Universite de Montpellier

Rapport de projet
Fact Cheking classification

Belkassim BOUZIDI 21715636
Chakib ELHOUITI 21813619
Massili KEZZOUL 21815514

18 mai 2021

1 Introduction

Ce projet a pour but de proposer des modèles de classification supervisée d'assertions faites par des figures politiques selon leur valeur de véracité, ou autrement dit, de proposer une approche de fact-checking automatique. Pour pouvoir ensuite analyser les résultats et en faire une comparaison entre ces modèles selon plusieurs paramètres de variabilité.

2 Compréhension des données

2.1 Les données

Le jeu de données utilisé est ClaimsKG. Il a été collecté à partir de sites de fact-checking par le LIRMM en collaboration avec plusieurs équipes de recherche européennes. Il est décrit en détail ici : https ://data.gesis.org/claimskg/site/.

2.2 Nettoyage des données

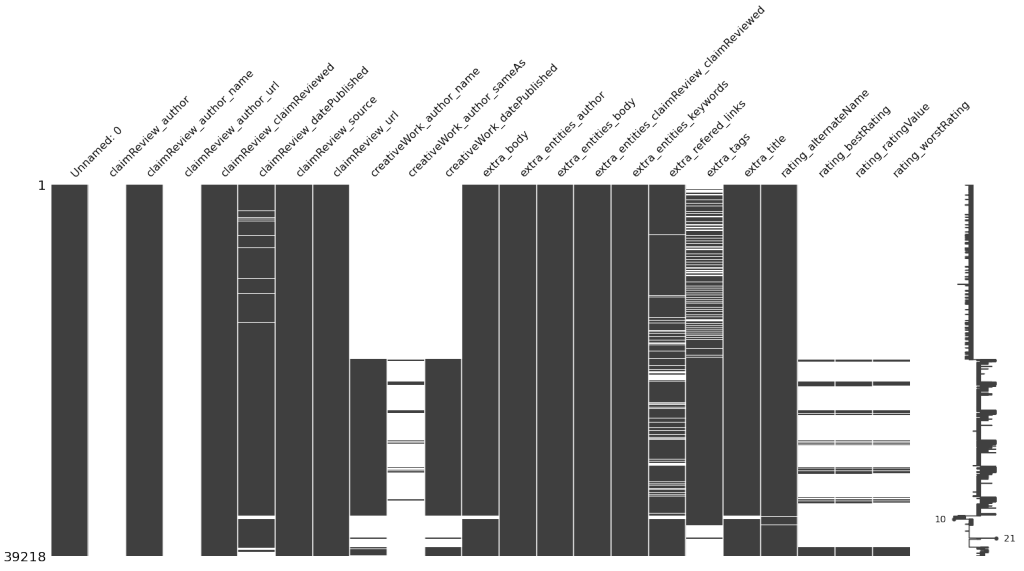
Dans un premier temps, on a commencé par prendre en mains les données fournis et voir si on peut les nettoyer et de garder seulement les données nécéssaires.

2.2.1 Les données manquantes

Certaines colonnes du fichier csv des données sont complétement ou majoritairement manquantes, On voit bien sur le schéma ci-dessous les données qui manquent :

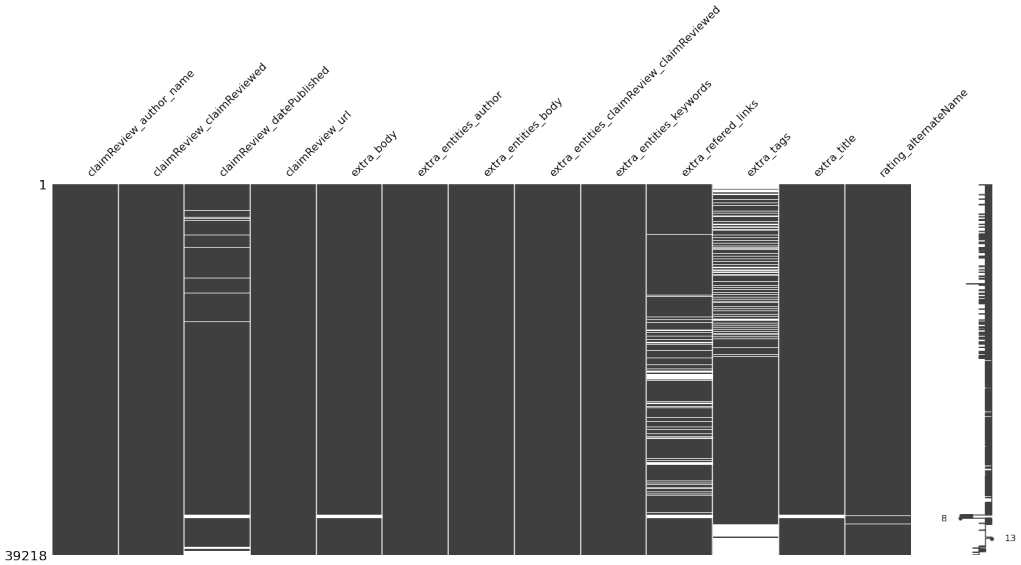
- ClaimReview_author
- ClaimReview_author_url
- creativeWork_author_name
- creativeWork_author_sameAs
- creativeWork_datePublished
- rating_bestRating
- rating_ratingValue
- rating_worstRating

Schéma 1 – Données brutes



Donc on a décidé de supprimer les colonnes manquantes car ils ne nous servirons pas pour entraîner notre modèle, on a obtenu le schéma ci-dessous

Schéma 2 – Données sans colonnes manquantes



2.2.2 Les doublons

On a constaté qu'il y a un certain nombre de lignes redondantes (5749). Il faut alors les supprimer parcequ'elles risquent d'apporter une certaine imprécision.

2.2.3 Autres erreurs dans les données

En parcourant les données, on a remarqué quelques incohérences dans les données qu'on préfère éliminer dès maintenant. Notamment, certaines lignes contiennent comme claim **true** ou **false**.

3 Prétraitements des données

Une fois les données nettoyées, on passe à leur prétraitement. En effet, on ne pourra pas passer les données brutes directement au modèle. Donc on a définit un ensemble de prétraitements :

3.1 Tokenisation

Découpage de l'assertion en Token (en mots).

3.2 Mise en miniscule

Ça consiste à mettre chaque lettre de chaque mot en miniscule (lowercase).

3.3 Traitement des contractions et ponctuations

La suppression des ponctuations peut avoir des conséquences sur le qualité du modèle, par exemple dans la détection des opinions. Il est préferable de traiter d'abord les contractions dans

les phrases avant de supprimer les ponctuations (ex : couldn't).

3.4 numbers to words

Transformer les nombres en mots. Il faut utiliser ce prétraitement avant de traiter les ponctuations pour ne pas avoir des données erronées (par exemple 15,25 deviendra 15 et 25 si on utilise le traitement des ponctuations avant).

3.5 Stopwords

Supprimer les mots les plus fréquent de la langue. Dans notre cas :'the', 'a', 'an', 'in' ...
NB : Dans les stopwords fournis par défault par NLTK contiennent les formes de négation.

3.6 Pos-tagging

L'étiquetage morpho-syntaxique est le processus qui consiste à associer aux mots d'un texte les informations grammaticales correspondantes comme la partie du discours, le genre, etc.

3.7 Stemmatisation

Le stemmatisation (racinisation en français) vise à garder la racine du mot. La racine d'un mot correspond à la partie du mot restante une fois que l'on a supprimé son (ses) préfixe(s) et suffixe(s), à savoir son radical. Plusieurs variantes d'un terme peuvent ainsi être groupées dans une seule forme représentative. Il existe plusieurs algorithmes de stemmatisation, on a utilisé SnowBall Stemmer. Mais il existe aussi Lancaster Stemmer qui est considéré comme plus agressif.

3.8 Lemmatisation

La stemmatisation et la lemmatisation sont deux notions proches, mais il y a des différences fondamentales entre eux. La lemmatisation a pour objectif de retrouver le lemme d'un mot, par exemple l'infinitif pour les verbes. La racinisation consiste à supprimer la fin des mots, ce qui peut résulter en un mot qui n'existe pas dans la langue.

NB : La lemmatisation foncionnent beaucoup mieux si chaque mot vient avec son tag parts-of-speech (POS).

Au final, on a réalisé une fonction de prétraitements qui va utiliser une combinaison des fonctions citées plus haut. L'idée est donc d'entraîner le modèle avec quelques combinaisons pour voir leurs effets sur le processus d'apprentissage. Cette fonction va prendre en paramètre un texte brut et donner en résultat une liste de token, potentiellement accompagné avec leur tag POS.

Dans le cas pratique, nous avons mis en œuvre une classe *TextPreTraitement* qui sera plus facile à utiliser qu'une fonction.

4 Vectorisation

L'objectif de la vectorisation est de transformer les documents en vecteurs. Il existe deux approches principales :

- L'approche sac de mots dans laquelle il n'y a aucun ordre dans les termes utilisés et qui ne tient compte que du nombre d'occurrences des termes
- L'approche basée sur TF_IDF qui ne tient pas non plus compte de l'ordre des termes mais qui pondère les valeurs grâce à TF_IDF au lieu de la fréquence des termes.

NB : même si l'ordre des mots n'est pas pris en compte dans ces approches, les n-grammes peuvent partiellement servir à pallier ce problème.

Dans notre cas, on a utilisé TF_IDF : Le but de l'utilisation de tf-idf est de réduire l'impact des termes qui apparaissent très fréquemment dans un corpus donné et qui sont donc moins informatifs que les autres termes dans le corpus d'apprentissage.

CountVectorizer, en prenant en compte l'occurrence des mots, est souvent trop limité. Une alternative est d'utiliser la mesure TF-IDF (Term Frequency – Inverse Document) qui a pour but de réduire l'impact des termes qui apparaissent très fréquemment dans un corpus donné :

$tf\text{-}idf(d,t) = tf(t) * idf(d,t)$

où $tf(t)$= la fréquence du terme, i.e. le nombre de fois où le terme apparaît dans le document et $idf(d,t)$ = la fréquence du document, i.e. le nombre de documents 'd' qui contiennent le terme 't'.

Le principe est le même que pour CountVectorizer, cette opération se fait par :

— Création d'une instance de la classe TfidfVectorizer.
— Appel de la fonction fit() pour apprendre le vocabulaire.
— Appel de la fonction transform() sur un ou plusieurs documents afin de les encoder dans le vecteur.

Par exemple, en appliquant TF_IDF pour ces phrases [”This is an example, ! of CountVectorizer for creating a vector”, ”This is another example of CountVectorizer”, ”with or without parameters”] on obtient :

Schéma 3 – Vectorisation

	CountVectorizer for	This is	an example	another example	creating vector	example of	for creating	is an	is another	of CountVectorizer	or without	with or	without parameters
0	0.385323	0.293048	0.385323	0.00000	0.385323	0.293048	0.385323	0.385323	0.00000	0.293048	0.00000	0.00000	0.00000
1	0.000000	0.393511	0.000000	0.51742	0.000000	0.393511	0.000000	0.000000	0.51742	0.393511	0.00000	0.00000	0.00000
2	0.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.00000	0.000000	0.57735	0.57735	0.57735

5 Classification

Cette étape consiste à appliquer des algorithmes de machine learning pour pouvoir classifier nos données prétraitées.

5.1 Classifieurs

Comme l'indique le **No free lunch theorem**, il n'existe pas de classifieur universel. Il est donc toujours nécessaire d'en évaluer plusieurs afin de trouver le plus pertinent pour notre situation.

Les algorithmes testés sont les suivants :
— SVM
— Naive Bayes
— Decision Tree
— K-nearest neighbors
— Logistic Regression

5.2 Découpage en groupes

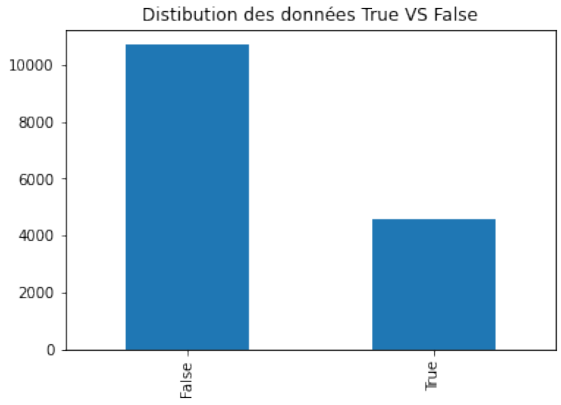
On a commencé par découper nos données en groupes de classification, afin que, pour chaque groupe, on applique les algorithmes de machine learning.

Dans notre cas, nous générons trois groupes de données :
— ‘kg_tf‘ contient les deux classes True → ‘1‘ et False → ‘0‘.
— ‘kg_tf_m‘ contient les deux classes True-False → ‘0‘ et Mixture → ‘1‘.
— ‘kg_tfm‘ contient les trois classes True → ‘1‘, False → ‘0‘ et Mixture → ‘2‘

5.3 Équilibrage des données

Comme on a des données de plusieurs classes, c’est possible que nos données ne soient pas équilibrées, comme le montre la figure suivante : On remarque bien que le nombre d’assertions

Schéma 4 – True VS False

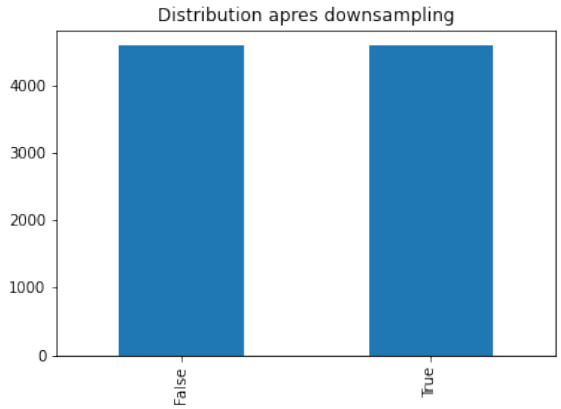


pour chaque classe est mal balancé. Ce qui peut conduire à un mauvais apprentissage de la part des modèles. En effet, par exemple, dans la tâche True VS False, il y a 70% d’assertions False. le modèle va donc être biaisé et être moins performant pour la prédiction des assertions False.

On a donc mis en point une fonction de sous-échantillonnage (Downsampling) pour remédier à ce problème.

On obtient cette figure :

Schéma 5 – True VS False après Downsampling



5.4 Jeu d'apprentissage et de test

Dans cette étape nous procédant à la création d'un jeu d'apprentissage et de test en utilsant la fonction 'train_test_split' de la librairie 'sklearn'.

Nous avons choisis de reserver 70% des données pour l'apprentissage et les 30% restantes pour l'étape de validation.

5.5 Prétraitements

Dans cette étape, On va utiliser la classe des prétraitements définit plus haut (TextPreTraitement), on a décidé de partir, sur une classe qui contient des prétraitements, qu'on juge nécéssaires (lowerCase, tokenize, number2word) et d'en rajouter à chaque fois un prétraitement (ponctuations, contractions, ...), pour voir comment les algorithmes réagissent avec ce prétraitement.

5.6 Vectorisation

L'objectif ici est de transformer l'ensemble des assertions (composées de mots) en un ensemble de vecteurs pour que les modèles puissent effectuer des opérations mathématique dessus. Nous utiliserons principalement TF-IDF pour la vectorisation des données.

5.7 Application des Algorithmes et résultats

Dans cette étape, on a utilisé la fonction gridSearchCV pour avoir les meilleurs paramètres et le meilleur score de chaque algorithme pour chaque prétraitement.
On obtient les graphes suivants (pour chaque groupe) :

Schéma 6 – Comparaison des algorithmes pour la tâche TRUE VS FALSE

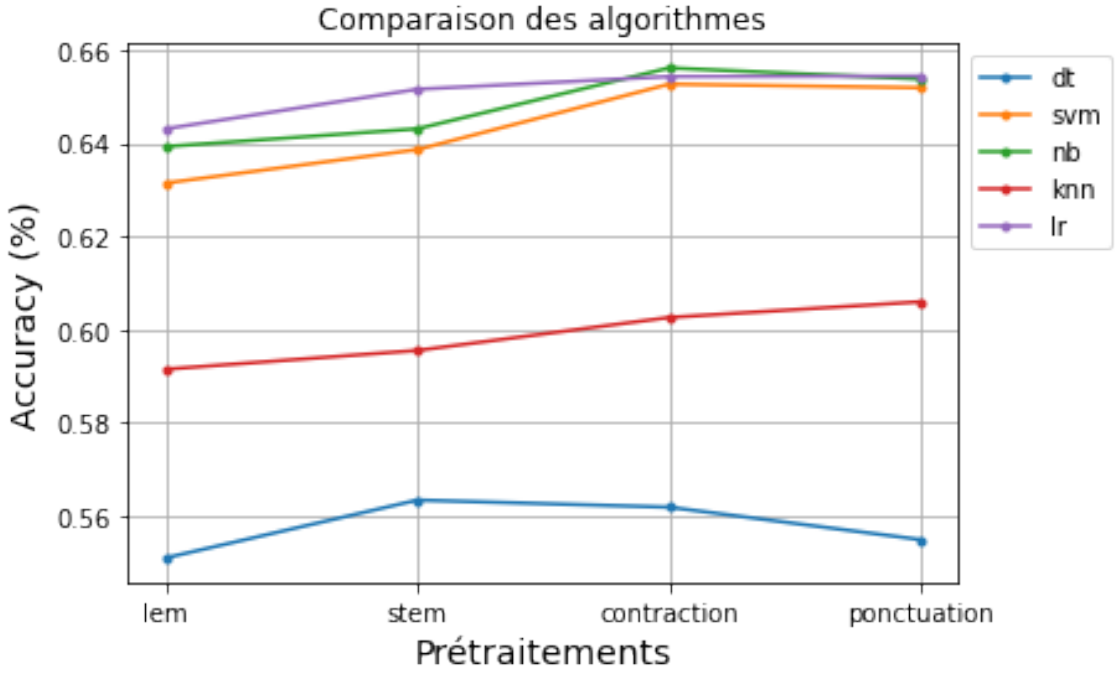


Schéma 7 – Comparaison des algorithmes pour la tâche TRUE et FALSE VS MIXTURE

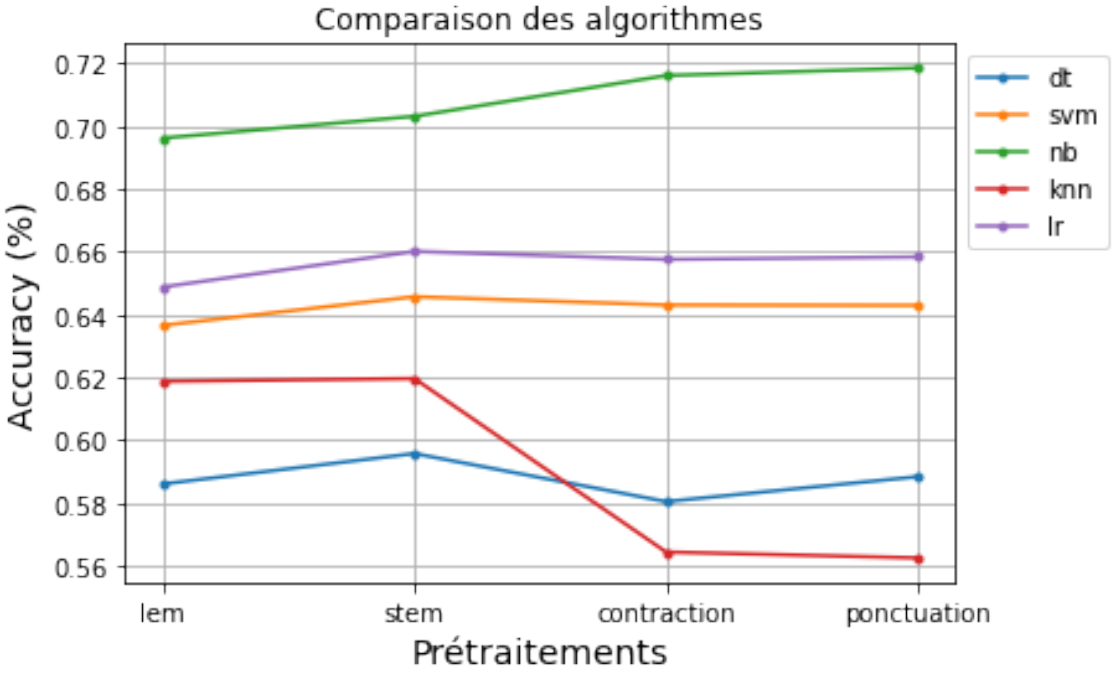
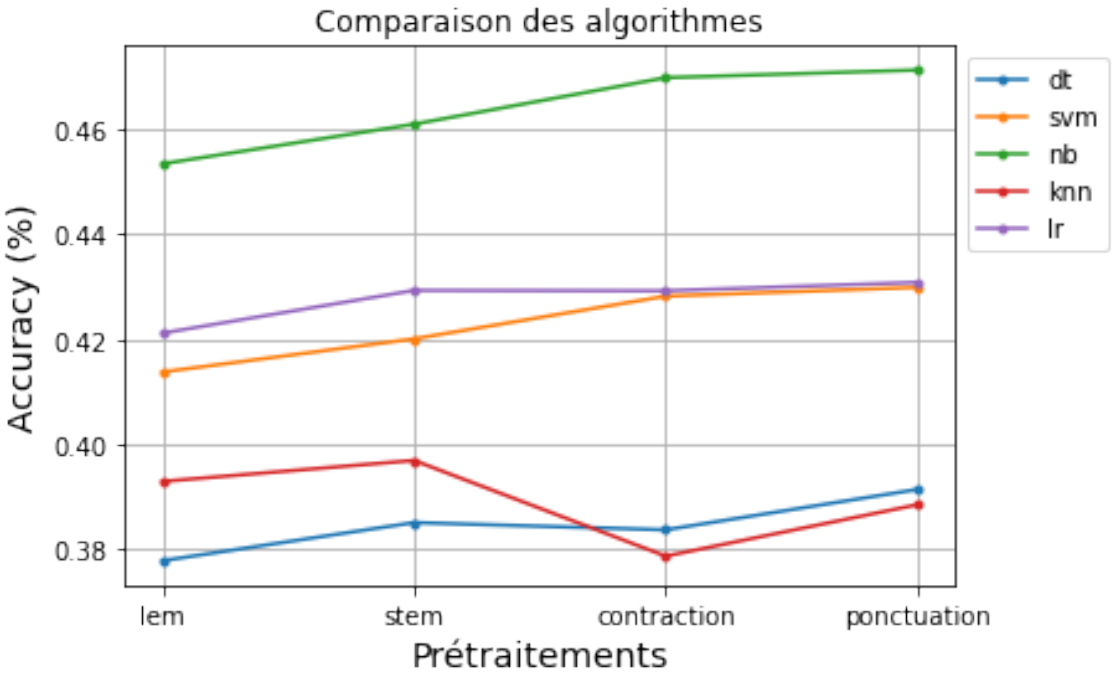


Schéma 8 – Comparaison des algorithmes pour la tâche TRUE VS FALSE VS MIXTURE

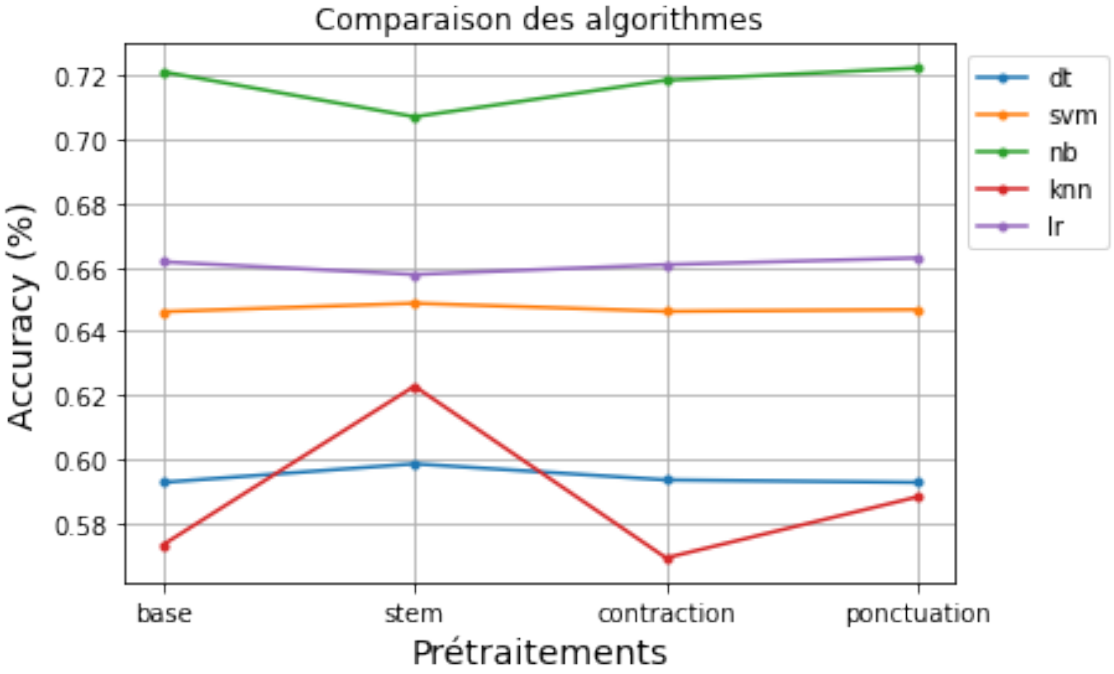


5.8 Analyse des résultats

Premièrement, la performance globale des algorithmes diminue dans la tâche à trois catégories (TRUE, FALSE, MIXTURE), par rapport aux deux tâches binaires avec une pointe atteinte par Naive Bayes à 72%. Cela peut s'éxpliquer par le faite que l'ajout d'une troisième catégorie complexifie le modèle ce qui induit à de moins bons résultats.
On remarque bien que pour chaque tâche de classification, le SVM, Logistic Regression et le Naive bayes sont bien meilleurs que le KNN ou le Decision Tree qui donnent de mauvais résultats quelque soit le prétraitement choisi. On remarque aussi que la stemmatisation est meilleure que la lemmatisation quelque soit l'algorithme de classification et la tâche de classification.

Le score du KNN et Decison Tree diminue à chaque fois qu'on enlève la lemmatisation et/ou la stemmatisation, par contre, la stemmatisation influe négativement sur la performance du Naive Bayes, ce qu'on peut voir à partir du graph suivant :

Schéma 9 – L'influence de la stemmatisation



6 Conclusion

Ce projet nous a permis de développer nos connaissances, dans le domaine de la science des données, notamment, de savoir faire du machine learning sur un nombre important de données en appliquant les différent algorithmes et de pouvoir enfin les comparer en terme de performance selon différents critères.