Chirac ou Mitterrand

Review des movies

I. Chirac ou Mitterrand

1. Introduction

Cette base est composée de 54413 expressions et phrases tirées de différents discours des anciens présidents français Chirac et Mitterand, chaque expression (document) est étiquetée par un label C ou M (pour Chirac ou Mitterrand) et qui indique le président ayant prononcé cette expression.

Il est à noter que les classes ne sont pas équilibrées, on note une forte présence des expressions du président Chirac que celles du président Mitterrand.

L'objectif de ce projet est d'apprendre un classifieur binaire qui permet de prédire pour une expression donnée (Nouvelle expression provenant de l'un des deux présidents), si elle a été prononcée par le président Chirac ou Mitterrand. Le modèle proposé sera ensuite appliqué sur un nouveau test-set auquel on n'a pas accès durant la phase d'apprentissage.

2. Analyse préliminaire de la base d'apprentissage:

Nous commençons notre étude par une analyse préliminaire de notre base d'apprentissage.

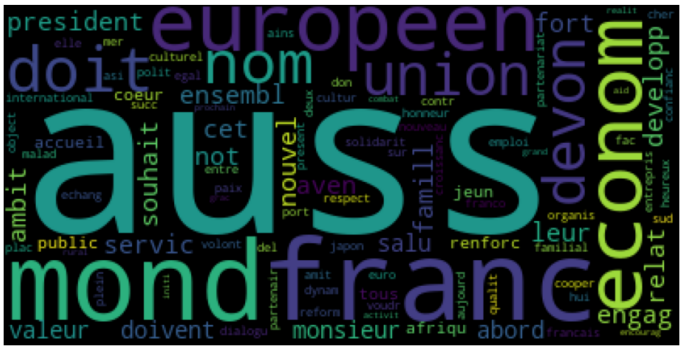
Il est tout d'abord à noter que la base de données est extrêmement déséquilibrée, avec 86,9 % des données étiquetées sous la classe 'C' pour le président Chirac, contre seulement 13,1 % pour Mitterrand (soit respectivement 49 890 et 7 523 phrases). Nous craignons que cette disparité pousse notre classifieur à prédire exclusivement la classe majoritaire 'C', ce qui nécessitera soit d'équilibrer la base de données, soit de modifier notre modèle afin de prendre en compte ce déséquilibre.

Un wordcloud sur les phrases propres à Chirac et à Mitterrand ne nous permettent pas de différencier à vue d'œil les discours des deux présidents, même en supprimant les stopwords.
Voici le wordcloud pour le président Chirac à gauche, pour Mitterrand à droite :

Chirac

Mitterrand

Nous nous proposons donc de refaire un wordcloud en prenant cette fois-ci en compte les odds ratios afin de déterminer les mots les plus discriminants pour chaque classe.
Voici le wordcloud pour le président Chirac à gauche, pour Mitterrand à droite :



Chirac

Mitterand

Nous observons que Chirac a tendance à utiliser fréquemment des mots tels que 'France', 'économie', 'social', 'engagement' et 'développement', ainsi que leurs dérivés, tandis que Mitterrand fait davantage référence à des termes comme 'guerre', 'Europe' et 'industrialisation'. De plus, il est intéressant de noter que Chirac utilise des mots qui encouragent la confiance et la cohésion, tels que 'solidarité', 'respect', 'valeur', 'coeur', 'amitié' et 'ensemble', qui se révèlent être des critères de discrimination importants pour lui, contrairement à Mitterrand qui ne présente pas cette tendance.

3. Pré-processing de la base d'apprentissage:

Il est crucial de procéder à une phase de prétraitement pour réduire la dimensionnalité de notre base de données avant de la transformer en liste de vecteurs sur le vocabulaire.

Les pré-traitements que nous considérons sont notamment :

- Les pré-traitements que nous considérons sont notamment :
- La mise en minuscule
- La normalisation afin de supprimer les accents et les caractères non normalisés
- La suppression des chiffres et de la ponctuation
- Le stemming
- L'élimination des stopwords

Ces pré-traitements s'effectuent par paramétrage (True ou False) d'une fonction dédiée au processing du corpus. Il s'agira pour nous de retrouver par GridSearch la/les combinaisons optimisant la métrique choisie sur la base d'apprentissage en validation croisée.
Suivant le modèle de machine learning choisi, le paramètre de régularisation et le type de représentation des phrases seront aussi des paramètres à optimiser, nous devons déjà faire une pré-sélection sur ces pré-traitement afin de réduire le nombre de combinaisons à tester. Nous choisissons de fixer la suppression des chiffres et de la ponctuation à True : les discours ont été prononcés mais dans la base, les phrases ont été écrites de manière neutre (la ponctuation n'apporte donc pas d'information supplémentaire).

4. Modèles de Machine Learning:

Nous choisissons de tester trois modèles de machine learning :

- Un SVM linéaire (LinearSVC),
- Un modèle Naive Bayes (MultinomialNB)
- Une régression logistique (LogisticRegression). En particulier, Sklearn propose également un paramètre de régularisation C pour les modèles de régression linéaire et SVM que nous nous attacherons à optimiser.

5. Sélection de modèles:

De manière générale, les scores calculés pour un modèle de sklearn sont des taux de bonne classification (accuracy) par défaut. Toutefois, cette métrique ne convient pas à notre situation car un classifieur qui prédit uniquement la classe majoritaire (qui représente plus de 80% de nos données) aura une accuracy supérieure à 0.8. Par conséquent, nous avons décidé d'optimiser d'autres scores (en particulier F1-Score et Roc-Auc).

6. Schéma expérimental:

1. **Première campagne d'expériences:** Nous effectuons en premier lieu grid search sur les pré-traitements ainsi que sur les modèle de machine learning décrits plus hauts. Le type de représentation des phrases (vectorisation par comptage ou par tf-idf) en plus des types de n-grams sont également des hyperparamètres à prendre en compte.
2. **Deuxième campagne d'expériences:**Comme nous l'avons précisé plus haut, les données sont fortement déséquilibrées et la majorité d'entre elles appartient à la classe Chirac. Nous risquons ainsi d'apprendre un classifieur ne prédisant presque que la classe majoritaire :
    a. Sur-échantillonner la classe minoritaire, sous-échantillonner la classe minoritaire, ou faire un mélange des deux. Nos expériences n'ont pas été très concluantes, les résultats obtenus sont moins bons que sur la base d'origine.
    b. Changer la fonction de coût pour pénaliser plus fortement les mauvaises prédictions sur la classe minoritaire. Le module sklearn propose pour cela pour chacun de ses modèles un paramètre ***class_weight*** nous permettant d'attribuer un ratio de pénalité sur les différentes classes.
3. **Troisième campagne d'expériences:** Utiliser les meilleurs modèles appris précédemment avec une technique d'a priori.

7. Campagne d'experiences:

Première campagne d'expériences:

Nous commençons par identifier les paramètres qui produisent les meilleures performances, mais plutôt que de simplement choisir la combinaison qui donne le score le plus élevé, nous examinons les 20 meilleures combinaisons pour en tirer une agrégation qui semble optimale. Par exemple, bien que les meilleures combinaisons aient le paramètre stemming à True et no_stopwords à False, un grand nombre de combinaisons avec des scores similaires (environ 0,8-0,9) ont le paramètre stemming à False et no_stopwords à True. Nous choisissons cette dernière combinaison pour réduire la dimensionnalité de nos matrices (en supprimant les stopwords).

```
scores_df = load_scores_df(params_file, scores_files, scores_names)
sorted_by_f1 = scores_df.sort_values(by='F1 score', ascending=False)
sorted_by_f1[:20]
```

[55] Python

	Language	Line	Lower	Remove digit	Remove punctuation	Remove stopwords	Normalize	Stemming	N-gram range	Vectorizer	Model	C	Accuracy score	F1 score	ROC-AUC score
106	french	None	False	True	True	False	False	True	(1, 2)	Tf-Idf	Linear SVC	10	0.861946	0.918055	0.876216
108	french	None	False	True	True	False	False	True	(1, 2)	Tf-Idf	Linear SVC	30	0.861660	0.917790	0.875132
107	french	None	False	True	True	False	False	True	(1, 2)	Tf-Idf	Linear SVC	20	0.860801	0.917434	0.872758
197	french	None	False	True	True	False	True	False	(1, 2)	Tf-Idf	Linear SVC	20	0.859657	0.917151	0.876559
17	french	None	False	True	True	False	False	False	(1, 2)	Tf-Idf	Linear SVC	20	0.859371	0.917069	0.871855
198	french	None	False	True	True	False	True	False	(1, 2)	Tf-Idf	Linear SVC	30	0.859371	0.916956	0.875275
109	french	None	False	True	True	False	False	True	(1, 2)	Tf-Idf	Linear SVC	40	0.859943	0.916922	0.875264
199	french	None	False	True	True	False	True	False	(1, 2)	Tf-Idf	Linear SVC	40	0.858226	0.916298	0.875760
18	french	None	False	True	True	False	False	False	(1, 2)	Tf-Idf	Linear SVC	30	0.857797	0.916024	0.875756
168	french	None	False	True	True	False	False	True	(1, 2)	Tf-Idf	Logistic Regression	30	0.856080	0.916021	0.878578
16	french	None	False	True	True	False	False	False	(1, 2)	Tf-Idf	Linear SVC	10	0.857225	0.915848	0.871758
196	french	None	False	True	True	False	True	False	(1, 2)	Tf-Idf	Linear SVC	10	0.856795	0.915615	0.874232
66	french	None	False	True	True	False	False	False	(1, 1)	Tf-Idf	Logistic Regression	10	0.857082	0.915172	0.863699
19	french	None	False	True	True	False	False	False	(1, 2)	Tf-Idf	Linear SVC	40	0.855937	0.915060	0.872367
169	french	None	False	True	True	False	False	True	(1, 2)	Tf-Idf	Logistic Regression	40	0.854793	0.915004	0.876402
67	french	None	False	True	True	False	False	False	(1, 1)	Tf-Idf	Logistic Regression	20	0.857082	0.914713	0.860441

Meilleurs 20 combinaisons de paramètres

Nous avons ainsi décidé des paramètres suivants :

- Pré-traitements : aucune mise en minuscule, suppression des chiffres et de la ponctuation, aucune normalisation, aucun stemming, suppression des stopwords.
- Représentation vectorielle : tfidf, avec une plage de n-grammes de (1,2) pour prendre en compte les unigrammes et les bigrammes.
- Modèle : régression logistique avec C compris entre 10 et 100.

```
read_param_file("test.txt",889)
```

[51] Python

```
{'processing': {'language': 'french',
  'line': None,
  'lowercase': True,
  'no_num': True,
  'no_punc': True,
  'no_stopwords': False,
  'norm': False,
  'stemming': True},
 'vectorizer': {'ngram_range': (1, 2),
  'type': sklearn.feature_extraction.text.TfidfVectorizer},
 'model': {'m': sklearn.linear_model._logistic.LogisticRegression},
 'cross_val': {'C': 30}}
```

Combinaison des meilleurs paramètres #1

Deuxième campagne d'expériences:

Comme nous l'avons précisé plus haut, les données sont fortement déséquilibrées et la majorité d'entre elles appartiennent à la classe Chirac. Nous risquons ainsi d'apprendre un classifieur ne prédisant presque que la classe majoritaire.

Nous lançons une deuxième campagne d'expériences afin d'essayer de balancer les classes, nous allons pour cela reprendre les meilleurs modèles de la première campagne et tenir compte des poids de classes, en utilisant le paramètre ***"balanced"*** pour l'attribut ***class_weight.*** Nous remarquons que les résultats sont légèrement meilleurs.

Nous avons également utilisé des techniques de sur-échantillonnage et de sous-échantillonnage:

```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

# Over-sampling

ros = RandomOverSampler(random_state=0, sampling_strategy=1)
ros.fit(X_train_embedded, Y_train)
xtrain_resampled, ytrain_resampled = ros.fit_resample(X_train_embedded, Y_train)

M, C = np.unique( ytrain_resampled, return_counts=True)[1]
T = M + C
print('{}% Chirac, {}% Mitterrand\n\n' . format( round((C/T)*100,2), round((M/T)*100,2) ))

# Création modèle, phase d'apprentissage
clf = LinearSVC( C = 1 )
clf.fit(xtrain_resampled, ytrain_resampled)

# Prédictions sur quelques données
yhat = clf.predict(X_test_embedded)

print('Accuracy score : ', accuracy_score(Y_test, yhat))
print('F1 Score : ', f1_score(Y_test, yhat))
print('ROC AUC Score : ', roc_auc_score(Y_test, yhat))
# 50.0% Chirac, 50.0% Mitterrand
```

[55] Python

```
50.0% Chirac, 50.0% Mitterrand


Accuracy score :  0.8438215102974829
F1 Score :  0.9045788185948969
ROC AUC Score :  0.7471100913229461
```

Amélioration des résultats sur le test set avec l'échantillonnage pour l'équilibrage des données

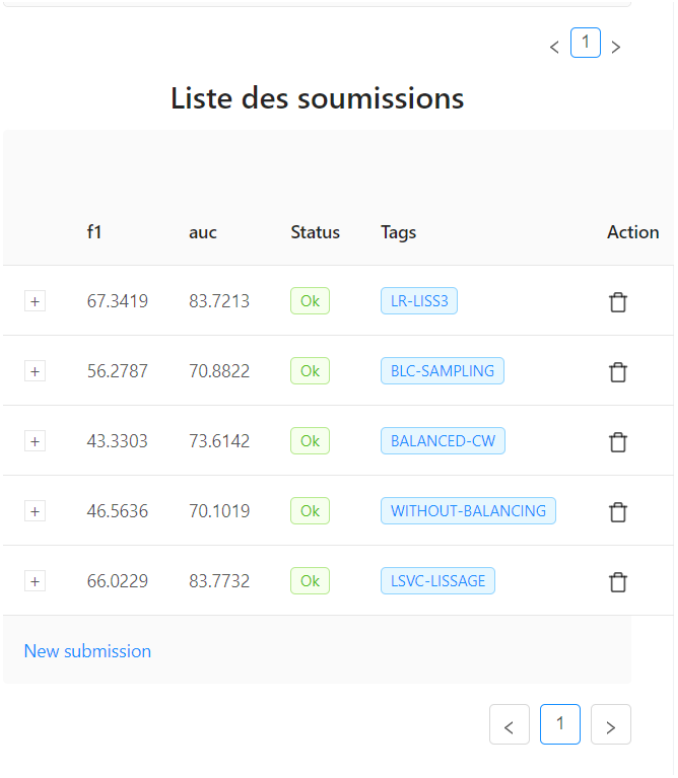
Troisième campagne d'expériences:

Nous avons remarqué précédemment que les phrases prononcées par les présidents étaient regroupées par blocs dans la base d'apprentissage. Nous supposons que les données de test suivent la même distribution et nous proposons de lisser les résultats afin d'obtenir également des blocs dans nos prédictions. Cette décision est justifiée car nos classifieurs ont tendance à prédire moins bien les phrases de la classe minoritaire, et chaque prédiction "-1" peut alors être importante si elle n'est pas isolée. Nous effectuons trois types de lissage :

- Premier lissage: chaque prédiction devient le signe de la somme pondérée de son voisinage, en accordant un poids plus important aux voisins valant -1. Le nombre de voisins pris en compte est un hyperparamètre que nous fixons à 12 après optimisation par tests.
- Deuxième lissage: cette fois-ci nous nous intéressons aux labels isolés (ie entourés par deux labels de l'autre classe), ou aux séquences de labels différentes (-1,1,-1,1,-1). Dans le premier cas nous fixons la classe du label isolé aux labels qui l'entourent, dans le deuxième cas nous prenons le label majoritaire dans le voisinage en conflit
- Troisième lissage: Nous remarquons un effet de bord sur les labels prédits en apprentissage : il arrive que les séquences de -1 soient plus courtes que prévues car les labels en bordure de séquence, bien que intialement prédites comme -1, ont été moyennes à 1. Nous proposons donc un dernier lissage qui consiste à regarder les voisins directs en début et en fin de séquences de -1, si leurs prédictions étaient originellement à -1 (avant lissage). Si oui on met à -1 les suites de 1 originellement à -1 dans le voisinage direct des bords de séquences de -1.

8. Résultats:

Les résultats obtenus sur le fichier de test varient suivant le modèle utilisé, et la technique d'équilibrage des données, nous allons illustrer les différents scores obtenus dans la figure suivante:



I. Analyse des sentiments : revues positives et négatives des films

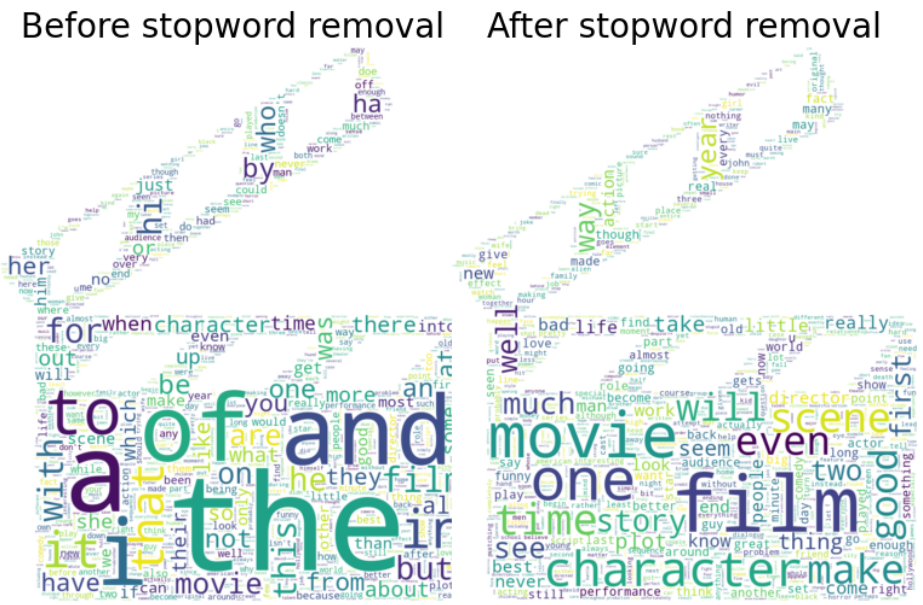
1. Introduction :

Notre arsenal de données est fourni par la base de données iMDB, qui nous gratifie d'une liste conséquente de revues de films. Cette liste, aussi diverse que contrastée, contient 12500 d'avis négatifs et 12500 d'avis positifs (soit 25000 avis pour les données d'apprentissage), les données sont aussi réparties de la même manière dans l'ensemble de test (25000 avis avec une moitié étant positive et l'autre négative). Ces avis sont étiquetés par des labels explicites, soit 0 pour les critiques négatives, soit 1 pour les critiques positives. Par ailleurs, cette montagne de données nous sert de substrat pour notre apprentissage, qui consiste à façonner un classifieur de pointe, un outil redoutable capable de labelliser, en un tour de main, un corpus conséquent d'avis de films. En somme, notre défi consiste à dresser un système de classification puissant, capable de donner vie à nos données en les organisant de manière pertinente.

Dans un premier temps, nous avons utilisé une base réduite qui contient que 1000 avis négatifs, et 1000 avis positifs pour mieux comprendre les données et pour faire une analyse préliminaire. Après, pour l'apprentissage et l'expérimentation avec les différents modèles, nous avons directement utilisé la grande base de données, ce qui nous a obligé de bien choisir les paramètres qu'on veut tester pour réduire le temps de calcul.

2. Analyse préliminaire :

Avant de plonger dans les méandres de notre étude, nous avons pris le temps d'examiner notre base d'apprentissage sous toutes les coutures. La première chose importante qu'on remarque, c'est le parfait équilibre de notre liste de revues de films iMDB, en contraste avec la base de données présidentielle. Autrement dit, nous avons la chance de travailler avec une base de données équilibrée, ce qui écarte d'emblée le risque de sous-apprentissage d'une classe minoritaire. En tentant de creuser un peu plus loin, nous nous sommes aperçus que le simple fait d'effectuer un wordcloud sur chaque classe de revues ne nous a pas permis d'aboutir à une conclusion satisfaisante, même en utilisant une vectorisation tf-idf et en supprimant les stopwords. Dans un premier temps, nous retrouvons un champ lexical lié au cinéma ('movie', 'director', 'actor', etc…) ce qui ne décrit pas les sentiments des revues.



Transformation paramétrique du texte (pré-traitements) :

Les pré-traitements que nous avons considéré sont les suivants :

- La suppression de la ponctuation et les chiffres.
- La mise en minuscule.
- La normalisation afin de supprimer les accents et les caractères spéciaux.
- Le stemming.
- L'élimination des stopwords.

Pour limiter le nombre de combinaisons à tester par le grid searching après, nous appliquons toujours la suppression de la ponctuation ainsi que les chiffres, la mise en minuscule, et la suppression des stopwords.

3. Extraction du vocabulaire :

3.1. La taille originale du vocabulaire :

En utilisant le petit jeu de données (1600 exemples pour l'apprentissage), on retrouve une taille de vocabulaire = 36,246. En utilisant le grand jeu de données (25000 exemples pour l'apprentissage), on retrouve une taille de vocabulaire = 74,849. On peut déjà constater que si on garde la totalité du vocabulaire, il y a une forte chance qu'on se retrouve avec des problèmes de sur-apprentissage, cela est dû à la malédiction de la dimensionnalité (il y a beaucoup plus de features qu'il y a d'exemples).

3.2. Quels sont les 100 mots les plus discriminants au sens de odds ratio?

Dans la classification des sentiments, le odds ratio est une mesure de l'association entre une caractéristique (par exemple, un mot) et une classe particulière (par exemple, un sentiment positif ou négatif). La formule de l'odds ratio est la suivante :

$$\text{odds ratio}(\text{mot}) = \frac{p(\text{mot}|\text{positif}) * (1 - p(\text{mot}|\text{negatif}))}{p(\text{mot}|\text{negatif}) * (1 - p(\text{mot}|\text{positif}))}$$

où:

p(mot|positif) est la probabilité d'observer le mot étant donné que le sentiment est positif.

p(mot|négatif) est la probabilité d'observer le mot étant donné que le sentiment est négatif.

L'odds ratio nous indique combien il est plus probable que le mot apparaisse dans la classe positive par rapport à la classe négative. Si l'odds ratio est supérieur à 1, cela suggère que le mot est davantage associé à la classe positive, tandis que si l'odds ratio est inférieur à 1, cela suggère que le mot est associé à la classe négative.

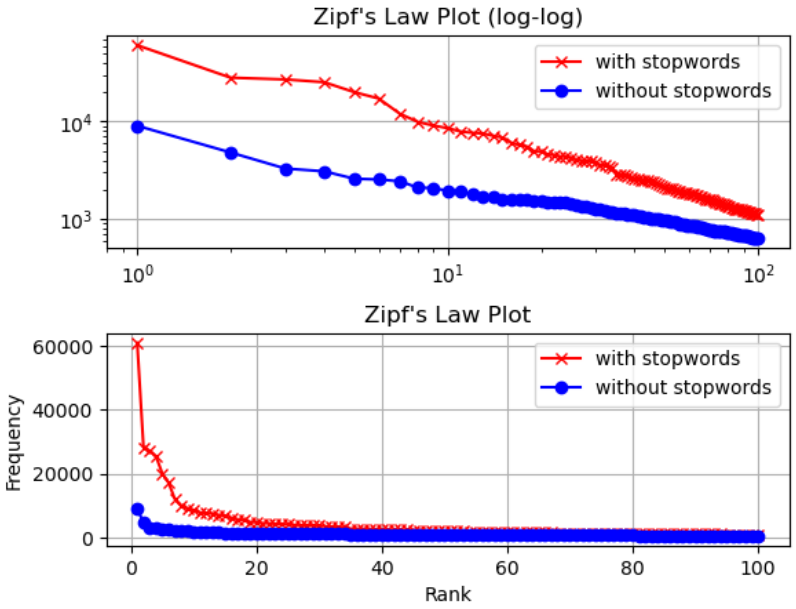
On peut déjà remarquer que la technique d'odds ratios nous permet de mieux distinguer les sentiments négatifs de deux qui sont positifs, avec des champs lexicaux assez descriptifs ('bad', 'stupid','bore'...) pour les revues négatives, ('great','best','perfect'...) pour les revues positives. Cela est achevé avec un modèle BoW en utilisant CountVectorizer.

3.3. Quelle est la distribution d'apparition des mots (Zipf) ?

Selon la loi de Zipf, dans une langue naturelle, les mots les plus fréquemment utilisés ont une fréquence d'utilisation beaucoup plus élevée que les mots les moins fréquemment utilisés. De plus, la fréquence

relative d'utilisation des mots est inversement proportionnelle à leur rang dans la liste des mots classés par fréquence d'utilisation. Autrement dit, le deuxième mot le plus fréquemment utilisé aura une fréquence d'utilisation environ deux fois plus faible que le mot le plus fréquemment utilisé, le troisième mot aura une fréquence d'utilisation environ trois fois plus faible que le mot le plus fréquemment utilisé, et ainsi de suite.

La loi de Zipf peut être représentée par une courbe log-log, où la fréquence relative d'utilisation des mots est représentée sur l'axe des y, et leur rang sur l'axe des x. La courbe est généralement proche d'une droite, ce qui montre que la relation entre la fréquence d'utilisation et le rang est presque linéaire.



3.4. Quels sont les 100 bigrammes/trigrammes les plus fréquents?

TOP 10 BIGRAMS AND THEIR RESPECTIVE FREQUENCIES :
('special effect', 317) ('look like', 251) ('year old', 186) ('even though', 179) ('new york', 168) ('seem like', 164) ('high school', 156) ('star war', 156) ('film like', 152) ('bad movi', 149)

TOP 10 TRIGRAMS AND THEIR RESPECTIVE FREQUENCIES :
('know last summer', 58) ('save privat ryan', 46) ('tommi lee jone', 37) ('new york citi', 35) ('blair witch project', 32) ('scienc fiction film', 29) ('saturday night live', 28) ('jay silent bob', 25) ('wild wild west', 25)
('film take place', 24)

On peut déjà commencer à remarquer que l'inclusion des bi-grammes dans le vocabulaire du corpus ajoute de la sémantique et du sens. Dans le cas où on utilise que des mots unaire, on se retrouve avec pas mal de mots qui décrivent les sentiments de la personne qui rédige, mais il existe un nombre important de mots qui ne décrivent pas les sentiments s'ils sont présentés en forme de 1-gramme ou Bi-gramme, par exemple:

- **"Movie" VS "Bad Movie" :** le mot "Movie" ne permet pas de dire si la personne qui a rédigé le review a aimé le film ou non, contrairement à Bad Movie.
- **"One" VS "One Best"** : le mot "One" ne permet pas de dire si la personne qui a rédigé le review aimé le film ou non, contrairement à "One Best" qui correspond à "One of the Best" après la suppression des stop words "the" et "of".

Avec les tri-grammes, on obtient encore plus de détails sur les reviews, tels que les noms trigrammes des films et les noms des acteurs associés à ces films. Il est à noter que ce n'est pas nécessairement une bonne chose. La classification des sentiments doit rester assez générale pour éviter de tomber dans le piège du surapprentissage.

Une autre chose à noter est l'explosion de la taille du vocabulaire quand on considère des bi-grammes ou des tr-gramme :

- Taille du dictionnaire composé des mots d'unité seulement : 22835
- Taille du dictionnaire composé des bi-grammes seulement : 398547
- La taille du vocabulaire est **17 fois** plus grande.

Il faut absolument mettre un seuil max pour la taille du vocabulaire si on veut maximiser la sémantique qu'on veut obtenir des données pour la classifications, sinon, le temps de calcul va être pénible et les modèles vont trop coller aux données.

En utilisant les bi-grammes et les monogrammes pour représenter le vocabulaire, on obtient déjà plus d'informations que la représentation précédente. Cette fois-ci, des noms très connus sautent aux yeux dans les word clouds générés. Un exemple assez évident est la présence de : star wars, pulp fiction, de niro dans le wordcloud des reviews positifs. Alors que le vocabulaire composé des mots unaires permet de classifier le sentiment à travers des mots distinctifs tel que good ou bad, l'effet que les bi-grammes peuvent apporter est l'association de certains films et certains acteurs à la classe des reviews positifs, indiquant que ces derniers sont appréciés par les utilisateurs, ou bien dans la classe des reviews négatifs. On constate alors que l'utilisation des bi-grammes peut être enrichissante si on veut tirer d'autres informations mis à part les sentiments, tels que les films cultes et les grands acteurs.

3.5. TF-IDF EMBEDDING :

Nous allons maintenant utiliser un embedding TF-IDF et comparer les résultats avec un Embedding basé sur le comptage.

NR (CountVectorizer)

NR (TfidfVectorizer)

PR (CountVectorizer)

PR (TfidfVectorizer)

En comparant les résultats obtenus en utilisant un CountVectorizer VS TfidfVectorizer, on remarque :

- **Negative reviews :** TF-IDF donne des résultats qui mettent en évidence plus de mots négatifs que le Count-Vectorizer, tel que "stupid", ou "waste".
- **Positive reviews :** Les résultats entre les deux sont plus ou moins les mêmes.

Mais pourquoi?

Le TF-IDF est meilleur que le Count-Vectorizer lorsqu'il s'agit d'un problème de classification de sentiments grâce à la Pondération des termes : Le TF-IDF attribue des poids différents aux termes en fonction de leur fréquence dans le document et leur fréquence inverse dans l'ensemble du corpus. Le Count Vectorizer tient compte que de la fréquence d'apparition des mots, alors que TF-IDF attribue une certaine "importance" au mots via la formule précisée en haut. Ainsi, les termes qui apparaissent fréquemment dans un document mais rarement dans l'ensemble du corpus seront pondérés plus lourdement. Cela aide à atténuer l'impact des mots courants qui ne sont pas spécifiques à un document particulier.

3.6. Que-ce-passe t'il si on ne retire pas les stop-words pour TF-IDF?

Bizarrement, le wordcloud des reviews négatifs reste toujours capable de mettre en évidence quelques mots qui décrivent des mauvais sentiments, alors que le wordcloud des reviews positifs est complètement bombardé avec les stopwords classiques de la langue anglaise.

Explication :

Dans certains cas, TF-IDF peut être plus efficace pour la classification des sentiments négatifs que pour les sentiments positifs. Cela peut s'expliquer par le fait que les mots négatifs sont souvent plus rares et plus spécifiques. En autre mots :

Les mots associés à des sentiments négatifs tels que "hate", "horrible" et "terrible" sont souvent moins fréquents que les mots associés à des sentiments positifs tels que "good", "nice" et "like". Cela signifie que les mots négatifs peuvent avoir un poids plus important dans la méthode TF-IDF, car ils sont plus spécifiques à un sentiment négatif donné. Les mots positifs, par contre, auront un poids moins important car ils sont moins exclusifs au sentiment positif (On trouve des mots positifs dans des documents négatifs, par exemple "i don't like it", "not good", si on utilise des bi-gramme, on peut s'y retrouver).

On a plus tendance à dire : "not good" pour exprimer un sentiment négatif, pour être moins méchant, que de dire "not bad" pour exprimer un sentiment positif, au lieu d'utiliser un terme plus gentil tel que "good", ou "great". En gros, on a tendance à exprimer nos sentiments en essayant d'être le plus gentil possible, ha!

Dans certains domaines, il peut être plus important de détecter les sentiments négatifs que les sentiments positifs. Par exemple, dans le domaine de la surveillance de la réputation en ligne, il est plus intéressant de détecter les critiques négatives que les commentaires positifs. Dans ces cas, une erreur de classification pour un sentiment positif peut être moins préjudiciable qu'une erreur de classification pour un sentiment négatif.

3.7. Réduction de la taille du vocabulaire :

La réduction du vocabulaire peut améliorer les résultats, plus spécifiquement :

- Ça peut aider à éviter le surapprentissage (overfitting) du modèle. Le surapprentissage se produit lorsque le modèle apprend à mémoriser les données d'entraînement plutôt que de généraliser à de nouvelles données. Si le vocabulaire est trop grand, il est possible que le modèle apprenne à mémoriser les mots spécifiques à l'ensemble d'entraînement, plutôt que d'apprendre à reconnaître les motifs généraux qui permettent de classifier les sentiments. En réduisant le vocabulaire, on peut réduire le risque de surapprentissage et améliorer la capacité du modèle à généraliser à de nouvelles données.

- Ça peut permettre d'améliorer la vitesse de traitement des données, ce qui peut être particulièrement important dans le cas de grandes bases de données. En effet, plus le vocabulaire est grand, plus le temps de calcul nécessaire pour traiter les données sera important. En réduisant le vocabulaire, on peut accélérer le traitement des données et rendre le modèle plus efficace.

4. Modèles de Machine Learning :

Remarque : Dans cette partie, on utilise majoritairement le grand corpus de données.

Dans notre quête de perfectionnement, nous avons décidé de tester trois modèles de machine learning différents. Il y a tout d'abord le modèle **SVM linéaire**, qui porte le nom de **LinearSVC**, un modèle solide et fiable qui a fait ses preuves dans le domaine de l'apprentissage automatique. Nous avons également porté notre choix sur un modèle **Naive Bayes**, plus spécifiquement le **MultinomialNB**, qui a l'avantage d'être simple à mettre en place et de donner de bons résultats sur les données textuelles. Enfin, nous avons opté pour une **régression logistique**, également connue sous le nom de **LogisticRegression**, un modèle performant et polyvalent qui peut être appliqué à de nombreux types de problèmes. Et pour pousser l'optimisation de nos modèles encore plus loin, nous avons pris soin d'explorer un paramètre de régularisation **C** proposé par sklearn, un paramètre qui se montre particulièrement efficace pour les modèles de régression logistique et SVM.

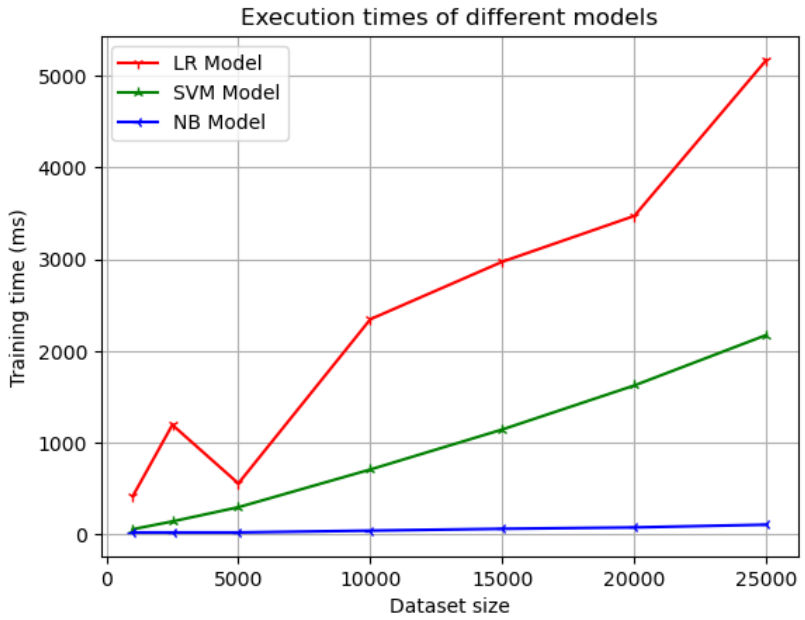
4.1. Choix de la métrique à optimiser :

Nous avons utilisé les trois métriques suivantes pour déterminer les meilleurs paramètres :

- Accuracy
- Courbe ROC-AUC
- F1-Score

Comme les données ici ne sont pas déséquilibrées, l'accuracy est un bon indicateur sur les performances du modèle.

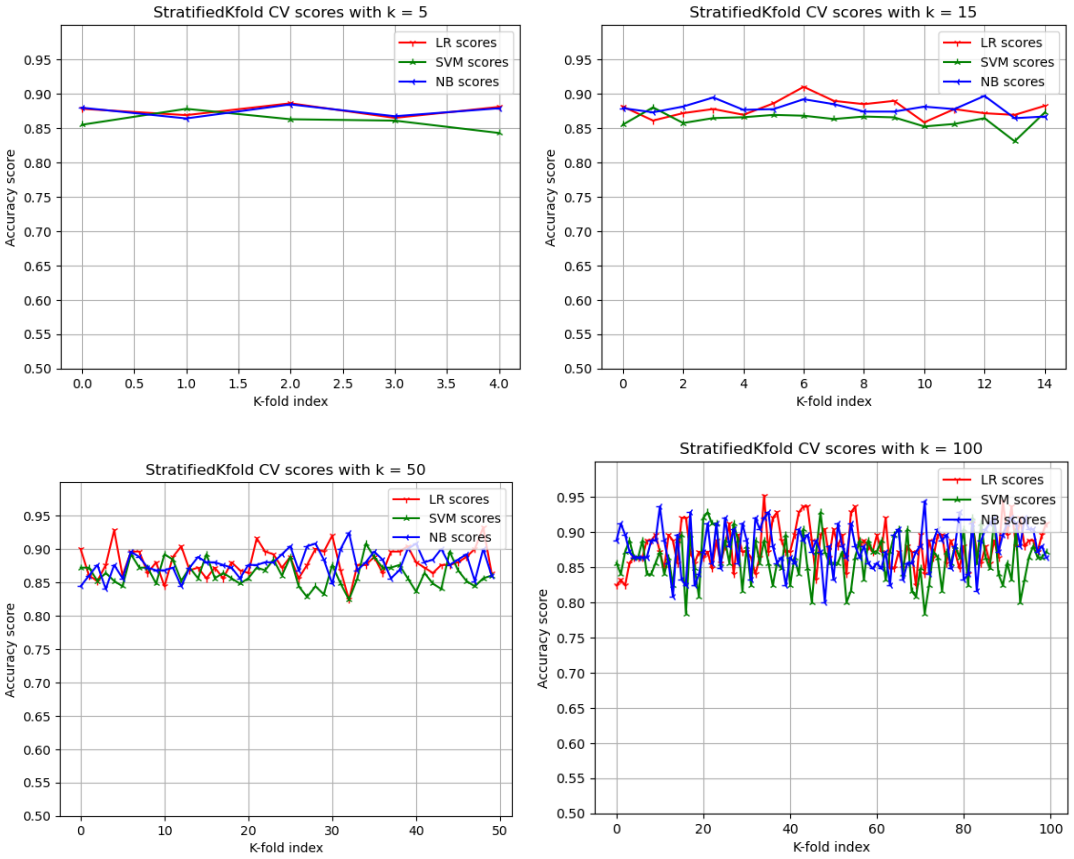
4.2. Evolution du temps de calcul en fonction de la taille du dataset :



La vitesse à laquelle un modèle d'apprentissage automatique fonctionne dépend de nombreuses variables, notamment de la taille de l'ensemble de données, du nombre de caractéristiques et de la complexité du modèle. Cependant, en général, Naive Bayes est perçu comme l'un des algorithmes de classification les plus rapides, suivi par la régression logistique et le SVM linéaire. Naive Bayes est notamment rapide car il s'agit d'un modèle probabiliste simple qui ne nécessite pas d'optimisation itérative comme la régression logistique ou le SVM. Il évalue directement les probabilités a posteriori des classes en fonction des caractéristiques en utilisant le théorème de Bayes et peut ainsi créer rapidement des prédictions.

4.3. Evolution du temps de calcul en fonction de la taille du vocabulaire :

4.4. La validation croisée est-elle stable?



Au fur et à mesure que le nombre de plis augmente, la variance de l'estimation de la validation croisée diminue généralement et l'estimation devient plus stable. Toutefois, l'augmentation du nombre de plis réduit également le nombre d'échantillons dans chaque pli, ce qui peut entraîner une variance plus élevée dans

l'estimation de la performance du modèle. On peut remarquer à partir des figures précédentes qu'à partir de K = 15, la variance dans le score de validation croisée devient plus intense, cela revient à la taille fortement réduite des plis. Par conséquent, la stabilité de l'estimation par validation croisée dépend de l'équilibre entre ces deux effets. Une bonne valeur de k-folds donne autant de plis que possible, qui ne sont pas très différents en ce qui concerne le score. Généralement on choisit K = 5 ou K = 10.

4.5. Grid Searching :

Dans cette partie, nous allons faire un grid search sur les différents paramètres de pré-traitement de texte, d'embedding ainsi que des modèles utilisés pour l'apprentissage. Pour des raisons de complexité, nous allons décomposer le grid search en batchs de combinaisons de paramètres, et générer les résultats au fur et à mesure dans des fichiers externes. Par exemple, soit ‘**n’** la n-ième combinaison de paramètres, nous allons alors trouver les résultats Accuracy, F1 score, Roc-Auc dans un fichier nommé **"output_X_to_Y.txt"**, avec n étant une valeur entre X et Y. Cela nous permet de facilement identifier les paramètres utilisé pour l'obtention des résultats qui maximisent les différents scores, et ça nous donne la possibilité d'interrompre le grid searching à n'importe quel moment puisqu'il y'a une écriture dans un fichier en temps réel (les résultats sont sauvegardés).

Nous allons sélectionner les modèles qui maximisent les différents scores obtenues par validation croisée, puis vérifier leurs performances avec des données de test. On rappelle que les exemples sont entraînés sur 25000 revues qui composent le corpus d’apprentissage/validation, puis les tester sur 25000 revues qui composent le corpus de test.

4.5.1. Choix des paramètres à tester :

Afin de réduire le temps de calcul le plus possible, tout en étant exhaustifs dans notre grid-searching, nous avons décidé de ne pas tester sur certains params:

- **`no_punc` :** limité à True seulement, car la ponctuation n'apporte aucune information sur les sentiments.
- **`no_num` :** limité à True seulement, car les chiffres n'apportent pas une information sur les sentiments non-plus.
- **`lowercase` :** limité à True seulement, permet de réduire le nombre des combinaisons à tester et d'éviter le sur-apprentissage (il ne doit pas y avoir une différence entre Good et good et GOOD).
- **`no_stopwords` :** limité à True seulement, car les stopwords vont causer du sur-apprentissage à coup sûr.
- **`vectorizer` :** limité à Tfidf seulement, vu qu'il parait plus robuste que **`CountVectorizer`** dans les tests initiaux.
- **`ngram_range`** : limité à (1,1) et (1,2) seulement, car on a pu constater précédemment que les bigrammes apportent du contexte supplémentaire, mais qu'ils ne sont pas suffisants pour décrire les sentiments (les monogrammes sont nécessaires aussi), alors que les trigrammes ne décrivent pas grand chose.

Pour le reste des paramètres, nous n'avons pas suffisamment d'informations sur les résultats attendus donc nous avons choisi plusieurs valeurs possibles. Cette partie a pris le plus de temps dans ce projet, car nous utilisons directement le grand jeu de données pour des résultats fiables.

Voici les paramètres ainsi que leurs différentes valeurs qui seront utilisés pour la génération des différentes combinaisons :

```
params = {"processing": { "language": ['english'],
                          "line": [None],
                          "no_punc": [True],
                          "no_num": [True],
                          "lowercase": [True],
                          "norm": [False,True],
                          "no_stopwords": [True],
                          "stemming": [False,True] },
          "vectorizer": {"type": [ TfidfVectorizer ], "ngram_range": [(1,1),(1,2)], "max_features" :
[5000, 10000, 15000, 30000, 50000]},
          "model": {"m": [ LinearSVC, MultinomialNB, LogisticRegression ] },
          "cross_val": { "C": [ 1 if i == 0 else i * 10 for i in range(5) ] }
         }
```

4.5.2. Quels sont les modèles qui donnent les meilleurs résultats?

Les top 10 combinaisons de paramètres (VOCABULAIRE = 5000) :

```
In [28]: scores_df[scores_df['Max Features'] == 5000].sort_values(by=['F1 score','Accuracy score','ROC-AUC score'], ascending=False)[:10]
```

Out[28]:

ID	Language	Line	Lower	Remove digit	Remove punctuation	Remove stopwords	Normalize	Stemming	Max Features	N-gram range	Vectorizer	Model	C	Accuracy score	F1 score	ROC-AUC score
401	english	None	True	True	True	True	True	False	5000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88404	0.885476	0.952251
406	english	None	True	True	True	True	True	False	5000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88380	0.885222	0.953106
106	english	None	True	True	True	True	False	False	5000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88368	0.885068	0.953254
101	english	None	True	True	True	True	False	False	5000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88296	0.884452	0.952737
556	english	None	True	True	True	True	True	True	5000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88272	0.884412	0.951938
551	english	None	True	True	True	True	True	True	5000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88240	0.884143	0.950775
256	english	None	True	True	True	True	False	True	5000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88232	0.883641	0.951671
251	english	None	True	True	True	True	False	True	5000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88168	0.883445	0.950954
552	english	None	True	True	True	True	True	True	5000	(1, 1)	Tf-Idf	Logistic Regression	10	0.87664	0.877779	0.947426
407	english	None	True	True	True	True	True	False	5000	(1, 2)	Tf-Idf	Logistic Regression	10	0.87676	0.877605	0.947809

Les top 10 combinaisons de paramètres (VOCABULAIRE = 10000) :

In [25]:
```
scores_df[scores_df['Max Features'] == 10000].sort_values(by=['F1 score','Accuracy score','ROC-AUC score'], ascending=False)[:10]
```

Out[25]:

ID	Language	Line	Lower	Remove digit	Remove punctuation	Remove stopwords	Normalize	Stemming	Max Features	N-gram range	Vectorizer	Model	C	Accuracy score	F1 score	ROC-AUC score
116	english	None	True	True	True	True	False	False	10000	(1, 2)	Tf-Idf	Logistic Regression	1	0.89028	0.891930	0.956476
416	english	None	True	True	True	True	True	False	10000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88896	0.890561	0.956073
566	english	None	True	True	True	True	True	True	10000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88812	0.889780	0.955312
266	english	None	True	True	True	True	False	True	10000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88832	0.889678	0.955404
111	english	None	True	True	True	True	False	False	10000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88784	0.889289	0.955618
411	english	None	True	True	True	True	True	False	10000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88736	0.889027	0.955859
117	english	None	True	True	True	True	False	False	10000	(1, 2)	Tf-Idf	Logistic Regression	10	0.88692	0.887857	0.954512
417	english	None	True	True	True	True	True	False	10000	(1, 2)	Tf-Idf	Logistic Regression	10	0.88608	0.886801	0.953939
412	english	None	True	True	True	True	True	False	10000	(1, 1)	Tf-Idf	Logistic Regression	10	0.88580	0.886604	0.954610
561	english	None	True	True	True	True	True	True	10000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88456	0.886366	0.953694

Les top 10 combinaisons de paramètres (VOCABULAIRE = 15000) :

In [29]:
```
scores_df[scores_df['Max Features'] == 15000].sort_values(by=['F1 score','Accuracy score','ROC-AUC score'], ascending=False)[:10]
```

Out[29]:

ID	Language	Line	Lower	Remove digit	Remove punctuation	Remove stopwords	Normalize	Stemming	Max Features	N-gram range	Vectorizer	Model	C	Accuracy score	F1 score	ROC-AUC score
126	english	None	True	True	True	True	False	False	15000	(1, 2)	Tf-Idf	Logistic Regression	1	0.89184	0.893377	0.957671
277	english	None	True	True	True	True	False	True	15000	(1, 2)	Tf-Idf	Logistic Regression	10	0.89144	0.892326	0.956874
127	english	None	True	True	True	True	False	False	15000	(1, 2)	Tf-Idf	Logistic Regression	10	0.89096	0.891852	0.958126
426	english	None	True	True	True	True	True	False	15000	(1, 2)	Tf-Idf	Logistic Regression	1	0.89020	0.891811	0.957654
577	english	None	True	True	True	True	True	True	15000	(1, 2)	Tf-Idf	Logistic Regression	10	0.89056	0.891401	0.955873
576	english	None	True	True	True	True	True	True	15000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88932	0.890850	0.956319
427	english	None	True	True	True	True	True	False	15000	(1, 2)	Tf-Idf	Logistic Regression	10	0.88988	0.890794	0.957725
421	english	None	True	True	True	True	True	False	15000	(1, 1)	Tf-Idf	Logistic Regression	1	0.88896	0.890537	0.956175
122	english	None	True	True	True	True	False	False	15000	(1, 1)	Tf-Idf	Logistic Regression	10	0.88944	0.890458	0.956162
276	english	None	True	True	True	True	False	True	15000	(1, 2)	Tf-Idf	Logistic Regression	1	0.88872	0.890399	0.956535

D'après les résultats obtenus, il semble que la **régression logistique** est la plus performante, avec des scores dans le voisinage de **0.9**. Nous allons analyser trois modèles de régression logistique, mais au lieu de choisir celui en premier, on va choisir le premier dans les TOP 10 qui semble être :

- **Le plus généralisant :** normalisation et stemming activés. Cela permet d'éviter le surapprentissage le plus possible. Le modèle classé en 1er n'est pas forcément meilleur que celui ci, car un modèle plus généralisant et qui donne des performances quasi identiques est favorisé.
- Mais aussi plus riche en terme de vocabulaire **(Pas trop riche, c'est pour ça qu'on se limite à 5000, 10000 et 15000, car si on choisit plus que ça, on aura un nombre de features qui est plus grand que la taille du dataset)**

En utilisant ce raisonnement :

- Pour vocabulaire limité à 5000 : on choisit le modèle `556` (5ème ligne)
- Pour vocabulaire limité à 10000 : on choisit le modèle `566` (5ème ligne)
- Pour vocabulaire limité à 15000 : on choisit le modèle `577` (5ème ligne)

Ainsi, nous avons décider de choisir les paramètres suivants :

```
{'processing': {'language': 'english',
  'line': None,
  'lowercase': True,
  'no_num': True,
  'no_punc': True,
  'no_stopwords': True,
  'norm': True,
  'stemming': True}
```

```
{'vectorizer': {'max_features': 5000, 10000, 15000
  'ngram_range': (1, 2),
  'type': sklearn.feature_extraction.text.TfidfVectorizer}
```

```
'model': {'m': sklearn.linear_model._logistic.LogisticRegression},
 'cross_val': {'C': 1, 10}}
```

```
Logistic regression model with max_features =  2000
              precision    recall  f1-score   support

           0       0.88      0.87      0.87     12500
           1       0.87      0.88      0.88     12500

    accuracy                           0.87     25000
   macro avg       0.87      0.87      0.87     25000
weighted avg       0.87      0.87      0.87     25000


Logistic regression model with max_features =  5000
              precision    recall  f1-score   support

           0       0.89      0.88      0.88     12500
           1       0.88      0.89      0.88     12500

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000


Logistic regression model with max_features =  10000
              precision    recall  f1-score   support

           0       0.89      0.88      0.88     12500
           1       0.88      0.89      0.88     12500

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000


Logistic regression model with max_features =  15000
              precision    recall  f1-score   support

           0       0.89      0.88      0.88     12500
           1       0.88      0.89      0.89     12500

    accuracy                           0.88     25000
   macro avg       0.88      0.88      0.88     25000
weighted avg       0.88      0.88      0.88     25000
```

On remarque qu'il y a une amélioration dans les différents scores sur les données de test en allant de 2000 à 5000 mots de vocabulaire par exemple. Mais après, l'amélioration est négligeable, ce qui favorise le modèle entraîné avec 5000 features, puisqu'il est plus petit en termes de dimensions et donc moins sujet à l'overfitting.

4.5.3. Et les autres modèles?

Il est important de comparer les résultats des meilleurs modèles de régression logistique avec les meilleurs modèles SVM et de Naive Bayes. Nous avons récupéré les TOP 10s SVMs et NBs pour analyser les résultats un peu plus. Avec les mêmes démarches, nous avons constaté que le modèle SVM donne des performances plus ou moins identiques à celles de la régression logistique, et que Naive Bayes est un peu plus faible en terme d'accuracy à cause de l'hypothèse forte d'indépendance des features, mais qui reste un bon modèle excellent grâce à sa rapidité.