieur à prédire correctement les échantillons positifs

LinearSVC :. Voici une rapide visualisation de nos résultats triés dans l'ordre décroissant, nous permettant de déterminer les paramètres qui marchent le mieux, mais également de vérifier l'hypothèse sur la difficulté de prédictions des classes.

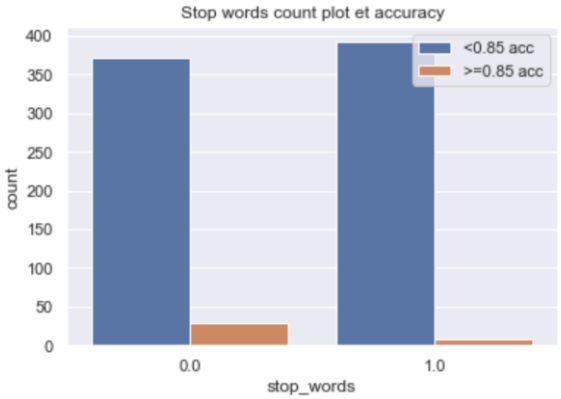
f1_pos	f1_neg	prec_pos	prec_neg	rec_pos	rec_neg	accuracy	stop_words	max_features	min_df	max_df	ngram
0.860000	0.860000	0.851485	0.868687	0.868687	0.868687	0.8600	0.0	50000.0	1.0	0.60	(1, 2)
0.860000	0.860000	0.851485	0.868687	0.868687	0.868687	0.8600	0.0	50000.0	1.0	0.55	(1, 2)
0.859259	0.855696	0.840580	0.875648	0.878788	0.875648	0.8575	0.0	50000.0	5.0	0.60	(1, 2)
0.859951	0.854962	0.837321	0.879581	0.883838	0.879581	0.8575	0.0	50000.0	4.0	0.60	(1, 2)
0.857143	0.857855	0.850746	0.864322	0.863636	0.864322	0.8575	0.0	50000.0	2.0	0.60	(1, 2)

On observe que max_features = 50000 maximise cette accuracy. On observe que le ngram = (1,2) propose, pour chaque valeur de max_features, une accuracy supérieure aux restes des nrams.



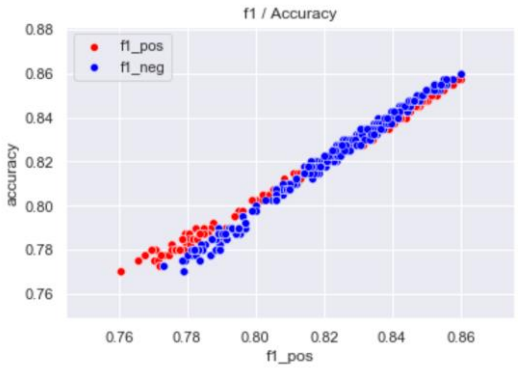
(La couleur rose couvrant les accuracy > 0.8 désigne le ngram (1,2) et non (2,3)

On fait également le choix de ne pas fournir de stop words :



Stop words par rapport à l'accuracy obtenue

De plus, nous observons que l'accuracy est une bonne mesure en observant la corrélation positive entre les f1 de chaque classe et l'accuracy. Aucune classe n'est plus difficile à prédire que l'autre :



Finalement, on retient les hyperparamètres suivants pour une accuracy en validation de ***0.86*** pour le TFIDFVectorizer :

max_features =50000, min_df=1, max_df=0.60, ngram=(1,2).

Nous avons cherché à optimiser notre modèle en déterminant ses meilleurs hyperparamètres à l'aide d'un ***GridSearchCv***, permettant d'optimiser un modèle en analysant le score en validation croisée de ce dernier. Après optimisation, nous obtenons ***0.8724*** en accuracy sur le jeu de validation.

Nous terminons les expériences pour ce modèle en obtenant une ***accuracy en test*** *de* ***0.84 :***

	precision	recall	f1-score	support
0	0.83	0.84	0.84	199
1	0.84	0.84	0.84	201
accuracy			0.84	400
macro avg	0.84	0.84	0.84	400
weighted avg	0.84	0.84	0.84	400

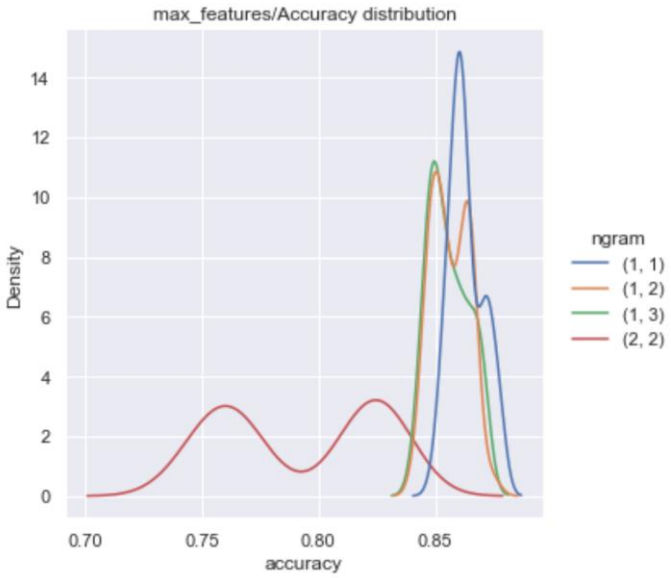
Evaluation sur les données de tests

<u>Régression Logistique :</u> Nous gardons la même méthodologie en optimisant les hyperparamètres du ***TFIDFVectorizer***, puis du modèle. Voici les résultats obtenus :

f1_pos	f1_neg	prec_pos	prec_neg	rec_pos	rec_neg	accuracy	stop_words	max_features	min_df	max_df	ngram
0.874036	0.880779	0.890052	0.866029	0.858586	0.866029	0.8775	0.0	20000.0	3.0	0.60	(1, 1)
0.873385	0.881356	0.894180	0.862559	0.853535	0.862559	0.8775	0.0	50000.0	4.0	0.60	(1, 1)
0.873385	0.881356	0.894180	0.862559	0.853535	0.862559	0.8775	0.0	20000.0	4.0	0.60	(1, 1)
0.874036	0.880779	0.890052	0.866029	0.858586	0.866029	0.8775	0.0	50000.0	3.0	0.60	(1, 1)
0.871134	0.878641	0.889474	0.861905	0.853535	0.861905	0.8750	0.0	20000.0	4.0	0.55	(1, 1)

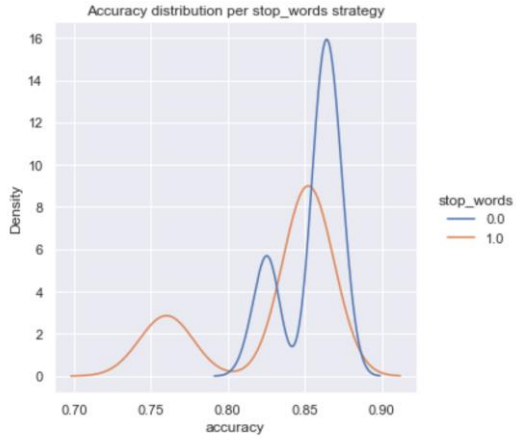
On observe une accuracy supérieure pour ce modèle (en validation)

Distribution de l'accuracy par rapport au ngram :



On se concentre sur ngram = (1,1) ou (1,2)

A nouveau, l'élimination des stop words ne permet pas de maximiser l'accuracy :



Distribution de l'accuracy selon l'utilisation ou non des stop words

On retient les paramètres suivants :

Max_features = 20000, min_df=3, max_df = 0.60, ngram = (1,1) pour une accuracy de ***0.877*** en validation. Après un gridsearch, nous obtenons une accuracy en validation de : ***0.87.***

Finalement, on obtient de bien meilleures performances sur les données de test qu'avec notre modèle linearSVC : ***0.86***

0	0.88	0.82	0.85	196
1	0.84	0.90	0.87	204
accuracy			0.86	400
macro avg	0.86	0.86	0.86	400
weighted avg	0.86	0.86	0.86	400

Performances sur les données de tests.

Réseau de neurones : Pour ce modèle, nous utiliserons la librairie ***Keras***. Le choix d'un réseau de neurones peut sembler audacieux au vu du faible nombre de données dont nous disposions. Cependant, nous comptions sur une ***régularisation intensive*** pour combattre le sur-apprentissage.

Nous avons appliqué la méthode du ***dropout***, consistant à désactiver certains neurones selon une probabilité p pour une couche, afin de mieux répartir les poids sur l'ensemble des neurones de la couche, et éviter que des neurones soient responsables de la prédiction.

Nous avons également ajouté une pénalité L2 se formulant comme suit :

$$L(x,y) \equiv \sum_{i=1}^{n} (y_i - h_\theta(x_i))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

Pénalité L2 ajoutée à la fonction de coût

Cette pénalité permet de faire converger les poids vers 0, et donc d'éviter le sur apprentissage en obtenant une architecture « moins complexe ».

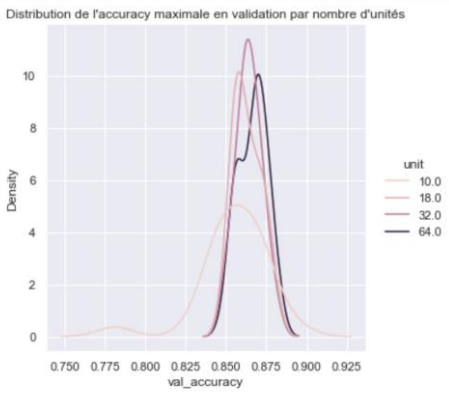
Le réseau consiste d'une couche cachée (activation « relu ») avec dropout puis d'une couche finale de 1 neurone (activation « sigmoid ») pour obtenir la prédiction.

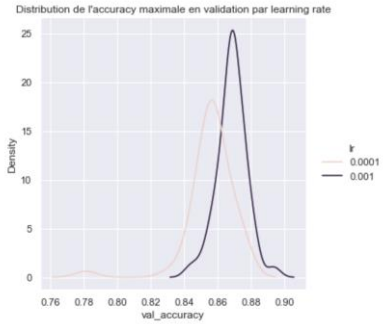
Nous avons effectué deux expériences concernant les réseaux de neurones. Une avec un ***TFIDFVectorizer***, l'autre avec un ***CountVectorizer***. En utilisant le TFIDF, nous obtenons les résultats suivants :

lr	epoch	batch_size	accuracy	val_accuracy	loss	val_loss	max_features	min_df	max_df	ngram	stop_words
0.001	40.0	16.0	1.0	0.89375	0.671410	0.638902	25000.0	1.0	0.60	(1, 2)	0.0
0.001	40.0	16.0	1.0	0.88125	0.635764	0.550543	25000.0	5.0	0.60	(1, 2)	0.0
0.001	40.0	16.0	1.0	0.88125	0.656722	0.602466	25000.0	2.0	0.55	(1, 2)	0.0
0.001	40.0	16.0	1.0	0.88125	0.646582	0.576410	25000.0	2.0	0.55	(1, 2)	0.0
0.001	40.0	16.0	1.0	0.88125	0.666783	0.623749	25000.0	2.0	0.55	(1, 2)	0.0

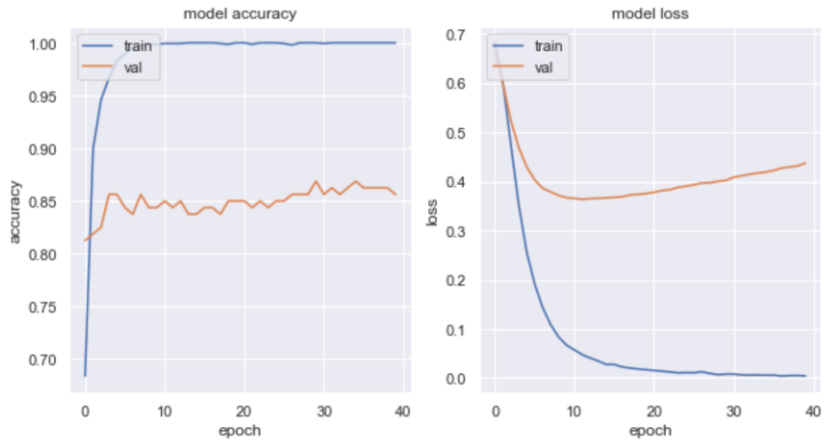
Meilleurs résultats tfidf-réseau.

On observe la distribution de l'accuracy selon le nombre de neurones dans la couche profonde, puis selon le learning rate :





Il est important de noter qu'un phénomène de sur-apprentissage a été observé. Voici un exemple, mais la totalité des entraînements se sont déroulés de façons similaires :



Sur-apprentissage, le coût en validation remonte/ne descend plus après 10 epochs

Finalement, on retient la configuration suivante : learning_rate =0.001, epoch=40, batch_size=16, neurones=10, max_features=17000, min_df=1, max_df=0.6, ngram=(1,8), et on obtient une accuracy en validation de 0.85 avec une stratégie ***d'early stopping***, autrement dit, d'arrêt de l'entraînement lorsque l'accuracy en validation n'augmente plus ou stagne.

On obtient finalement sur les données de test une accuracy de ***0.87 :***

	precision	recall	f1-score	support
0	0.88	0.88	0.88	214
1	0.86	0.87	0.86	186
accuracy			0.87	400
macro avg	0.87	0.87	0.87	400
weighted avg	0.87	0.87	0.87	400

Pour notre dernière expérience, nous avons utilisé un CountVectorizer et nous obtenons en test :

	precision	recall	f1-score	support
0	0.87	0.90	0.89	203
1	0.89	0.86	0.88	197
accuracy			0.88	400
macro avg	0.88	0.88	0.88	400
weighted avg	0.88	0.88	0.88	400

Nous obtenons les meilleures performances sur cette dernière expérience.

3.2 Discussion

Afin d'avancer dans nos expérimentations, nous avons dû contraindre le nombre d'hyperparamètres à tester pour les TFIDFVectorizer et CountVectorizer. Nous aurions souhaité ajouter davantage de paramètres afin d'affiner nos recherches, et également tester la version CountVectorizer sur tous les modèles (ce que nous n'avons pas fait par manque de temps).

Cependant, nous sommes satisfaits des résultats obtenus, servant de ligne directrice pour de futures recherches de paramètres.