Rapport de Projet : Détection de Fraude à la Carte de Crédit

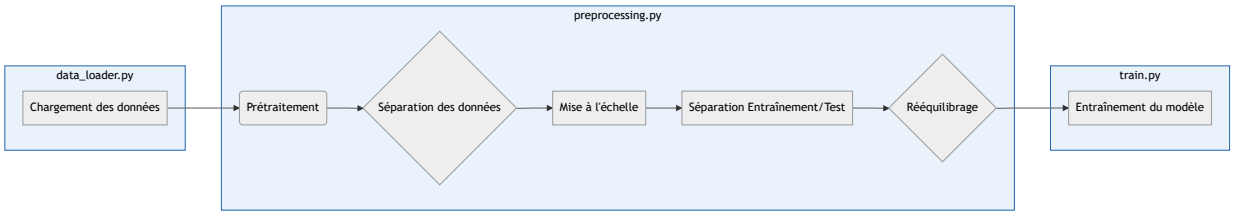
Ce document détaille la conception, la mise en œuvre et les résultats d'un projet visant à développer un modèle d'intelligence artificielle pour la détection de transactions frauduleuses par carte de crédit.

1. Description du Projet

L'objectif principal de ce projet est de construire un classificateur binaire capable d'identifier avec précision les transactions frauduleuses à partir d'un ensemble de données de transactions réelles. Étant donné le coût élevé des fraudes non détectées, le modèle doit être particulièrement performant pour identifier la classe minoritaire (fraudes).

Données et Prétraitement

Le projet utilise un jeu de données de transactions par carte de crédit fourni sous forme de fichier CSV. Le flux de traitement des données est le suivant :



Les étapes de prétraitement, gérées par `preprocessing.py` , sont :

- **Séparation des caractéristiques et de la cible :** La variable cible `Class` (0 pour légitime, 1 pour fraude) est isolée des autres caractéristiques.
- **Mise à l'échelle :** La caractéristique `Amount` (montant de la transaction) est normalisée à l'aide d'un `StandardScaler` pour éviter que son échelle n'influence indûment le modèle.
- **Division stratifiée :** Les données sont divisées en ensembles d'entraînement (80%) et de test (20%). Une division stratifiée (`stratify=labels`) est utilisée pour garantir que la proportion de fraudes et de transactions légitimes soit la même dans les deux ensembles, ce qui est essentiel pour les données déséquilibrées.
- **Rééquilibrage des classes :** Le jeu de données d'entraînement est fortement déséquilibré, avec beaucoup moins de fraudes que de transactions légitimes. Pour corriger cela, la technique **SMOTE** (Synthetic Minority Over-sampling Technique) est appliquée. SMOTE crée des échantillons synthétiques de la classe minoritaire (fraude) pour équilibrer la distribution des classes, permettant au modèle de mieux apprendre les caractéristiques des fraudes.

  Afin de garantir une évaluation impartiale, les éventuels doublons entre l'ensemble d'entraînement (augmenté par SMOTE) et l'ensemble de test sont ensuite supprimés.

2. Convergence et Surapprentissage

Convergence

La convergence est le stade de l'entraînement où le modèle n'améliore plus ses performances sur les données d'entraînement. La fonction de perte (loss) stagne, indiquant que le modèle a appris autant que possible des données. Le suivi de la courbe de perte est un moyen standard de visualiser la convergence.

Surapprentissage (Overfitting)

Le surapprentissage se produit lorsqu'un modèle apprend "par cœur" les données d'entraînement, y compris le bruit. Il devient alors trop spécialisé et perd sa capacité à généraliser à de nouvelles données (comme l'ensemble de test), affichant des performances bien moindres sur celles-ci.

Méthodes mises en œuvre

Plusieurs techniques ont été utilisées dans ce projet pour assurer une bonne convergence et éviter le surapprentissage :

- **Arrêt précoce (Early Stopping) :** Le `MLPClassifier` est configuré avec `early_stopping=True`. Cette option interrompt l'entraînement si la performance sur un ensemble de validation interne ne s'améliore pas pendant un certain nombre d'époques consécutives, empêchant ainsi le modèle de surapprendre.
- **Régularisation L2 :** Le paramètre `alpha` (ici `0.001`) du `MLPClassifier` introduit une pénalité sur la taille des poids du modèle. Cela décourage les poids trop élevés, menant à un modèle plus simple et moins susceptible de surapprendre.
- **Validation croisée (Cross-Validation) :** Lors de la recherche des hyperparamètres, une validation croisée à 5 plis (`cv=5`) est utilisée. Cette méthode entraîne et valide le modèle sur 5 sous-ensembles différents des données d'entraînement, fournissant une estimation plus robuste de sa capacité de généralisation.

3. Architectures et Hyperparamètres Testés

Le modèle choisi est un Perceptron Multi-Couches (`MLPClassifier`), un type de réseau de neurones. Pour trouver la meilleure configuration, nous avons utilisé `GridSearchCV` de Scikit-learn, qui teste systématiquement différentes combinaisons d'hyperparamètres.

- **Grille de recherche :** Les hyperparamètres testés étaient les suivants :
  - `hidden_layer_sizes` : Définit l'architecture du réseau, c'est-à-dire le nombre de couches cachées et le nombre de neurones dans chaque couche.
  - `activation` : La fonction appliquée à chaque neurone pour introduire de la non-linéarité, permettant au modèle d'apprendre des relations complexes.
  - `solver` : L'algorithme utilisé pour ajuster les poids du réseau afin de minimiser la fonction de perte (l'erreur du modèle).
  - `alpha` : Le paramètre de régularisation L2, qui aide à prévenir le surapprentissage en pénalisant les poids de valeur élevée.
  - `learning_rate` : Le taux d'apprentissage, qui contrôle la vitesse à laquelle le modèle ajuste ses poids pendant l'entraînement.
- **Meilleurs paramètres :** La recherche a identifié la configuration suivante comme étant la plus performante (basée sur le score F1) : `{'activation': 'relu',`

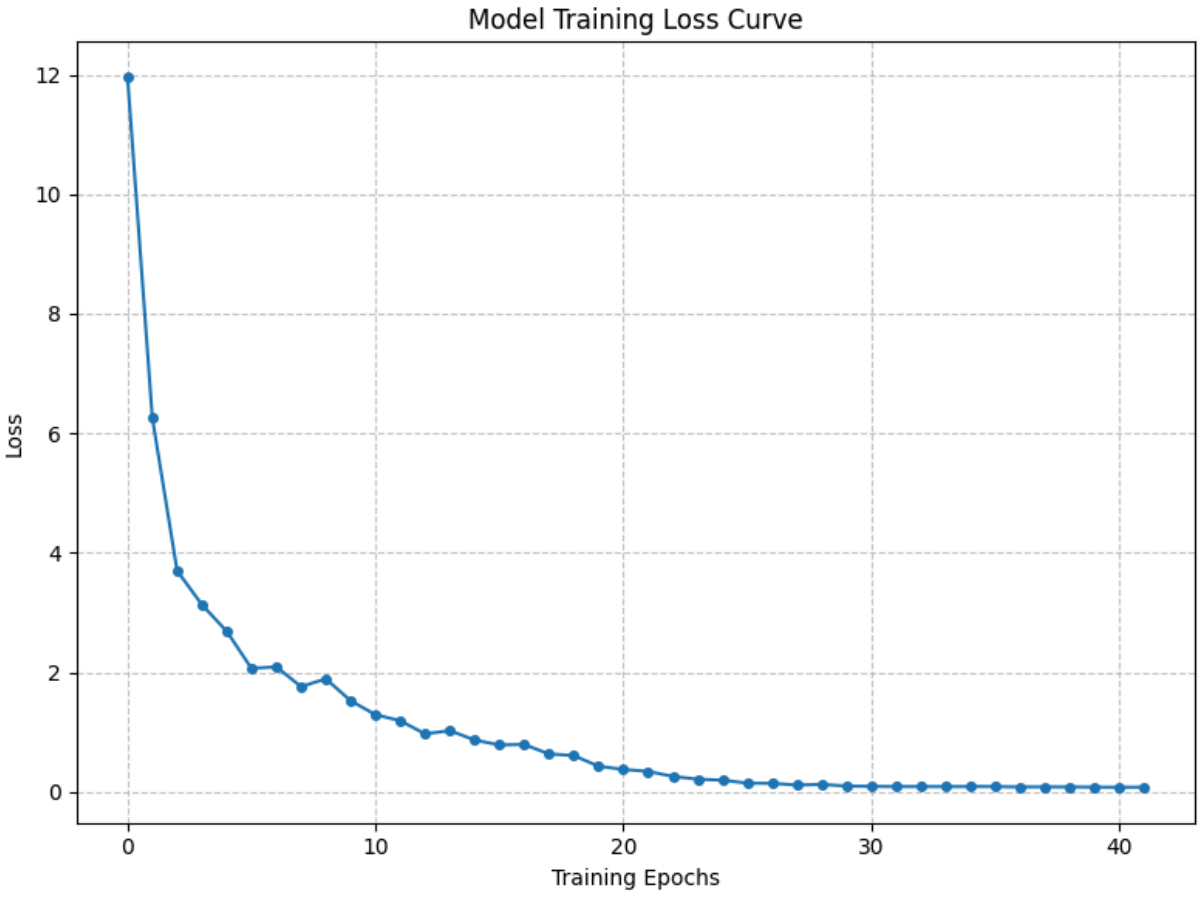
`'alpha': 0.001, 'hidden_layer_sizes': (64, 64), 'learning_rate': 'adaptive', 'solver': 'adam'}`.

- **Métrique d'évaluation :** Le **F1-score** a été choisi pour optimiser les hyperparamètres. C'est une moyenne harmonique de la précision et du rappel, ce qui en fait un excellent choix pour les problèmes de classification avec des classes déséquilibrées.

4. Étude de la Convergence

La convergence du modèle est analysée via la courbe de la fonction de perte (loss) au fil des époques d'entraînement.

Le graphique montre que la perte diminue très rapidement au cours des 10 premières époques. Ensuite, elle continue de diminuer plus lentement avant de se stabiliser complètement autour de la 40ème époque. Cela indique que le modèle a convergé efficacement et a cessé d'apprendre après ce point. Le mécanisme d'arrêt précoce a probablement mis fin à l'entraînement peu après.



5. Analyse des Résultats et Conclusion

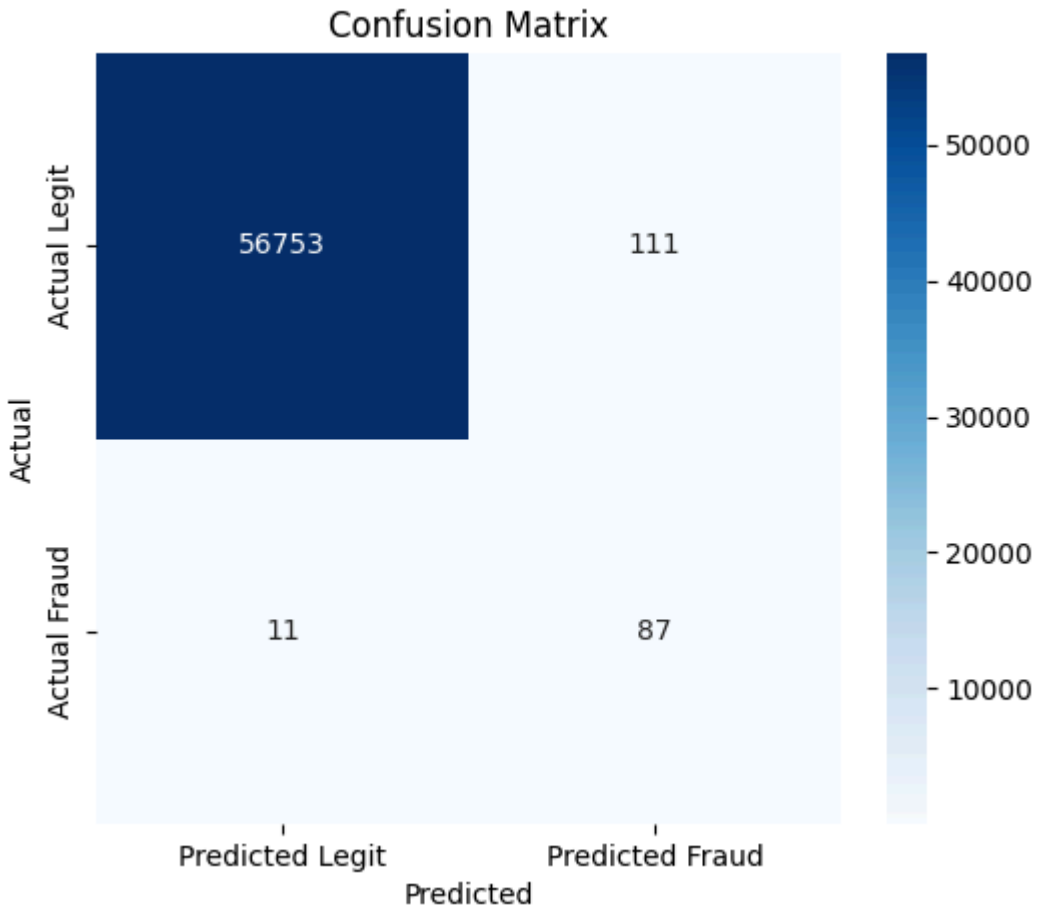
L'évaluation finale du modèle est réalisée sur l'ensemble de test, qui n'a pas été utilisé pendant l'entraînement ou le rééquilibrage.

Matrice de Confusion

La matrice de confusion visualise les performances du classificateur en détaillant les vrais positifs, les faux positifs, les vrais négatifs et les faux négatifs.

- **Transactions légitimes correctement prédites (TN) :** 56753
- **Transactions frauduleuses correctement prédites (TP) :** 87
- **Transactions légitimes prédites comme fraudes (FP) :** 111
- **Transactions frauduleuses manquées (FN) :** 11

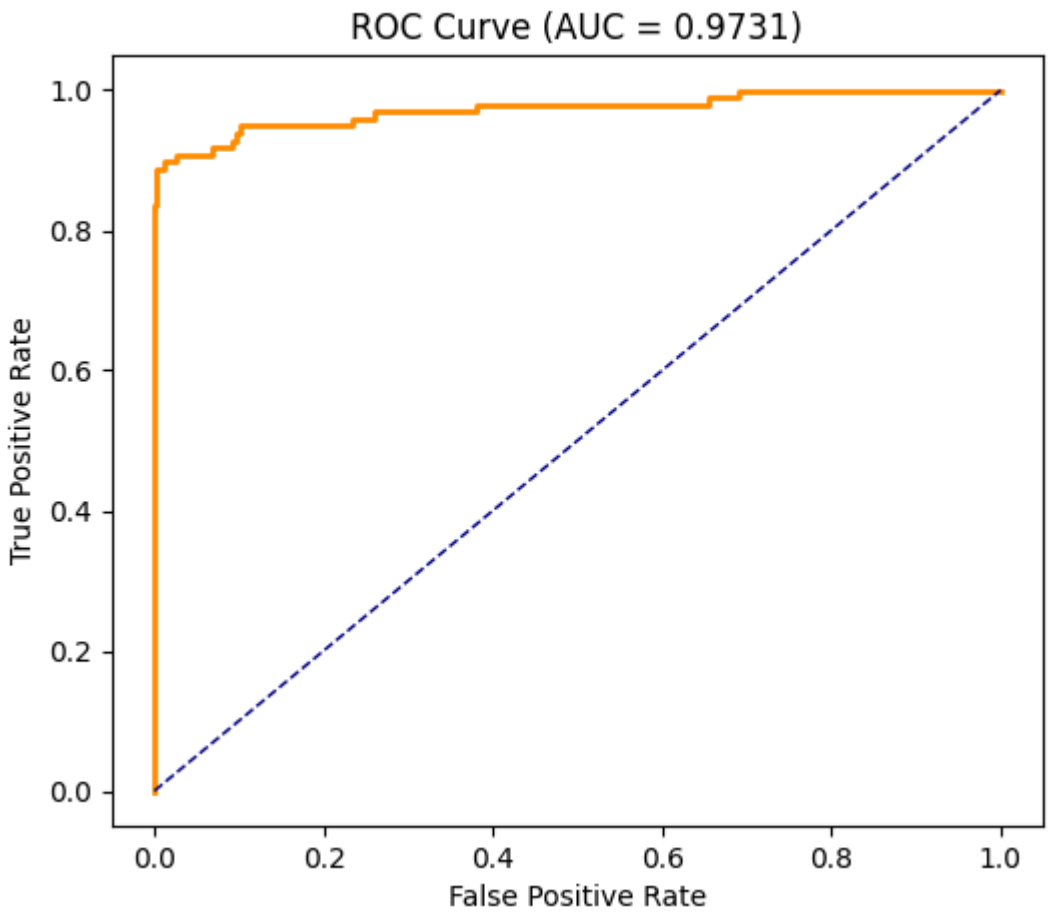
Le modèle est extrêmement précis pour les transactions légitimes. Plus important encore, il a réussi à identifier **87 des 98 fraudes** présentes dans l'ensemble de test, n'en manquant que 11.



Courbe ROC et AUC

La courbe ROC (Receiver Operating Characteristic) évalue la capacité du modèle à distinguer les deux classes. L'aire sous cette courbe (AUC) est une mesure synthétique de sa performance.

L'AUC est de **0.9731**, ce qui est un excellent score. Un score proche de 1 indique un modèle très performant, capable de faire une distinction claire entre les transactions frauduleuses et légitimes.



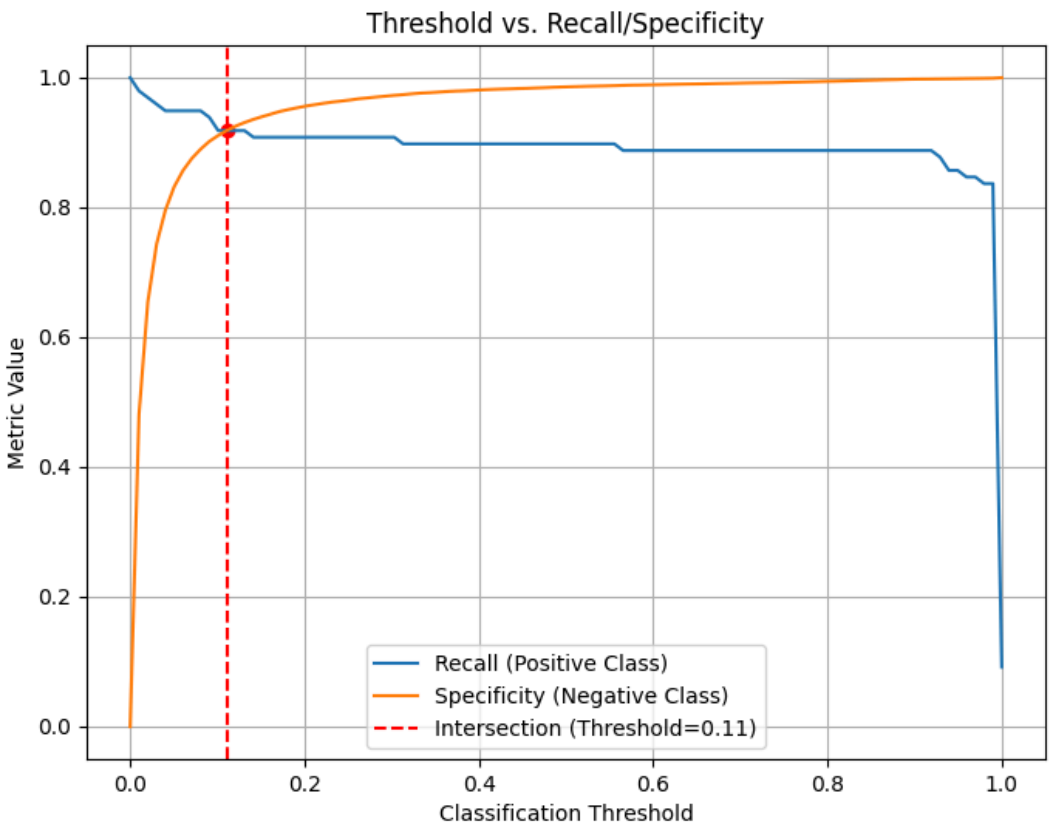
Le Seuil de Classification

Un modèle de classification comme le nôtre ne prédit pas directement "Fraude" ou "Légitime". Il produit en réalité une probabilité (un score entre 0 et 1) qu'une transaction soit une fraude. Le **seuil de classification** est la valeur à laquelle nous décidons de faire basculer cette probabilité en une décision finale.

- Si `Probabilité > Seuil` , la transaction est classée comme **Fraude**.
- Si `Probabilité <= Seuil` , la transaction est classée comme **Légitime**.

Un seuil **bas** permet de détecter plus de fraudes (meilleur Rappel) mais génère plus de fausses alertes. À l'inverse, un seuil **haut** réduit les fausses alertes (meilleure Spécificité) au risque de manquer plus de fraudes réelles.

Le code d'évaluation utilise un seuil de **0.925**. Ce seuil élevé est un choix conservateur : il vise à réduire drastiquement le nombre de faux positifs (transactions légitimes bloquées à tort), quitte à manquer quelques fraudes. L'intersection des courbes de rappel (recall) et de spécificité se situe autour de 0.11, ce qui représenterait un compromis plus équilibré. Le choix du seuil optimal dépend en fin de compte des objectifs de la banque.



Conclusion

Le projet a permis de développer un modèle de détection de fraude très efficace. Grâce à un prétraitement soigné, notamment le rééquilibrage avec SMOTE, et à une optimisation des hyperparamètres, le modèle atteint d'excellentes performances (AUC de 0.97). Il démontre une forte capacité à identifier les fraudes tout en maintenant un faible taux de fausses alertes, ce qui le rend apte à un déploiement en conditions réelles.

6. Répartition des Tâches

Notre méthodologie reposait sur explorer individuellement, puis construire ensemble.

Pour chaque étape majeure, que ce soit le prétraitement des données, le choix de l'architecture du modèle ou les stratégies d'évaluation, chaque membre commençait par rechercher et prototyper des solutions de son côté.

Nous organisions ensuite des points réguliers pour mettre nos trouvailles en commun. Une fois la stratégie choisie, nous la mettions en œuvre.

7. Code Source

Le code source du projet est structuré en plusieurs modules Python :

- `main.py` : Point d'entrée pour exécuter l'entraînement ou l'évaluation.
- `data_loader.py` : Gère le chargement des données depuis le fichier CSV.
- `preprocessing.py` : Contient toutes les fonctions de prétraitement des données.
- `model.py` : Définit l'architecture du modèle MLP et la grille de recherche d'hyperparamètres.
- `train.py` : Orchestre le processus d'entraînement, y compris la recherche par grille et la sauvegarde du modèle.
- `evaluate.py` : Évalue le modèle entraîné sur les données de test.
- `visualize.py` : Génère les graphiques de performance (matrice de confusion, courbe ROC, etc.).

8. Analyse Critique du Projet

Difficultés Rencontrées

- **Déséquilibre des classes :** Le principal défi était le faible nombre de fraudes par rapport aux transactions légitimes. Sans une stratégie de rééquilibrage comme SMOTE, le modèle aurait simplement appris à tout classer comme "légitime".
- **Choix du seuil :** Déterminer le seuil de classification optimal est un compromis délicat entre la détection maximale de fraudes et la minimisation des perturbations pour les clients légitimes.

Aspects Réussis

- La structuration du code en modules distincts a rendu le projet facile à gérer, à tester et à faire évoluer.
- L'utilisation de `GridSearchCV` pour l'optimisation et la division stratifiée pour l'évaluation a garanti des résultats fiables.
- Le modèle final est très performant, ce qui valide les choix techniques effectués tout au long du projet.