

## Comparaison des Approches de Modélisation et Démarche MLOps

### Introduction

Dans le domaine du Machine Learning appliqué au traitement du langage naturel (NLP), plusieurs modèles peuvent être utilisés selon le niveau de complexité et les besoins en performance. Dans cet article, nous allons comparer trois approches :

1. **Modèle sur mesure simple** : Régression logistique
2. **Modèle sur mesure avancé** : LSTM avec Word2Vec et LSTM avec Glove
3. **Modèle avancé BERT**

Nous présenterons également la démarche MLOps mise en place pour le déploiement et la maintenance de ces modèles.

---

### 1. Comparaison des Approches

#### [P7 - Réalisez une analyse de sentiments](#)

##### **1.1 Modèle sur mesure simple : Régression Logistique**

La régression logistique est un modèle classique et efficace pour la classification de texte. Elle présente les avantages suivants :

- Simplicité d'implémentation
- Faible coût computationnel
- Facilité de déploiement

Cependant, elle est limitée lorsqu'il s'agit de capturer des relations complexes dans le texte.

##### **1.2 Modèles sur mesure avancés : LSTM avec Word2Vec et Glove**

Les modèles LSTM permettent de capturer la dépendance contextuelle dans le texte.

- **LSTM + Word2Vec** : Entraîné sur un corpus présent, ce modèle capture mieux la structure sémantique.
- **LSTM + Glove** : Utilise des embeddings pré-entraînés pour une meilleure compréhension des mots.

Ces modèles sont plus performants que la régression logistique mais plus complexes à déployer.

##### **1.3 Modèle Avancé : BERT**

BERT est un modèle de pointe qui capte efficacement le contexte bidirectionnel d'un texte. Il présente les avantages suivants :

- Excellentes performances sur les tâches NLP
- Gère des relations complexes dans le texte

Cependant, BERT est coûteux en ressources et nécessite une architecture MLOps robuste.

### Comparaison des modèles de classification NLP avec MLOps

En sélectionnant les différentes expériences correspondant à chaque approche, puis en choisissant les exécutions (runs) les plus performantes pour chaque modèle, et enfin en utilisant la fonction de comparaison, on obtient un récapitulatif des différentes métriques enregistrées pour chaque modèle.

The screenshot shows the mlflow interface version 2.20.1. On the left, under 'Experiments', several runs are listed: 'Default' (unchecked), 'LSTM\_Word2Vec\_Text\_Classifi...' (checked), 'LSTM\_GloVe\_Text\_Classification' (checked), 'tweet\_prediction\_by\_regressi...' (checked), 'BERT\_Text\_Classification' (checked), and 'LSTM Experiment with Hyper...' (unchecked). On the right, under 'Displaying Runs from 4 Experiments', the 'Experimental' tab is selected. A green box highlights the 'Compare' button. Below it is a table comparing four runs:

Run Name	Created	Dataset
BERT_Text_Classification_R...	3 days ago	-
indecisive-swan-579	5 days ago	-
LSTM_GloVe_Text_Classifi...	6 days ago	-
Best_Logistic_Regression	7 days ago	-
Logistic Regression Model	7 days ago	-

Table de comparaison des modèles

Mesure	BERT	LSTM GloVe	LSTM Word2Vec	Logistic Regression
Test Accuracy	0.766	0.785	0.766	0.767
Test Loss	0.482	0.453	0.482	0.482
Train Accuracy	0.974	0.778	0.774	0.778
Train Loss	0.071	0.471	0.473	0.471
Validation Accuracy	0.816	0.788	0.788	0.770
Validation Loss	0.706	0.45	0.46	0.484

Le modèle de régression logistique atteint des résultats encourageants :

- ❖ **Test Accuracy : 76%**
- ❖ **Train Accuracy : 77%**
- ❖ **Validation Accuracy : 77 %**

Ces performances témoignent de la capacité du modèle à généraliser correctement, tout en restant optimisé pour une mise en production rapide.

---

## 2. Démarque MLOps

La mise en production de modèles NLP passe par une approche MLOps structurée. Voici les étapes mises en place :

### 2.1 Principes MLOps

L'approche MLOps repose sur les principes suivants :

- Automatisation du pipeline ML
- Reproductibilité des expériences
- Suivi de la performance en production
- Maintenance continue et amélioration du modèle

### 2.2 Mise en œuvre

#### a) Tracking des Expériences

L'expérimentation est suivie avec MLflow afin de :

- Enregistrer les métriques (précision, rappel, f1-score)
- Stocker les hyperparamètres des modèles

The screenshot shows the MLflow UI interface. At the top, there is a code editor window containing Python code for logging metrics:

```
mlflow.log_metric("test_accuracy", test_accuracy)
mlflow.log_metric("test_auc", test_auc)
mlflow.log_metric("training_time", training_time)
mlflow.log_metric("val_accuracy", val_accuracy)
```

Below the code editor is a navigation bar with tabs: Runs, Evaluation, Experimental (which is selected), and Traces. There are also buttons for Provide Feedback, Add Description, and Share.

The main area displays a table of experimental runs. The columns include Run Name, Created (sorted by creation date), Duration, Models, and various metrics: Test AUC, Test Accuracy, Test Log-Loss, Train AUC, and Train Acc. The table lists multiple runs, each with a unique color-coded icon next to its name. One run is explicitly labeled "Best\_Logistic\_Regression".

Metrics								
	Run Name	Created	Duration	Models	Test AUC	Test Accuracy	Test Log-Loss	Train AUC
	Best_Logistic_Regression	7 days ago	20.3s	sklearn	0.849681427...	0.7672	0.482021325...	0.858320322...
	Logistic_Regression_Mod...	7 days ago	21.7s	sklearn	0.849273427...	0.76788	0.482597813...	0.857881422...
	Logistic_Regression_Mod...	7 days ago	23.1s	sklearn	0.849030057...	0.76708	0.482955251...	0.858844066...
	Logistic_Regression_Mod...	7 days ago	19.0s	sklearn	0.849133197...	0.7677	0.483024502...	0.856462167...
	Logistic_Regression_Mod...	7 days ago	22.4s	sklearn	0.849681427...	0.7672	0.482021325...	0.858320322...
	Logistic_Regression_Mod...	7 days ago	21.1s	sklearn	0.845484305...	0.76442	0.494806015...	0.850629219...
	Logistic_Regression_Mod...	7 days ago	20.2s	sklearn	0.845708248...	0.76472	0.494731405...	0.851067701...
	Logistic_Regression_Mod...	7 days ago	20.3s	sklearn	0.825872306...	0.74694	0.553747115...	0.828077888...
	Logistic_Regression_Mod...	7 days ago	34.6s	sklearn	0.825911410...	0.7469	0.553753099...	0.828120352...
	Best_Logistic_Regression	20 days ago	4.6s	sklearn	0.810915268...	0.726875	0.533017371...	0.897319114...

## b) Stockage et Gestion de Version des Modèles

Le modèle est sauvegardé sous forme d'un fichier **best\_model.pkl**, qui contient un **pipeline** intégrant à la fois :

- **Le vectorizer TF-IDF** (*best\_vectorizer*) pour transformer les nouveaux tweets,
- **Le modèle de régression logistique** (*LogisticRegression*) entraîné pour la classification.

Cette approche permet de charger facilement le pipeline pour l'inférence sans avoir à refaire le prétraitement ni à réentraîner le modèle.

```
# GridSearch et suivi MLflow
pipeline = Pipeline([
    ('tfidf', best_vectorizer),
    ('logreg', LogisticRegression(max_iter=1000))
])

# Enregistrement du meilleur modèle
best_model = grid_search.best_estimator_
joblib.dump(best_model, 'best_model.pkl')
print("Modèle sauvegardé : best_model.pkl")
```

## c) Tests Unitaires

[test.py - P7 Analyse de Sentiments](#)

## Tests unitaires

Des tests unitaires ont été réalisés pour garantir la robustesse de l'API de prédiction déployée avec FastAPI. Les tests, implémentés avec pytest, vérifient les points suivants:

- **Accessibilité de l'API :** Vérification que la route racine ( / ) est accessible et répond correctement.
- **Prédiction avec un texte valide :** Vérifie que l'API retourne bien une prédiction de sentiment.
- **Gestion des erreurs :**
  - Texte manquant dans la requête.
  - Texte vide.
  - Texte contenant uniquement des chiffres.
  - Requête mal formatée (non JSON).

```
def test_predict_missing_text():
    """Test avec un champ 'text' manquant"""
    input_data = {}
    response = client.post("/predict/", json=input_data)
    assert response.status_code == 422 # Unprocessable Entity
    assert "detail" in response.json()
    assert "loc" in response.json()["detail"][0] # Validation de la présence de 'loc'

def test_predict_empty_text():
    """Test avec un texte vide"""
    input_data = {"text": ""}
    response = client.post("/predict/", json=input_data)
    assert response.status_code == 400 # Mauvaise requête (texte vide)
    assert response.json() == {"detail": "Le texte ne peut pas être vide."}

def test_predict_number_text():
    """Test avec un texte contenant uniquement un nombre"""
    input_data = {"text": "12345"}
    response = client.post("/predict/", json=input_data)
    assert response.status_code == 400 # Texte ne peut pas être un nombre
    assert response.json() == {"detail": "Le texte ne peut pas être un nombre."}
```

## d) Déploiement

 [pythonsentimentsapp.azurewebsites.net](https://pythonsentimentsapp.azurewebsites.net)

Nous avons mis en place un pipeline **CI/CD** avec **GitHub Actions** pour automatiser le déploiement sur Azure. Voici les étapes clés du workflow :

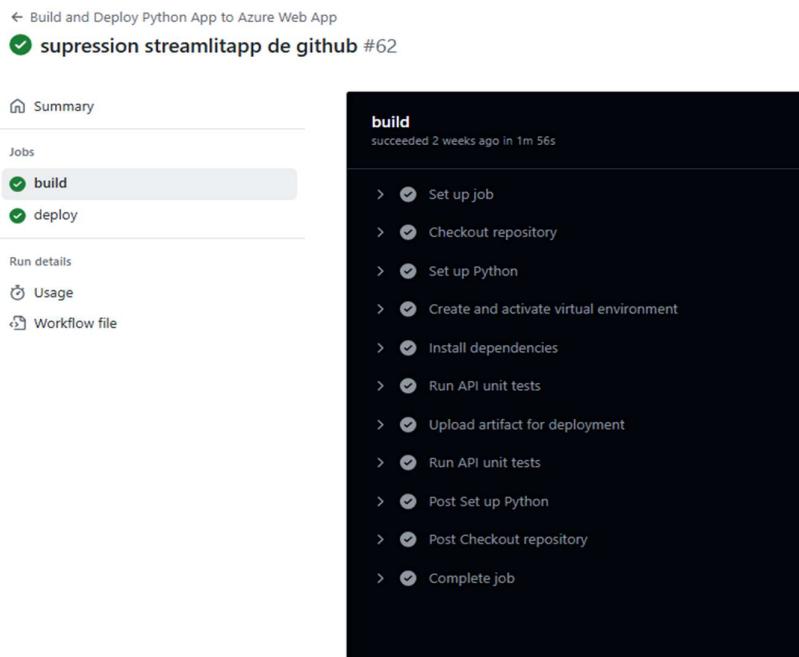
1. **Push sur GitHub** : Dès qu'un changement est poussé dans le dépôt, GitHub Actions est déclenché.
2. **Build et test du modèle** : Le modèle est entraîné et validé automatiquement.
3. **Upload sur Azure** : Le modèle est envoyé et enregistré dans Azure.

4. **Déploiement automatique** : Une fois validé, le modèle est mis en production et exposé via un endpoint Azure.

### Avantages du Déploiement avec GitHub Actions

- **Automatisation complète** : Pas besoin de déployer manuellement le modèle.
- **Traçabilité** : Chaque version du modèle est associée à un commit GitHub.
- **Facilité d'intégration** : Le modèle peut être utilisé par d'autres services Azure sans intervention manuelle.

En résumé, ce pipeline **CI/CD** avec **GitHub Actions** permet un **déploiement rapide et fiable** du modèle sur **Azure**.



← Build and Deploy Python App to Azure Web App  
✓ suppression streamlitapp de github #62

The screenshot shows a GitHub Actions deployment log for a job named 'deploy'. The job succeeded 2 weeks ago in 2m 23s. The steps listed are: Set up job, Pre Login to Azure, Download artifact, Login to Azure, Deploy to Azure Web App, Post Login to Azure, and Complete job. All steps are marked as completed with green checkmarks.

#### e) Suivi en Production : Traces et Alertes avec Azure Application Insights

Afin d'assurer un suivi en temps réel des prédictions et d'optimiser les performances du modèle, j'ai intégré **Azure Application Insights** pour la **collecte des logs** et la **surveillance des erreurs**.

#### Interface Utilisateur avec Streamlit

J'ai développé une interface avec **Streamlit**, permettant aux utilisateurs d'entrer du texte et d'obtenir immédiatement la prédiction du sentiment. En cas de sentiment négatif, un log est automatiquement envoyé à **Azure Application Insights** pour analyse.

#### Paramétrage d'une Alerte pour les Sentiments Négatifs

Pour anticiper les anomalies, j'ai configuré une **alerte automatique** qui se déclenche si trois sentiments négatifs sont détectés en **moins de 5 minutes**. Cette alerte envoie une notification par e-mail, permettant une réaction rapide si des tendances négatives émergent.

The screenshot shows the Azure Monitor Alerts interface. At the top, there's a search bar, filter options for 'Resource name: pythonsentimentsapp\_insights', 'Time range: Past 24 hours', 'Alert condition: Fired', 'Severity: all', and a 'Add filter' button. Below this, a summary bar shows 'Total alerts: 1', 'Critical: 0', 'Error: 0', 'Warning: 0', 'Informational: 1', and 'Verbose: 0'. A dropdown menu 'No grouping' is open. Below the summary are sorting options: 'Name ↑↓', 'Severity ↑↓', 'Affected resource ↑↓', 'Alert condition ↑↓', 'User response ↑↓', and 'Fire time ↑↓'. A single alert card is displayed, titled 'Détection d'un pic de logs - 3 - Informational' for the resource 'pythonsentimentsapp\_insights'. The alert status is 'Fired' (indicated by a red triangle icon) and was created 'New' at '2/17/2025, 11:18 PM'. The alert text reads:

**Resolved:Sev3 Azure Monitor Alert Alerte**  
**Sentiment Négatif - 10 Traces 5mn on**  
**pythonsentimentsapp\_insights (**  
**microsoft.insights/components ) at 3/11/2025**  
**12:18:29 PM**

Below the alert text are two buttons: 'View the alert in Azure Monitor >' (blue) and 'Investigate >' (white). The overall page has a light blue header with the Microsoft Azure logo and a 'Se désabonner' (Unsubscribe) link.

## Collecte des Logs et Suivi des Erreurs

Chaque détection d'un sentiment négatif génère un **log d'alerte**, permettant d'identifier les tendances et d'ajuster le modèle en conséquence. J'ai ainsi pu :

- Surveiller le nombre de sentiments négatifs détectés.
- Suivre les erreurs de connexion à l'API.
- Observer en temps réel le comportement de l'API et les retours des utilisateurs.

Grâce à ces données, j'ai pu affiner mon modèle et améliorer la précision des prédictions en production.

The screenshot shows the Azure Application Insights interface. On the left, there's a navigation sidebar with sections like 'Investigate', 'Logs' (which is selected), and 'Metrics'. The main area is titled 'New Query 1\*' and shows a table of log entries. The table has columns: timestamp [UTC], message, severityLevel, itemType, and customDimensions. The data in the table is as follows:

timestamp [UTC]	message	severityLevel	itemType	customDimensions
> 2/11/2025, 8:29:17.066 PM	⚠️ Sentiment négatif détecté : ugly	2	trace	{"process": "MainProcess", "module": "streamlit"}
> 2/11/2025, 8:29:17.066 PM	Prédiction reçue : 0, Sentiment : Negative	2	trace	{"process": "MainProcess", "module": "streamlit"}
> 2/11/2025, 8:29:17.066 PM	⚠️ Sentiment négatif détecté : ugly	2	trace	{"process": "MainProcess", "module": "streamlit"}
> 2/11/2025, 8:29:17.066 PM	Prédiction reçue : 0, Sentiment : Negative	2	trace	{"process": "MainProcess", "module": "streamlit"}
> 2/11/2025, 8:29:17.066 PM	⚠️ Sentiment négatif détecté : ugly	2	trace	{"process": "MainProcess", "module": "streamlit"}
> 2/11/2025, 8:29:17.066 PM	Prédiction reçue : 0, Sentiment : Negative	2	trace	{"process": "MainProcess", "module": "streamlit"}

## 2.3 Amélioration Continue

Pour garantir la performance du modèle sur le long terme, plusieurs actions sont mises en place :

### Analyse des erreurs en production

- Collecte des logs via Azure Application Insights, notamment pour les prédictions négatives.
- Suivi des erreurs pour identifier les tendances et comprendre dans quelles conditions le modèle se trompe.

### Identification des causes des erreurs

- Analyse des prédictions incorrectes : comparaison avec les labels réels pour détecter les faux positifs et faux négatifs.
- Évaluation des performances : utilisation de tableaux de confusion et de métriques comme la précision et le rappel.
- Examen des données en entrée : vérification de la qualité des textes (orthographe, abréviations, emojis) et détection des biais éventuels dans le dataset.

### Fine-tuning et mises à jour régulières

- Ajustement des hyperparamètres et réentraînement sur des données mises à jour.
- Intégration continue des nouvelles données pour adapter le modèle aux évolutions du langage et des usages.
- Amélioration du pipeline de traitement afin d'optimiser les performances en production.

En appliquant ces stratégies, le modèle évolue continuellement et s'adapte aux nouveaux contextes, garantissant ainsi une meilleure robustesse et fiabilité.

---

### **3. Conclusion**

Nous avons comparé trois approches de modélisation NLP et mis en place une démarche MLOps complète. Chaque modèle présente des avantages et des inconvénients, et le choix dépend des ressources et de la complexité du projet. Grâce à ce projet, j'ai construit un pipeline complet, de l'entraînement du modèle à son déploiement automatisé sur Azure via GitHub Actions. L'intégration d'Azure Application Insights a permis un suivi en temps réel, garantissant une gestion optimale et une amélioration continue du modèle.