UNIVERSITE PARIS-SUD

Master1 Informatique

Année 2017-2018

Rapport TER : Classification des Tweets

REALISE PAR :

- SI SAID Nafaa

- LASLA Youcef

Enseignant : alexandre allauzen

SOMMAIRE :

I) INTRODUCTION :

l'ordinateur peut apprendre à sa manière par des programmes d'apprentissage. Ces programmes ont pour objectif d'automatiser certaines tâches en permettant à la machine de déduire d'elle même les valeurs, ce qui permet de traiter des opérations complexes par des algorithmes relativement simples. Par exemple, prédire l'avis d'une critique d'un film à partir d'un dictionnaire de mots accompagné de la nature de chaque mot.

L'objectif de ce TER est d'étudier différents programmes d'apprentissage en nous basant sur le traitement de messages envoyés par le biais du site twitter.com. Ces messages, également nommés « tweets », seront ainsi affectés d'une valeur par la machine selon un algorithme d'apprentissage : 1 si le tweet est considéré comme positif, 0 s'il est considéré comme négatif ou neutre. On utilisera pour cela le langage de programmation Python.

Nous nous sommes inspirés des travaux d'apprentissage vu en cours, ainsi que certains modèles souvent utilisés pour répondre aux difficultés de la classification binaire en apprentissage machine.

II) DONNÉES :

Notre objectif est de donner à chaque tweet sa classification. Un tweet peut être classé comme étant positif, négatif, ou neutre. Pour simplifier, on considérera un tweet neutre comme dans la même catégorie qu'un tweet négatif.

Une grande liste de tweets a été fournie avec le sujet, plus exactement 1 578 627 tweets sur un fichier CSV. Sur chaque ligne figure le numéro du tweet, sa catégorie, ainsi que son contenu. Le contenu du message est limité à 140 caractères, étant donné que c'est une limitation imposée par Twitter même. La catégorie est représentée par un booléen. Sa valeur est de 1 pour les tweets positifs, 0 pour les tweets neutres ou négatifs.

Nous avons décidé de travailler tout d'abord avec un échantillon réduit d'exemples de tweets. Cela nous a permit d'avoir des résultats convenables avec un temps de traitement relativement court.

Pour traiter cette liste de tweets nous avons tout d'abord décidé de créer un dictionnaire de données listant tous les mots. Par la suite nous avons décidé de ne prendre en compte que les mots les plus représentatifs, c'est à dire les mots qui, dans tout le dictionnaire, apparaissent plus de 20 fois, et moins que 10 000 fois. Le test sur la borne inférieure (plus de 20 fois) nous permet de filtrer les fautes d'orthographe que les utilisateurs pourraient faire ainsi que des mots qui ne sont pas courants, et qui donc nous seraient inutiles pour l'analyse. Pour la borne supérieure nous avons décidé de prendre les mots qui apparaissent moins de 10 000 fois car les mots qui apparaissent trop souvent sont souvent neutres et ne nous donnent pas d'informations pertinentes sur la phrase.

Finalement, on omettra également tous les mots commençant par "&" et "@", car ils sont issus d'erreurs lors de la recopie des tweets. Ce sont des balises que seulement Twitter reconnaît. Les liens sont traités dans leur ensemble comme des mots individuels. Les mots vides ou constitués uniquement du caractère espace sont également ignorés.

Nous avons donc un vocabulaire qui passe de 118 257 mots, avant le filtrage, à 4121 mots. On peut donc voir que d'après nos critères seulement quelques mots nous sont utiles pour l'apprentissage.

III) MODÈLES :

Pour ce projet d'apprentissage nous avons décidé de développer deux modèles fin de pouvoir les comparer et chercher le modèle le plus optimal pour la classification de tweets : un modèle séquentiel et un modèle Bayésien Naïf. Pour chaque modèle nous avons fait de légères variations afin d'essayer d'optimiser le plus possible.

Nous avons abordé le sujet tout d'abord en créant une fonction qui transforme les données, c'est à dire chaque mot de chaque tweet, à l'indice correspondant dans notre dictionnaire. Si le mot est absent dans notre vocabulaire alors l'indice sera 1. Nous avons décidé, dans un premier temps, de mettre l'indice d'un mot absent dans notre vocabulaire à 0, cependant cela nous pose des problèmes avec la couche embedding, car celle ci reconnaît 0 comme l'absence de données au lieu du mot absent de notre vocabulaire, mais présent dans nos données.

Cette fonction nous retourne un vecteur de taille variable, par rapport à la taille du tweet original. Avec cette fonction on peut donc créer une matrice de vecteurs avec tous les tweets traités.

Grâce à ce résultat nous pouvons créer une autre fonction qui nous permettra par la suite de créer un vecteur pour chaque tweet qui, pour chaque mot, nous dit s'il est présent ou absent dans notre dictionnaire. Ce vecteur est donc composé de 1 pour mot présent et 0 pour mot absent. Et c'est avec ces données qu'on lance notre modèle d'apprentissage.

Nous verrons par la suite que la complexité de ces fonctions nous poseront quelques difficultés par la suite, surtout au niveau du temps de traitement.

Pour régler ce problème nous avons mis de côté nos fonctions de transformation des mots à vecteurs binaires et nous avons décidé d'utiliser des fonctions définies sur keras, qui ne prennent plus toute la matrice de vecteurs binaires mais seulement chaque tweet comme un ensemble de symboles de longueur maximale 140.

1)SÉQUENTIEL :

Nous avons choisi de développer un premier modèle s’inspirant du TP effectué en TER (plus précisément le tp2). C’est à dire le modèle séquentiel.

Le modèle séquentiel est le modèle le plus simple que l’on a testé. Nous avons décidé de mettre 4 layout, deux denses et deux d’activations.

Nous avons décidé de lancer notre programme d'apprentissage sur 5 époques avec un batch size de 32 éléments.

Voici donc le résumé:

Étant un des modèles les plus simple, il n’a pas une complexité temporelle très élevée.

En effet on peut voir que les scores se rapprochent de 70% au bout des 5 époques d'apprentissage .

On peut voir que l'apprentissage n'évolue presque pas en fonction des époques. Cela veut dire qu’on se trouve probablement dans le cas de sur apprentissage.Nous avons donc décidé d’utiliser des fonctions définies sur keras.

Voici les résultats obtenus :

Les époques	Le score
1	0.6943
2	0.6923
3	0.6934
4	0.6853
5	0.6898

2) Bayesien Naïf :

- Première partie :

Le modèle Bayesien naïf, contrairement aux autres modèles, est un modèle probabiliste. C'est à dire qui utilise des fonctions mathématiques pour apprendre. Dans ce cas précis, il s'agit des fonctions Bayésiennes.

Comme pour les autres modèles, nous prenons 1 000 000 tweets pour l'apprentissage et 500 000 pour le test.

Pour ce modèle nous cherchons tout d'abord à connaître le nombre de jugement qui sont positifs. Pour cela on crée une fonction qui nous permet de parcourir tous les tweets et de créer une liste qui va contenir ces tweets. Les données ont déjà un attribut positif/négatif, donc il suffit de prendre les tweets qui ont cet attribut à 1.

L'étape suivante est de connaître le nombre d'apparition d'un mot sur les tweets positifs ou négatifs (On ne teste que les mots qui appartiennent à notre dictionnaire, donc on rappelle que c'est les mots qui apparaissent plus de 20 fois et moins de 10 000 fois). Grâce à cette fonction nous aurons déjà un aperçu plus probabiliste du problème.

Ensuite, grâce à l'étape antérieure, on peut calculer les probabilités d'avoir un élément sachant qu'il est sur un tweet positif ou négatif (p1=P(xi|y=1) p0=P(xi|y=0), 1 pour positif et 0 pour négatif). Cependant nous avons toujours le problème des mots absents. Pour cela nous avons utilisé le lissage :

Plus le corpus utilisé pour ces estimations est grand, plus on peut espérer obtenir des estimations de probabilité justes. Cependant, quelle que soit la taille du corpus d'entraînement, il y a toujours des mots ou des séquences de mots absents du corpus (par exemple le mot « no » dans notre corpus, car il apparait plus de 10 000 fois). Pour ces mots ou séquences de mots, leur estimation de probabilité est 0.

Afin de généraliser les modèle, on voudrait assouplir cette attribution systématique de probabilité nulle aux mots ou séquences de mots non rencontrés.

Cette procédure qui consiste à attribuer une probabilité non-nulle á ces éléments est appelée le lissage.

Le lissage peut aussi être vu comme une façon d'éviter le surentraînement d'un modèle sur un corpus, et de doter du modèle d'une plus grande capacité de généralisation

Comme notre but est de comparer les probabilités de P(z=1|xi) et P(z=0|xi) on peut appliquer la formule suivante:

P(z=1|xi)=P(xi|z=1)*P(z=1) , Donc :

Si P(xi|z=1)*P(z=1)>P(xi|z=0)*P(z=0) Alors :

Le score et la perte de modèle bayésien naif:

perte : 0.299954
score : 0.700046

- Deuxième partie :

Nous nous sommes retournés encore une fois sur des modèles déjà créés dans les librairies Python, cette fois SKlearn qui a déjà implémenté Bayesien naïf, et en appliquant les données prétraités nous obtenons le résultat suivant:

score : 0.77409

IV) Conclusion :

En conclusion on peut dire que ce projet nous a permit d’approfondir nos connaissances sur l’apprentissage automatique, non supervisé. Il nous a permit d’avoir aussi une première approche avec des problèmes avec des données très larges, et les difficultés que cela entraîne.

Finalement ce TER nous a aussi poussé à chercher des solutions sur des articles scientifiques et regarder qui dans le domaine de la recherche s'était posé les mêmes questions que nous.

Nous n’avons pas testé l’ensemble des 1 500 000 tweets, nos expériences on été effectués que pour un peu moins de 10% des données totales, donc on peut supposer qu’avec un échantillon de données plus grand on trouve des résultats plus satisfaisant.