



Recherche d'Information et Traitement Automatique de la
Langue

RAPPORT DU PROJET TAL

Projet Bag-of-Words

CELIK Simay, 28713301
SOYKOK Aylin, 28711545

Année universitaire : 2023/2024

Sommaire

1	Introduction	2
2	Classification des sentiments (Movies)	2
2.1	Analyse des données et des prétraitements	2
2.1.1	Les données et la distribution des labels	2
2.1.2	La transformation paramétrique du texte avec des pré-traitements	3
2.1.3	Extraction du vocabulaire (BoW) et N-grammes	5
2.1.4	Expérimentations sur la réduction du vocabulaire	6
2.1.5	La différence entre undersampling et oversampling	8
2.2	Modèles de ML et métriques d'évaluation	8
2.2.1	Durée de classification selon la taille de vocabulaire et le classifieur	8
2.2.2	Mise en place de la validation croisée au lieu de train-test-split	9
2.2.3	Recherche du meilleur k pour la validation croisée	9
2.2.4	Expérimentations pour trouver les paramètres de vectorizer utiles	9
2.2.5	Stabilité de la validation croisée	10
2.3	Evaluations sur le serveur de tests	10
2.3.1	Conclusions sur les tests sur le serveur	10
3	Reconnaissance du locuteur (Chirac/Mitterrand)	12
3.1	Analyse des données et des prétraitements	12
3.1.1	Distribution des labels	12
3.1.2	Extraction du vocabulaire (BoW) et N-grammes	12
3.1.3	Tests de BoW sur différents pré-traitements	14
3.1.4	Tests de Suréchantillonnage et de Sous-échantillonnage	15
3.2	Expérimentations des paramètres	16
3.2.1	Pour TfIdfVectorizer	16
3.2.2	Paramètres des modèles	17
3.3	Modèles de ML et métriques d'évaluation	20
3.3.1	Durée de classification selon la taille de vocabulaire et le classifieur	20
3.3.2	Mise en place de la validation croisée au lieu de train-test-split	22
3.3.3	Recherche du meilleur k pour la validation croisée	22
3.3.4	Stabilité de la validation croisée	22
3.4	Evaluations sur le serveur de tests	23
4	Conclusion	23
5	Annexe	24
5.1	La distribution des fichiers	24
5.2	Les paramètres des tests sur le serveur d'évaluation	24

1 Introduction

Le but de notre projet est de l'utiliser Bag-of-Words afin de classifier des données du type text. Nous avons deux bases de données, la première contient des critiques sur des films et cette fois-ci, nous voulons pouvoir analyser le sentiment d'une critique. L'autre contient des discours entre Chirac et Mitterrand et nous voulons pouvoir apprendre comment distinguer le locuteur d'un discours en apprenant à partir de cette base.

Le modèle BoW nous permet d'extraire des caractéristiques des mots selon leur présence dans un texte.

2 Classification des sentiments (Movies)

Nous avons deux notebooks movies_analyse_init.ipynb et movies_analyse_suite.ipynb.

2.1 Analyse des données et des prétraitements

Dans movies_analyse_init.ipynb, nous nous familiarisons avec les données donc ce fichier contient nos premières expérimentations sur les données et les pré-traitements possibles afin de faciliter le déroulement du BoW. Voici nos analyses.

2.1.1 Les données et la distribution des labels

Au total nous avons 2000 reviews dans le dataset train labelés 0 si le sentiment général est négatif et 1 si positif.

Dans la figure fig:sentiments on voit la distribution des deux labels dans le dataset train. La distribution est équilibrée donc nous n'avons pas besoin d'appliquer le processus d'équilibrage.

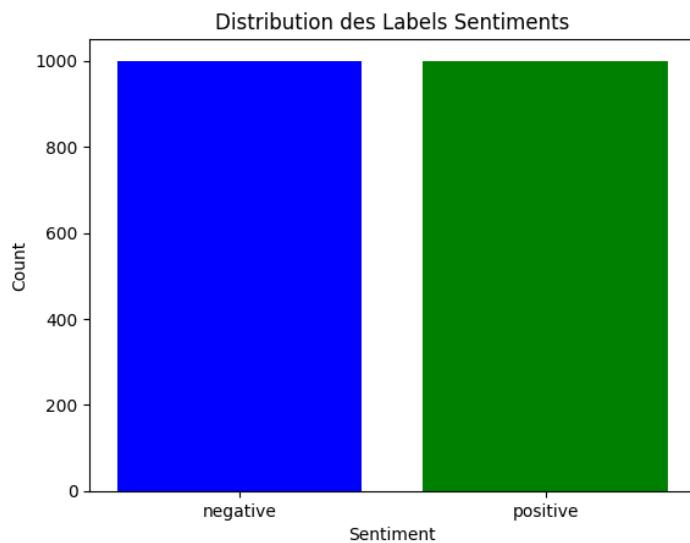


FIGURE 1 – Distribution des labels sentiments

La figure ci-dessous contient une partie d'un exemple de donnée et son label 0. A partir de son label, on sait qu'il est classifié dans la classe négative et même dans une partie du review, on voit des mots comme "terrible". Donc notre but est d'apprendre les mots démontrant des sentiments négatifs ou positifs et de configurer comment les utiliser afin de classifier des données tests.

```

plot : two teen couples go to a church party , drink and then drive .
they get into an accident .
one of the guys dies , but his girlfriend continues to see him in her life , and has nightmares .
what's the deal ?
watch the movie and " sorta " find out . .
critique : a mind-fuck movie for the teen generation that touches on a very cool idea , but presents it in a very bad package .
which is what makes this review an even harder one to write , since i generally applaud films which attempt to break the mold ,
mess with your head and such ( lost highway & memento ) , but there are good and bad ways of making all types of films , and
these folks just didn't snag this one correctly .
they seem to have taken this pretty neat concept , but executed it terribly .
...

```

1

FIGURE 2 – Exemple d'un review négatif

2.1.2 La transformation paramétrique du texte avec des pré-traitements

Dans cette partie, nous avons testé différents types de pré-traitement à appliquer sur les textes afin de les transformer dans un format plus convenable pour BoW. Nous avons vérifié comment la présence d'un pré-traitement fait varier l'accuracy selon trois classificateurs différents, LogisticRegression, LinearSVC et MultinomialNB. Nous avons aussi entré en détail et les testé sur deux vectoriseurs différents, TfidfVectorizer et CountVectorizer. L'évaluation est fait avec train-test-split et les résultats sont ceux de test. Voici les résultats.

LogisticRegression : Taux d'accuracy de TfIdf contre Count : 1.0					
LinearSVC : Taux d'accuracy de TfIdf contre Count : 1.0					
MultinomialNB : Taux d'accuracy de TfIdf contre Count : 0.5					
Taux d'accuracy de LinReg contre LinSVM : 0.0					
Taux d'accuracy de LinReg contre Multinom : 1.0					
Model	Vectorizer	Acc	F1	ROC-AUC	AP
0 Logistic Regression	Tfidf	0.8450	0.850962	0.933400	0.934248
1 Logistic Regression	CountVectorizer	0.8300	0.833333	0.921700	0.920377
2 SVM	Tfidf	0.8500	0.857143	0.934950	0.936000
3 SVM	CountVectorizer	0.8300	0.832512	0.913825	0.910888
4 MultinomialNB	Tfidf	0.8100	0.801047	0.899275	0.902274
5 MultinomialNB	CountVectorizer	0.8175	0.818859	0.883313	0.861430

LogisticRegression : Taux d'accuracy de TfIdf contre Count : 1.0					
LinearSVC : Taux d'accuracy de TfIdf contre Count : 1.0					
MultinomialNB : Taux d'accuracy de TfIdf contre Count : 0.5					
Taux d'accuracy de LinReg contre LinSVM : 0.0					
Taux d'accuracy de LinReg contre Multinom : 1.0					
Model	Vectorizer	Acc	F1	ROC-AUC	AP
0 Logistic Regression	Tfidf	0.8550	0.859903	0.937125	0.936461
1 Logistic Regression	CountVectorizer	0.8275	0.830467	0.925758	0.924425
2 SVM	Tfidf	0.8575	0.861985	0.937875	0.937265
3 SVM	CountVectorizer	0.8250	0.826733	0.917850	0.913471
4 MultinomialNB	Tfidf	0.8025	0.797954	0.903300	0.907646
5 MultinomialNB	CountVectorizer	0.8125	0.814815	0.885400	0.864915

Taux d'accuracy de ponc_sup contre rien : 1.0					
-----------------------------------------------	--	--	--	--	--

LogisticRegression : Taux d'accuracy de TfIdf contre Count : 1.0					
LinearSVC : Taux d'accuracy de TfIdf contre Count : 1.0					
MultinomialNB : Taux d'accuracy de TfIdf contre Count : 0.5					
Taux d'accuracy de LinReg contre LinSVM : 0.0					
Taux d'accuracy de LinReg contre Multinom : 1.0					
Model	Vectorizer	Acc	F1	ROC-AUC	AP
0 Logistic Regression	Tfidf	0.8450	0.850962	0.933400	0.934248
1 Logistic Regression	CountVectorizer	0.8300	0.833333	0.921700	0.920377
2 SVM	Tfidf	0.8500	0.857143	0.934950	0.936000
3 SVM	CountVectorizer	0.8450	0.847291	0.914450	0.912242
4 MultinomialNB	Tfidf	0.8125	0.804178	0.900875	0.902424
5 MultinomialNB	CountVectorizer	0.8200	0.820896	0.885850	0.862376

Taux d'accuracy de chiffre_sup contre rien : 1.0					
--------------------------------------------------	--	--	--	--	--

LogisticRegression : Taux d'accuracy de TfIdf contre Count : 0.5					
LinearSVC : Taux d'accuracy de TfIdf contre Count : 1.0					
MultinomialNB : Taux d'accuracy de TfIdf contre Count : 0.5					
Taux d'accuracy de LinReg contre LinSVM : 0.0					
Taux d'accuracy de LinReg contre Multinom : 1.0					
Model	Vectorizer	Acc	F1	ROC-AUC	AP
0 Logistic Regression	Tfidf	0.8350	0.841346	0.929400	0.931870
1 Logistic Regression	CountVectorizer	0.8400	0.844660	0.928050	0.917027
2 SVM	Tfidf	0.8425	0.847458	0.931775	0.934191
3 SVM	CountVectorizer	0.8425	0.845209	0.917900	0.905973
4 MultinomialNB	Tfidf	0.8000	0.792746	0.891750	0.896354
5 MultinomialNB	CountVectorizer	0.8100	0.810000	0.878675	0.861005

Taux d'accuracy de stem contre rien : 0.0					
-------------------------------------------	--	--	--	--	--

(e) Stemming					
--------------	--	--	--	--	--

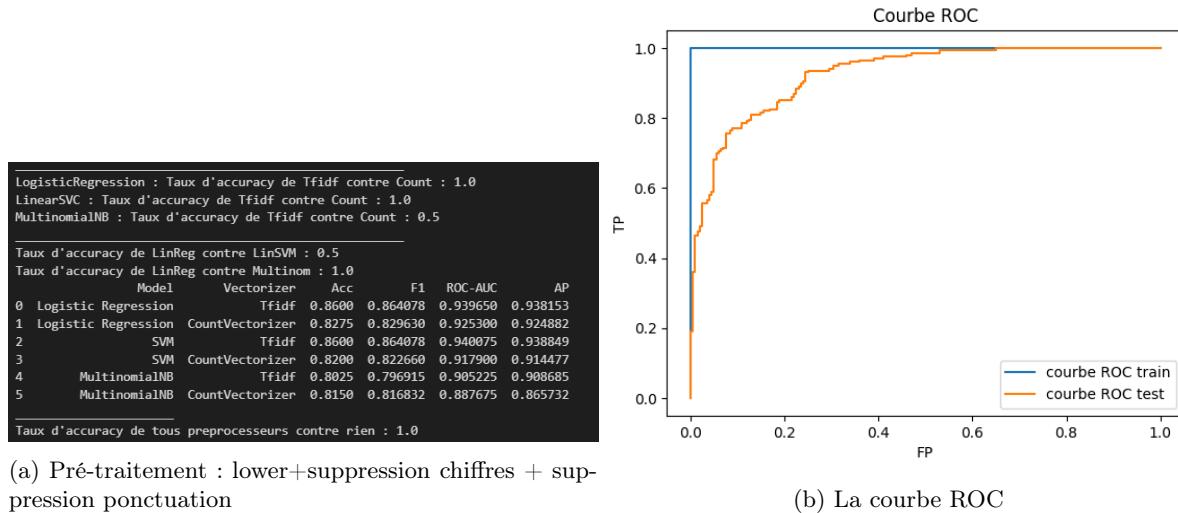
FIGURE 3 – Les résultats d'apprentissage après les pré-traitement

Après avoir vu les résultats, nous nous avons posé les questions suivantes : **Pourquoi les résultats sont-ils assez similaires et pourquoi les métriques d'évaluation sont satisfaisants sans pré-traitement ?** Les points importants à considérer sont les suivants : comme vu dans la figure 2,

la base de train contient des textes propres. Il y a une espace entre les ponctuations, à priori, il n'y a pas de lettre en majuscule et il n'y a pas des balises donc nous n'avons pas eu besoin de tester ce pré-traitement (mais nous avons compté la base des tests et il y a 14667 balises donc ce pré-traitement est certainement utile et ne coûte pas cher). Et dans notre modèle, nous n'avons pas encore commencé à utiliser des stopwords, donc les BoW sont saturés avec des mots qui n'apportent pas d'information à cause de leur fréquence. Donc à ce moment, nous ne pouvons rien conclure. Mais il est intéressant que stemming a diminué légèrement la performance car grâce à stemming, les mots ayant la même origine sont comptés ensemble. La classification des sentiments dépend de la fréquence des mots "positifs" et des mots "négatifs". Les mots apparaissant dans des reviews négatifs vont être considérés négatifs. Donc par l'intuition, fusionner les mots doit nous servir. Cependant, les scores sont toujours élevés donc on peut garder le stemming qui sera normalement utile.

Comparons les scores obtenus. Afin de voir la différence entre TfIdfVectorizer et CountVectorizer, nous avons testé tous les modèles avec eux. Dans tous les cas, le score de Tf-Idf était supérieur ou égale à celui de Count. C'est parce que Tf-idf considère les termes dans le contexte d'un document et dans le contexte de tout le corpus qui permet à distinguer les termes. En plus, par définition Tf-idf "ignore" les stopwords et donc la sémantique est plus cohérente. Tous les modèles sont fidèles pour classifier les sentiments. Néanmoins, la régression logistique et SVM sont plus précis que MultinomialNB. LogisticRegression et LinearSVC mettent de l'importance sur les caractéristiques contrairement à MultinomialNB qui permet de donner la priorité aux mots significatifs.

Combinons quelques pré-processeurs et traçons la courbe ROC.



(a) Pré-traitement : lower+suppression chiffres + suppression ponctuation

(b) La courbe ROC

FIGURE 4 – Les résultats après une série de pré-traitement et sa courbe ROC

Dans la courbe de ROC ci-dessous 4b, on voit que la courbe de train nous donne la courbe ROC parfaite où le taux de vrais positifs $TP_{rate} = \frac{TP}{TP+FP}$ et donc le rappel est égal à 1. Mais la classification de test n'est pas parfaite et donc il y a des faux positifs. Donc on obtient une courbe ROC et la zone sous la courbe nous donne AUC.

2.1.3 Extraction du vocabulaire (BoW) et N-grammes



(a) 100 mots les plus fréquents (b) 100 bi-grammes les plus fréquentes (c) 100 tri-grammes les plus fréquentes (avec stopwords)

FIGURE 5 – Nuages des mots : sans supprimer les stopwords



FIGURE 6 – Nuages des mots : en supprimant les stopwords

Les nuages des mots en dessus nous montre l'importance d'éliminer les stopwords lors de BoW. Dans Figure 5, les mots "the, in, on, as, that, for..." saturent les nuage, même celui des bigrammes, le seul mot qui a pu entrer dans la liste des mots les plus fréquents est "film" qui n'est pas surprenant. Tfifd est assez forte contre les stopwords mais il est toujours important de les traiter particulièrement quand on utilise CountVectorizer (comme nous avons fait ici). Dans Figure 6, nous voyons un nuage sans les stopwords et la différence est évidente. En l'absence des stopwords, le thème est visible et cohérent. Nous pouvons voir quelques groupes des mots indiquant des sentiments comme "much better", "bad movie" etc.



FIGURE 7 – 100 mots les plus discriminants : odds ratio

Dans le nuage des mots les plus discriminants au sein de rapport des cotes peut nous permet de distinguer les termes qui indiquent un fort sentiment. Ici, nous observons plutôt des termes spécifiques à un contexte. Par exemple "shrek", qui est le titre et le nom du protagoniste du film Shrek. Donc, l'apparition de ce mot dans un autre document qui ne parle pas de Shrek est assez faible. Mais si on fait odds ratio au sens des sentiments, il nous servira à distinguer les mots positifs des mots négatifs.

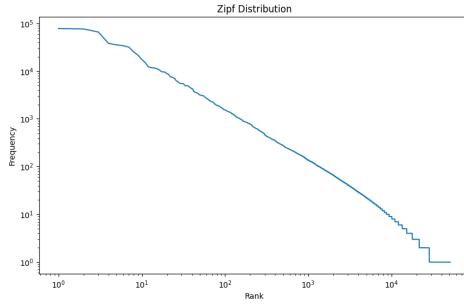


FIGURE 8 – Le graphe de la distribution des mots : Movies

Nous voyons que la distributions des mots dans la base suit la loi de Zipf. On entre un peu plus en détail dans la partie Chirac-Mitterrand

2.1.4 Expérimentations sur la réduction du vocabulaire

Nous avons fait quelques expérimentations sur les effets de la réduction du vocabulaire afin de trouver les meilleurs paramètres pour nos vectorizers (`movies_analyse_init.ipynb`).

vectorizer	TfidfVectorizer
paramètres vectorizer sauf l'expérimenté	{}
classifier	LinearSVC
paramètres classifier	{'random_state' : 0, 'dual' : False}
preprocessor	None (pire cas)
evaluation	train-test-split

TABLE 1 – Les invariants des expérimentations

meilleur max_features Afin de trouver le meilleur `max_features` à utiliser dans notre `TfidfVectorizer`, nous avons testé différents valeurs de `max_features` entre 100 et la taille du vocabulaire totale, si au moins deux scores d'une évaluation (train-test-split) sont supérieurs à ceux courants, alors ce paramètre devient le meilleur. Le paramètre qui donne les meilleurs résultats est $\text{max_features} = 3000$ avec les résultats ci-dessous.

Acc	F1	ROC-AUC	AP
0.8600	0.8641	0.9388	0.9378

TABLE 2 – Résultats du test best `max_feature`

meilleur min_df Afin de trouver le meilleur `min_df` à utiliser dans notre `TfidfVectorizer`, nous avons testé différents valeurs de `min_dfs` entre 0.0, 1.0 en les incrémentant par 0.1. Le paramètre qui donne les meilleurs résultats est $\text{min_df} = 0.0$ avec les résultats ci-dessous. Donc, on n'ignore aucun mot en raison d'apparaître dans peu de documents.

Acc	F1	ROC-AUC	AP
0.8500	0.8571	0.9350	0.9360

TABLE 3 – Résultats du test best `min_df`

meilleur max_df Afin de trouver le meilleur `max_df` à utiliser dans notre `TfidfVectorizer`, nous avons testé différents valeurs de `max_dfs` entre 0.0, 1.0 en les incrémentant par 0.1. Le paramètre qui donne les meilleurs résultats est $\text{max_df} = 1.0$ avec les résultats ci-dessous (idem aux ceux de `min_df`). Donc, on n'ignore aucun mot en raison d'apparaître dans trop de documents.

Acc	F1	ROC-AUC	AP
0.8500	0.8571	0.9350	0.9360

TABLE 4 – Résultats du test best max_df

Quand nous faisons une évaluation avec tous les meilleurs paramètres sur la taille de vocabulaire, on obtient le résultat qu'on a obtenu en dessus pour l'expérimentation de max_features.

Acc	F1	ROC-AUC	AP
0.8600	0.8641	0.9388	0.9378

TABLE 5 – Résultats avec tous les meilleurs paramètres

Quand nous faisons une autre évaluation avec aucune de ces meilleurs paramètres, on obtient des résultats plus basses.

Acc	F1	ROC-AUC	AP
0.8500	0.8571	0.9350	0.9360

TABLE 6 – Résultats sans les meilleurs paramètres

stop_words Au cours de l'apprentissage, il est important d'enlever les stop words même si Tfidf gère les mots trop fréquents avec un poids nul. Nous avons utilisé les stop words en anglais. Voici le résultat qu'on a obtenu.

Acc	F1	ROC-AUC	AP
0.8325	0.8370	0.9261	0.9320

TABLE 7 – Résultats du stopwords

ngram_range Voyons la différence entre BoW binaire, BoW avec bi-grams et BoW avec tri-grams.

Acc	F1	ROC-AUC	AP
0.8500	0.8571	0.9350	0.9360

TABLE 8 – Résultats du BoW binaire

Acc	F1	ROC-AUC	AP
0.8175	0.8249	0.9180	0.9275

TABLE 9 – Résultats du BoW bi-grams

Acc	F1	ROC-AUC	AP
0.8200	0.8218	0.8830	0.8804

TABLE 10 – Résultats du BoW tri-grams

Nous observons une diminution quand on encode des n-grams. Nous pouvons conclure que le BoW binaire est suffisant et effectif pour faire une classification des sentiments. Par l'intuition, on peut dire qu'il suffit de voir les mots "bad", "boring" dans un review.

2.1.5 La différence entre undersampling et oversampling

Quand on fait un sur-échantillonnage ou un sous-échantillonnage sur la base de train, on obtient des résultats identiques. Cela vient du fait que nos données sont déjà équilibrées et contrairement aux données de reconnaissance du locuteur, il n'est pas nécessaire de ré-équilibrer les données.

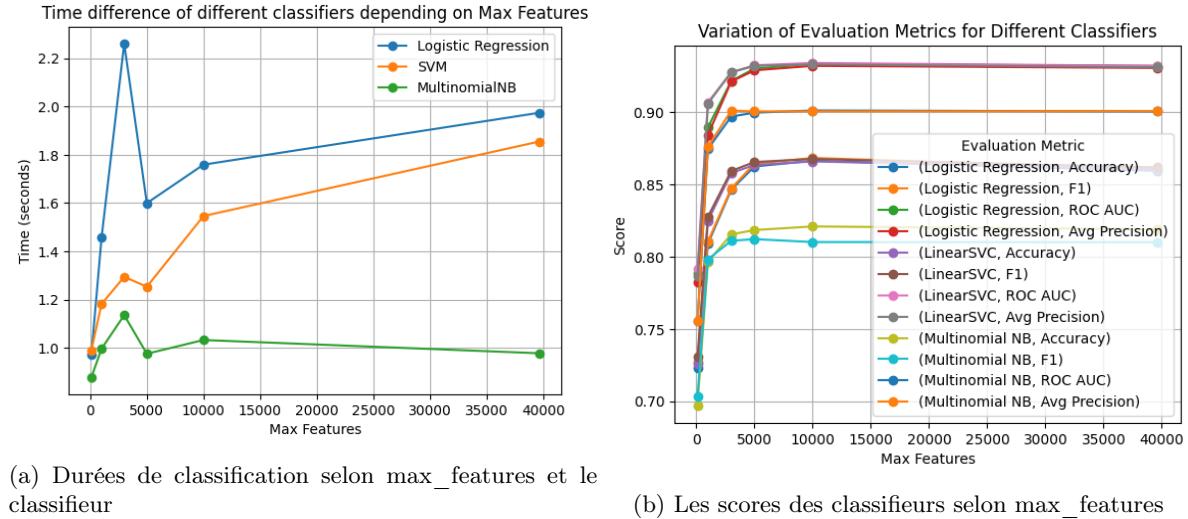
2.2 Modèles de ML et métriques d'évaluation

2.2.1 Durée de classification selon la taille de vocabulaire et le classifieur

Dans cette partie, nous avons testé comment le paramètre `max_features` fait varier la durée de classification. On teste les valeurs de `max_features = {100,1000,3000,5000,10000,taille_vocabulaire}`, simultanément, on calcule la durée de l'apprentissage et la classification et les scores afin de trouver le meilleur `max_features` avec un pré-traitement.

vectorizer	TfidfVectorizer
paramètres vectorizer sauf <code>max_feature</code>	{}
classifiers	LogisticRegression + LinearSVC + MultinomialNB
paramètres LogisticRegression	{'random_state': 0, 'dual': False}
paramètres SVM	{'random_state': 0, 'dual': False}
preprocessor	remove_tags
	transform_uppercase
	chiffre_suppression
	ponc_suppression
evaluation	validation croisée

TABLE 11 – Les invariants des expérimentations



(a) Durées de classification selon `max_features` et le classifieur

(b) Les scores des classificateurs selon `max_features`

FIGURE 9 – Les résultats de l'expérimentation sur la taille de vocabulaire

Dans Figure 9a, nous avons un graphe de la variance de la durée de la classification selon le paramètre `max_features` et le classifieur. On voit que la durée n'est pas linéaire selon `max_features`, la durée de `max_feature = 3000` est supérieure à celle de `5000` pour tous les modèles de classification. Nous voyons que globalement, MultinomialNB est le plus vite et LogisticRegression est le plus lent, SVM est entre les deux. Regardons Figure 9b. Il est difficile à voir mais les scores de LogisticRegression et les scores de LinearSVC sont très proches et majorent ceux de MultinomialNB. Donc dans notre cas utilisation de LinearSVC peut être un bon compromis entre la performance et la précision.

2.2.2 Mise en place de la validation croisée au lieu de train-test-split

Les paramètres et les choix durant la validation croisée sont idem aux ceux en dessus. Dans la figure en dessous, on a les scores obtenus grâce à la validation croisée. La différence entre crossval et train-test-split est négligeable.

	Accuracy	F1	ROC AUC	Avg Precision
Logistic Regression	0.86	0.861249	0.932110	0.930808
LinearSVC	0.86	0.861111	0.932135	0.931117
Multinomial NB	0.82	0.811518	0.900795	0.901007

FIGURE 10 – Les scores de l'évaluation avec la validation croisée

2.2.3 Recherche du meilleur k pour la validation croisée

Nous avons testé plusieurs valeurs de k de la validation croisée plu. La figure en dessous contient une partie du graphe des différents métriques d'évaluation selon k. Nous avons choisi le k maximisant accuracy et f1. Nous n'avons pas considéré ROC-AUC car même s'il est un métrique valable, il était toujours haut au cours du projet donc nous avons voulu augmenter le F1 qui est approprié pour le scénario de la classification des sentiments. Car ici, un faux positif et un faux négatif ont des effets similaires.

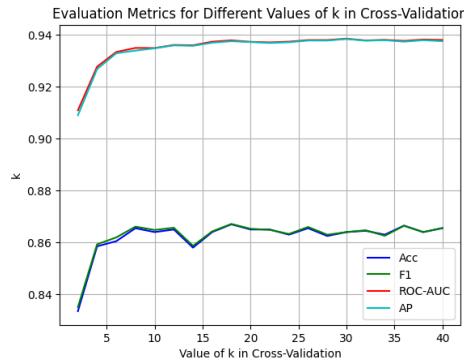


FIGURE 11 – Les scores de l'évaluation selon k dans la validation croisée

Le meilleur k trouvé est 18. Nous continuerons à la suite de cette partie avec k = 18 pour la validation croisée.

2.2.4 Expérimentations pour trouver les paramètres de vectorizer utiles

vectorizer	TfidfVectorizer
paramètres vectorizer sauf max_feature	{}
preprocessor	remove_tags transform_uppercase chiffre_suppression ponc_suppression
paramètres LogisticRegression	{'random_state': 0, 'dual': False}
paramètres SVM	{'random_state': 0, 'dual': False}
evaluation	validation croisee avec k = 18

TABLE 12 – Les invariants des expérimentations

meilleur max_features Nous avons refait l'expérimentation sur les max_features afin de trouver le meilleur encore une fois, en utilisant la validation croisée avec k = 18. Cette fois-ci, le meilleur max_features trouvé était 10000 avec les scores données par le modèle LinearSVC dans Table 13.

Acc	F1	ROC-AUC	AP
0.8640	0.8648	0.9387	0.9387

TABLE 13 – Scores du meilleur max_feature (cross validation)

2.2.5 Stabilité de la validation croisée

Nous avons utilisé la validation croisée mais cette méthode est-elle stable et cohérente ? La validation croisée est stable si les performances obtenues après plusieurs répétitions différentes sont similaires et donc n'ont pas une variance remarquable. En utilisant StratifiedKFold avec des grains différents, nous avons examiné la variance entre les scores obtenus. Les deux tables suivantes montrent les résultats. Dans la table 14b, nous sommes intéressés par la ligne *std* qui nous donne l'écart type des scores obtenus au fur et à mesure. L'écart types de précision de ROC-AUC et de AP sont environ 0.001 lorsque ceux de Accuracy et F1 sont environ 0.003. Donc notre modèle est assez stable.

(a) Les métriques de performance avec grains différents

Grain	Acc	F1	ROC-AUC	AP
3	0.868 644	0.869 198	0.938 660	0.938 360
10	0.869 772	0.870 526	0.936 327	0.936 865
42	0.867 655	0.868 282	0.937 303	0.937 309
120	0.869 243	0.869 932	0.938 142	0.937 528
300	0.875 596	0.875 858	0.937 026	0.935 630

(b) Statistiques des métriques de performance

	Acc	F1	ROC-AUC	AP
count	5.000 000	5.000 000	5.000 000	5.000 000
mean	0.870 182	0.870 759	0.937 492	0.937 138
std	0.003 127	0.002 971	0.000 922	0.001 003
min	0.867 655	0.868 282	0.936 327	0.935 630
25%	0.868 644	0.869 198	0.937 026	0.936 865
50%	0.869 243	0.869 932	0.937 303	0.937 309
75%	0.869 772	0.870 526	0.938 142	0.937 528
max	0.875 596	0.875 858	0.938 660	0.938 360

TABLE 14 – Comparaison des métriques de performance

2.3 Evaluations sur le serveur de tests

Dans la table suivante, nous avons quelques résultats obtenus dans le serveur de tests. Les paramètres des tests sont fournis dans Table 40 dans l'annexe.

accuracy	precision	recall	f1	auc	ap	Status	Tags	Action
81.9040	84.6783	77.9040	81.1500	NaN	NaN	Ok	TEST_17	
82.3920	82.8866	81.6400	82.2586	NaN	NaN	Ok	TEST_16	
82.3080	83.1541	81.0320	82.0793	NaN	NaN	Ok	TEST_15	
80.7080	86.8060	72.4240	78.9655	NaN	NaN	Ok	NOTHING_WHATSOEVER	

TABLE 15 – Evaluations sur les données de test

2.3.1 Conclusions sur les tests sur le serveur

Nous observons que la classification NOTHING_WHATSOEVER a eu les pires scores mais étant donné qu'on n'a rien mis comme paramètre nous montre que Tfidf et les classificateurs sont eux même assez efficaces.

La seule différence entre TEST_16 et TEST_15 est que nous avons utilisé LinearSVC et LogisticRegression respectivement. Les scores sont extrêmement similaires mais LinearSVC a donné des meilleurs résultats sauf pour précision. Nous pouvons théoriquement conclure que les données de tests sont assez équilibrées car LinearSVC n'est pas le meilleur classifieur dans ce cas.

La seule différence entre TEST_16 et TEST_17 est le stemming fait dans TEST_16. Sans stemming, sauf la précision, les scores ont diminué. Cela nous montre que stemming permet le classifieur de différencier les sentiments plus facilement. Comme nous avons parlé avant, plusieurs mots indiquant un sentiment peuvent avoir la même racine. Donc en faisant du stemming, on les catégorise ensemble et on obtient un vocabulaire plus compact et cohérent.

Les scores obtenus dans les données de tests sont certainement pires que les résultats des tests créés à partir de la base d'entraînement. Ceci n'est pas surprenant car il y a de différence entre les deux sets et les paramètres qu'on trouve au cours du projet sur les données d'entraînement ne seront pas nécessairement en parfaite harmonie avec les données de test.

3 Reconnaissance du locuteur (Chirac/Mitterrand)

Nous avons trois notebooks chirac_mitterrand-analyse-pretraitement.ipynb, chirac_mitterrand-parametres.ipynb et chirac_mitterrand_ML.ipynb

3.1 Analyse des données et des prétraitements

Dans chirac_mitterrand-analyse-pretraitement.ipynb, nous explorons les données et faisons nos premières expérimentations sur les pré-traitements possibles.

3.1.1 Distribution des labels

Il y a deux classes afin de classifier les locuteurs :

classe 1 : Chirac

classe -1 : Mitterrand

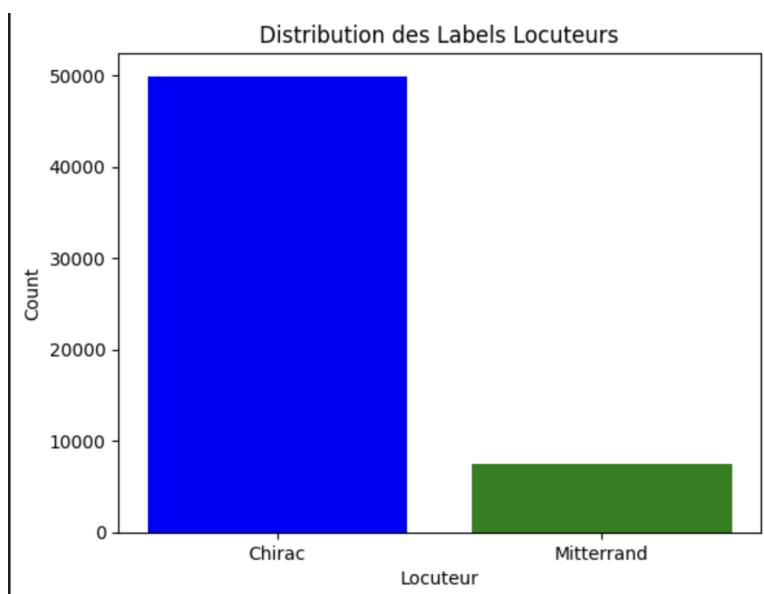


FIGURE 12 – Distribution des labels

Nous constatons que les labels sont très déséquilibrés. Il y a beaucoup plus de labels Chirac que Mitterrand.

3.1.2 Extraction du vocabulaire (BoW) et N-grammes

La taille d'origine du vocabulaire sans pre-processing est 28524 mots.

Sans aucun pré-traitement, la plupart des mots les plus fréquents sont des prépositions, des conjonctions et des pronoms. On voit aussi 'tous' 'tout' 'toute' 'toutes' qui sont le même mot en différentes formes. Ces problèmes peuvent être résolus avec le stemming et les stopwords.

100 mots les plus fréquents



FIGURE 13 – Nuage de mots plus fréquents

100 mots les plus discriminants au sens de odds ratio

Ici, on calcule odds ratio de chaque mot selon la différence de log vraisemblance de ce mot d'appartenir dans la classe positive (1) ou la classe négative (0). Si le taux de cotes est grand, alors il est plus vraisemblant que le mot appartient dans la classe positive. Donc dans ce nuage de mots, on voit les 100 mots les plus discriminants entre les classes.



FIGURE 14 – Nuage de mots plus discriminants odds ratio

Distribution d'apparition des mots (Zipf)

Ici, on observe la loi de Zipf sur les fréquences des mots. Selon cette loi, dans NLP, la fréquence d'un mot est inversement proportionnel à son rang. Donc le n-ième mot le plus fréquent va apparaître $(n+1)$ fois plus que $(n+1)$ -ième mot. $freq(nb_occ) = \frac{k}{nb_occ}$.

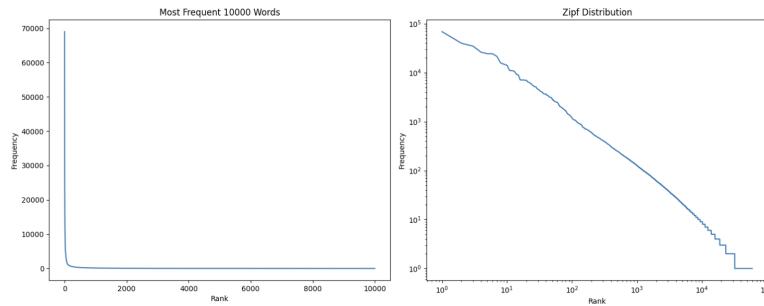


FIGURE 15 – Most frequent 100000 words vs Zipf Law

100 Bigrammes les plus fréquents

Voici les 100 bigrammes. La fréquence d'apparition de 'de la' est trop haute que les autres bigrammes.



FIGURE 16 – 100 Bigrammes les plus fréquent

100 trigrammes les plus fréquents



FIGURE 17 – 100 trigrammes les plus fréquents

Stopwords

Stopwords sont des mots trop utilisés d'une langue tel qu'ils n'apportent pas de contexte significatif à un texte. Donc les ignorer peut créer des résultats plus cohérents.

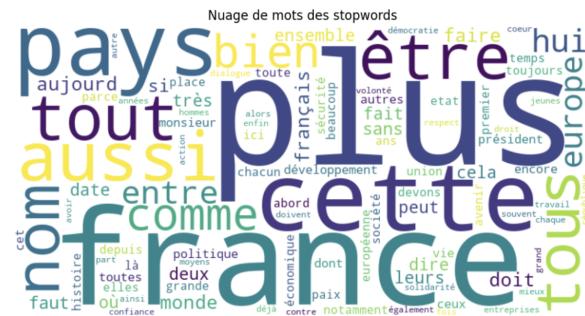


FIGURE 18 – les 100 mots les plus fréquents sans considérer les stopwords

3.1.3 Tests de BoW sur différents pré-traitements

Dans cette partie, on teste des différents pré-traitements mentionnés au dessus sans entrer dans détail. L'optimisation des paramètres est faite dans un autre notebook.

Pour chaque moyenne de prétraitement, on a deux variantes : CountVectorizer et TfidfVectorizer sans paramètres. Le but est de voir si les pré-traitement sont utiles dans ce contexte et de voir quel vectorizer nous donne des meilleurs résultats.

A partir des résultats, on a vu que les pré-traitements sauf stemming améliorent la prédiction. Dans le cas de stemming, appliquer ce traitement dans des données non prétraitées avait moins de chance à réussir. Un autre observation est que AP sur Mitterrand marche mieux avec CountVectorizer et a une

tendance à diminuer avec pré-traitement.

Tests des combinaisons possibles de pré-traitements :

test avec suppression de la ponctuation et des chiffres, transformation des mots entièrement en majuscule en marqueurs spécifiques, suppression des balises

Acc	F1	ROC-AUC	AP:
0.8943	0.5550	0.8685	0.7258

test avec suppression de la ponctuation et des chiffres, transformation des mots entièrement en majuscule en marqueurs spécifiques, suppression des balises, stemming

Acc	F1	ROC-AUC	AP:
0.8888	0.5194	0.8540	0.7304

On voit que lorsque F1 et ROC-AUC augmentent, AP diminue. Afin de trouver les meilleurs paramètres, il faut les tester avec des modèles et des paramètres de vectorizers différents.

3.1.4 Tests de Suréchantillonnage et de Sous-échantillonnage

OverSampling :

test avec suppression de la ponctuation et des chiffres, transformation des mots entièrement en majuscule en marqueurs spécifiques, suppression des balises

Acc	F1	ROC-AUC	AP:
0.8888	0.5194	0.8540	0.7304

test avec suppression de la ponctuation et des chiffres

Acc	F1	ROC-AUC	AP:
0.8693	0.5464	0.8594	0.7287

UnderSampling :

test avec suppression de la ponctuation et des chiffres, transformation des mots entièrement en majuscule en marqueurs spécifiques, suppression des balises

Acc	F1	ROC-AUC	AP:
0.7785	0.4774	0.8595	0.7298

test avec suppression de la ponctuation et des chiffres

Acc	F1	ROC-AUC	AP:
0.7791	0.4785	0.8604	0.7295

Après nos expérimentations, nous sommes arrivées aux conclusions suivantes :

- Stemming ne marche pas trop bien dans ce cas car la reconnaissance de locuteur est un processus compliqué et l'utilisation des différents formes des mots nous apporter de l'information sur le personnage. Quand on fait stemming, on perd cette information.
- Les autres pré-traitements sont utiles dans reconnaissance.
- Surapprentissage est plus précis sur des données d'entraînement que sous-échantillonage.

3.2 Expérimentations des paramètres

Dans chirac_mitterrand-parametres.ipynb, nous avons fait des expérimentations sur les effets des paramètres afin de trouver les meilleurs paramètres pour les vectorizers et les modèles.

3.2.1 Pour TfIdfVectorizer

vectorizer	TfidfVectorizer
paramètres vectorizer sauf l'expérimenté	{}
classifier	LogisticRegression
paramètres classifier	{'C' : 100.0, 'solver' : 'liblinear'}
preprocessor	remove_tags
	transform_uppercase
	accent_suppression
	chiffre_suppression
	punc_suppression
evaluation	train-test-split

TABLE 16 – Les invariants des expérimentations

sublinear_tf Nous avons vu que sublinear_tf=True était mieux que sublinear_tf=False.

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.5339	0.8533	0.7307

TABLE 17 – Résultats du test sublinear_tf

ngram_range

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.6433	0.9033	0.7315

TABLE 18 – Résultats du test avec ngram_range=(1,2)

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.6427	0.9080	0.7144

TABLE 19 – Résultats du test avec ngram_range=(1,3)

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.6401	0.9060	0.7149

TABLE 20 – Résultats du test avec ngram_range=(1,4)

Recherche des meilleurs paramètres pour TfIdfVectorizer

Nous avons utilisé une recherche exhaustive sur les combinaisons possibles des paramètres pour trouver la combinaison qui maximise la performance.

Nous avons appliqué de la sur-échantillonnage (RandomOverSampler) pour équilibrer les classes.

Nous avons encore utilisé LogisticRegression avec les mêmes paramètres.

L'évaluation a été fait avec le F1 score sur la classe minoritaire.

Paramètres explorés :

TABLE 21 – Paramètres de TfIdfVectoizer

Parameter	Values
stop_words	french None
max_df	0.5 0.75 1.0
min_df	2 3 5
ngram_range	(1, 3) (2, 3)
binary	True False
use_idf	True False
sublinear_tf	True False
max_features	None 1000 5000 10000

Le résultat :

Paramètre	Valeur
stop_words	None
max_df	0.5
min_df	2
ngram_range	(1, 3)
binary	True
use_idf	True
sublinear_tf	True
max_features	None

TABLE 22 – Paramètres obtenus

3.2.2 Paramètres des modèles

En utilisant GridSearchCV, nous avons testé les paramètres pour les modèles en utilisant pour TfIdfVectoizer les paramètres obtenus précédemment.

LogisticRegression

Paramètres explorés :

TABLE 23 – Paramètres de LogisticRegression

Paramètre	Valeurs
solver	liblinear
C	0.1
	1
	10
	100
penalty	l1 l2

En faisant 2 tests, 1 évaluant sur le F1 score sur la classe minoritaire et l'autre sur roc-auc, on a obtenu les mêmes paramètres.

Le résultat :

TABLE 24 – Paramètres obtenus pour LogisticRegression

Paramètre	Valeurs
solver	liblinear
C	10
penalty	l2

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.6411	0.9063	0.7147

TABLE 25 – Résultat avec les nouvelles paramètres de LogisticRegression et TfIdfVectoizer

Naive Bayes
Paramètres explorés :

TABLE 26 – Paramètres de NaiveBayes

Paramètre	Valeurs
alpha	np.linspace(0.5, 1.5, 6)
fit_prior	True False

Le résultat :

TABLE 27 – Paramètres obtenus pour NaiveBayes en maximisant F1 sur la classe minoritaire

Paramètre	Valeurs
alpha	0.5
fit_prior	True

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.5775	0.9081	0.7146

TABLE 28 – Résultat avec ces paramètres pour NaiveBayes

TABLE 29 – Paramètres obtenus pour NaiveBayes en maximisant roc auc

Paramètre	Valeurs
alpha	1.3
fit_prior	True

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.5466	0.9090	0.7143

TABLE 30 – Résultat avec ces paramètres pour NaiveBayes

On voit que les résultats avec LogisticRegression sont mieux qu'avec NaiveBayes.

XGBoost

On a utilisé RandomizedSearchCV pour XGBoost car GridSearchCV prenait longtemps.

Paramètres explorés :

TABLE 31 – Paramètres de XGBoost

Paramètre	Valeurs
subsample	0.6
	0.8
	1.0
min_child_weight	1
	5
	10
gamma	0.5
	1
	1.5
	2
	5
colsample_bytree	0.6
	0.8
	1.0
max_depth	3
	4
	5

On a obtenu les mêmes paramètres avec les tests pour maximiser F1 sur la classe minoritaire et roc-auc.

Le résultat :

TABLE 32 – Paramètres obtenus pour XGBoost

Paramètre	Valeurs
subsample	1.0
min_child_weight	5
gamma	1.5
colsample_bytree	0.6
max_depth	5

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.4816	0.8375	0.7362

TABLE 33 – Résultat avec ces paramètres pour XGBoost

LinearSVC

En expérimentant avec les paramètres de LinearSVC, nous avons vu que le meilleur résultat est obtenu avec les paramètres par défaut.

Dans nos données de train, nous avons obtenu un score F1 de 0.65 ; dans le serveur nous avons obtenu 0.6046 pour le F1, 0.8987 pour auc-roc et 0.6913.

LightGBM

On a utilisé RandomizedSearchCV.

Paramètres explorés :

TABLE 34 – Paramètres de LightGBM

Paramètre	Valeurs
num_leaves	5
	20
	30
	50
learning_rate	0.05
	0.1
	0.2
n_estimators	50
	100
	150

En maximisant le F1 score sur la classe minoritaire, nous avons obtenu les paramètres suivantes :

Paramètres explorés :

TABLE 35 – Paramètres obtenus pour LightGBM

Paramètre	Valeurs
num_leaves	30
learning_rate	0.2
n_estimators	150

F1 sur Mitterrand (minoritaire)	ROC-AUC	AP sur Mitterrand (minoritaire)
0.5441	0.8608	0.7284

TABLE 36 – Résultat avec ces paramètres pour LightGBM

3.3 Modèles de ML et métriques d'évaluation

3.3.1 Durée de classification selon la taille de vocabulaire et le classifieur

Dans cette partie, nous avons testé comment le paramètre max_features fait varier la durée de classification. On teste les valeurs de max_features = {100,1000,3000,5000,10000,taille_vocabulaire}, simultanément, on calcule la durée de l'apprentissage et la classification et les scores afin de trouver le meilleur max_features avec un pré-traitement.

vectorizer	TfidfVectorizer
paramètres vectorizer sauf max_feature	{}
classifiers	LogisticRegression + LinearSVC + MultinomialNB
paramètres LogisticRegression	{'C' : 100, 'solver' : 'liblinear'}
preprocessor	remove_tags
	transform_uppercase
	chiffre_suppression
	ponc_suppression
evaluation	validation croisée

TABLE 37 – Les invariants des expérimentations

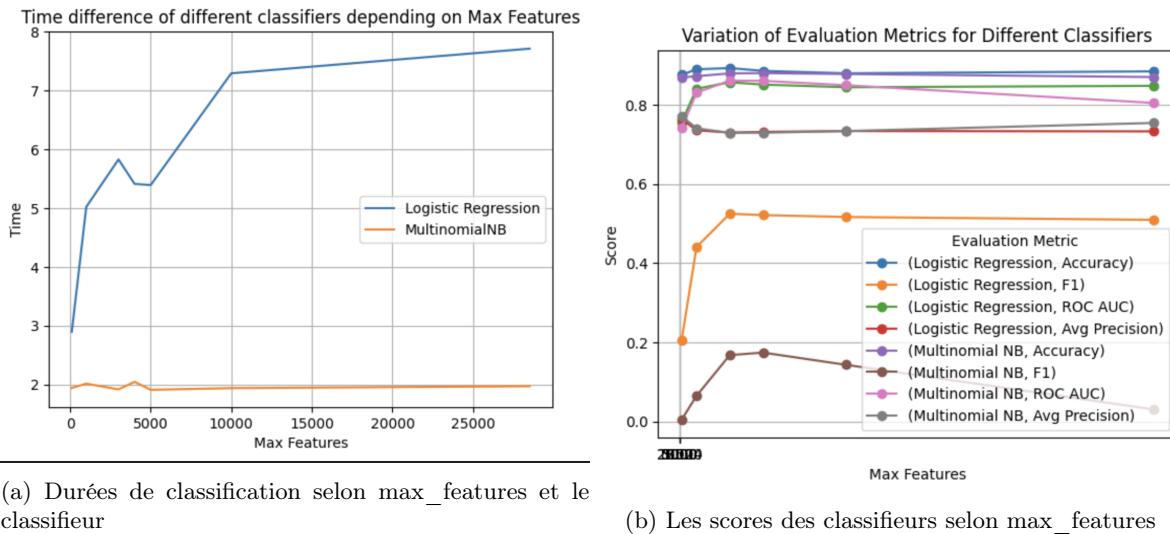


FIGURE 19 – Les résultats de l’expérimentation sur la taille de vocabulaire

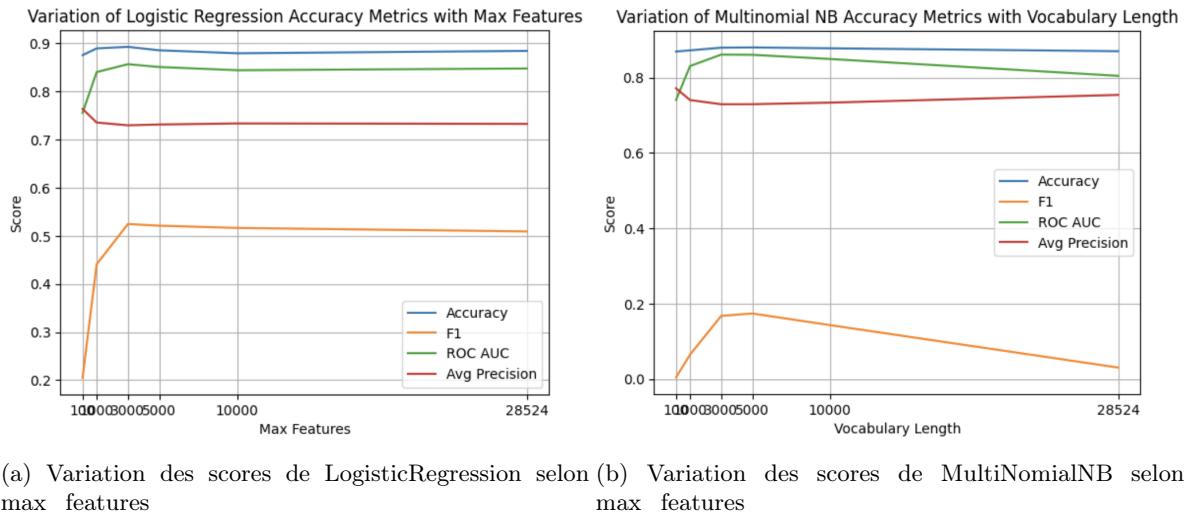


FIGURE 20 – Comparaison des modèles selon la taille de vocabulaire

Dans Figure 19a, nous avons un graphe de la variance de la durée de la classification selon le paramètre max_features et le classifieur. On voit que la durée n'est pas linéaire selon max_features pour LogisticRegression mais reste assez linéaire pour MultinomialNB, la durée de max_feature = 3000 est

supérieure à celle de 5000 pour tous les modèles de classification. Nous voyons que globalement, MultinomialNB est plus vite que LogisticRegression. Regardons Figure 19b. Les scores de LogisticRegression sont mieux que ceux de MultinomialNB sauf pour AP où ils sont très proches et celui de MultinomialNB est un peu mieux.

3.3.2 Mise en place de la validation croisée au lieu de train-test-split

Les paramètres et les choix durant la validation croisée sont idem aux ceux en dessus. Dans la figure en dessous, on a les scores obtenus grâce à la validation croisée. La différence entre crossval et train-test-split est négligeable.

	Accuracy	F1	ROC AUC	Avg Precision
Logistic Regression	0.884573	0.508127	0.848035	0.732752
Multinomial NB	0.869995	0.027619	0.801031	0.755628

FIGURE 21 – Les scores de l'évaluation avec la validation croisée

3.3.3 Recherche du meilleur k pour la validation croisée

Nous avons testé plusieurs valeurs de k pour la validation croisée en utilisant LogisticRegression comme modèle. Nous avons choisi le k maximisant accuracy et F1 sur la classe minoritaire.

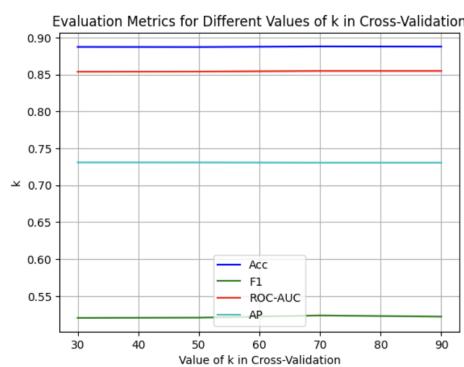


FIGURE 22 – Les scores de l'évaluation selon k dans la validation croisée

Le meilleur k trouvé est 70. Nous continuerons à la suite de cette partie avec k = 70 pour la validation croisée.

3.3.4 Stabilité de la validation croisée

En utilisant StratifiedKFold avec des grains différents, nous avons examiné la variance entre les scores obtenus. Les deux tables suivantes montrent les résultats. Dans la table 38b, nous sommes intéressées par la ligne *std* qui nous donne l'écart type des scores obtenus au fur et au mesure. L'écart types de précision de ROC-AUC et de AP sont environ 0.001 lorsque ceux de Accuracy et F1 sont environ 0.003. Donc notre modèle est assez stable.

(a) Les métriques de performance avec grains différents

Grain	Acc	F1	ROC-AUC	AP
3	0.8964	0.416 061	0.874 959	0.724 345
10	0.8964	0.416 061	0.874 959	0.724 345
42	0.8964	0.416 061	0.874 959	0.724 345
120	0.8964	0.416 061	0.874 959	0.724 345
300	0.8964	0.416 061	0.874 959	0.724 345

(b) Statistiques des métriques de performance

	Acc	F1	ROC-AUC	AP
count	5.000 000	5.000 000	5.000 000	5.000 000
mean	8.963 998	0.416 061	0.874 959	0.724 345 8
std	1.241 267	0.000 000	0.000 000	0.000 000
min	8.963 998	0.416 061	0.874 959	0.724 345
25%	8.963 998	0.416 061	0.874 959	0.724 345
50%	8.963 998	0.416 061	0.874 959	0.724 345
75%	8.963 998	0.416 061	0.874 959	0.724 345
max	8.963 998	0.416 061	0.874 959	0.724 345

TABLE 38 – Comparaison des métriques de performance

3.4 Evaluations sur le serveur de tests

Dans la table suivante, nous avons quelques résultats obtenus dans le serveur de tests. Les paramètres des tests sont fournis dans Table 41 dans l'annexe.

accuracy	precision	recall	f1	auc	ap	Status	Tags	Action
NaN	NaN	NaN	64.3866	89.8767	70.0321	Ok	TEST3	
NaN	NaN	NaN	60.4674	89.8742	69.8312	Ok	SVM	
NaN	NaN	NaN	55.5662	85.8915	60.2151	Ok	GBM	
NaN	NaN	NaN	63.0674	89.5281	69.0381	Ok	LOGREG AGAIN	

TABLE 39 – Evaluations sur les données de test

4 Conclusion

Au cours du projet, nous avons eu la chance de nous familiariser avec le modèle Bag of Words et ses applications dans le traitement automatique de la langue. Nous avons utilisé plusieurs composants de NLP comme CountVectorizer, TfidfVectorizer, LogisticRegression, LinearSVC, MultinomialNB. Cela nous a permis de pouvoir différencier leurs caractéristiques et leur domain d'application. Nous avons aussi expérimenté sur des différents pré-traitements et nous avons observé leur effets sur NLP. Nous avons testé plusieurs techniques d'évaluation de performance dans l'apprentissage : la validation croisée et la validation croisée K-fold qui sont robustes et train-test-split qui est facile et rapide.

5 Annexe

5.1 La distribution des fichiers

Chaque partie est dans son propre répertoire. Vous pourriez trouver des notebooks qui contiennent les tests que nous avons montré dans ce document et les codes sources qui sont écrits au fur et au mesure et donc ils sont un peu dense et compliqués (evaluation.py pour les fonction d'évaluation et d'apprentissage et utils_donnee.py pour les fonctions de chargement et de pré-traitement des données.). Dans predictions, il y a quelques résultats de classification faite sur les données de test.

5.2 Les paramètres des tests sur le serveur d'évaluation

tag	preprocessor	vectorizer	vectorizer params	classifier	classifier params
TEST_17	remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'stop_words': words('english') 'max_features':10000	LogisticRegression	'random_state': 0 'dual': False
TEST_16	stem remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'stop_words': words('english') 'max_features':10000	LinearSVM	?
TEST_15	stem remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'stop_words': words('english') 'max_features':10000	LogisticRegression	'random_state': 0 'dual': False
NOTHING WHATSOEVER	None	Tfidf	None	LogisticRegression	None

TABLE 40 – paramètres de l’expérimentation sur le serveur de tests - Movies

tag	preprocessor	vectorizer	vectorizer params	classifier & params
TEST3	remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'ngram_range' : (1,3)	LogisticRegression 'C' : 100.0 'solver' : 'liblinear'
SVM	remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'stop_words' : None 'max_features' : None 'binary' : True 'ngram_range' : (1, 3) 'use_idf' : True 'sublinear_tf' : True	LinearSVM 'random_state' : 42
GBM	remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'stop_words' : None 'max_features' :None 'ngram_range' : (1, 3) 'binary' : True 'use_idf' : True 'sublinear_tf' : True	LGBMClassifier 'num_leaves' : 30 'objective' : 'binary' 'learning_rate' : 0.2 'n_estimators' : 150
LOGREG AGAIN	remove_tags transform_uppercase chiffre_suppression ponc_suppression	Tfidf	'ngram_range' : (1,3) 'stop_words' : None 'max_df' : 0.5 'min_df' : 2	LogisticRegression 'C' : 10 'penalty' : 'l2' 'solver' : 'liblinear'

TABLE 41 – paramètres de l’expérimentation sur le serveur de tests - Presidents