

# Projet Tensorflow

Équipe:

Abdenour BARECHE

Guanhua WANG

Xi RONG

Yunbei ZHANG

2ème semestre 2018

# Introduction

Dans le cadre du cours “Méthodes d’apprentissage automatique” et “Outils du traitement de corpus”, nous sommes demandés de réaliser un projet en concentrant sur la classification des textes “Fake News”(nouvelles fausses). La détection du “Fake News” est actuellement un sujet important pour les médias sociaux et la politique. Le pivot de la détection automatique est la classification textuelle en utilisant les méthodes d’apprentissage automatique.

Ce projet s’est réalisé sous forme de Tensorflow.

## Problématique

À l’aide de la structure Tensorflow du deep learning, nous avons fait l’hypothèse que la performance de la classification des textes “trusted”, “fake” et “parodic” sera influencée beaucoup en changeant les paramètres tels que le pas d’apprentissage et l’époch. C’est-à-dire, le résultat se varie d’une grande partie en fonction de la valeur de ces deux paramètres. L’algorithme que nous allons utiliser est le réseau neurone du Deep Learning.

## Outils

- La structure Tensorflow du Deep Learning
- Module Python Etree pour parser le corpus
- Module Python NLTK pour le Pos-tagging du corpus
- Module Python Sklearn pour l’extraction des features et leur transformation en matrice
- Module Python Numpy pour le calcul matriciel

# Méthodes

- **Préparation des corpus:** nous avons travaillé sur le corpus qu'on a construit ensemble en classe. Il est constitué d'articles de presse en format texte brute, collectés principalement des deux sites « Réseaux international » et « 20 minutes ». Pour structurer notre corpus par article, nous avons utilisé le module « Etree » en python pour avoir en sortie un corpus en XML. Puis pour chaque article nous avons ajouté une balise qui contient le même article étiqueté morpho-syntaxiquement qui va nous servir à extraire des lemmes. Enfin nous avons divisé notre corpus en deux parties. La première partie servira à entraîner notre modèle, la deuxième nous l'utiliserons pour le tester.
- **Extraction des données du corpus:** en rappelant la fonction `corpus_extraction` du fichier python "mat\_tfidf.py", une matrice contenant les valeurs qui représentent la nature de chaque document est construite. Cette matrice a une taille de 3 colonnes (fake, trusted, parodic) et le nombre de rangs égale au nombre de documents. En référant à l'attribut "class", les valeurs (soit 1 pour indiquer la présence, soit 0 pour l'absence) sont mises dans les cases du document correspondant. Finalement il extrait pour sortie les textes du corpus (sous une liste), le POS de chaque texte, et la matrice générée.
- **Prétraitement du corpus:** le prétraitement du corpus consiste à raciniser les mots dans le corpus et encore, enlever les "stop words"(mots vides), par le biais des outils (PorterStemmer, etc) du module NLTK. Ainsi, il ne faut pas oublier de transformer tous les majuscules en minuscules.
- **Vectorisation du corpus:** à l'aide du module sklearn, nous avons converti les corpus de train et de test en vecteurs (constituant une matrice enfin). Chaque matrice contient des features de tokens en valeur de tf-idf. Une autre

transformation est mise en place après, la procédure est pareille sauf que cette fois-ci est pour le POS au lieu des tokens.

- **Entraînement des modèles:** ensuite nous avons commencé à entraîner le modèle en changeant au fur et à mesure le taux d'apprentissage (learning rate), "epoch" et features (soit des tokens, soit leurs pos tags). Par modifier ces paramètres de l'algorithme, le taux d'exactitude se varie dépendamment.

## Évaluation qualitative

Nous avons fait deux matrices de features, dont "corpus" s'est constituée par la valeur TF-IDF de chaque token du corpus, et que "tags" par la valeur TF-IDF des tags de tokens dans le corpus.

Selon ces deux configurations, nous avons amené des expériences diverses.

En mettant le "learning rate" à 0,001 avec l'epoch qui était de 10 000, nous avons retenu notre baseline qui est à 36,73%. Cette baseline était obtenu en s'appuyant sur la configuration avec la matrice "corpus".

Avec la matrice "corpus", les résultats ne sont pas assez satisfaisants pour les expériences que nous avons réalisés, de 34,69% à 53,06% qui se varie de temps en temps selon la modification du "learning rate" et le changement de l'epoch. Ces taux d'exactitude assez bas nous ont inspiré de combiner les deux configurations pour des expériences à plus loin.

Nous avons donc concaténé les deux matrices pour l'expérience. Par contre, le taux d'exactitude de cette mesure a ramené à 36,73%, qui est juste un peu plus haut que notre baseline. Ce qui n'est pas encore performante.

Finalement, nous avons constaté que la manière en utilisant la matrice "tags" a atteint le taux le plus haut, avec le résultat le meilleur parmi ces trois moyens, de 40,80% à 53,06% même si le meilleur résultat est le même que celui du moyen en utilisant seulement la matrice "corpus", la portée des résultats s'est réduite par rapport aux deux autres.

L'image au-dessous présente le meilleur résultat de la matrice "tags" (le taux d'exactitude du "test" = 53,06%, celui du "train" alors = 98,99%) où le "learning rate" était changé à 0.008 et "epoch" à 10 000:

```
step 9930, change in cost 0.00209332
step 9940, training accuracy 0.979798
step 9940, cost 2.2711
step 9940, change in cost 0.00209284
step 9950, training accuracy 0.979798
step 9950, cost 2.269
step 9950, change in cost 0.00209236
step 9960, training accuracy 0.979798
step 9960, cost 2.26691
step 9960, change in cost 0.00209212
step 9970, training accuracy 0.979798
step 9970, cost 2.26482
step 9970, change in cost 0.00209188
step 9980, training accuracy 0.979798
step 9980, cost 2.26273
step 9980, change in cost 0.00209069
step 9990, training accuracy 0.979798
step 9990, cost 2.26064
step 9990, change in cost 0.00209022
final accuracy on test set: 0.97979796
```

## Résultats

En nous appuyant sur la mesure de la matrice "tags", nous avons extrait les résultats de la prédiction pour chaque texte et l'avons gardé dans un attribut dans le corpus XML. En comparant ces valeurs avec les classes correctes des textes du corpus:



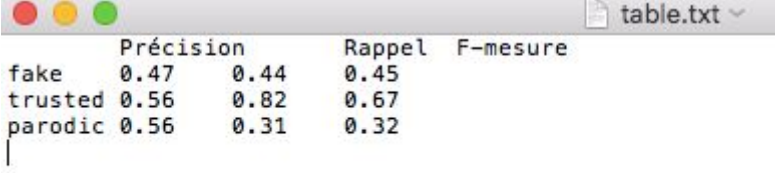
```
output_meilleur.xml x
<corpus><doc class="trusted" classpredict="trusted" source="lep
dur pour les chômeurs" url="http://www.leparisien.fr/economie/
un-bareme-de-sanctions-plus-dur-pour-les-chomeurs-19-03-2018-76
Le ministère du Travail a dévoilé ce lundi une nouvelle gri
envers les demandeurs d'emploi qui ne cherchent pas activem
Nouveau barème de sanctions pouvant aller jusqu'à quatre mo
définition de l'offre « raisonnable » d'emploi et expérimen
```

Nous avons ensuite calculé les précision, rappel et F-mesure en utilisant le petit script evaluate.py et finalement obtenu le résultat final:

La classification des textes “fake” est d’une précision de 0,47, d’un rappel 0,44 et sa F-mesure est de 0,45.

Pour les textes “trusted”, la précision est de 0,56, son rappel est de 0,82 avec une F-mesure à 0,67.

La précision est de la même valeur pour les textes “parodic” (0,56), par contre le rappel a baissé jusqu’à 0,31 et malheureusement la F-mesure n’est pas haute non plus: 0,32.



	Précision		Rappel	F-mesure
fake	0.47	0.44	0.45	
trusted	0.56	0.82	0.67	
parodic	0.56	0.31	0.32	

## Discussions & Perspectives

1. Le taux d'apprentissage (learning rate) peut avoir un grand impact sur le taux d'exactitude de l'expérience. Cette valeur ne doit pas être trop grande qui va sans doute dépasser la valeur minimale de la fonction de la perte (cost function). Ce qui empêche la convergence. Alors qu'elle ne doit pas être trop petite non plus. Ce qui va peut-être trop ralentir le rythme de la convergence dans la descente gradient.
2. Il faut éviter le surapprentissage dans l'entraînement du modèle. Le surapprentissage peut paraître lorsque les paramètres sont trop nombreux ou complexes. Il se présente par un taux d'exactitude parfait lors de l'évaluation sur un corpus spécifique, alors que pour les corpus aléatoires, ce taux baisse sévèrement. Pour éviter cette situation, il se peut que la validation croisée se met en place. Ce qui n'est pas pris en compte dans nos expériences.
3. Pour améliorer les performance du modèle il faudra normaliser au maximum notre corpus. Les modules de la bibliothèque NLTK marche très bien sur l'anglais mais malheureusement elle n'est pas assez performante sur le

français et c'est ce que nous avons constaté lors de son utilisation sur notre corpus. Pour cela nous avons dû implémenter nos propres fonctions pour normaliser notre corpus notamment la lemmatisation. Le but est d'exploiter chaque mot en supprimant les différentes dérivations comme la conjugaison et le pluriel etc. cela nous permet d'avoir un meilleur TF-IDF.

4. La taille du corpus joue un rôle très important dans l'apprentissage malheureusement le nôtre n'est pas assez grand pour avoir un score très élevé. c'est pour cela nous pensons qu'en augmentant le corpus d'apprentissage nous pourrions avoir un score plus élevé.

## Usages des programmes du projet

1. Faire fonctionner le programme pour entraîner les modèles:

- `python3 train.py all_train.xml Model all_test.xml`

Il est à noter que *Model* est en effet un nom du dossier où stocker les modèles, au lieu d'un seul fichier modèle.

2. Faire fonctionner le programme pour la prédiction des classes:

- `python3 predict.py all_train.xml Model all_test.xml output.xml`

```
[yunbeidemacbook-air:TF1 yunbeizhang$ python3 train.py /Users/yunbeizhang/Desktop/TIM-M2/Corpus/all-train.xml Model /Users/yunbeizhang/Desktop/TIM-M2/Corpus/all-test.xml  
<class 'str'>  
Reading corpus and finding features  
Reading corpus and finding features
```

output.xml est le fichier du résultat qui ajoute la classe classpredict à côté de la classe de gold standard pour chaque document donné.

3. Faire fonctionner le programme pour l'évaluation du résultat:

- `python3 evaluate.py output.xml`

```
[yunbeidemacbook-air:TF1 yunbeizhang$ python3 predict.py /Users/yunbeizhang/Desktop/TIM-M2/Corpus/all-train.xml Model /Users/yunbeizhang/Desktop/TIM-M2/Corpus/all-test.xml output.xml  
Reading corpus and finding features  
Reading corpus and finding features  
Reading corpus and finding features
```

Le fichier en sortie *table.txt* est le fichier contenant le tableau de la précision, le rappel et la F-mesure de l'évaluation, en comparant du résultat de la prédiction.