Cahier des Charges Projet MLOps

Patricia Wintrebert
Mimoune Louarradi

11/01/2024

Organisme : DataScientest

1 Contexte et Objectifs

Ce projet s'inscrit dans une démarche d'optimisation de l'organisation des forces de l'ordre de la ville de Chicago, notamment en étant capable de prédire des pics de criminalité. Pour une grande ville comme Chicago, il est pertinent de pouvoir prévenir si cela est possible l'apparition de crimes et de surcroît renforcer les équipes lorsque cela est nécessaire, ce qui améliore les délais d'intervention.

La ville de Chicago souhaite la mise en place d'une application évolutive et qui analyse et prédit les données en temps réel. Cette API lui permettra d'améliorer son organisation, sa prise de décision et à optimiser ses opérations et ses ressources humaines. Pour la réalisation de cette API, la ville met à disposition des ressources, c'est-à-dire un dataset contenant le rapport des crimes survenus dans la ville depuis 2001. Ce dataset est mis à jour régulièrement.

Les utilisateurs principaux sont tous utilisateurs finaux ayant les droits d'accès à l'application (Forces de l'ordre, administration etc.)

L'administration et la maintenance de l'application sera effectuée par l'équipe technique en charge de la ville de Chicago. Ceux-ci auront la responsabilité de maintenir à jour, déployer, surveiller les performances et améliorer si besoin l'API pour assurer son fonctionnement optimal.

L'API sera hébergée sur une infrastructure cloud, assurant ainsi flexibilité, haute disponibilité et adaptabilité en fonction de la demande. Cette configuration cloud favorise également une harmonisation aisée avec les systèmes et services déjà en place.

Pour une interaction directe, les utilisateurs finaux pourront accéder à l'API via une plateforme web dotée d'une interface utilisateur conviviale. Par ailleurs, pour une intégration plus poussée et une automatisation avancée, les développeurs et les administrateurs bénéficieront d'un accès direct à l'API, permettant une synergie fluide avec d'autres outils et processus existants.

Pour faire face à cette problématique, nous envisageons la mise en place d'une API spécifiquement conçue pour la prédiction des événements criminels. En exploitant les technologies de pointe et en l'hébergeant sur une infrastructure cloud, nous assurons une solution flexible et hautement disponible. Cette API offrira la capacité de réaliser des prédictions en temps réel en se basant sur des données existantes.

2 Modèle

2.1 Collecte des données

La collecte des données sur les crimes est organisée de manière, où chaque incident est classifié non seulement par région, mais aussi chronologiquement par mois. Cette approche transforme le nombre de crimes en une série temporelle, permettant ainsi une analyse des tendances et des modèles dans le temps.

2.2 Types de crime

- DECEPTIVE PRACTICE
- OTHER OFFENSE
- THEFT
- ...

2.3 Type de région

- Rogers Park
- West Ridge
- Uptown
- ...

2.4 Modélisation et Prédiction

Pour anticiper et comprendre l'évolution future de ces tendances criminelles, nous avons intégré l'utilisation d'un outil analytique et de prédiction des séries temporelles Prophet, accessible via PyPI. Le package Prophet intègre et traite les variations saisonnières ainsi que les impacts des jours fériés, ce qui rend ses prédictions particulièrement robustes et fiables. Ces prédictions sont essentielles pour la mise en place d'une stratégie de planification efficace et pour une allocation optimale des ressources dans le cadre de la lutte contre la criminalité.

2.5 Code et Implémentation

Le cœur de notre modèle est incarné par la classe `ChicagoCrimePredictor`, structurée comme suit : `ChicagoCrimePredictor(months_pred=12, data_dir=DATA_DIR)`.

Cette classe est conçue pour être flexible et adaptative, avec deux arguments principaux lors de l'initialisation :

- `months_pred`: Cet argument détermine la période de prédiction, permettant aux utilisateurs de spécifier le nombre de mois pour lesquels ils souhaitent obtenir des prévisions.
- `data_dir`: Il s'agit du répertoire contenant les données brutes, permettant au modèle d'accéder et de traiter l'ensemble des informations nécessaires.

2.6 Exemple d'application

Dans le cas d'étude présenté, le modèle est entraîné sur les données collectées depuis l'année 2001 jusqu'en décembre 2022. À partir de ces données, des prédictions sont générées pour une durée d'un an, couvrant la période de janvier 2023 à décembre 2023. Cette approche permet non seulement de comprendre les tendances passées mais aussi d'anticiper avec une certaine précision les évolutions futures dans le domaine de la criminalité.

Focus sur la Criminalité dans la Région d'Austin: Pour cette étude, nous avons choisi de concentrer nos efforts sur la prédiction des crimes de type "ASSAULT" dans la région spécifique d'Austin. Cette focalisation est guidée par l'importance croissante de comprendre et de prévenir les actes de violence dans cette zone. Notre modèle, grâce à la méthode `return_data`, est configurable et peut être adapté selon différents paramètres, notamment:

- `type_incident` : Permet de spécifier le type de crime à analyser, dans notre cas "ASSAULT".
- `community_area` : Cible une région spécifique pour la prédiction, ici la région d'Austin.

2.7 Métriques d'évaluation

2.7.1 Mean Absolute Error (MAE)

Formule:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Description : Le MAE mesure la différence moyenne entre les valeurs observées (réelles) et les valeurs prédites par le modèle. Pour le calculer, on prend la différence absolue (sans tenir compte du signe) entre chaque paire de valeurs observée et prédite, et ensuite on calcule la moyenne de ces différences.

Interprétation : Un MAE plus petit indique une meilleure performance du modèle. C'est une mesure facile à comprendre car elle est exprimée dans les mêmes unités que les données observées. Par contre, comme elle utilise la valeur absolue, elle ne tient pas compte de la direction des erreurs (sous-estimation ou sur-estimation).

2.7.2 Root Mean Squared Error (RMSE)

Formule:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Description : Le RMSE est similaire au MAE mais donne plus de poids aux erreurs plus grandes. On calcule d'abord la différence entre chaque valeur observée et prédite, on élève ces différences au carré, puis on en calcule la moyenne. Finalement, on prend la racine carrée de cette moyenne.

Interprétation : Un RMSE plus faible signifie une meilleure précision du modèle. Le RMSE est plus sensible aux erreurs importantes, ce qui peut être un avantage dans certains contextes où ces erreurs sont plus problématiques. Comme le MAE, le RMSE est exprimé dans les mêmes unités que les valeurs observées.

2.7.3 R2 (Coefficient de détermination)

Formule:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Description : Le R2 mesure la proportion de la variance des valeurs observées qui est expliquée par le modèle. Il est calculé en comparant la variance des erreurs du modèle avec la variance totale des valeurs observées.

Interprétation : Un R2 proche de 1 indique que le modèle explique une grande partie de la variance des données, tandis qu'un R2 proche de 0 indique que le modèle n'explique pas bien la variance des données. Le R2 est une mesure utile pour comparer différents modèles entre eux, mais il ne doit pas être le seul critère de jugement, surtout si les données ne suivent pas une distribution normale ou si le modèle est complexe.

2.8 Résultats d'évaluation

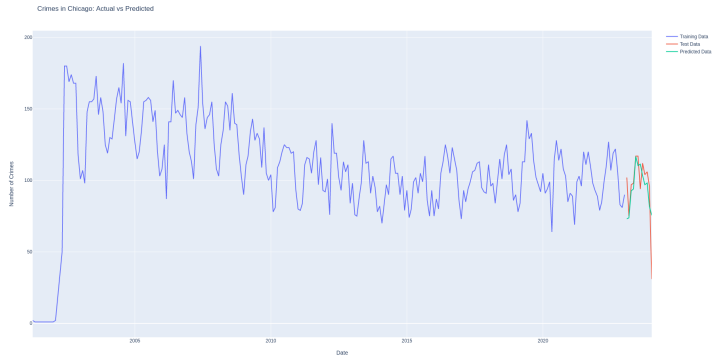
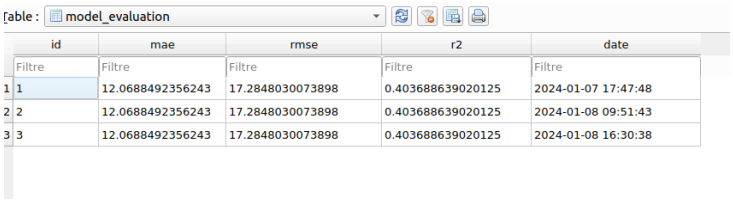


Figure 1: Résultats de l'évaluation pour les crimes de type ASSAULT dans la région d'Austin.

Table 1: Métriques du modèle pour les crimes de type ASSAULT dans la région d'Austin

Métrique	Valeur
Mean Absolute Error (MAE)	12.06
Root Mean Squared Error (RMSE)	17.28
R^2	0.4

Une base de données SQLite a été créée, permettant ainsi l'enregistrement des métriques pour chaque exécution.



	id	mae	rmse	r2	date
	Filtre	Filtre	Filtre	Filtre	Filtre
1	1	12.0688492356243	17.2848030073898	0.403688639020125	2024-01-07 17:47:48
2	2	12.0688492356243	17.2848030073898	0.403688639020125	2024-01-08 09:51:43
3	3	12.0688492356243	17.2848030073898	0.403688639020125	2024-01-08 16:30:38

Figure 2: Résultats de l'évaluation : capture base de donnée Sqlite

3 API

- **Authentification** : Ce point d'accès est crucial pour garantir la sécurité et l'intégrité de l'API. Il permet l'authentification des utilisateurs et administrateurs, en utilisant la méthode d'authentification jetons OAuth. Cette étape assure que seuls les utilisateurs autorisés peuvent accéder aux fonctionnalités de l'API.

- **Prédiction** : Ce endpoint joue un rôle central dans l'application en permettant aux utilisateurs d'obtenir des prédictions sur les crimes. Il utilise des modèles d'apprentissage automatique en tant que séries temporelles et de traitement de données pour analyser et interpréter les tendances des crimes.

- **Evaluation** : Ce point d'accès fournit des métriques essentielles sur la performance du modèle de prédiction. Il offre des statistiques telles que l'Erreur Moyenne Absolue (MAE), l'Erreur Quadratique Moyenne (MSE), et le coefficient de détermination (R2). Ces métriques aident à évaluer l'exactitude et la fiabilité du modèle de prédiction.

- **Réentraînement** : Introduit comme une nouvelle fonctionnalité, ce endpoint permet de réentraîner le modèle de prédiction automatiquement lorsque les métriques de performance tombent en dessous d'un seuil prédéfini. Ce processus assure que le modèle reste performant et précis, en s'adaptant aux nouvelles tendances et données.

- **Mise à jour des données** : Ce point d'accès est essentiel pour maintenir la pertinence du modèle de prédiction. Il permet une mise à jour régulière et automatique des données utilisées par le modèle, en intégrant les dernières informations relatives aux crimes. Cette mise à jour

garantit que le modèle reste efficace et adapté à l'évolution constante des schémas de criminalité.

4 Testing & Monitoring

Lors de chaque mise à jour de la branche master du repository, un workflow lance automatiquement différents tests :

1. `test_auth.py`: Les tests de ce script utilisent la bibliothèque TestClient et aident à s'assurer que l'application FastAPI fonctionne correctement en vérifiant diverses fonctionnalités telles que l'authentification, la génération de tokens, la vérification de mots de passe et l'accès aux données sécurisées.

2. `test_verify_password()` : Ce test vérifie la fonction verify_password en utilisant une comparaison avec des mots de passe hashés. Il vérifie si la fonction verify_password renvoie True lorsque le mot de passe fourni correspond au mot de passe hashé, et False lorsque le mot de passe fourni est incorrect.

3. `test_generate_token()` : Ce test vérifie la fonction generate_token. Il génère un token en utilisant un nom d'utilisateur fictif et vérifie si le résultat est une chaîne de caractères non vide (une chaîne de token valide).

4. `test_login_success()` : Ce test vérifie la route /login lorsque la tentative de connexion est réussie. Il envoie une requête POST avec un nom d'utilisateur et un mot de passe valides et vérifie si la réponse renvoie un code de statut HTTP 200 (OK) et si un jeton d'accès (access_token) est présent dans la réponse JSON.

5. `test_login_failure()` : Ce test vérifie la route /login lorsque la tentative de connexion échoue en envoyant un mot de passe incorrect. Il envoie une requête POST avec un nom d'utilisateur valide mais un mot de passe incorrect, puis vérifie si la réponse renvoie un code de statut HTTP 401 (Unauthorized) et si aucun jeton d'accès n'est présent dans la réponse JSON.

6. `test_secure_data_with_token()` : Ce test vérifie la route /secure-data en utilisant un jeton d'accès valide. Il génère un jeton d'accès pour un utilisateur fictif, ajoute ce jeton aux en-têtes de la requête GET, puis vérifie si la réponse renvoie un code de statut HTTP 200 (OK) et si le message dans la réponse JSON indique que les données sont sécurisées pour l'utilisateur spécifié.

7. `test_secure_data_without_token()` : Ce test vérifie la route /secure-data sans fournir de jeton d'accès. Il envoie une requête GET sans jeton d'accès dans les en-têtes et vérifie si la réponse renvoie un code de statut HTTP 401 (Unauthorized), ce qui signifie que l'accès aux données sécurisées est refusé en l'absence de jeton d'accès.

8. `test.bdd.py` : Ce test utilise la bibliothèque requests pour effectuer une requête GET vers une URL spécifique et vérifier si l'API répond correctement:

    - `test_api_url()` : Ce test vise à vérifier si l'URL de l'API réagit correctement. Il crée une variable url contenant l'URL de l'API cible, qui est "https://data.cityofchicago.org/resource/ijzp-q8t2.json" dans ce cas. Ensuite, il envoie une requête GET à l'URL à l'aide de requests.get(url). Le test vérifie ensuite si la réponse de la requête renvoie un code de statut HTTP 200 (OK) en utilisant l'assertion assert response.status_code == 200.

9. `test_chicago_crime_predictor.py` : Ce script contient une série de tests unitaires pour la classe ChicagoCrimePredictor, qui est une classe utilisée pour prédire les taux de criminalité à Chicago:

    - `test_init()`: Ce test vérifie le constructeur de la classe ChicagoCrimePredictor. Il crée une instance de ChicagoCrimePredictor avec des paramètres spécifiques (6 mois de prévision et un répertoire de données 'data') et vérifie que les attributs de l'instance sont corrects, notamment _month_pred est égal à 6, data_dir est un objet Path pointant vers 'data', et que l'attribut model est initialisé à None.
    - `@fixture mock_df_crime` et `@fixture mock_df_socio`: Ces décorateurs de fixture définissent deux DataFrames fictifs (mock_df_crime

et mock_df_socio) qui serviront de données simulées pour les tests.

- `test_load_df_crimes()`: Ce test vérifie la méthode load_df_crimes() de la classe ChicagoCrimePredictor. Il utilise mocker.patch pour simuler l'appel à pandas.read_csv et vérifie que la méthode retourne un DataFrame qui correspond à mock_df_crime.
- `test_load_df_socio()`: Ce test vérifie la méthode load_df_socio() de la classe ChicagoCrimePredictor. Il utilise également mocker.patch pour simuler l'appel à pandas.read_csv et vérifie que la méthode retourne un DataFrame qui correspond à mock_df_socio.
- `test_return_data()`: Ce test vérifie la méthode return_data() de la classe ChicagoCrimePredictor. Il utilise des simulations pour charger des DataFrames factices et appelle ensuite la méthode pour obtenir les données d'entraînement et de test. Il vérifie que les données retournées sont des objets DataFrame.
- `test_model_train()`: Ce test vérifie la méthode model_train() de la classe ChicagoCrimePredictor. Il utilise mocker.patch pour simuler l'appel à la méthode fit de la classe Prophet (un modèle de séries temporelles) et vérifie que la méthode model_train() attribue un modèle Prophet à l'instance de predictor.
- `test_model_predict()`: Ce test vérifie la méthode model_predict() de la classe ChicagoCrimePredictor. Il simule l'utilisation d'un modèle pré-entraîné, charge ce modèle, attribue le modèle à l'instance de predictor et vérifie que la méthode model_predict() renvoie des prédictions non vides.

5 Schéma d'implémentation