# ARTICLE DE BLOG

## DETECTER LES BAD BUZZ GRACE AU DEEP LEARNING

MODESTE Khadija | 27/12/2022

# INTRODUCTION

## CONTEXTE

L'intelligence artificielle est un outil incontournable très puissant pour résoudre de divers problématiques dans de différents domaines en particulier le marketing. C'est pour cette raison que la direction Marketing de la compagnie aérienne **Air Paradis** a missionné notre entreprise MIC (Marketing Intelligence Consulting) pour créer un produit IA permettant d'anticiper les Bad Buzz postés sur Twitter.

Notre mission consiste à créer un prototype d'un produit IA permettant de **prédire le sentiment associé à un tweet**. Il s'agit d'un problème de **NLP** (**Natural Language Processing** ou **Traitement Automatique du Langage Naturel**). On propose donc :

- Une approche « Modèle sur mesure simple » basé sur le Machine Learning
- Une approche « Modèle sur mesure avancé » basé sur les réseaux de neurones
- Un modèle BERT
- Le déploiement sur le Cloud

## METHODOLOGIE

Pour répondre à ce sujet on va procéder comme suit :

- Exploration des données et pré-traitement de texte
  - Exploration des données
  - Séparation des données
  - Traitement de texte
- Elaboration d'un modèle sur mesure simple
  - Vectorisation
  - Entrainement de la régression logistique
  - Evaluation du modèle
- Elaboration d'un modèle sur mesure avancé
  - Modèle Keras avec une couche embedding
  - Modèle Keras avec une couche embedding et une couche LSTM bidirectionnel
- Elaboration d'un modèle BERT
- Comparaison des résultats des 3 approches
- Déploiement sur Azure

# EXPLORATION DES DONNEES ET PRE-TRAITEMENT DE TEXTE

## EXPLORATION DES DONNEES

On va utiliser les données sur site de Kaggle. Il s'agit de 1 600 000 tweets extraits à l'aide de l'API Twitter, les tweets ont été annotés (0 = négatif, 4 = positif). Après avoir importé les données, la première chose à faire quand on est face à un problème de classification est de de vérifier si les classes sont équilibrées.
En effet, si les classes sont déséquilibrées, les algorithmes auront tendance à prédire la classe majoritaire.

## SEPARATION DES DONNEES

Pour réduire le temps de calcul, on va utiliser qu'une partie des données (1 000 000 de tweets)

Avant de procéder au nettoyage des tweets et pour éviter la fuite de l'information, on va séparer les données en données de traitement qui présente des données qu'on va utiliser pour entraîner les modèles, données d'évaluation pour évaluer les modèles et données de test qu'on va les utiliser pour comparer les modèles finaux.

## PRE-TRAITEMENT DE TEXTE

Le pré-traitement des données est une phase très importante dans un projet de classification. Cette étape est primordiale pour améliorer la performance des résultats d'un modèle. Dans notre cas, on va procéder au nettoyage de texte avec la librairie **Spacy** en suivant les étapes suivantes :

- Tokeniser des tweets
- Supprimer les tokens de longueur 1
- Supprimer les stopwords
- Supprimer les espaces vides
- Supprimer les nombres
- Supprimer les adresses mails
- Supprimer les URLs
- Supprimer les noms d'utilisateurs

Après premier nettoyage, on va tester deux types de nettoyage :

- La lemmatisation qui consiste à garder la forme canonique du mot
  Exemple *: Caring ----> Care*
- La racinisation qui vise à garder la racine du mot
  Exemple *: Caring ----> Car*

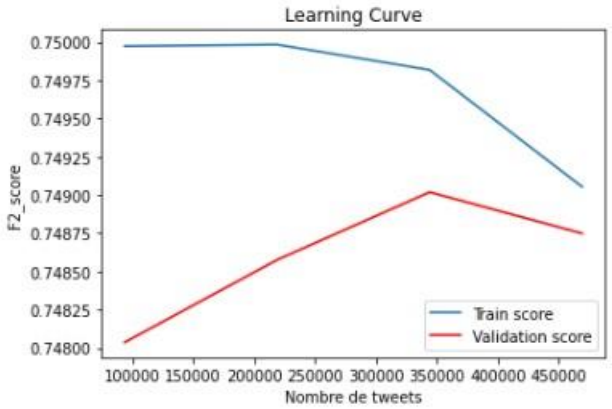
# ELABORATION D'UN MODELE SUR MESURE SIMPLE

## VECTORISATION

On dispose maintenant d'un corpus de tweets prétraités mais les modèles de Machine Learning, en général, on besoin des entrées numériques, donc on doit vectoriser nos tokens avant d'entrainer nos modèles.
Il y a plusieurs méthodes de vectorisation, mais on va choisir **Word2Vec** qui est une méthode de **plongement de mot** ou **Word Embedding** pour entrainer notre premier modèle.

## ENTRAINEMENT DU MODELE

Le premier modèle qu'on va utiliser est la régression logistique qu'on va entrainer sur les tweets lemmatisés et encodés selon Word2Vec. C'est un modèle léger et rapide.
Le premier problème souvent rencontré dans un projet de Machine Learning est le surapprentissage. Pour vérifier qu'on notre modèle est bon, on doit réaliser la cross-validation ou tracer le Learning Curve comme ci-dessous




En effet, le Learning Curve répond bien aux critères d'un bon modèle.

## EVALUATION

La régression logistique a donné une précision de 74%, qui est un résultat qui n'est pas mal par rapport à la rapidité du modèle.

# ELABORATION D'UN MODELE SUR MESURE AVANCE

Contrairement aux algorithmes classiques du machine learning, dont la capacité d'apprentissage est limitée quelle que soit la quantité de données acquise, les réseaux de neurones peuvent améliorer leur performance en accédant en davantage de données. On dispose dans notre cas d'une quantité de données relativement satisfaisante pour entrainer des modèles avancés. On va tester deux types d'architectures de réseaux de neurones :

- Réseaux de neurones avec une couche embedding
- Réseaux de neurones avec une couche embedding et une couche LSTM

On va tester ces deux modèles sur des tweets :

- Lemmatisés
- Stemmés

Et on va tester aussi deux types d'embedding

- Vectorisation selon l'embedding Word2vec
- Vectorisation selon l'embedding Glove

## MODELE KERAS AVEC UNE COUCHE EMBEDDING

On a travaillé sur un modèle de réseaux de neurones avec une couche embedding et une couche dense, pour éviter le surapprentissage, on a rajouté une couche de Pooling et on a supprimé quelques connexions dans les couches Dense. Ce modèle dont l'architecture ci-dessous est appliqué sur des tweets lemmatisés encodés selon Word2Vec.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_3 (Embedding)     (None, 205, 200)          30879600

 global_average_pooling1d_1   (None, 200)              0
 (GlobalAveragePooling1D)

 flatten_3 (Flatten)         (None, 200)               0

 dense_4 (Dense)             (None, 40)                8040

 dropout_1 (Dropout)         (None, 40)                0

 dense_5 (Dense)             (None, 20)                820

 dropout_2 (Dropout)         (None, 20)                0

 dense_6 (Dense)             (None, 1)                 21

=================================================================
Total params: 30,888,481
Trainable params: 8,881
Non-trainable params: 30,879,600
_________________________________________________________________
```

Le modèle donné une précision de 75% avec un temps d'entrainement de 5.83 minutes

## MODELE KERAS AVEC UNE COUCHE EMBEDDING ET UNE COUCHE LSTM BIDIRECTIONNELS

Certes, le modèle précédent a amélioré le résultat par rapport à la régression logistique mais on peut augmenter la précision si on adopte les réseaux **LSTM (**Long short-term memory**)** bidirectionnels, qui est un modèle capable de traiter les longues séquences grâce à sa mémoire à court terme, on peut supprimer quelques connexions pour éviter le surapprentissage. On applique ce modèle sur des données stemmées et encodées selon l'embedding Glove.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 embedding_1 (Embedding)      (None, 205, 200)          30879600

 bidirectional_1 (Bidirectio  (None, 20)                16880
 nal)

 dropout (Dropout)            (None, 20)                0

 flatten_1 (Flatten)          (None, 20)                0

 dense_1 (Dense)              (None, 10)                210

 dropout_1 (Dropout)          (None, 10)                0

 dense_2 (Dense)              (None, 1)                 11

=================================================================
Total params: 30,896,701
Trainable params: 17,101
Non-trainable params: 30,879,600
_________________________________________________________________
```

On obtient une précision de 0.77 plus élevés que le modèle précédent et un temps de traitement de 104.42 minutes

## ELABORATION D'UN MODELE BERT

Le dernier modèle qu'on va tester est un modèle de Deep Learning pré-entraîné appelé le modèle **BERT** qui a été publié en 2018 par la filière de l'intelligence artificielle de **Google (Google IA)**. C'est un modèle très avancé qui permet une meilleure compréhension de texte

Ce modèle utilise une nouvelle technique appelée **Masked LM (**MLM) : **masquage aléatoire des mots**, il masque quelques mots puis il essaie de les prédire.

Il y a plusieurs modèles BERT, on va télécharger le modèle **bert-base-uncased** et son Tokeniser puis on va entrainer sa dernière couche sur les tweets lemmatisés :

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)       multiple                  109482240

 dropout_37 (Dropout)         multiple                  0

 classifier (Dense)           multiple                  1538

=================================================================
Total params: 109,483,778
Trainable params: 1,538
Non-trainable params: 109,482,240
_________________________________________________________________
None
```

BERT donne une précision de 0.80 avec un temps d'entrainement de 108.16 minutes.

# COMPARAISON DES RESULTATS

Voici le tableau qui résume les résultats de tous les modèles élaborés :

| | Model_name | Accuracy_score | Learning_time in minutes |
|---|---|---|---|
| 0 | Logistic Regression | 0.744 | 0.340 |
| 1 | Embedding_lemm_w2v | 0.755 | 5.835 |
| 2 | Embedding_lemm_glove | 0.756 | 7.285 |
| 3 | Embedding_stem_w2v | 0.753 | 7.108 |
| 4 | Embedding_stem_glove | 0.755 | 6.976 |
| 5 | Bidirect_lstm_lemm_w2v | 0.771 | 102.364 |
| 6 | Bidirect_lstm_lemm_glove | 0.775 | 102.034 |
| 7 | Bidirect_lstm_stem_w2v | 0.775 | 104.534 |
| 8 | Bidirect_lstm_stem_glove | 0.773 | 104.423 |
| 9 | Bert_base_uncased | 0.797 | 108.161 |

La régression logistique est rapide mais il donne que 0.74 de précision, contrairement aux modèle BERT qui donne un résultat très intéressant mais avec un temps d'entrainement qui est très élevé.
Avec les moyens qu'on dispose, on a choisi de déployer le modèle simple grâce à sa légèreté. En effet, la régression logistique donne un bon compromis précision/complexité

# DEPLOIEMENT SUR AZURE

Après choix et sauvegarde du modèle, on a suivi les étapes suivantes pour le déploiement :

- Déploiement en local avec le Framework Flask
- Envoie des fichiers sur Github
- Déploiement sur Azure

Le prototype reçoie un tweet et envoie le sentiment positif ou négatif d'un tweet


Le prototype est disponible sur le lien suivant : https://tweet-sentiment-api.azurewebsites.net/

# C O N C L U S I O N

Comme on a vu précédemment, les modèles avec une couche LSTM donne une meilleure précision mais ils sont très complexes et demandent des machines très puissante en calcul, le modèle BERT est encore plus complexe que les réseaux LSTM et son facteur limitant est que ce modèle utilise le GPU.

On peut améliorer davantage le résultat de LSTM par l'optimisation de ses hyperparamètres.

Il y a d'autres modèles pré-entraînés à tester comme BERTweet qui pourraient donner encore des résultats satisfaisants.