الوكالة الوطنية لتقنين المواصلات

Agence Nationale de Réglementation des Télécommunications (ANRT)

المعهد الوطني للبريد و المواصلات

Institut National des Postes et Télécommunications (INPT)

Projet de Fin d'Année
Option : Ingénieur des Sciences de Données (Data Engineering)

Développement d'une application de détection de fraudes en utilisant la librairie de Machine Learning « MLlib » d'Apache Spark

Réalisé par :

Berrissoul Saad

Mouadi Mohamed Khalil

Nibgourine Younes

Année Universitaire : 2021 – 2022

Abstract:

Fraud detection is a set of activities undertaken to prevent money or property from being obtained through false pretenses. Thus, Financial fraud detection is a challenge that must be taken seriously. Although fraudulent transactions are rare and represent only a very small fraction of the activity within an organization. It still can quickly turn into significant financial losses and negative reputational effects on the organization.

It is for this reason that effective fraud detection mechanisms play an essential role in protecting the interests of organizations against these negative effects.

The primary objective of this work is to develop an application for detecting fraudulent transactions using the Apache Spark MLlib library, then compare the different fraud detection algorithms.

We used a synthetic dataset based on a sample of actual transactions pulled from a month's financial logs from a mobile money service implemented in an African country. We worked on four algorithms : Logistic Regression, Decision Tree, Random Forest and Gradient-Boosted Tree. Performance analysis proved that the Gradient Boosted Tree algorithm is the best among these algorithms in term of Recall.

Résumé:

La détection de la fraude est un ensemble d'activités effectuées par l’entreprise pour empêcher l'obtention d'argent ou de biens sous de faux prétextes.

Ainsi, la détection de la fraude financière est un défi qui doit être pris au sérieux.

Bien que les transactions frauduleuses soient rares et ne représentent qu'une très petite fraction de l'activité au sein d'une organisation.

Cela peut encore rapidement se transformer en pertes financières importantes et en effets négatifs sur la réputation de l'organisation. C'est pour cette raison que des mécanismes efficaces de détection des fraudes jouent un rôle essentiel dans la protection des intérêts des organisations contre ces effets malhonnêtes.

On a travaillé sur quatre algorithmes : Régression logistique, Arbre de décision, Forêt aléatoire et Gradient-Boosted Tree.

Les analyses des performances ont montré que l’algorithme Gradient Boosted Tree est le meilleur parmi ces algorithmes en terme de détections.

Table des matières:

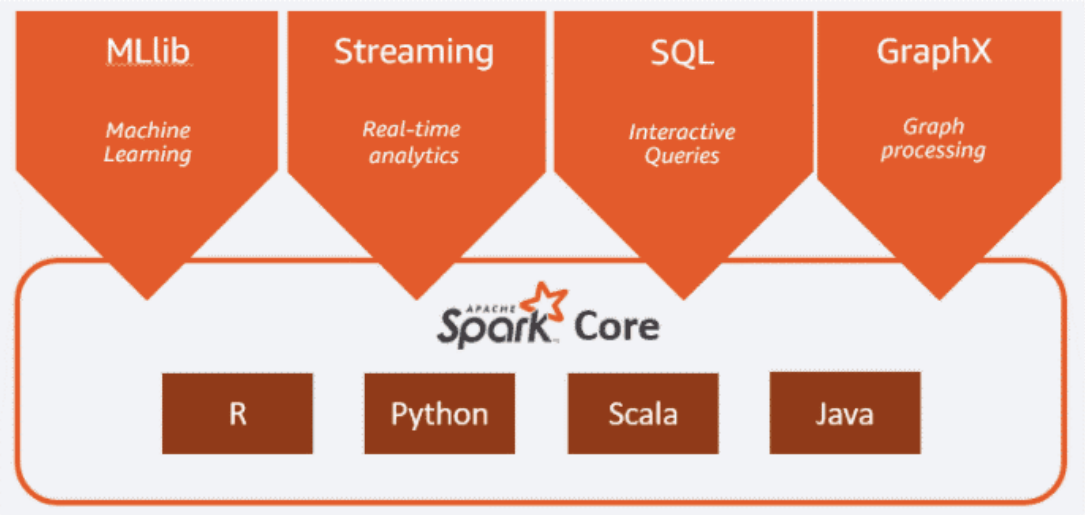
Introduction générale:

Les journaux parlent fréquemment de fraudes commises par des personnes et des organisations. Ces fraudes ont des répercussions énormes et variées. Avec l'explosion des données massives, les banques font désormais appel au Big Data et au Machine Learning pour détecter et prédire ces fraudes. La finalité de ce travail est de répondre à la problématique « Comment utiliser la librairie MLlib de machine learning d'Apache Spark pour détecter les fraudes bancaires ».

Afin de simplifier notre démarche, nous avons travaillé sur un problème d'apprentissage supervisé (supervised learning) et avec des données dont la variable à prédire « isFraud » est labellisée en fraude (1) ou non-fraude (0). Le travail s'effectue en Apache Spark à l'aide de la librairie MLlib de l'API Pyspark en utilisant le langage de programmation Python.

Apache Spark est un moteur de traitement de données rapide dédié au Big Data. Il permet d'effectuer un traitement de larges volumes de données de manière distribuée.

Apache Spark a été développé comme une alternative plus rapide à Hadoop, Hadoop-MapReduce lit et écrit les données à partir du disque, ce qui ralentit la vitesse de traitement. Alors que, Apache Spark stocke ces données en mémoire ce qui permet de réduire le cycle de lecture/écriture. Spark a été développé en Scala et propose des APIs pour Java, Python et R. Son architecture distribuée permet d'exécuter les algorithmes d'apprentissage automatique plus rapidement sans compromettre leurs performances.



Apache Spark framework (source: AWS)

Apache Spark MLlib est une bibliothèque open source de machine learning d'Apache Spark, Il offre un ensemble d'algorithmes d'apprentissage automatique pour la classification, la régression, le clustering…etc.

Dans cette étude, nous implémentons quatre algorithmes de classification d'apprentissage automatique supervisé à savoir la régression logistique, Arbre de décision, Forêt aléatoire et Gradient Boosted Tree..

Chapitre 1: Description de l'ensemble de données

1.1 Les données financières

Les ensembles de données financières sont importants pour de nombreux chercheurs et en particulier pour nous, dans ce travail de recherche sur la détection de fraudes financières. Une partie du problème est la nature intrinsèquement privée des transactions financières, qui conduit à l'absence d'ensembles de données accessibles au public.

Dans ce projet, nous avons utilisé un jeu de données synthétiques généré à l'aide d'un simulateur (Pysim) sur la base d'un échantillon de transactions réelles extraites d'un mois de journaux financiers d'un service d'argent mobile mis en œuvre dans un pays Africain. Les journaux d'origine ont été fournis par une société multinationale, qui est le fournisseur du service financier mobile qui fonctionne actuellement dans plus de 14 pays à travers le monde.

Cet ensemble de données synthétiques est réduit à 1/4 de l'ensemble de données d'origine et il est disponible sur Kaggle.

1.2 Description des variables

Par la suite, nous citons l'ensemble des variables ainsi que leurs descriptions, vu l'importance de bien les assimiler.

-**Step (String)**: c'est une unité de temps dans le monde réel, dans ce cas 1 step représente une heure (autrement dit 744 step représente 30 jours).

-**Type (String)**: c'est le type de la transaction faite, ça peut être CASH-IN, CASH-OUT, DEBIT, PAIEMENT, ou encore TRANSFER.

-**Amount (Double)**: c'est le montant de la transaction en monnaie locale.

-**NameOrig (String)**: c'est le nom du client qui a commencé la transaction.

-**OldbalanceOrig (Double)**: Solde initial avant la transaction.

-**NewbalanceOrig (Double)**: Nouveau solde après la transaction.

-**nameDest (String)**: Nom du client qui a reçu la transaction.

-**oldbalanceDest (Double)**: solde initial du destinataire avant qu'il reçoive la transaction. Notez qu'il n'y a pas d'informations pour les clients commençant par M (commerçants).

-**NewbalanceDest (Double)**: Nouveau solde du destinataire après avoir reçu la transaction.
Notez qu'il n'y a pas d'informations pour les clients commençant par M (commerçants).

-**IsFraud (Integer)**: Il s'agit des transactions effectuées par des agents frauduleux à l'intérieur de la simulation. En effet, dans cette base de données, ces agents profitent en contrôlant les comptes des clients, et par suite ils vident leurs fonds en transférant ces derniers à d'autres comptes, puis en les encaissant du système.
Cette variable est représentée par 1 si la transaction est frauduleuse, et 0 sinon.

-**IsFlaggedFraud (integer)**: Cette variable est représentée par 1 si la transaction est signalée frauduleuse et 0 sinon.
Sachez que le business model vise à contrôler les transferts massifs d'un compte à un autre et à signaler les tentatives illégales.
Une tentative illégale dans cette base de données est une tentative de transfert de plus de 200 000 en une seule transaction.

Chapitre 2: Prétraitement des données

Le prétraitement des données représente une étape primordiale dans le processus de la prédiction, et peut avoir la part du lion vu le temps qu'il nécessite.
En effet, le but du Machine Learning est la généralisation (autrement dit, prédire le comportement de nouvelles données et non pas celles avec lesquelles le modèle a été entraîné).
Et pour ce faire, il faut nettoyer nos données afin d'éviter toute confusion de notre modèle (le modèle peut détecter un schéma qui n'existe pas,on dit souvent que le modèle hallucine) causée par exemple par des valeurs aberrantes ou bien des fautes typographiques.
Dans cette analyse, nous nous servirons du framework pyspark afin de nettoyer nos données et maximiser nos chances d'obtenir un bon modèle qui généralise bien.
La machine ne travaille qu'avec des données numériques, donc il s'avère nécessaire de transformer les donnés catégorielles en données numériques.

2.1 Importer les données

Tout d'abord, nous importons les bibliothèques que nous utiliserons tout au long de notre projet, et nous créons un objet SparkSession qui nous permettra de télécharger notre base de données et de la transformer en une DataFrame.

Entrée [4]:
```
from pyspark.sql import SparkSession

# Create SparkSession object
spark = SparkSession.builder \
                    .master('local[*]') \
                    .getOrCreate()
```

Importing Data

Entrée [5]:
```
df = spark.read.csv('..\\Dataset\\credit.csv', sep=',',header=True)
```

Par la suite, nous voulons connaître les dimensions de notre base de données.

Entrée [7]:
```
# Records number
print('Records number : {}.'.format(df.count()))
```

```
Records number : 6362620.
```

Entrée [8]:
```
# Columns number
print('Columns number : {}'.format(len(df.columns)))
```

```
Columns number : 11
```

Notre base de données contient plus de 6 millions d'enregistrements et 11 colonnes (c'est-à-dire les variables), c'est une des raisons pour laquelle nous avons choisi d'utiliser spark.
Vu les capacités limitées de nos machines, nous travaillerons sur la moitié de cette base de données, pour cela nous avons effectué un tirage sans remise, en spécifiant que nous voulons juste la moitié des enregistrements.

Entrée [9]:
```
# The Dataset is so huge, so we will take only its half, we will select randomly 3M records
df = df.sample(False, 0.5, 42)
```

2.2 Traitement des données manquantes

Parfois, nous avons des données manquantes, suite à des fautes de frappe par exemple, donc il fallait les traiter, en éliminant les lignes ou une valeur manquante existe, ou en les remplaçant par une autre valeur que nous spécifions en étudiant le problème, ou parfois en supprimant toute la colonne.
Heureusement, dans cette base de données, nous n'avons pas de valeurs manquantes.

```
Entrée [11]: # Missing values
df.select([count(when(isnan(c),c)).alias(c) for c in df.columns]).show()
```

```
+----+----+------+--------+-------------+--------------+--------+--------------+--------------+-------+--------------+
|step|type|amount|nameOrig|oldbalanceOrg|newbalanceOrig|nameDest|oldbalanceDest|newbalanceDest|isFraud|isFlaggedFraud|
+----+----+------+--------+-------------+--------------+--------+--------------+--------------+-------+--------------+
|   0|   0|     0|       0|            0|             0|       0|             0|             0|      0|             0|
+----+----+------+--------+-------------+--------------+--------+--------------+--------------+-------+--------------+
```

```
Entrée [12]: # So the Dataset is clean, there is no missing values.
```

2.3 Traitement des types de données

Après avoir traiter les valeurs manquantes, nous passons maintenant aux types des variables, pour s'assurer de leurs homogénéité (par exemple elles doivent avoir la même unité etc..),et encore transformer ceux catégorielles en numérique. Pour ce faire, nous utilisons la méthode printShema().

```
Entrée [13]: # Schema
df.printSchema()
```

```
root
 |-- step: string (nullable = true)
 |-- type: string (nullable = true)
 |-- amount: string (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: string (nullable = true)
 |-- newbalanceOrig: string (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: string (nullable = true)
 |-- newbalanceDest: string (nullable = true)
 |-- isFraud: string (nullable = true)
 |-- isFlaggedFraud: string (nullable = true)
```

Nous remarquons que les variables suivantes: step,amount, oldbalanceorig, oldbalanceorig, oldbalanceDest, newbalanceDest, isFraud et isFlaggedFraud sont désormais être des string alors qu'elles sont censés être integer ou bien double.
Pour effectuer cette transformation, nous utilisons la méthode withcolumn qui prend en arguments la colonne sur laquelle nous désirons effectuer nos transformations et la transformation désirée.
Cette dernière a été réalisée à l'aide de la méthode cast.

```
Entrée [14]:
# columns transformation
from pyspark.sql.types import DoubleType

df = df.withColumn("step", df.step.cast(DoubleType()))
df = df.withColumn("amount", df.amount.cast(DoubleType()))
df = df.withColumn("oldbalanceOrg", df.oldbalanceOrg.cast(DoubleType()))
df = df.withColumn("newbalanceOrig", df.newbalanceOrig.cast(DoubleType()))
df = df.withColumn("oldbalanceDest", df.oldbalanceDest.cast(DoubleType()))
df = df.withColumn("newbalanceDest", df.newbalanceDest.cast(DoubleType()))
df = df.withColumn("isFraud", df.isFraud.cast('int'))
df = df.withColumn("isFlaggedFraud", df.isFlaggedFraud.cast('int'))
```

Nous nous assurons que les types sont ceux désirés.

```
Entrée [15]:
# Checking Data types
df.dtypes

Out[15]:
[('step', 'double'),
 ('type', 'string'),
 ('amount', 'double'),
 ('nameOrig', 'string'),
 ('oldbalanceOrg', 'double'),
 ('newbalanceOrig', 'double'),
 ('nameDest', 'string'),
 ('oldbalanceDest', 'double'),
 ('newbalanceDest', 'double'),
 ('isFraud', 'int'),
 ('isFlaggedFraud', 'int')]
```

Ensuite, il faut séparer la variable que nous voulons prédire du reste des variables prédictives.

```
Entrée [16]:
df = df.drop('isFlaggedFraud')

Entrée [17]:
# Predictors : step, type, amount, nameOrig, oldbalanceOrg, newbalanceOrig, oldbalanceDest, newbalanceDest
# Target : isFraud

Entrée [18]:
df = df.select('step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig', 'nameDest','oldbalanceDest', 'newbalan
```

Chapitre 3: L’analyse exploratoire des données

L’analyse exploratoire de données nous permet de dégager les caractéristiques et les aspects les plus intéressants de la structure de celles-ci. Dans cette étape, on cherche à mieux comprendre les données en calculant les différentes mesures récapitulatives, telles que la moyenne, la valeur maximale, la valeur minimale et variance…etc, et en faisant des visualisations sur l’ensemble de données. Et tout cela, a pour but de mieux choisir le modèle qui performera le mieux sur cette base de données sans être forcé d’en essayer beaucoup.

3.1 Statistique descriptive et visualisations

Nous commençons par par calculer les mesures récapitulatives pour les variables entières :

```
Entrée [19]: df.describe().show()
+-------+------------------+--------+------------------+-----------+-----------------+-----------------+-----------+--------------
----+------------------+--------------------+
|summary|              step|    type|            amount|   nameOrig|    oldbalanceOrg|   newbalanceOrig|   nameDest|    oldbalance
Dest|    newbalanceDest|             isFraud|
+-------+------------------+--------+------------------+-----------+-----------------+-----------------+-----------+--------------
----+------------------+--------------------+
|  count|           3182505| 3182505|           3182505|    3182505|          3182505|           3182505|    3182505|           318
2505|           3182505|             3182505|
|   mean|243.32051827098465|    null|180108.63530602978|       null|832749.6707201254| 853986.0712083996|       null|1104812.368802
8697|1229522.0882875763|0.001285779598146743|
| stddev| 142.3514799023828|    null| 603080.4075271396|       null|2884824.170356285|2920635.8624181743|       null|3427265.176649
3297|3700988.0592913437|0.035834714627960236|
|    min|               1.0| CASH_IN|               0.0|C1000001337|              0.0|               0.0|C1000004082|
0.0|               0.0|                   0|
|    max|             743.0|TRANSFER|     9.244551664E7| C999999614|    5.958504037E7|     4.958504037E7| M999999543|    3.55381433
61E8|     3.555534163E8|                   1|
+-------+------------------+--------+------------------+-----------+-----------------+-----------------+-----------+--------------
----+------------------+--------------------+
```

Ensuite nous désirons savoir la répartition de notre base de données entre transactions frauduleuses et autres non frauduleuses.

Entrée [21]:
```
# Count the occurrences of fraud and no fraud and print them
df.groupby('isFraud').count().show()
```

```
+-------+-------+
|isFraud|  count|
+-------+-------+
|      1|   4092|
|      0|3178413|
+-------+-------+
```

Entrée [22]:
```
# Print the ratio of fraud cases
occ = pd_df.isFraud.value_counts()
ratio = occ[1]/occ[0]
print('Ratio of fraud cases :',(ratio*100).round(3),'%')
```

```
Ratio of fraud cases : 0.129 %
```

Nous avons une base de données déséquilibrée, puisque la portion des transactions non frauduleuses ne représente que 0.12 pour cent de la totalité des données, ce qui est vraiment insuffisant, nous reviendrons vers ceci prochainement.

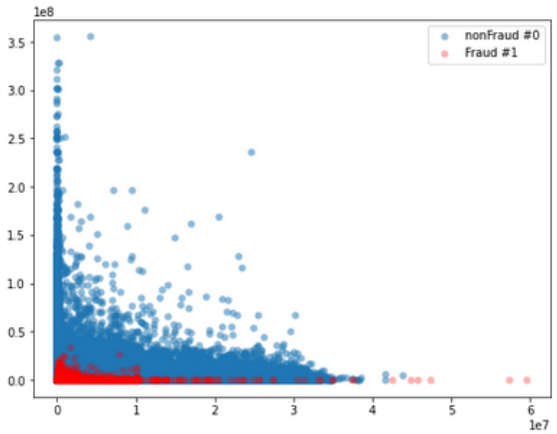
3.1.1 Scatter plot

Par la suite,nous souhaitons voir la répartition de nos données, pour cela nous avons effectué un scatter plot, dont deux variables (oldbalanceOrg et oldbalanceDest) sont représentées.

Entrée [24]:
```
# Define a function to create a scatter plot of our data and labels
def plot_data(X, y):
    plt.figure(figsize=(8,6))
    plt.scatter(X[y == 0].oldbalanceOrg, X[y == 0].oldbalanceDest, label="nonFraud #0", alpha=0.5, linewidth=0.15)
    plt.scatter(X[y == 1].oldbalanceOrg, X[y == 1].oldbalanceDest, label="Fraud #1", alpha=0.3, linewidth=0.15, c='r')
    plt.legend()
    return plt.show()
```

Entrée [25]:
```
# Create X and y from the prep_data function
X, y = prep_data(pd_df)
```

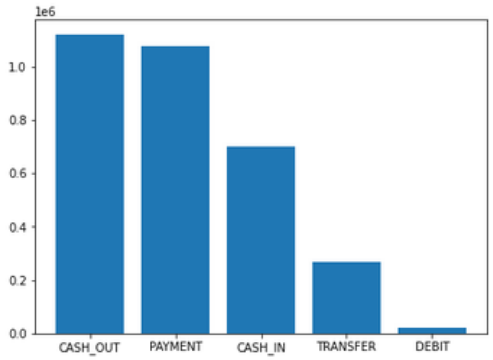
Entrée [26]:
```
# Plot our data by running our plot data function on X and y
plot_data(X,y)
```



Nous remarquons que les données non frauduleuses forment un groupe avec quelques données aberrantes, et la même chose pour les données frauduleuses.

3.1.2 Types des transactions

```
Entrée [27]: plt.figure(figsize=(7,5))
             plt.bar(pd_df['type'].value_counts().index, pd_df['type'].value_counts().values)
             plt.show()
```



Nous remarquons que la plupart des transactions sont sous la forme de Cash-out, Payment, Cash-in et Transfert alors qu'une minorité est sous la forme de débit. Donc une première réflexion serait ,par exemple,d'éliminer les lignes dont la transaction est de type débit, car le modèle n'aura pas suffisamment d'informations en déduire une conclusion.

3.1.3 Type des transactions frauduleuses

Fraudulent actions by transaction type :

Entrée [28]:
```
plt.figure(figsize=(7,5))
# Dataframe with only fraudulent actions
fraud = pd_df[pd_df['isFraud']==1]
sns.countplot(fraud['type'])
plt.show()
```

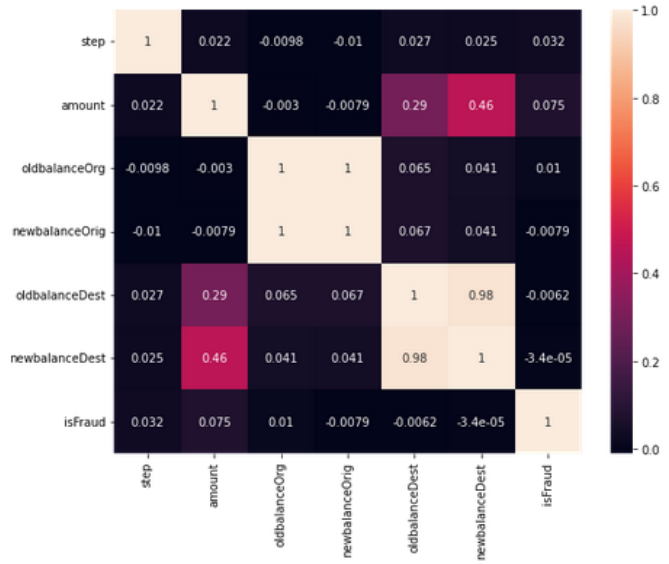
Nous remarquons que les actions frauduleuses sont uniquement dans les transactions de types cash_out et transfer pour les autres, il y'en a pas de fraudes. En plus, leur pourcentage est le même, 50 pour cent chacun.

3.1.4 Analyse de corrélation

Nous allons par suite générer une matrice de corrélation, qui représente les relations entre l'ensemble des variables numériques, pour ainsi étudier leurs dépendances, et si par exemple deux variables sont très dépendantes entre eux (coefficient de corrélation est à peu près égale à 1), nous pouvons éliminer une d'entre eux.

Correlation Matrix :

```
Entrée [29]: plt.figure(figsize=(9,7))
             sns.heatmap(pd_df.corr(), annot=True)
             plt.show()
```



Et vu la forte relation entre les deux variables oldbalanceOrg et newbalanceOrd, et aussi entre newbalanceOrig et oldbalanceOrig, nous avons choisi d'éliminer newbalanceOrig et newbalanceDest.

We see that there is many variables correlated to each other, so we will delete some of them

```
Entrée [30]: df = df.drop('newbalanceOrig', 'newbalanceDest')
             pd_df = df.toPandas()
```

3.2 Préparation des données

Nous commençons cette phase par une description des valeurs numériques, à savoir la moyenne, la variance, l'écart-type ainsi que le minimum et maximum de chaque variable.

Numerical Data :

```
Entrée [31]: num_cols = [ t[0] for t in df.dtypes if t[1] in ['double','int'] ]

Entrée [32]: df.select(num_cols).describe().toPandas()
```

Out[32]:

	summary	step	amount	oldbalanceOrg	oldbalanceDest	isFraud
0	count	3182505	3182505	3182505	3182505	3182505
1	mean	243.32051827098465	180108.63530602978	832749.6707201254	1104812.3688028697	0.001285779598146743
2	stddev	142.3514799023828	603080.4075271396	2884824.170356285	3427265.1766493297	0.035834714627960236
3	min	1.0	0.0	0.0	0.0	0
4	max	743.0	9.244551664E7	5.958504037E7	3.5538143361E8	1

Par la suite, nous traiterons les variables nameOrig et nameDest car elles sont de type string, alors que les modèles de machine learning de pyspark ne performe que sur les variables de type integer ou double (en général, les algorithmes de machine learning performe mieux sur les variables réelles).
Pour ce faire, nous allons séparer les caractères 'C' ou 'M' du reste des ces variables pour y laisser que des chiffres et par suite nous allons les convertir en double . Et à propos des deux caractères 'C' et 'M', nous utiliserons le oneHotEncoder pour les transformer en valeurs binaires.

Nous ferons la même chose (oneHotEncoder) pour la variable catégorielle type, donc ses valeurs seront représentées par les chiffres 0,1,2,3 et 4.

Separate nameOrig and nameDest :

Entrée [33]:
```
import pyspark.sql.functions as F
```

Entrée [34]:
```
data = df

data = (
    data.withColumn('str_orig', F.substring('nameOrig',1,1))
    .withColumn('num_orig',F.col('nameOrig').substr(F.lit(2), F.length('nameOrig') - F.lit(1)))
)

data = data.withColumn("num_orig", data.num_orig.cast(DoubleType()))

data = data.drop('nameOrig')

data.show(5)
```

```
+----+--------+-------+-------------+-----------+--------------+-------+--------+-------------+
|step|    type| amount|oldbalanceOrg|   nameDest|oldbalanceDest|isFraud|str_orig|     num_orig|
+----+--------+-------+-------------+-----------+--------------+-------+--------+-------------+
| 1.0|CASH_OUT|  181.0|        181.0|  C38997010|       21182.0|      1|       C| 8.40083671E8|
| 1.0| PAYMENT|7861.64|    176087.23| M633326333|           0.0|      0|       C|1.912850431E9|
| 1.0|   DEBIT|9644.94|       4465.0| C997608398|       10845.0|      0|       C|1.900366749E9|
| 1.0| PAYMENT|2560.74|       5070.0| M972865270|           0.0|      0|       C|1.648232591E9|
| 1.0| PAYMENT|1563.82|        450.0|M1731217984|           0.0|      0|       C| 7.61750706E8|
+----+--------+-------+-------------+-----------+--------------+-------+--------+-------------+
only showing top 5 rows
```

Entrée [35]:
```
data = (
    data.withColumn('str_dest', F.substring('nameDest',1,1))
    .withColumn('num_dest',F.col('nameDest').substr(F.lit(2), F.length('nameDest') - F.lit(1)))
)

data = data.withColumn("num_dest", data.num_orig.cast(DoubleType()))

data = data.drop('nameDest')

data.show(5)
```

```
+----+--------+-------+-------------+--------------+-------+--------+-------------+--------+-------------+
|step|    type| amount|oldbalanceOrg|oldbalanceDest|isFraud|str_orig|     num_orig|str_dest|     num_dest|
+----+--------+-------+-------------+--------------+-------+--------+-------------+--------+-------------+
| 1.0|CASH_OUT|  181.0|        181.0|       21182.0|      1|       C| 8.40083671E8|       C| 8.40083671E8|
| 1.0| PAYMENT|7861.64|    176087.23|           0.0|      0|       C|1.912850431E9|       M|1.912850431E9|
| 1.0|   DEBIT|9644.94|       4465.0|       10845.0|      0|       C|1.900366749E9|       C|1.900366749E9|
| 1.0| PAYMENT|2560.74|       5070.0|           0.0|      0|       C|1.648232591E9|       M|1.648232591E9|
| 1.0| PAYMENT|1563.82|        450.0|           0.0|      0|       C| 7.61750706E8|       M| 7.61750706E8|
+----+--------+-------+-------------+--------------+-------+--------+-------------+--------+-------------+
only showing top 5 rows
```

Entrée [35]:
```
data = (
    data.withColumn('str_dest', F.substring('nameDest',1,1))
    .withColumn('num_dest',F.col('nameDest').substr(F.lit(2), F.length('nameDest') - F.lit(1)))
)

data = data.withColumn("num_dest", data.num_orig.cast(DoubleType()))

data = data.drop('nameDest')

data.show(5)
```

```
+----+--------+-------+-------------+--------------+-------+--------+-------------+--------+-------------+
|step|    type| amount|oldbalanceOrg|oldbalanceDest|isFraud|str_orig|     num_orig|str_dest|     num_dest|
+----+--------+-------+-------------+--------------+-------+--------+-------------+--------+-------------+
| 1.0|CASH_OUT|  181.0|        181.0|       21182.0|      1|       C| 8.40083671E8|       C| 8.40083671E8|
| 1.0| PAYMENT|7861.64|    176087.23|           0.0|      0|       C|1.912850431E9|       M|1.912850431E9|
| 1.0|   DEBIT|9644.94|       4465.0|       10845.0|      0|       C|1.900366749E9|       C|1.900366749E9|
| 1.0| PAYMENT|2560.74|       5070.0|           0.0|      0|       C|1.648232591E9|       M|1.648232591E9|
| 1.0| PAYMENT|1563.82|        450.0|           0.0|      0|       C| 7.61750706E8|       M| 7.61750706E8|
+----+--------+-------+-------------+--------------+-------+--------+-------------+--------+-------------+
only showing top 5 rows
```

Applying OneHotEncoder to type, str_orig, str_dest

Entrée [36]:
```
from pyspark.ml.feature import StringIndexer
```

type column :

Entrée [37]:
```
indexer = StringIndexer(inputCol='type', outputCol='typeIndex')
data = indexer.fit(data).transform(data)
data = data.drop('type')
data.show(5)
```

```
+----+-------+-------------+--------------+-------+--------+-------------+--------+-------------+---------+
|step| amount|oldbalanceOrg|oldbalanceDest|isFraud|str_orig|     num_orig|str_dest|     num_dest|typeIndex|
+----+-------+-------------+--------------+-------+--------+-------------+--------+-------------+---------+
| 1.0|  181.0|        181.0|       21182.0|      1|       C| 8.40083671E8|       C| 8.40083671E8|      0.0|
| 1.0|7861.64|     176087.23|           0.0|      0|       C|1.912850431E9|       M|1.912850431E9|      1.0|
| 1.0|9644.94|       4465.0|       10845.0|      0|       C|1.900366749E9|       C|1.900366749E9|      4.0|
| 1.0|2560.74|       5070.0|           0.0|      0|       C|1.648232591E9|       M|1.648232591E9|      1.0|
| 1.0|1563.82|        450.0|           0.0|      0|       C| 7.61750706E8|       M| 7.61750706E8|      1.0|
+----+-------+-------------+--------------+-------+--------+-------------+--------+-------------+---------+
only showing top 5 rows
```

Entrée [38]:
```
data.groupBy('typeIndex').count().show()
```

```
+---------+-------+
|typeIndex|  count|
+---------+-------+
|      0.0|1119800|
|      1.0|1075637|
|      4.0|  20621|
|      3.0| 266850|
|      2.0| 699597|
+---------+-------+
```

str_orig column :

Entrée [39]:
```
indexer = StringIndexer(inputCol='str_orig', outputCol='nameOrigIndex')
data = indexer.fit(data).transform(data)
data = data.drop('str_orig')
data.show(5)
```

```
+----+-------+-------------+--------------+-------+-------------+--------+-------------+---------+-------------+
|step| amount|oldbalanceOrg|oldbalanceDest|isFraud|     num_orig|str_dest|     num_dest|typeIndex|nameOrigIndex|
+----+-------+-------------+--------------+-------+-------------+--------+-------------+---------+-------------+
| 1.0|  181.0|        181.0|       21182.0|      1| 8.40083671E8|       C| 8.40083671E8|      0.0|          0.0|
| 1.0|7861.64|     176087.23|           0.0|      0|1.912850431E9|       M|1.912850431E9|      1.0|          0.0|
| 1.0|9644.94|       4465.0|       10845.0|      0|1.900366749E9|       C|1.900366749E9|      4.0|          0.0|
| 1.0|2560.74|       5070.0|           0.0|      0|1.648232591E9|       M|1.648232591E9|      1.0|          0.0|
| 1.0|1563.82|        450.0|           0.0|      0| 7.61750706E8|       M| 7.61750706E8|      1.0|          0.0|
+----+-------+-------------+--------------+-------+-------------+--------+-------------+---------+-------------+
only showing top 5 rows
```

Entrée [40]:
```
data.groupBy('nameOrigIndex').count().show()
```

```
+-------------+-------+
|nameOrigIndex|  count|
+-------------+-------+
|          0.0|3182505|
+-------------+-------+
```

str_dest column :

Entrée [41]:
```
indexer = StringIndexer(inputCol='str_dest', outputCol='nameDestIndex')
data = indexer.fit(data).transform(data)
data = data.drop('str_dest')
data.show(5)
```

```
+----+-------+-------------+-------------+-------+-------------+-------------+---------+-------------+-------------+
|step| amount|oldbalanceOrg|oldbalanceDest|isFraud|     num_orig|     num_dest|typeIndex|nameOrigIndex|nameDestIndex|
+----+-------+-------------+-------------+-------+-------------+-------------+---------+-------------+-------------+
| 1.0|  181.0|        181.0|       21182.0|      1| 8.40083671E8| 8.40083671E8|      0.0|          0.0|          0.0|
| 1.0|7861.64|    176087.23|          0.0|      0|1.912850431E9|1.912850431E9|      1.0|          0.0|          1.0|
| 1.0|9644.94|       4465.0|       10845.0|      0|1.900366749E9|1.900366749E9|      4.0|          0.0|          0.0|
| 1.0|2560.74|       5070.0|          0.0|      0|1.648232591E9|1.648232591E9|      1.0|          0.0|          1.0|
| 1.0|1563.82|        450.0|          0.0|      0| 7.61750706E8| 7.61750706E8|      1.0|          0.0|          1.0|
+----+-------+-------------+-------------+-------+-------------+-------------+---------+-------------+-------------+
only showing top 5 rows
```

Entrée [42]:
```
data.groupBy('nameDestIndex').count().show()
```

```
+-------------+-------+
|nameDestIndex|  count|
+-------------+-------+
|          0.0|2106868|
|          1.0|1075637|
+-------------+-------+
```

3.3 Traitement du déséquilibre de la répartition des données

Nous avons déjà cité que la proportion des transactions frauduleuses est très faible, à savoir 0,12 pour cent de la totalité des données.
Alors pour remédier à ceci, nous allons utiliser une technique qui s'appelle Synthetic minority Oversampling Technique (SMOTE).
En effet, cette technique analyse les données de la classe dont la proportion est faible, et utilise le principe de l'algorithme nommé nearest neighbors pour générer des nouvelles données qui sont censées appartenir à cette classe, s'elles étaient dans la base de données.
Mais attention, il ne faut pas utiliser cette technique si les données sont lointaines l'une des autres, cela ne va introduire que du bruit qui va affecter négativement la généralisation.

Entrée [43]:
```
pd_data = data.toPandas()
```

Entrée [44]:
```
pd_data.head()
```

Out[44]:

	step	amount	oldbalanceOrg	oldbalanceDest	isFraud	num_orig	num_dest	typeIndex	nameOrigIndex	nameDestIndex
0	1.0	181.00	181.00	21182.0	1	8.400837e+08	8.400837e+08	0.0	0.0	0.0
1	1.0	7861.64	176087.23	0.0	0	1.912850e+09	1.912850e+09	1.0	0.0	1.0
2	1.0	9644.94	4465.00	10845.0	0	1.900367e+09	1.900367e+09	4.0	0.0	0.0
3	1.0	2560.74	5070.00	0.0	0	1.648233e+09	1.648233e+09	1.0	0.0	1.0
4	1.0	1563.82	450.00	0.0	0	7.617507e+08	7.617507e+08	1.0	0.0	1.0

Entrée [45]:
```
# Run the prep_data function
X,y = prep_data(pd_data)
```

Entrée [46]:
```
print(f'X shape: {X.shape}\ny shape: {y.shape}')
```

```
X shape: (3182505, 9)
y shape: (3182505,)
```

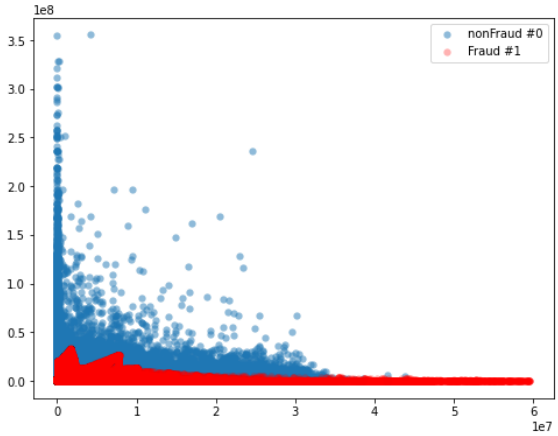
Entrée [47]:
```
# Define the resampling method
method = SMOTE()
```

Entrée [48]:
```
# Create the resampled feature set
X_resampled, y_resampled = method.fit_resample(X, y)
```

Entrée [49]:
```
y_resampled.value_counts()
```

Out[49]:
```
1    3178413
0    3178413
Name: isFraud, dtype: int64
```

Entrée [50]:
```
# Plot the resampled data
plot_data(X_resampled, y_resampled)
```

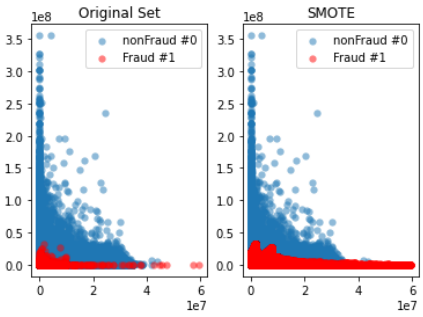


Pour voir l'efficacité de la méthode SMOTE, nous avons décidé de dessiner deux scatter plot, le premier qui représente les données de deux variables de la première base de données (avant SMOTE), et le deuxième celles de la nouvelle base de données (après SMOTE).

Compare SMOTE to original data

Entrée [51]:
```
def compare_plot(X, y, X_resampled, y_resampled, method):
    plt.subplot(1, 2, 1)
    plt.scatter(X[y == 0].oldbalanceOrg, X[y == 0].oldbalanceDest, label="nonFraud #0", alpha=0.5, linewidth=0.15)
    plt.scatter(X[y == 1].oldbalanceOrg, X[y == 1].oldbalanceDest, label="Fraud #1", alpha=0.5, linewidth=0.15, c='r')
    plt.title('Original Set')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.scatter(X_resampled[y_resampled == 0].oldbalanceOrg, X_resampled[y_resampled == 0].oldbalanceDest, label="nonFraud
    plt.scatter(X_resampled[y_resampled == 1].oldbalanceOrg, X_resampled[y_resampled == 1].oldbalanceDest, label="Fraud #1"
    plt.title(method)
    plt.legend()
    plt.show()
```

Entrée [52]:
```
compare_plot(X, y, X_resampled, y_resampled, method='SMOTE')
```



Chapitre 4: Créer une application de Machine Learning Apache Spark MLlib

4.1 La méthode traditionnelle de la détection des fraudes

La méthode traditionnelle consiste à fixer un seuil pour chaque variable, et si la transaction vérifie quelques conditions, alors on en déduit que c'est une fraude. En gros, c'est une méthode basée sur des lois que des personnes ont spécifiées.
Alors, il est évident que cette démarche est très limitée. En effet, elle est incapable de s'adapter avec le temps, au contraire de la méthode basée sur du Machine Learning qui s'adapte tant que des nouvelles données y arrivent.

4 - Exploring the traditional method of fraud detection :

Entrée [53]:
```
pd_data.groupby('isFraud').mean()
```

Out[53]:

isFraud	step	amount	oldbalanceOrg	oldbalanceDest	num_orig	num_dest	typeIndex	nameOrigIndex	nameDestIndex
0	243.157005	1.784843e+05	8.317111e+05	1.105574e+06	1.073637e+09	1.073637e+09	1.054533	0.0	0.33842
1	370.327224	1.441792e+06	1.639435e+06	5.131857e+05	1.073219e+09	1.073219e+09	1.496334	0.0	0.00000

Entrée [54]:
```
pd_data['flag_as_fraud'] = np.where(np.logical_and(pd_data.amount > 1.5e6, pd_data.oldbalanceOrg > 1.7e6), 1, 0)
```

Entrée [55]:
```
pd.crosstab(pd_data.isFraud, pd_data.flag_as_fraud, rownames=['Actual Fraud'], colnames=['Flagged Fraud'])
```

Out[55]:

Actual Fraud \ Flagged Fraud	0	1
0	3178384	29
1	3167	925

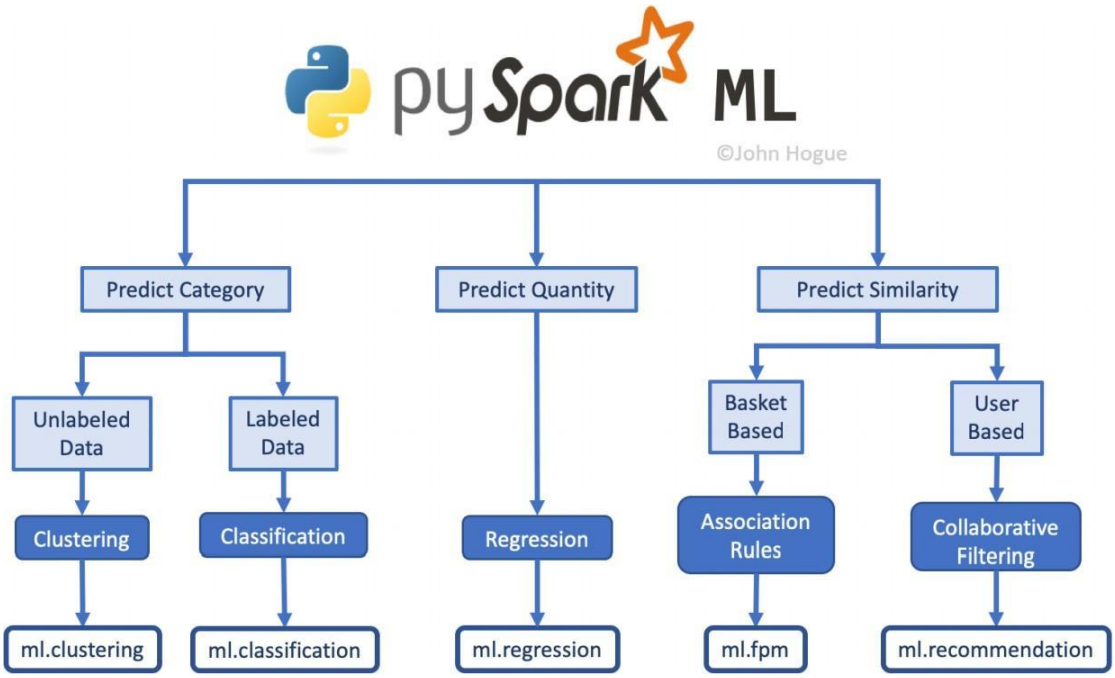
Nous voyons que cette démarche a un recall très faible, en effet elle n'a pu détecter 3167 transactions frauduleuses, ce qui est beaucoup, et par conséquent cette méthode manque beaucoup d'efficacité.

4.2 Choix des modèles d'apprentissage

Notre objectif est de construire des modèles d'apprentissage automatique pour prédire si une transaction est frauduleuse ou non. La colonne «isFraud» est notre variable cible (également appelée label) et le reste des colonnes sont les variables indépendantes (également appelées caractéristiques).

Nous avons des données étiquetées et la sortie de notre modèle est une valeur de la classe 0, 1, donc c'est un problème de classification supervisé.

Dans ce projet on se focalise sur cinq algorithmes parmi les algorithmes de classification supervisés de la librairie MLlib : Régression logistique, Arbre de décision, Forêt aléatoire, Gradient-Boosted Tree.



4.3 Préparation des données

Après avoir choisi les modèles que nous voulons utiliser, nous avons supprimer la variable cible qui est 'isFraud' et par la suite nous avons fusionné les variables prédictives en un seul vecteur que nous nommerons features.

5.2 - Vector Assembler :

Entrée [57]:
```
from pyspark.ml.feature import VectorAssembler
```

Entrée [58]:
```
print('Predictors :')
print(data.drop('isFraud').columns)
```

```
Predictors :
['step', 'amount', 'oldbalanceOrg', 'oldbalanceDest', 'num_orig', 'num_dest', 'typeIndex', 'nameOrigIndex', 'nameDestIndex
']
```

Entrée [59]:
```
assembler = VectorAssembler(
    inputCols = ['step', 'amount', 'oldbalanceOrg', 'oldbalanceDest', 'num_orig', 'num_dest', 'typeIndex', 'nameOrigIndex',
    outputCol = 'features'
)
```

Entrée [60]:
```
asm = assembler.transform(data)
```

Entrée [61]:
```
asm.select('features','isFraud').show(5)
```

```
+--------------------+-------+
|            features|isFraud|
+--------------------+-------+
|[1.0,181.0,181.0,...|      1|
|[1.0,7861.64,1760...|      0|
|[1.0,9644.94,4465...|      0|
|[1.0,2560.74,5070...|      0|
|[1.0,1563.82,450....|      0|
+--------------------+-------+
only showing top 5 rows
```

Ensuite nous allons renommer la variable cible par label, contrainte de MLlib.

Entrée [62]:
```
df = asm.select(
    F.col('features').alias('features'),
    F.col('isFraud').alias('label'),
)
```

Entrée [63]:
```
df.show(5, truncate=False)
```

```
+-----------------------------------------------------------------------+-----+
|features                                                               |label|
+-----------------------------------------------------------------------+-----+
|[1.0,181.0,181.0,21182.0,8.40083671E8,8.40083671E8,0.0,0.0,0.0]        |1    |
|[1.0,7861.64,176087.23,0.0,1.912850431E9,1.912850431E9,1.0,0.0,1.0]    |0    |
|[1.0,9644.94,4465.0,10845.0,1.900366749E9,1.900366749E9,4.0,0.0,0.0]   |0    |
|[1.0,2560.74,5070.0,0.0,1.648232591E9,1.648232591E9,1.0,0.0,1.0]       |0    |
|[1.0,1563.82,450.0,0.0,7.61750706E8,7.61750706E8,1.0,0.0,1.0]          |0    |
+-----------------------------------------------------------------------+-----+
only showing top 5 rows
```

Il est temps de diviser nos données en test_set et train_set. En effet, nous allons entraîner nos données sur la tain_set et les tester sur la test_set pour estimer l'erreur de généralisation.

```
Entrée [64]: # train test split :
             train, test = df.randomSplit([0.8, 0.2], seed=45)

Entrée [65]: [train.count(), test.count()]

   Out[65]: [2546764, 635741]
```

4.4 Préparer le modèle d'apprentissage automatique:

4.4.1 logistic regression

Dans cette partie on va expliquer comment implémenter la régression logistique, et ça sera le même workflow pour les autres algorithmes.

On instancier d'abord le modèle en passant en paramètre le label et le vecteur des features, puis nous faisons l'entraînement avec la méthode .fit() sur les données d'entraînement, et ensuite on va prédire les labels des données de test, pour que nous pouvons par suite comparer entre ce qui a été prédit et ce qui est vrai d'office.

5.3 - Logistic Regression :

```
Entrée [66]: from pyspark.ml.classification import LogisticRegression

Entrée [67]: logistic = LogisticRegression()
             lrModel = logistic.fit(train)

Entrée [68]: lr_predicted = lrModel.transform(test)

Entrée [69]: lr_predicted.groupBy('label','prediction').count().show()

+-----+----------+------+
|label|prediction| count|
+-----+----------+------+
|    1|       0.0|   791|
|    0|       0.0|634924|
|    1|       1.0|    23|
|    0|       1.0|     3|
+-----+----------+------+
```

Afin d'automatiser la tâche d'évaluation d'un modèle en se basant sur différents metrics nous avons créé une fonctionmodel_evaluation(predicted, model) qui prend en arguments les valeurs prédites par le modèle ainsi que le nom de ce dernier, et renvoie la précision, recall, f1_score ,AUC, AU_PR et la matrice de confusion.

Entrée [57]:
```
def model_evaluation(predicted, model):
    tp = predicted[(predicted.label==1) & (predicted.prediction==1)].count()
    fp = predicted[(predicted.label==0) & (predicted.prediction==1)].count()
    tn = predicted[(predicted.label==0) & (predicted.prediction==0)].count()
    fn = predicted[(predicted.label==1) & (predicted.prediction==0)].count()
    precision = tp/(tp + fp)
    recall = tp/(tp + fn)
    f1 = (2*precision*recall)/(precision+recall)

    print('------ '+model+' ------')
    print('precision_score :',round((precision*100),2),'%')
    print('recall_score :',round((recall*100),2),'%')
    print('f1 score :',round((f1*100),2),'%')

    evaluator = BinaryClassificationEvaluator()
    auc = evaluator.evaluate(lr_predicted, {evaluator.metricName : "areaUnderROC"})
    print("AUC score :",round((auc*100),2),"%")
    au_pr = evaluator.evaluate(lr_predicted, {evaluator.metricName : "areaUnderPR"})
    print("AU_PR score :",round((au_pr*100),2),"%")

    conf_mat = np.array([
        [tp, fn],
        [fp, tn]
    ])
    print('Confusion Matrix :')
    plt.imshow(conf_mat, vmax = fn+tp)
    for i in range(len(conf_mat)):
        for j in range(len(conf_mat[0])):
            plt.text(i,j,conf_mat[i][j])
    plt.colorbar()
    plt.show()
    return precision, recall, f1, auc, au_pr
```

Après avoir passé les valeurs prédites par la régression logistique ainsi que son nom,nous trouvons que la précision est assez bonne, mais à un recall très faible,c'est-à-dire qu'elle n'a détecté que 3 pour cent des transactions frauduleuses.

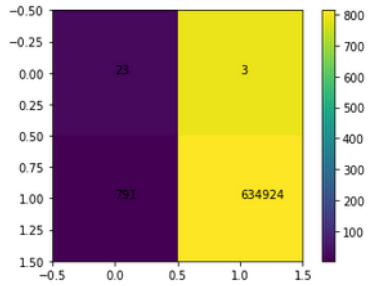
La surface sous la courbe(AUC) est élevée, et c'est normal car dans cette courbe, nous dessinons recall par rapport au FPR, donc cette courbe n'a traité que 3 pour cent des transactions frauduleuses,dont la précision du modèle était bonne.

A vrai dire, le choix de la régression logistique n'était pas pertinent.

Model evaluation :

Entrée [72]:
```
precision, recall, f1, auc, au_pr = model_evaluation(lr_predicted, 'Logistic Regression')
```

```
------ Logistic Regression ------
precision_score : 88.46 %
recall_score : 2.83 %
f1 score : 5.48 %
AUC score : 87.41 %
AU_PR score : 5.34 %
Confusion Matrix :
```



4.4.2 DecisionTree Classifier

5.4 - DecisionTree Classifier :

Entrée [75]:
```
from pyspark.ml.classification import DecisionTreeClassifier
```

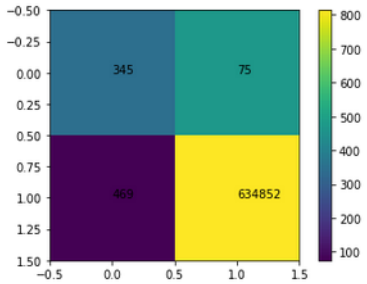
Entrée [76]:
```
dt = DecisionTreeClassifier()
dtModel = dt.fit(train)
```

Entrée [77]:
```
dt_predicted = dtModel.transform(test)
```

Model evaluation :

Entrée [78]:
```
precision, recall, f1, auc, au_pr = model_evaluation(dt_predicted, 'DecisionTree Classifier')
```

```
------ DecisionTree Classifier ------
precision_score : 82.14 %
recall_score : 42.38 %
f1 score : 55.92 %
AUC score : 87.41 %
AU_PR score : 5.34 %
Confusion Matrix :
```



Nous remarquons que la précision est élevée, et le recall est à peu près 43 pour cent, donc ce modèle n'a détecté que 43 pour cent des transactions frauduleuses,dont 83 pour cent sont classifiées dans la classe correcte.

4.4.3 RandomForest Classifier

5.5 - RandomForest Classifier :

Entrée [80]:
```
from pyspark.ml.classification import RandomForestClassifier
```

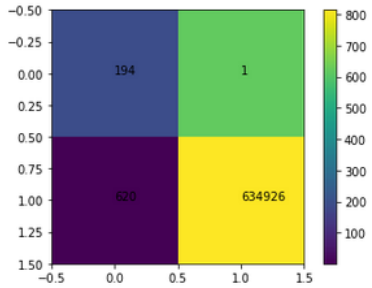
Entrée [81]:
```
rf = RandomForestClassifier(numTrees=30)
rfModel = rf.fit(train)
```

Entrée [82]:
```
rf_predicted = rfModel.transform(test)
```

Model evaluation :

Entrée [83]:
```
precision, recall, f1, auc, au_pr = model_evaluation(rf_predicted, 'RandomForest Classifier')
```

```
------ RandomForest Classifier ------
precision_score : 99.49 %
recall_score : 23.83 %
f1 score : 38.45 %
AUC score : 87.41 %
AU_PR score : 5.34 %
Confusion Matrix :
```



Nous remarquons que la précision est élevée, ce qui n'est pas étonnant car decisionTree Classifier avait une bonne précision,donc si on fait du bagging pour réduire la variance, ca serait encore mieux. Ensuite, le recall est à peu près 24 pour cent, donc ce modèle n'a détecté que 24 pour cent des transactions frauduleuses,dont 99 pour cent sont classifiées dans la classe correcte.

4.4.4 Gradient-boosted tree Classifier

5.6 - Gradient-boosted tree Classifier :

Entrée [85]:
```
from pyspark.ml.classification import GBTClassifier
```

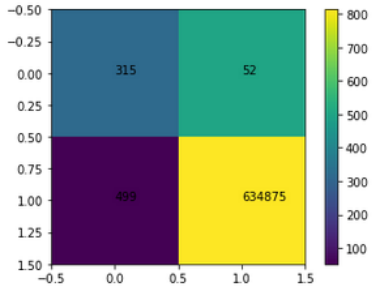
Entrée [86]:
```
gbt = GBTClassifier(maxIter=10)
gbtModel = gbt.fit(train)
```

Entrée [87]:
```
gbt_predicted = gbtModel.transform(test)
```

Model evaluation :

Entrée [88]:
```
precision, recall, f1, auc, au_pr = model_evaluation(gbt_predicted, 'GBT Classifier')
```

```
------ GBT Classifier ------
precision_score : 85.83 %
recall_score : 38.7 %
f1 score : 53.34 %
AUC score : 87.41 %
AU_PR score : 5.34 %
Confusion Matrix :
```



Nous remarquons que la précision est élevée, et le recall est à peu près 39 pour cent, car en effectuant du boosting, nous avons diminué le bias et par suite augmenter le recall et la précision, et nous pouvons avoir même mieux, si on augmente le max_iter , mais cela prend beaucoup du temps. Donc ce modèle n'a détecté que 39 pour cent des transactions frauduleuses, dont 86 pour cent sont classifiées dans la classe correcte.

4.5 Comparaison des modèles

6 - Models Comparaison :

Entrée [90]:
```
scores = pd.DataFrame(results, index = ['precision', 'recall', 'f1_score', 'auc', 'au_PR']).T
scores
```

Out[90]:

	precision	recall	f1_score	auc	au_PR
Logistic Regression	0.884615	0.028256	0.054762	0.874101	0.053413
Decision Tree	0.821429	0.423833	0.559157	0.874079	0.053413
Random Forest	0.994872	0.238329	0.384539	0.874094	0.053413
GBT	0.858311	0.386978	0.533446	0.874100	0.053413

Entrée [91]:
```
# Sort by recall :
scores.sort_values('recall', ascending=False)
```

Out[91]:

	precision	recall	f1_score	auc	au_PR
Decision Tree	0.821429	0.423833	0.559157	0.874079	0.053413
GBT	0.858311	0.386978	0.533446	0.874100	0.053413
Random Forest	0.994872	0.238329	0.384539	0.874094	0.053413
Logistic Regression	0.884615	0.028256	0.054762	0.874101	0.053413

Les modèles n'ont pas bien performé et ceci est dû au déséquilibre entre les portions des deux classes dans la base de données, mais recall ici est la mesure la plus importante, car nous voulons détecter le maximum possible des fraudes, même si on déclenche parfois des fausses alarmes, cela ne coûtera pas cher à l'entreprise ,ce qui est le cas de ne pas détecter une transaction qui est frauduleuse.

Donc le modèle le plus performant ici est Decision Tree avec un recall de 42 pour cent et une précision de 82 pour cent.

Conclusion

Pour conclure, nous ferons un récapitulatif de toute la démarche.
Alors, la base de données avec laquelle nous avons travaillé contient les labels, c'est pour cela que nous avons utilisé des algorithmes de supervised learning.
Ensuite, La finalité de ce projet était de prédire si une classification est frauduleuse ou pas, donc le problème est réduit à une classification binaire.
Alors après avoir décrit les variables prédictives pour bien assimiler le problème, nous avons effectué le prétraitement des données, ce qui s'avérait nécessaire, sinon cela sera simplement garbage in, garbage out.
Postérieurement, nous avons fait l'analyse exploratoire des données, qui consistait à faire une étude statistique(mean,var,std..),afin de voir par exemple l'intervalle de répartition des valeurs numériques,et si par exemple nous avons besoin du scaling..Par la suite nous avons effectué la visualisation des données, pour obtenir des idées afin de choisir un algorithme qui fonctionnera le mieux.
Ultérieurement nous avons converti les variables numériques en variables catégorielles, car la plupart des algorithmes fonctionne mieux sur les variables numériques.
Pour traiter le déséquilibre de la répartition des données entre transactions frauduleuses et non frauduleuses, nous avons utilisé la technique du SMOTE..
Et finalement, nous avons choisi quatre algorithmes ,que nous avons comparé et c'était decisionTree Classifier qui généraliser le mieux.
A vrai dire, les algorithmes d'apprentissage automatique fonctionnent généralement mieux lorsque les différentes classes contenues dans l'ensemble de données sont plus ou moins égales. S'il y a peu de cas de fraude, alors il y a peu de données pour apprendre à les identifier. Et c'est l'un des principaux défis de la détection de la fraude bancaire.

Références bibliographiques

[1] Nick Pentreath, « Machine Learning with Spark », Packt Publishing, Year : 2014

[2]Machine Learning Library (MLlib) Guide, https://spark.apache.org/docs/latest/ml-guide.html, Last retrieved on June 2021

[3]Conference Paper : IEEE Big Data 2017, « Big Data Machine Learning using Apache Spark MLlib », https://www.researchgate.net/publication/321149692_Big_Data_Machine_Learning_using_Apache_Spark_MLlib

[4]Edgar Lopez-Rojas, « Synthetic Financial Datasets For Fraud Detection », Synthetic datasets generated by the PaySim mobile money simulator, https://www.kaggle.com/ealaxi/paysim1, Last retrieved on April 2021

[5]Cem Dilmegani, « Synthetic Data Generation : Techniques, Best Practices & Tools », https://research.aimultiple.com/synthetic-data-generation/ January 13, 2021, Last retrieved on April 2021

[6]Dinara Rzayeva, Saber Malekzadeh, « Fraud Detection on Credit Card Transactions Using Machine learning », https://www.researchgate.net/publication/347487399_Fraud_Detection_on_Credit_Card_Transactions_Using_Machine_learning, December 2020, Last retrieved on May 2021

[7] Apache Spark, https://spark.apache.org/downloads.html, Last retrieved on April 2021

[8]Karlijn Willems, Apache Spark Tutorial : ML with PySpark, https://www.datacamp.com/community/tutorials/apache-spark-tutorial-machine-learning, Last retrieved on May 2021