Rapport de projet :
prédire la conversion des leads

Introduction

Ce rapport présente les résultats de notre projet, qui visait à développer un modèle de Machine Learning de classification binaire pour prédire les conversions de prospects en clients. De plus, nous proposons une méthode pratique pour déployer les prédictions du modèle dans un environnement de production cloud.

Choix techniques

Prétraitement des données
Avant le déploiement du modèle, le prétraitement des données est une étape cruciale. Nous avons développé un script Python pour le prétraitement des données, qui transforme les données, gère les valeurs manquantes, effectue un encodage one-hot sur les variables catégorielles et normalise les données. Ces étapes garantissent que les données d'entrée fournies au modèle respectent les exigences de format nécessaires. De plus, pour garantir la qualité des données, nous calculons la différence de temps en jours entre "effective_start_date" et "submitted_at".

Sélection du modèle
Pour la phase actuelle du projet, nous avons choisi le RandomForestClassifier de scikit-learn comme modèle de Machine Learning préféré. Le RandomForestClassifier est un algorithme d'apprentissage ensembliste connu pour son efficacité dans les tâches de classification et de régression.

Containerization avec Docker
L'adoption de Docker container est un choix technique important. Docker offre un environnement isolé pour l'application, garantissant des performances cohérentes sur différentes plates-formes et environnements. Cela simplifie le déploiement et élimine le problème "ça fonctionne sur ma machine".

Déploiement de l'API Flask
Pour servir notre modèle de classification binaire, nous avons décidé de déployer une API Flask. Flask, un framework web micro léger et polyvalent pour Python, a été sélectionné pour sa capacité à créer rapidement et efficacement un service web pour le déploiement

de modèles. Cette API Flask fournit une interface conviviale pour notre modèle de classification binaire, améliorant l'accessibilité et l'utilité.

Architecture cloud

1. Scalability de la solution et ses limites :

La solution est scalable, et pour améliorer la scalability, nous pouvons utiliser une plateforme serverless et d'orchestration de container pour allouer automatiquement des ressources en fonction de la demande. Cependant, la scalability peut être limitée par la disponibilité des ressources et les considérations de coût.
Le transfert de grands datasets à partir d'un data warehouse peut créer des goulots d'étranglement, et l'expansion à travers des régions ou des centres de données cloud peut introduire une latence réseau et des complexités supplémentaires. Pour améliorer la scalability, nous pouvons également mettre en œuvre l'intégration de l'entrepôt de données.

2. Idée architecturale proposée :

Notre proposition pour une configuration basée sur le cloud comprend les composants suivants :

- **Ingestion de données :** Utiliser des processus ETL (Extract, Transform, Load) pour extraire les données de data warehouse, les transformer pour l'entraînement et les charger dans un système de stockage (par exemple, un stockage cloud comme Amazon S3 ou Google Cloud Storage). Des technologies telles qu'Apache Airflow, AWS Glue ou des scripts personnalisés peuvent être utilisées.

- **Entraînement et mise à jour des modèles :** Utilisez des services basés sur le cloud pour l'entraînement distribuée de grands modèles, tout en automatisant le processus d'entraînement des modèles à l'aide des données nouvellement ingérées. Les technologies incluent AWS SageMaker, Google AI Platform, les pipelines CI/CD, Kubeflow ou des scripts personnalisés.

- **Versioning du modèle :** Sérialiser et versionner les modèles entraînés pour un déploiement et un retour en arrière faciles à l'aide de technologies telles que MLflow, Kubeflow ou des solutions personnalisées.

- **Deployment de modèle et passerelle API :** Déployez le modèle en tant qu'API, conteneur Docker ou fonction serverless sur le cloud à l'aide de technologies telles qu'AWS Lambda, Google Cloud Run ou Kubernetes, et exposez le modèle via une passerelle API pour une consommation plus aisée à l'aide de technologies telles qu'AWS API Gateway, Google Cloud Endpoints ou des API REST personnalisées.

Pour flexibilité et scalability, le déploiement devrait s'exécuter dans des Docker containers.
- **Déclencheur de mise à jour :** Mettre en œuvre un mécanisme de déclenchement qui détecte les nouvelles données dans le data warehouse. Les technologies incluent des fonctions serverless comme AWS Lambda.

3. Points dans une approche FinOps :

En FinOps, l'accent est mis sur la surveillance des coûts cloud, l'optimisation de l'utilisation des ressources, le respect des budgets et l'identification des opportunités d'économies de coûts sans compromettre les performances.

4. Modifications pour un temps de réponse en millisecondes :

Pour atteindre un temps de réponse en millisecondes, il faudrait optimiser pour la vitesse. considérez les améliorations suivantes :

- **Optimisation du modèle :** Utiliser des modèles légers qui s'exécutent plus rapidement.
- **Load Balancing :** Mettre en œuvre l'équilibrage de charge pour distribuer les demandes de manière égale.

Cette vue d'ensemble simplifiée met en évidence les considérations clés liées à la scalability, à la conception architecturale, aux opérations financières et à l'obtention de temps de réponse plus rapides. Des discussions techniques détaillées peuvent avoir lieu pendant la phase de mise en œuvre.

Citations:
[1] https://cloud.google.com/api-gateway
[2] https://www.qlik.com/us/cloud-data-migration/cloud-data-warehouse
[3] https://cloud.google.com/learn/what-is-a-data-warehouse
[4] https://aws.amazon.com/redshift/
[5] https://www.ibm.com/blog/api-gateway/
[6] https://technologymagazine.com/company-reports/infor-os-platform-leveraging-api-gateway-and-data-unlock-human-potential