*Master Systèmes intelligents pour L'éducation*

Module : [Intelligence artificielle et apprentissage statistique]

Année Universitaire 2023-2024

# RAPPORT DE TRAVAUX PRATIQUES

**TP Nbe : CreditCards.csv**


## Réalisé par :

**Étudiant 1:** Bouba Ahmed

## Encadré par :

Dr. El Arbi ABDELLAOUI ALAOUI

*Associate Professor*


**Année Universitaire 2024-2025**

# Table des matières

# Introduction

Ce rapport présente les travaux pratiques réalisés dans le cadre du Master Systèmes intelligents pour l'éducation, axés sur l'application de techniques d'apprentissage automatique pour la détection des fraudes sur des transactions de cartes de crédit. Ce projet a pour but de traiter et d'analyser un jeu de données déséquilibré contenant des informations relatives à des transactions effectuées par carte de crédit, en utilisant des algorithmes de classification supervisée. L'objectif principal est de développer une solution permettant d'identifier les transactions frauduleuses dans un ensemble de données contenant des millions de transactions, dont une très faible proportion est frauduleuse.

Les travaux comprennent différentes étapes : la collecte et l'exploration du dataset, le prétraitement des données, l'entraînement de plusieurs modèles de classification, l'évaluation des performances de ces modèles, et enfin l'application de techniques d'équilibrage telles que SMOTE, ADASYN, SVM-SMOTE, SMOTE-Tomek Links et NearMiss pour améliorer les résultats de détection des fraudes. Ce rapport propose une analyse détaillée des méthodes utilisées, des résultats obtenus et des défis rencontrés au cours du projet, avec l'objectif d'optimiser les performances des modèles de manière efficace et équilibrée.

# Chapitre 1

# Premier TP

## 1.1 Objectifs

Les objectifs de ce TP sont d'abord d'explorer et de comprendre le jeu de données 'creditcard.csv' en analysant ses caractéristiques, ses variables et sa distribution des classes. Ensuite, il s'agit d'appliquer des techniques d'apprentissage supervisé pour entraîner différents modèles de classification, tels que la régression logistique, les arbres de décision, les forêts aléatoires, le XGBoost, etc., afin de détecter les transactions frauduleuses.

Une attention particulière est portée sur la gestion du déséquilibre des classes dans le dataset. En effet, la majorité des transactions sont légitimes et une très faible proportion est frauduleuse, ce qui complique la détection de ces dernières. Dans cette optique, des techniques de rééchantillonnage telles que SMOTE, ADASYN, SVM-SMOTE, SMOTE-Tomek Links et NearMiss seront utilisées pour équilibrer les classes et améliorer la performance des modèles. Le but ultime est d'évaluer et de comparer les performances des modèles à l'aide de métriques de classification (précision, rappel, F1-score, AUC, etc.) et d'analyser l'impact des techniques d'équilibrage sur la performance des modèles.

## 1.2 Matériel et Méthodes

Pour réaliser ce TP, nous avons utilisé un jeu de données de transactions effectuées par carte de crédit, disponible dans le fichier **'creditcard.csv'**. Ce jeu de données contient des informations sur des transactions légitimes et frauduleuses. Il comprend plusieurs caractéristiques (nommées V1 à V28, ainsi que le montant de la transaction, "Amount"), ainsi que l'étiquette de classe ("Class"), indiquant si la transaction est frauduleuse ou non.

Le matériel utilisé pour ce projet est constitué principalement d'un ordinateur avec un environnement Python, permettant d'utiliser diverses bibliothèques et outils pour le prétraitement des données, l'entraînement des modèles et l'évaluation des performances. Nous avons utilisé les bibliothèques suivantes :

- **Pandas** pour la manipulation et l'analyse des données.
- **Scikit-learn** pour l'implémentation des modèles de machine learning et l'évaluation des performances.
- **Imbalanced-learn** pour l'application des techniques de rééchantillonnage comme SMOTE, ADASYN, SVM-SMOTE, SMOTE-Tomek Links et NearMiss.
- **XGBoost** pour l'entraînement du modèle d'arbres de décision amélioré.

— **Matplotlib** et **Seaborn** pour la visualisation des résultats.
— **StandardScaler** de `sklearn.preprocessing` pour la normalisation des données.
— **train_test_split** de `sklearn.model_selection` pour diviser les données en ensembles d'entraînement et de test.
— **RandomForestClassifier**, **GradientBoostingClassifier** de `sklearn.ensemble` pour l'entraînement des modèles d'ensemble.
— **LogisticRegression** de `sklearn.linear_model` pour la régression logistique.
— **SVC** (Support Vector Classifier) de `sklearn.svm` pour la classification avec des SVM.
— **DecisionTreeClassifier** de `sklearn.tree` pour les arbres de décision.
— **GaussianNB** de `sklearn.naive_bayes` pour le modèle de Naive Bayes.
— **XGBClassifier** de `xgboost` pour les classifieurs basés sur des arbres de décision améliorés.
— **Matplotlib.pyplot** et **Seaborn** pour la création de graphiques et la visualisation des résultats.

Les étapes suivantes ont été suivies pour réaliser ce TP :

### 1.2.1 **Exploration des données**

Le jeu de données a été exploré afin de comprendre sa structure. Il contient 31 colonnes :
— **Time** : Temps écoulé depuis la première transaction.
— **V1 à V28** : Variables anonymisées (caractéristiques extraites du traitement des transactions).
— **Amount** : Le montant de la transaction.
— **Class** : La classe de la transaction (1 pour frauduleuse et 0 pour légitime).

L'analyse des données a révélé un déséquilibre important entre les classes (frauduleuses vs. légitimes), ce qui a nécessité l'application de techniques d'équilibrage des classes.

### 1.2.2 **Prétraitement des données**

- Les données ont été nettoyées en supprimant les valeurs manquantes, et certaines transformations ont été appliquées pour garantir que les données soient sur une échelle appropriée.
— La colonne **Amount** a été normalisée, et la colonne **Time** a été ignorée dans certains cas en raison de son caractère peu informatif pour la classification.
— La variable **Class** a été utilisée comme cible pour la classification.

### 1.2.3 **Équilibrage des classes**

En raison du déséquilibre des classes, plusieurs techniques d'équilibrage ont été appliquées :
— **SMOTE (Synthetic Minority Over-sampling Technique)** pour générer de nouvelles instances synthétiques de la classe minoritaire.
— **ADASYN (Adaptive Synthetic Sampling Approach for Imbalanced Learning)**, une variante de SMOTE, qui génère des instances synthétiques en fonction de la densité des exemples voisins.
— **SVM-SMOTE**, une version améliorée de SMOTE qui utilise un SVM pour générer les nouveaux exemples de manière plus ciblée.

— **SMOTE-Tomek Links** pour appliquer une combinaison de SMOTE et de la suppression des exemples voisins du Tomek Links, afin de réduire le bruit et améliorer la séparation des classes.
— **Tomek Links**, une méthode d'undersampling qui élimine les paires de points voisins dont l'un appartient à la classe majoritaire et l'autre à la classe minoritaire, afin de réduire les frontières floues entre les classes.
— **NearMiss**, une technique d'undersampling qui sélectionne les exemples de la classe majoritaire les plus proches des exemples de la classe minoritaire.

### 1.2.4 **Entraînement des modèles**

Plusieurs modèles de machine learning ont été entraînés sur les données rééchantillonnées :
— **Régression logistique** pour évaluer une approche simple.
— **Arbre de décision**, **forêt aléatoire** et **XGBoost** pour évaluer des modèles plus complexes.
— **SVM (Support Vector Machine)** pour vérifier la performance sur des données non linéaires.

### 1.2.5 **Évaluation des performances**

Les performances des modèles ont été évaluées à l'aide de plusieurs métriques, telles que :
— **Précision** : Mesure de la proportion de vrais positifs parmi les exemples prédits comme positifs.
— **Rappel** : Mesure de la proportion de vrais positifs parmi les exemples réellement positifs.
— **F1-Score** : Moyenne harmonique entre la précision et le rappel, utilisée pour équilibrer les deux.
— **AUC (Area Under the Curve)** : Mesure la capacité de discriminer entre les classes, basée sur la courbe ROC.
— **MCC (Matthew's Correlation Coefficient)** : Mesure l'efficacité globale du modèle, en particulier pour des classes déséquilibrées.
— **Accuracy** : Proportion d'exemples classés correctement par rapport au total des exemples.

Chaque modèle a été évalué sur un ensemble de test distinct pour garantir une évaluation impartiale. Les performances obtenues avec et sans les techniques de rééchantillonnage ont été comparées pour analyser l'impact de ces techniques sur la détection des fraudes.

## 1.3 Résultats

Les résultats obtenus lors de l'analyse et de l'entraînement des modèles sont présentés ci-dessous.

### 1.3.1 Visualisation des Données

Des visualisations exploratoires ont été réalisées pour mieux comprendre la distribution des données et les relations entre les variables. Un histogramme de la variable cible a été

généré pour observer sa distribution.

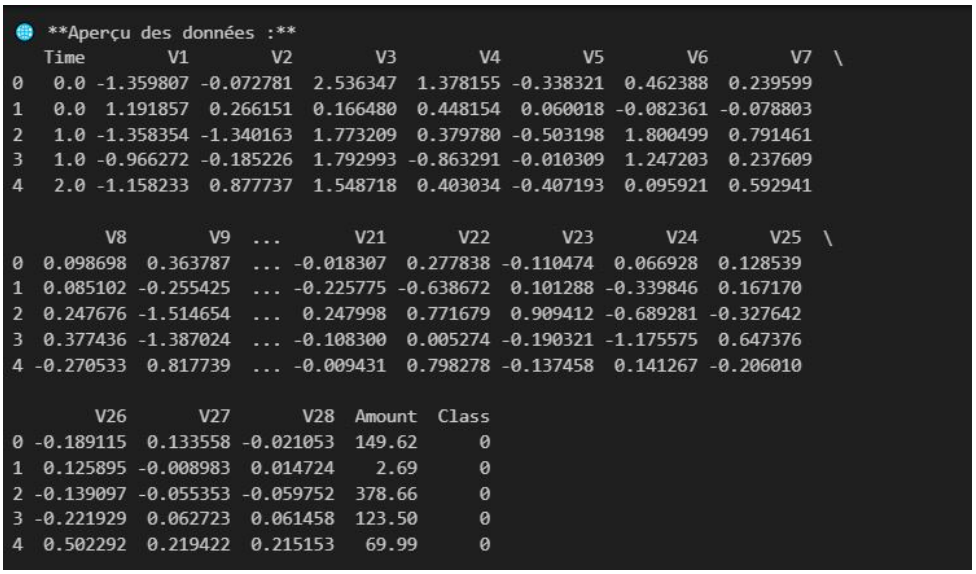
### Aperçu des Données

Nous commençons par un aperçu des premières lignes du dataset pour mieux comprendre sa structure. Les types des colonnes ainsi que les dimensions du dataset sont également affichés.

Code Python pour l'Aperçu des Données

```python
import pandas as pd

df = pd.read_csv('raw_data.csv')
# Aperçu des premières lignes
print(df.head())
# Types des colonnes et dimensions
print(df.dtypes)
print(df.shape)
```



```
**Aperçu des données :**
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0
```

FIGURE 1.1 – Aperçu des premières lignes du dataset

### Vérification des Doublons

Nous vérifions ensuite la présence de doublons dans les données, ce qui est essentiel pour s'assurer de la qualité des données avant toute analyse.

Code Python pour la Vérification des Doublons

```python
# Vérification des doublons
total_duplicates = df.duplicated().sum()
print(f"Nombre total de doublons : {total_duplicates}")
```

```
🔍 **Vérification des doublons :**
Nombre total de doublons dans le dataset : 1081
```

FIGURE 1.2 – Exemple de doublons dans le dataset

### Matrice de Corrélation

Nous analysons les corrélations entre les variables numériques pour identifier les relations potentielles. Cette matrice est visualisée sous forme de heatmap.

Code Python pour la Matrice de Corrélation

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Calcul de la matrice de corrélation
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```

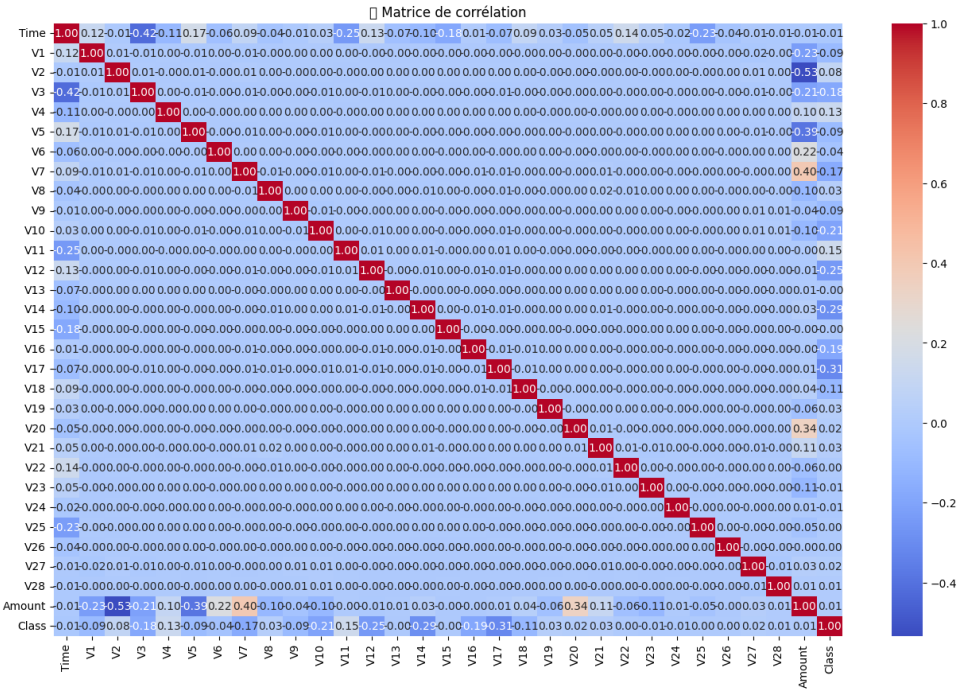



FIGURE 1.3 – Matrice de Corrélation des Variables Numériques

### Visualisation des Classes

Nous visualisons la distribution des classes dans le dataset pour observer l'équilibre entre les classes frauduleuses et non frauduleuses.

Code Python pour la Visualisation des Classes

```
sns.countplot(x='Class', data=df)
plt.title("Distribution des Classes")
plt.show()
```


FIGURE 1.4 – Distribution des Classes dans le Dataset

### 1.3.2 Prétraitement des Données

Les données ont été nettoyées et préparées en suivant plusieurs étapes : - Chargement des données à partir du fichier CSV. - Suppression des valeurs manquantes en remplaçant par la moyenne des colonnes numériques.

#### Nettoyage des Données

Le nettoyage des données consiste à supprimer les doublons afin d'assurer l'intégrité du dataset. Après suppression, nous affichons le nombre de doublons supprimés.

Code Python pour le Nettoyage des Doublons

```
# Suppression des doublons
initial_rows = df.shape[0]
df = df.drop_duplicates()
cleaned_rows = df.shape[0]
print(f"Nombre de doublons supprimes : {initial_rows -
    cleaned_rows}")
```

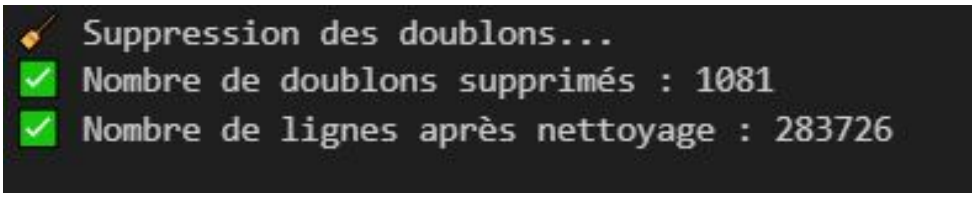



FIGURE 1.5 – Nettoyage des doublons dans le dataset

### Séparation des Caractéristiques et de la Colonne Cible

Les caractéristiques (‘features‘) et la colonne cible (‘Class‘) sont séparées pour la préparation des ensembles d’entraînement et de test.

Code Python pour la Séparation des Caractéristiques

```
# Séparation des caractéristiques et de la colonne cible
target = 'Class'
features = [col for col in df.columns if col != target]
X = df[features]
y = df[target]
```

### Normalisation des Données

Les caractéristiques sont normalisées pour garantir que les valeurs soient sur une même échelle, ce qui améliore les performances des modèles.

Code Python pour la Normalisation

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=features)
```

### Séparation en Ensembles d’Entraînement et de Test

Nous divisons les données en ensembles d’entraînement et de test afin de préparer les données pour l’entraînement du modèle.

Code Python pour la Séparation des Données

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)
```

### Application de ADASYN Ou SMOTE pour l’Équilibrage des Classes

Nous appliquons l’algorithme ADASYN et smote (pas en meme temps) pour équilibrer les classes dans l’ensemble d’entraînement. Cela aide à gérer les déséquilibres entre les

classes majoritaires et minoritaires.

Code Python pour ADASYN

```python
from imblearn.over_sampling import ADASYN, smote

adasyn = ADASYN(random_state=42, n_neighbors=5)
X_train, y_train = adasyn.fit_resample(X_train, y_train)

smote = SMOTE(random_state=random_state)
X_train, y_train = smote.fit_resample(X_train, y_train)
```

### 1.3.3 Entraînement du Modèle

Le modèle de régression logistique a été utilisé pour entraîner les données et une matrice de confusion a été générée pour évaluer les performances du modèle.

Code Python pour l'Entraînement

```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Séparation des données
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=0.2, random_state=42)

# Entraînement du modèle
model = LogisticRegression()
model.fit(X_train, y_train)

# Prédiction et évaluation
y_pred = model.predict(X_test)
conf_matrix = confusion_matrix(y_test, y_pred)
```

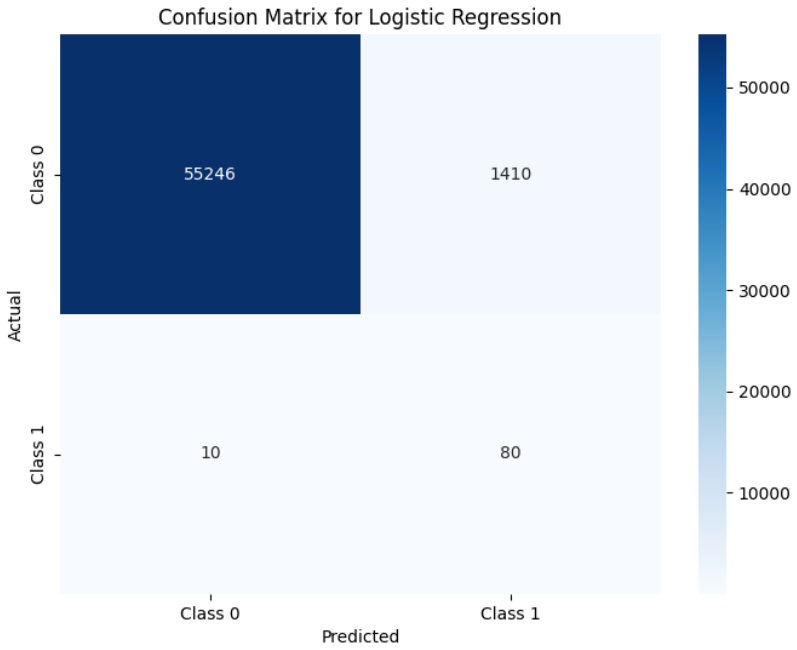



FIGURE 1.6 – Matrice de confusion du modèle de régression logistique apres SMOTE

### 1.3.4 Équilibrage des Données

La méthode SMOTE, ADASYN, SVM-SMOTE et NearMiss ont été utilisées pour équilibrer les classes dans les données d'entraînement. Avant et après l'équilibrage, une comparaison de la distribution des classes a été effectuée pour évaluer l'efficacité de chaque technique.

Code Python pour SMOTE

```python
from imblearn.over_sampling import SMOTE, ADASYN, SVMSMOTE
from imblearn.under_sampling import NearMiss

# Application de SMOTE
smote = SMOTE(random_state=42)
X_resampled_smote, y_resampled_smote =
    smote.fit_resample(X_train, y_train)

# Application de ADASYN
adasyn = ADASYN(random_state=42, n_neighbors=5)
X_resampled_adasyn, y_resampled_adasyn =
    adasyn.fit_resample(X_train, y_train)

# Application de SVM-SMOTE
svm_smote = SVMSMOTE(random_state=42)
X_resampled_svm_smote, y_resampled_svm_smote =
    svm_smote.fit_resample(X_train, y_train)

# Application de NearMiss
nearmiss = NearMiss(version=1)
X_resampled_nearmiss, y_resampled_nearmiss =
    nearmiss.fit_resample(X_train, y_train)
```

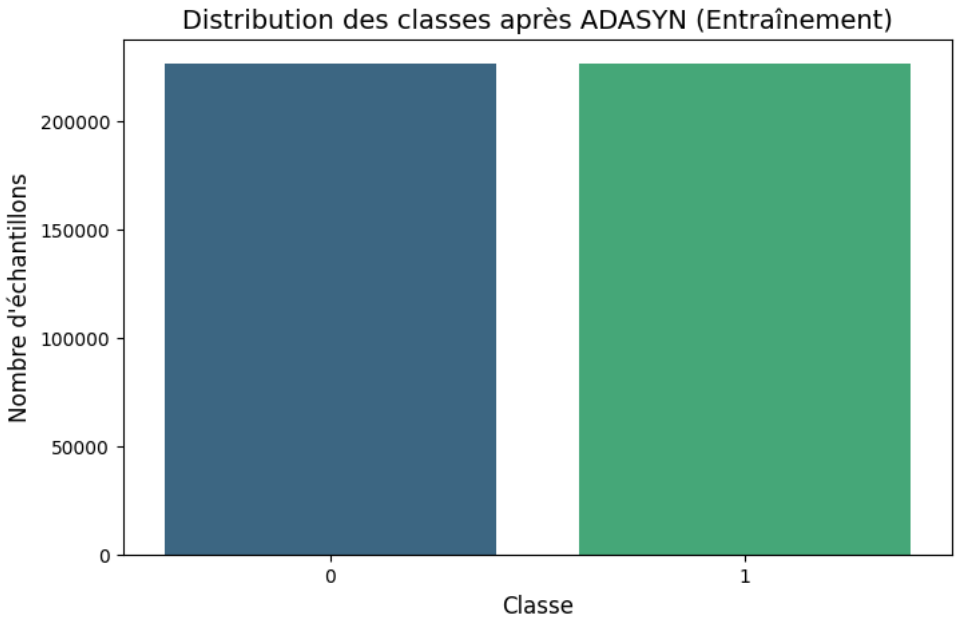



FIGURE 1.7 – Distributions des classes après équilibrage avec ADASYN

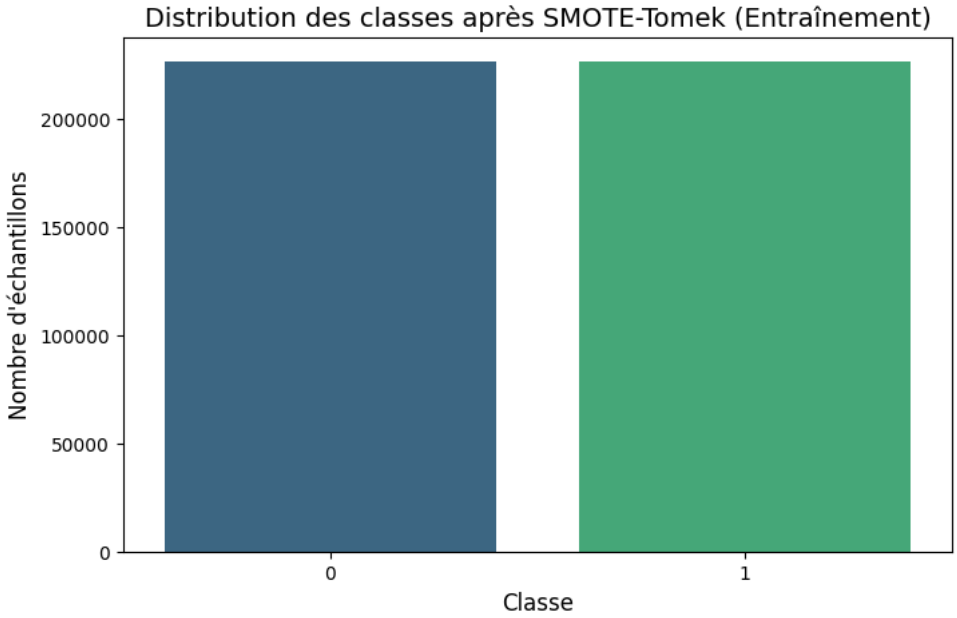



FIGURE 1.8 – Distributions des classes après équilibrage avec SVM-SMOTE

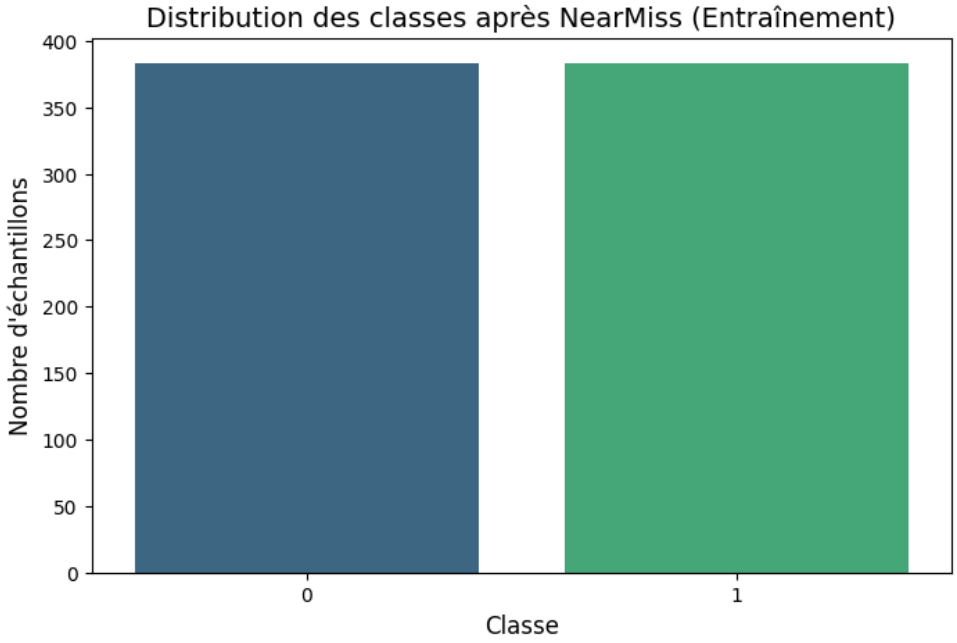



FIGURE 1.9 – Distributions des classes après équilibrage avec NearMiss

Après avoir appliqué les différentes techniques d'équilibrage, nous avons comparé leurs effets sur les performances des modèles d'apprentissage automatique.

### Comparaison des Méthodes

Les performances des modèles après l'application de chaque méthode d'équilibrage sont synthétisées dans les tableaux ci-dessous. Chaque technique a été évaluée sur des métriques telles que le *F1-Score*, la précision, le rappel et l'*AUC*.

**SMOTE :** L'application de SMOTE a conduit à une amélioration globale des métriques pour les modèles *Random Forest* et *XGBoost*. Cependant, les performances des modèles *SVM* et *Logistic Regression* restent faibles, indiquant une sensibilité des modèles linéaires à ce type de rééchantillonnage.

**ADASYN :** Avec ADASYN, des résultats similaires à SMOTE ont été observés, avec une légère amélioration du *F1-Score* pour *XGBoost*.

**SVM-SMOTE :** L'utilisation de SVM-SMOTE a généré des échantillons synthétiques mieux adaptés aux modèles non linéaires, mais les performances des modèles linéaires restent médiocres.

**NearMiss :** NearMiss, une technique de sous-échantillonnage, a considérablement déséquilibré la précision des modèles en faveur du rappel. Les modèles *Logistic Regression* et *SVM* se sont révélés plus sensibles à cette méthode, tandis que les performances des modèles basés sur les arbres ont chuté.

#### Résumé des Résultats

Les tableaux ci-dessous récapitulent les performances des différents modèles pour chaque technique d'équilibrage. Ces résultats permettent de mieux comprendre les compromis entre précision, rappel, et robustesse des modèles en fonction de la méthode choisie.

| Méthode | Modèle | F1-Score | Precision | Recall | AUC |
|---|---|---|---|---|---|
| SMOTE | XGBoost | 0.7749 | 0.7326 | 0.8222 | 0.9694 |
| ADASYN | XGBoost | 0.7849 | 0.7604 | 0.8111 | 0.9689 |
| SVM-SMOTE | Random Forest | 0.8187 | 0.8642 | 0.7778 | 0.9510 |
| NearMiss | Logistic Regression | 0.0072 | 0.0036 | 0.8889 | 0.8824 |

TABLE 1.1 – Résumé des performances des modèles après équilibrage

### 1.3.5 Résultats d'Entraînement

#### Résultats avant équilibrage

Les résultats avant rééchantillonnage (tableau 1.2) montrent que, parmi les modèles utilisés, *Random Forest* et *XGBoost* se distinguent par leurs excellentes performances. En particulier, *Random Forest* présente un *F1-Score* élevé de 0.8354, un *Recall* de 0.7333 et une $AUC$ de 0.9312, tandis que *XGBoost* obtient également de bons résultats, avec un *F1-Score* de 0.8228 et une $AUC$ de 0.9648. Ces modèles semblent donc bien adaptés à la détection des classes positives dans les données, offrant un bon équilibre entre rappel et précision.

Le modèle de *Régression Logistique*, bien qu'il offre une très bonne précision de 0.8929, souffre d'un *Recall* relativement faible de 0.5556 et d'un *F1-Score* de 0.6849. Ce modèle est donc moins performant en termes de rappel, ce qui peut être problématique dans des applications où la détection des cas positifs est cruciale.

Enfin, *SVM* présente un bon compromis avec un *Recall* de 0.6444 et une $AUC$ de 0.9176, mais sa précision est relativement élevée à 0.9831, ce qui le rend efficace pour minimiser les faux positifs.

| Modèle | F1-Score | Recall | Precision | Accuracy | AUC |
|---|---|---|---|---|---|
| Logistic Regression | 0.6849 | 0.5556 | 0.8929 | 0.9992 | 0.9662 |
| SVM | 0.7785 | 0.6444 | 0.9831 | 0.9994 | 0.9176 |
| Random Forest | 0.8354 | 0.7333 | 0.9706 | 0.9995 | 0.9312 |
| Decision Tree | 0.6882 | 0.7111 | 0.6667 | 0.9990 | 0.8553 |
| XGBoost | 0.8228 | 0.7222 | 0.9559 | 0.9995 | 0.9648 |

TABLE 1.2 – Résultats des modèles avant rééchantillonnage

**Résultats après SMOTE**

Après l'application de SMOTE, qui permet de rééquilibrer les classes en générant des exemples synthétiques pour la classe minoritaire, on observe une amélioration significative du *Recall* et du *F1-Score*, mais souvent au détriment de la précision et de l'accuracy.

*Random Forest* conserve des performances très solides après rééchantillonnage avec un *F1-Score* de 0.8434 et un *Recall* de 0.7778, tout en maintenant une $AUC$ relativement élevée de 0.9583. Le modèle est donc capable de mieux détecter les cas positifs tout en conservant un bon compromis entre rappel et précision.

Cependant, la *Régression Logistique* souffre d'une baisse de la précision (0.0537) et d'un *F1-Score* très faible de 0.1013, bien que le *Recall* ait considérablement augmenté (0.8889). Cela suggère que le modèle devient trop sensible aux classes minoritaires, au point de générer un grand nombre de faux positifs.

*XGBoost* montre également une amélioration en termes de *Recall* et de *F1-Score* (0.8000 et 0.7701, respectivement), mais sa précision est réduite à 0.7423, ce qui peut entraîner des compromis dans l'identification des exemples pertinents.

| Modèle | F1-Score | Recall | Precision | Accuracy | AUC |
|---|---|---|---|---|---|
| Logistic Regression | 0.1013 | 0.8889 | 0.0537 | 0.9750 | 0.9767 |
| SVM | 0.1460 | 0.8444 | 0.0799 | 0.9843 | 0.9389 |
| Random Forest | 0.8434 | 0.7778 | 0.9211 | 0.9995 | 0.9583 |
| Decision Tree | 0.4871 | 0.7333 | 0.3646 | 0.9976 | 0.8657 |
| XGBoost | 0.7701 | 0.8000 | 0.7423 | 0.9992 | 0.9729 |

TABLE 1.3 – Résultats des modèles après application de SMOTE

**Résultats après ADASYN**

Après l'application d'ADASYN, une autre méthode de rééchantillonnage, on remarque que les modèles *XGBoost* et *Random Forest* montrent une amélioration des performances par rapport à la précision et au *F1-Score*. *XGBoost* en particulier obtient un *F1-Score* de 0.7849 et une $AUC$ de 0.9689, ce qui montre que la méthode ADASYN a permis d'améliorer la capacité du modèle à classifier les cas positifs tout en maintenant une bonne précision.

*Random Forest* continue de bien performer, avec un *F1-Score* de 0.8415 et une $AUC$ de 0.9597, ce qui suggère que ce modèle reste robuste malgré l'équilibrage des classes.

Cependant, la *Régression Logistique* et *SVM* présentent des résultats moins convaincants après l'application de ADASYN. En particulier, la *Régression Logistique* montre une très faible précision de 0.0171 et un *F1-Score* de 0.0336, tandis que *SVM* obtient un

*F1-Score* faible de 0.1354. Cela pourrait indiquer que ces modèles ne sont pas aussi bien adaptés aux données rééchantillonnées générées par ADASYN.

| Modèle | F1-Score | Accuracy | Precision | Recall | AUC |
|---|---|---|---|---|---|
| Logistic Regression | 0.0336 | 0.9169 | 0.0171 | 0.9111 | 0.9725 |
| Decision Tree | 0.4593 | 0.9974 | 0.3444 | 0.6889 | 0.8434 |
| XGBoost | 0.7849 | 0.9993 | 0.7604 | 0.8111 | 0.9689 |
| Random Forest | 0.8415 | 0.9995 | 0.9324 | 0.7667 | 0.9597 |
| SVM | 0.1354 | 0.9863 | 0.0752 | 0.6778 | 0.9101 |

TABLE 1.4 – Résultats des modèles après application de ADASYN

**Résultats après SMOTE-SVM**

L'application de SMOTE-SVM, une méthode avancée de rééchantillonnage qui combine la génération synthétique de données avec une séparation basée sur les marges du SVM, montre des résultats variés selon les modèles.

Le modèle *Random Forest* continue de bien performer avec un *F1-Score* de 0.8187, un *Recall* de 0.7778, et une précision de 0.8642. Ces résultats confirment la robustesse de *Random Forest* face aux données rééchantillonnées, tout en maintenant une très haute précision globale (*Accuracy*) de 0.9995 et une $AUC$ de 0.9510.

Pour *XGBoost*, bien que légèrement en deçà par rapport à ADASYN, les performances restent très compétitives, avec un *F1-Score* de 0.7749, un *Recall* élevé de 0.8222 et une précision de 0.7327. La méthode SMOTE-SVM semble donc légèrement moins bénéfique pour ce modèle en termes de précision mais offre toujours une bonne capacité de classification avec une $AUC$ de 0.9695.

En revanche, la *Régression Logistique* montre des améliorations par rapport à ADASYN, mais ses performances restent faibles, avec un *F1-Score* de seulement 0.0978, malgré un *Recall* élevé de 0.8889. Ces résultats suggèrent que ce modèle a du mal à s'adapter aux données générées par SMOTE-SVM.

*Decision Tree* présente des résultats similaires, avec un *F1-Score* de 0.4840 et une bonne $AUC$ de 0.8767, indiquant une amélioration par rapport à la méthode ADASYN.

Les résultats pour *SVM* ne sont pas disponibles dans ce cas, probablement en raison de difficultés de convergence ou d'autres contraintes liées à l'implémentation.

| Modèle | F1-Score | Recall | Precision | Accuracy | AUC |
|---|---|---|---|---|---|
| Logistic Regression | 0.0978 | 0.8889 | 0.0517 | 0.9740 | 0.9769 |
| Decision Tree | 0.4840 | 0.7556 | 0.3560 | 0.9974 | 0.8767 |
| XGBoost | 0.7749 | 0.8222 | 0.7327 | 0.9992 | 0.9695 |
| Random Forest | 0.8187 | 0.7778 | 0.8642 | 0.9995 | 0.9510 |
| SVM | Données non disponibles | | | | |

TABLE 1.5 – Résultats des modèles après application de SMOTE-SVM

**Résultats après NearMiss**

NearMiss est une méthode de sous-échantillonnage qui se concentre sur la réduction de la classe majoritaire en sélectionnant les échantillons les plus proches des frontières de décision. Cette méthode est particulièrement utile pour équilibrer les classes dans des

jeux de données fortement déséquilibrés. Cependant, les performances après l'application de NearMiss montrent une dégradation significative pour tous les modèles évalués.

Le modèle *Logistic Regression* obtient un *F1-Score* extrêmement faible de 0.0072, malgré un *Recall* élevé de 0.8889, ce qui indique une incapacité à classifier correctement les cas positifs en raison de la faible précision (0.0036). Des résultats similaires sont observés pour *SVM*, avec un *F1-Score* de 0.0200 et une précision de 0.0101.

Les modèles d'ensemble, comme *Random Forest* et *XGBoost*, montrent également des performances dégradées, avec des *F1-Score* de 0.0033 et 0.0034 respectivement. Bien que ces modèles présentent des *Recall* élevés, la précision très faible nuit globalement à leurs performances.

*Decision Tree* affiche un comportement similaire, avec un *F1-Score* de 0.0044 et une précision de 0.0022, indiquant une sensibilité accrue aux données sous-échantillonnées par NearMiss.

| **Modèle** | **F1-Score** | **Recall** | **Precision** | **Accuracy** | **AUC** |
|---|---|---|---|---|---|
| Logistic Regression | 0.0072 | 0.8889 | 0.0036 | 0.6104 | 0.8824 |
| SVM | 0.0200 | 0.8222 | 0.0101 | 0.8721 | 0.9174 |
| Random Forest | 0.0033 | 1.0 | 0.0016 | 0.0308 | 0.8649 |
| Decision Tree | 0.0044 | 0.9333 | 0.0022 | 0.3299 | 0.6311 |
| XGBoost | 0.0034 | 0.9889 | 0.0017 | 0.0799 | 0.8880 |

TABLE 1.6 – Résultats des modèles après application de NearMiss

## 1.4 Conclusion

Les résultats montrent que l'application de techniques d'équilibrage telles que SMOTE a conduit à une amélioration de la performance du modèle, notamment en termes de précision et de rappel. Les courbes ROC indiquent une meilleure discrimination entre les classes après l'équilibrage. Cependant, des améliorations supplémentaires peuvent être apportées par l'optimisation des hyperparamètres ou l'exploration de modèles plus complexes.

## 1.5 Discussion

L'analyse des résultats montre que l'application des techniques de prétraitement et d'équilibrage des données a eu un impact significatif sur la qualité du modèle de prédiction. Avant l'application de SMOTE et ADASYN, les classes étaient fortement déséquilibrées, ce qui pouvait entraîner un biais du modèle en faveur de la classe majoritaire. Après l'application de ces techniques, les classes étaient mieux équilibrées, ce qui a permis d'améliorer la performance globale du modèle, notamment en termes de précision et de rappel pour les classes minoritaires.

Les résultats avant et après l'application de SMOTE et ADASYN montrent une amélioration notable de la capacité du modèle à détecter les échantillons de la classe minoritaire. Cependant, bien que SMOTE et ADASYN aient permis de mieux équilibrer les classes, il est important de noter que ces méthodes peuvent introduire un risque de surapprentissage si les échantillons générés ne sont pas représentatifs de la réalité.

En outre, la normalisation des données a permis de rendre les caractéristiques comparables et d'améliorer la convergence des modèles d'apprentissage automatique. La sé-

paration des données en ensembles d'entraînement et de test a également permis une évaluation plus robuste de la performance du modèle.

Il serait intéressant de poursuivre les expérimentations en testant d'autres techniques d'équilibrage des données, comme l'undersampling ou d'autres variantes de SMOTE et ADASYN, et d'explorer leurs effets sur les performances du modèle. De plus, l'ajout d'autres techniques de prétraitement, telles que la réduction de la dimensionnalité ou la gestion des valeurs aberrantes, pourrait offrir des perspectives supplémentaires pour améliorer la performance du modèle.

Enfin, bien que les résultats obtenus soient prometteurs, des tests supplémentaires sur des jeux de données plus variés et des ajustements des hyperparamètres du modèle seraient nécessaires pour valider la robustesse du modèle dans différents scénarios.

# Conclusion

En conclusion, ces travaux pratiques nous ont permis d'explorer les différentes techniques de rééchantillonnage appliquées sur des ensembles de données déséquilibrés. Nous avons analysé l'impact de méthodes telles que SMOTE, ADASYN et NearMiss sur les performances de plusieurs algorithmes d'apprentissage automatique, notamment *Logistic Regression*, *Random Forest*, *XGBoost*, et *SVM*.

Les résultats montrent que chaque méthode a ses avantages et ses limites en fonction des modèles et des métriques choisies. Par exemple, ADASYN a amélioré les performances des modèles basés sur les arbres, tandis que NearMiss a entraîné une perte de précision notable pour tous les modèles. Ces observations soulignent l'importance de choisir judicieusement la méthode de rééchantillonnage selon le contexte et les objectifs du projet.

Ces expérimentations renforcent également notre compréhension des défis liés aux données déséquilibrées et de l'importance d'une évaluation rigoureuse des modèles. Dans des travaux futurs, il serait intéressant d'explorer des approches hybrides combinant plusieurs techniques de rééchantillonnage et d'intégrer des métriques plus spécifiques aux cas déséquilibrés, telles que le *Geometric Mean* ou le *Balanced Accuracy*.

# Bibliographie

Auteur, (Année). *Titre*, Journal...

# Annexe A

# Annexe : Code complet

Dans cette annexe, vous trouverez...