Classification supervisée et traitement du langage naturel

Durand Enzo[†], **Serraf Dan**[‡]

Abstract. Nous avons mené une campagne d'expérience afin de comprendre 2 jeux de données, puis de prédire l'appartenance à une classe de nouvelles données. La première étape est de comprendre les données textuelles grâce à plusieurs techniques de visualisation de données. Il faut ensuite faire beaucoup de différents tests, car les intuitions ne suffisent pas toujours. Pour cela, il faut réfléchir à l'organisation d'un pipeline car on ne peut pas tester toutes les combinaisons possibles de paramètres. Il faut veiller à utiliser les métriques les plus intéréssantes, mais aussi faire attention au sur-apprentissage.

1. Jeu de données des présidents

Dans cette partie, on s'intéressera au jeu de données sur les présidents Jacque Chirac et Francois Mitterrand. Nous avons un fichier de données contenant des blocs de discours entre ces deux hommes, avec les étiquettes associées. Un deuxième fichier contient des blocs de discours mais cette fois-ci non étiquetés. Le but de ce projet est d'arriver a prédire qui est le locuteur de chacune des phrases contenu dans le fichier de test.

1.1. Analyse des données

Lorsque l'on travaille sur un problème de classification, une des premières étapes est de visualiser la répartition des classes dans nos données. En effet, cela est très important car un modèle de machine learning fera toujours au plus simple afin de maximiser un score de précision. Par exemple dans notre cas, nous avons 87% exemples pour Jacques Chirac et 13% d'exemples pour Francois Mitterand. Cela veut dire que notre classifieur peut avoir 87% de précision en prédisant que toutes les phrases proviennent de la classe majoritaire.

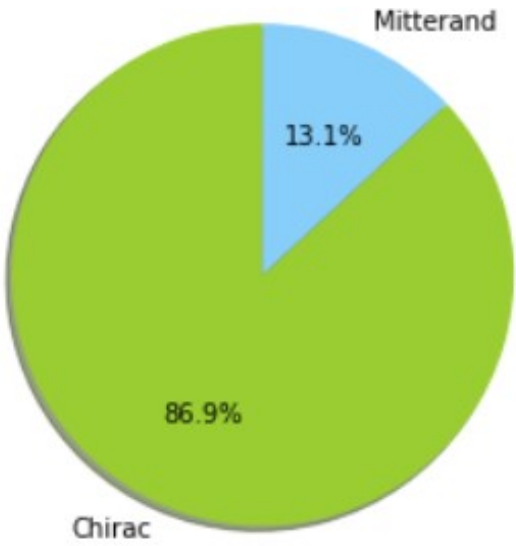


Figure 1. Répartition classes Présidents.

Nous avons utilisé la librairie wordcloud afin de visualiser les mots apparaissant le plus dans notre jeu de données. Sans surprise, nous obtenons presque uniquement des **stopwords**.

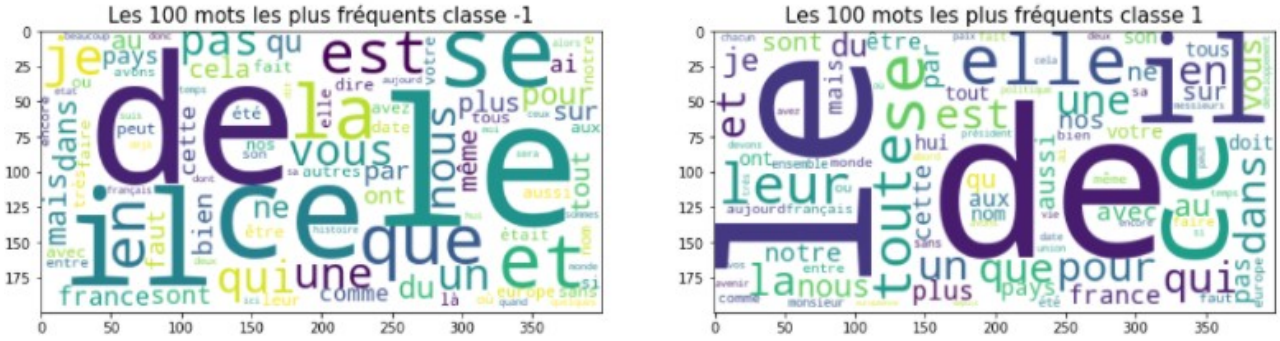


Figure 2. Top 100 mots avant pré-processing

Effectivement la distribution des mots dans notre dataset suit une loi de Zipf.

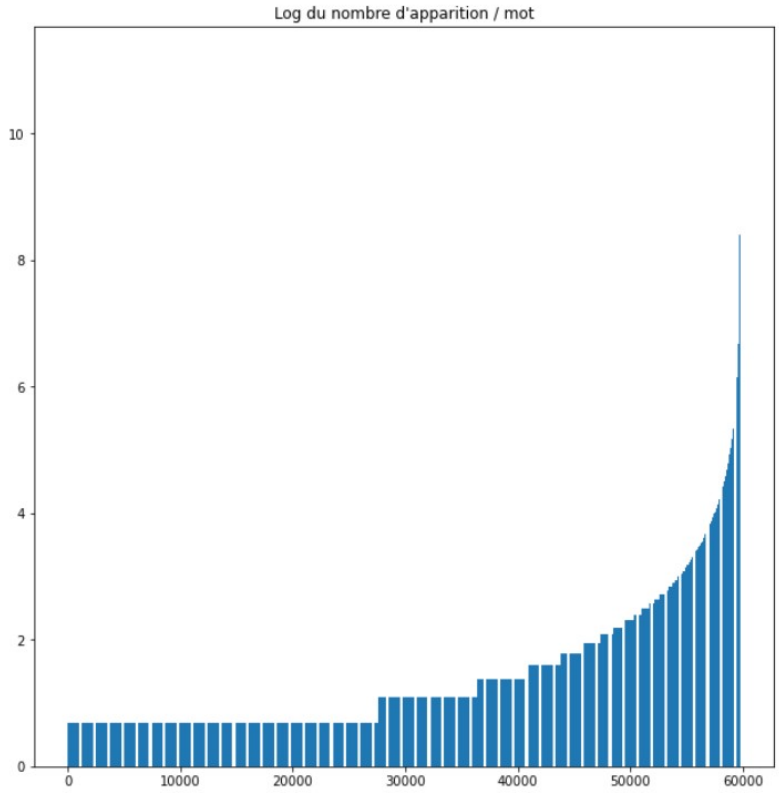


Figure 3. Loi de zipf Présidents.

Ces mots ne nous seront probablement pas utiles afin de différencier qui est le locuteur dans une phrase. Deux choix s'ouvrent donc a nous, utiliser des listes de **stopwords** prédéfinis, ou utiliser le parametre **max_df** de notre vectorizer. La différence étant que le paramètre **max_df** permet de supprimer les stopwords propre au dataset.

Nous avons affiché plusieurs wordcloud mais même en supprimant des **stopwords**, il est compliqué de voir les différences dans les données. C'est pourquoi nous avons décidé d'utiliser le odds ratio afin d'afficher des wordclouds assez différents suivant les deux classes.

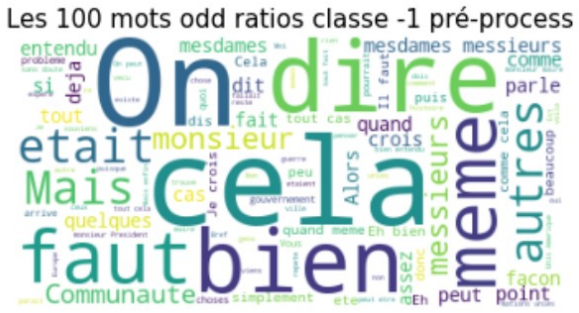


Figure 4. Top 100 mots après pré-processing Mitterand.

Figure 5. Top 100 mots après pré-processing Chirac.

1.2. Recherche des paramètres optimaux pour la représentation des données

1.2.1. Représentation des données

Il y a plusieurs moyens de representer les donn ´ ees textuelles afin d'utiliser des mod ´ eles de machine learning. ‘ Nous avons opte pour le tfidfvectorizer de sklearn apr ´ es avoir test ‘ e plusieurs possibilit ´ es. Le tfidfvectorizer est ´ en fait la combinaison d'un countvectorizer et d'un tfidftransformer. Il permet de representer les mots sous un ´ format statistique qui est interessant pour la suite.

1.2.2. Organisation

Nous avons mene une campagne d'expérience afin de trouver les paramètres optimaux pour notre problème. Premierement, nous avons décidé d'utiliser les fonctions de gridsearch et pipeline de la librairie sklearn. Ces fonctions permettent d'effectuer une recherche de parametres optimaux suivant une métrique et une liste de parametres. Il est possible d'utiliser gridsearch sur des vectorizer (countvectorizer et tfidfvectorizer) ainsi que sur des modeles (LR, MultinomialNB et SVC). Il est possible de paralléliser les opérations grâce au paramêtre **njobs** de gridsearch, ce qui est non negligeable car une des ressources limitantes est le temps.

Un des principaux problèmes est que le nombre de combinaisons de paramètres à tester augmente très rapidement. On ne peut donc pas faire un gridsearch sur l'ensemble des paramètres, il faut faire plusieurs choix.

Nous avons décidé de séparer notre recherche en 3 parties :

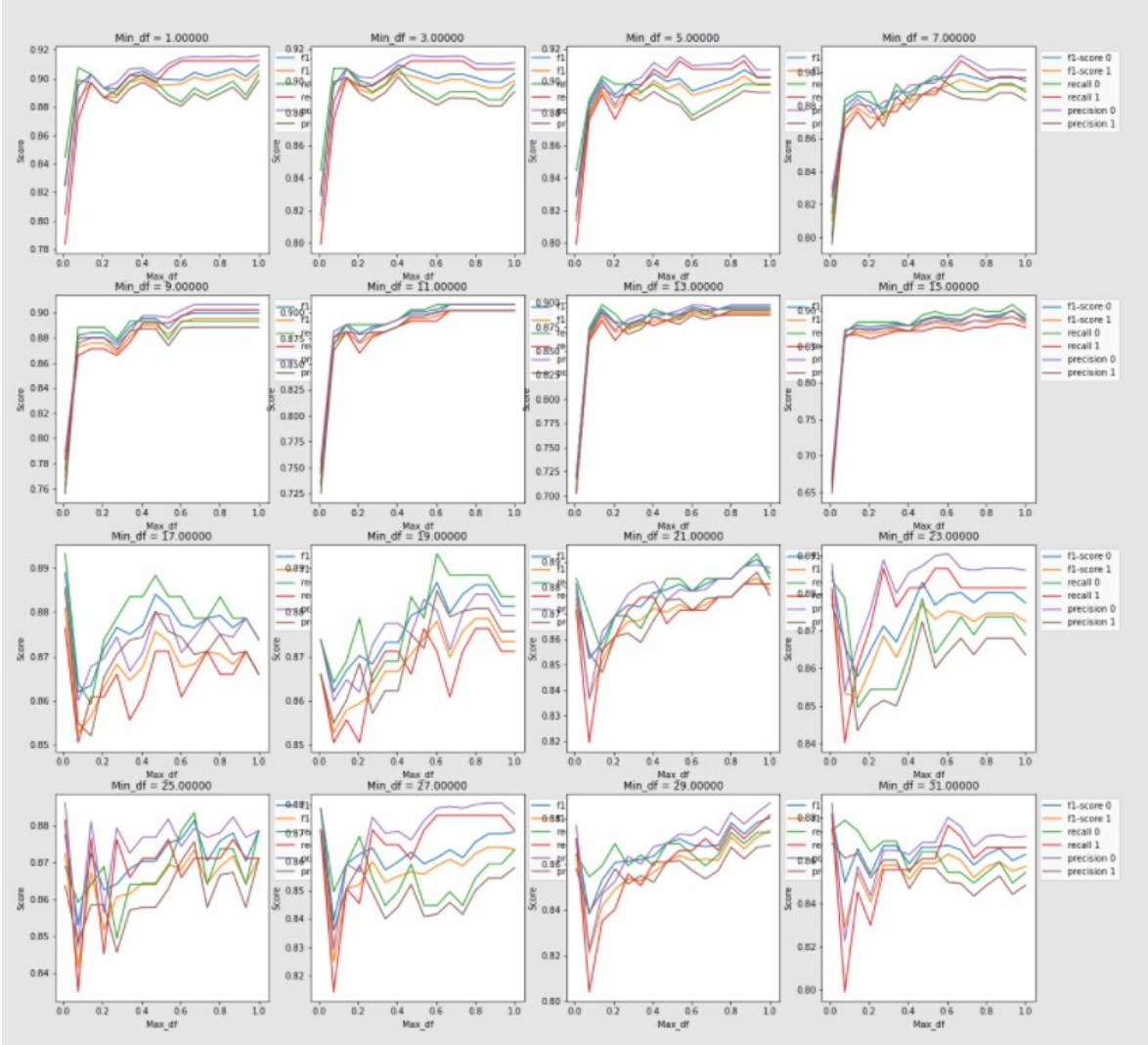
- La recherche des paramètres optimaux pour les données continues (**min_df**, **max_df**, **max_features**).
- La recherche des paramètres booléens ou discret (**use_idf**, **smooth_idf**, **sublinear_idf**, **lowercase**, **strip_accent**, **stopwords**, **ngram_range**).
- La recherche des paramètres des modèles de machine learning (**C**, **alpha**, **class_weights**, **max_iter**).

Afin d'éviter de faire un gridsearch comparant des modèles en sur-apprentissage, il faut prendre quelques

précautions. Lors du gridsearch sur les vectorizers nous utiliserons un modèle prédéfinis dans le pipeline. Ce modèle sera un SVM linéaire, ayant comme paramètres **class_weights="balanced"** et **C=100**. On utilisera aussi un **max_iter=1000** afin de gagner du temps sur la recherche de paramètre. La métrique la plus intéressante est le **f1** score lorsque nous avons des classes non-équilibrées. Nous avons donc utilisé cette métrique dans notre gridsearch. Les métriques **f1_micro** et **rog_auc** ont aussi été essayés mais elles donnent des résultats moins concluants.

1.2.3. Paramètres continus

Après avoir effectué quelques expériences, il semble que les paramètres continus (**min_df**, **max_df**, **max_features**) soient ceux qui influent le plus sur nos résultats. C'est pourquoi nous décidons de faire une première recherche de paramètres optimaux en ne prenant en compte que ceux-ci. Nous sommes conscients que nous perdons une partie de l'espace de recherche en séparant ainsi nos recherches, mais nous diminuons exponentiellement le temps de recherche.



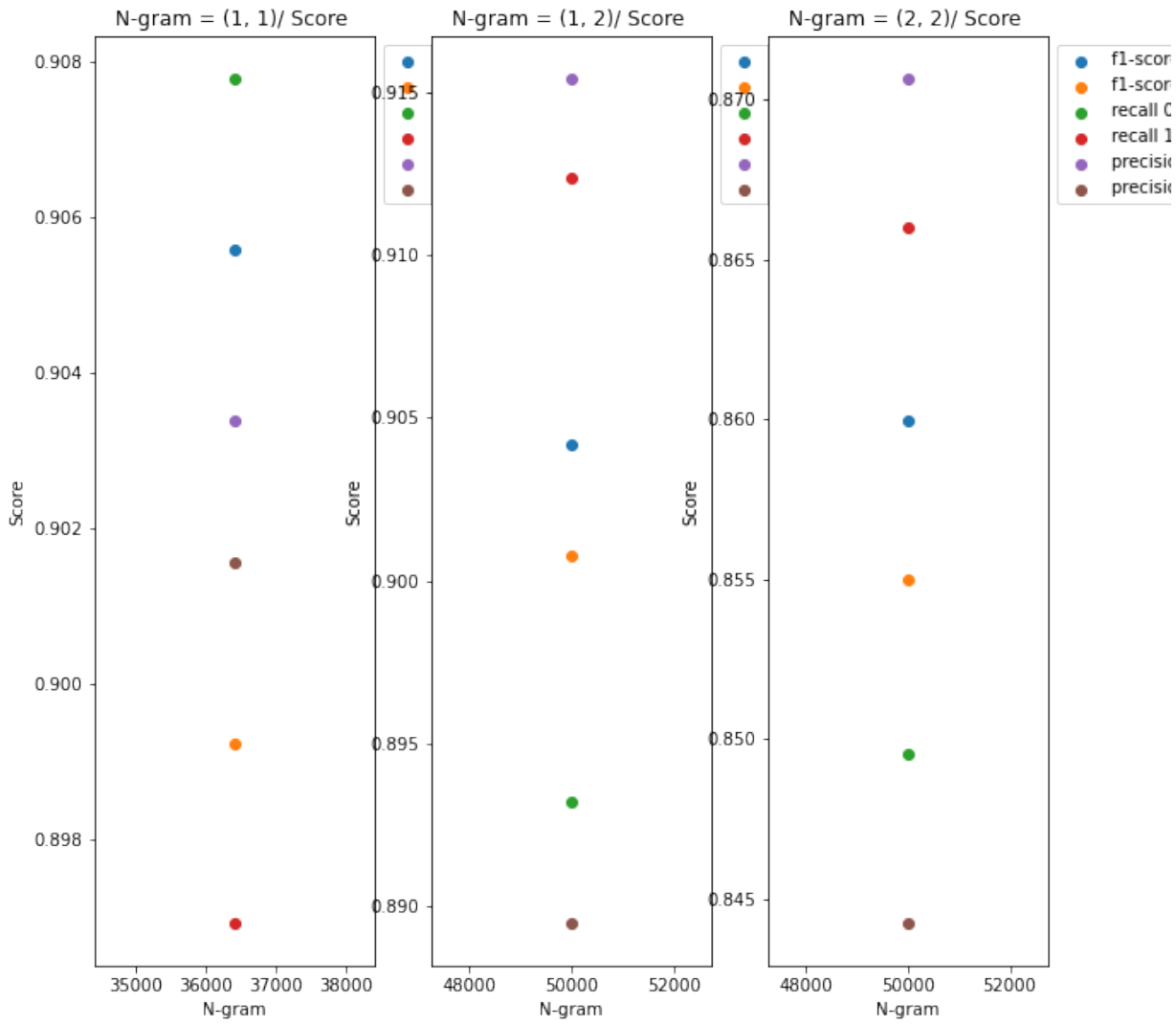


Figure 7. Variation du score suivant n-gram.

Nous utilisons le paramètres **max_features** afin d'optimiser le temps de recherche. Nous essayons de minimiser ce paramètre car il réduit la dimensionalité du problème. Il faut cependant faire attention à ne pas perdre trop de performances.

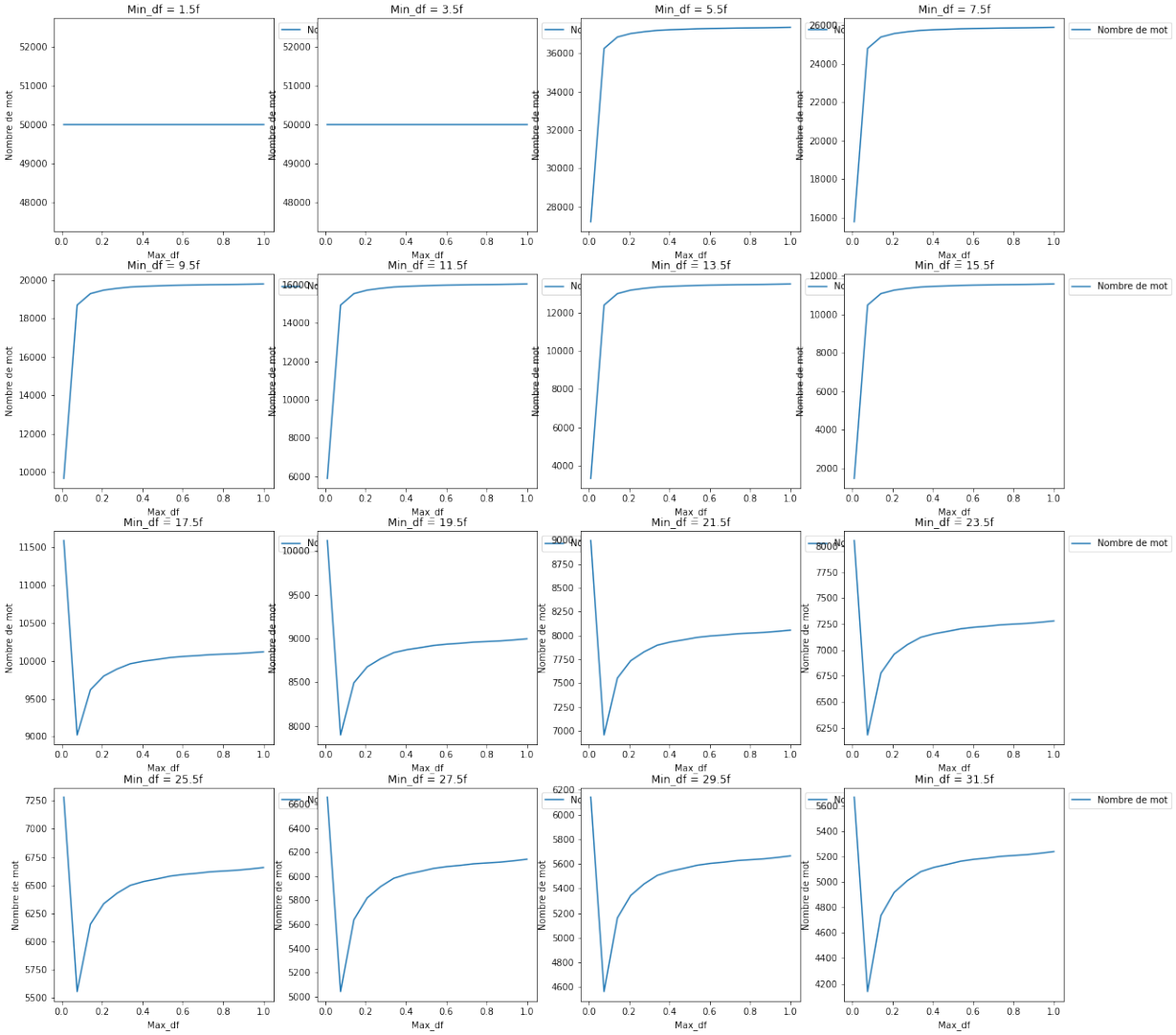


Figure 8. Variation du nombre de mots.

1.2.4. Paramètres booléens ou discrets

Nous voulions tester tous les paramètres existants de nos vectorizer. Certains comme **smooth_idf**, **sublinear_idf** influent très peu sur nos performances. D'autres comme **use_idf** sont assez importantes car il y a de grandes différences lorsque l'on utilise la normalisation par idf. Notre paramètre **use_idf** sera tout le temps activé car nous avons déjà fait notre choix lorsque nous avons choisi d'utiliser le tfidfvectorizer.

Les paramètres comme **ngram_range**, **stopwords**, **lowercase** et **strip_accent** sont eux plutôt influents sur nos performances. Le paramètre **ngram_range** semblent être optimal lorsque nous utilisons à la fois les unigrammes et bigrammes. En effet, les bigrammes ajoutent du contexte aux mots, ce qui est très intéressant, mais ils augmentent aussi fortement la taille du dictionnaire. Le bon compromis semble donc être l'utilisation d'unigramme et de bigramme. Pour notre problème, la suppression des **stopwords** fait baisser nos scores, cette liste de mots étant générale, on perd probablement des **stopwords** utiles à la discrimination des locuteurs (Francois Mitterrand utilise peut être beaucoup le mot "je" par exemple).

image courbe ngram lowercase etc...

1.3. Recherche des paramètres optimaux pour les modèles de machine learning

Après avoir trouvé les paramètres optimaux pour le tfidfvectorizer nous avons utilisé la matrice obtenue afin d'entraîner différents modèles. Nous avons le choix d'optimisation pour les paramètres suivants :

- LinearSVC : **C**, **class_weight**, **max_iter**
- MultinomialNB : **alpha**
- LogisticRegression : **C**, **class_weight**, **max_iter**

Le paramètre **class_weight** sera toujours fixé à "balanced" car la répartition des étiquettes n'est pas du tout équilibrée comme vu précédemment. Le paramètres **max_iter** sera cette fois si plus elevé, on peut se le permettre car il y a beaucoup moins de paramètres à tester pour les modèles de machine learning. L'optimisation se fait donc sur le paramètres **C** pour le SVC et la LR ou sur le paramètres **alpha** pour le MultinomialNB.

image courbe parametre C et alpha

1.4. Choix du modèle et vote

Nous avons maintenant 3 modèles optimisés ainsi qu'une bonne représentation des données. Après avoir envoyé quelques prédictions et reçut plusieurs résultats, notre modèle SVC est celui qui donne les meilleurs scores, suivis du LR. Les résultats du SVC et LR étant plutôt proche, nous essayons de faire voter ces classifieurs afin d'essayer d'améliorer notre score. Nous utilisons le module votingclassifier de sklearn et le paramètre **voting** afin de préciser que nous voulons utiliser le soft voting. Nous utilisons aussi un hard voting sur les 3 classifiers mais il ne donne pas de bon résultat sur le jeu d'entraînement, car le f1 score du MultinomialNB est assez loin des scores du SVC et LR. Après envoi des résultats, le modèle SVC reste en tête devant le modèle de soft voting. Notre intuition sur ces résultats est que nous avons entraîné notre voting classifier sur le tfidfvectorizer optimisé pour le modèle SVC seulement.

1.5. Post-processing des données

Jusqu'ici, nous obtenons un f1 score sur la classe minoritaire qui est d'environ 55%. Nous avons cependant un à priori sur ce à quoi devrait ressembler notre vecteur. En effet, le texte en entrée doit être un discours entre deux présidents, il doit donc y avoir une structure par bloc. On peut donc faire un post processing des prédictions

les lissant. Pour cela, on fait passer deux algorithmes sur nos meilleures prédictions. Le premier passe sur chaque prédiction et modifie la classe de celle-ci si ses voisins font majoritairement partie de l'autre classe. On a ici 2 paramètres qui sont la **window_size** et le **threshold**. Nous apprenons ces deux paramètres grâce à une comparaison du f1 score sur le jeu de donnée. Nous obtenons en général un **threshold** de 0.6 et une **window_size** de 12

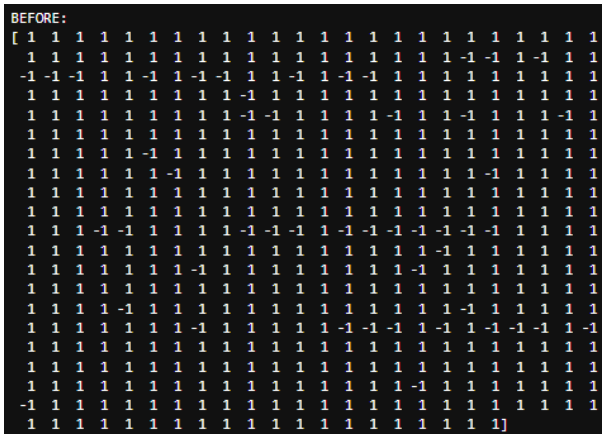


Figure 9. Avant post-processing.

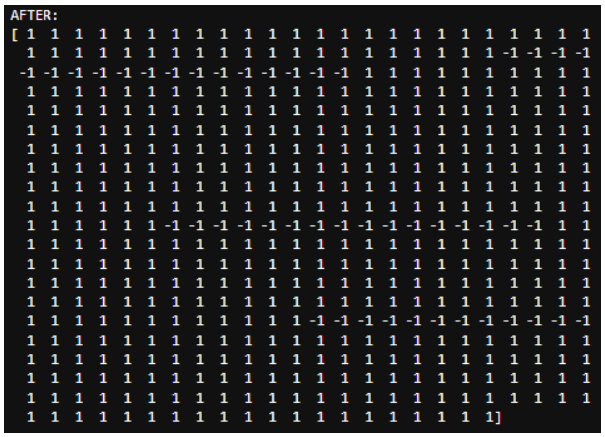


Figure 10. Après post-processing.

1.6. Conclusion

TfidfVectorizer	
lowercase	False
stop_words	None
max_features	100 000
min_df	1
max_df	0.5
ngram_range	(1 ,2)
strip accent	ascii
use_idf	True
Smooth_idf	True
Sublinear_tf	False

Figure 11. Paramètres pré-processing.

SVC	
Max_iter	10 000
C	100
class_weight	balanced

Pré-process	
Threshold	0.6
Windows	12

Figure 12. Paramètres du modèle et post-processing.

2. Jeu de données des films

Dans cette partie, on s'intéressera au jeu de données sur les avis de films. Nous avons un dossier d'apprentissage contenant deux dossiers, l'un contient des avis positifs et l'autre contient des avis négatifs. Comme pour le jeu de données précédent, le but est de classifier des avis d'utilisateur contenu dans un fichier de test.

2.1. Analyse des données

Encore une fois, la première étape est de visualiser la répartition des classes dans nos données. Cette fois-ci par opposition au premier jeu de donnée, nous avons 50% de phrases à connotation positive et 50% de phrases à connotation négative. On remarque que pour ce jeu de donnée les données sont équilibrés.

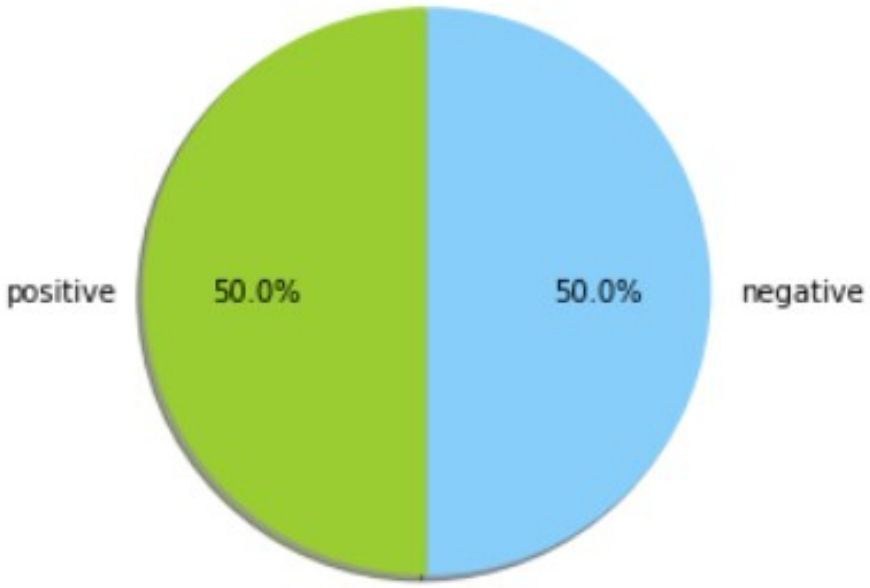


Figure 13. Répartition classes Films.

De manière analogue au jeu de donnée des présidents nous avons réalisé les mêmes campagnes d'expériences. Nous avons d'abord utilisé la librairie wordcloud afin de visualiser les mots apparaissant le plus dans notre jeu de données.

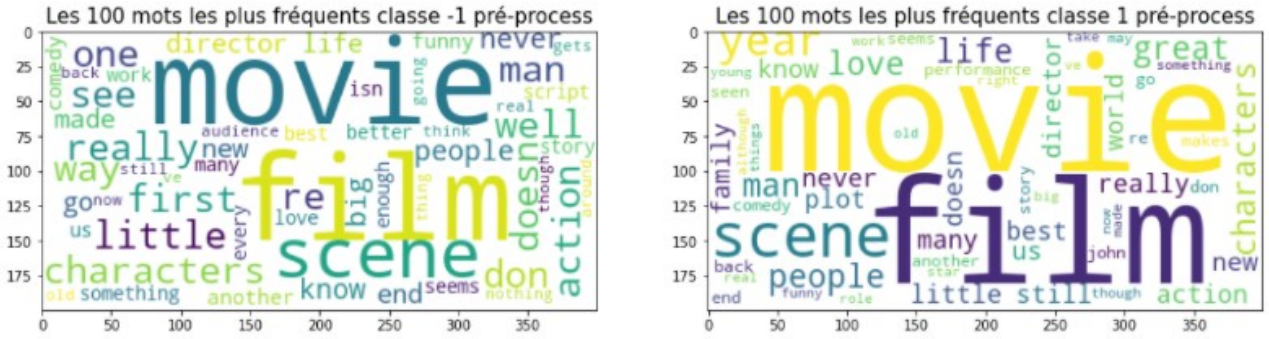


Figure 14. Top 100 mots avant pré-processing

Puis comme nous avons obtenu presque que des **stopwords**, Alors on a utilisé des listes de **stopwords** prédéfinis, et on a utilisé le paramètre **max_df** de notre vectorizer pour filtrer les mots qui n'étaient pas pertinents pour la représentation.

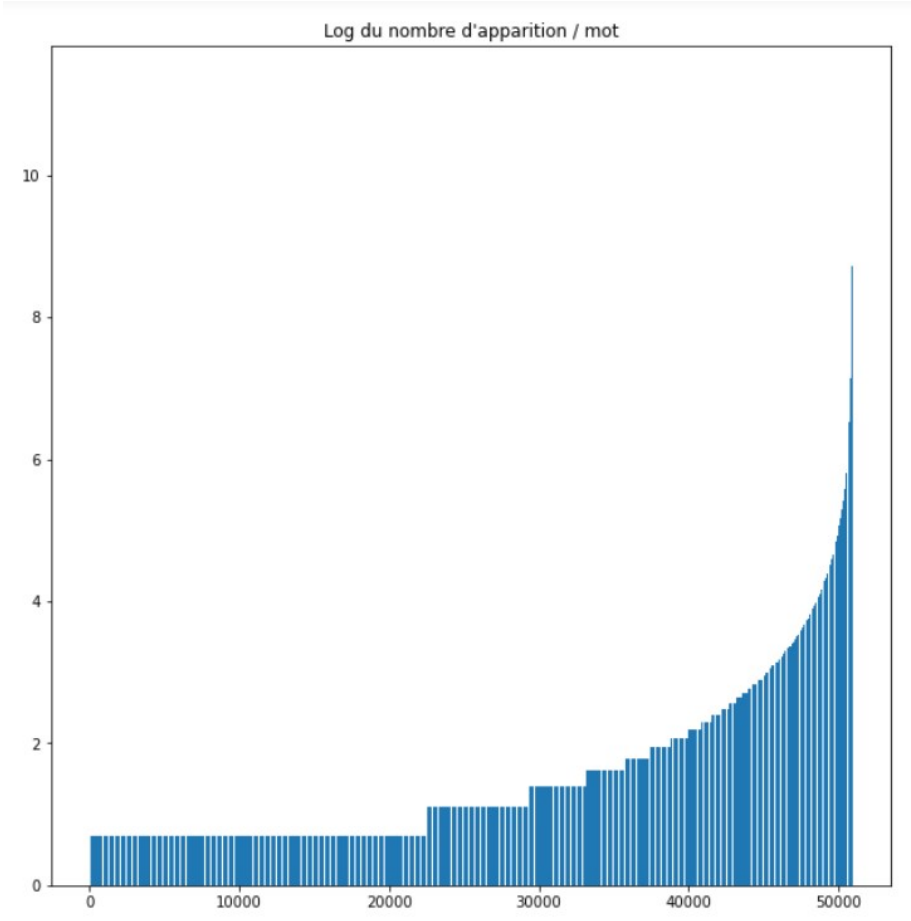


Figure 15. Loi de zipf Films.

Enfin pour confirmer nos résultats on a décidé encore une fois d'afficher les oddsratio qui nous a permit d'avoir

de meilleurs résultats.

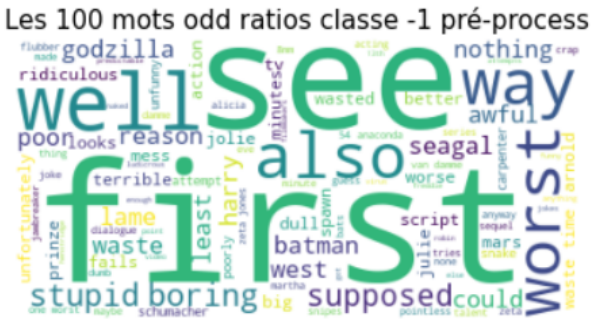


Figure 16. Top 100 mots après pré-processing négatif.

Figure 17. Top 100 mots après pré-processing positif.

2.2. Recherche des paramètres optimaux pour la représentation des données

2.2.1. Représentation des données

De nouveau, pour représenter les données la représentation qui nous a été la plus performante a été le tfidfvectorizer. La comparaison a été faite entre le countvectorizer et le tfidfvectorizer.

2.2.2. Organisation

Nous avons mené une campagne d'expérience afin de trouver les paramètres optimaux pour notre problème. En effet, il faut réaliser encore une fois des fonctions de gridsearch avec différentes pipelines pour trouver les paramètres optimaux. Les métriques et les paramètres change d'un jeu de données à l'autre, c'est pourquoi on a du réentrainer nos modèles. Cette fois-ci, puisque les données étaient équilibrées l'accuracy fut suffisante pour évaluer nos données. Pour cette expérience, nous avons utilisé gridsearch sur les vectorizer (countvectorizer et tfidfvectorizer) ainsi que sur les modèles (LR, MultinomialNB et SVC).

Pour trouver les paramètre optimaux on a dus réalises de nombreuse combinaison. On a séparer l'ensemble des paramètres afin de gagner en temps de calcul. En effet en séparant les différents paramètres cela nous a permit d'éviter de nombreuses combinaisons qui n'était pas pertinente. En ce qui concerne l'organisation, nous avons séparer nos recherches en 3 parties de la même manière que dans le jeu de données des présidents.

2.3. Recherche des paramètres optimaux pour les modèles de machine learning

Après avoir trouvé les paramètres optimaux pour le tfidfvectorizer nous avons utilisé la matrice obtenue afin d'entraîner différents modèles. Nous avons le choix d'optimisation pour les paramètres suivants :

- LinearSVC : **C**, **max_iter**
- MultinomialNB : **alpha**
- LogisticRegression : **C**, **max_iter**

L'optimisation se fait sur le paramètres **C** pour le SVC et la LR ou sur le paramètres **alpha** pour le MultinomialNB.

image courbe parametre C et alpha

2.4. Choix du modèle et vote

Nous avons maintenant 3 modèles optimisés ainsi qu'une bonne représentation des données. Après avoir envoyé quelques prédictions et reçut plusieurs résultats. Cette fois-ci, notre modèle LR est celui qui donne les meilleurs scores, suivis du SVC. Les résultats du LR et SVC étant plutôt proche, nous essayons de faire voter ces classifieurs afin d'essayer d'améliorer notre score. Mais encore une fois ces résultats n'était pas concluants, il faudrait optimiser séparement les différents modèles puis agréger les résultats. On retiendra que notre modèle le plus performant est LR.

2.5. Conclusion

TfidfVectorizer	
lowercase	False
stop_words	None
max_features	10 000
min_df	5
max_df	0.5
ngram_range	(1 ,2)
strip Accent	None
use_idf	True
umooth_idf	True
sublinear_tf	False

Figure 18. Paramètres du vectorizer.

LR	
Max_iter	10 000
C	100
Class_weight	None

Figure 19. Paramètres du modèle.