Projet Accidents

Modèle prédictif d’aide à l’évaluation de la gravité d’un accident

DataScientest

LUDOVIC CALMETTES

NADJIDINE BOINALI

ELKHALIL BETROUNI

CYRILLE QUESSON

TABLE DES MATIERES

1. Contexte et Objectifs

1.1. Mise en situation :

Nous souhaitons proposer un logiciel qui aide à estimer la gravité d'un accident, en fonction des premières observations réalisées sur site. (Caractéristiques du lieu d'accident, des véhicules impliqués, du type de choc …).

Cette prédiction de gravité pourra permettre de définir plus précisément les besoins en ressources pour cet accident. De plus, il sera possible d'anticiper en contactant à l'avance les hôpitaux les plus proches pour les avertir de l'arrivée d'un cas grave.

En faisant correspondre les ressources envoyées aux besoins réels, cela permet également de libérer davantage de ressources pour d'autres urgences qui peuvent survenir en même temps.

Les observations pourront provenir de différentes sources :

- Systèmes des véhicules (GPS, détection d'impact…)
- Appels téléphoniques aux services de secours
- Caméra de surveillance des routes
- Etc…

Ces informations pourront subir un prétraitement pour déterminer d'autres caractéristiques déduites.

Le logiciel fournira une prédiction de gravité en se basant sur un algorithme ML:

La gravité d'un accident correspondra à l'état de la personne la plus sévèrement blessée.

- Classe 0 : Indemnes ou Blessés légers (sans hospitalisation)
- Classe 1 : Blessés à hospitaliser ou Tués

Ce qui permettra par exemple une Assistance à la prise de décision

- Si Classe 0 : Envoi de police secours seule pour sécurisation
- Si Classe 1 : Envoi de police secours pour la sécurisation + SAMU pour les premiers secours

1.2. Principe de fonctionnement

Étape	Tâche	Acteur
1	Recueil et prétraitement des informations	Centre d'appel des secours (112)
2	Saisie dans l'application	API
3	Réalisation d'une prédiction	Modèle
4	Transmission de la prédiction	API
5	Utilisation de la prédiction	Centre d'appel des secours

1.3. Utilisateurs :

Commanditaires : Services de sécurité routière, Service des Urgences.

Utilisateurs : Opérateurs des services d'urgences et de sécurité routière

2. Fonctionnement du Modèle

Fonctionnement

Le modèle analyse les caractéristiques d'un accident (28 variables d'entrées possibles) pour en prédire la gravité (variable cible)

- Classe 1 : Blessés & Tués
- Classe 0 : Indemnes ou blessés légers

Adaptations possibles :

Les caractéristiques pouvant être recueillies rapidement par les services de secours ne comporteront pas forcément les 28 variables actuellement utilisées. Le modèle de ML sera revu et adapté en fonction des caractéristiques réellement disponibles (définies par le commanditaire).

Type de modèle : Random Forest

Différents modèles ont étés testés, et on a retenu un algorithme d'apprentissage Random Forest qui est le plus performant sur la base de données actuelle.

Définition des métriques: f1-score

Dans le cas de la détection de gravité, les faux négatifs (ne pas détecter un accident prioritaire) sont souvent plus critiques que les faux positifs (fausse alarme). On veut donc non seulement avoir une bonne précision (accuracy), mais surtout un bon rappel (recall) pour avoir peu de faux négatifs.

On choisit donc comme métrique le f1-score qui intègre bien ces deux contraintes.

Pour information, un exemple des valeurs pour les accidents de 2021 :

Gravité	Nombre d'accidents	Proportion
Non prioritaire (Indemnes ou blessés légers)	36 965	65,40 %
Prioritaire (Tués ou nécessitant une hospitalisation)	19 553	34,59 %
TOTAL	56 518	

Sur ces données de 2021, le f1-score obtenu par le modèle est de 0,61.

Entrainement du modèle

Le modèle sera ré entrainé et évalué:

- A chaque modification du code (github Action)
- Tous les mois pour tenir compte des modifications de la Base de Donnée Source

3. Base de données

Accidents :

- Base de données annuelles des accidents corporels de la circulation routière fournie par l'Observatoire national interministériel de la Sécurité routière (ONISR):
- https://www.data.gouv.fr/en/datasets/bases-de-donnees-annuelles-des-accidents-corporels-de-la-circulation-routiere-annees-de-2005-a-2022/

Utilisateurs : users.json et admins.json

Logs : A faire

Mise à jour de la Base de Donnée "accidents":

- Par rechargement de la base de données source fournie par ONISR. Une fois par mois.
- Par ajout manuel d'une entrée "accident"

Mise à jour de la Base de Donnée "utilisateurs":

- Via l'API

4. Rôle de l'API

Fonctionnalités

- Utilisation du modèle pour prédire la gravité des accidents (Classe 0 ou 1)
- Enregistrement des prédictions dans des logs.
- Authentification et gestion des nouveaux utilisateurs et administrateurs

Endpoints

1. /status : vérification du fonctionnement de l'API
2. /new_user : inscription d'un utilisateur
3. /new_admin : inscription d'un administrateur

4. /delete_user : suppression d'un utilisateur

5. /delete_admin : suppression d'un administrateur

6. /prediction: prédictions de priorité à partir de données saisies

Endpoints de test/développement:

/test/users and admin list

/test/verify_user

5. Isolation, Intégration Continue et Livraison Continue

L'objectif ici est de s'assurer que la mise à disposition du modèle au client finale se déroule en suivant des critères de qualités suffisants et conformes aux règles de l'art dans le milieu des datasciences. Pour cela, on s'appuie sur les bonnes pratiques suivantes :

- découplage du cycle de vie du modèle
    - des tierces parties susceptibles de le consommer
    - du serveur d'API qui a pour rôle d'exposer le modèle au reste du monde et de gérer les aspects d'identifications.
- apprentissage automatique du modèle lors d'une mise à jour de la base de données
- évaluation automatique du modèle suite à chaque nouvel apprentissage
- tests automatiques de l'API en cas de mise à jour de cette dernière
- déploiement automatique après modification du modèle ou de l'API

5.1. Isolation du modèle et de son API

Pour rappel, le modèle permet de réaliser des prédictions, et une API basée sur le protocole *HttpRequest* permet de l'interroger. Une technologie de conteneurisation (basé sur Docker) contenant à la fois le modèle et le serveur d'API permet de s'assurer de l'isolation de l'applicatif.

Un Docker file est disponible dans le dépôt. Il construit une image qui va contenir notre modèle et le serveur d'API. Cette image est construite sur la base de l'image publique **tiangolo/uvicorn-gunicorn-fastapi:python3.8**

Lorsque le conteneur est démarré, un serveur web Apache est lancé, la commande *uvicorn* permet de faire le mapping entre le serveur web Apache et le serveur web *fastAPI* qui contient les endpoints. De cette façon, on s'assure de la robustesse d'un serveur Apache.

5.2. Partie CI, déclenchement sur mise à jour du dépôt

La partie intégration continue est réalisée à l'aide des pipelines GitHub Actions proposés par le service GitHub. L'intégration continue utilise deux workflows, chacun décris par un manifeste.

Le premier workflow se charge :

- de lancer l'apprentissage du modèle
- de tester automatiquement le modèle
- de tester automatiquement l'API
- en cas de succès, de construire l'image docker
- de mettre à disposition cette image docker dans un registery (*DockerHub*)

Le premier workflow se déclenche en cas de demande d'une PR pour merger une branche de travail dans la branche de référence.

Le second workflow se déclenche à la fin du premier. Une condition de succès est nécessaire pour qu'il réalise sa tâche. Son rôle est de mettre à disposition l'API au reste du monde. Pour cela, une image docker est déployée dans une machine virtuelle.

5.3. Partie CD, mise à disposition du modèle sur Internet

Le pipeline est conçu pour assurer une livraison continue également. Ainsi, la mise à jour du modèle ou de l'API provoquera le déploiement du serveur sur le web. Tout ceci se fait de façon automatique, mais le déploiement ne sera effectif qu'aux deux conditions suivantes :

1. Le modèle, après apprentissage, est toujours performant
2. L'API est testé avec succès

On s'assure de cette façon un niveau de qualité jugé suffisant pour la mise à disposition. Les parties tierces qui consomment le modèle ne seront donc pas impactées par les mises à jours.

Pour cela, on utilise un service de type *CaaS* nommé **Heroku**. C'est le rôle du second workflow de s'en occuper. En pratique, le service CaaS Heroku récupère le dépôt git, reconstruit une image docker, instancie automatiquement une machine virtuelle et démarre l'image docker dedans. Les ports sont automatiquement exposés. L'API est disponible à cette adresse : **https://dst-mar-accident-ee3901cfb695.herokuapp.com**

Note : une image est également disponible dans le DockerHub https://hub.docker.com/r/cyrilleq/mar24accd. pour pouvoir être utilisée par un autre service cloud comme Microsoft Azure ou Amazon Web Services.

5.4. Schéma d'implémentation

Revoir mise en forme