# *Analyse des sentiments à l'aide de RNN et de word embedding*

*Ce notebook est écrit en Python3, le but était d'analyser un jeu de données de 16.000.000 tweets en utilisant RNN et word embedding pour faire une analyse des sentiments*

*Le notebook est organisé comme ceci : les titres sont en caractères gras et des raccourcis sont disponibles dans le sommaire, les explications sur ce que nous faisons sont dans les commentaires (Markdown) et les notes de quelques choses que nous avons essayées et travaillées mais qui ne fonctionnaient pas réellement dans le rendu final sont en texte brut (pour NBconvert) et le code à l'intérieur ne sera pas exécuté.*

## Contexte :

Dans le cadre de notre cours de NLP (Natural Language Processing), on nous a attribué une base de données comprenant un total de 16 millions de tweets, organisés et labélisés positifs et négatifs dans un classeur en .csv, disponible sur https://www.kaggle.com/kazanova/sentiment140.

Le but était donc de créer et entrainer un modèle à partir de ces tweets, qui pourrait prédire si le contenu d'un tweet est positif ou négatif.

Cette approche est aussi appelée TF-IDF, et est désormais un incontournable dans la façon dont les entreprises recherchent à avoir des retours clients par exemple.

Pour ce devoir, nous allons devoir utiliser un réseau de neurones récurrents (RNN) ainsi que du word embedding pour pouvoir entrainer notre modèle.

# Approche utilisée en détail :

Pour pouvoir résoudre ce problème, nous avons pu utiliser diverses bibliothèques pour nous simplifier la tâche :

```
import pandas as pd
import numpy as np
import string
import re
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.preprocessing import sequence
import keras
from keras.layers import LSTM, Activation, Dropout, Dense, Input, Embedding
from keras.models import Model
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
import nltk
from nltk.tokenize import RegexpTokenizer
import matplotlib.pyplot as plt
from mlxtend.plotting import plot_confusion_matrix

nltk.download('punkt')
```

- Pandas est un outil de manipulation et analyse de données qui va beaucoup servir dans le code
- *Numpy* va nous permettre de faire différentes opérations et manipulation informatiques
- *String* va nous aider à avoir toutes les ponctuations possibles pour le preprocessing
- *Re* va nous permettre d'effectuer des opérations à base d'expressions rationnelles (par exemple utiliser .sub pour le preprocessing)
- *Tensorflow* est une plateforme Open Source qui est dédiée au machine learning
- *Keras* va nous permettre de créer notre modèle de deep learning
- *Sklearn* (Scikit-learn) est une bibliothèque libre Python destinée à l'apprentissage automatique
- *Nltk* (Natural Language Toolkit) est une bibliothèque logicielle en Python permettant un traitement automatique des langues
- *Matplotlib* est destinée à tracer et visualiser des données sous formes de graphiques, ce qui nous permettra d'afficher nos résultats d'évaluation du modèle
- *Mlxtend* (machine learning extensions) est composée d'outils pour certaines tâches, ici elle nous permettra d'afficher facilement une matrice de confusion

## Importer les données

Notre approche a donc été de d'abord importer les données, réduire leur taille pour pouvoir avoir un temps de traitement raisonnable (travaillant en local avec Jupyter Notebook sur un ordinateur sans GPU très puissant, nous avons préféré prendre seulement 100 tweets positifs et 100 négatifs avec 5 epochs pour construire tout le code, puis ensuite pour le travail final nous avons entrainé le modèle avec un total de 20000 tweets et 15 epochs).

Nous avons donc pu commencer par importer ces données et les labéliser pour mieux prendre en main le classeur pour au final ne garder que les colonnes comportant le texte des tweets et le sentiment (positif ou négatif).

Ensuite nous avons séparé les tweets positifs et négatifs, puis pris seulement un petit nombre de tweets également répartis.

Pour pouvoir entraîner le modèle et que les résultats soient interprétables pour la précision (accuracy), nous avons dû assigner 1 au lieu de 4 pour le label des tweets positifs.

La dernière étape de l'importation a été de replacer (en concaténant les 2 tableaux positifs et négatifs) les tweets en écrasant le tableau data initial.

| | text | sentiment |
|---|---|---|
| **800000** | I LOVE @Health4UandPets u guys r the best!! | 1 |
| **800001** | im meeting up with one of my besties tonight! ... | 1 |
| **800002** | @DaRealSunisaKim Thanks for the Twitter add, S... | 1 |
| **800003** | Being sick can be really cheap when it hurts t... | 1 |
| **800004** | @LovesBrooklyn2 he has that effect on everyone | 1 |
| **...** | ... | ... |
| **9995** | Aww that's sad | 0 |
| **9996** | stupid dvds stuffing up the good bits in jaws. | 0 |
| **9997** | @Dandy_Sephy No. Only close friends and family... | 0 |
| **9998** | CRAP! After looking when I last tweeted... WHY... | 0 |
| **9999** | Its Another Rainboot day | 0 |

## Prétraitement

Le prétraitement va représenter une partie conséquente du travail, en effet, il est primordial d'avoir des données bien traitées pour pouvoir les analyser après et que le réseau de neurones soit efficace.

La première étape sera donc de mettre en minuscule tous les tweets à analyser. Ensuite, on enlèvera les stop-words, mots qui sont le plus présent dans une langue et qui ne vont pas donner le sens de celle-ci. Pour ça nous allons utiliser la bibliothèque NLTK qui va nous fournir une liste de stop-words que nous n'aurons qu'à soustraire des tweets.

On peut également enlever les caractères qui se répètent pour avoir un corpus plus propre, et ensuite enlever les @ (arobas, at en anglais), et donc par la même occasion les emails et mentions dans les tweets (il suffit d'enlever les caractères qui sont collés à @.

Une bonne chose sera également de supprimer les URLs si l'on n'en a pas besoin, ainsi que les nombres, toujours en utilisant la commande .sub issue de la librairie re.

Enfin, on enlèvera la ponctuation, cette fois-ci grâce à la bibliothèque string.

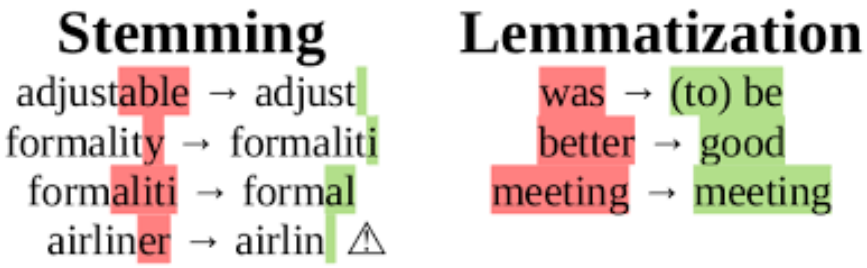
## Tokenizer

Une fois notre texte bien propre, on peut tokeniser sont contenu. Globalement la tokenisation va être le processus de mettre dans un tableau tous les mots d'une même phrase, ce qui va nous permettre par la suite de mettre différents poids sur chaque mot et pouvoir les analyser comme on le souhaite. Pour cela, nous allons utiliser la Keras, qui va nous aider à le faire rapidement et sans encombre.

Par exemple, la phrase :
« Je ne suis pas content »
donnera le tableau
['Je','ne','suis','pas','content']

Lorsque la tokenisation est terminée, on va pouvoir appliquer le stemming, ce qui va nous donner une forme plus générale d'un mot, pour le stemming on va utiliser encore une fois NLTK qui fournit un stemmer très correct.

Enfin, on pourra appliquer un lemmatizer qui va lui plutôt s'attaquer aux formes initiales des mots, on utilisera d'ailleurs également NLTK pour ce processus.




Source : https://devopedia.org

## Répartition des données de train et de test

On va maintenant mélanger le dataset et le séparer en plusieurs parties : une première (x), qui va représenter le texte et une seconde (y) qui va représenter le sentiment.

On va convertir les mots du texte sous forme de tableaux et donner au maximum 500 caractéristiques par mots sélectionnés pour la formation. Ces 500 mots seront sélectionnés en fonction de l'importance qui distinguera les tweets positifs et les tweets négatifs.

Ici, nous utilisons train_test_split pour diviser nos données pour la formation et les tests, x étant le texte et y le sentiment.

Nous définissons maintenant notre modèle, nous utiliserons Keras, qui sera composé d'une couche d'entrée, d'une couche d'embedding, puis de LSTM, d'une couche dense, d'une activation utilisant ReLU, puis du dropout et d'une autre couche dense avant de finir avec une dernière couche d'activation à l'aide de sigmoïde cette fois.

```
def tensorflow_model():
    inputs = Input(name='inputs',shape=[MAX_SEQUENCE_LENGTH])
    layer = Embedding(2000,50,input_length=MAX_SEQUENCE_LENGTH)(inputs)
    layer = LSTM(64)(layer)
    layer = Dense(256,name='FC1')(layer)
    layer = Activation('relu')(layer)
    layer = Dropout(0.5)(layer)
    layer = Dense(1,name='out_layer')(layer)
    layer = Activation('sigmoid')(layer)
    model = Model(inputs=inputs,outputs=layer)
    return model
```

## Compilation du modèle et évaluation

Nous voulons appeler notre modèle, nous utiliserons donc 2 classes, si nous définissons "binary_crossentropy" et utilisons plus de deux classes alors nous utiliserons "categorical_crossentropy".

On peut également modifier les fonctionnalités du réseau neuronal telles que le taux d'apprentissage avec la fonction d'optimisation afin de réduire les pertes (loss).

À ce moment-là, on pourra enfin entraîner notre modèle et le valider ou non.

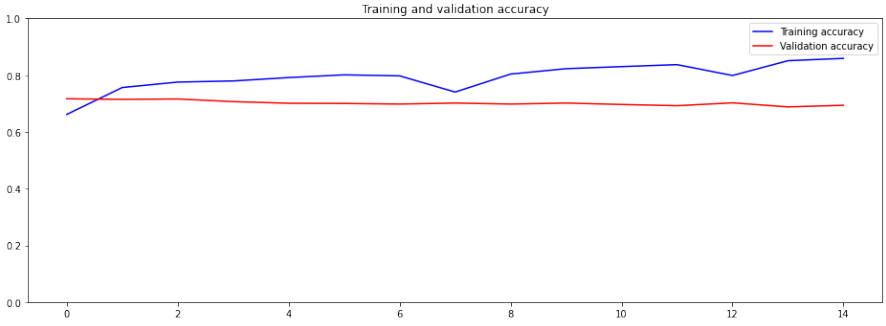
```
Entrée [42]: history = model.fit(x_train,y_train,batch_size=80,epochs=15, validation_split=0.1)
accuracy: 0.6986
Epoch 10/15
158/158 [==============================] - 49s 309ms/step - loss: 0.3774 - accuracy: 0.8272 - val_loss: 0.7019 - val_
accuracy: 0.7021
Epoch 11/15
158/158 [==============================] - 49s 307ms/step - loss: 0.3594 - accuracy: 0.8365 - val_loss: 0.6704 - val_
accuracy: 0.6971
Epoch 12/15
158/158 [==============================] - 49s 310ms/step - loss: 0.3465 - accuracy: 0.8422 - val_loss: 0.6935 - val_
accuracy: 0.6929
Epoch 13/15
158/158 [==============================] - 50s 319ms/step - loss: 0.4912 - accuracy: 0.8181 - val_loss: 0.7671 - val_
accuracy: 0.7029
Epoch 14/15
158/158 [==============================] - 58s 365ms/step - loss: 0.3177 - accuracy: 0.8564 - val_loss: 0.7200 - val_
accuracy: 0.6886
Epoch 15/15
158/158 [==============================] - 274s 2s/step - loss: 0.3056 - accuracy: 0.8675 - val_loss: 0.8225 - val_ac
curacy: 0.6943
```

On pourra évidement avoir notre précision, ici selon l'entrainement environ à 70%.

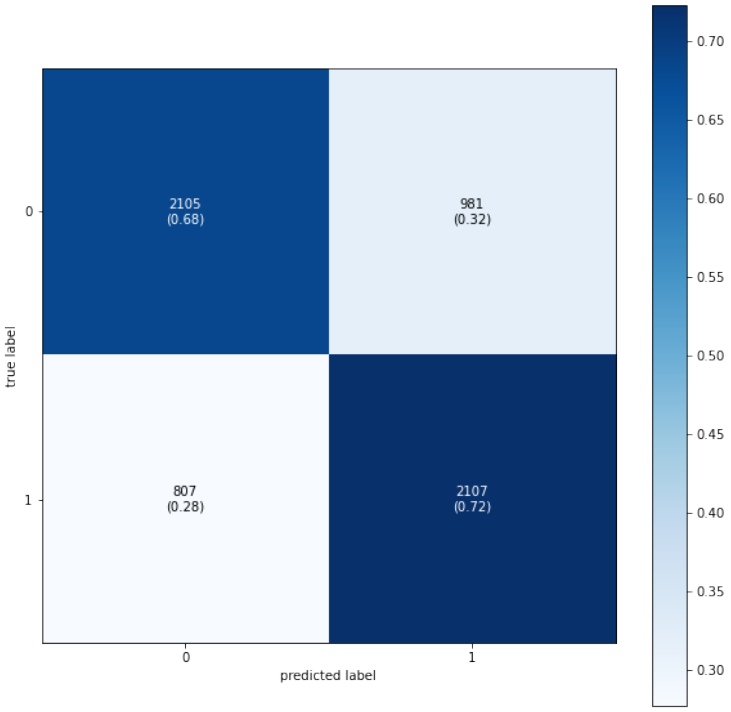
```
Entrée [43]: accuracy = model.evaluate(x_test,y_test) #we are starting to test the model here

             print('Accuracy: ', accuracy[1])
188/188 [==============================] - 10s 54ms/step - loss: 0.8117 - accuracy: 0.7020
Accuracy:  0.7020000219345093
```

Pour avoir des résultats plus clairs et graphiques il y a de nombreuses façons pour afficher les performances de notre modèle. Ici nous avons choisi seulement 2 façons de montrer nos résultats.

La première va être, avec l'utilisation de la librairie Python Matplotlib, d'afficher la précision et les pertes par rapport au nombre d'epochs.




On voit selon nos graphiques que notre répartition est plutôt fidèle.




Une autre façon d'évaluer notre modèle est de construire une matrice de confusion, donc nous devons d'abord obtenir la prédiction, puis la tracer dans une matrice de confusion (pour avoir des résultats rapides, nous utiliserons directement la bibliothèque sklearn.metrics ici).

## Améliorations possibles du projet :

Le projet présenté ici retourne des résultats assez satisfaisants, la précision étant assez bonne, mais pourrait être largement améliorée.

Pour commencer, rien que le simple fait de rajouter des itérations d'apprentissage et utiliser tout le dataset au lieu de simplement une petite partie serait déjà un gros avancement. Les tests que nous avons pu réaliser ont montré que l'on pouvait déjà avoir de petites améliorations simplement en changeant donc ces paramètres.

Cependant par la suite ce qui pourrait vraiment améliorer le code serait de repenser le code, s'attarder plus en profondeur encore sur le preprocessing mais également surtout sur le modèle utilisé. En effet les couches utilisées ici sont plutôt standard, et il est possible de trouver des modèles plus efficaces pour ce genre de travail.

Enfin, dans les consignes du devoir était demandé d'utiliser l'API Yelp afin de tester notre pipeline, et bien qu'ayant essayé de tester notre modèle avec l'API, nous ne sommes pas parvenus à tester convenablement notre solution. Un futur travail sera donc également de plonger plus en profondeur dans l'API, mieux apprendre la façon dont elle fonctionne et surtout comment en tirer profit pour tester notre pipeline.