**Document complémentaire**

Dans ce document, je vais expliquer davantage la démarche suivie, les analyses faites et les résultats obtenues.

1. **Démarche suivie :**

Le but du projet est de classifier des discours (Chirac/Mitterrand). Le point de départ est de charger les données. J’ai choisi d’utiliser la librairie Pandas vu qu’elle supporte la lecture des fichier « .pkl » et elle facilite le traitement des données sous format des DataFrames.

Ainsi, j’ai créé un Dataframe de deux colonnes « sentences » pour les textes et « labels » pour labeliser (C : Chirac, M : Mitterrand).

La deuxième étape est de préparer les données. J’ai appliqué ces traitements :

1. Enlever les ponctuation (. ?;! …) et les tabulations
2. Mettre les textes en minuscules
3. Supprimer les mots vides (stopwords de la librairie stop-words)
4. Calculer le nombre des mots par sentence pour faire des analyses
5. D’autres traitements à voir dans la section Modélisation

L’étape suivante est d’analyser les données (EDA). Je détaillerai ce point dans section II.

La dernière étape est d’implémenter des modèles de classification. J’ai testé TF-IDF et Word2Vec pour la représentation des mots. J’ai appliqué une régression logistique et SVM comme des modèles de ML. J’ai utilisé la métrique f-score. Dans la section modélisation, je détaillerai les résultats obtenus.

1. **EDA :**

Il est nécessaire d’analyser les données pour mieux les comprendre.

Pour éviter d’être répétitive, je vous invite à consulter la section EDA de mon notebook où j’ai implémenté, visualisé et interprété les analyses.

1. **Modélisation** :

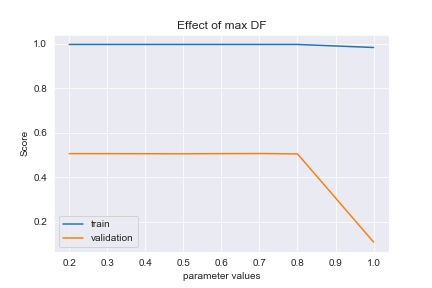
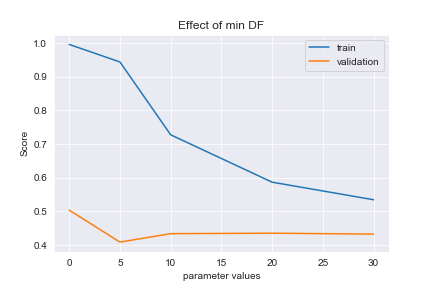
Dans cette partie, j’ai commencé par faire le split des données en 80% pour l’entrainement et 20% pour le teste. J’ai fixé le random\_state à 1 pour avoir les mêmes résultats pour plusieurs exécutions (pour comparer). J’ai pris en compte le déséquilibre entre les deux classes en faisant un split assurant d’avoir la même distribution pour les deux jeux de données.

Après, j’ai implémenté plusieurs pipeline de classification en aplliquant une validation croisée pour éviter les problèmes de data leakage.

J’ai appliqué TF-IDF et Word2Vec pour la conversion des données textuelles en des valeurs numériques.

* 1. TF-IDF : Cette représentation se base sur deux termes, nombre d’occurrence des mots (TF) et le nombre des document contenant le mot (DF). L’idée est de créer un matrice de vocabulaire x document. Comme la taille de cette matrice sera très grande, il faut éliminer les mots inutiles. Dans cet optique, j’ai configuré les paramètres de l’implémentation ScikitLearn pour supprimer les mots vides, appliquer un stemmer et créer des ngrames (de 1 à 3). Je précise que j’ai testé un FrenStemmer qui supporte la langue française. On peut tester d’autres.

J’ai fait une étude afin de fixer les paramètres max\_df et min\_df représentée par les figure ci-dessous :

On voit clairement, que pour le max\_df (maximum des fréquences des documents), il y a une déviation du score si on dépasse les 80%. C’est pour dire que si un mot est présent dans plus de 80% des documents, ce mot a un effet négatif sur la performance.

D’autre part, le min\_df a un effet substantiel pour toutes les valeurs. Ça s’explique par le présence de mots exclusive pour chaque classe (voir EDA).

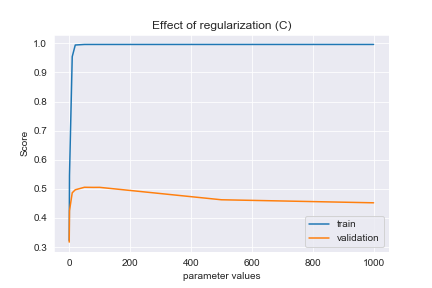
* 1. Word2Vec : Comme mentionné avant, j’ai appliqué aussi Word2Vec pour la représentation des mots. Ce choix est suite aux mauvaises performances de TF-IDF. Aussi, Word2Vec suit une autre logique. TF-IDF se base sur l’occurrences des mots tandis que Word2vec pénètre plus dans le syntaxe des textes.

Vu les contraintes de performance de calcul (pas de GPU) et de temps, j’ai décidé d’utiliser un modèle pré-entrainé. C’était difficile de trouver un modèle en langue française. J’ai utilisé les vecteurs de ce [lien](https://github.com/Kyubyong/wordvectors) . Il faut noter que ce modèle contient 3millions vecteurs de taille 52.

J’ai implémenté après une classe transformateur « word2vec\_transformer » pour pouvoir intégrer word2vec dans un pipeline. Ce transformateur calcule le vecteur d’un texte en faisant les moyennes des vecteurs des mots de ce texte.

Comme modèle de classification, j’ai appliqué la régression logistique et SVM. Le premier est un modèle probabiliste et itératif. Le deuxième a pour objectif de trouver la séparation optimale entre deux classes et aussi il supporte les données non linéairement séparable.

Une validation a été faite pour trouver les bons paramètres pour chaque modèle. J’ai fait une étude sur le coefficient de régularisation C de la régression logistique :

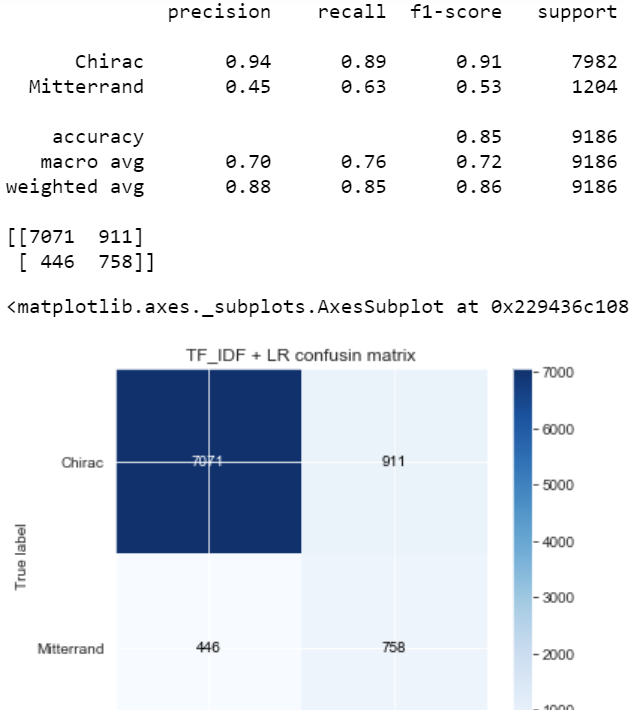
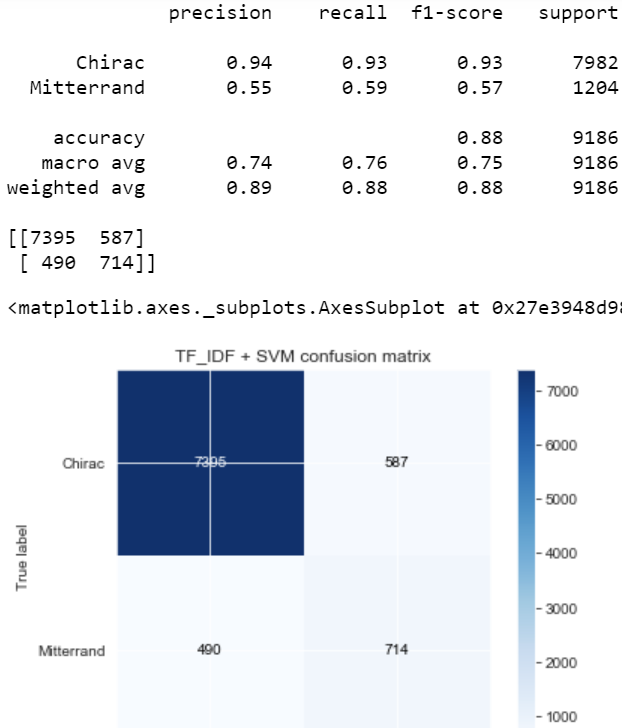


Le graphe montre que la valeur optimale pour C est de 50.

Pour tous les graphes que j’ai montré, il a une différence importante entre les scores de train et de test ce qui résulte un overfitting du modèle.

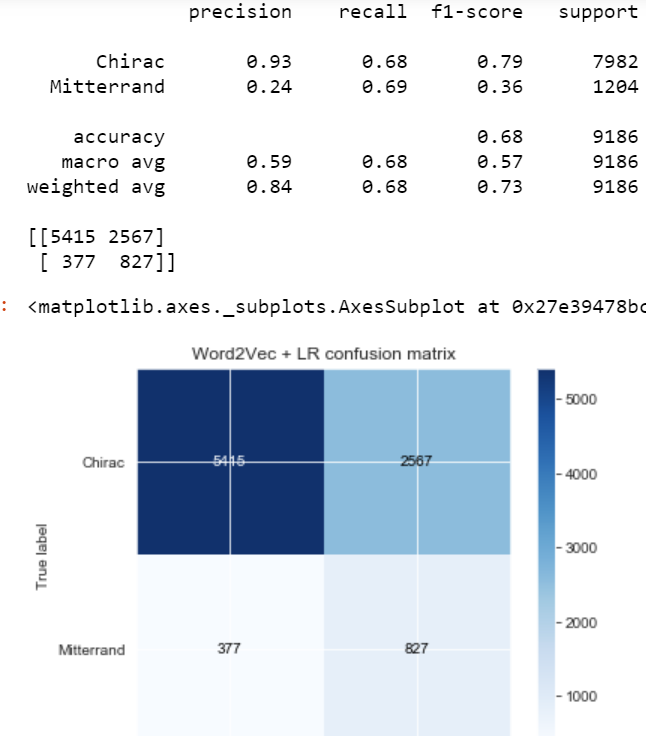
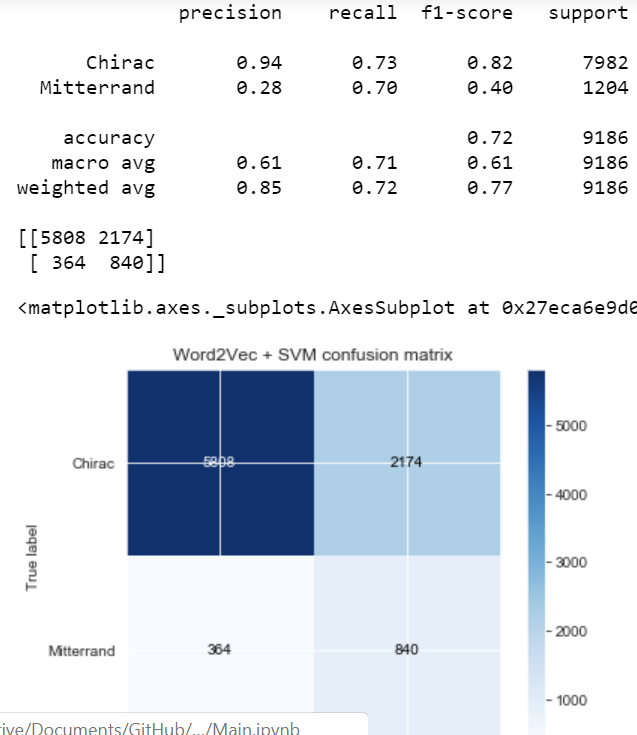
1. **Résultats :**

Ci-dessous les matrices de confusion pour les 4 modèles testés.

Chirac Mitterrand

Chirac Mitterrand

Chirac Mitterrand

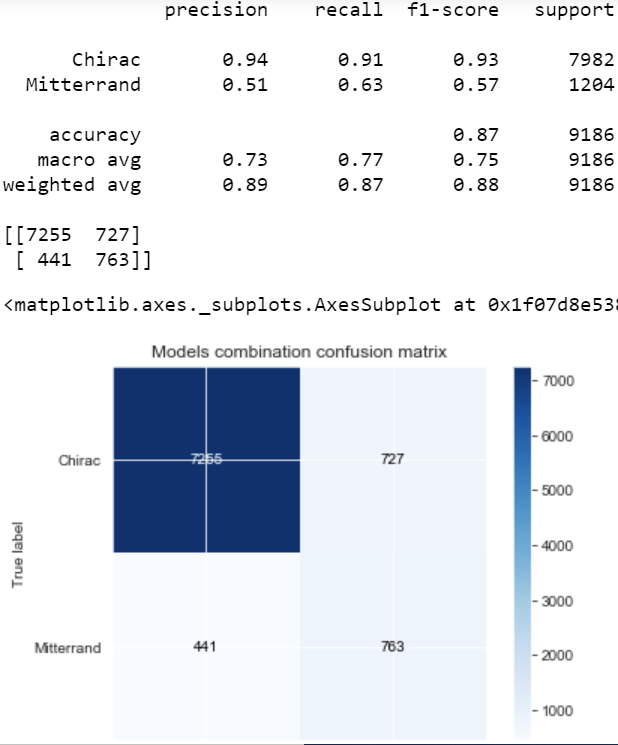
Chirac Mitterrand

Pour les 4 modèles, j’ai pas eu de bons résultats en se basant sur la métrique f\_score. En analysant de près les matrices de confusion, on remarque que le problème venait plus de la classe Mitterrand vu le déséquilibre entre les deux classes.

On voit aussi que SVM dépasse Régression logistique dans les deux cas. Je tiens à préciser que pour TF\_IDF, SVM a utilisé un noyau linéaire et pour Word2Vec un noyau gaussien. J’explique ça par le fait que TF\_IDF génère une matrice de taille très grande de façon que les textes sont très loin dans l’espace (d’où la possibilité de trouver une séparation linéaire).

En termes de représentation, TF\_IDF dépasse Word2Vec. Le choix du modèle Word2Vec pré\_entrainé est à l’origine de ce fait.

J’ai remarqué aussi qu’avec TF\_IDF, la classification pour « Chirac » est mieux qu’avec Word2Vec. C’est le sens inverse pour la classe « Mitterrand ». J’ai eu l’idée ainsi de faire la combinaison des 3 meilleurs modèles représentée par la figure ci-dessous :



La combinaison des modèles ne changent pas les performances pour la classe « Mitterrand ».

Le modèle TF\_IDF + SVM est le meilleur modèle dans ce travail.

En se basant sur le métrique « accuracy » , le modèle TF\_IDF + SVM est le meilleur parmi ceux testés.

1. **Futur travail :**

Comme futur travail, je propose de tester ces points :

* Equilibrer les deux classes en prenant 6000 textes pour chacune
* Appliquer d’autres stemmers
* Entrainer un modèle Word2Vec et tester plusieurs dimensions
* Faire les tests sur d’autres jeux de données (modifier le random\_state)
* Utiliser la métrique AUC-ROC