Classification de texte.

SIHAMDI Mostefa, BOUSBA Abdellah

UE RITAL 2020-2021, Encadrant : Vincent Guigue

M1 DAC

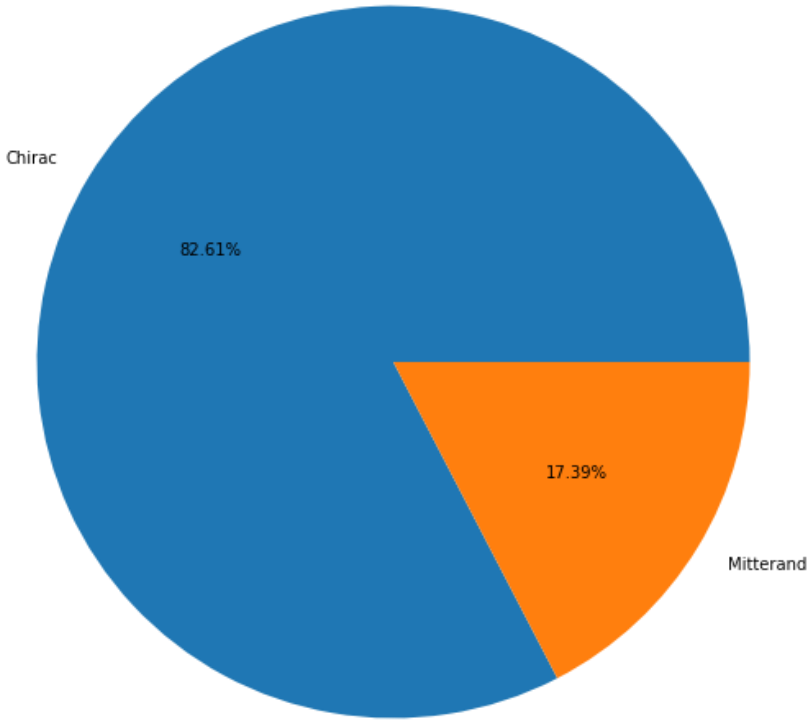
Introduction:

Ce rapport représente les campagnes d'expériences effectué afin d'optimiser les performances de classification de documents qui sont des discours de Jacques Chirac et François Mitterrand et des sentiments positif ou négatif sur les films. En comparant plusieurs méthodes de traitement du texte et algorithmes de classification.

Classification de discours présidentiel

I. Pré-traitement :

Avant d'effectuer des changements sur le texte lui-même on remarque un d'équilibrage entre les deux classes dans notre dataset train 82.61% Chirac et 17.39% Mitterrand ce qui ne nous permets pas à bien apprendre la classe minoritaire la solution la plus évidente est de faire un équilibrage par sous-échantillonnage sur la classe minoritaire, donc nos expériences seront sur notre base originale et une base équilibrée.



- Pour avoir une bonne estimation des performances de nos algorithmes nous avons choisi la métrique f1_score pour chaque classe (plus précisément celle de Mitterrand).

Equilibrage des donnes :

	Chirac Mean	Chirac Std	Mitterrand Mean	Mitterrand Std
SVM	0.934872	0.001887	0.487983	0.017170
Naive Bayes	0.931559	0.002962	0.518929	0.010911
Reg Logistique	0.941576	0.002016	0.488863	0.017523

Score F1 par cross validation sans équilibrage CountVectorizer

	Chirac Mean	Chirac Std	Mitterrand Mean	Mitterrand Std
SVM	0.742652	0.011267	0.738125	0.009267
Naive Bayes	0.769704	0.009202	0.776069	0.003987
Reg Logistique	0.768454	0.008353	0.760677	0.006033

Score F1 par cross validation avec équilibrage CountVectorizer

Combinaison optimale pour le traitement du texte :

	to_lower	keep_punct	stemming	keep_stopwords	f1_C	f1_M
0	1.0	1.0	1.0	1.0	0.748263	0.701984
1	1.0	1.0	1.0	0.0	0.738003	0.689421
2	1.0	1.0	0.0	1.0	0.752391	0.704109
3	1.0	1.0	0.0	0.0	0.736992	0.687667
4	1.0	0.0	1.0	1.0	0.747347	0.700097
5	1.0	0.0	1.0	0.0	0.733292	0.684401
6	1.0	0.0	0.0	1.0	0.752391	0.704109
7	1.0	0.0	0.0	0.0	0.736304	0.688104
8	0.0	1.0	1.0	1.0	0.748263	0.701984
9	0.0	1.0	1.0	0.0	0.740167	0.690609
10	0.0	1.0	0.0	1.0	0.765724	0.720292
11	0.0	1.0	0.0	0.0	0.761340	0.717465
12	0.0	0.0	1.0	1.0	0.747347	0.700097
13	0.0	0.0	1.0	0.0	0.735036	0.686184
14	0.0	0.0	0.0	1.0	0.765724	0.720292
15	0.0	0.0	0.0	0.0	0.755064	0.710939

GridSearch SVM CountVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_C	f1_M
0	1.0	1.0	1.0	1.0	0.763770	0.719341
1	1.0	1.0	1.0	0.0	0.751592	0.709445
2	1.0	1.0	0.0	1.0	0.769734	0.727888
3	1.0	1.0	0.0	0.0	0.755674	0.711143
4	1.0	0.0	1.0	1.0	0.764118	0.719379
5	1.0	0.0	1.0	0.0	0.749331	0.708363
6	1.0	0.0	0.0	1.0	0.769734	0.727888
7	1.0	0.0	0.0	0.0	0.750153	0.705144
8	0.0	1.0	1.0	1.0	0.763770	0.719341
9	0.0	1.0	1.0	0.0	0.756923	0.714664
10	0.0	1.0	0.0	1.0	0.786912	0.748188
11	0.0	1.0	0.0	0.0	0.776394	0.734931
12	0.0	0.0	1.0	1.0	0.764118	0.719379
13	0.0	0.0	1.0	0.0	0.752260	0.710096
14	0.0	0.0	0.0	1.0	0.786912	0.748188
15	0.0	0.0	0.0	0.0	0.773569	0.736211

GridSearch SVM TfidfVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_C	f1_M
0	1.0	1.0	1.0	1.0	0.773267	0.742249
1	1.0	1.0	1.0	0.0	0.762244	0.732706
2	1.0	1.0	0.0	1.0	0.779882	0.755379
3	1.0	1.0	0.0	0.0	0.768583	0.743164
4	1.0	0.0	1.0	1.0	0.774005	0.743317
5	1.0	0.0	1.0	0.0	0.761406	0.731765
6	1.0	0.0	0.0	1.0	0.779882	0.755379
7	1.0	0.0	0.0	0.0	0.765246	0.734906
8	0.0	1.0	1.0	1.0	0.773267	0.742249
9	0.0	1.0	1.0	0.0	0.765602	0.734954
10	0.0	1.0	0.0	1.0	0.791745	0.763886
11	0.0	1.0	0.0	0.0	0.788132	0.760962
12	0.0	0.0	1.0	1.0	0.774005	0.743317
13	0.0	0.0	1.0	0.0	0.771464	0.737993
14	0.0	0.0	0.0	1.0	0.791745	0.763886
15	0.0	0.0	0.0	0.0	0.786906	0.758507

GridSearch Naïve Bayes CountVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_C	f1_M
0	1.0	1.0	1.0	1.0	0.744124	0.744577
1	1.0	1.0	1.0	0.0	0.744074	0.739267
2	1.0	1.0	0.0	1.0	0.725051	0.736385
3	1.0	1.0	0.0	0.0	0.733819	0.732282
4	1.0	0.0	1.0	1.0	0.742414	0.742300
5	1.0	0.0	1.0	0.0	0.743231	0.735612
6	1.0	0.0	0.0	1.0	0.725051	0.736385
7	1.0	0.0	0.0	0.0	0.734900	0.730279
8	0.0	1.0	1.0	1.0	0.744124	0.744577
9	0.0	1.0	1.0	0.0	0.746924	0.742627
10	0.0	1.0	0.0	1.0	0.739336	0.748749
11	0.0	1.0	0.0	0.0	0.753367	0.751723
12	0.0	0.0	1.0	1.0	0.742414	0.742300
13	0.0	0.0	1.0	0.0	0.747598	0.740108
14	0.0	0.0	0.0	1.0	0.739336	0.748749
15	0.0	0.0	0.0	0.0	0.756970	0.752515

GridSearch Naïve Bayes TfidfVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_C	f1_M
0	1.0	1.0	1.0	1.0	0.773720	0.731282
1	1.0	1.0	1.0	0.0	0.759071	0.713246
2	1.0	1.0	0.0	1.0	0.779173	0.733627
3	1.0	1.0	0.0	0.0	0.766850	0.721885
4	1.0	0.0	1.0	1.0	0.772801	0.728863
5	1.0	0.0	1.0	0.0	0.754840	0.708081
6	1.0	0.0	0.0	1.0	0.779173	0.733627
7	1.0	0.0	0.0	0.0	0.763067	0.716614
8	0.0	1.0	1.0	1.0	0.773720	0.731282
9	0.0	1.0	1.0	0.0	0.763770	0.719341
10	0.0	1.0	0.0	1.0	0.790224	0.749879
11	0.0	1.0	0.0	0.0	0.782466	0.743849
12	0.0	0.0	1.0	1.0	0.772801	0.728863
13	0.0	0.0	1.0	0.0	0.765507	0.719533
14	0.0	0.0	0.0	1.0	0.790224	0.749879
15	0.0	0.0	0.0	0.0	0.774856	0.736538

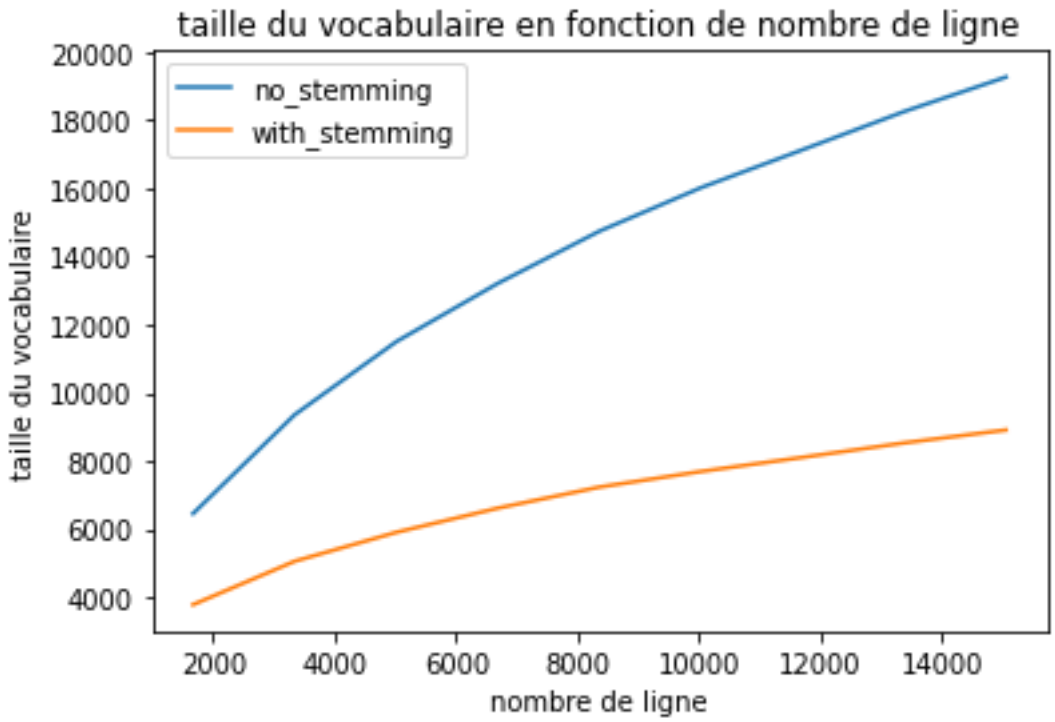
GridSearch RegLog CountVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_C	f1_M
0	1.0	1.0	1.0	1.0	0.776284	0.733495
1	1.0	1.0	1.0	0.0	0.762139	0.720039
2	1.0	1.0	0.0	1.0	0.770995	0.730342
3	1.0	1.0	0.0	0.0	0.766374	0.722559
4	1.0	0.0	1.0	1.0	0.780844	0.736585
5	1.0	0.0	1.0	0.0	0.756103	0.713701
6	1.0	0.0	0.0	1.0	0.770995	0.730342
7	1.0	0.0	0.0	0.0	0.760320	0.719404
8	0.0	1.0	1.0	1.0	0.776284	0.733495
9	0.0	1.0	1.0	0.0	0.766571	0.728058
10	0.0	1.0	0.0	1.0	0.785583	0.747407
11	0.0	1.0	0.0	0.0	0.783556	0.747057
12	0.0	0.0	1.0	1.0	0.780844	0.736585
13	0.0	0.0	1.0	0.0	0.760964	0.721649
14	0.0	0.0	0.0	1.0	0.785583	0.747407
15	0.0	0.0	0.0	0.0	0.778856	0.744086

GridSearch RegLog TfidfVectorizer

On peut clairement voir que la suppression des stopwords affecte le score négativement dans tout les modele sauf NaiveBayes avec tfidfVectorizer, ce qui peut être expliquer par le fait que le concept de stopwords peut varie d'une base a une autre, dans notre cas la suppression supprime des informations utiles pour la classification, de même pour les minuscules, par contre même si le stemming ne contribue pas trop pour le score on testera ensuite son effet sur la taille du vocabulaire pour décider ensuite si ça vaut le cout de le garder ou pas.

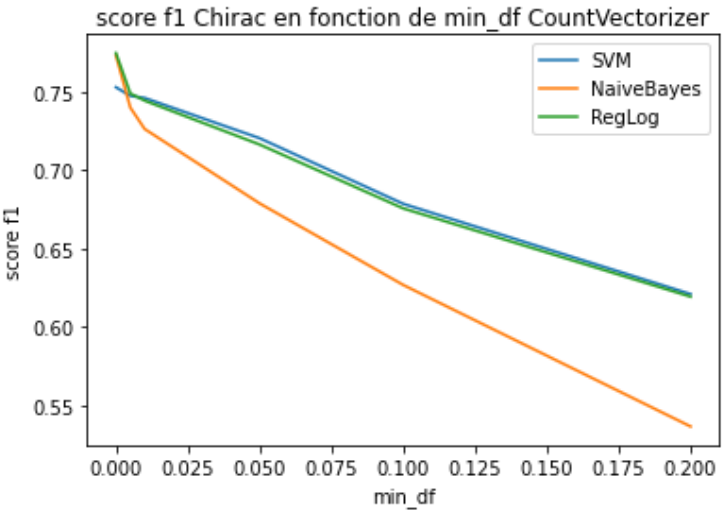
Effet du stemming :

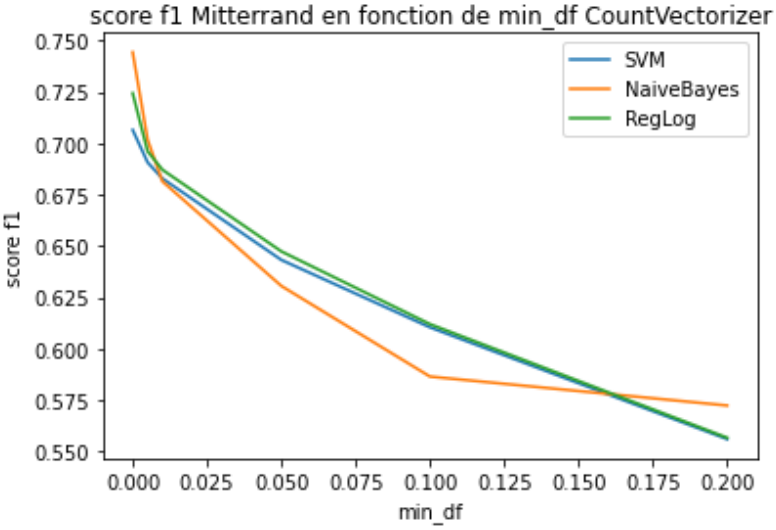


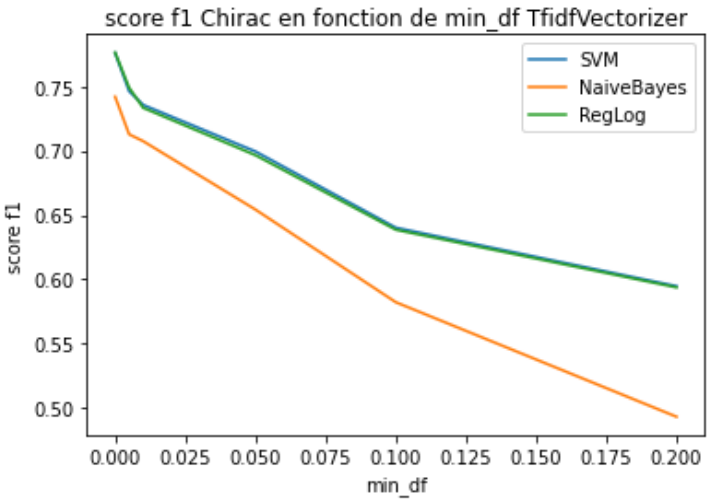
On remarque que le stemming diminue significativement la taille du vocabulaire ce qui nous permettra d'avoir un meilleur temps d'exécution.

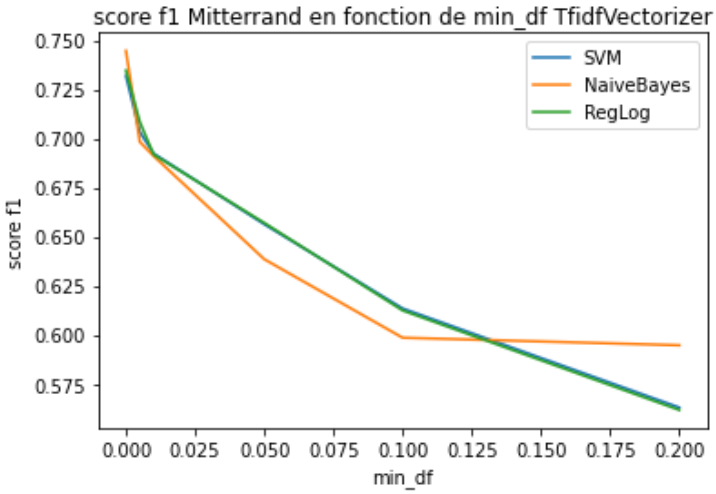
Fréquence documentaire des mots :

min_df :



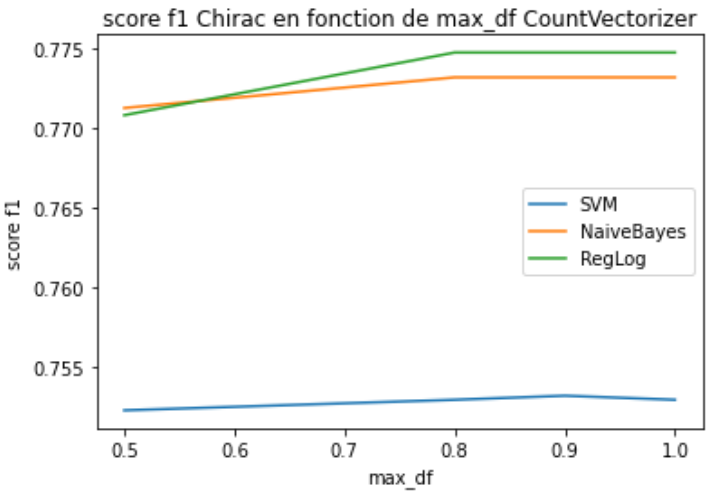


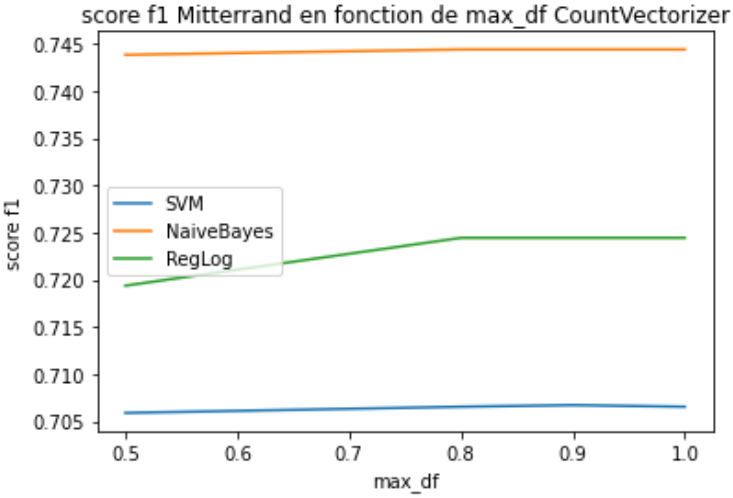


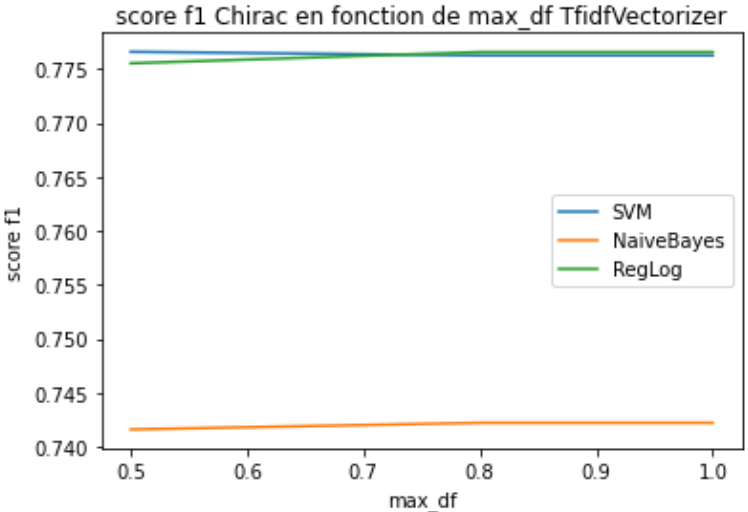


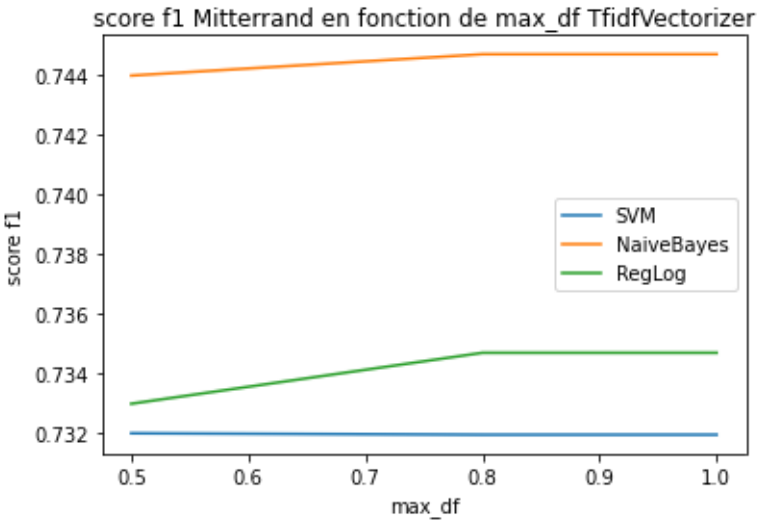
Apres avoir tester la suppression des mots en fonction de la fréquence documentaire minimum, on remarque des résultats similaires pour CountVectorizer et TfidfVectorizer, en plus ce traitement n'améliore pas le score.

max_df :



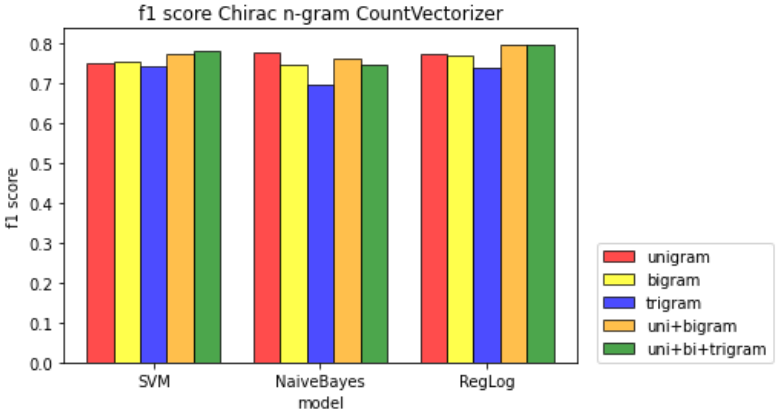


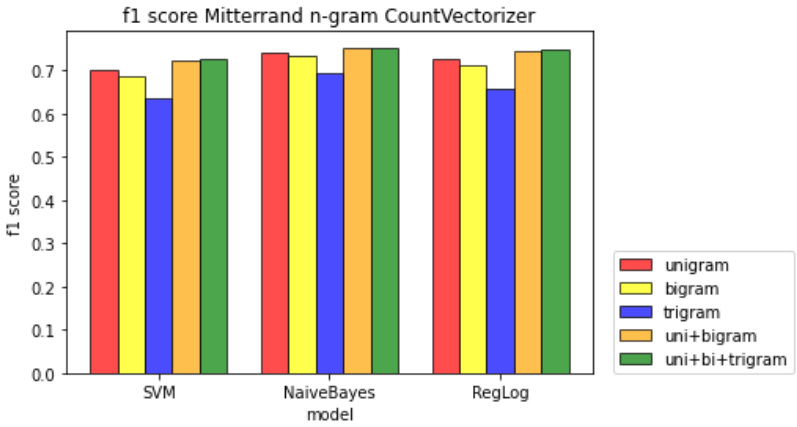


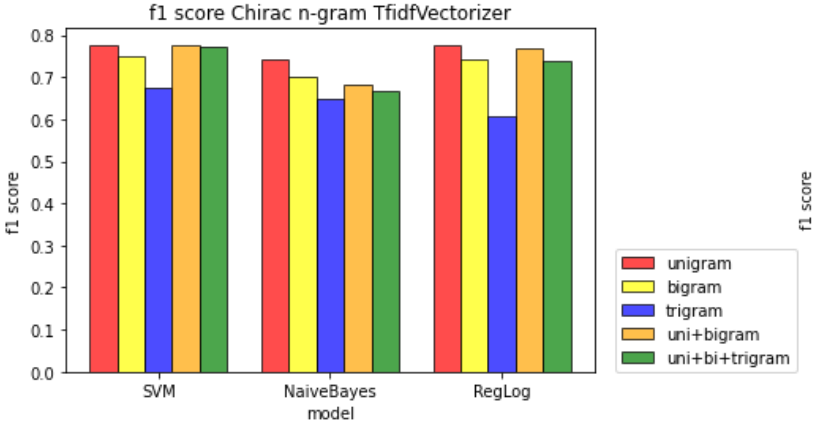


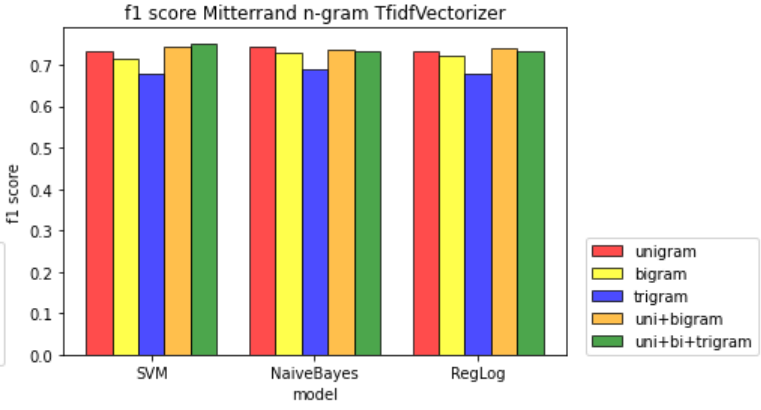
Apres avoir tester la suppression des mots en fonction de la fréquence documentaire maximum, on remarque des résultats similaires pour CountVectorizer et TfidfVectorizer sauf qu'on préfère utiliser CountVectorizer au lieu de TFIDF avec NaiveBayes, en plus pour le classe Mitterand on remarque que NaiveBayes a des résultats plus intéressante, la suppression des mots avec une grande fréquence documentaire ne contribue pas pour le score .

n-gram :





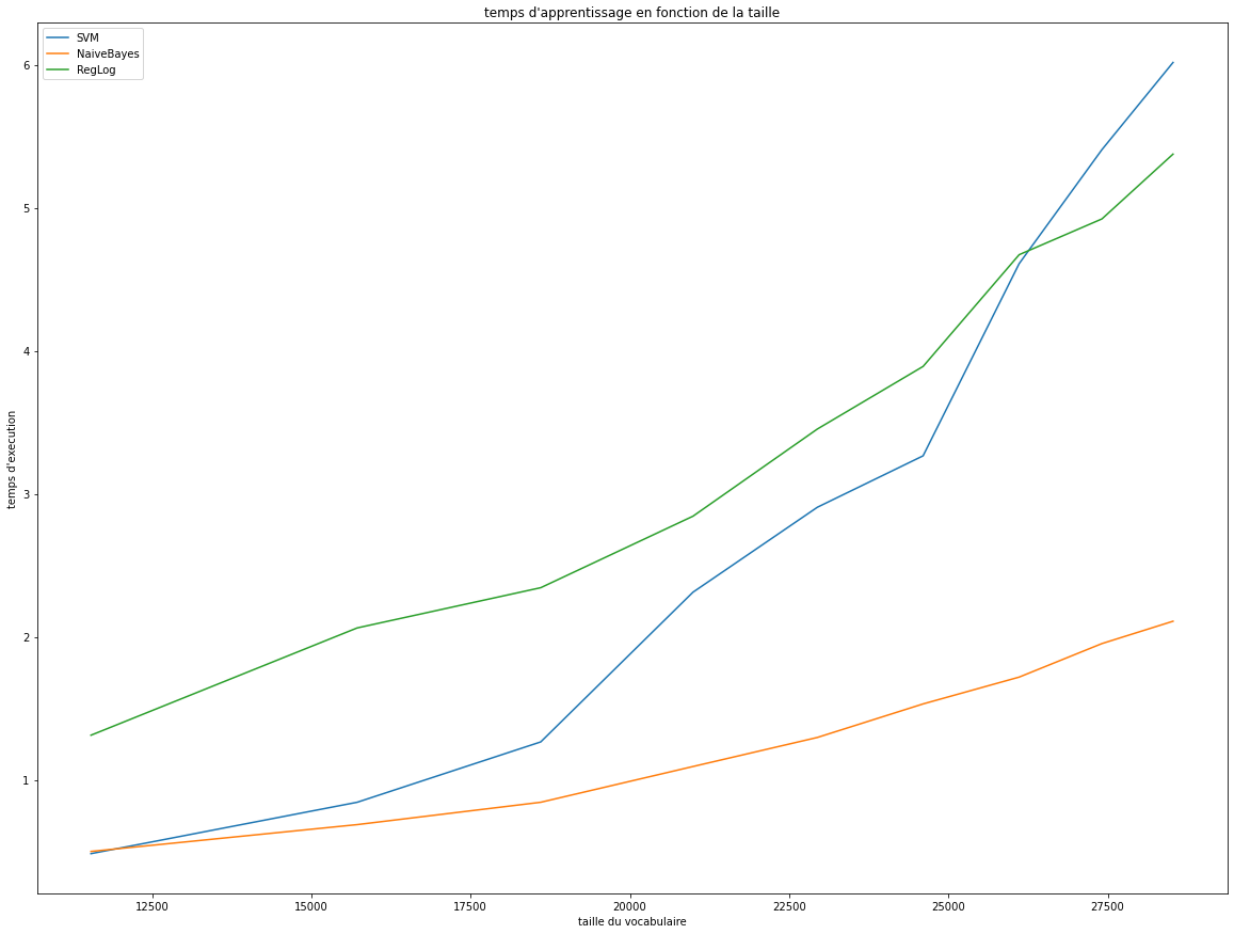




Concernant les tests au-dessus on ne remarque que chaque algorithme est affecter différemment, pour le **SVM** de la meilleurs combinaison est le mélange entre uni bi et trigram en plus le TfidfVectorizer retourne des resultats plus elevé, pour le **Naive Bayes** si on utilise CountVectorizer un melange entre uni et bigram parrait mieux, mais si on utilise TDFIDFVectorizer il suffit de prendre des unigram sachant que CountVectorizer est mieux approprié pour cette algorithme, et les memes remarques peuvent etre dites sur la **Régression Logistique**

II. Classifieur:

Temps d'execution:



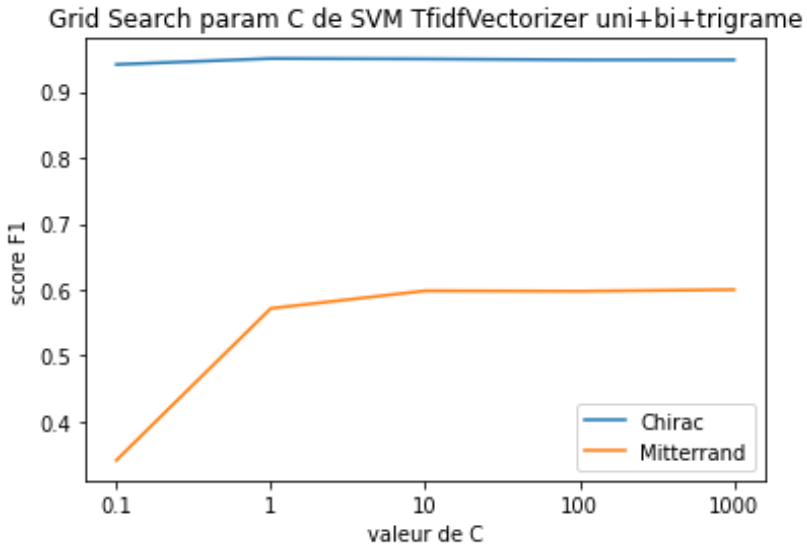
Apres avoir essayer plusieurs combinaisons on arrive à la conclusion que le modèle avec meilleurs résultats est :

- ***SVM : TfidfVectorizer + unigram+bigram+trigram***

Même si le modèle Naive Bayes est plus rapide, on a décidé d' utiliser le SVM dans le but de gagner de la précision au lieu du temps. Afin d'avoir une meilleure estimation de performance nous avons découper notre dataset train en 70% train et 30% test en regardant le id des documents au lieu d'utiliser une cross validation ou split train test avec shuffle pour garder la même structure des données. Et après exécution voici les résultats obtenus :

Apres avoir fait un équilibrage de train notre score f1_score Mitterrand : 0.478, Chirac : 0.852, par contre en laissant les données déséquilibrées on obtient un meilleur score f1_score **Mitterrand : 0.597,**

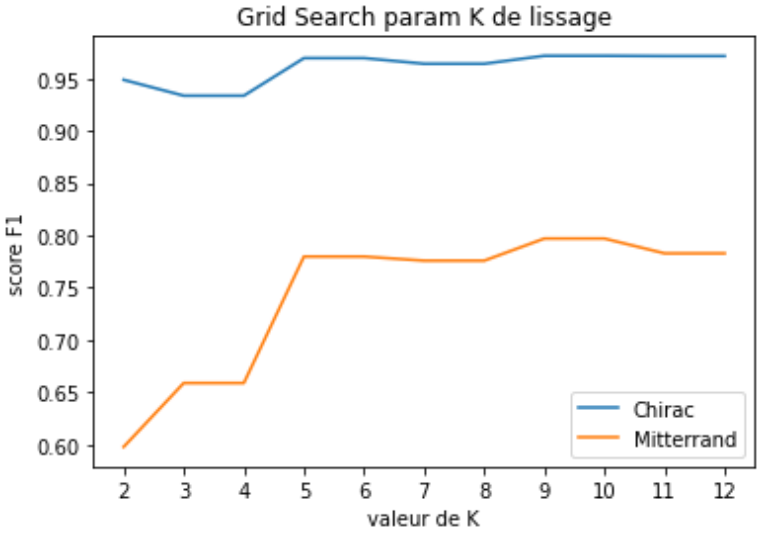
Chirac : 0.948, on peut expliquer ça par la taille des données train qui est très petite dans le cas d'équilibrage. Pour aller plus loin on peut s'en servir de nos connaissances sur le jeu de données pour faire un lissage sur les étiquettes.



Le changement du paramètre C n'affecte pas le score de Chirac mais pour C=100 on améliore bien le score.

III. Post- traitement:

Nous observons que les données sont sous formes de blocs. C'est-à-dire si un président parle, il dit plusieurs lignes à la fois, sachant que Chirac parle plus que Mitterrand, donc on fait une transformation basant sur les K plus proche voisin avec un poids de 2.5 pour le nombre des étiquettes Mitterrand.



En faisant un GridSearch pour le meilleur param de K on remarque que 9 était la meilleure valeur avec le nouveau score après lissage : **Mitterrand : 0.796, Chirac : 0.971.**

Détections de sentiments

Pré-traitement :

Cette fois nous avons une base de données équilibrer 1000 documents positifs et 1000 négatifs. Nous allons utiliser le f1 score moyen pour l'évaluation des modèles.

Combinaison optimale pour le traitement du texte :

	to_lower	keep_punct	stemming	keep_stopwords	f1_score
0	1.0	1.0	1.0	1.0	0.820062
1	1.0	1.0	1.0	0.0	0.835994
2	1.0	1.0	0.0	1.0	0.819297
3	1.0	1.0	0.0	0.0	0.831064
4	1.0	0.0	1.0	1.0	0.828210
5	1.0	0.0	1.0	0.0	0.833756
6	1.0	0.0	0.0	1.0	0.821965
7	1.0	0.0	0.0	0.0	0.832969
8	0.0	1.0	1.0	1.0	0.819902
9	0.0	1.0	1.0	0.0	0.835994
10	0.0	1.0	0.0	1.0	0.819297
11	0.0	1.0	0.0	0.0	0.831064
12	0.0	0.0	1.0	1.0	0.828210
13	0.0	0.0	1.0	0.0	0.833756
14	0.0	0.0	0.0	1.0	0.821965
15	0.0	0.0	0.0	0.0	0.832969

GridSearch SVM CountVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_score
0	1.0	1.0	1.0	1.0	0.854965
1	1.0	1.0	1.0	0.0	0.845894
2	1.0	1.0	0.0	1.0	0.858942
3	1.0	1.0	0.0	0.0	0.849257
4	1.0	0.0	1.0	1.0	0.861076
5	1.0	0.0	1.0	0.0	0.842841
6	1.0	0.0	0.0	1.0	0.856397
7	1.0	0.0	0.0	0.0	0.853680
8	0.0	1.0	1.0	1.0	0.854965
9	0.0	1.0	1.0	0.0	0.845894
10	0.0	1.0	0.0	1.0	0.858942
11	0.0	1.0	0.0	0.0	0.849257
12	0.0	0.0	1.0	1.0	0.861076
13	0.0	0.0	1.0	0.0	0.842841
14	0.0	0.0	0.0	1.0	0.856397
15	0.0	0.0	0.0	0.0	0.853680

GridSearch SVM TfidfVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_score
0	1.0	1.0	1.0	1.0	0.807540
1	1.0	1.0	1.0	0.0	0.805099
2	1.0	1.0	0.0	1.0	0.807059
3	1.0	1.0	0.0	0.0	0.806591
4	1.0	0.0	1.0	1.0	0.808616
5	1.0	0.0	1.0	0.0	0.807272
6	1.0	0.0	0.0	1.0	0.805388
7	1.0	0.0	0.0	0.0	0.809060
8	0.0	1.0	1.0	1.0	0.807540
9	0.0	1.0	1.0	0.0	0.805099
10	0.0	1.0	0.0	1.0	0.807059
11	0.0	1.0	0.0	0.0	0.806591
12	0.0	0.0	1.0	1.0	0.808616
13	0.0	0.0	1.0	0.0	0.807272
14	0.0	0.0	0.0	1.0	0.805388
15	0.0	0.0	0.0	0.0	0.809060

GridSearch NaiveBayes CountVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_score
0	1.0	1.0	1.0	1.0	0.796144
1	1.0	1.0	1.0	0.0	0.805893
2	1.0	1.0	0.0	1.0	0.795079
3	1.0	1.0	0.0	0.0	0.813194
4	1.0	0.0	1.0	1.0	0.794443
5	1.0	0.0	1.0	0.0	0.807710
6	1.0	0.0	0.0	1.0	0.794755
7	1.0	0.0	0.0	0.0	0.816911
8	0.0	1.0	1.0	1.0	0.796144
9	0.0	1.0	1.0	0.0	0.805893
10	0.0	1.0	0.0	1.0	0.795079
11	0.0	1.0	0.0	0.0	0.813194
12	0.0	0.0	1.0	1.0	0.794443
13	0.0	0.0	1.0	0.0	0.807710
14	0.0	0.0	0.0	1.0	0.794755
15	0.0	0.0	0.0	0.0	0.816911

GridSearch NaiveBayes TfidfVectorizer

	to_lower	keep_punct	stemming	keep_stopwords	f1_score
0	1.0	1.0	1.0	1.0	0.832368
1	1.0	1.0	1.0	0.0	0.845270
2	1.0	1.0	0.0	1.0	0.834245
3	1.0	1.0	0.0	0.0	0.844923
4	1.0	0.0	1.0	1.0	0.834339
5	1.0	0.0	1.0	0.0	0.841426
6	1.0	0.0	0.0	1.0	0.835547
7	1.0	0.0	0.0	0.0	0.842890
8	0.0	1.0	1.0	1.0	0.832368
9	0.0	1.0	1.0	0.0	0.845270
10	0.0	1.0	0.0	1.0	0.834245
11	0.0	1.0	0.0	0.0	0.844923
12	0.0	0.0	1.0	1.0	0.834339
13	0.0	0.0	1.0	0.0	0.841426
14	0.0	0.0	0.0	1.0	0.835547
15	0.0	0.0	0.0	0.0	0.842890

GridSearch RegLog CountVectorizer

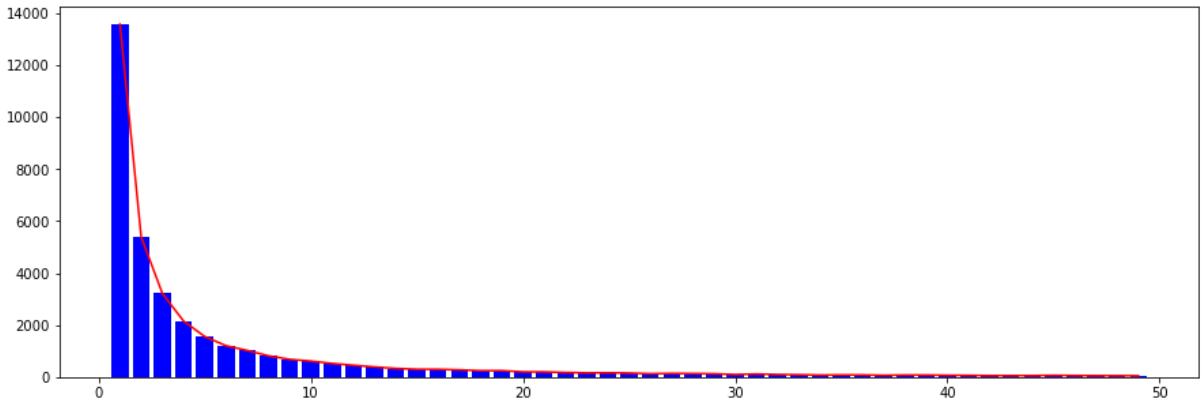
	to_lower	keep_punct	stemming	keep_stopwords	f1_score
0	1.0	1.0	1.0	1.0	0.826082
1	1.0	1.0	1.0	0.0	0.834047
2	1.0	1.0	0.0	1.0	0.821725
3	1.0	1.0	0.0	0.0	0.827519
4	1.0	0.0	1.0	1.0	0.825161
5	1.0	0.0	1.0	0.0	0.837335
6	1.0	0.0	0.0	1.0	0.823727
7	1.0	0.0	0.0	0.0	0.827544
8	0.0	1.0	1.0	1.0	0.826082
9	0.0	1.0	1.0	0.0	0.834047
10	0.0	1.0	0.0	1.0	0.821725
11	0.0	1.0	0.0	0.0	0.827519
12	0.0	0.0	1.0	1.0	0.825161
13	0.0	0.0	1.0	0.0	0.837335
14	0.0	0.0	0.0	1.0	0.823727
15	0.0	0.0	0.0	0.0	0.827544

GridSearch RegLog TfIdfVectorizer

Remarques :

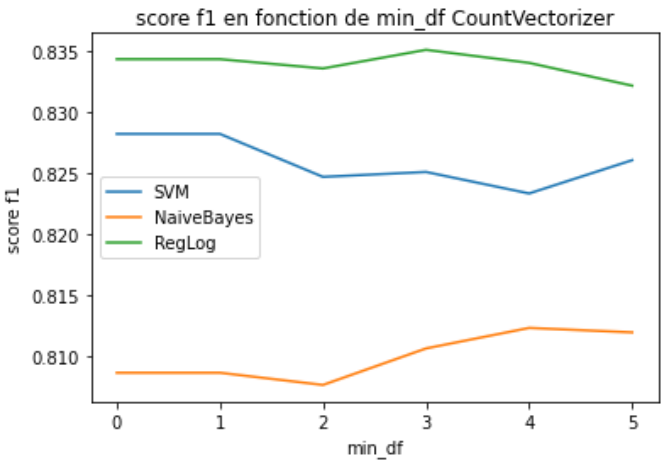
- Chaque modèle se comporte différemment avec les combinaisons
- Contrairement a la base Chirac et Mitterrand la transformation en minuscule n'affecte pas le score ce qui est logique car on travail sur les significations des mots et pas la manière d'écriture.
- Cette fois la suppression des stopwords affecte positivement le score sauf avec le modèle SVM TfIdfVectorizer.
- Il est préférable d'utiliser le stemming pour SVM et RegLog mais pas pour NaiveBayes.

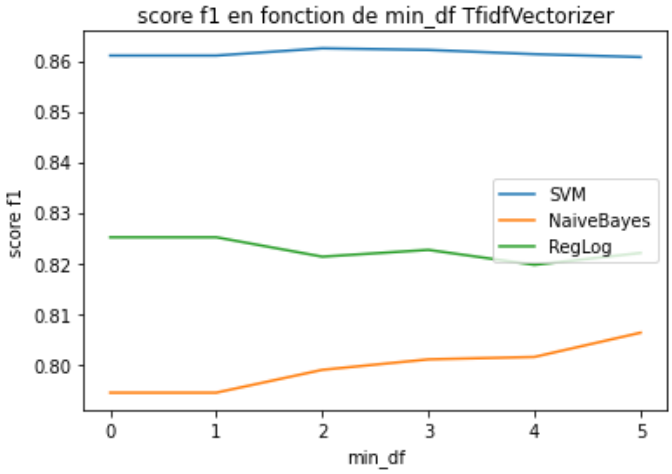
Fréquence documentaire des mots :



Nombre de mots par fréquence documentaire

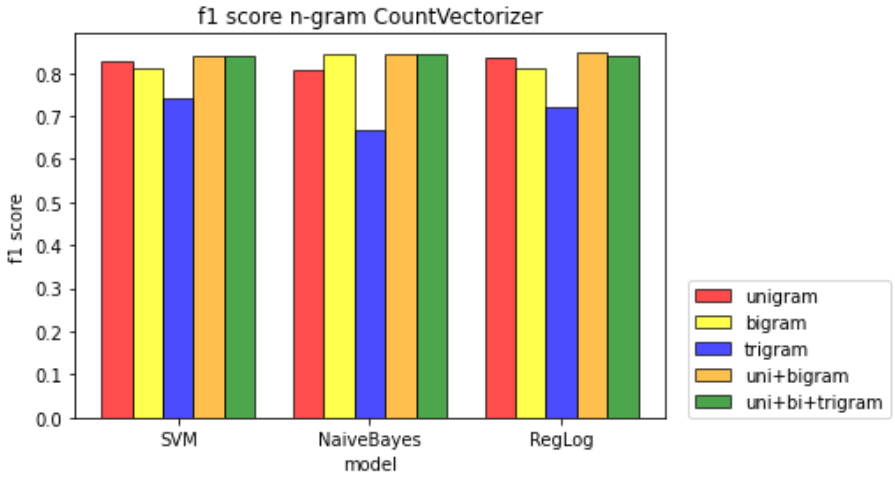
La majorité des documents ont une petite frequence documentaire, nous allons etudier l'effet de suppression de ces mots sur le score.

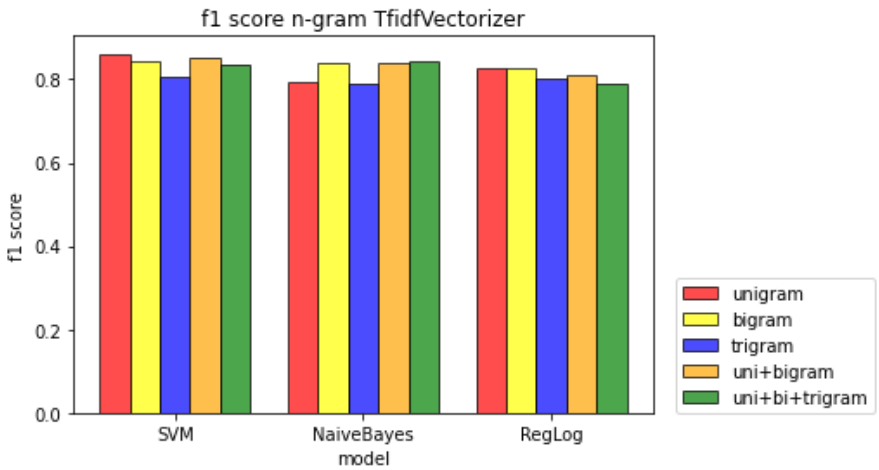




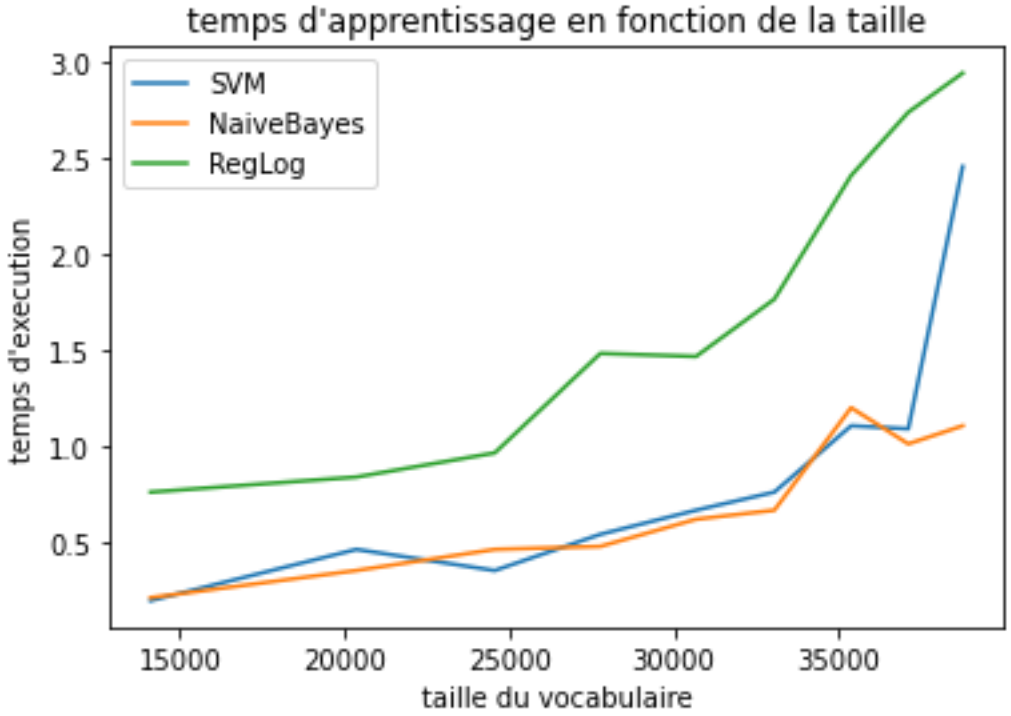
On peut voir que la suppression des mots rares n'affecte pas trop le score surtouts avec l'utilisation de TFidfVectorizer.

ngram :





D’après les résultats au-dessus si on utilise CountVectorizer un mélange entre unigram et bigram est le meilleur choix, par contre pour TfidfVectorizer des bigram suffit pour Naive Bayes et RegLog mais concernant le SVM qui a le meilleur score on préfère les unigram.



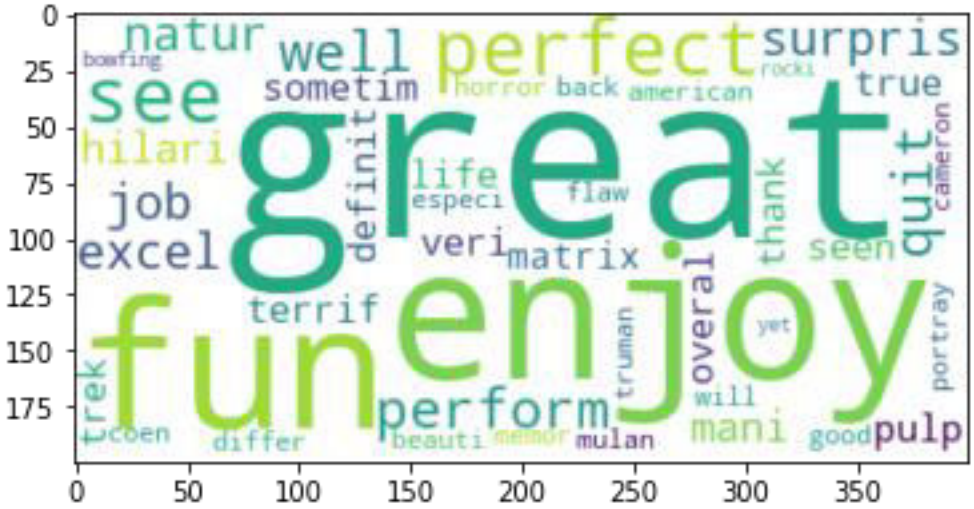
On remarque qu’à chaque fois que la taille du vocabulaire augmente le temps d’exécution augment, et aussi la régression logistique prendre beaucoup de temps que les autres modèles.

Apres avoir essayer plusieurs combinaisons on arrive à la conclusion que le modèle avec meilleurs résultats est :

- ***SVM : TfidfVectorizer + unigram (avec stemming et en gardant les stopwords)***

Meilleur score : **0.86107**

Termes discriminants positifs :



Termes discriminants negatifs :

Conclusion :

Apres avoir tester plusieurs choses on arrive à la conclusion que travailler avec des données déséquilibrer est un vrai challenge, nous avons retenu que les stopwords ne sont pas toujours considéré comme bruit, les lettres majuscules contient aussi de l'information, le stemming n'apporte pas une grande différence mais diminue la taille du vocabulaire. De plus, le choix du classifieurs est très important après les hyperparamètre les plus optimaux et la représentation correcte des données. Enfin la connaissance du format de notre dataset est très importante comme on a vu que grâce au post-traitement le score à augmenter significativement.