

Détection automatique du parti politique d'un locuteur du Parlement Européen

Leila Hassani¹ and Pan Liu²

¹leilahassani@hotmail.fr

²liupan.lucine@gmail.com

January 4, 2023

Contents

1	Introduction	1
2	Formatage des données	1
2.1	Constitution des fichiers	1
2.2	Parsing du XML	2
3	Preprocessing	3
3.1	Distribution des données	3
3.2	TF-IDF	3
4	Modèle et résultats	4
4.1	Présentation des métriques : matrice de confusion et rapport de classification	4
4.2	Observation des prédictions	7
5	Conclusion	7

1 Introduction

Le but de la tâche 3 du DEfi Fouille de Texte 2009 est de créer un modèle d'apprentissage supervisé afin de détecter automatiquement le parti politique à partir d'un discours prononcé au parlement européen. Pour cela, nous avons accès à des données au format XML et au format texte fournies par les organisateurs du défi. La tâche inclut le formatage des données, les étapes de preprocessing des données si nécessaire ainsi que l'apprentissage et l'évaluation du modèle. Le but de ce rapport est de présenter le travail que nous avons fourni, les méthodes utilisées ainsi que les résultats obtenus sur cette tâche.

2 Formatage des données

2.1 Constitution des fichiers

Les données du DEfi Fouille de Texte 2009 sont des interventions de partis politiques lors de débats au parlement européen. Elles sont disponibles en anglais, en français et en italien, mais nous avons fait le choix de nous concentrer exclusivement sur les données en français. Ces données sont divisées en deux fichiers au format XML et un fichier au format texte :

Abstract

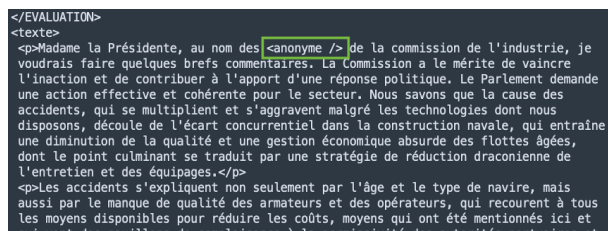
Ce rapport décrit le travail que nous avons effectué pour la tâche 3 du DEfi Fouille de Texte 2009 ainsi que les résultats obtenus sur la tâche de détection automatique du parti politique d'un locuteur du Parlement Européen.

- Un fichier *deft09-parlement_appr_fr.xml* contenant les données d'entraînement, identifiés par des attributs *id*, et les labels associés, contenus dans l'attribut *valeur* de la balise <PARTI>.
- Un fichier *deft09-parlement_test_fr.xml* contenant les données de test, cette fois-ci non associées à leurs labels directement dans le fichier XML.
- Les labels des données de test sont contenus dans un fichier texte séparé nommé *deft09-parlement_ref_fr.txt*. L'ordre des valeurs entre les données du fichier XML et les labels du fichier texte sont les mêmes (c'est-à-dire que le label du document 1 est en position 1, le label du document 2 en position 2, etc.). Les labels du fichier texte sont associés à un identifiant qui correspond à celui du document du fichier XML.

2.2 Parsing du XML

La partie la plus compliquée dans le formatage des données est l'association entre les données du fichier de test et le label correspondant du fichier texte. Le XML étant long à *parser*, nous avons fait le choix de reformater les données en données TSV avant de les exploiter. Ceci permet ensuite de les récupérer facilement avec le module *pandas* et donc de travailler avec un *dataframe*, ce qui facilite les tâches de *pre-processing* et de division du corpus en différents jeux de données.

Un autre élément qui a compliqué la constitution des fichiers TSV est la manière dont sont construits les fichiers XML. En effet, dans les fichiers XML d'origine, un texte est divisé en plusieurs balises paragraphes (<p>...</p>) et peut être entrecoupé de balises orphelines :



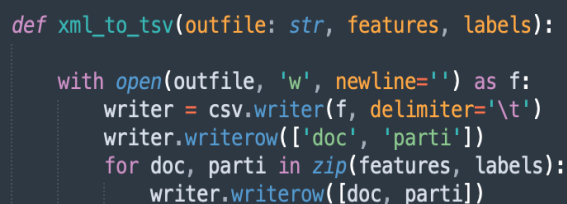
```
</EVALUATION>
<texte>
<p>Madame la Présidente, au nom des <anonyme /> de la commission de l'industrie, je voudrais faire quelques brefs commentaires. La Commission a le mérite de vaincre l'inaction et de contribuer à l'apport d'une réponse politique. Le Parlement demande une action effective et cohérente pour le secteur. Nous savons que la cause des accidents, qui se multiplient et s'aggravent malgré les technologies dont nous disposons, découle de l'écart concurrentiel dans la construction navale, qui entraîne une diminution de la qualité et une gestion économique absurde des flottes âgées, dont le point culminant se traduit par une stratégie de réduction draconienne de l'entretien et des équipages.</p>
<p>Les accidents s'expliquent non seulement par l'âge et le type de navire, mais aussi par le manque de qualité des armateurs et des opérateurs, qui recourent à tous les moyens disponibles pour réduire les coûts, moyens qui ont été mentionnés ici et qui vont des pavillons de complaisance à la complicité des autorités portuaires et
```

Fig 1 : Capture d'écran du fichier *deft09-parlement_appr_fr.xml*

Il est donc nécessaire à la fois de faire en sorte de récupérer tous les paragraphes qui constituent un seul texte, mais également de s'assurer qu'un paragraphe ne soit pas coupé au niveau des balises orphelines, ce qui s'est avéré une tâche plus compliquée que prévue. Les essais avec le module *lxml* se sont avérés peu concluants, notamment dû aux balises orphelines qui empêchaient la récupération de la seconde partie d'un paragraphe.

Le passage du XML a donc été effectué grâce à l'API *ElementTree XML*, qui nous a permis de passer outre ces problèmes. Dans le fichier *data_formating.py*, la fonction *xml_to_list* nous permet de récupérer chacun des textes associés à chacun de leur label : si le fichier est de type *train*, la fonction récupère les données contenues dans les balises <p> ainsi que les données contenues dans l'attribut *valeurs* des balises <PARTI>. Si le fichier est de type *test*, la fonction récupère les *features* des balises <p>, puis les labels du fichier texte grâce à un simple *split()*. Dans les deux cas, la fonction renvoie deux listes : une liste *features* contenant les textes, et une liste *labels* contenant les partis.

Puisqu'une liste est un élément ordonné, nous savons qu'un élément à la position 0 d'une liste est associé à l'élément à la position 0 de l'autre liste, etc. Il nous reste donc seulement à envoyer les listes dans un fichier TSV, chose très facile à faire grâce au module *csv* :



```
def xml_to_tsv(outfile: str, features, labels):

    with open(outfile, 'w', newline='') as f:
        writer = csv.writer(f, delimiter='\t')
        writer.writerow(['doc', 'parti'])
        for doc, parti in zip(features, labels):
            writer.writerow([doc, parti])
```

Fig 2 : Fonction permettant de formater les données XML dans un fichier CSV

Nous pouvons évaluer la durée et l'avancement du *parsing* du XML grâce à la bibliothèque *tqdm*. Le *parsing* dure environ 4 minutes pour le fichier d'entraînement, et environ 2 minutes pour le fichier de test sur nos Mac M1 2020. La constitution de fichiers TSV permet donc d'éviter ce temps de traite-

ment à chaque lancement du script d'entraînement (l'importation des données avec un fichier TSV n'est pas instantanée, mais plus rapide).

3 Preprocessing

3.1 Distribution des données

La première étape de notre *preprocessing* consistait en la normalisation de nos données. En effet, si nous visualisons les données, nous remarquons qu'elles sont très biaisées pour certaines catégories, et qu'il y a donc des risques de biais lors des prédictions après l'apprentissage.

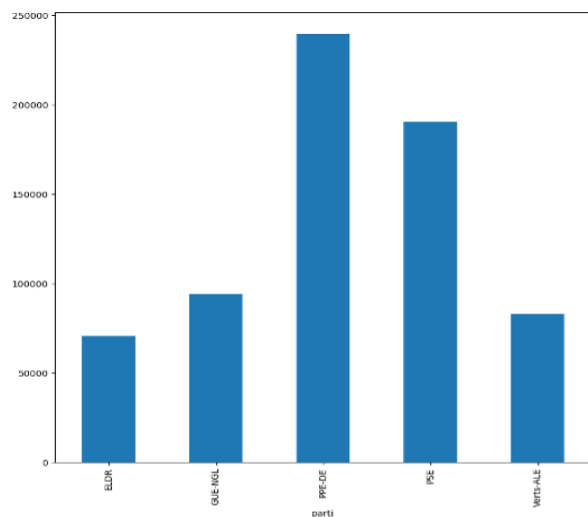


Fig 3 : Distribution de chaque catégorie du corpus original

Nous remarquons que les catégories PPE-DE et PSE sont particulièrement sur-représentées. Deux solutions se présentent donc à nous afin de diminuer ce biais : l'*oversampling* et l'*undersampling*. Théoriquement, les deux sont possibles dans notre cas car nous ne prenons pas la totalité du corpus (seulement 35% pour l'entraînement). La première consisterait à récupérer certaines des données que nous n'avons pas gardées dans notre corpus d'entraînement afin de l'alimenter en catégories

minoritaires. Cependant, nous n'avons pas gardé cette technique pour des questions de mémoires insuffisantes. Le corpus original étant très large, nous avons gardé la technique de l'*undersampling*. Afin de compenser les pertes de données importantes, nous avons augmenté le pourcentage de base du corpus d'entraînement que nous conservons (de 25% sans *undersampling* à 35% avec). Cette technique nous convient très bien car bien que nous aurions pu garder jusqu'à 50% des données d'entraînement (le pourcentage maximum supporté par nos machines), nous avons remarqué qu'au-dessus de 35%, les résultats ne s'améliorent pas. Voici une visualisation de notre corpus d'entraînement après l'*undersampling* :

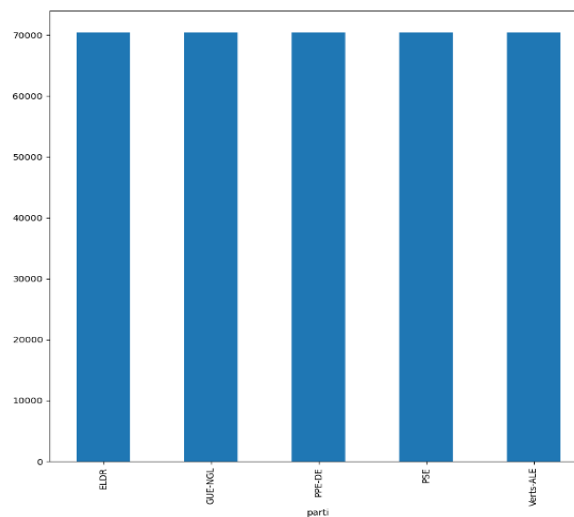


Fig 4 : Distribution de chaque catégorie du corpus après l'*undersampling*

3.2 TF-IDF

Nous avons opté pour la méthode des *bag-of-words*, avec la méthode de pondération *tf*idf*, afin d'attribuer moins d'importance aux mots qui apparaissent souvent dans le corpus, et qui ne seraient pas d'une grande aide à la distinction entre les différents partis (par exemple, la majorité des textes du corpus commencent par *Monsieur le Président*).

L'objet *TfidfVectorizer* du module *scikit-learn* permet de faciliter l'étape de *preprocessing*. Nous pouvons choisir de modifier (ou non) les paramètres suivants :

- *preprocessor* : paramètre auquel il est possible de passer une fonction de *preprocessing*, et donc d'outrepasser la fonction de *preprocessing* native. Nous n'avons pas touché à ce paramètre car nos propres fonctions n'apportaient pas de meilleurs résultats par rapport aux options fournies par la fonction native.

- *lowercase* : paramètre qui prend une valeur booléenne – *True* convertit toutes les chaînes de caractères en minuscule avant l'étape de tokenisation. Par défaut, *lowercase* est égal à *True*. Nous n'avons pas changé ce paramètre, bien qu'il ne semble pas faire une différence particulière dans les performances du modèle.

- *strip_accents* : comme son nom l'indique, il permet d'enlever les accents. Ce paramètre est par défaut égal à *None*, mais peut prendre comme valeur *ascii* (permet de faire correspondre chaque caractère à un caractère présent dans la table ASCII) et *unicode* (correspondance avec les caractères de la table Unicode). Changer la valeur de ce paramètre dégradait énormément les performances du modèle ; nous avons donc fait le choix de garder les accents.

- *token_pattern* : ce paramètre nous permet de choisir la manière de tokeniser le texte. Nous avons utilisé une expression régulière qui permet de garder les mots composés séparés par des tirets, trouvée sur ce notebook, ce qui a beaucoup amélioré le modèle.

- *ngram_range* : ce paramètre prend en entrée un tuple et permet de changer les *ngrams* extraits. Par exemple, *ngram_range=(1,1)* extrait des unigrammes, *ngram_range=(1,2)* extrait des unigrammes et des bigrammes, *ngram_range=(2,2)* extrait des bigrammes, etc. Nous avons choisi *ngram_range=(2,2)* car c'est celui qui nous fournissait les meilleurs résultats.

- *analyzer* : permet de définir le type de *features* considéré. Par défaut, ce paramètre est égal à *word* (il considère donc les mots lors de la création de *ngrams*), mais peut être également égal à *char* ou *char_wb* (bytes). Les options *char* et *char_wb* dégradaient énormément les performances du modèle

; nous avons donc gardé la valeur par défaut.

D'autres paramètres spécifiques au calcul du *tf*idf* ont été considérés ou modifiés :

- *min_df* : ce paramètre définit le nombre minimal d'apparition d'un mot dans les documents nécessaire pour être considéré. Par défaut, il est égal à 1, ce qui signifie que si un mot apparaît dans un seul document, il n'est pas considéré. Nous l'avons défini à 5. Ce paramètre a eu un impact positif sur les performances du modèle.

- *max_df* : ce paramètre permet d'ignorer les termes qui apparaissent dans *n* pourcents des documents. Nous avons défini une valeur de 0.3 pour ce paramètre. Il a également eu un impact positif sur les performances du modèle.

4 Modèle et résultats

4.1 Présentation des métriques : matrice de confusion et rapport de classification

Nous avons testé de nombreux modèles de la librairie *scikit-learn* (*DecisionTreeClassifier*, *MultinomialNB*, *GaussianNB*, etc.), mais celui qui nous fournissait les meilleurs résultats était la régression logistique. Nous avons spécifié que nous voulions effectuer une classification multinomiale avec le paramètre *multi_class*, et nous avons augmenté le nombre d'itérations maximum possibles afin de permettre au modèle de converger. Avec un simple *undersampling* et les étapes de *preprocessing* détaillées plus haut, nous sommes parvenus à un résultat de 81% de précision :

Nous remarquons que les faux positifs sont plus fréquents que les faux négatifs, notamment sur les catégories les moins représentées dans le corpus de test (ELDR et Verts-Ale).

Par curiosité, nous avons testé le modèle sur des données de test où chaque catégorie est représentée à part égale, comme lors de l'entraînement.

Nous remarquons que cela améliore énormément les résultats sur la précision pour ELDR et Verts-Ale (les catégories les moins représentées originelle-

Table 1: Rapport de classification sur des données à distribution variable

	P	R	F1	Sup.
ELDR	0.86	0.70	0.77	462
GUE-NGL	0.87	0.86	0.86	641
PPE-DE	0.80	0.87	0.83	1596
PSE	0.78	0.80	0.79	1244
Verts-Ale	0.84	0.72	0.78	578
accuracy			0.81	4521
macro avg	0.83	0.79	0.81	4521
weighted avg	0.81	0.81	0.81	4521

ment), mais ils baissent énormément pour PPE-DE et PSE. On remarque une corrélation entre les mauvais résultats de la précision de PPE-DE et PSE et les mauvais résultats du recall d'ELDR et Verts-Ale : le modèle a tendance à sur-prédire ELDR et Verts-Ale. Cette perte dans la précision de PPE-DE et PSE ne permet pas de compenser les bons résultats sur ELDR, Verts-Ale et GUE-NGL, ce qui nous laisse penser que le modèle a plus de facilité à prédire correctement les discours prononcés par PPE-DE et PSE, mais a tendance à sur-prédire ELDR et Verts-Ale.

La matrice de confusion (pour le modèle entraîné sur des données à distribution variable) nous donne plus d'information sur la nature des erreurs du modèle :

Table 2: Rapport de classification sur des données à distribution égale

	P	R	F1	Sup.
ELDR	0.93	0.70	0.80	462
GUE-NGL	0.90	0.86	0.88	462
PPE-DE	0.64	0.85	0.73	462
PSE	0.69	0.80	0.74	462
Verts-Ale	0.88	0.71	0.79	462
accuracy			0.79	2310
macro avg	0.81	0.79	0.79	2310
weighted avg	0.81	0.79	0.79	2310

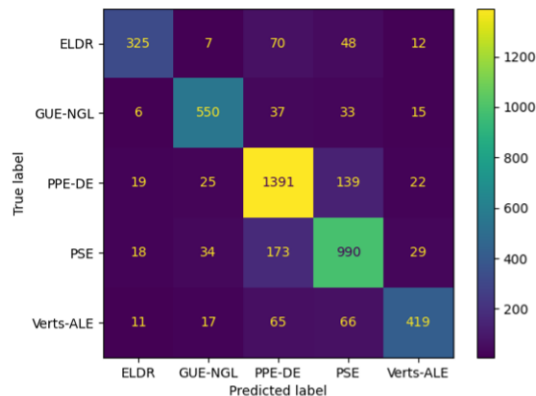


Fig 5 : Matrice de confusion pour le modèle entraîné sur des données à distribution variable

En général, on remarque que le plus grand nombre d'erreur se trouve entre PPE-DE et PSE, les données les plus représentées dans le corpus original. On remarque également des erreurs importantes entre ELDR/PPE-DE, ELDR/PSE, Verts-Ale/PPE-DE et Verts-Ale/PSE. Notre hypothèse selon laquelle le modèle a tendance à attribuer les discours de PPE-DE et de PSE à d'autres partis semble étayée par cette matrice. Nous avons donc souhaité tester une dernière théorie : celle que l'entraînement avec un léger biais pour les valeurs PPE-DE et PSE pourrait aider le modèle à mieux les reconnaître. Nous

avons donc fait un dernier entraînement, cette fois avec un corpus de train légèrement biaisé en faveur de PPE-DE et PSE. Voici une représentation de la distribution des données après introduction du biais :

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

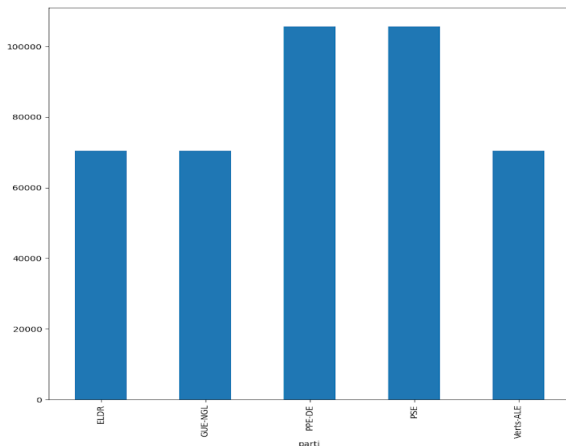


Fig 6 : Visualisation des données après introduction d'un biais sur les données PPE-DE et PSE dans le corpus d'entraînement

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consecte-

tuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nos tests ont montré qu'introduire un biais de moins de 50% ne permettait pas d'améliorer les résultats, notamment à cause d'une légère perte de *recall* pour chacune des catégories non compensées par la hausse dans la précision des autres catégories. Cependant, lorsque le biais reste entre 50 et 60% de la taille de la catégorie *baseline* utilisé pour l'*undersampling* (ELDR), il permet en effet d'améliorer légèrement les résultats, du aux améliorations de la précision pour toutes les catégories. L'amélioration n'est cependant que très légère : la précision pour toutes les catégories n'augmente que d'un pourcent, de 81 à 82%.

Table 3: Rapport de classification sur des données à distribution variable, avec introduction de biais sur les catégories PSE et PPE-DE dans le jeu d'entraînement

	P	R	F1	Sup.
ELDR	0.89	0.69	0.78	462
GUE-NGL	0.90	0.85	0.87	641
PPE-DE	0.79	0.88	0.83	1596
PSE	0.77	0.81	0.79	1244
VERTS-ALE	0.80	0.71	0.79	462
accuracy			0.82	4521
macro avg	0.85	0.79	0.81	4521
weighted avg	0.82	0.82	0.81	4521

La matrice de confusion nous montre que ce biais n'a que légèrement amélioré les résultats, mais étonnamment, les améliorations ne se sont pas produites dans la différentiation entre PPE-DE, PSE et ELDR, mais entre PPE-DE/Verts-Ale/GUE-NGL et PSE/Verts-Ale/GUE-NGL :

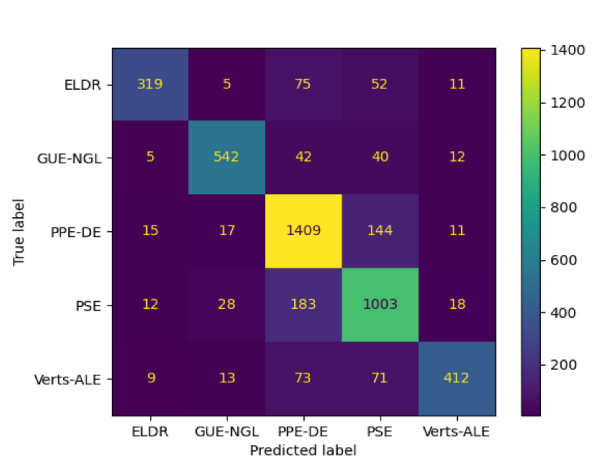


Fig 7 : Matrice de confusion pour le modèle entraîné sur des données biaisées pour PPE-DE et PSE

En fait, cette amélioration est trompeuse, car l'introduction d'un biais ne permet ici que de prendre avantage des données de test déjà biaisées en faveur de PPE-DE et de PSE : le modèle devient meilleur à prédire ces deux résultats, et donc performe légèrement mieux sur les données de test lorsqu'elles ont beaucoup de valeurs PPE-DE et PSE à prédire. Mais si le nombre d'occurrence catégories du corpus de test est égalisé, ce biais n'apporte rien de à la précision finale, si ce n'est une meilleure chance de prédire PPE-DE et PSE correctement (contre des prédictions moins bonnes pour ELDR). Les bons résultats sur ce dernier modèle, avec introduction de biais, sont donc en majorité du au biais déjà existant dans les données de test. En l'occurrence, ceci est bénéfique pour nous ; mais si nous souhaitons créer un modèle performant pour toutes les catégories, il ne suffit pas d'avoir un modèle très bon pour la prédiction de certaines catégories, et très mauvais pour d'autres. Le choix du meilleur modèle devra donc se baser sur des observations empiriques ; est-ce que, en règle générale, il y a une probabilité plus im-

portante pour que les partis PPE-DE et PSE prononcent un discours au parlement européen par rapport aux autres partis ? Dans ce cas, un modèle qui a de moins bons résultats sur les autres catégories mais d'excellents résultats sur PPE-DE et PSE pourrait être bénéfique. Dans notre cas, et en l'absence de d'informations contextuelles, nous considérons notre meilleur modèle comme celui sans biais, avec une précision de 81% sur les données de test biaisées et 79% sur les données de test non biaisées.

4.2 Observation des prédictions

Afin d'observer les résultats du modèle sur les données de test, nous effectuons une prédiction sur n discours du corpus de test (en l'occurrence 50, mais le nombre peut être modifié 1.147 du fichier *classification-model.py*) que nous formatons dans un fichier Excel (dans *eval/test_model_predictions.xlsx*). Ce fichier contient le texte, les prédictions du modèle ainsi que les réponses attendues.

A1												
	A	B	C	D	E	F	G	H	I	J	K	L
1		text	prediction	true								
2	0	Tout en app PSE	PSE									
3	1	Clarifier la i GUE-NGL	GUE-NGL									
4	2	On ne s'atti PSE	PSE									
5	3	Monsieur li PPE-DE	PPE-DE									
6	4	Monsieur li PPE-DE	PPE-DE									
7	5	Monsieur li PSE	PSE									
8	6	Monsieur li ELDR	ELDR									
9	7	Monsieur li PPE-DE	PPE-DE									
10	8	Monsieur li GUE-NGL	GUE-NGL									
11	9	Monsieur li PSE	PSE									
12	10	Monsieur li Verts-ALE	Verts-ALE									
13	11	Monsieur li PPE-DE	ELDR									
14	12	Monsieur li PPE-DE	PPE-DE									

Fig 8 : Capture d'écran du fichier *eval/test_model_predictions.xlsx*

5 Conclusion

La tâche de détection des partis politiques étant très complexe, notre meilleur modèle a atteint une précision de 81%. Ce résultat est satisfaisant en vue des résultats obtenus sur la même tâche par des testeurs humains (source) :

Rappel/précision. Pour chaque testeur humain, nous avons calculé le rappel et la précision qu'il a obtenu pour chacun des partis à identifier et avons ensuite calculé un macro-rappel ainsi qu'une macro-précision. Le tableau 8 montre les valeurs de macro-rappel et macro-précision obtenues par chaque testeur humain.

Testeur	1	2	3	4	5	6
Rappel	0,41	0,35	0,39	0,23	0,47	0,35
Précision	0,42	0,34	0,37	0,27	0,42	0,34

TAB. 8 – Rappel et précision obtenus par les testeurs humains sur la tâche d'identification des partis politiques.

Fig 9 : Résultats sur la tâche de classification de texte en fonction des partis politiques effectuée par des annotateurs humains

Nos résultats sont également supérieurs à ceux obtenus par l'équipe de Montréal qui a participé au défi en 2009 :

Soumission	Type de valeurs	ELDR	GUE-NGL	PPE-DE	PSE	Verts/ALE	F-mesure
Nombre de documents attendus		1338	1794	4571	3626	1585	
1	Rappel	0,189	0,393	0,437	0,360	0,233	0,320
	Précision	0,210	0,345	0,447	0,365	0,226	
2	Rappel	0,231	0,332	0,498	0,394	0,207	0,339
	Précision	0,236	0,422	0,452	0,370	0,252	
3	Rappel	0,202	0,376	0,462	0,383	0,243	0,334
	Précision	0,205	0,384	0,462	0,369	0,255	

TAB. 10 – Rappels et précisions obtenus par parti politique pour chacune des trois soumissions de la tâche 3 par l'équipe de Montréal.

Fig 10 : Résultats obtenus par l'équipe de Montréal

Il serait cependant intéressant d'aller plus loin en observant les résultats avec du Deep Learning, notamment les CNN, RNN et BiLSTM qui sont particulièrement adaptés à la tâche de classification multi-classe de textes.