**Détection de Sentiments sur Twitter :**

**Comparaison des Modèles**

**et Déploiement sur Azure**

L'analyse des sentiments des messages publiés sur les réseaux sociaux est un défi majeur pour les entreprises cherchant à comprendre leurs clients et réagir rapidement aux tendances.

L’analyse de sentiment est un problème classique de NLP (Natural Language Processing) qui permet de déterminer si le sentiment du texte analysé est positif ou négatif.

Dans ce projet, nous voulons plus particulièrement anticiper les « bad buzz », sous forme de tweets, qui pourraient ternir l’image d’une entreprise. Pour cela, après une exploration et un pré-traitement des données que nous ne détaillerons pas dans cet article, nous avons exploré plusieurs modèles de classification des sentiments, puis déployé le meilleur sur Azure, tout en respectant une démarche MLOps pour automatiser le suivi et l'amélioration continue du modèle déployé en production.

Table des matières

[1. Comparaison des modèles de classification 2](#_Toc194674311)

[a) Modèle simple : Régression logistique 2](#_Toc194674312)

[b) Modèles avancé : RNN (Recurrent Neural Network) 2](#_Toc194674313)

[c) Modèle BERT : ModernBERT 4](#_Toc194674314)

[d) Résultats comparatifs 5](#_Toc194674315)

[2. Démarche MLOps : Du développement au déploiement 6](#_Toc194674316)

[a) Principes du MLOps 6](#_Toc194674317)

[b) Étapes mises en œuvre 6](#_Toc194674318)

[3. Suivi de la performance en production 9](#_Toc194674319)

[a) Monitoring avec Azure Application Insights 9](#_Toc194674320)

[b) Analyse des performances et amélioration continue 9](#_Toc194674321)

# 1. Comparaison des modèles de classification

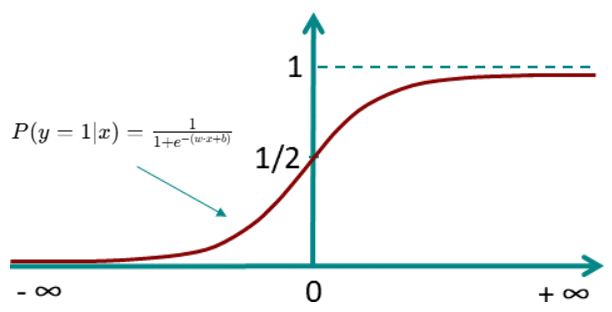
## a) Modèle simple : Régression logistique

La **régression logistique** constitue une approche simple et rapide qui permet d’avoir une base de comparaison pour les autres modèles testés.

Nous avons testé ce modèle statistique avec plusieurs configurations :

* Avec ou sans racinisation par stemming,
* Vectorisation via CountVectorizer ou TF-IDF.

Les tweets transformés en vecteur via une **vectorisation TF-IDF** (Term Frequency – Inverse Document Frequency) sont utilisés par la régression logistique pour la classification binaire qui estime la probabilité qu’une observation appartienne à une classe donnée à l’aide de la fonction logistique :



Une image contenant texte, Police, capture d’écran, blanc

Le contenu généré par l’IA peut être incorrect.

La limitation de ce modèle est qu’il ne capture pas bien la sémantique ce qui réduit les performances de classification.

## b) Modèles avancé : RNN (Recurrent Neural Network)

Le **réseau de neurones récurrents (RNN)** est une architecture adaptée aux séquences de données comme le texte. Dans ce projet, nous avons utilisé un **LSTM (Long Short-Term Memory)**, une amélioration du RNN classique, conçue pour surmonter les limitations liées à la gestion des informations pertinentes sur de longues périodes, ce qui le rend plus adapté aux tâches de classification de texte.

Chaque cellule LSTM comprend plusieurs portes qui régulent le flux d’informations :

* **La porte d’oubli** : détermine quelle information doit être oubliée. Elle prend comme entrée la sortie précédente (état caché h(t-1)) et l'entrée courante de la séquence x(t), puis on y applique la fonction sigmoïde. Si la sortie de la sigmoïde est proche de 0, cela signifie que l’on doit oublier l’information et si on est proche de 1 alors il faut la mémoriser pour la suite.
* **La porte d’entrée** : contrôle quelles informations seront ajoutées à la cellule de mémoire. Son fonctionnement est le même que celle d’oubli, on applique ensuite une fonction tanh et le produit des deux permettra de ne garder que les informations importantes.
* **L’état de la cellule :** obtenu en multipliant la sortie de la porte de l’oubli avec l’ancien état de la cellule c(t-1). Cela permet d’oublier certaines informations de l’état précédent qui ne servent pas pour la prédiction. Ensuite, on additionne le tout avec la sortie de la porte d’entrée, ce qui permet d’enregistrer dans l’état de la cellule c(t) ce que le LSTM a jugé pertinent.
* **La porte de sortie** : détermine quel sera le prochain état caché h(t). La première étape est identique aux deux premières portes. On applique également une fonction tanh au nouvel état de la cellule. Le produit des deux créera une sortie qui est ensuite envoyée à l'étape suivante du réseau.

Une image contenant texte, ligne, diagramme, capture d’écran

Le contenu généré par l’IA peut être incorrect.

Ce modèle a été testé sans embeddings pré-entrainés, puis avec des embeddings GloVe et FastText afin d’améliorer la représentation sémantique des séquences. **GloVe** (Global Vectors for Word Representation) se base sur la **co-occurrence des mots** et capture les relations entre les mots en analysant leur fréquence d’apparition conjointe dans une fenêtre contextuelle. **FastText**, contrairement à GloVe, ne considère pas uniquement les mots, mais aussi leurs **sous-mots (n-grams)**, ce qui lui permet de mieux gérer les mots rares et les variations morphologiques.

## c) Modèle BERT : ModernBERT

Contrairement aux modèles récurrents comme le LSTM, qui traitent un mot à la fois, **ModernBERT** repose sur l’architecture **Transformers**, qui permet de capturer le contexte global d’une phrase **en un seul passage** grâce à un **mécanisme d’attention**.

BERT (**Bidirectional Encoder Representations from Transformers**) a été conçu pour apprendre des **représentations bidirectionnelles**, ce qui signifie qu’il tient compte de **tout le contexte d’un mot**, en regardant **à la fois avant et après lui** dans la phrase. Il se compose de plusieurs **blocs encodeurs empilés**, et **ModernBERT** en reprend l'architecture tout en l’optimisant pour plus d’efficacité.

Chaque **encodeur** contient deux sous-couches principales :

* **Mécanisme d’auto-attention multi-têtes** : pour chaque mot, détermine son importance relative par rapport aux autres mots de la phrase.

Pour cela, chaque mot est projeté en trois vecteurs :

* **Query (Q)** : représente le mot actuel
* **Key (K)** : représente les caractéristiques des autres mots de la phrase
* **Value (V)** : représente les informations que le mot porte.

L’attention est définie par la formule :

Une image contenant texte, Police, blanc, ligne

Le contenu généré par l’IA peut être incorrect.

Où on calcule la similarité entre le **Query (Q)** et les **Keys (K)** des autres mots à l’aide de la multiplication scalaire normalisé par la racine de la dimension des clés dk, puis on applique une fonction **softmax** sur ces similarités pour obtenir un **poids d'attention**. Enfin, ces poids d'attention sont utilisés pour **pondérer les valeurs V (Values)** ce qui permet de sélectionner les informations pertinentes.

Une image contenant diagramme, capture d’écran, texte, cercle

Le contenu généré par l’IA peut être incorrect.

ModernBERT utilise **plusieurs têtes d’attention en parallèle**, appelées **multi-head attention** : cela permet au modèle d’extraire **différents types de relations sémantiques** entre les mots (par exemple : sujet-verbe, adjectif-nom, etc.).

* **Réseau Feed-forward positionnel :**permet au modèle d'apprendre des relations complexes dans les données en transformant la sortie de la couche d'attention via **deux couches linéaires** séparées par une fonction d'activation non linéaire (typiquement une fonction **ReLU** (Rectified Linear Unit)). Ensuite, la **couche de normalisation** permet de stabiliser l'entraînement et d'améliorer la convergence en réduisant la variance dans les activations.

Le terme positionnel fait référence à l'intégration de l'information positionnelle dans le modèle, ce qui est nécessaire pour

Puisque les Transformers ne traitent pas les séquences de manière récursive comme les RNNs, mais de manière parallèle, les embeddings positionnels sont ajoutés aux entrées de chaque mot dans le modèle, permettant ainsi au modèle de tenir compte de la position de chaque mot dans la phrase.

Dans ModernBERT, ces éléments sont optimisés pour réduire la complexité et améliorer la rapidité d'entraînement.

## d) Résultats comparatifs

Après avoir testé différents modèles, il est essentiel de comparer leurs performances sur un même jeu de données.

Les modèles ont été comparés selon plusieurs métriques classiques :

* **Recall Classe 0** : rappel sur la classe négative, métrique prioritaire ici
* **Accuracy** : part des prédictions correctes
* **F1-score** : équilibre entre précision et rappel, moyenné sur toutes les classes
* **AUC-ROC** : performance globale du modèle à distinguer les classes

Voici le tableau des résultats :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Recall Classe 0 | Accuracy | F1-score | AUC-ROC |
| Régression logistique | 0.75 | 0.76 | 0.76 | 0.84 |
| LSTM sans embeddings | 0.76 | 0.62 | 0.61 | 0.69 |
| LSTM + GloVe | 0.74 | 0.67 | 0.67 | 0.76 |
| LSTM + FastText | 0.75 | 0.71 | 0.70 | 0.79 |
| ModernBERT | 0.77 | 0.73 | 0.73 | 0.81 |

Les performances obtenues révèlent que la **régression logistique** reste étonnamment compétitive. Malgré sa simplicité, ce modèle capte efficacement les signaux textuels grâce à une représentation des mots très informative.

Les modèles plus complexes comme **LSTM** et **ModernBERT**, bien qu’architecturalement plus avancés, ne surpassent pas largement cette baseline. La taille des données ainsi que la longueur des tweets limitées peuvent expliquer que ces modèles ne puissent exprimer tout leur potentiel.

Cela nous enseigne que, dans un contexte contraint, **la simplicité peut rimer avec efficacité.**

Notre métrique prioritaire étant le rappel de la classe négative, nous déploierons l’algorithme ModernBERT.

# 2. Démarche MLOps : Du développement au déploiement

Une fois le modèle sélectionné, une nouvelle question se pose : **comment industrialiser cette solution** ?

C’est là qu’intervient le **MLOps** (Machine Learning Operations), un ensemble de pratiques qui visent à fiabiliser, automatiser et surveiller l’ensemble du cycle de vie d’un modèle, de la conception à la mise en production.

## a) Principes du MLOps

* **Traçabilité** : pouvoir retrouver à tout moment les données, les paramètres, le code et les métriques associés à un modèle.
* **Automatisation** : des pipelines reproductibles pour éviter les erreurs humaines.
* **Versioning** : suivi rigoureux des versions de code, de données, et de modèles.
* **Testabilité** : validation systématique du bon fonctionnement (tests unitaires, validation des performances).
* **Surveillance continue** : suivi en production avec alertes en cas de dérive de performance.

## b) Étapes mises en œuvre

* **Tracking et stockage des expériences avec MLflow**

Chaque expérience d’entraînement a été enregistrée dans MLflow avec :

* **Les hyperparamètres** (nombre d’unités LSTM, taille des séquences …)
* **Les métriques de performance**(rappel de la classe négative, F1-score…)
* **Le fichier du modèle entraîné**
* **Les artefacts**(matrice de confusion)
* **Un identifiant de version unique**

Cela permet de comparer objectivement les modèles, identifier rapidement le meilleur, le sauvegarder, suivre ses versions et garantir sa reproductibilité.

Le stockage des modèles accompagné de ses métadonnées et conditions d’entraînement permet de les réutiliser sans les entrainer à nouveau, de déployer la même version dans plusieurs environnement (dev, prod, …) et de revenir à une version précédente en cas de besoin.

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

* **Déploiement automatisé avec tests unitaires**

Le modèle validé a été packagé sous forme de service API REST via **FastAPI** et déployé dans l’environnement cloud **Azure**, de façon automatisée via **Github Actions**.

Pour valider le bon fonctionnement de l’API et donc valider le déploiement automatique, des **tests unitaires** ont été mis en place juste avant le déploiement pour vérifier la bonne prédiction du modèle avec un tweet positif et un tweet négatif.

* **Suivi de la performance en production avec Azure Application Insights**

Une fois en production, le modèle est **monitoré en temps réel** pour anticiper les dérives de performances en collectant les feedbacks utilisateurs avec **Azure Application Insights**.

Une image contenant texte, capture d’écran, nombre, affichage

Le contenu généré par l’IA peut être incorrect.

Une alerte a également été configuré : si 3 feedbacks négatifs sont envoyés dans un intervalle de 5 minutes, un mail d’alerte est envoyé. Cela permettra de trouver une stratégie (réentraînement, optimisation du pré-processing, …) pour améliorer la satisfaction des utilisateurs.

Ce projet m’a permis de mettre en œuvre un pipeline complet de **classification binaire**, allant du pré-traitement des données à l’évaluation des performances de notre API, en passant par le déploiement du modèle ModernBERT. Grâce à cette démarche **MLOps**, le cycle de vie du machine learning devient plus **structuré**, plus **traçable** et plus **réactif**. L’**intégration de MLflow** m’a également offert une valeur ajoutée en facilitant le suivi des versions et des performances de chaque modèle. Cette approche permet non seulement de **gagner en efficacité opérationnelle**, mais aussi de **renforcer la confiance** des utilisateurs.