Classification de discours politiques

Julie Halbout
Nanterre Université

Delphine Nguyen-Durandet
Sorbonne-Nouvelle

Aurelien Said Housseini
Sorbonne-Nouvelle

Abstract

Cet article a pour objectif de présenter notre travail sur la tâche 3 de classification de texte de DEFT'09. Nous y présentons notre méthodologie ainsi que le meilleur classifieur obtenu. Le meilleur classifieur pour nos données est un modèle comprenant un classifieur de régression logistique avec un vectorizer comptant l'occurrence des tokens. L'ajout d'autres features n'a pas été pertinent pour notre tâche. Nous discutons également des perspectives pour améliorer la classification de nos données.

1 Introduction

Pour cette tâche de classification, nous avons eu pour objectif d'attribuer aux discours politiques le parti politique auquel l'orateur appartient. Cinq partis européens sont représentés dans nos données, qui proviennent de débats parlementaires européen [1]. Nous avons choisi de nous concentrer sur les discours en français seulement au vu de la complexité de la tâche. L'annotation manuelle de cette tâche a été évaluée par un Kappa de Cohen qui a varié entre $-0,14$ et $0,24$ en fonction des couples d'annotateurs. Cet accord a été qualifié de mauvais à médiocre (Hoareau and El Ghali, 2009, 47). Nous avons constaté que les discours politiques appartenant clairement à la gauche ou à la droite étaient plus facilement annotés que les autres. Cela était attendu, mais nous souhaitons maintenant voir comment dépasser cette simple dichotomie gauche/droite et améliorer notre taux d'exactitude.

Pour cette tâche, nous avons utilisé le module python sklearn (Pedregosa et al., 2011). En ce qui concerne le choix des classifieurs, il a été effectué en suivant les recommandations de sklearn[2], qui suggère d'utiliser l'un de ces deux classifieurs :

- Un classifieur linéaire Support Vector Machine : algorithme de classification supervisée qui utilise l'analyse de données vectorielles pour trouver un hyperplan de séparation optimale entre différentes classes.
- Un classifieur Naive Bayes : algorithme de classification supervisée basé sur l'application du théorème de Bayes et l'hypothèse de l'indépendance des caractéristiques.

En explorant la littérature (Aly; Kowsari et al.), nous avons également identifié d'autres classifieurs qui pourraient être adaptés à notre tâche, tels que :

- Random Forest : algorithme de classification et de régression utilisant un ensemble d'arbres de décision. Il fonctionne en entraînant plusieurs arbres de décision sur des sous-ensembles aléatoires des données d'entraînement, puis en faisant une prédiction en utilisant la majorité des prédictions des arbres.
- La régression logistique : algorithme de classification utilisé pour prédire la probabilité qu'un événement appartienne à une certaine classe. Elle est souvent utilisée dans les cas où il y a deux classes (appelées classe positive et classe négative). On a également la possibilité d'utiliser des options de multi_class pour répondre à notre besoin de classification ici.
- KNN (K Nearest Neighbors) : algorithme de classification supervisée utilisé pour prédire la classe d'une donnée en regardant les classes des données les plus proches dans l'espace de données. Nous avons donc décidé d'intégrer ces classifieurs à nos expériences.

Pour réaliser cette tâche, nous nous sommes aidés des différentes techniques de vectorisations à notre disposition :

[1] Le site Internet du Parlement européen : http://www.europarl.europa.eu/ propose un libre accès à ses archives.

[2] Lien vers les recommandations de sklearn : https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

- GloVe (Pennington et al.) (Global Vectors for Word Representation) est un modèle de traitement du langage naturel qui utilise un algorithme de co-occurrence global pour apprendre des représentations vectorielles de mots à partir d'un grand corpus de textes.

- Word2Vec (Mikolov et al.) est un modèle de traitement du langage naturel qui a été initialement développé par Google en 2013. Il a été conçu pour apprendre les relations sémantiques entre mots en analysant la fréquence de co-occurrence de ces mots dans un grand corpus de textes. Le modèle utilise un réseau de neurones pour apprendre à prédire un mot à partir de ses voisins contextuels dans un texte.

- FastText (Bojanowski et al., 2016) est un modèle de traitement du langage naturel développé par Bojanowski, Grave, Joulin et Mikolov (2016). FastText utilise non seulement les mots dans leur forme entière, mais aussi les sous-sequences de caractères qui composent ces mots, ce qui permet au modèle de mieux gérer les mots rares et les mots inconnus.

Dans cet article, nous allons décrire avec le plus de précision possible la méthodologie que nous avons suivie (Section 2). Nous présenterons ensuite les résultats (Section 3), avant de les analyser (Section 4). Enfin, nous évoquerons quelques pistes dans la partie discussion (Section 5).

L'ensemble des ressources crées à l'issue de notre travail sont disponibles sur notre GitHub[3].

2 Méthodologie

En vu de trouver le meilleur classifieur, nous avons déterminer plusieurs étapes de travail. Il a fallu dans un premier temps nettoyer les données avant d'effectuer des traitements dessus. Ensuite, nous avons choisi différents classifieurs à tester, pour lesquelles nous avons utilisés des grilles de recherche afin de déterminer les meilleurs hyperparamètres. Enfin, nous avons comparé ces classifieurs pour chaque technique de vectorisation utilisée pour ne garder que le meilleur. Sur ce meilleur classifieur, nous avons testé l'ajout d'autres features afin de voir si le modèle pouvait être amélioré. Tous nos traitements se sont fait avec des pipelines comme dans la Figure 1.

[3] https://github.com/Julie921/Apprentissage_Artificiel

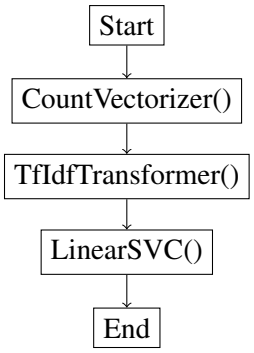

Figure 1: Schéma d'une pipeline contenant deux prétraitement et un classifieur LinearSVC()

2.1 Nettoyage des données

Avant d'effectuer un quelconque traitement sur nos données, nous les avons nettoyées. La ponctuation a été supprimée (y compris les apostrophes) et les caractères ont été transposés en minuscules. Ce nettoyage a été fait sur les données d'entraînement et sur les données de test.

2.2 Traitements des données

Pour les données textuelles, plusieurs pistes ont été envisagées pour la vectorisation des données :

- Compte des occurrences des tokens (séparation sur les espaces) et ajout d'un Tf-Idf.

- Vectorisation de chaque token : Le modèle utilisé pour obtenir des vectoriseur GloVe et word2vec est un corpus Europarl (identique à notre jeu de données). Ce corpus a subi le même prétraitement que nos données pour avoir des tokens identiques. Le modèle utilisé pour obtenir des vecteurs FastText est un corpus de web (modèle disponible sur fasttext directement).

  - vecteurs GloVe : on remplace chaque token par des vecteurs obtenu avec GloVe.
  - vecteurs word2vec : on remplace chaque token par des vecteurs obtenu avec word2vec.
  - vecteurs FastText : on remplace chaque token par des vecteurs obtenu avec FastText.

  Dans chaque cas, on obtient une matrice où chaque mot est remplacé par son vecteur (les tailles sont différentes en fonction des techniques de vectorisation).

Pour les classes, nous avons choisi de changer l'encodage. Initialement, ce sont des données textuelles. Or, les classifieurs fonctionnement mieux avec des labels dans un format numérique.

Nous avons utilisé la fonction LabelEncoder() de sklearn pour obtenir une matrice de label. Chaque label textuel de parti politique a été converti en un nombre allant de 1 à 5..

2.3 Grilles de recherche

Nous sommes ensuite passé à la construction de nos pipelines pour chaque classifieur. Ces pipelines nous ont permis d'utiliser facilement des grilles de recherche avec sklearn (GridSearchCV()) pour trouver les meilleurs hyperparamètres. En effet, ces grilles de recherches permettent de faire varier les hyperparamètres et de créer toutes les combinaisons possibles pour faire sortir le meilleur modèle. GridSearch enregistre également le meilleur modèle.

Ces pipelines contiennent donc à la fois le traitement de nos données, et notre classifieur. Parfois, il a fallu ajouter un traitement supplémentaire entre le traitement des données et le classifieur pour éviter les problèmes de convergence. Nous avons donc parfois utiliser StandardScaler() de sklearn dans nos pipelines, qui permet de normaliser nos données.

De plus, la grille de recherche fait une cross validation, ce qui nous permet d'avoir un meilleur aperçu des résultats de notre modèle.

2.4 Sélection du meilleur classifieur

Une fois les classifieurs testés, nous avons gardé les meilleurs pour chaque type de vectorizer. Nous avons prédit les résultats sur nos données test et utiliser la fonction classification_report() pour voir nos résultats. Nous avions alors l'accuracy pour chaque modèle. C'est sur la base de ces résultats que nous avons selectionné le meilleur de nos classifieurs.

2.5 Ajout de features

Une fois que nous avons sélectionné le meilleur modèle, nous pouvons explorer l'ajout de features et voir comment se comporte notre classifieur. Nous avons tester :

- Ajout du calcul de la compléxité du vocabulaire : l'idée est de compter l'utilisation des mots sur l'ensemble des mots utilisés dans le discours pour déterminer la complexité du vocabulaire utilisé. Plus les mots utilisés sont divers, plus la complexité est grande.
- Ajout de classification par thème : utilisation d'un modèle de classification pré-entraîné du type "zero shot". L'intérêt des modèles comme "camembert-base-xnli" peut classer des énoncés dans les catégories de notre choix même en ayant jamais été entraîné sur ces catégories en particulier.
- Ajout de l'analyse en sentiment : calcul de la polarité des documents.

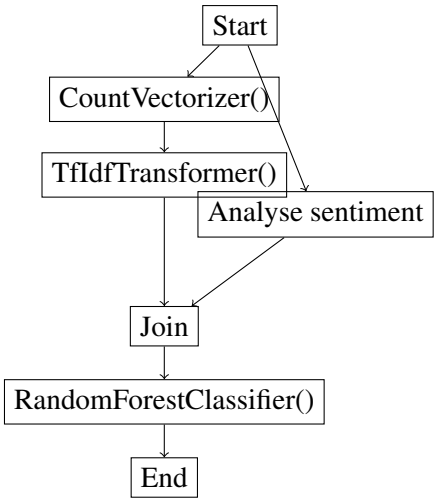

Figure 2: Schéma d'une pipeline avec deux traitements parallèles, et un classifieur Random Forest.

Les pipelines contiennent maintenant des traitements parallèles comme représentés dans la Figure 2.

3 Résultats

Au vu des résultats obtenus dans la Table 1, nous avons selectionné le classifieur en Régression Logistique avec la fonction CountVectorizer() pour la suite de nos traitements. Les meilleurs hyperparamètres trouvés par la grille de recherche sont les suivants :

- CountVectorizer() :
  - $min\_df = 1$: on ne prend pas les tokens qui apparaissent uniquement dans un document.
  - $ngram\_range = (1, 1)$: on prend les unigrams.
- LogiticRegression() :

Vectorisation	Modèle	TD-IDF	Score
Word2vec	RandomForest	False	0.77
FastText	Random Forest	False	0.76
Glove	Random Forest	False	0.76
CountVectorizer	Régression logistique	False	0.78

Table 1: Accuracy des meilleurs classifieurs en fonction du choix de vectorisation.

- $max\_iter = 10000$: on fait 10000 itérations maximum.
- $multi\_class$ = "ovr" : on entraînement autant de modèle différent qu'il y a de classe en les traitant chacun comme des modèles binaires.

On a donc ajouté des features (cf Figure 2) sur ce modèle. Nous avons rencontré des problèmes de temps de traitements. En effet, il a été impossible de lancer dans des temps raisonnables l'analyse en sentiment et la classification par thème sur nos l'ensemble de nos données d'entraînements. Néanmoins, sur un petit échantillon, les résultats étaient prometteurs ($+6\%$ sur un échantillon de 100). D'un autre côté, l'ajout du calcul de la complexité du vocabulaire a diminué nos résultats (accuracy : 0.71).

4 Analyse des résultats

Nous avons sorti la matrice de confusion (cf Figure 4) pour voir à quoi ressemblait les prédictions de notre modèle. On peut voir que les classes PPDE et GUE-NGL sont les mieux reconnues (réussite de 81% et 83%). La classe la moins bien reconnue est ELDR (68%). Les classes PPDE-DE et PSE sont les plus confondues, mais se sont également les deux classes les plus représentées dans nos données. Les discours de ces partis sont trois fois plus présents que ceux des autres partis.

La graphique sur la courbe d'apprentissage montre l'évolution de l'accuracy en fonction de l'avancement du modèle sur les échantillons. La cross-validation tourne autour de 40%. C'est un résultat particulièrement étonnant qui nous a suivi tout du long de nos entraînements. On remarque par exemple que lorsqu'on prend un score directement sur le corpus de train, il est très bon, ce qui est attendu, les données ont été vues par le classifieur. De même, les résultats sur le corpus de test sont très bons, mais durant les cross validation, il n'a pas été possible d'atteindre un score de 0.5.

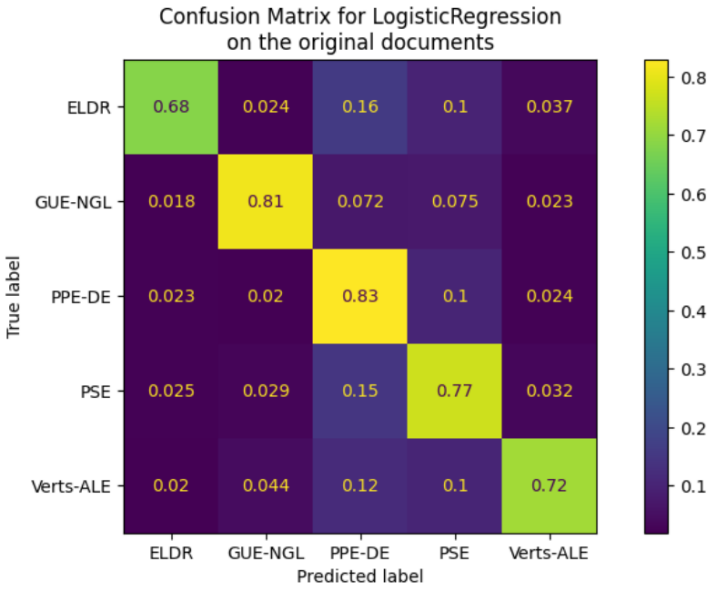

Figure 3: Matrice de confusion normalisée

Figure 4: Courbe d'apprentissage

5 Discussion

Nous avons été surpris de voir que l'utilisation de vectoriseurs GloVe, Word2Vec ou FastText ne surpassait pas les résultats obtenus avec la simple combinaison CountVectorizer() + TfIdfTransformer() proposée par sklearn.

De plus, un autre nettoyage de nos données auraient pu nous modifier nos résultats, mais les temps de traitements étaient trop lourd pour faire ce que nous souhaitions.

L'utilisation de poids pour pondérées les classes au vu de la dispersion hétérogène de nos classes aurait pu nous permettre d'obtenir de meilleur résultat. Néanmoins, nous n'avons pas trouvé de manière efficace de les obtenir pour que cela ait un effet positif sur nos résultats.

Enfin, au vu de nos données, il aurait pu être interessant de voir ce qu'un réseau de neuronnes récurrent (RNN) aurait pu donner comme résultat.

References

Mohamed Aly. Survey on multiclass classification methods.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Yann Vigile Hoareau and Adil El Ghali. 2009. Approche multi-traces et catégorisation de textes avec random indexing. *Actes du cinquième DÉfi Fouille de Textes*, page 55.

Kowsari, Jafari Meimandi, Heidarysafa, Mendu, Barnes, and Brown. Text classification algorithms: A survey. 10(4):150.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543. Association for Computational Linguistics.