Détecter les bad buzz avec l'intelligence artificielle

Le contexte

Cela fait plusieurs années que l'on entend parler de « bad buzz » ou de crises d'images. Avec l'essor de plus en plus faramineux des réseaux sociaux, il est devenu très simple de ruiner l'image d'une marque ou d'une entreprise. Il suffit parfois d'un rien pour qu'un internaute s'offusque, et qu'il embarque avec lui des millions d'autres personnes. Si les Hommes sont capables de faire preuve d'esprit critique dans la vie réelle, ils le sont souvent moins dans la vie virtuelle. Ces dernières années, nous avons pu observer des boycotts d'entreprise dantesques, suis à un bad buzz.

Je suis ingénieur IA chez Marketing intelligence Consulting, une entreprise de conseil spécialisée sur les problématiques de marketing digital. Notre cabinet a été missionné par Air Paradis pour créer un produit IA permettant d'anticiper les bad buzz sur les réseaux sociaux, et plus particulièrement Twitter.

J'ai donc testé plusieurs approches pour créer un prototype de produit d'Intelligence Artificielle capable de déterminer la polarité du sentiment associé à un tweet : Il s'agit de savoir si un tweet exprime un sentiment positif ou négatif.

Les données

Les données sont au cœur de la technologie de l'intelligence artificielle. Pour ce projet, j'ai utilisé un jeu de données de 1 600 000 tweets labellisés avec le sentiment lié ("positif" représenté par le chiffre 1, ou "négatif" représenté par le chiffre 0). Notons que ce jeu de données est équilibré, c'est-à-dire qu'il y a autant de tweets labellisés 0 que de tweets labellisés 1, ce qui simplifie les choses pour l'apprentissage, car il n'y a pas de pondération à faire.

J'ai réalisé plusieurs échantillonages de ces données, en respectant l'équilibre des classes. J'ai nommé ces échantillons :

- Comparison (cmp) : pour comparer les trois approches
- Validation (val) : pour sélectionner le meilleur modèle avancé
- Train : pour l'entraînement des modèles
- Train_sample : extrait du jeu « train » pour entraîner plus rapidement les modèles
- Test : pour l'évaluation des modèles

L'évaluation des modèles

Pour comparer les différentes approches, j'ai mis de côté un échantillon de 1600 tweets. Les modèles n'auront jamais vu ces données, et j'ai comparé les prédictions des différents modèles sur ces données avec leurs vrais labels.

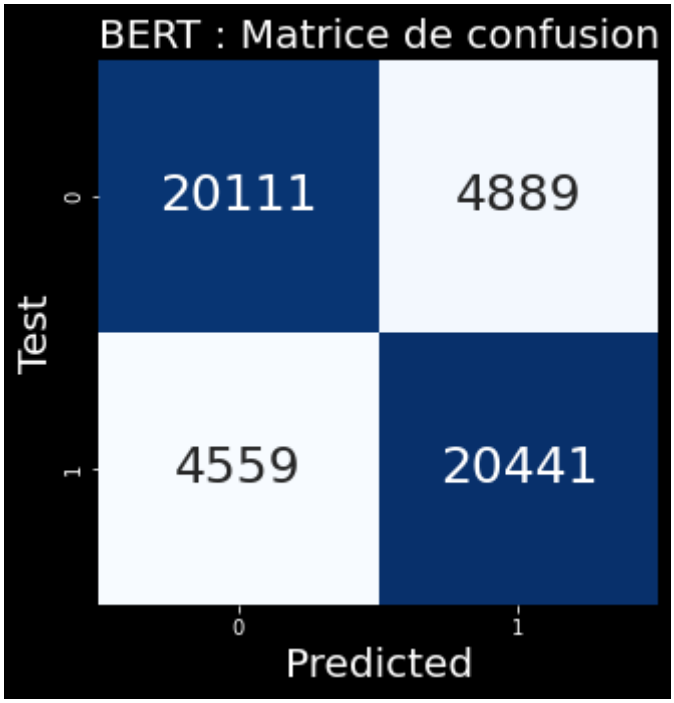
La métrique

Comme les données sont équilibrées, et que l'on souhaite des performances égales pour les prédictions des deux classes (positif et négatif), mon choix s'est porté sur l'**Accuracy** car cette métrique permet de décrire la performance d'un modèle sur les individus positifs et négatifs de façon symétrique.

Pour rappel, l'**Accuracy** est basée sur la **matrice de confusion**, qui est composée de 4 valeurs :

- Le nombre de **vrais négatifs** : le nombre de fois où le modèle a prédit 0 et a eu raison.
- Le nombre de **faux négatifs** : le nombre de fois où le modèle a prédit 0 et a eu tort.
- Le nombre de **vrais positifs** : le nombre de fois où le modèle a prédit 1 et a eu raison.
- Le nombre de **faux positifs** : le nombre de fois où le modèle a prédit 1 et a eu tort.

Voici un exemple de matrice de confusion :



L'**Accuracy** a pour formule :

$$\frac{\text{vrais positifs} + \text{vrais négatifs}}{\text{vrais positifs} + \text{vrais négatifs} + \text{faux positifs} + \text{faux négatifs}}$$

Ce qui peut donc se résumer ainsi :

$$\frac{\text{prédictions correctes}}{\text{ensemble des prédictions}}$$

L'accuracy mesure donc quel est le pourcentage de prédictions correctes sur l'ensemble des prédictions d'un modèle.

Première approche : API Azure Cognitive Service

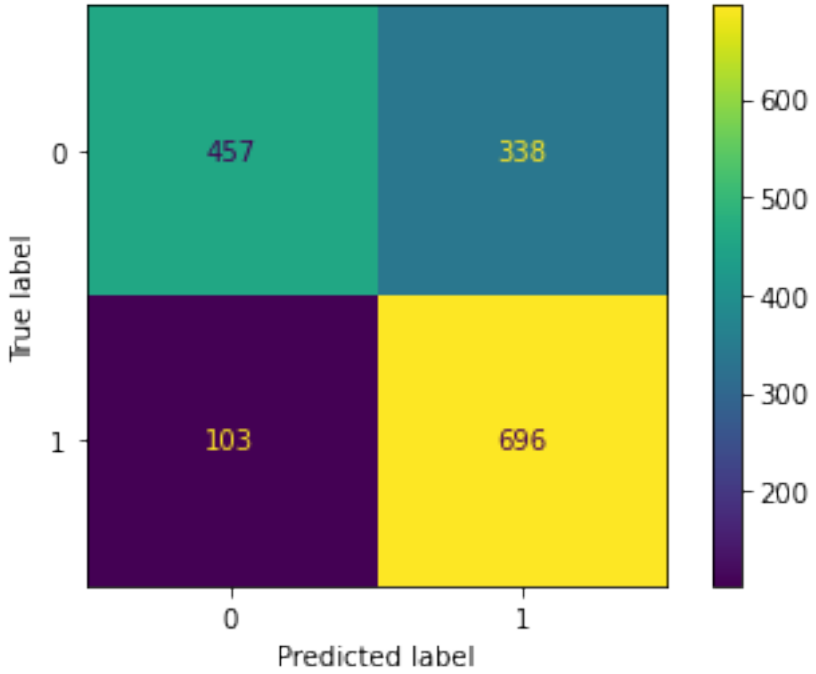
Cette approche utilise le service **Azure Cognitive Service** pour l'analyse de sentiments. **Azure Cognitive Service** regroupe plusieurs modèles déjà entraînés et déployés dans les domaines suivant : Vision (vision par ordinateur : modération de contenu, analyse d'émotions, détection de visage, indexion de vidéos...), Speech (reconnaissance vocale, synthèse vocale, traduction vocale, reconnaissance de l'orateur), Langage (reconnaissance d'entités, analyse d'opinions, réponse au questions, compréhension du langage naturel, traduction), Décision (modération de contenu, détection d'anomalies)... Pour ce projet, c'est donc le service Langage qui nous intéresse, pour sa fonctionnalité d'analyse textuelle de sentiment.

Ces services sont accessibles grâce à une **API REST.**

Après la création du service dans Azure, j'ai développé un script Python permettant d'interroger le service à distance. Les données sont échangées au format JSON. Les étapes du script sont les suivantes :

- Chargement des données.
- Récupération de la clé d'authentification et du endpoint de connection à l'API Azure dans les variables d'environnement (l'URL à laquelle nous envoyons les requêtes).
- Authentification auprès de l'API.
- Envoi de la requête et récupération du résultat Le service renvoie une probabilité (score) pour chacun des sentiments suivants : 'Positif', 'Neutre' et 'Négatif'. Notre jeu de données ne comportant pas de sentiment 'Neutre', nous avons adapté le résultat renvoyé de la façon suivante : le sentiment est considéré comme négatif si le score lié au sentiment négatif est supérieur à 0.5 et positif dans le cas contraire.

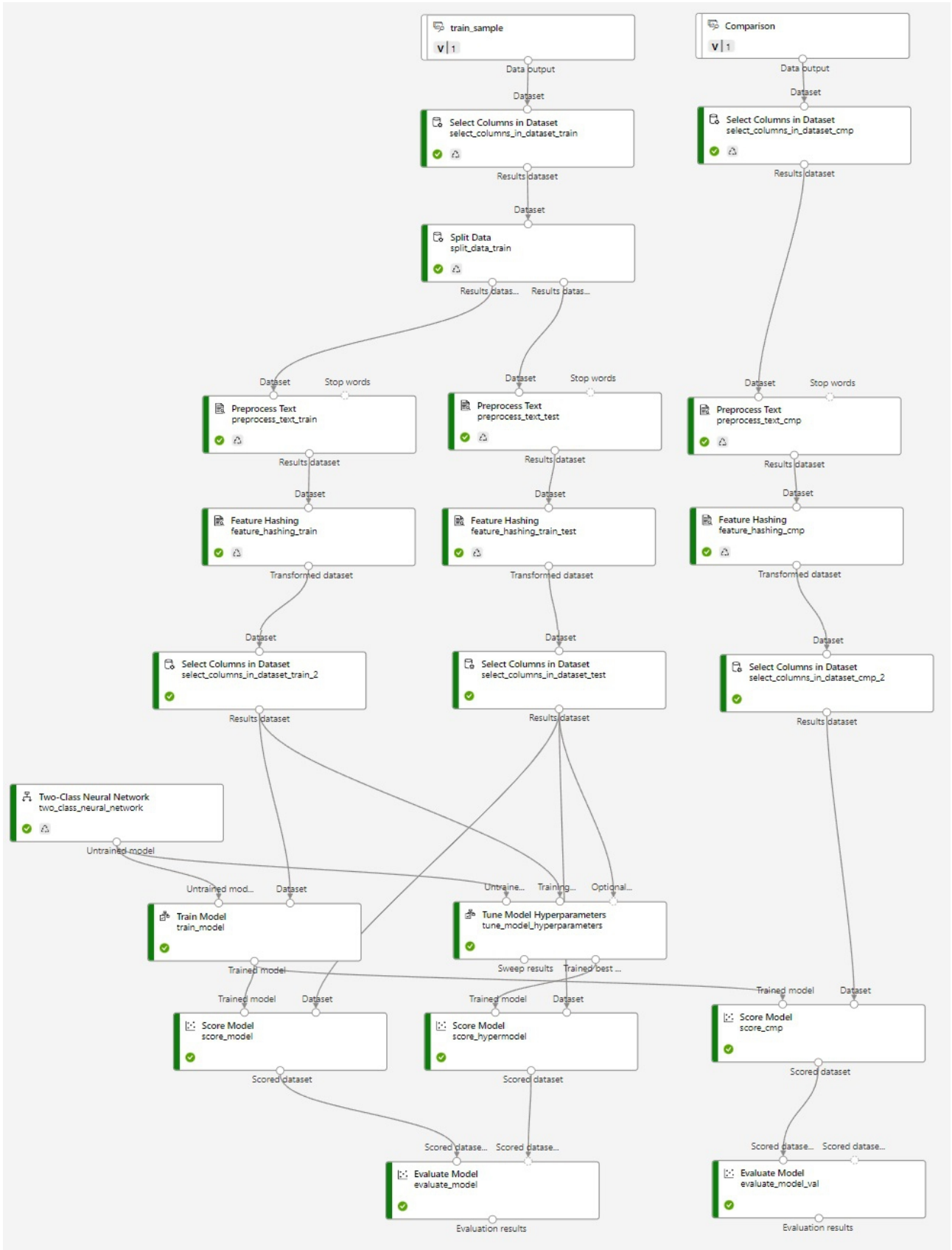
Le score d'Accuracy est de **0.72**



Seconde approche : Modèle sur mesure simple

Pour cette seconde approche, j'ai utilisé le « Designer » proposé par Azure Machine Learning. Avec ce service, il est possible de développer un modèle de Machine Learning classique avec une interface graphique en « drag & drop ». Cette interface nous permet de choisir les différentes briques du modèle (dataset, preprocessing, train, score...) et de les assembler.

Voici le modèle que j'ai réalisé :



Ce modèle est un réseau neuronal effectuant les prédictions sur deux classes (parfait dans notre cas), qui prend les tweets préprocessés en entrée.

Le score d'Accuracy est de **0.631**

Troisième approche : modèle sur mesure avancé

Pour cette troisième approche, j'ai développé un modèle sur mesure dans un notebook jupyter, puis je l'ai déployé sous la forme d'une API avec FastAPI et uvicorn et j'ai codé une page web permettant d'interroger l'API. J'ai tout d'abord préparé les données, en commençant par identifier et analyser la cible (la polarité du sentiment), et j'ai séparé les jeux de données en différents échantillons, en conservant un nombre d'individu équilibré pour chacune des deux classes. Ensuite, j'ai effectué un preprocessing des données textuelles. Pour faire cela, j'ai utilisé deux techniques de pré-traitement : j'ai nettoyé les tweets à l'aide de la librairie Tweet-Preprocessor, puis j'ai effectué une lemmatisation (la lemmatisation rapporte les mots à leur forme canonique, c'est à dire la forme qu'ils ont dans un dictionnaire, en tenant compte du contexte).

J'ai ensuite créé un modèle basique de référence : une régression logistique, à l'aide de la librairie SciKit-Learn et un plongement de mots de type TF-IDF (term frequency-inverse document frequency). Ce modèle de référence a obtenu un score d'Accuracy de **0.753**

Je suis subséquemment passé à la modélisation de différents modèles de Deep Learning, en utilisant différentes architectures de réseaux de neurones, comparé différents types de couches de neurones récurrents (RNN, GRU, LSTM, BGRU et BLSTM), et comparé deux types de plongements de mots pour chacun d'eux : GloVe (préentrainé) et FastText.

J'ai également calculé les temps d'exécution, afin de comparer les modèles.

Il y a eu deux modèles qui arrivaient en tête, avant optimisation, avec des scores d'Accuracy au coude-à-coude : LSTM avec plongement de mots GloVe (accuracy de 0.746 sur le jeu de données de test, et avant optimisation), et GRU avec plongement de mots GloVe (accuracy de 0.745 sur le jeu de données de test, et avant optimisation). J'ai noté que l'utilisation du plongement de mots GloVe pré-entraîné était globalement plus performant que FastText, mais également que son utilisation réduisait l'overfitting.

J'ai donc ensuite optimisé plusieurs hyperparamètres, à savoir les Dropout, le nombre de neurones et le Learning Rate, à l'aide de Keras-Tuner, puis j'ai cherché le nombre d'epochs optimal.

Pour le modèle GRU avec embedding GloVe, je suis arrivé, après optimisation, à un score d'Accuracy de 0,805 sur les données de test, et de 0,808 sur les données de comparaison.

En ce qui concerne le modèle LSTM avec embedding GloVe, le score d'Accuracy est de XXX sur les données de test après optimisation, et de 0.803 sur les données de comparaison.

Pour cette approche, j'ai également tenté autre chose : un transfer learning à partir du modèle BERT pré-entraîné, qui a donné un score d'accuracy de 0,812

C'est donc le modèle BERT que j'ai sélectionné, et que j'ai déployé ensuite à l'aide de FastAPI et de uvicorn. J'ai également créé une page web permettant d'interroger l'IA via l'API avec une saisie utilisateur.

Analyse de sentiments

Texte *

i love this place

ANALYSER

Analyse de sentiments

Résultat de l'analyse :

Sentiment positif

Ce texte exprime un sentiment positif !

Analyse de sentiments

Texte *

this place really sucks

ANALYSER

Analyse de sentiments

Résultat de l'analyse :

Sentiment négatif

Ce texte exprime un sentiment négatif !