stack overflow

CATEGORISEZ AUTOMATIQUEMENT DES QUESTIONS

NOTE TECHNIQUE MLOPS

Projet 5 OPENCLASSROOMS – Machine Learning Engineer

Bastien Moreno

TABLE DES MATIERES

CONTEXTUALISATION DU PROJET

Stack Overflow est une plateforme de questions-réponses largement plébiscitée par les développeurs du monde entier. Fondée en 2008, elle est devenue une référence incontournable pour ceux en quête de solutions à des problèmes de programmation, de conseils sur les meilleures pratiques de développement, ou encore pour participer à des discussions sur des sujets techniques variés.

La plateforme fonctionne sur un modèle communautaire où les utilisateurs peuvent poser des questions, fournir des réponses, voter pour les meilleures réponses et commenter. Les questions sont organisées autour de tags, des mots-clés qui identifient le sujet principal de chaque publication. Les utilisateurs ont également la possibilité d'attribuer des tags à leurs propres questions pour les rendre plus facilement accessibles aux autres membres de la communauté.

Stack Overflow abrite une vaste quantité de données, comprenant un nombre considérable de questions et de réponses portant sur une multitude de sujets de programmation. Ces données peuvent être extraites du site grâce à une API publique, fournissant ainsi les données nécessaires à l'élaboration d'un futur modèle de machine learning.

OBJECTIF DU PROJET

Notre objectif est de développer un modèle de machine learning capable de suggérer automatiquement des tags lorsque l'utilisateur rédige sa question. Pour ce faire, nous avons extrait les 50 000 questions les plus consultées, ayant au moins une réponse et comportant au moins 5 tags. L'objectif était de transformer ces données afin de les utiliser pour l'entraînement d'un modèle de machine learning, qui serait ensuite déployé en production via une API comme dans le mock-up ci-dessous :

A PROPOS DE CETTE NOTE TECHNIQUE

Cette note technique se concentre principalement sur les aspects liés à la gestion du cycle de vie du modèle, en mettant l'accent sur l'approche MLOps. Nous n'entrerons donc pas dans les détails du processus de nettoyage des données ni de la sélection du modèle de machine learning lui-même. Notre objectif est plutôt de proposer une étude approfondie sur les approches et les outils visant à généraliser l'approche MLOps dans le cadre de ce projet. Nous explorerons en détail les concepts liés au suivi de la performance du modèle, ainsi que les outils et les techniques pour gérer efficacement les alertes en cas de dégradation de la performance.

1) INTRODUCTION AU MLOPS

L'objectif principal du MLOps est de faciliter le déploiement de solutions de machine learning en production, permettant ainsi aux utilisateurs finaux d'accéder aux fonctionnalités du modèle.

Dans le cadre de notre projet, nous avons développé un modèle de prédiction automatique visant à caractériser les questions des utilisateurs du site Stack Overflow à l'aide de tags, qui sont des mots-clés associés à chaque question.

Pour mener à bien ce projet, nous avons divisé le travail en deux principales phases :

Première phase : Collecte de données, prétraitement de celles-ci et développement du modèle.
Cette phase consiste à rassembler les données pertinentes, à les prétraiter pour les rendre utilisables par le modèle, puis à développer le modèle de prédiction.

Seconde phase : Test, déploiement et surveillance du modèle.
Une fois le modèle développé, il est testé pour évaluer sa performance, puis déployé en production pour une utilisation par les utilisateurs finaux.

Cependant, une approche aussi linéaire peut s'avérer réductrice et ne pas correspondre pleinement à la complexité des cas d'utilisation réels. En réalité, le cycle de vie d'un modèle en production est un processus continu et itératif, c'est-à-dire, un processus qui se répète de manière cyclique et qui implique des phases de développement, de test, de déploiement et de surveillance.

Aller de la collecte de données à la mise en production sans prendre en compte les besoins d'itération, d'optimisation et de maintenance continue peut conduire à des résultats insatisfaisants. Ainsi, il est essentiel d'adopter une approche de type MLOps, intégrant des pratiques telles que l'optimisation continue du modèle, la gestion du drift et la réévaluation périodique des besoins et des performances.

On va donc parler du cycle de vie du machine learning qui est illustré par le schéma ci-dessous :

Nous adoptons une approche dynamique du cycle de vie du modèle, où chaque étape est suivie d'une itération visant à améliorer le modèle en fonction des données d'utilisation et de nouvelles données disponibles. Cela signifie que le modèle ne reste pas figé, mais évolue et s'adapte en continu pour refléter les changements dans l'environnement et les besoins des utilisateurs.

À chaque itération, nous prenons en compte l'utilisation réelle de l'application pour identifier les lacunes et les opportunités d'amélioration. Ces observations nous guident dans la mise à jour du modèle, puis dans une nouvelle mise en production. Ce processus itératif assure que le modèle reste pertinent, précis et efficace au fil du temps, garantissant ainsi une expérience utilisateur optimale.

Cependant, respecter ce cycle itératif ne garantit pas nécessairement que le modèle restera à jour et pertinent à long terme. Le défi consiste donc à accélérer ce processus itératif pour éviter que le modèle ne devienne obsolète.

Nous allons donc explorer ensemble les différentes stratégies et les outils disponibles pour atteindre cet objectif dans le cadre spécifique de notre projet. En identifiant les meilleurs pratiques et en utilisant des outils spécialisés, nous visons à maximiser l'efficacité et la rapidité de notre cycle de vie du modèle tout en garantissant la qualité et la fiabilité du modèle en production.

2) MISE A DISPOSITION DU MODELE

Dans cette section, nous abordons directement l'étape de développement du modèle après avoir effectué le prétraitement des données et la tokenisation des documents du corpus. Nous avons entraîné différents types de modèles supervisés pour répondre aux besoins spécifiques de notre projet.

A) UTILISATION DE MLFLOW TRACKING

Nous avons intégré MLFlow Tracking dans notre processus de développement pour suivre et enregistrer les différentes expérimentations des modèles. L'objectif était de pouvoir enregistrer directement les scores et d'utiliser la fonction de suivi pour sélectionner le modèle présentant les meilleures performances. Dans cette optique, nous avons choisi l'indice de Jaccard pour évaluer la similarité entre les ensembles de tags prédits par le modèle et les tags réels. Nous avons ainsi identifié que le modèle de régression logistique utilisant la matrice USE (Universal Sentence Encoding) a obtenu le meilleur indice de Jaccard, indiquant une meilleure adéquation entre les prédictions du modèle et les vérités terrain.

Run Name	Created	Duration	accuracy	f1_score	jaccard_score	precision	recall
Use_reg_logistic_best_par...	7 days ago	12.2s	0.41085719...	0.62215801...	0.45154525...	0.75656683...	0.52830188...
Use_xgboost_best_params	6 days ago	8.7min	0.41126969...	0.61909620...	0.44832682...	0.82298226...	0.49617384...
Word2Vec_xgboost_best_...	9 days ago	1.4min	0.32901575...	0.52478360...	0.35573330...	0.78204534...	0.39488287...
Bert_reg_logistic_best_par...	9 days ago	53.7s	0.31053543...	0.52209412...	0.35326615...	0.73971722...	0.40341141...
Word2Vec_reg_logistic_b...	9 days ago	4.5s	0.31235046...	0.51396861...	0.34586659...	0.74520031...	0.39225422...
Use_random_forest_best_...	7 days ago	1.7h	0.25740450...	0.43186769...	0.27540258...	0.90379605...	0.28371984...
Bert_xgboost_best_params	7 days ago	21.7min	0.24280174...	0.43175378...	0.27530994...	0.85085910...	0.28926923...
Tfidf_xgboost_best_params	9 days ago	28.8s	0.19924098...	0.37392222...	0.22995346...	0.75717085...	0.24826216...
Tfidf_reg_logistic_best_pa...	9 days ago	5.1s	0.17630558...	0.33742445...	0.20295285...	0.82646322...	0.21198668...
Tfidf_random_forest_best...	9 days ago	6.3min	0.16863295...	0.32771793...	0.19597048...	0.87329286...	0.20170570...
Word2Vec_random_forest...	9 days ago	18.1min	0.17069548...	0.28812402...	0.16830893...	0.87091069...	0.17261522...
Bert_random_forest_best_...	8 days ago	50.3min	0.08052140...	0.16544469...	0.09018245...	0.91426893...	0.09095157...

MLFlow nous permet non seulement de suivre et d'enregistrer le meilleur modèle, mais également d'enregistrer les conditions de ce résultat, y compris les hyperparamètres utilisés, et de sauvegarder le modèle en format pickle, garantissant ainsi sa reproductibilité et sa disponibilité ultérieure.

B) ENREGISTREMENT DU MODELE VIA MLFLOW REGISTRY

Grâce à la fonctionnalité de registry de MLFlow, nous pouvons mettre à disposition et enregistrer la version du modèle, assurant ainsi sa disponibilité tout au long du cycle de machine learning. Cette approche permet également d'effectuer un versionnage du modèle, facilitant sa mise à jour et son déploiement ultérieur en cas de besoin. De plus, le versionnage du modèle permet une traçabilité complète des différentes itérations et des modifications apportées, ce qui est essentiel pour garantir la conformité et la qualité du modèle dans un environnement de production.

3) MISE EN PRODUCTION ET DEPLOIEMENT

Après avoir développé notre modèle de prédiction de tags, l'objectif était de le rendre accessible aux utilisateurs via une interface interactive. Pour ce faire, nous avons élaboré une solution complète, combinant une API Flask côté backend et une interface utilisateur conviviale grâce à Streamlit côté frontend.

Nous avons dû adapter notre modèle pour répondre aux contraintes de l'instance EC2 d'AWS, optant ainsi pour une approche plus légère avec une régression logistique et un encodage Tf-idf. Cependant, nous avons continué à travailler avec notre modèle le plus performant en local (Régression logistique avec encodage USE) pour des tests approfondis. L'idée était de fournir une version initiale de notre application sur AWS tout en prévoyant une évolution future vers une solution plus robuste.

A) TESTS DE L'APPLICATION AVEC PYTEST

Avant de procéder au déploiement, nous avons minutieusement testé notre application pour garantir sa stabilité et son bon fonctionnement. L'utilisation de Pytest nous a permis de créer une suite de tests exhaustive, incluant une configuration environnementale dédiée à cette fin.

```python
import pytest
from Moreno_Bastien_5_code_API_022024 import create_app

@pytest.fixture
def client():
    app = create_app({"TESTING": True})
    with app.test_client() as client:
        yield client
```

Ces tests ont validé le bon fonctionnement de l'application, notamment en vérifiant les réponses aux requêtes utilisateurs.

```python
from conftest import client
import json

def test_should_status_code_ok_get(client):
    response = client.get('/')
    assert response.status_code == 200

def test_should_return_msg(client):
    response = client.get('/')
    data = response.data.decode()
    assert data == "API Flask pour utilisation dans Streamlit, voici l'URL : https://categoriser-automatiquement-des-questions.streamlit.app/"

def test_should_status_code_ok_post(client):
    data = {"text": "Python is very cool !"}
    response = client.post('/predict', json=data)
    assert response.status_code == 200

def test_should_return_predict(client):
    data = {"text": "Python is very cool !"}
    response = client.post('/predict', json=data)
    json_data = json.loads(response.data.decode())
    assert json_data['predicted_tags'] != []
```

```
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
========== 4 passed, 10 warnings in 0.95s ==========
```

B) GESTION DU CODE AVEC GIT ET GITHUB

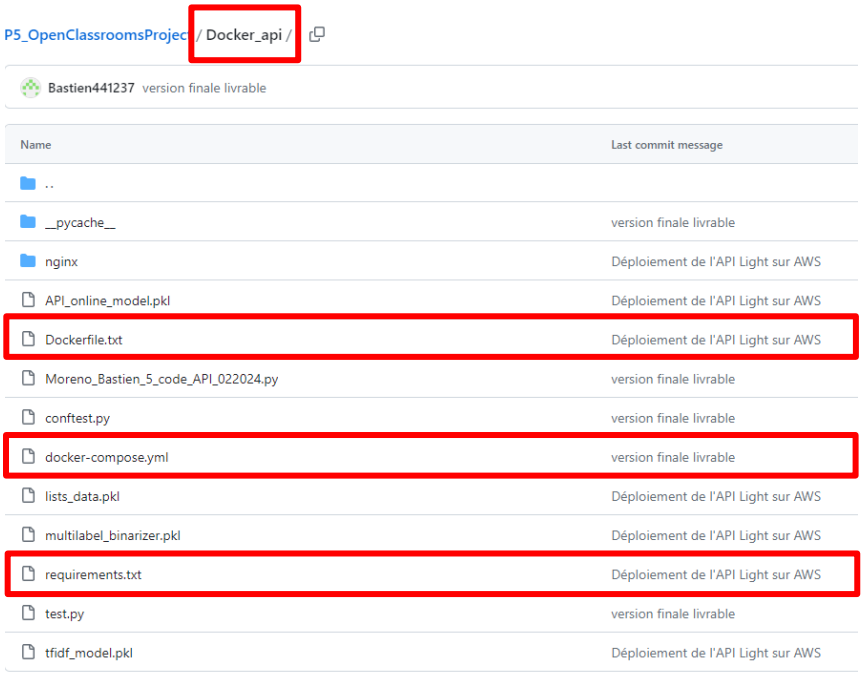
La gestion efficace du code source était primordiale pour assurer un développement fluide et cohérent. Nous avons choisi Git comme système de contrôle de version, permettant de suivre les modifications apportées au code source au fil du temps. Cette approche nous a également permis de travailler sur différentes fonctionnalités de manière isolée grâce aux branches Git, tout en maintenant une cohérence globale du projet.

Github a été utilisé comme plateforme centralisée pour héberger notre code source, offrant un espace de stockage sécurisé afin de pouvoir utiliser le repository sur AWS et Streamlit.

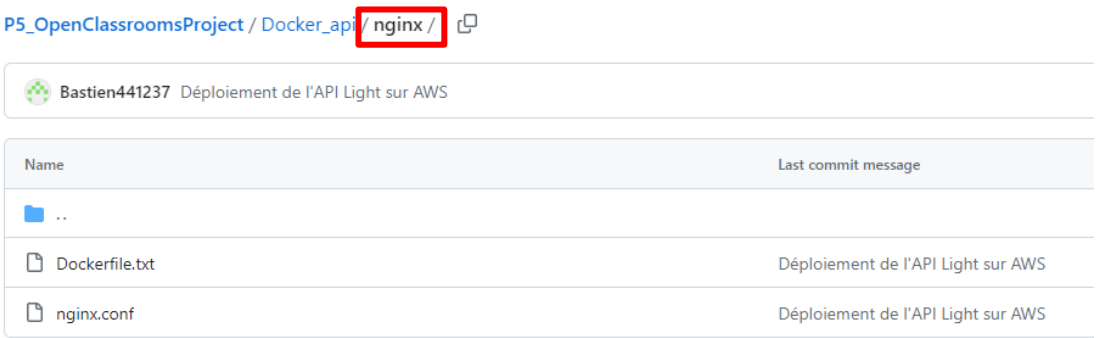
C) DEPLOIEMENT SUR AWS AVEC DOCKER

Une fois le code source disponible sur Github, nous avons procédé au déploiement de l'API Flask sur AWS en utilisant Docker pour la gestion des conteneurs. Cette approche nous a permis de garantir la portabilité de notre application et de simplifier le processus de déploiement en assurant la compatibilité des packages sur un environnement de production.

Pour orchestrer le déploiement, nous avons utilisé un fichier docker-compose.yml. Dans ce fichier, nous avons spécifié les services nécessaires pour notre application, à savoir Flask et NGINX. Dans la section Flask, nous avons défini la construction de l'image Docker à partir du Dockerfile situé dans le répertoire Docker_api, en veillant à installer les dépendances répertoriées dans notre fichier requirements.txt. Le service Flask expose le port 8080, correspondant au port sur lequel notre application Flask écoute à l'intérieur du conteneur.



Le service NGINX, quant à lui, utilise l'image NGINX officielle et copie le fichier de configuration nginx.conf dans le répertoire approprié à l'intérieur du conteneur. Ce fichier de configuration assure la redirection du trafic entrant du port 80 vers le port 8080, où notre application Flask est configurée pour répondre.



En déployant ces services via Docker sur une instance EC2 d'AWS, nous avons pu garantir que notre application Flask était correctement isolée et fonctionnait de manière fiable dans un environnement de production. En outre, l'utilisation de Docker nous a permis de gérer efficacement les dépendances de notre application à l'aide du fichier requirements.txt, assurant ainsi la reproductibilité de notre environnement de développement en production.

Pour permettre le trafic entrant vers notre application, nous avons dû configurer les groupes de sécurité de notre instance EC2. Nous avons ouvert le port 80 pour permettre le trafic HTTP entrant vers NGINX et le port 8080 pour le trafic HTTP entrant vers Flask.

D) UTILISATION DE L'INTERFACE STREAMLIT EN LIGNE

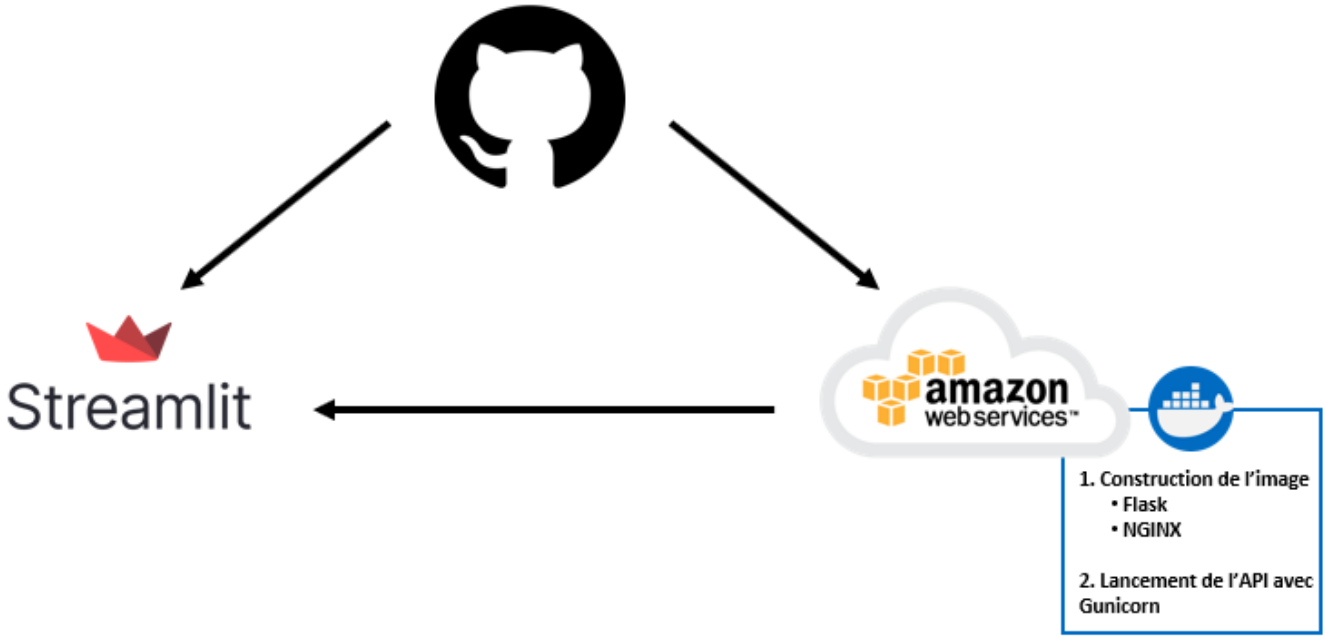
Parallèlement au déploiement de l'API Flask, nous avons également rendu l'interface utilisateur disponible en ligne grâce à Streamlit. Cette plateforme nous a offert une solution simple et efficace pour créer l'application web à partir de scripts Python.

En utilisant la fonctionnalité de Streamlit qui permet de déployer directement l'application à partir d'un fichier Python, nous avons lié l'interface à notre repository Github. Cela nous a permis de configurer l'application en récupérant automatiquement le fichier .py correspondant dans le repository. Ainsi, toute modification apportée au code de l'interface sur Github se reflète automatiquement dans l'application en ligne, assurant une mise à jour facile et rapide de l'interface pour les utilisateurs finaux.

L'interface est disponible à l'adresse : https://categoriser-automatiquement-des-questions.streamlit.app/

E) SCHEMA DU PROCESSUS DE DEPLOIEMENT

Pour une meilleure compréhension du processus de déploiement, voici un schéma simplifié illustrant les différentes étapes :



4) SUIVI DE LA PERFORMANCE ET MONITORING

Dans cette section, nous aborderons les concepts clés liés au suivi de la performance du modèle, ainsi que la mise en place d'un système de stockage d'évènements et de gestion d'alerte en cas de dégradation de la performance.

A) CONCEPT DE SUIVI DE LA PERFORMANCE DU MODELE

Dans le cadre de notre projet, il est essentiel de mettre en place un système de suivi de la performance pour garantir que le modèle continue de fonctionner comme prévu face à l'activité du monde réel. Ce suivi nous permet de détecter tout changement dans le comportement du modèle et de prendre des mesures correctives en cas de dégradation de sa performance.

Nous allons d'abords essayer de résumé les différentes dérives de données que nous pouvons rencontrer :

- **Model Drift** : Le Model Drift se produit lorsque les performances d'un modèle de machine learning déployé en production se dégradent au fil du temps. Cela peut être dû à des changements dans les tendances des données d'entrée. Par exemple, si le modèle est entraîné sur des données qui ne reflètent pas les nouvelles tendances ou les nouveaux comportements des utilisateurs, sa capacité à généraliser efficacement peut être compromise. Pour détecter le Model Drift, nous devons surveiller régulièrement les performances du modèle et comparer ses prédictions avec les résultats attendus.
- **Data Drift** : Le Data Drift se produit lorsque la distribution des données d'entrée utilisées par le modèle change au fil du temps. Cela peut se produire en raison de changements dans les comportements des utilisateurs, dans les sources de données, ou dans l'environnement dans lequel le modèle opère. Si le modèle est entraîné sur des données qui ne reflètent pas la distribution des nouvelles données, sa performance peut diminuer. Pour détecter le Data Drift, nous devons surveiller les changements dans la distribution des données d'entrée et ajuster le modèle en conséquence.
- **Concept Drift** : Le Concept Drift se produit lorsque la relation entre les variables d'entrée et la variable cible du modèle change au fil du temps. Cela peut se produire en raison de changements dans le comportement des utilisateurs, des conditions environnementales, ou des préférences des utilisateurs. Par exemple, dans le cas d'un modèle de prédiction de tags pour des questions Stack Overflow, le concept drift peut se produire si les sujets ou les préoccupations des utilisateurs changent au fil du temps. Pour détecter le Concept Drift, nous devons surveiller les performances du modèle et ajuster ses paramètres en fonction des nouvelles conditions.

B) OUTILS ENVISAGES POUR METTRE EN ŒUVRE LE SYSTEME DE SUIVI

Divers outils de monitoring sont disponibles pour surveiller et maintenir la performance des modèles de machine learning en production. Voici quelques-uns de ces outils :

- **Prometheus** : Outil open-source de surveillance et d'alerte des systèmes. Prometheus collecte et stocke les métriques sous forme de données chronologiques, ce qui permet de suivre l'évolution des performances au fil du temps.
- **Amazon SageMaker Model Monitor** : Intégré à la plateforme Amazon SageMaker, cet outil permet d'automatiser la surveillance des modèles de machine learning en production. Il alerte les utilisateurs en cas de problèmes de qualité des données.
- **Fiddler** : Offrant une interface conviviale et claire, Fiddler permet de surveiller les performances des modèles, d'analyser leur comportement, et de déployer des modèles à grande échelle. Il facilite également la gestion des modèles et des ensembles de données.

- **Hydrosphere Monitoring** : Ce module de la plateforme Hydrosphere permet de surveiller les modèles de machine learning en temps réel. Il offre une visibilité complète sur les performances des modèles déployés, ainsi que sur leur comportement dans des conditions de production.
- **Evidently** : Solution open-source d'analyse des modèles de machine learning, Evidently génère des rapports interactifs à partir de données DataFrame. Il accompagne les utilisateurs tout au long du cycle de vie des modèles, de la phase de développement à la production.

C) PROPOSITION D'UN SYSTEME DE SUIVI DE LA PERFORMANCE

Dans le cadre de notre projet, nous allons présenter la mise en place possible d'un système de suivi de la performance du modèle afin de garantir son bon fonctionnement et de détecter rapidement toute dégradation de ses performances.

Ce système repose sur la collecte d'événements relatifs aux prédictions réalisées par l'API et sur une gestion d'alerte en cas de dégradation significative de la performance.

CONCEPTION DU SYSTEME DE SUIVI DE LA PERFORMANCE

Pour concevoir notre système de suivi de la performance adapté au projet, nous avons identifié les indicateurs et mesures à mettre en œuvre, ainsi que les types d'alerte préconisés.

INDICATEURS ET MESURES :

Nous surveillons en permanence l'évolution de l'indice de Jaccard associé au F1-score, qui mesure la similarité entre les ensembles de tags prédits par le modèle et les tags réels. Cette mesure nous permet d'évaluer la précision du modèle dans ses prédictions.

Nous avons également intégré une mesure du taux de prédictions données par le modèle. Ce taux représente la proportion de questions pour lesquelles le modèle est en mesure de fournir une prédiction de tags. Une baisse significative de ce taux pourrait indiquer un problème de généralisation du modèle ou l'apparition de nouveaux tags non traités.

TYPES D'ALERTES PRECONISEES :

Nous préconisons la mise en place d'alertes pour signaler les dégradations significatives de la performance du modèle. Les alertes peuvent être déclenchées en cas de :

- Détection de model drift, data drift ou concept drift.
- Chute importante des métriques de performance.
- Écart significatif entre les prédictions du modèle et les résultats attendus.
- Baisse du taux de prédictions données par le modèle en dessous d'un seuil prédéfini, indiquant un possible problème de généralisation ou l'apparition de nouveaux tags non traités.

UTILISATION DES OUTILS ENVISAGES POUR METTRE EN ŒUVRE LE SYSTEME DE SUIVI

Pour implémenter notre système de suivi de la performance, nous envisageons d'utiliser des outils spécifiques adaptés à chaque étape du processus.

- **Prometheus** : Cet outil open-source de surveillance et d'alerte des systèmes collecte et stocke les métriques sous forme de données chronologiques. Nous envisageons d'utiliser Prometheus pour suivre l'évolution des performances du modèle au fil du temps.

- **Amazon SageMaker Model Monitor** : Intégré à la plateforme Amazon SageMaker, cet outil automatiserait la surveillance des modèles de machine learning en production, nous alertant en cas de problèmes de qualité des données.
- **Fiddler** : Avec son interface conviviale et claire, Fiddler serait utilisé pour surveiller les performances des modèles, analyser leur comportement et détecter les éventuelles dégradations de performance.
- **Hydrosphere Monitoring** : Ce module de la plateforme Hydrosphere offrirait une visibilité complète sur les performances des modèles déployés, ainsi que sur leur comportement dans des conditions de production.
- **Evidently** : Pour analyser les performances du modèle et générer des rapports interactifs, nous utiliserions Evidently qui accompagne les utilisateurs tout au long du cycle de vie des modèles, de la phase de développement à la production.

En intégrant ces outils dans notre processus de suivi de la performance, nous nous assurerons que notre modèle reste fiable et efficace dans un environnement de production dynamique.

CONCLUSION

En conclusion, cette note technique a présenté une approche méthodique et complète pour la gestion du cycle de vie d'un modèle de machine learning, en mettant l'accent sur l'importance de l'approche MLOps dans le développement, le déploiement et le suivi de la performance du modèle. À travers les différentes sections, nous avons abordé les étapes clés du processus, depuis la collecte des données jusqu'à la surveillance continue en production.

En adoptant une approche itérative et proactive du cycle de vie du modèle, nous nous engageons à maintenir la qualité et la pertinence de notre solution de machine learning à long terme. Nous restons conscients des défis potentiels liés à l'évolution des données et des besoins des utilisateurs, mais nous sommes convaincus que notre approche MLOps nous permettra de relever ces défis avec succès.