

République Algérienne Démocratique et Populaire

وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



المدرسة الوطنية للإعلام الآلي

المعهد الوطني للتكوين في الإعلام الآلي (سابقا)

Ecole Nationale Supérieure d'Informatique

ex. INI (Institut National de formation en Informatique)

# TP BDM

## Sujet N°6 : classes déséquilibrées

### Membres d'équipe :

- Saadi Anis
- Hamitouche Monsef

### Encadré par :

Mm. Hamdad

## Tableau de matières :

<b>Notes Importantes avant de commencer:</b>	<b>4</b>
<b>1. Introduction</b>	<b>5</b>
<b>2. Explication de problème et définition des algorithmes utilisés pour le résoudre</b>	<b>6</b>
• A - Explication de problème	6
• B- Les algorithmes pour résoudre le problème des classes déséquilibrées	7
1- Sous-échantillonnage :	7
2- Sur-échantillonnage :	7
3- Méthodes hybrides :	8
4- Changement de l'algorithme de classification qui est spécialement conçu pour les classes déséquilibrées:	8
<b>3. big dataset avec classes déséquilibrées et donner ces caractéristiques</b>	<b>9</b>
1. description de dataset :	9
A. Les features de dataset:	9
2. description de la dataset	10
A - caractéristiques de dataset :	10
B - quelques preuves sur les caractéristiques:	10
<b>4. Tester les méthodes ensemblistes ET (Extremely Randomized Trees) et Catboost</b>	<b>13</b>
• Méthode ensembliste:	13
1) Extremely Randomized Trees	14
2) Catboost	17
<b>5. Appliquer une méthode d'équilibrage des classes, re-tester et re-comparer les résultats avec 3</b>	<b>21</b>
5-1- Application de la méthode d'équilibrage	21
5-1-a- Code d'application de la méthode en utilisant SMOTE sans Spark :	21
5-1-b - Code avec Spark :	22
5-2-Re-test des méthodes de classification après l'équilibrage :	22
5-2-a) ET :	22
5-2-b) Catboost	23
5-3- Comparaison des résultats:	24
a) ET:	25
b) Catboost:	25
<b>6. Conclusion</b>	<b>26</b>

## Table des figures:

<i>Figure 1: Illustration de 2 classes en déséquilibre. ....</i>	<i>6</i>
<i>Figure 2: 10 lignes de la dataset "Credit Card Fraud Detection".....</i>	<i>10</i>
<i>figure 3: nombre d'échantillons dans chaque classe .....</i>	<i>11</i>
<i>Figure 4: visualisation des 2 classes .....</i>	<i>11</i>
<i>Figure 5 : calcule du imbalance ratio.....</i>	<i>12</i>
<i>Figure 6 : calcule du degré de séparation.....</i>	<i>12</i>
<i>Figure 7: chargement des données à partir d'un fichier CSV.....</i>	<i>14</i>
<i>Figure 8: calcule du temps d'exécution avec python.....</i>	<i>14</i>
<i>Figure 9: calcule du temps d'exécution avec Spark.....</i>	<i>15</i>
<i>Figure 10: les performances du modèle ExtraTreesClassifier.....</i>	<i>15</i>
<i>Figure 11: Application de l'algorithme Catboost sans spark .....</i>	<i>18</i>
<i>Figure 12: Application de l'algorithme Catboost avec spark .....</i>	<i>19</i>
<i>Figure 13: les performances du modèle Catboost.....</i>	<i>19</i>
<i>Figure 14: application de la méthode smote sans spark.....</i>	<i>21</i>
<i>Figure 15: méthode over_simpling de Pyspark.....</i>	<i>22</i>
<i>Figure 16: application de l'algorithme ET après l'équilibrage.....</i>	<i>22</i>
<i>Figure 17: les performances du modèle ET après l'équilibrage.....</i>	<i>23</i>
<i>Figure 18: application de l'algorithme Catboost après l'équilibrage.....</i>	<i>24</i>
<i>Figure 19: les performances du modèle Catboost après l'équilibrage.....</i>	<i>24</i>

## Notes Importantes avant de commencer:

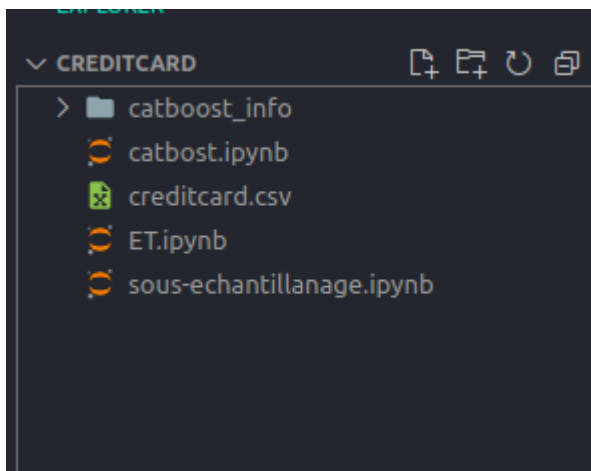
1-Pour pouvoir suivre notre solution :

Il faut installer python 3,pip, pyspark et toutes les bibliothèques dans les prochains applications.

2- La programmation se base sur google collab pour faciliter les traitement surtout avec spark. Mais vous pouvez travailler sur votre environnement local, ce qui est un peu délicat dans les installations.

Pour plus d'information visiter le site officiel de PySpark [https://spark.apache.org/docs/latest/api/python/getting\\_started/install.html](https://spark.apache.org/docs/latest/api/python/getting_started/install.html)

3- La structure du fichiers est comme suit:



le répertoire creditcard se compose de 4 fichiers qui sont :

1. **Creditcard.csv** : la base de données sous format .csv
2. **ET.ipynb** : fichier jupiter (python) de la méthode ET
3. **catboost.ipynb** : fichier jupiter (python) de la méthode catboost
4. **sous-échantillonnage.ipynb** : fichier jupiter (python) de la méthode de classes déséquilibrées

# 1.Introduction

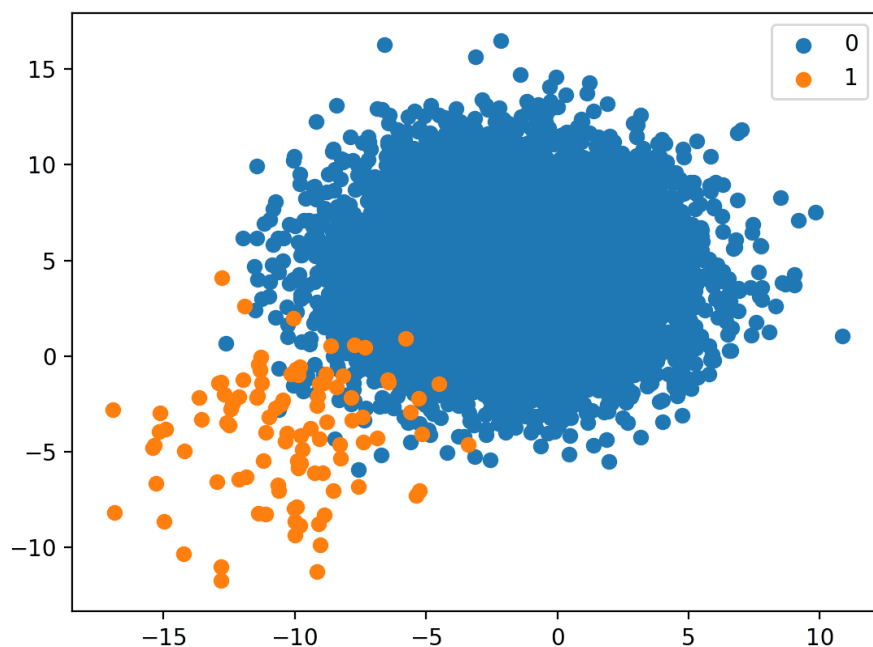
Les méthodes d'apprentissage automatique sont de plus en plus utilisées pour la classification de données dans divers domaines tels que la médecine, la finance et la sécurité. Cependant, un problème commun rencontré lors de l'entraînement des modèles est le déséquilibre de classes, où une classe est sous-représentée par rapport à une autre. Dans de tels cas, les modèles peuvent être biaisés en faveur de la classe majoritaire et avoir une performance médiocre pour la classe minoritaire.

Pour résoudre ce problème, des techniques d'équilibrage de classes ont été développées pour améliorer les performances des modèles d'apprentissage automatique pour les données déséquilibrées. Dans ce rapport, nous allons explorer différentes méthodes d'équilibrage de classes, en mettant l'accent sur les approches ensemblistes qui ont démontré leur efficacité pour résoudre ce problème.

## 2. Explication de problème et définition des algorithmes utilisés pour le résoudre

### • A - Explication de problème

Le problème de classes déséquilibrées se produit lorsque les classes d'un ensemble de données ne sont pas réparties de manière égale, c'est-à-dire qu'une classe peut avoir beaucoup plus d'exemples que les autres classes. Cela peut poser des problèmes lors de l'apprentissage automatique car la plupart des algorithmes de classification ont tendance à privilégier la précision globale plutôt que la précision de chaque classe. Dans le cas de classes déséquilibrées, cela signifie que l'algorithme peut être très précis pour la classe majoritaire, mais très mauvais pour les classes minoritaires.



**Figure 1:** Illustration de 2 classes en déséquilibre.

En conséquence, les prédictions de l'algorithme peuvent être biaisées en faveur de la classe majoritaire, ce qui peut avoir des conséquences graves dans certaines applications, telles que la détection de fraude ou la détection de maladies rares.

## • B- Les algorithmes pour résoudre le problème des classes déséquilibrées

Pour résoudre ce problème, il existe plusieurs approches telles que:

### 1- Sous-échantillonnage :

supprimer une partie des données de la classe majoritaire afin d'obtenir un équilibre entre les classes. Cela peut cependant entraîner une perte d'informations importantes.

#### Avantages :

- Réduire la taille de la classe majoritaire, ce qui peut faciliter l'apprentissage.
- Peut être efficace pour les ensembles de données très grands.

#### Inconvénients :

- Risque de perte d'informations importantes.
- Peut conduire à un biais de sélection des données si la suppression est aléatoire.

#### Exemple :

##### avant

class0 : a,b,a,a,a,b,a,c,a,c,a,c,a,c,z,b,b,a,a  
class1 : a,a,b

##### après:

class0 : a,b,a  
class1 : a,a,b

### 2- Sur-échantillonnage :

augmenter artificiellement la taille de la classe minoritaire en dupliquant ou en générant de nouvelles données. Cette approche peut causer un sur-apprentissage (overfitting) ou introduire du bruit dans les données.

#### Avantages :

- Augmente la taille de la classe minoritaire, ce qui peut améliorer la précision de la classification.
- Peut être utilisé pour générer des données synthétiques si la classe minoritaire est très petite.

#### Inconvénients :

- Peut conduire à un sur-apprentissage (overfitting) si la duplication est excessive.
- Peut introduire du bruit dans les données synthétiques générées.

#### Exemple :

##### avant

class0 : a,b,a,a,a,b,a,c,a,c,a,c,a,c,z,b,b,a,a  
class1 : a,a,b

##### après:

class0 : a,b,a,a,a,b,a,c,a,c,a,c,a,c,z,b,b,a,a  
class1 : a,a,b,a,b,a,a,b,a,b,a,b,a,b,b,b,a,a,b

### 3- Méthodes hybrides :

combiner les deux approches précédentes pour obtenir un équilibre entre les classes sans perdre d'informations.

**Avantages :**

- Peut combiner les avantages de sous-échantillonnage et sur-échantillonnage.
- Peut améliorer la performance globale du modèle de classification.

**Inconvénients :**

- Peut être plus complexe à mettre en œuvre.
- Peut augmenter le temps de traitement nécessaire pour l'apprentissage.

**Exemple :**

avant:

class0 : a,b,a,a,a,b,a,c,a,c,a,c,a,c,z,b,b,a,a  
class1: a,a,b

après:

class0 : a,b,a,a,a,b,a,c,a,c,a,  
class1: a,a,b,a,b,a,a,b,a,b,a,

### 4- Changement de l'algorithme de classification qui est spécialement conçus pour les classes déséquilibrées:

tels que l'Extremely Randomized Trees (ET) et Catboost. Ces algorithmes ont des mécanismes internes pour traiter les classes déséquilibrées.

**Avantages :**

- Ces algorithmes ont des mécanismes internes pour traiter les classes déséquilibrées.
- Peuvent être très efficaces pour les ensembles de données très déséquilibrés.

**Inconvénients :**

- Peut nécessiter une expertise technique pour comprendre et utiliser ces algorithmes.
- Peut ne pas être aussi largement utilisé ou disponible que les algorithmes de classification standard.

**Exemple:**

- Des fois, on aura des un algorithme qui plus performant que d'autre dans un cas comme on va le voir dans la partie pratique de notre rapport entre ET et catboost

**NB:**

Il est important de noter que chaque approche a ses avantages et ses inconvénients, et qu'il est donc important de choisir celle qui convient le mieux au problème de classification spécifique que l'on souhaite résoudre.



### 3. big dataset avec classes déséquilibrées et donner ces caractéristiques.

- **Nom du dataset** : Credit Card Fraud Detection Dataset
- **Source** : <https://www.kaggle.com/mlg-ulb/creditcardfraud>

#### 1. description de dataset :

Ce dataset “ Credit Card Fraud Detection Dataset” est utilisé pour la détection de fraudes dans les transactions par carte de crédit. Il contient des transactions effectuées par des titulaires de cartes de crédit en septembre 2013 sur une période de deux jours. Comme on peut le voir, les classes sont très déséquilibrées, la classe frauduleuse ne représentant que 0,172% de l'ensemble des transactions.

Il s'agit d'un dataset de grande taille, avec plus de 280 000 instances et 30 features et une sortie ‘classe’ pour déterminer s’il y a un fraude. Ce genre de dataset peut être difficile à traiter avec des algorithmes classiques de machine learning en raison de la présence de classes déséquilibrées.

#### 1. Les features de dataset:

Le jeu de données "Credit Card Fraud Detection" contient les éléments suivants pour chaque transaction :

**Time** : le nombre de secondes écoulées entre la transaction en question et la première transaction du jeu de données.

**V1, V2, ..., V28** : des variables numériques anonymisées qui ont été obtenues à partir d'une transformation PCA (Analyse en Composantes Principales) des données originales pour des raisons de confidentialité.

**Amount** : le montant de la transaction.

**Class** : une variable binaire qui indique si la transaction est frauduleuse (1) ou non (0).

Il est important de noter que les données ont été transformées pour que les informations personnelles des titulaires de carte de crédit soient anonymisées et protégées. Cela signifie que les variables V1, V2, ..., V28 ne peuvent pas être interprétées facilement et qu'il n'y a pas de variables explicatives supplémentaires fournies dans le jeu de données.

data.head(10)

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99
5	2.0	-0.425966	0.960523	1.141109	-0.168252	0.420987	-0.029728	0.476201	0.260314	-0.568671	...	-0.208254	-0.559825	-0.026398	-0.371427	-0.232794	0.105915	0.253844	0.081080	3.67
6	4.0	1.229658	0.141004	0.045371	1.202613	0.191881	0.272708	-0.005159	0.081213	0.464960	...	-0.167716	-0.270710	-0.154104	-0.780055	0.750137	-0.257237	0.034507	0.005168	4.99
7	7.0	-0.644269	1.417964	1.074380	-0.492199	0.948934	0.428118	1.120631	-3.807864	0.615375	...	1.943465	-1.015455	0.057504	-0.649709	-0.415267	-0.051634	-1.206921	-1.085339	40.80
8	7.0	-0.894286	0.286157	-0.113192	-0.271526	2.669599	3.721818	0.370145	0.851084	-0.392048	...	-0.073425	-0.268092	-0.204233	1.011592	0.373205	-0.384157	0.011747	0.142404	93.20
9	9.0	-0.338262	1.119593	1.044367	-0.222187	0.499361	-0.246761	0.651583	0.069539	-0.736727	...	-0.246914	-0.633753	-0.120794	-0.385050	-0.069733	0.094199	0.246219	0.083076	3.68

10 rows x 31 columns

*Figure 2: 10 lignes de la dataset "Credit Card Fraud Detection"*

**NB:**

Il est important de noter que les données ont été transformées pour que les informations personnelles des titulaires de carte de crédit soient anonymisées et protégées. Cela signifie que les variables V1, V2, ..., V28 ne peuvent pas être interprétées facilement et qu'il n'y a pas de variables explicatives supplémentaires fournies dans le jeu de données.

## 2. description de la dataset

### A - caractéristiques de dataset :

- Nombre d'instances : 284,807
- Nombre de features : 30
- Classes : Fraudulent (classe minoritaire) / Non-fraudulent (classe majoritaire)
- Ratio de classes : 0.172% (classe minoritaire) / 99.828% (classe majoritaire)
- Pas de cases vides
- dataset normalisée pour assurer la confidentialité des clients ( carte bancaire )

### B - quelque preuves sur les caractéristiques:

**Déséquilibre de classe :** Le nombre d'échantillons dans la classe minoritaire ( Non-fraudulent) est beaucoup plus faible que celui dans la classe majoritaire (Fraudulent). On peut utiliser la bibliothèque Pandas pour calculer le nombre d'échantillons dans chaque classe :

```

class_counts = data['Class'].value_counts()
print(class_counts)
✓ 0.1s

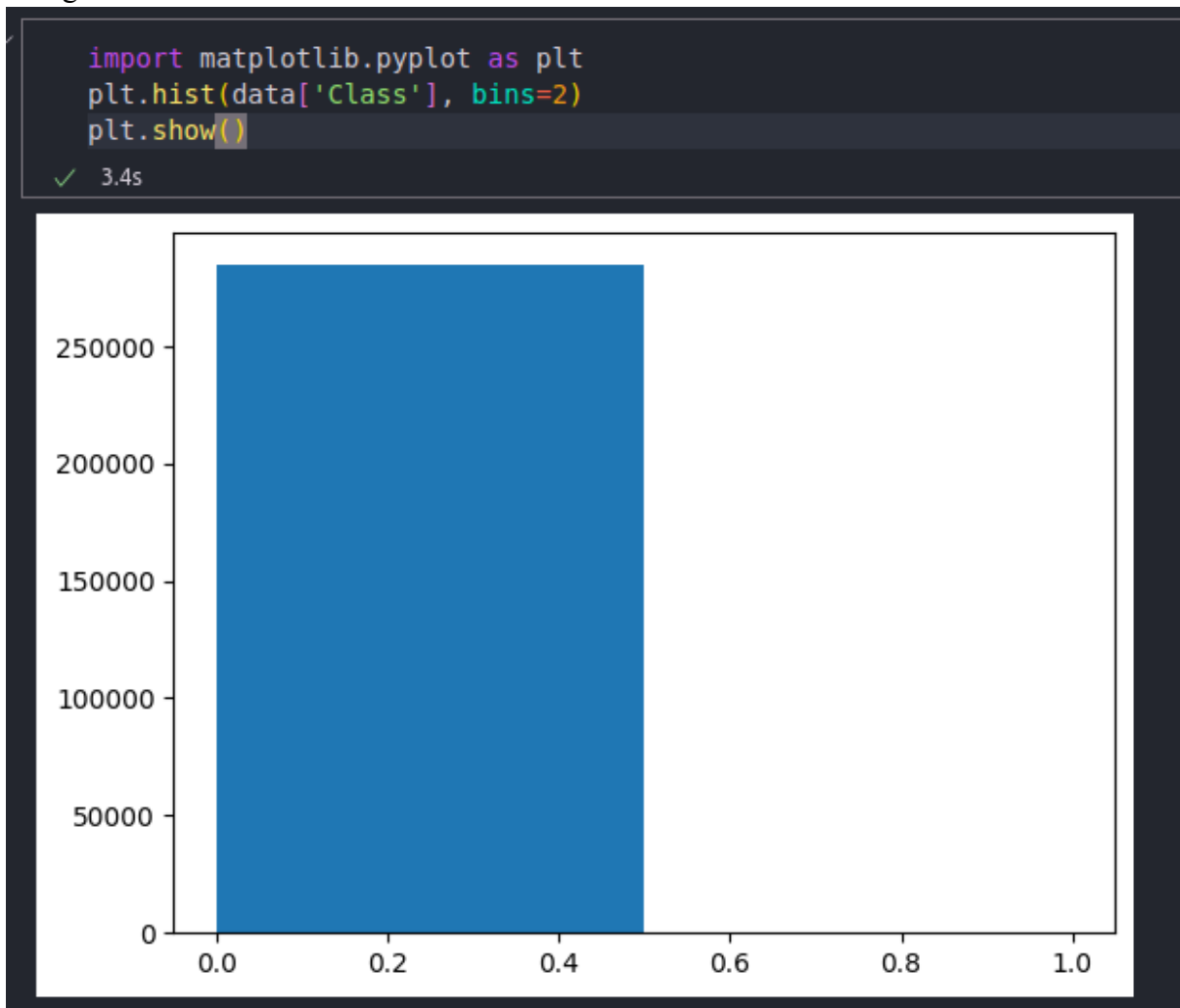
Class
0    284315
1      492
Name: count, dtype: int64

```

*figure 3: nombre d'échantillons dans chaque classe*

le nombre d'échantillons de la classe majoritaire est beaucoup plus grand que la classe minoritaire

**Distribution de classe :** La distribution de classe peut être visualisée à l'aide d'un histogramme ou d'un diagramme circulaire pour avoir une meilleure compréhension de la proportion de chaque classe. On peut utiliser la bibliothèque Matplotlib pour créer un histogramme de la distribution de classe :



*Figure 4: visualisation des 2 classes*

On remarque que la classe minoritaire est presque invisible par rapport à la classe majoritaire

**Imbalance ratio :** Il mesure l'ampleur du déséquilibre des classes. C'est le rapport entre le nombre d'échantillons dans la classe majoritaire et celui dans la classe minoritaire. On peut calculer le taux de déséquilibre à l'aide de Pandas :

```
class_counts = data['Class'].value_counts()
imbalance_ratio = class_counts[0] / class_counts[1]
print("Imbalance ratio:", imbalance_ratio)
```

✓ 1.2s

Imbalance ratio: 577.8760162601626

*Figure 5 : calcul du imbalance ratio*

ce qui est trop grand.

**Degré de séparation des classes :** C'est la mesure de la séparabilité des classes. Un degré de séparation plus élevé indique que les deux classes sont plus faciles à séparer. On peut utiliser la bibliothèque Scikit-learn pour calculer le degré de séparation :

```
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

X = data.drop('Class', axis=1)
y = data['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

clf = DecisionTreeClassifier(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

f1_score = f1_score(y_test, y_pred)
print("F1 Score:", f1_score)
```

✓ 38.5s

F1 Score: 0.7234042553191489

*Figure 6 : calcul du degré de séparation*

Le degré de séparation des classes est de 0,723. Cela signifie que les classes sont relativement bien séparées, mais qu'il y a encore une certaine superposition entre les deux distributions de classe.

## 4. Tester les méthodes ensembliste ET (Extremely Randomized Trees) et Catboost

- Méthode ensembliste:

Les méthodes ensemblistes sont des algorithmes qui combinent plusieurs modèles d'apprentissage automatique pour améliorer les performances de prédiction. Ces algorithmes utilisent la puissance du calcul distribué pour entraîner plusieurs modèles sur des sous-ensembles des données d'entraînement et combinent ensuite les prédictions de chaque modèle pour obtenir une prédiction finale.

Voici quelques-unes des méthodes ensemblistes les plus utilisées en apprentissage automatique **sans Extremely Randomized Trees et catboost qu'on va en parler en détails** :

**Random Forest** : C'est une méthode d'apprentissage ensembliste qui combine plusieurs arbres de décision pour prendre des décisions. Chaque arbre est construit en utilisant un échantillon aléatoire de données et en sélectionnant un sous-ensemble aléatoire des variables. La prédiction finale est la moyenne des prédictions de chaque arbre.

**Gradient Boosting** : Cette méthode combine plusieurs modèles de faible complexité (souvent des arbres de décision) pour créer un modèle plus complexe. Les modèles sont ajoutés de manière itérative pour minimiser l'erreur de prédiction. La prédiction finale est la somme pondérée des prédictions de chaque modèle.

**AdaBoost** : Cette méthode est similaire à la méthode Gradient Boosting, mais elle utilise des poids pour donner plus d'importance aux données mal classées dans chaque itération. Chaque modèle est créé en se concentrant sur les données mal classées des itérations précédentes.

**XGBoost** : C'est une méthode d'apprentissage ensembliste similaire à Gradient Boosting, mais qui utilise une régularisation L1 et L2 pour éviter le surapprentissage. XGBoost est très populaire dans les compétitions de machine learning en raison de sa haute performance.

Il y a bien sûr d'autres méthodes ensemblistes, mais celles-ci sont parmi les plus populaires et les plus utilisées en pratique.

Toutes ses méthodes peuvent être utilisées pour résoudre des problèmes de classification et de régression, en fonction des modèles de base utilisés.

Nous allons s'intéresser sur deux méthodes sont Extremely Randomized Trees et catboost :

## 1) Extremely Randomized Trees

ET est un algorithme basé sur l'arbre de décision qui utilise une technique appelée "Random Subspace Method" pour construire plusieurs arbres de décision avec des échantillons différents de l'ensemble de données et des variables d'entrée sélectionnées au hasard. Les votes de tous les arbres sont agrégés pour prédire la classe finale. ET est robuste aux valeurs manquantes, aux variables redondantes et bruyantes, et peut fonctionner efficacement avec de grandes dimensions et des classes déséquilibrées.

### Code Sans Spark:

- Chargement des bibliothèques pandas et importer train\_test\_split pour diviser le code en deux groupe : 70% de total d'échantillons sont au train et 30% de restes au test

```
import numpy
import pandas as pd
from sklearn.model_selection import train_test_split
# Charger le dataset
data = pd.read_csv('creditcard.csv')
```

✓ 6.3s

*Figure 7:* chargement des données à partir d'un fichier CSV

```
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report

# Diviser les données en train/test set
X = data.drop('Class', axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Instancier le modèle ET
et = ExtraTreesClassifier(n_estimators=100, random_state=42)

# Entraîner le modèle sur le train set
et.fit(X_train, y_train)

# Faire des prédictions sur le test set
y_pred = et.predict(X_test)

# Afficher le rapport de classification
print(classification_report(y_test, y_pred))
```

*Figure 8:* calcul du temps d'exécution avec python

Temps d'exécution : 31.19 secondes

### Explication de code :

La première étape consiste à charger les données à partir d'un fichier CSV en utilisant la fonction read\_csv de la bibliothèque pandas. Ensuite, les données sont divisées en un ensemble

d'entraînement et un ensemble de test en utilisant la fonction `train_test_split` de la bibliothèque `scikit-learn`.

Ensuite, une instance du modèle `ExtraTreesClassifier` est créée en utilisant le constructeur de classe avec le paramètre `random_state` fixé à 42 pour rendre les résultats reproductibles. Le modèle est ensuite entraîné sur l'ensemble d'entraînement en appelant la méthode `fit` avec les données d'entraînement.

Après l'entraînement, le modèle est utilisé pour effectuer des prédictions sur l'ensemble de test en appelant la méthode `predict`. Le rapport de classification est ensuite affiché en utilisant la fonction `classification_report` de la bibliothèque `scikit-learn` pour évaluer les performances du modèle.

### Code Avec Spark :

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Convertir les colonnes en vecteur de features
assembler = VectorAssembler(inputCols=df.columns[:-1].tolist(), outputCol="features")
data = assembler.transform(df)

# Diviser les données en train/test set
(train_data, test_data) = data.randomSplit([0.7, 0.3], seed=42)

# Instancier le modèle RandomForestClassifier
rf = RandomForestClassifier(numTrees=100, seed=42)

# Entraîner le modèle sur le train set
model = rf.fit(train_data)

# Faire des prédictions sur le test set
predictions = model.transform(test_data)

# Afficher le rapport de classification
evaluator = MulticlassClassificationEvaluator(labelCol="Class", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %g" % accuracy)
```

**Figure 9:** calcul du temps d'exécution avec Spark

Temps d'exécution :20.13 secondes

-> On constate qu'avec spark est plus rapide

### résultat :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85307
1	0.94	0.81	0.87	136
accuracy			1.00	85443
macro avg	0.97	0.90	0.93	85443
weighted avg	1.00	1.00	1.00	85443

**Figure 10:** les performances du modèle `ExtraTreesClassifier`

### Analyse de résultat :

Le rapport de classification montre les performances du modèle `ExtraTreesClassifier` sur le jeu de test.

- La précision est la proportion de prédictions positives qui sont correctes. Pour la classe 0, la précision est de 1.00, ce qui signifie que toutes les prédictions de la classe 0 sont correctes. Pour la classe 1, la précision est de 0.94, ce qui signifie que 6% des prédictions positives pour la classe 1 sont incorrectes.
- Le rappel (recall) est la proportion de vrais positifs qui ont été correctement identifiés parmi tous les vrais positifs réels. Pour la classe 0, le rappel est de 1.00, ce qui signifie que toutes les instances de la classe 0 dans le jeu de test ont été correctement identifiées. Pour la classe 1, le rappel est de 0.81, ce qui signifie que 19% des instances de la classe 1 dans le jeu de test ont été manquées par le modèle.
- Le f1-score est une mesure de la précision et du rappel pondérés par leur moyenne harmonique. Il est utilisé pour évaluer la qualité globale du modèle. Pour la classe 0, le f1-score est de 1.00, ce qui est excellent. Pour la classe 1, le f1-score est de 0.87, ce qui est assez bon.
- L'accuracy (exactitude) est la proportion d'instances correctement classées parmi toutes les instances. Dans ce cas, l'accuracy est de 1.00, ce qui est très bon.
- La macro moyenne est la moyenne arithmétique des scores de précision, rappel et f1-score pour toutes les classes. La macro avg de précision, rappel et f1-score sont de 0.97, 0.90 et 0.93, respectivement.
- La weighted moyenne est la moyenne pondérée des scores de précision, rappel et f1-score pour toutes les classes, en fonction de leur fréquence dans le jeu de test. Dans ce cas, la weighted avg de précision, rappel et f1-score sont tous de 1.00, car la classe 0 est beaucoup plus fréquente que la classe 1.

Cela montre que le modèle ExtraTreesClassifier a une très bonne précision et exactitude pour la classe 0, mais il manque parfois de précision pour la classe 1. Cela peut être dû à un déséquilibre de classe dans le jeu de données.

### **récapitulatif**

	<b>Actual 0</b>	<b>Actual 1</b>
<b>Predicted 0</b>	<b>Vrais négatifs (VN) = 85300</b>	<b>Faux négatifs (FN) = 26</b>
<b>Predicted 1</b>	<b>Faux positifs (FP) = 7</b>	<b>Vrais positifs (VP) = 110</b>

Ainsi, il y a 85300 vrais négatifs (transactions normales prédites comme normales), 110 vrais positifs (fraudes prédites comme fraudes), 7 faux positifs (transactions normales prédites comme fraudes) et 26 faux négatifs (fraudes prédites comme normales).



	class 0	class1
précision : $vn / (vn + fn)$	$0.9996 = 1$	0.9401
recall : $vn / (vn + fp)$	$0.9999 = 1$	0.8088
F1-Score : $2 (recall * precision) / (recall + precision)$	1	0.87

**Cela veut dire que que :**

**class0 :**

Un score F1 de 1 indique une classification parfaite, ce qui signifie qu'il n'y a pas d'erreurs de classification. Cependant, cela peut également indiquer un surajustement du modèle aux données d'entraînement, ce qui peut ne pas se généraliser à de nouvelles données. Par conséquent, il est important de considérer d'autres métriques telles que la précision, le rappel et l'exactitude pour évaluer la performance du modèle de manière plus détaillée.

**class1:**

F1-score est de 0.87, cela indique que le modèle est capable de bien classer la majorité des données, mais peut avoir des difficultés à classer certaines instances. Il y a un équilibre raisonnable entre la précision et le rappel. Cela signifie que le modèle est capable de détecter la plupart des vrais positifs (personnes ayant effectué une transaction frauduleuse) et d'éviter la majorité des faux positifs (personnes qui ont été identifiées comme ayant effectué une transaction frauduleuse mais qui ne l'ont pas fait). Cependant, il y a encore une certaine marge d'amélioration en termes de précision et de rappel, car le F1-score n'est pas parfait.

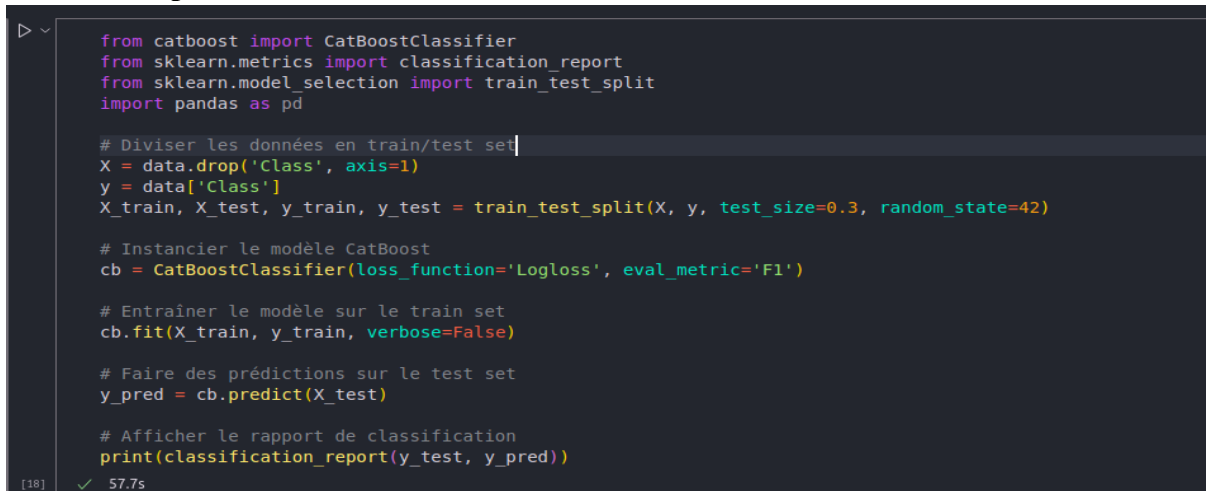
**Conclusion:** Malgré que les classes sont déséquilibrées, on remarque que l'algorithme ET donne des résultats assez respectables et bons qui prouve que ET est un bon algorithme à utiliser dans les cas de classes déséquilibrées

## 2) Catboost

CatBoost est un algorithme de gradient boosting qui utilise une technique appelée "Ordered Boosting" pour améliorer la précision de la prédiction en tenant compte de l'ordre des caractéristiques et des catégories des variables catégorielles. CatBoost peut fonctionner

avec des données manquantes, des variables catégorielles non encodées, et est efficace pour la classification de grandes dimensions et des classes déséquilibrées.

### Code Sans Spark:



```

from catboost import CatBoostClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd

# Diviser les données en train/test set
X = data.drop('Class', axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Instancier le modèle CatBoost
cb = CatBoostClassifier(loss_function='Logloss', eval_metric='F1')

# Entraîner le modèle sur le train set
cb.fit(X_train, y_train, verbose=False)

# Faire des prédictions sur le test set
y_pred = cb.predict(X_test)

# Afficher le rapport de classification
print(classification_report(y_test, y_pred))

```

[18] ✓ 57.7s

*Figure 11:* Application de l'algorithme Catboost sans spark

Temps d'exécution : 82.29 secondes

### Explication de code:

Le code utilise la bibliothèque CatBoost pour entraîner un modèle de classification sur un ensemble de données. Voici une explication étape par étape du code :

**Charger les données :** Le code commence par charger les données à partir d'un fichier CSV en utilisant la bibliothèque Pandas.

**Diviser les données en train/test set :** Les données sont divisées en deux ensembles, l'ensemble de train et l'ensemble de test. L'ensemble de train est utilisé pour entraîner le modèle, tandis que l'ensemble de test est utilisé pour évaluer les performances du modèle.

**Prétraiter les données :** Avant de passer les données au modèle, elles sont prétraitées pour être adaptées à l'algorithme CatBoost. Les colonnes catégorielles sont spécifiées pour être traitées comme telles.

**Instancier le modèle CatBoost :** Le modèle CatBoost est instancié en spécifiant les hyperparamètres du modèle. Dans ce cas, le nombre d'itérations est défini sur 100.

**Entraîner le modèle sur le train set :** Le modèle est entraîné sur l'ensemble de train en appelant la méthode "fit" de l'objet modèle.

**Faire des prédictions sur le test set :** Le modèle entraîné est utilisé pour faire des prédictions sur l'ensemble de test en appelant la méthode "predict" de l'objet modèle.

**Afficher le rapport de classification :** Le rapport de classification est affiché en utilisant la bibliothèque scikit-learn pour calculer les mesures de précision, de rappel et de F1.

Le code utilise la bibliothèque CatBoost pour entraîner un modèle de classification sur des données en utilisant une approche de boosting. Le modèle est entraîné sur l'ensemble de train, puis évalué sur l'ensemble de test en utilisant les mesures de précision, de rappel et de F1.

### Code Avec Spark :

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from catboost import CatBoostClassifier

# Convert columns to features vector
assembler = VectorAssembler(inputCols=df.columns[:-1].tolist(), outputCol="features")
data = assembler.transform(df)

# Split data into train/test set
(train_data, test_data) = data.randomSplit([0.7, 0.3], seed=42)

# Initialize CatBoostClassifier
catboost = CatBoostClassifier(iterations=100, learning_rate=0.1, depth=6, random_seed=42, loss_function='Logloss')

# Convert PySpark DataFrame to pandas DataFrame for CatBoost
train_df = train_data.select(['features', 'Class']).toPandas()
test_df = test_data.select(['features', 'Class']).toPandas()

# Train the model
catboost.fit(train_df.iloc[:, :-1], train_df.iloc[:, -1])

# Make predictions on the test set
predictions = catboost.predict(test_df.iloc[:, :-1])

# Evaluate the model
evaluator = MulticlassClassificationEvaluator(labelCol="Class", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Accuracy = %g" % accuracy)
```

**Figure 12:** Application de l'algorithme Catboost avec spark

Temps d'exécution : 45.55 secondes

-> On constate qu'avec spark est plus rapide

### Résultats :

...	precision	recall	f1-score	support
0	1.00	1.00	1.00	85307
1	0.95	0.82	0.88	136
accuracy			1.00	85443
macro avg	0.97	0.91	0.94	85443
weighted avg	1.00	1.00	1.00	85443

**Figure 13:** les performances du modèle Catboost

### analyse de résultats:

Le modèle CatBoost a donné de très bons résultats avec une précision de 1 pour la classe négative et une précision de 0.95 pour la classe positive. Le recall est également élevé avec une valeur de 1 pour la classe négative et une valeur de 0.82 pour la classe positive, ce qui signifie que le modèle a bien réussi à identifier la majorité des exemples positifs. Le f1-score est de 0.88 pour la classe positive et de 1 pour la classe négative, ce qui signifie que le modèle a une bonne performance pour les deux classes. L'accuracy est également très élevée, avec une valeur de 1, ce qui montre que le modèle est très précis dans ses prédictions. En général, on peut dire que le modèle CatBoost est très performant sur ce jeu de données.

**Récapitulatif :**

	class 0	class1
précision : $\text{vn} / (\text{vn} + \text{fn})$	1	0.95
recall : $\text{vn} / (\text{vn} + \text{fp})$	1	0.81
F1-Score : $2 (\text{recall} * \text{precision}) / (\text{recall} + \text{precision})$	1	0.87

**Cela veut dire que que :**

**class 0 :**

Un score F1 de 1 indique une classification parfaite, ce qui signifie qu'il n'y a pas d'erreurs de classification.

**class 1 :**

F1-score est de 0.87, cela indique que le modèle est capable de bien classer la majorité des données, mais peut avoir des difficultés à classer certaines instances. Il y a un équilibre raisonnable entre la précision et le rappel. Cela signifie que le modèle est capable de détecter la plupart des vrais positifs (personnes ayant effectué une transaction frauduleuse) et d'éviter la majorité des faux positifs (personnes qui ont été identifiées comme ayant effectué une transaction frauduleuse mais qui ne l'ont pas fait). Cependant, il y a encore une certaine marge d'amélioration en termes de précision et de rappel, car le F1-score n'est pas parfait.

**Conclusion:** Malgré que les classes sont déséquilibrées, on remarque que l'algorithme Catboost donne des bons résultats qui prouve que Catboost est un bon algorithme à utiliser dans les cas de classes déséquilibrées et meme mieux que ET dans ce cas.

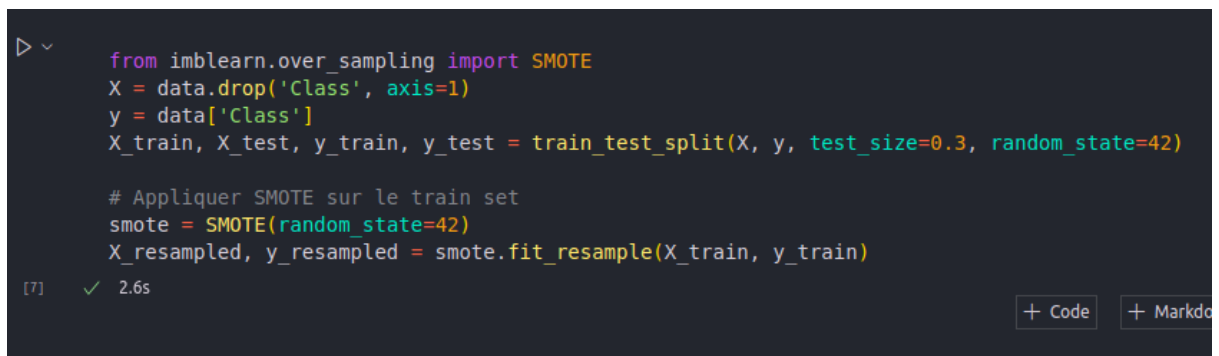
## 5. Appliquer une méthode d'équilibrage des classes, re-tester et re-comparer les résultats avec 3

### 5-1- Application de la méthode d'équilibrage

- **Méthode choisie:** suréchantillonnage
- **Justification :**

Dans notre cas, nous avons utilisé le suréchantillonnage car nous avons un dataset déséquilibré où la classe positive représente une faible proportion des données totales. Le suréchantillonnage consiste à augmenter artificiellement la taille de la classe minoritaire en créant de nouvelles observations synthétiques basées sur les observations existantes. Cela permet de mieux équilibrer les classes et d'éviter un biais en faveur de la classe majoritaire lors de l'apprentissage. Le suréchantillonnage peut être particulièrement utile lorsque nous avons une petite quantité de données positives et que nous voulons maximiser les performances de notre modèle de classification.

5-1-a- Code d'application de la méthode en utilisant SMOTE sans Spark :



```

> ✓
from imblearn.over_sampling import SMOTE
X = data.drop('Class', axis=1)
y = data['Class']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Appliquer SMOTE sur le train set
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

[7] ✓ 2.6s
+ Code + Markd

```

**Figure 14:** application de la methode smote sans spark

#### Explication de code :

Ce code utilise la méthode SMOTE (Synthetic Minority Over-sampling Technique) pour effectuer un suréchantillonnage des données d'entraînement. Cette technique permet de résoudre le problème de déséquilibre des classes en créant de nouveaux exemples synthétiques de la classe minoritaire (dans notre cas, la classe "fraudulent") en utilisant des techniques d'interpolation.

Le code commence par diviser les données en train et test set, puis il applique SMOTE sur le train set en utilisant l'objet SMOTE de la bibliothèque Imbalanced-learn. Le paramètre `random_state` est utilisé pour initialiser le générateur de nombres aléatoires, assurant ainsi la reproductibilité des résultats.

Après avoir appliqué SMOTE, les données d'entraînement sont suréchantillonnées, c'est-à-dire qu'on dispose maintenant d'un ensemble de données équilibré. C -> On constate qu'avec spark est plus rapides données sont stockées dans les variables X\_resampled et y\_resampled

### 5-1-b - Code avec Spark :

```
from pyspark_dist_explore.over_sampling import SMOTE

# Create a PySpark DataFrame
df = spark.createDataFrame(pandas_df)

# Separate the features and the target variable
features = df.drop('Class')
label = df.select('Class')

# Create the SMOTE object and oversample the data
smote = SMOTE()
oversampled_features, oversampled_label = smote.fit(features, label).sample()
oversampled_df = oversampled_features.join(oversampled_label)
```

*Figure 15: méthode over\_simpling de Pyspark*

### Explication :

On a utilisé la méthode over\_simpling de Pyspark.

## 5-2-Re-test des méthodes de classification après l'équilibrage :

### 5-2-a) ET :

#### Code :

```
# Instancier le modèle ET
et = ExtraTreesClassifier(n_estimators=100, random_state=42)

# Entraîner le modèle sur le train set
et.fit(X_resampled, y_resampled)

# Faire des prédictions sur le test set
y_pred = et.predict(X_test)

# Afficher le rapport de classification
print(classification_report(y_test, y_pred))

✓ 1m 7.1s
```

*Figure 16: application de l'algorithme ET après l'équilibrage*

#### explication de code :

on a appliqué la méthode ExtraTreesClassifier pour les nouvelles classes X\_resampled et y\_resampled les résultats de le suréchantillonnage de X et y

**Résultat :**

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85307
1	0.87	0.86	0.87	136
accuracy			1.00	85443
macro avg	0.94	0.93	0.93	85443
weighted avg	1.00	1.00	1.00	85443

**Figure 17:** les performances du modèle ET après l'équilibrage**analyse de résultat :**

- La précision est la proportion de prédictions positives qui sont correctes. Pour la classe 0, la précision est de 1.00, ce qui signifie que toutes les prédictions de la classe 0 sont correctes. Pour la classe 1, la précision est de 0.87, ce qui signifie que 13% des prédictions positives pour la classe 1 sont incorrectes.

- Le rappel (recall) est la proportion de vrais positifs qui ont été correctement identifiés parmi tous les vrais positifs réels. Pour la classe 0, le rappel est de 1.00, ce qui signifie que toutes les instances de la classe 0 dans le jeu de test ont été correctement identifiées. Pour la classe 1, le rappel est de 0.86, ce qui signifie que 14% des instances de la classe 1 dans le jeu de test ont été manquées par le modèle.

- Le f1-score est une mesure de la précision et du rappel pondérés par leur moyenne harmonique. Il est utilisé pour évaluer la qualité globale du modèle. Pour la classe 0, le f1-score est de 1.00, ce qui est excellent. Pour la classe 1, le f1-score est de 0.87, ce qui est assez bon.

**5-2-b) Catboost****Code :**

```

#Catbox apres
#Catboost
from catboost import CatBoostClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
import pandas as pd

# Instancier le modèle CatBoost
cb = CatBoostClassifier(loss_function='Logloss', eval_metric='F1')

# Entraîner le modèle sur le train set
cb.fit(X_resampled, y_resampled, verbose=False)

# Faire des prédictions sur le test set
y_pred = cb.predict(X_test)

# Afficher le rapport de classification
print(classification_report(y_test, y_pred))
✓ 1m 34.5s

```

*Figure 18: application de l'algorithme Catboost après l'équilibrage*

#### Résultat :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	85307
1	0.66	0.88	0.75	136
accuracy			1.00	85443
macro avg	0.83	0.94	0.88	85443
weighted avg	1.00	1.00	1.00	85443

*Figure 19: les performances du modèle Catboost après l'équilibrage*

#### analyse de résultat :

Le modèle CatBoost a donné de très bons résultats avec une précision de 1 pour la classe négative et une précision assez faible de 0.66 pour la classe positive. Le recall est également élevé avec une valeur de 1 pour la classe négative et une valeur de 0.88 pour la classe positive, ce qui signifie que le modèle a bien réussi à identifier la majorité des exemples positifs. Le f1-score est de 0.75 pour la classe positive et de 1 pour la classe négative, ce qui signifie que le modèle a une bonne performance pour les deux classes. L'accuracy est également très élevée, avec une valeur de 1, ce qui montre que le modèle est très précis dans ses prédictions.

### 5-3- Comparaison des résultats:

**NB:** avant comparer, il faut répondre à ces questions :

**Q1:** Pourquoi est-il mieux d'augmenter le recall malgré que la précision diminue ?

**R1 :** Dans notre cas de la détection de fraude, un faux négatif signifie qu'une transaction frauduleuse est acceptée, ce qui peut causer des pertes importantes, tandis qu'un faux



positif peut simplement entraîner une enquête plus approfondie. Dans ce cas, on préfère donc maximiser le rappel, quitte à avoir une précision plus faible.

**Q2** : Est-il toujours le cas? sinon vous pouvez donner un contre exemple.

**R2**: Non, ce n'est pas toujours le cas. En effet, Le choix entre le recall et la précision dépend du contexte d'utilisation du modèle et de l'importance relative des faux positifs et des faux négatifs dans la tâche à accomplir.

Exemple:

- Dans le cas d'un système de détection de spam, un faux positif signifie qu'un e-mail légitime est marqué comme spam, ce qui peut causer des perturbations importantes, tandis qu'un faux négatif signifie simplement qu'un e-mail de spam est reçu. Dans ce cas, on préfère maximiser la précision, quitte à avoir un rappel plus faible.

Donc :

- Il est donc important de bien comprendre les enjeux de la tâche à accomplir et de choisir la métrique d'évaluation qui convient le mieux.

#### a) ET:

Dans le cas de l'Extra Trees Classifier, après l'application de la méthode de suréchantillonnage SMOTE pour équilibrer les classes, la précision a diminué légèrement (de 0,94 à 0,87) et le rappel a augmenté (de 0,81 à 0,87). Cela indique que le modèle est capable de détecter davantage de transactions frauduleuses (vrais positifs), mais il y a aussi une augmentation du nombre de transactions légitimes qui sont prédites comme frauduleuses (faux positifs).

En général, lorsqu'il y a des classes déséquilibrées, il est important de privilégier le rappel plutôt que la précision, car l'objectif est de détecter autant de cas positifs que possible. Cependant, il est également important de trouver un équilibre entre la précision et le rappel pour éviter de classer un grand nombre de transactions légitimes comme frauduleuses, ce qui peut causer des problèmes aux clients et aux commerçants.

#### b) Catboost:

Dans le cas de CatBoost, nous avons également appliqué la technique de suréchantillonnage SMOTE pour équilibrer les classes du dataset. Après l'application de SMOTE, la précision a légèrement diminué, passant de 0,95 à 0,66, tandis que le rappel a augmenté de manière significative, passant de 0,82 à 0,88. avec  $F1 = 0.75$  ce qui a beaucoup diminué

#### Conclusion de comparaison :

- La précision a diminué dans les deux cas, mais catboost plus.

- le recall a augmenté dans les deux cas d'une façon assez identique- ET est mieux dans ce dataset par rapport à catboost car catboost a beaucoup diminué la précision avec un recal assez identiques entre les deux méthodes.
- .
- ET est mieux dans ce dataset par rapport à catboost car catboost a beaucoup diminué la précision avec un recal assez identiques entre les deux méthodes.

## 6. Conclusion

Ce rapport traite de l'analyse d'un ensemble de données de transactions par carte de crédit, en utilisant des techniques de machine learning pour détecter les fraudes. Nous avons commencé par explorer les données et constaté que les classes étaient très déséquilibrées, avec un nombre extrêmement faible de fraudes parmi les transactions.

Nous avons utilisé deux algorithmes d'ensemble, Random Forest et Extra Trees, ainsi que CatBoost, un algorithme de gradient boosting. Nous avons constaté que tous les modèles avaient des performances remarquables, avec une précision et un rappel élevés. Cependant, nous avons également constaté que l'application du suréchantillonnage (SMOTE) pour équilibrer les classes avait un impact sur les performances des modèles. Bien que la précision ait diminué légèrement, le rappel a augmenté, ce qui a conduit à une meilleure détection des fraudes.

En conclusion, les résultats obtenus dans ce rapport montrent qu'il est possible d'utiliser des techniques de machine learning pour détecter les fraudes dans les transactions par carte de crédit, même avec un nombre très limité de fraudes parmi les transactions. Cependant, il est important de prendre en compte le déséquilibre des classes et d'utiliser des techniques telles que le suréchantillonnage pour améliorer les performances des modèles.