ETE 2025

Rapport de Stage-Projet

Rédigé par :

Mariama Ciré Camara

CAMM06609200

Détection de Fraude en Assurance

Introduction

Dans le cadre de mon stage en science des données, j'ai réalisé un projet de détection de fraude en assurance automobile. L'objectif principal était de concevoir un système intelligent capable d'identifier les cas potentiels de fraude à partir des données de réclamations, aussi bien tabulaires qu'images. Le projet intègre plusieurs étapes allant de l'analyse exploratoire des données à la conception d'une API de prédiction et d'une interface utilisateur.

1. Analyse Exploratoire des Données (EDA)

Données tabulaires :

Les données proviennent d'un jeu de données ouvert sur **Kaggle** comportant des informations sur les réclamations d'assurance automobile. Les variables comprennent des champs tels que le type de police, le nombre de véhicules, la date de la réclamation, l'âge du contrat, etc.

Étapes réalisées :

- Nettoyage des données : Après exploration je n'ai pas eu de nettoyage majeur à faire car il n'y avait pas de données manquantes ni de valeurs aberrantes.
- Étude de la distribution des variables.
- Analyse des corrélations et relations entre les variables.
- Identification des variables discriminantes pour la fraude.

Données images :

Le jeu de données comprend des images de véhicules endommagés. L'objectif était de détecter, à partir de ces images, les cas suspects de fraude.

Étapes réalisées :

- Redimensionnement des images.
- Normalisation des pixels.
- Création de labels binaires (fraude / non fraude).
- Prévisualisation des images et exploration des classes.

2. Préparation des Données et Modélisation

Données tabulaires :

Pour les données tabulaires, j'ai utilisé **OneHotEncoder** pour encoder les variables catégorielles puis divisé les données en entrainement et de test.

Pour la modélisation j'ai testé plusieurs modèles à savoir : la Régression Logistique, Random Forest, XGBoost, et Arbre de Décision. Le modèle retenu est **XGBoostClassifier**, optimisé en gérant le déséquilibre de classe et en sélectionnant les variables importantes qui définisse mieux le model. Les métriques de performance utilisées incluent Accuracy : 0.87 ; Précision sur la fraude: 81%.

Données images :

Pour les images, j'ai appliqué une stratégie d'**augmentation de données** afin de compenser le déséquilibre des classes (rotation, zoom, flips horizontaux).

J'ai utilisé le modèle **ResNet50**, pré-entraîné sur ImageNet puis affiné avec nos propres images (fine-tuning). Comme résultat je n'ai pas eu une bonne performance, il y avait beaucoup de faux positifs, j'ai ajusté le seuil à **0.3** pour maximiser la détection des fraudes.

3. Conception de l'API Flask

Une API a été développée avec **Flask**, permettant d'exposer deux points d'accès :

- /predict/tabulaire : pour les données de réclamations.
- /predict/image : pour l'envoi d'images de véhicules.

Les modèles sont chargés à partir de fichiers pré-enregistrés et téléchargés depuis Google Drive (car trop volumineux). Le traitement inclut la transformation des données d'entrée, l'appel aux modèles, et la génération de la prédiction.

4. Interface Utilisateur avec Streamlit

Une interface simple et intuitive a été créée avec **Streamlit**, permettant :

- Le remplissage d'un formulaire pour les données tabulaires.
- Le téléversement d'une image pour évaluation par le modèle ResNet.
- L'affichage des résultats en temps réel.

5. Déploiement

Tentatives de déploiement :

- **Render.com** : Échec dû à la taille des fichiers (limite de 100 Mo dépassée par le modèle ResNet).
- **Streamlit Cloud** : Problèmes de compatibilité avec TensorFlow et la limite de ressources.
- **Heroku** : Plusieurs tentatives infructueuses à cause de :
    - Dépassement de la limite de taille de l'application (slug > 500 Mo)
    - Incompatibilité de version avec tensorflow
    - Modules manquants (xgboost, tensorflow, etc.)

7. Difficultés Rencontrées

- Problèmes liés à la taille des fichiers lors du versionnage GitHub.
- Problèmes de compatibilité de versions Python avec certains packages (ex. : tensorflow, streamlit).
- Limitations des plateformes de déploiement gratuites.
- Débit Internet local instable ayant retardé l'entrainement des modèles, le téléchargement et le déploiement des composants.

Conclusion

Ce projet de détection de fraude en assurance a été bien plus qu'une simple application de mes connaissances académiques : il a représenté une véritable **expérience de croissance personnelle et professionnelle**. En mettant en œuvre des techniques de machine learning, d'analyse d'images, de développement d'API et d'interfaces utilisateur, j'ai pu concrétiser plusieurs compétences vues en cours. Mais surtout, j'ai **appris énormément au-delà du programme**.

Ce stage m'a permis de **découvrir de nouveaux outils** comme Streamlit, Google Drive API, Render et Heroku, et de **me familiariser avec la gestion d'environnements virtuels, le déploiement cloud, et les contraintes réelles d'un projet technique**. Chaque blocage, chaque erreur (parfois frustrante) m'a poussé à chercher, à expérimenter, à lire la documentation, à poser des questions, bref, à **adopter une posture proactive** de résolution de problème. Ces échecs m'ont forgé : ils m'ont aidé à développer une **meilleure rigueur, persévérance et autonomie** dans mon travail.

Au final, bien que le déploiement en ligne n'ait pas pu être finalisé comme souhaité, l'application fonctionne parfaitement en local, avec un fichier requirements.txt pour l'installation des dépendances, un script app.py qui représente le backend et un script streamlit_app.py pour le frontend. Cette solution permet d'assurer une stabilité et une accessibilité pour les démonstrations futures.