RITAL

Compte-Rendu
Traitement Automatique de la Langue

Master DAC

CHOI Esther - 3800370

PHO Vinh-Son - 3802052

Sommaire

1 Introduction

L'objectif de ce projet est la réalisation d'une chaîne de traitement pour la classification de textes. Nous allons nous concentrer sur deux types de problèmes :

- La prédiction de l'orateur d'une phrase parmi Chirac et Mitterand
- L'analyse de sentiment sur des avis de films

2 Prédiction sur des discours d'anciens présidents

2.1 Introduction

L'objectif est, comme dit précédemment, la réalisation totale d'une chaîne de traitement pour déterminer si un phrase a été dite par François Mitterrand (noté M ou -1) ou par Jacques Chirac (C ou 1).

Il s'agit ici d'un problème d'apprentissage supervisé : nous avons à notre disposition deux fichiers, l'un contenant l'ensemble d'apprentissage (57000 phrases étiquetées) et l'autre l'ensemble de test (27000 phrases non étiquetées).
Il sera intéressant de noter que les données des deux fichiers sont ordonnées en blocs.
De plus les deux échantillons sont biaisés, on a beaucoup plus de phrases prononcées par Chirac que par Mitterrand. De ce fait, un algorithme ne prédisant que la classe majoritaire aura un taux de bonne prédiction de 87%.
Pour pallier ce problème, on évaluera l'efficacité des différents modèles avec un score F_1 sur la classe minoritaire :

$$F_1 = \frac{tp}{tp + \frac{1}{2}(fp + tn)}$$

Avec :

- tp est le nombre de phrases prononcées par Mitterrand bien étiquetées (vrais positifs)
- fp le nombre de phrases prononcées par Chirac mal étiquetées (faux positifs)
- fn le nombre de phrases prononcées par Mitterrand mal étiquetées (faux négatifs).

Les scores F_1 que l'on présente sont réalisés en validation croisée 5-fold.

2.2 Pre-processing

Le pre-processing correspond à la transformation des données brutes en un format utilisable par les différents algorithmes que dont nous allons nous servir.

Pour cette tâche, nous utilisons une représentation en sac de mots, c'est à dire que l'on représente les données en une grande matrice où chaque ligne correspond à un document (ici une phrase) et chaque colonne correspond à un mot.
On peut faire différents choix quant aux coefficients de cette matrice.

2.2.1 Codage

Nous avons testé 3 différentes manières d'encoder les données: présentiel, fréquentiel et TF-IDF.

Le codage présentiel (1 si le mot apparaît dans le document, 0 sinon) ne semble pas intéressant. En effet, un mot peut donner une information assez discriminante selon qu'il soit beaucoup prononcé ou non.
La notion de fréquence d'usage des mots paraît donc assez importante.

Le codage fréquentiel que nous avons utilisé nous semble le plus adapté au problème dans la mesure où il se concentre sur l'information que nous pensons être la plus pertinente : la fréquence d'apparition des mots.

Le codage TF-IDF est semblable au codage fréquentiel (TF signifiant *Term Frequency*), mais on pondère chaque coefficient par l'IDF (*Inverse Document Frequency*). Cela est intéressant pour la recherche d'information pour trouver des documents pertinents par rapport à une requête mais dans notre cas cela ne fait que rajouter du "bruit" dans nos données.

2.2.2 Mots vides

Dans le langage naturel, l'apparition des mots ne suit pas une distribution uniforme. Certains apparaissent beaucoup plus souvent que d'autres. Ces mots, qui apparaissent très fréquemment dans le langage naturel, ne sont pas discriminant pour identifier l'orateur. Pire, ils rajoutent du bruit dans nos données.
Pour s'en débarrasser, il y a 2 méthodes : utiliser une liste de mots vide et borner la fréquence d'apparition d'un mots dans tous les documents. La première méthode nous semblant plutôt arbitraire, nous avons décidé d'utiliser la seconde.
En test, nous avons vu que seuiller la DF (ignorer les mots aillant une plus grande DF) entre 0.1 et 0.2 était le plus efficace.

2.2.3 Taille de la matrice

Une fois qu'on a vu qu'il fallait ajouter une borne supérieure à la DF, on peut maintenant se poser la question de l'ajout une borne inférieure. En effet, certains termes n'apparaissant pas beaucoup ne sont pas très discriminant, on peut même les considérer comme du bruit.
Un autre problème nous appelant à utiliser une borne inf est la taille de la matrice. Une matrice trop grande cause de longs calculs. Dans cette optique, à la place d'utiliser une

borne inf pour la DF, nous avons préféré utiliser un paramètre choisissant le nombre de mots à conserver.
En test, nous avons vu qu'il ne fallait conserver qu'entre 10 000 et 15 000 mots pour optimiser le score.

2.2.4 Conversion en minuscule

Les majuscules sont utilisées en français pour les noms propres et au début des phrases. C'est la deuxième utilisation qui nous intéresse ici. Comme deux personnes n'ont pas la même façon de commencer une phrase, nous avons choisi de ne pas convertir tout le texte en minuscule.

2.2.5 Utilisation de n-grammes

Les n-grammes peuvent être intéressants notamment pour garder l'information d'expression comme "aujourd'hui". Néanmoins nous n'avons pas vu une grande amélioration des performances suite à l'utilisation de n-grammes, de plus leur utilisation augmentaient grandement le temps de calcul du pre-proccessing. De ce fait, nous avons choisi de ne pas les utiliser.

2.3 Processing/Apprentissage

Nous avons testé différents algorithmes d'apprentissage tels que NaiveBayes, SVM ou les algorithmes de régression. Les deux derniers avaient les meilleurs scores F_1. Pour choisir entre ces 2 modèles, nous nous sommes appuyés sur un autre problème venant des données : le biais de l'échantillon.
Pour résoudre ce problème, nous avons choisi d'utiliser de déplacer par la suite la frontière de décision et seule la régression logistique nous permettait cela.

Soit d un document. En temps normal, d est sur la frontière de décision si $P(1|d) = P(-1|d) = 0.5$
On définit la nouvelle frontière de la manière suivante :
Soit $b \in]0, 0.5[$. d se situe sur la frontière si $P(1|d) = 1 - b \Leftrightarrow P(-1|d) = b$
En d'autres termes, on étiquette -1 un texte si $P(-1|d) \geq b$.

Il est intéressant de noter que la valeur optimale de ce b dépend grandement de la qualité du post-processing. En effet pour améliorer le score F_1, on introduit par la même occasion beaucoup de bruit.

2.4 Post-processing/Lissage

Le lissage consiste à prendre en compte la structure des données sur lesquelles on travaille, pour "harmoniser" nos prédictions. Ici, les données sont présentées en bloc.
Nous avons donc testé différentes méthodes pour lisser les prédictions.

2.4.1 Transformée de Fourier et filtre passe-bas

On peut voir la prédiction avant lissage comme un signal bruité. Une idée assez simple consiste ainsi à passer un filtre passe-bas idéal sur notre prédiction, c'est-à-dire réaliser l'opération suivante $smoothed\_pred \leftarrow pred * sinc$.
Au lieu de réaliser une convolution avec un sinus cardinal, on préférera travailler dans le domaine fréquentiel:
On note $pred$ notre prédiction initiale, FT la transformée de Fourier.
Comme $FT(sinc(t)) = rect(f)$,
$smoothed\_pred \leftarrow pred * sinc \quad \Leftrightarrow \quad smoothed\_pred \leftarrow FT^{-1}[rect(f) \times FT(pred)]$

Il nous reste maintenant à définir la fréquence de coupure.

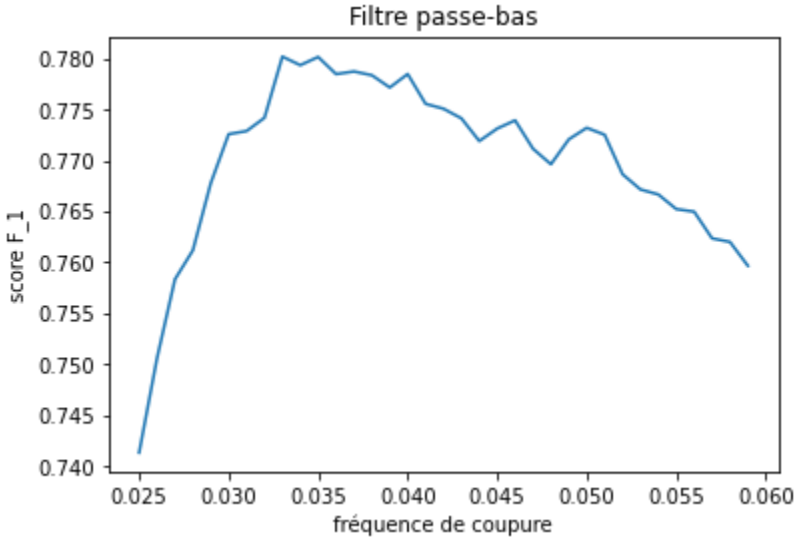


Figure 1: Score F_1 en fonction de la fréquence de coupure

On voit, expérimentalement, qu'une bonne fréquence de coupure est $f_c = 0.035$. Cette approche donne de bons résultats en prenant relativement peu de temps.

2.4.2 Filtre Gaussien

Un autre filtre que nous avons testé est le filtre gaussien. Celui-ci équivaut à faire une moyenne pondérée avec des coefficients calculés à partir d'une gaussienne que l'on a discrétisé. Un noyau gaussien a donc cette forme:

$ker =$ (0.03 0.22 0.68 1 0.68 0.22 0.03)

L'avantage de cette approche est que l'on prend en compte la notion de voisinage : pour assigner une valeur, on prend d'avantage en compte un point proche qu'un point éloigné.

Malheureusement, ce lissage ne permet pas d'obtenir de très bons scores. On peut donc supposer que ne passer qu'un seul filtre laisse encore beaucoup de bruit.

2.4.3 Filtre de Kolmogorov-Zurbenko

Le filtre de Kolmogorov-Zurbenko (ou KZF) est assez simple, il n'a que 2 paramètres, k et m. Il consiste en k itérations d'une moyenne glissante sur une fenêtre de taille m.

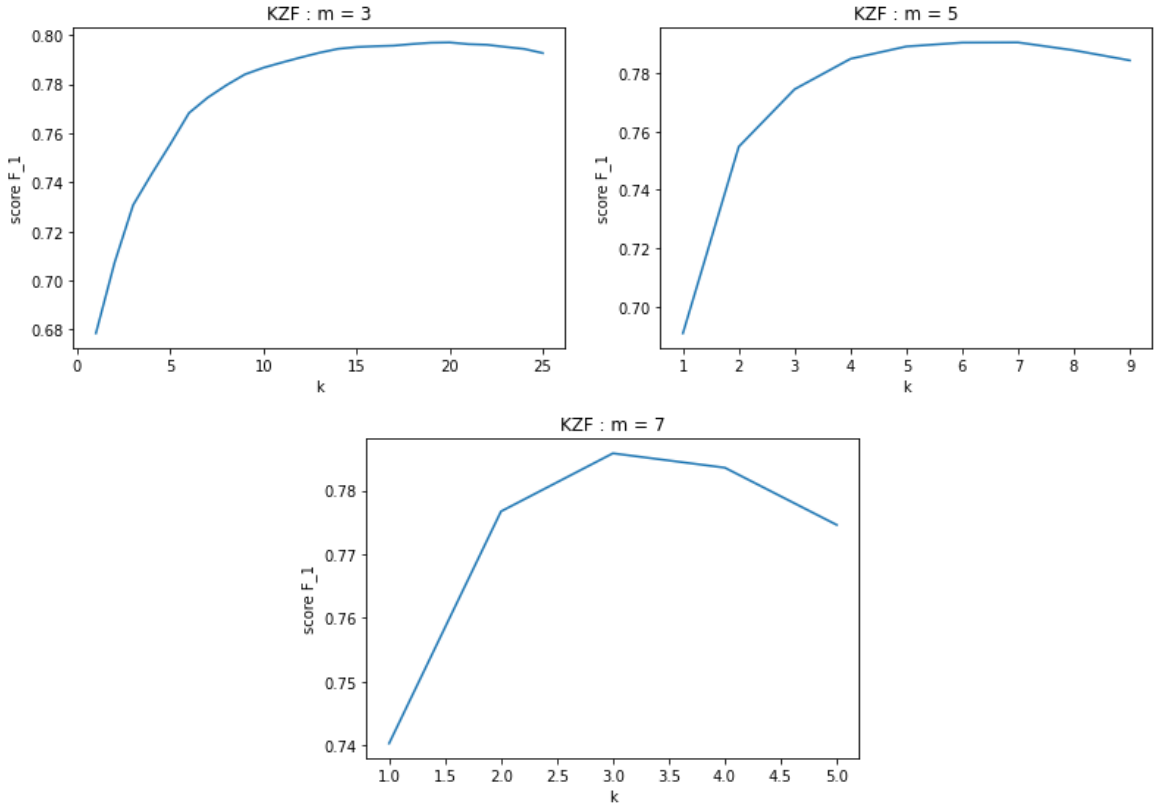


Figure 2: Score F_1 en fonction de k pour 3 valeurs de m : $m = 3$ (gauche), $m = 3$ (droite) et $m = 7$ (bas)

En testant différentes combinaisons de paramètres, on voit que les meilleurs paramètres dans notre cas sont $m = 3$ et $k = 20$. On a alors de très bons scores puisqu'on approche les 0.8 de F_1.

On peut encore faire mieux. Un désavantage de ce filtre est que l'on ne peut pas considérer en même temps l'échelle microscopique (les voisins très proches) et l'échelle macroscopique (le bloc de phrases).

2.4.4 KZF modifié

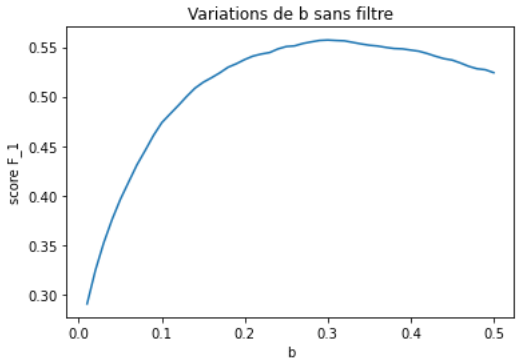
Pour avoir des meilleurs résultats, on modifie le KZF. Au lieu de réaliser k itérations d'une moyenne glissante sur une fenêtre de taille m, on réalise k itérations d'une moyenne glissante

mais sur une fenêtre de taille $2 \times i + 1$ où i est le numéro de l'itération courante.

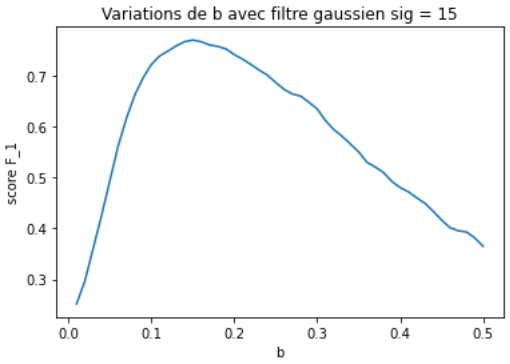
Cette modification permet, comme pour le filtre gaussien, de prendre en compte l'échelle microscopique et l'échelle macroscopique. Et ceci, tout en gardant les avantages du KZF, c'est à dire le passage de plusieurs filtres réduisant le bruit. Grâce à ce filtre, on dépasse confortablement les 0,8 de F_1.

2.4.5 Paramètre b

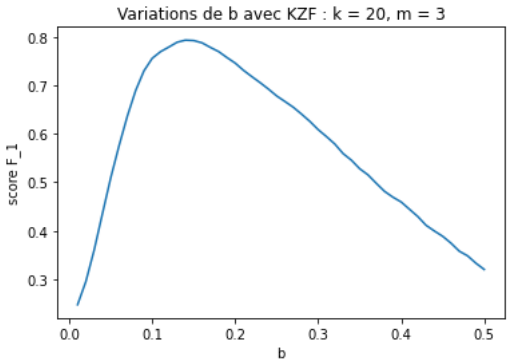
Comme dit précédemment, le paramètre b dépend beaucoup du lissage. Plus le lissage est bon, plus on peut faire baisser la valeur de b.



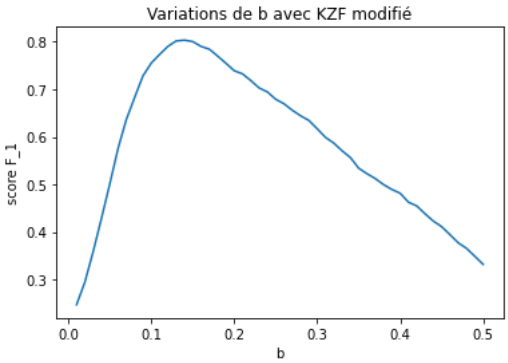
(a) Score F_1 sans filtre en fonction de b



(b) Score F_1 avec filtre gaussien en fonction de b



(c) Score F_1 avec KZF en fonction de b



(d) Score F_1 avec KZF modifié en fonction de b

Figure 3: Variation du paramètre b pour différents lissages

3 Analyse de sentiment

3.1 Première évaluation

Nous avons commencé par évaluer les performances du modèle obtenu en utilisant tous les meilleurs paramètres pour la classification de présidents, sur les avis de films. Pour que

l'expérience ait un sens, les données d'entrées sont encodées d'une manière similaire que précédemment, c'est-à-dire par blocs (ici de 10) alternant les avis positifs et négatifs.
Les donnes étant équilibrées (il y a autant d'avis positifs que d'avis négatifs), nous pouvons évaluer les performances sur le taux de bonne classification. Bien entendu les scores ne sont pas très bons : on obtient en moyenne uniquement 55% de taux de bonne classification.
Ceci nous confirme que les paramètres optimaux dépendent grandement du problème et des données.

3.2 Amélioration très simple

Nous avons ensuite cherché un moyen simple d'améliorer ces scores. Nous avons ainsi discuté des différents paramètres. Premièrement, n'ayant aucun a priori sur les données de test, nous n'allons pas ordonner les données en bloc mais simplement dans le désordre.

- `tf-idf` : Nous allons utiliser un codage TF-IDF, cela nous semble pertinent car il pénalise les mots présents dans beaucoup de documents : ce sont des mots qui ne sont pas très discriminants.
- `max_df` : Ce paramètre est très utile pour enlever les mots vides. En testant plusieurs valeurs, on retrouve une fréquence documentaire max aux environs de 20%. Nous avons également pu constater qu'avec une bonne valeur `max_df`, l'utilisation du paramètre `stop_words` est non nécessaire pour avoir de bons résultats.
- `max_feature` : Encore une fois, en test, on retrouve une valeur proche de la tâche précédente.
- `lowercase` : Les documents étant déjà en minuscule, ce paramètre ne nous intéressa pas.
- `Ponctuation` : On conserve la ponctuation, elle donne beaucoup d'information sur le sens d'un document.
- `lissage` : Comme les données ne sont pas structurées, nous ne pouvons pas réaliser de lissage.
- `paramètre b` : Ce paramètre ayant été introduit dans la première partie pour faire face à un biais d'échantillonage, nous n'en avons plus besoin pour des données équilibrées.

Ainsi, avec les paramètres `max_df` $= 0.32$ et `max_feature` $= 10000$, nous obtenons en moyenne 83% de taux de bonne classification, ce qui est bien meilleur que le 55% d'avant.
Nous pouvons donc voir que rien qu'avec une simple discussion pas très poussée des paramètres, on arrive à avoir des bons résultats.

4 Conclusion

À travers ce projet, nous avons pu nous initier à la réalisation d'une campagne d'expérience. Malgré la difficulté des problèmes de NLP, nous avons abouti à de très bons résultats en réfléchissant sur les trois volets de la classification : le pre-processing, l'apprentissage et le post-processing.
Pour encore améliorer nos résultats, nous aurions pu, pour la première tâche, tester d'avantage de méthodes pour résoudre le problème du biais d'échantillonage. Et pour la seconde tâche, une étude plus approfondie de différents classifieurs aurait pu être intéressante, notamment sur l'efficacité de l'utilisation de noyaux avec les SVMs.