Mission - Déployez et monitorez votre modèle de scoring

Comment allez-vous procéder ?

Cette mission simule la mise en production d'un modèle de scoring. Suivez les étapes pour réaliser vos livrables. Avant de démarrer, lisez attentivement la mission, consultez les étapes, et préparez vos questions pour la session de mentorat.

Prêt à mener la mission ?

Vous êtes Data Scientist dans l'entreprise "Prêt à Dépenser". Après avoir développé et versionné un modèle de scoring (Projet Initiez-vous au MLops), vous recevez un message Slack de Chloé Dubois, la Lead Data Scientist :

> " Salut ! Excellents résultats sur la dernière version du modèle de scoring ! Le département 'Crédit Express' est très impatient de l'utiliser pour traiter les nouvelles demandes en quasi temps réel. Il nous faut absolument une API fonctionnelle et déployable (Docker Ready!) d'ici la fin de la semaine prochaine. Peux-tu prioriser ça ? On a aussi besoin d'un dashboard ou rapport de suivi pour vérifier que tout se passe bien une fois en prod (distribution des scores, temps de réponse, ce genre de choses). Tiens-moi au courant de ton plan d'action ! Merci ! "

Vous voila donc chargé de piloter la mise en production effective du modèle de scoring. Cela inclut la création d'une API robuste, la conteneurisation pour un déploiement fluide, et la mise en place d'un monitoring proactif pour garantir la performance et la fiabilité du modèle dans le temps.

En structurant vos pensées et en préparant votre to do list, vous rédigez la liste des livrables que vous allez concevoir et présenter à Chloé :

1. **Un historique des versions** retraçant la construction du projet que vous rendrez disponible dans votre github en consultant la liste des commits.
2. **Des scripts** :
    1. Une **API fonctionnelle** (vous travaillerez probablement avec Gradio ou FastAPI) qui prend les données d'un client en entrée et retourne un score de prédiction.
    2. Des **tests unitaires** automatisés.
3. **Un dockerfile** pour la conteneurisation du code.
4. **Une analyse du Data Drift**:
    1. Un **tableau de bord** ou un rapport de **monitoring** (vous savez que vous pourrez le

etc.)

2. Des **screenshots** de la solution de stockage des données de production.

3. **Un pipeline CI/CD:** un fichier YAML (ou équivalent) démontrant l'automatisation de la mise en production et des tests lors d'un push sur la branche principale (à minima) du projet.
4. Une **documentation README** expliquant comment lancer l'API et interpréter le monitoring.

Dans ce projet, vous vous appuierez sur les livrables que vous avez réalisés lors du projet précédent intitulé *Initiez-vous au MLOps* (partie 1/2).Plus précisément, il s'agit de reprendre le modèle de scoring que vous avez développé, versionné et évalué précédemment avec MLflow. Ce modèle constitue désormais la base sur laquelle vous allez travailler pour le déployer en production.Vous devrez donc réutiliser les artefacts produits, les adapter si nécessaire, et construire autour un environnement complet de déploiement.

De plus, nous vous suggérons de travailler avec les deux outils présentés dans les ressources pédagogiques de ce projet actuel : Streamlit et Gradio. Vous êtes néanmoins libre de travailler avec d'autres outils si vous le souhaitez mais vous penserez à expliquer vos choix techniques pendant votre soutenance avec l'évaluateur.

Vous voilà prêt à travailler !

Cette mission est entièrement guidée.
Suivez les étapes ci-dessous.

Étapes

Étape 1 - Mettez en place le contrôle de version et le dépôt

Initialisez un dépôt Git pour votre projet. Structurez votre projet de manière claire (code source, tests, notebooks, Dockerfile, requirements, etc.). Ajoutez les fichiers de votre modèle au dépôt avec des commits explicites, le code de votre modèle, les scripts d'inférence, les notebooks d'analyse et la documentation initiale. Poussez votre code sur une plateforme distante comme GitHub.

Prérequis:

- Avoir installé Git.
- Avoir créé un compte sur une plateforme comme GitHub, GitLab ou Bitbucket.

Résultats attendus :

- Un lien vers un dépôt Git public (ex: GitHub) contenant le code du projet structuré.
- Un historique de commits clair et pertinent.

Recommandations:

- Utiliser des messages de commit explicites.
- Adopter une stratégie de branche si nécessaire.
- Utiliser un fichier `.gitignore` .

- Ne committez jamais de données sensibles.
- Assurez-vous que le dépôt est bien public.

Outils:

- Git.
- GitHub/GitLab/Bitbucket.

Ressources:

- Documentation Git.
- Quickstart GitHub.
- Cours - Gérez du code avec Git et GitHub

Étape 2 - Déployez le modèle via une API et automatisez avec CI/CD

Description: Développez une API (Gradio, FastAPI) pour exposer votre modèle. L'API doit recevoir des données d'entrée et retourner une prédiction. Conteneurisez cette API avec Docker. Ensuite, créez un pipeline d'Intégration Continue et de Déploiement Continu (CI/CD) (ex: GitHub Actions). Ce pipeline devra automatiquement :

1. Exécuter des tests (unitaires, intégration) sur votre code API et modèle.
2. Construire l'image Docker de l'API si les tests sont concluants.
3. Déployer l'image conteneurisée sur un environnement cible (simulé ou réel).

Prérequis:

- Avoir le code versionné sur une plateforme supportant la CI/CD.
- Avoir choisi un framework d'API.
- Avoir installé Docker.

Résultats attendus :

- Un code source fonctionnel pour l'API.
- Un `Dockerfile` pour créer une image Docker de l'API.
- Un pipeline CI/CD fonctionnel et automatisé visible sur la plateforme, qui déploie l'API.
- Des tests automatisés intégrés au pipeline.

Recommandations:

- Commencez par une API simple et un pipeline basique, puis itérez.
- Incluez une gestion des erreurs dans l'API et documentez-la (ex: Swagger).
- Séparez les étapes de build, test et déploiement dans le pipeline CI/CD.
- Utilisez des secrets pour gérer les credentials.
- Utilisez Hugging Face Spaces qui est particulièrement simple d'utilisation pour ce genre de déploiement.

Points de vigilance:

- Assurez-vous que les tests sont fiables et couvrent les cas critiques, par exemple
- des **entrées avec des données manquantes** pour des champs obligatoires,

- ou des **types de données incorrects** (ex: du texte là où un chiffre est attendu).
- Sécurisez l'API et le pipeline (gestion des secrets, validation d'entrée).
- Gérez correctement le chargement du modèle dans l'API.
  - Lorsque vous intégrez un **modèle de machine learning** dans une **API**, il est crucial de **ne pas charger le modèle à chaque requête**. Cela entraînerait des lenteurs importantes voire un échec sous charge. **Chargez le modèle une seule fois**, au moment du démarrage de l'API, puis **réutilisez le** dans toutes les requêtes.
  - Cela permet de :
- Réduire le temps de réponse de l'API.
- Éviter une surcharge mémoire.
- Améliorer la scalabilité.
- Vérifiez que l'environnement de déploiement dispose des ressources nécessaires.

Outils:

- Gradio/FastAPI
- Docker
- Postman/curl
- GitHub Actions/GitLab CI/Jenkins
- Pytest
- Plateformes de déploiement (Hugging Face, Heroku, Google Cloud Run...).

Ressources:

- Documentation FastAPI.
- Documentation Gradio.
- Docker.
- tutoriels sur les tests.
- Github actions.

Étape 3 - Implémentez le stockage et l'analyse des données de production

Description: Concevez et mettez en place une solution pour stocker les données pertinentes générées par votre API en production : logs d'appels, inputs, outputs, et temps d'exécution (à minima). Mettez en œuvre une analyse automatique de ces données pour détecter des anomalies, notamment la dérive des données (data drift), et des problèmes opérationnels (taux d'erreur, latence anormale).

> Un prototype (PoC) de cette solution peut être réalisé entièrement en local si vous n'avez pas de service cloud à votre disposition. Tant que tous les aspects requis pour cette partie sont correctement adressés, cela est suffisant. Exemple: les logs peuvent être générés par l'API (cloud) puis téléchargés, stockés et analysés localement. Les données collectées par l'API doivent permettre une analyse ultérieure du drift : assurez-vous de stocker les inputs/outputs du modèle et les métriques clés.

- Avoir déployé l'API via le pipeline CI/CD.
- Avoir identifié les données clés à logger depuis l'API et l'infrastructure.

Résultat attendu:

- Une solution de stockage des données de production décrite et/ou implémentée.
- Un script ou notebook réalisant l'analyse automatique des données stockées (détection de drift, anomalies).
- Une présentation de l'étude sur la dérive des données et les points de vigilances résultants.

Recommandations:

- Configurez le logging structuré (ex: JSON) dans votre API.
- Utilisez des bibliothèques dédiées à la détection de drift (ex: Evidently AI, NannyML).
- Pensez à visualiser les résultats de l'analyse (ex: dashboard).

Points de vigilance:

- Soyez conscient des contraintes de stockage et de coût.
- Assurez la conformité RGPD si nécessaire.
- La détection de drift nécessite une référence (données d'entraînement ou fenêtre stable (vous pouvez reprendre votre travail réalisé lors du projet Initiez-vous au MLOps partie 1).

Outils:

- Bibliothèques de logging Python,
- Analyse des logs: Fluentd, Logstash
- Bases de données: Elasticsearch, PostgreSQL
- Bibliothèques de détection de drift: Evidently AI, NannyML
- Outils de visualisation : Grafana, Kibana, Dash/Streamlit.

Ressources:

- Article sur le monitoring ML en python.
- Documentation Evidently.

Étape 4 - Analysez et optimisez les performances du modèle

Description: Maintenant que le modèle est déployé et monitoré, analysez ses performances réelles ou simulées en production.

- Utilisez les données de monitoring (temps d'inférence, latence, utilisation CPU / GPU) et des outils de profiling pour identifier les goulots d'étranglement.
- Testez des stratégies d'optimisation (quantification, optimisation de code, hardware) pour améliorer le temps d'inférence/réponse.
- Intégrez la version optimisée dans votre dépôt et laissez le pipeline CI/CD la déployer.
- Documentez les optimisations et leurs résultats.

- Avoir l'API déployée et un système de monitoring/logging en place (même basique).

Résultats attendus :

- Un rapport détaillant les tests d'optimisation effectués post-déploiement, les résultats et les goulots d'étranglement identifiés.
- Une version optimisée du modèle déployée via le pipeline CI/CD.
- Une justification de la configuration finale (librairies, software, hardware).
- L'amélioration du temps d'inférence et de réponse est démontrée.

Recommandations:

- Baser vos hypothèses d'optimisation sur les données de monitoring réelles.
- Documenter rigoureusement l'impact des optimisations sur la performance et la précision.

Points de vigilance:

- Assurez-vous que les optimisations n'introduisent pas de régressions (précision, biais).
- Validez la compatibilité des optimisations avec l'environnement de production.

Outils:

- Outils de profiling (ex: cProfile).
- Bibliothèques d'optimisation (ex: ONNX Runtime).

Ressources:

- Documentation cProfile.
- ONNX Runtime Home.

Vérifiez votre travail et faites le point avec votre mentor

Vérifiez votre travail et faites le point avec votre mentor

Pour vérifier que vous n'avez rien oublié dans la réalisation de votre exercice, téléchargez et complétez la fiche d'autoévaluation.

Parlez-en avec votre mentor durant votre dernière session de mentorat.

Avez-vous une suggestion pour nous ?

Précédent Suivant