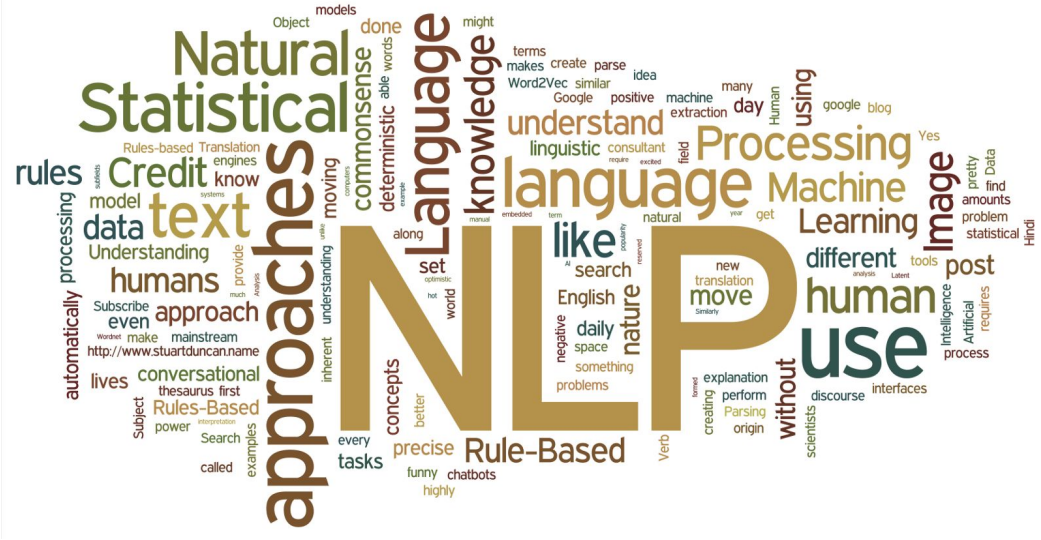
TAL

Classification de textes



I Reconnaissance Chirac / mitterrand

Introduction

Analyse du dataset :

Le but est d'entraîner un modèle à reconnaître si des phrases proviennent d'un discour prononcé par Chirac ou par Mitterrand.
On a un problème de classification à 2 classes avec les répartition suivante:

Chirac	49890	87%
Mitterand	7523	13%

Méthode de classification :

On transforme le texte en bag of words.
Grâce à cette transformation, toutes les phrases des discours sont sous forme de vecteurs de même dimension.
On peut alors utiliser des classifier classiques.

Problématiques :

→ Pre-processing :
    Comment transformer en bag-of-words ?
→ Gestion du déséquilibre des classes dans les données.
    → Comment adapter l'entrainement ?
    → Comment bien choisir la métrique ?

Imbalanced classes :

Test naïf

Supposons que l'on teste notre modèle multinomial sans pré-traitements particulier.
On a toutefois pensé désactiver l'apprentissage des probabilité des classe (fit_prior).

On peut trouver les scores suivants par cross-validation :

Precision	recall	f1_macro	roc_auc
0.94	0.86	0.70	0.84

Mais si on regarde la précision pour chacunes des classes séparément :

Accuracy chirac	Accuracy Mitterand
0.85	0.68

On remarque alors 2 choses :
→ Certaines métriques sont trompeuse et donne l'apparence de bonne performances.
→ Le modèle n'a pas beaucoup appris donne souvent la classe majoritaire.

Conclusion et choix sur les métriques :
Nous regarderons étudierons la précision des 2 classes séparément pour s'assurer que le modèle apprends.
Lorsqu'il ne sera pas commode de regarder 2 chiffres nous utiliserons le score f1_macro, car c'était le plus pessimiste.
Résumé des résultats :

f1_macro	Chirac	Mitterand
0.70	0.85	0.68

Comment améliorer l'apprentissage ?

On peut tenter de rééquilibrer les classes avant apprentissage. Par downsampling ou par up sampling.
On peut essayer des modèles qui sont moins sensibles aux déséquilibres entre classes.

Upsampling

Pour que l'apprentissage du modèle ne se résume pas à donner à principalement la classe majoritaire, on peut augmenter les échantillons de données de la classe minoritaire.
Lorsqu'on utilise cette technique il faut être particulièrement prudent lors d'un test de cross-validation pour éviter le modèle utilise les données à prédire lors de son apprentissage.
Nous avons utilisé SMOTE sur les données vectorielles sans pré-traitement.

Toutefois suite à ces test nous remarquons que tous les métriques chutent :

f1_macro	Chirac	Mitterand
0.58	0.77	0.48

La génération des données supplémentaire par SMOTE ne semblent pas efficace.
L'échec de cette technique peut être lié à l'inefficacité de SMOTE lorsque qu'il gèrent des données en grandes dimensions.

Downsampling

Avec le downsampling, on se contente de retirer des données de la classe majoritaires avant l'apprentissage. Suite on tests, on remarque que la justesse des prédictions devient plus équilibrée :

f1_macro	Chirac	Mitterand
0.77	0.74	0.81

Arbres de décision :

On peut utiliser des modèles d'arbre de décision dont le principe est basé sur la réduction d'entropie qui est considérée comme moins sensible aux déséquilibres des classes.

Nous avons testé le modèle RandomForest , mais les résultats de score f1 étaient toujours inférieurs à 0.7 peut importe les paramètre du modèle ou et les méthode de sampling utilisées. Et cela pour des temps d'apprentissage bien plus longs.

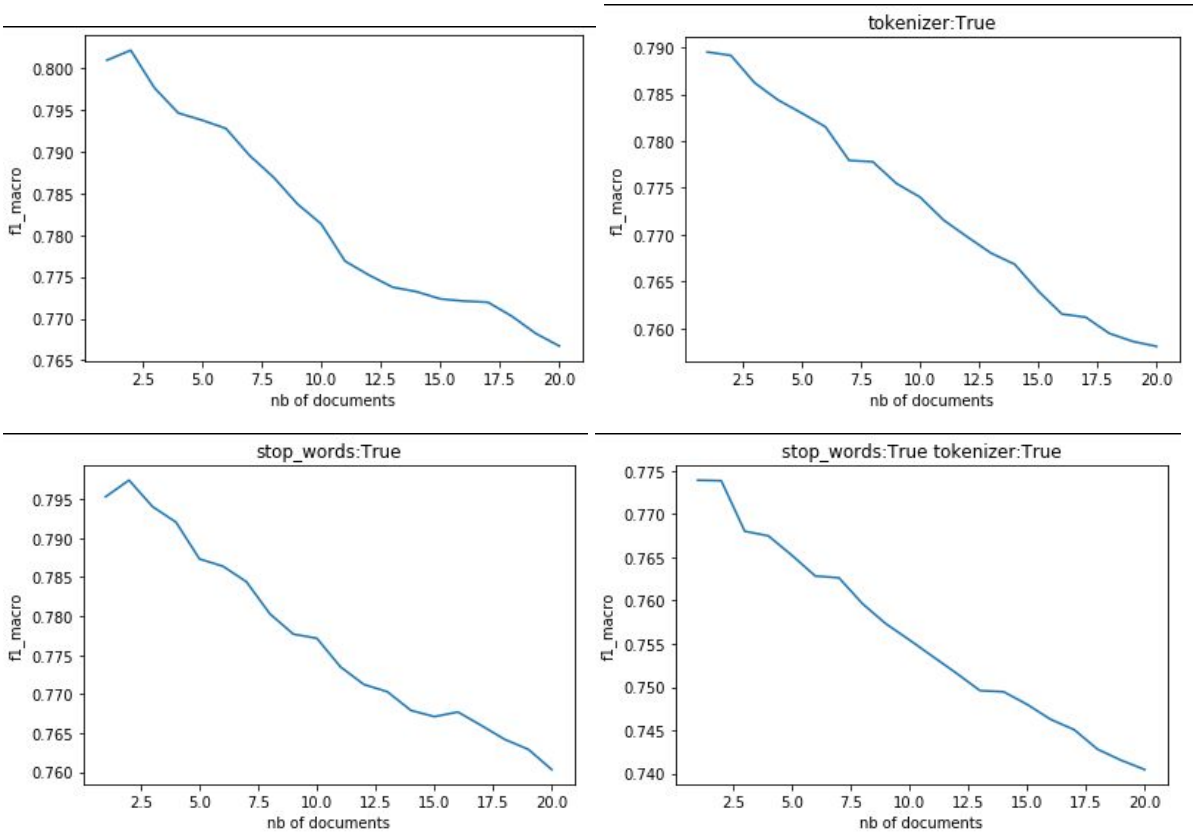
Pre-processing :

Dans les test suivant nous avons utilisé la méthode de downsampling car c'est celle qui produisait les meilleurs résultats à l'étape précédente.
On peut choisir de garder ou non les stops-words, en supposant qu'il font partie du style du locuteur.On peut aussi essayer de prendre uniquement la racine des mots.

On peut aussi faire des choix sur la composition du vocabulaire

- Nous avons choisit de prendre des n-gram et 1 et 2 mots seulement.
- Finalement on peut garder les mots qui sont présents dans plus de X documents.

Nous avons testé chacunes de ces alternatives en augmentant progressivement le nombre de document requis pour qu'un mot soit comptabilisé :



Nous remarquons que les score augmentent légèrement lorsqu'on retire les mots qui ne sont présents que dans 1 seul document. Par contre, il ont tendance à s'écrouler si on retire d'autres mots peu fréquents.

Il ressort de ces tests que ni la lemmatisation ni l'utilisation ou non de stop-words n'a d'impact fort sur les score.
Toutefois les scores sans pré-traitements semblent meilleurs.

Conclusion

Suit aux différents tests, la méthode la plus efficace est la suivante :
→ Downsampling des données, ce qui est problématique car nous ne disposons pas d'une grosse base de données.
→ Peu de preprocessing : retirer uniquement les mots présents dans moins de 2 documents.

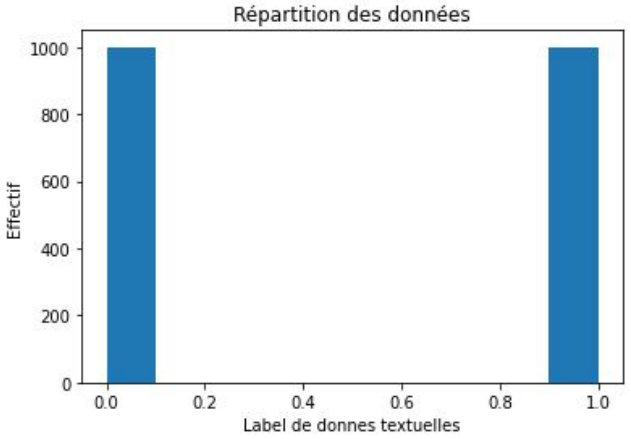
Nous n'avons pas d'optimisation par grid-search à faire étant donné que le modèle testé(naïve bayes multinomial) n'a pas d'hyper parameters.

II Détection de sentiments

Objectif :

Dans cette partie on s'intéresse à la classification des avis des internautes selon le sentiment qu'ils portent,On dispose d'une base de données textuelle labellisée divisée en deux classes de sentiments (positifs et négatifs) ,
L'objectif est donc de classifier un ensemble de textes non labellisés. Nous sommes donc ici confrontés à un problème de classification textuelle.

Prise en main des données :

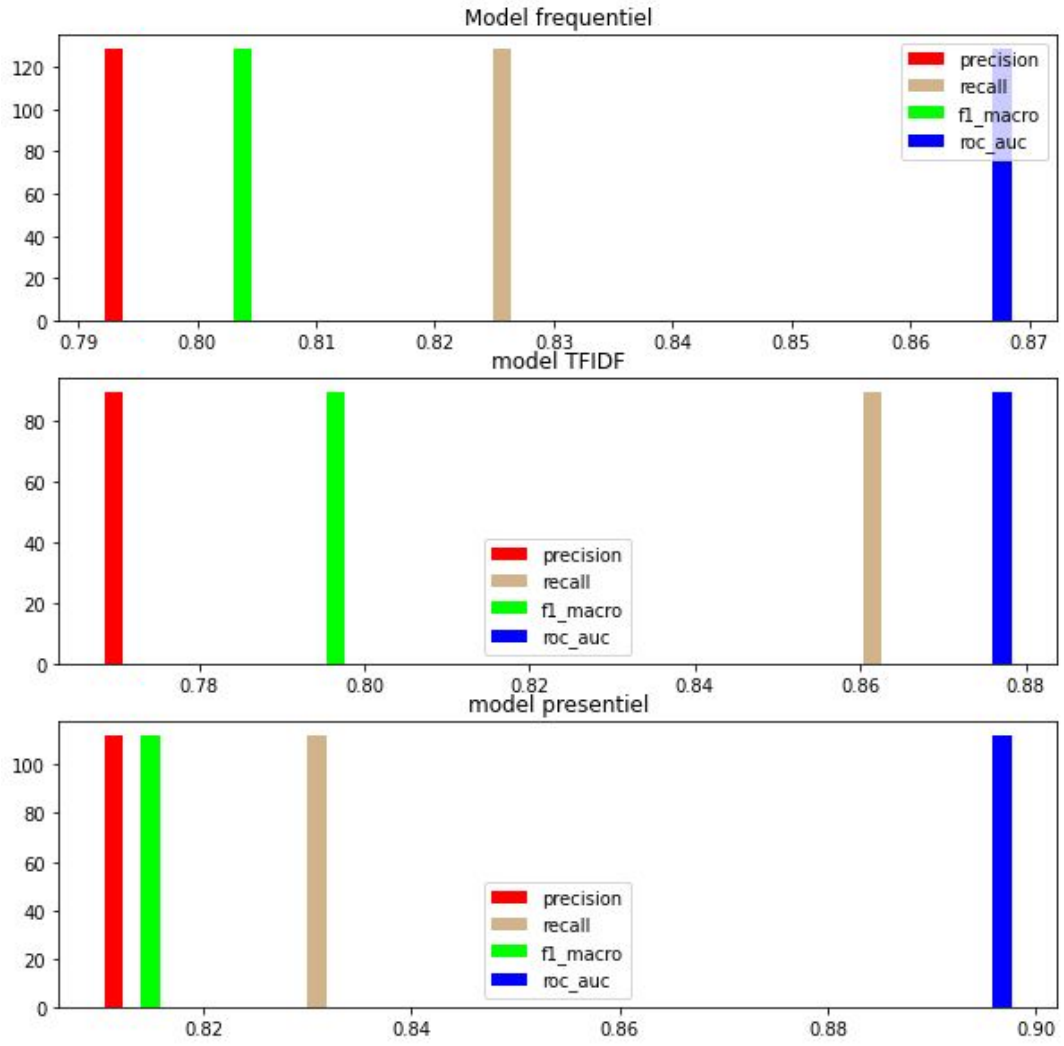


La figure ci-dessus montre la répartition des avis, On remarque que cette fois-ci, les données sont parfaitement équilibrées. Nous pouvons donc passer directement à l'optimisation de paramètres.
Pour optimiser les paramètres, nous pouvons cette fois utiliser la métrique d'accuracy (précision) qui semble mieux adaptée au vu de la répartition des données.

Pré-traitement de données

Comme précédemment, il est possible de faire une préparation des données
Cette fois-ci, le pré-traitement des données augmente les performances de tout les modèles. On obtient 82% d'accuracy ,en moyennes, avec les données brutes et 85.96% avec des données pré-traitées (stemming,tokenization, lettres minuscules, stopwords),
Nous travaillerons donc avec des données pré-traitées dans la suite de cette partie.

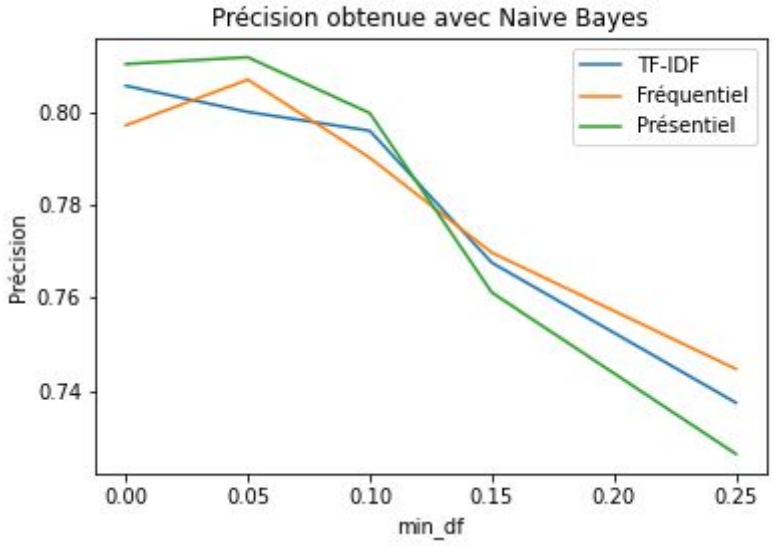
Représentation vectorielle des données et optimisation des paramètres :

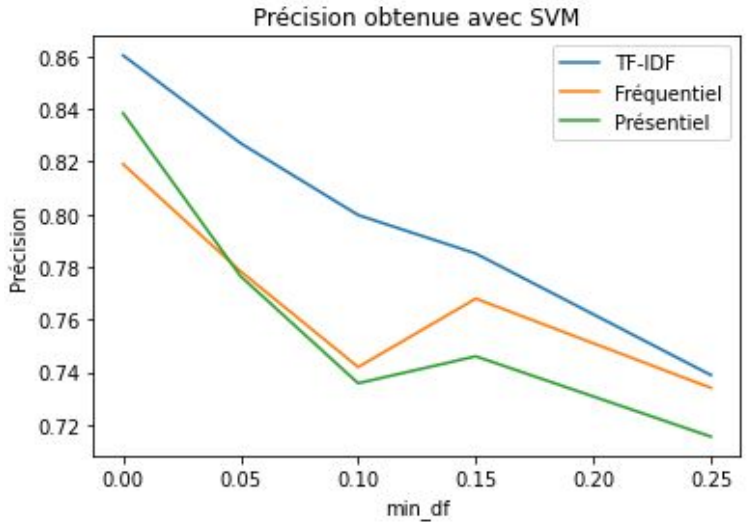


Pour représenter les données, on peut utiliser une représentation présentielle, fréquentielle ou TFIDF pour vectoriser les données textuelles en un dictionnaire de mots. Dans cette partie, nous allons comparer ces trois représentations lorsque différents paramètres de vectorisation varient.
La figure ci-dessus montre les différentes valeurs des métriques de mesures de performance obtenu en lançant le classifieur MultinomialNB (Naive Bayes) sur les trois type de vecteurs de mots,

Dans la suite nous allons tester les deux classifieurs SVM et NAIVE BAYES sur les trois type de dictionnaire de mot (fréquentiel,présentiel,TFIDF), Nous allons mesurer les scores des deux modèles en utilisant la métrique «précision » qui semble la plus adapté au problème,
Toutes les courbes qui vont être présentées dans la suite sont construites par validation croisée à 5 folds

1) Le premier paramètre testé est min df qui sert à éliminer les mots les moins fréquents à un seuil donné:

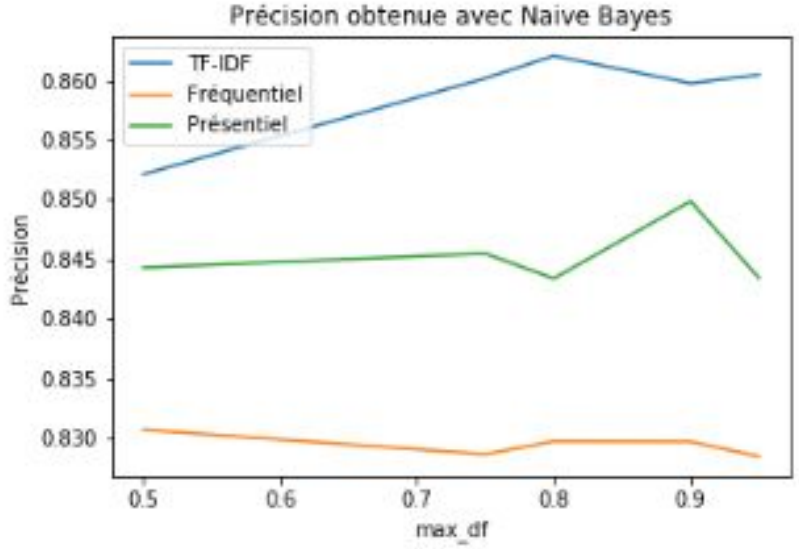


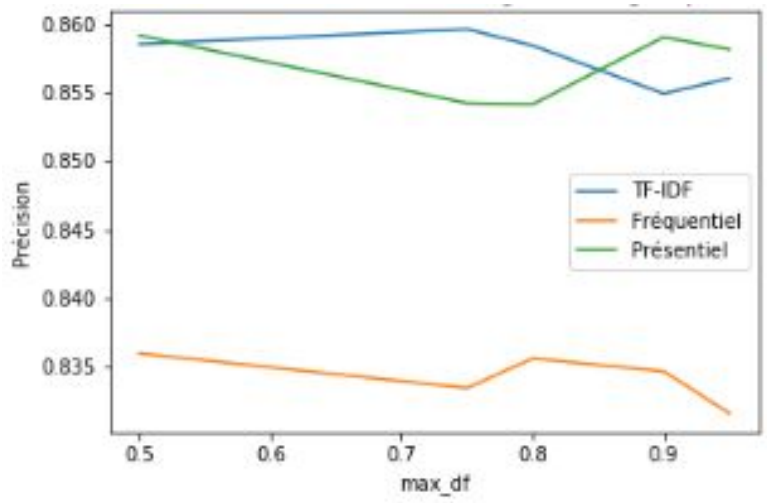


Les courbes montrent que le même résultat pour les trois clarificateurs et les trois vectoriseurs les résultats ,En effet la valeur de min_df et le score de précision sont inversement proportionnelles:plus la valeur de min_df augmente, plus le score de précision diminue. Cela est cohérent. En effet, les mots peu fréquents peuvent être très discriminants :

Se passer d'eux fait perdre de l'information, ce qui explique la chute des scores lorsque min_df augmente.

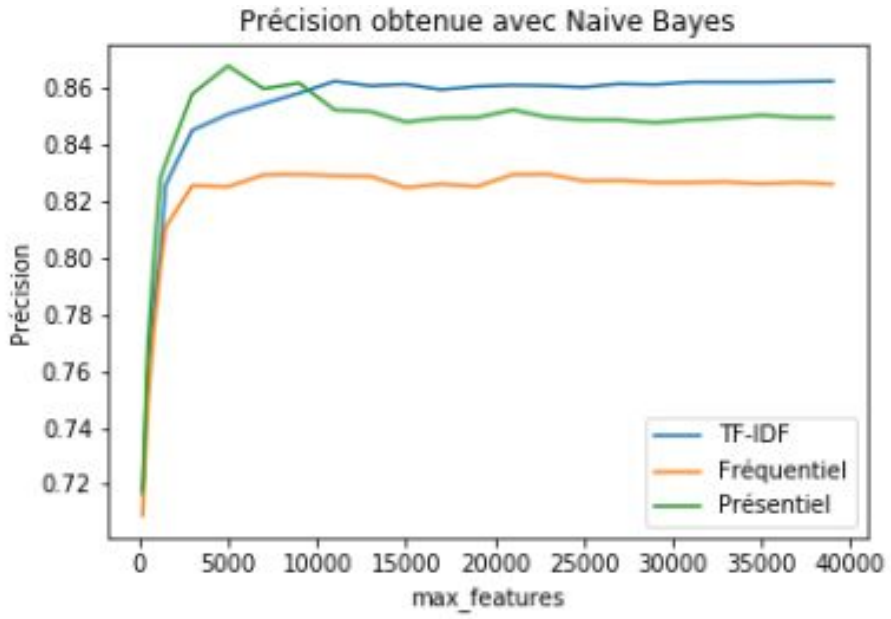
2) Le deuxième paramètre à tester est max_df qui permet de limiter la taille du dictionnaire en ignorant les mots les plus fréquents. Ainsi, max_df = 0.75 permettra d'ignorer les mots apparaissant dans plus de 75% du corpus. Seuls les mots apparaissant dans moins de 75% du corpus seront alors conservés pour la vectorisation.

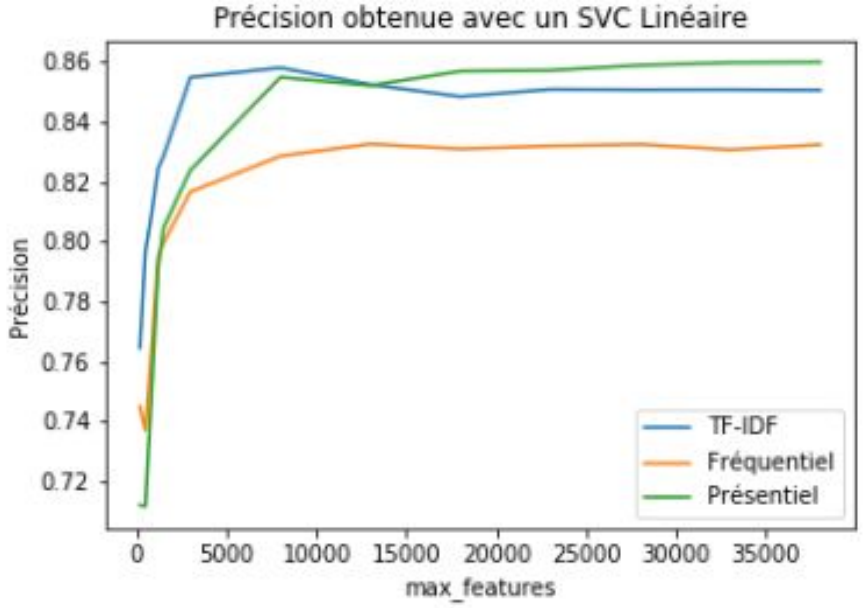




Les résultats obtenus sont intéressants. On peut voir sur les courbes que lorsque max_df varie dans l'intervalle [0.75, 1], les scores restent à peu près constants. Cela est cohérent avec le fait que les mots les plus fréquents sont peu discriminants et n'aident pas dans un problème de classification. On peut donc limiter la taille du dictionnaire en fixant max_df à une valeur d'environ 0.75,

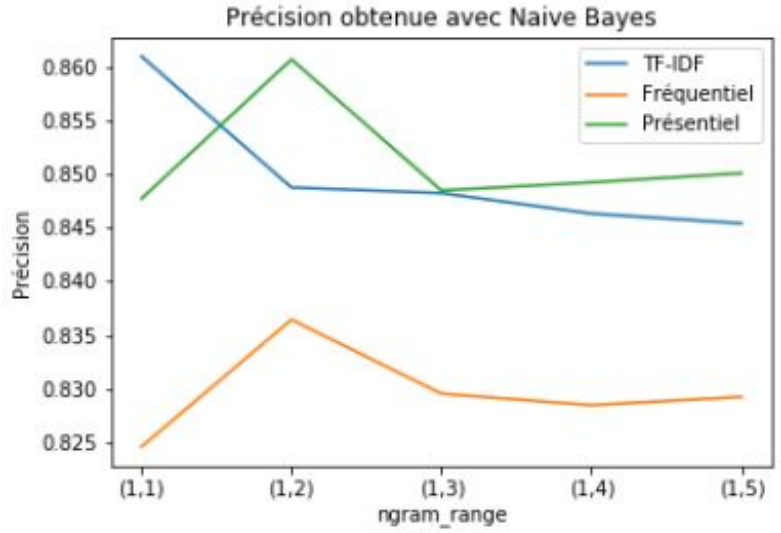
3) Un autre paramètre à tester est max_features qui définit la taille maximale du dictionnaire (i.e. le nombre de mots uniques maximal) utilisé pour la vectorisation des données.

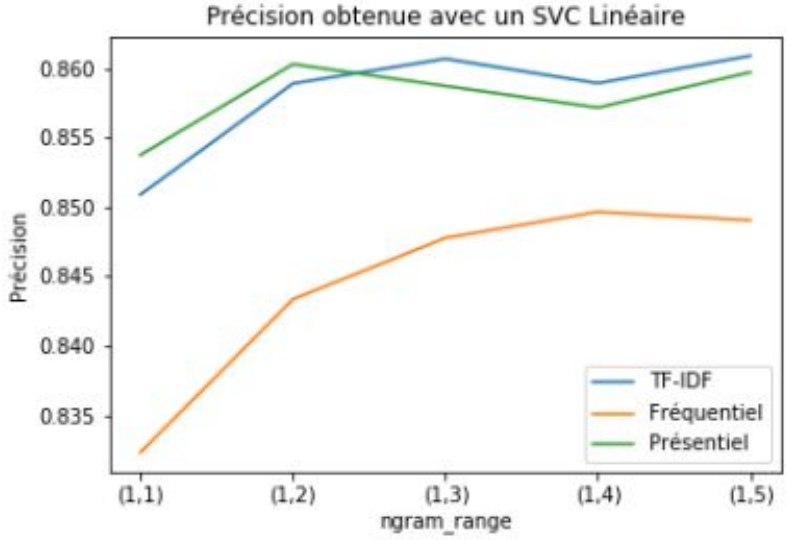




Les illustrations montrent que pour les deux classifieurs ainsi que les trois représentations, on observe globalement le même comportement : La courbe de précision croît rapidement jusqu'à un certain seuil puis elle reste constante à ce seuil. On peut en déduire qu'après une certaine taille, augmenter la taille du dictionnaire n'améliore pas les performances. On remarque qu'aux alentours de max_features = 2500, la précision est maximal,on remarque aussi que pour de faibles valeurs de max_features elle s'effondre, ce qui est cohérent : La perte d'information est trop importante pour obtenir une classification correcte

4) Le dernier paramètre à tester est n-gram range qui indique que les mot sont considérés seuls ou bien ils sont interprétés dans leurs contextes.





O remarque que considérer les mots comme des bi-grammes donne toujours de meilleurs scores que la représentation sous forme d'uni-grammes. Cela semble logique,en effet, Le contexte d'un mot joue un rôle important dans la classification de sentiments. Les

représentations qui semblent donner les meilleurs résultats sont les représentations sous forme de bi-grammes et tri-grammes(n-gram = (1,2) ou (1.3))

Conclusion:

Après un gridsearch sur chaque classifieur en faisant varier tous les paramètres testés plus haut en même temps, on trouve les meilleurs résultats avec une vectorisation TF-IDF
et un classifieur SVC linéaire, àpres un pre-processing des données (stemming,tokenization, lettres minuscules, stopwords), Les paramètres du vectoriseur TF-IDF sont les suivants :

- max_df=0.6
- min_df=0
- n-gram_range=(1.3)

Ça donne une précision moyenne de 82 %