## Rapport du cours 3335 :

## TP2 - classification et désambiguïsation de sens

Mahmoud Labidi, 20176755

Yacine Mkhinini , 20170474

Avant de commencer la classification et désambiguïsation de sens sur le corpus données, nous allons effectuer des préparatifs sur un ensemble de données existant dans le package afin de se familiariser avec la méthodologie de traitement. Ces préparatifs se décomposent en trois parties :

* Prétraitement
* Classification
* Sélection et pondération des features

Nous allons donc expliquer en détails chacune de ces étapes que nous avons appliqué sur notre corpus :

1. Prétraitement :

Pour le traitement de donnée nous étions partagées entre deux approches différentes :

La première consiste a travailler avec countvectorizer et TFIDVectorizer comme nous l’avions appris dans les préparatifs. Cependant, dans la première approche nous ignorons l’ordre des mots dans le corpus et donc par conséquence ne prends pas en compte la structure grammaticale de la phrase.

C’est pour remédier a cela que nous avons pensé à la deuxième méthode du dictionnaire qui consiste à créer un data frame où chaque mot dans la phrase a sa colonne qui l’associe a sa position dans la phrase

Pour faire cela nous avons définis deux méthodes qui traite chacune une des deux approches expliquées précédemment :

1. formatDict (wordFilter, stemming , before , after) sachant que :

* wordFilter (Boolean) : qui contrôle si oui ou non on enlève les mots de la stop List de nos données.
* stemming (Boolean) : qui contrôle si oui ou non on fait le stemming.
* Before (Int) : qui contrôle combien de mot on retient avant le mot en question.
* After (Int) : qui contrôle combien de mot on retient après le mot en question.

Dans cette méthode on commence par un formatage du texte entier. En effet, dans un premier temps, on supprime tous les caractères inutiles ($, [,] , la ponctuation inutiles .. ) puis on effectue le stemming et le filtrage des mots si les paramètres stemming et wordFilter sont égale à True .

Dans un deuxième temps, on créer un dictionnaire qui associe une valeur numérique a touts les mots restant après le filtrage afin d’avoir des données en format numérique qui seront plus tard utiliser dans les différents modèles.

Pour finir nous allons extraire les mentions « interest » ainsi que les mots l’entourant (selon le Before et After) puis nous les mettrons dans une dataframe dans leurs colonnes correspondantes.

2. formatVic (prends les mêmes paramètres):

Nous effectuons le même formatage initial de donnée mais cette fois ils seront passé à Countvectorizer d’abord puis a un DFITVectorizer

1. Comparaison des performances des deux dataset :

On passe les deux dataset dans des modèles d’apprentissage avec les mêmes paramètres afin de trouver le dataset le plus performant et ainsi s’avoir vers quelle méthode se tourner.

Nous avons défini la méthode trainScoreModel qui facilitera la comparaison en donnant à chaque fois l’accuracy du modèle utilisé avec le dataset mis en entrée :

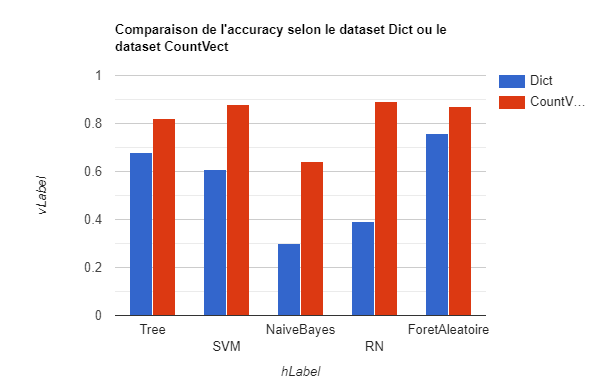
trainScoreModel (model, X, y) :

* -model : est le modèle d’apprentissage que nous allons utiliser
* -X : le dataset et donc soit avec le dictionnaire soit avec Countvectorizer
* -y : le target

On trouve les résultats suivants :

Une image contenant texte

Description générée automatiquement



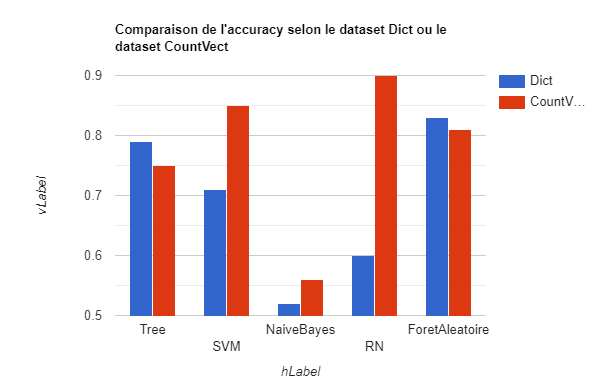
Il est clair que le dataset utilisant CountVectorizer donne de meilleurs résultats, il est donc plus performant et c’est celui la que nous allons utiliser.

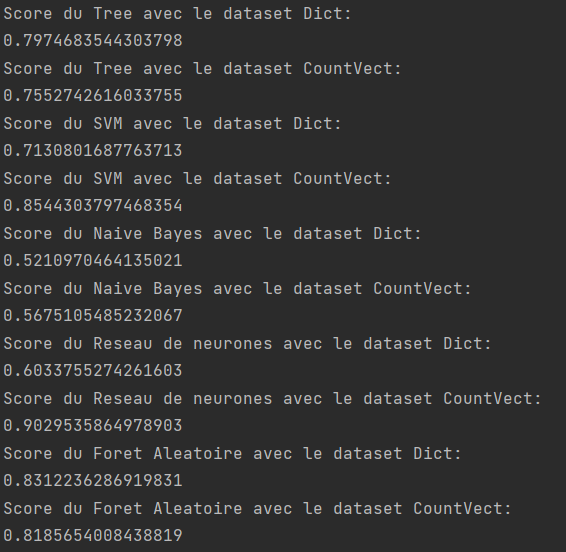
Afin de vérifier cela nous avons même comparer les deux différents dataset mais avec des paramètres différents de ceux de la première fois.

En effet avec cette combinaison :

* wordFilter = False
* stemming = False
* before = 4
* after = 4

Nous avons les résultats suivants qui confirment que le dataset qui est passé par countVectorizer est plus performant que le dataset qui utilise le dictionnaire pour n’importe quelle combinaison de paramètres :





1. Sélection et pondération des features

Nous allons étudier en particulier le réseau de neurones et le foret aléatoire avec le dataset de CountVectorizer.

Afin d’optimiser les résultats nous allons tester plusieurs combinaisons de stop list steaming before et after.

Pour le hyperparamètre tuning on va utiliser gridSearchCV avec les paramètres suivants :

Une image contenant texte

Description générée automatiquement

On définit donc la fonction hypertuneModels(model, param\_grid, X\_train, y\_train)  qui sert donc à trouver les hyperparamètres optimaux pour un modèle donnée et une range de combinaison possible .

Cas 1 :

* wordFilter = True
* stemming = True
* before = 2
* after = 2

Une image contenant texte

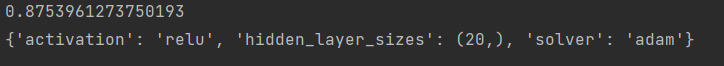
Description générée automatiquement



Cas 2 :

* wordFilter = False
* stemming = False
* before = 2
* after = 2

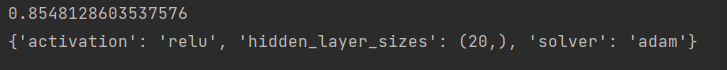
Aucune description disponible.



Cas 3 :

* wordFilter = True
* stemming = True
* before = 4
* after = 4

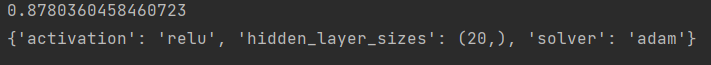
Aucune description disponible.



Cas 4 :

* wordFilter = False
* stemming = True
* before = 5
* after = 5

Aucune description disponible.



* Ainsi nous pouvons en conclure que la combinaison du cas 2 donne les meilleurs résultats.

En guise de conclusion nous pouvons dire que le fais de tester les modèles appris théoriquement de manière aussi fluide et de comparer les résultats à chaque fois était très instructif et amusant à faire.