

Développement d'un système de détection de fraudes en utilisant la librairie « MLlib » d'Apache Spark

Présenté par :

Bouchaara Oumayma.
Idhika Oumaima.

Encadré par :

Mme Zaidouni Dounia

Année universitaire :

2022/2023

Remerciements

Le travail présenté dans ce rapport a été effectué dans le cadre du projet de fin d'année .

En terme de ce projet, Nous tenons à exprimer notre profonde gratitude et notre immense respect à Mme Zaidouni , notre chère Enseignante chercheuse à l'institut national des postes et télécommunications pour sa disponibilité, ses avis éclairés, et ses judicieux conseils.

Dédicaces

Nous dédions ce travail à nos très Chers Parents . Tous les mots du monde ne sauraient exprimer l'immense amour que nous leur portons, ni la profonde gratitude que nous leur témoignons pour tous les efforts et les sacrifices qu'ils font pour notre instruction et notre bien-être.

Nous leur rendons hommage par ce modeste travail en guise de notre reconnaissance éternelle . Que Dieu les garde , les protège et leur accorde santé, bonheur et longue vie .

Nous dédions aussi ce travail à nos sœurs et frères pour leurs encouragements incessants et leur soutien tout au long de nos études , Que dieu les protège et leur offre la chance et le bonheur , à tous nos amis et collègues pour leurs aides précieuses et leur esprit de coopération appréciable .

Sommaire

Introduction générale	1
1.Présentation du projet	3
1.1 Problématique	4
1.2 Solution	4
1.3 Description du jeu de données	4
1.4 Spark et pySpark MLlib	5
2.Analyse exploratoire et modélisation	7
2.1 Analyse statistique et visualisations	8
2.2 Prétraitement des données	14
2.3 Prédiction et évaluation des différents algorithmes	15
2.4 Comparaison et choix du modèle	22
3.Deploiment par Streamlit	24
Conclusion et Perspectives	26

Introduction générale

Chaque année, la fraude coûte à l'économie mondiale plus de 5 000 milliards de dollars. En plus de faire payer un lourd tribut aux victimes individuelles, la fraude a un impact sur les entreprises sous la forme d'une perte de revenus et de productivité, ainsi que d'une réputation et de relations avec les clients endommagées. Les praticiens de l'IA sont en première ligne de cette bataille, construisant et déployant des modèles ML sophistiqués pour détecter les fraudes qui permettent aux organisations d'économiser des milliards de dollars chaque année.

Bien sûr, il s'agit d'une tâche difficile. La fraude prend de nombreuses formes et vecteurs d'attaque dans tous les secteurs, des services financiers aux soins de santé, en passant par les assurances, la technologie et les voyages. Vraiment, aucune organisation n'est à l'abri de la fraude dans le monde d'aujourd'hui, alors que les violations de données deviennent plus fréquentes dans les grandes entreprises, que les identifiants volés sont vendus sur le dark web et que les pirates ciblent tout, des infrastructures critiques aux chaînes d'hôpitaux locales, avec des ransomwares.

La persistance de la fraude s'explique en partie par la complexité de son identification dans le monde réel. Des modèles d'abus en constante évolution et des ensembles de données déséquilibrés peuvent facilement déjouer les systèmes rigides basés sur des règles. En raison de cette fragilité, les entreprises et les gouvernements ont depuis longtemps adopté les techniques de ML pour la détection des anomalies, la prévention de la fraude et d'autres domaines d'investissement dans la lutte contre les abus.

Présentation du projet

1.Problématique

Les transactions en ligne ont connu un essor considérable, tout comme les cas de fraude en ligne. Le réseau Consumer Sentinel Network géré par la FTC a reçu 3,2 millions de rapports d'usurpation d'identité et de fraude en ligne en 2019. Les fraudeurs étant de plus en plus habiles à trouver et à exploiter les failles des systèmes, la gestion de la fraude est devenue douloureuse pour le secteur bancaire et financier. Heureusement, l'apprentissage automatique pour la détection des fraudes est venu à la rescousse des organisations financières.

2.Solution

Notre finalité , à travers ce projet , est de pouvoir détecter ces fraudes en faisant appel aux outils Big Data et au Machine Learning surtout que les transactions sont reçus massivement et d'une manière quotidienne .

3.Description du jeu de données

Vu qu'il y a un manque d'ensembles de données publiques disponibles sur les services financiers, en particulier dans le domaine émergent des transactions d'argent mobile , On a eu recours à un jeu de données synthétique simulé grâce au logiciel Pysim , ur la base d'un échantillon de transactions réelles extraites des journaux financiers d'un mois d'un service d'argent mobile mis en place dans un pays africain. Les journaux originaux ont été fournis par une société multinationale, qui est le fournisseur du service financier mobile actuellement en place dans plus de 14 pays du monde entier.

Vous trouverez ci-joint le lien au dataset :

<https://www.kaggle.com/datasets/ealaxi/paysim1>

Ce jeu de données contient au delà de 6 millions de transactions dont les variables sont définies comme suit :

- step : correspond à une unité de temps dans le monde réel. Dans ce cas, 1 étape correspond à 1 heure de temps. Total des étapes 744 (simulation de 30 jours).
- type : type de transaction (CASH-IN, CASH-OUT, DEBIT, PAIEMENT et TRANSFERT).
- amount : montant de la transaction en monnaie locale.
- nameOrig : identifiant du client qui a commencé la transaction
- oldbalanceOrig : solde initial avant la transaction
- newbalanceOrig : nouveau solde après la transaction.
- nameDest : identifiant du destinataire de la transaction
- oldbalanceDest : solde initial du destinataire avant la transaction.
- newbalanceDest : nouveau solde du destinataire après la transaction.
- isFraud : on lui assigne 1 si la transaction est réellement frauduleuse , 0 sinon .
- isFlaggedFraud : on lui assigne 1 si le système traditionnel a pu détecter la transaction comme frauduleuse , 0 sinon .

4.Spark et pySpark MLlib



Spark est un framework de calcul distribué. Ce n'est donc pas un langage de programmation, c'est un ensemble d'outils informatiques écrits initialement en Scala, mais maintenant disponibles également en Python, R, Java et SQL.

Le calcul distribué consiste en la réalisation d'opérations sur des données qui ne sont pas stockées en un seul endroit, mais éparpillées au sein d'un réseau de différentes machines (un « cluster »). Un framework de calcul distribué comme Spark veille à la bonne mise en œuvre et l'orchestration de ces calculs, permettant ainsi d'assurer la cohérence des résultats.

Attention, Spark ne gère pas les données lui-même : il s'occupe uniquement du calcul en s'appuyant sur une infrastructure de données distribuées pour gérer leur stockage au sein du cluster. Hadoop est l'infrastructure de données distribuées avec laquelle Spark a été développé, il est cependant possible de compiler Spark avec des alternatives à Hadoop.

Apache Spark est livré avec une bibliothèque nommée MLlib pour effectuer des tâches d'apprentissage automatique à l'aide du cadre Spark. Puisqu'il existe une API Python pour Apache Spark, c'est-à-dire PySpark, vous pouvez également utiliser cette bibliothèque Spark ML dans PySpark. MLlib contient de nombreux algorithmes et utilitaires d'apprentissage automatique.

Analyse exploratoire et modélisation

1.Analyse statistique et visualisations

On importe tout d'abord les librairies qui nous sont indispensables et on crée une session spark à l'aide des commandes suivantes :

```
from pyspark.sql import SparkSession , SQLContext
from pyspark.sql.functions import col,isnan, when, count
import seaborn as sns
import matplotlib.pyplot as plt
import pyspark.sql.functions as f
```

```
Create spark session
```

```
spark = SparkSession.builder.master('local[*]').appName('Fraud Sys detection').getOrCreate()
```

On affiche le schéma de notre jeu de donnée sachant qu'il est important de vérifier les types de données avant de les traiter. Nous assurons ainsi l'exactitude et la cohérence de données.

```
data.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)
```

On constate , ainsi , que les types de données ne sont pas cohérents , et donc il faut les convertir aux types voulus , notamment convertir les colonnes amount , oldbalanceOrig , newBalanceOrig , oldbalanceDest , newbalanceDest de String au Float

```
cols_to_convert = ['step','amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest','isFraud','isFl
for col_name in cols_to_convert:
    data = data.withColumn(col_name, data[col_name].cast('float').alias(col_name))
data.printSchema()
root
 |-- step: float (nullable = true)
 |-- type: string (nullable = true)
 |-- amount: float (nullable = true)
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrg: float (nullable = true)
 |-- newbalanceOrig: float (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: float (nullable = true)
 |-- newbalanceDest: float (nullable = true)
 |-- isFraud: float (nullable = true)
 |-- isFlaggedFraud: float (nullable = true)
```

On calcule le nombre de lignes et de colonnes ainsi que les mesures statistiques pour l'ensemble de données :

```
print((data.count(), len(data.columns)))
```

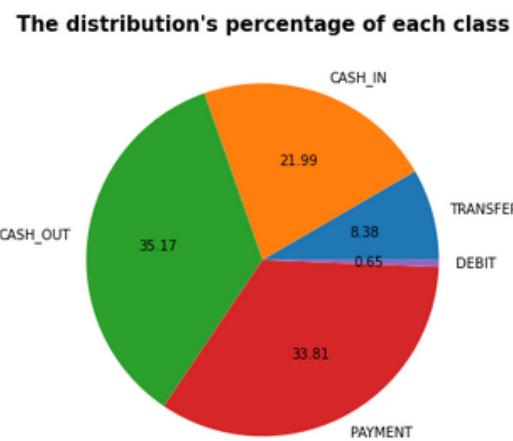
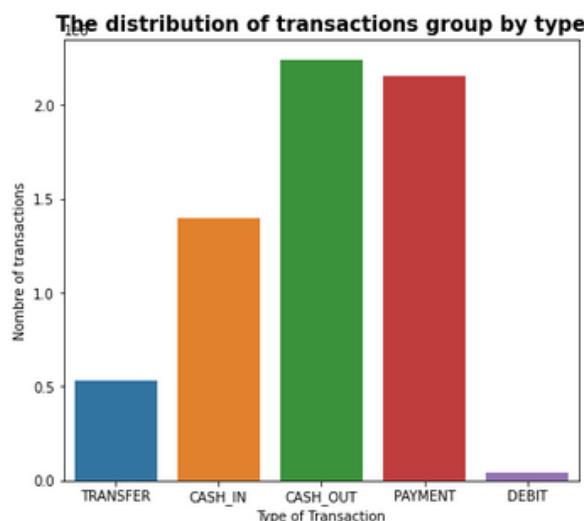
```
(6362620, 11)
```

```
data.describe(['amount','oldbalanceOrg','newbalanceOrig','oldbalanceDest','newbalanceDest']).show()
```

summary	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
count	6362620	6362620	6362620	6362620	6362620
mean	179861.9035579989	833883.1040482761	855113.6685519279	1100701.6665156623	1224996.398184315
stddev	603858.2316772637	2888242.6729989178	2924048.502910317	3399180.1117577194	3674128.941013759
min	0.0	0.0	0.0	0.0	0.0
max	9.244552E7	5.958504E7	4.958504E7	3.56015904E8	3.56179264E8

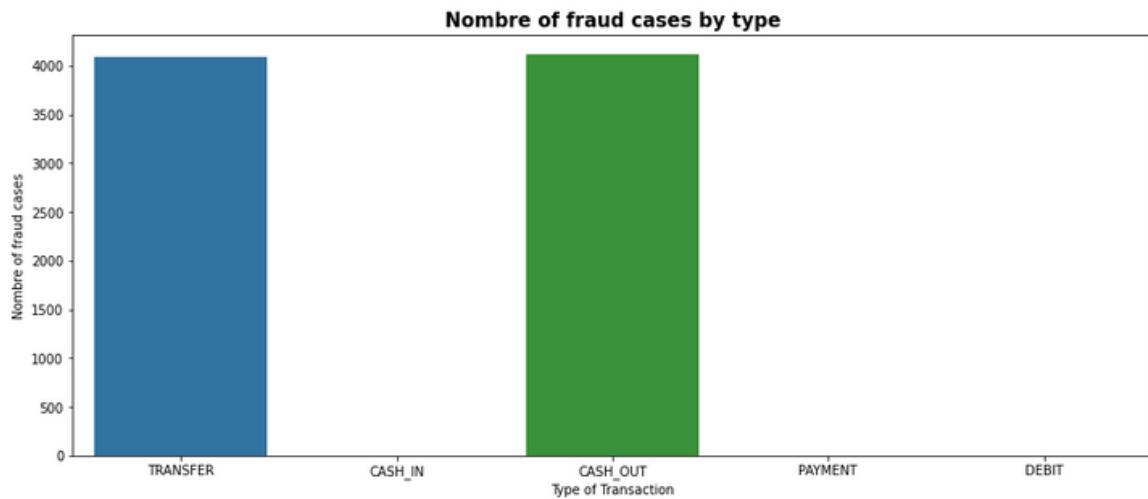
	type	count
0	TRANSFER	532909
1	CASH_IN	1399284
2	CASH_OUT	2237500
3	PAYMENT	2151495
4	DEBIT	41432

On visualise le nombre de transactions par type :
on constate que la plupart des transactions présentes dans notre jeu de données sont de type CASH_OUT et PAYEMENT



On affiche aussi la somme des transactions qui sont réellement frauduleuses par type de transaction :

	type	sum(isFraud)
0	TRANSFER	4097.0
1	CASH_IN	0.0
2	CASH_OUT	4116.0
3	PAYMENT	0.0
4	DEBIT	0.0



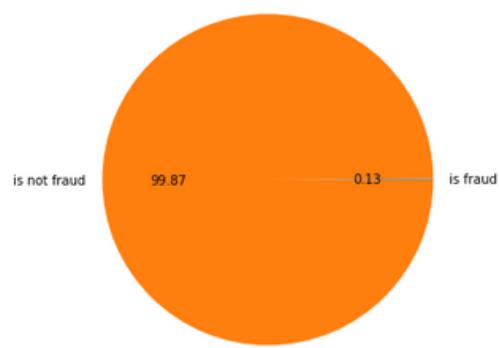
On constate bel et bien que les transactions frauduleuses soit toutes soit de type TRANSFER ou de type CASH_OUT .

On affiche ci-dessous le pourcentage des transactions (frauduleuses et non frauduleuses) :

isFraud	count
0	1.0 8213
1	0.0 6354407

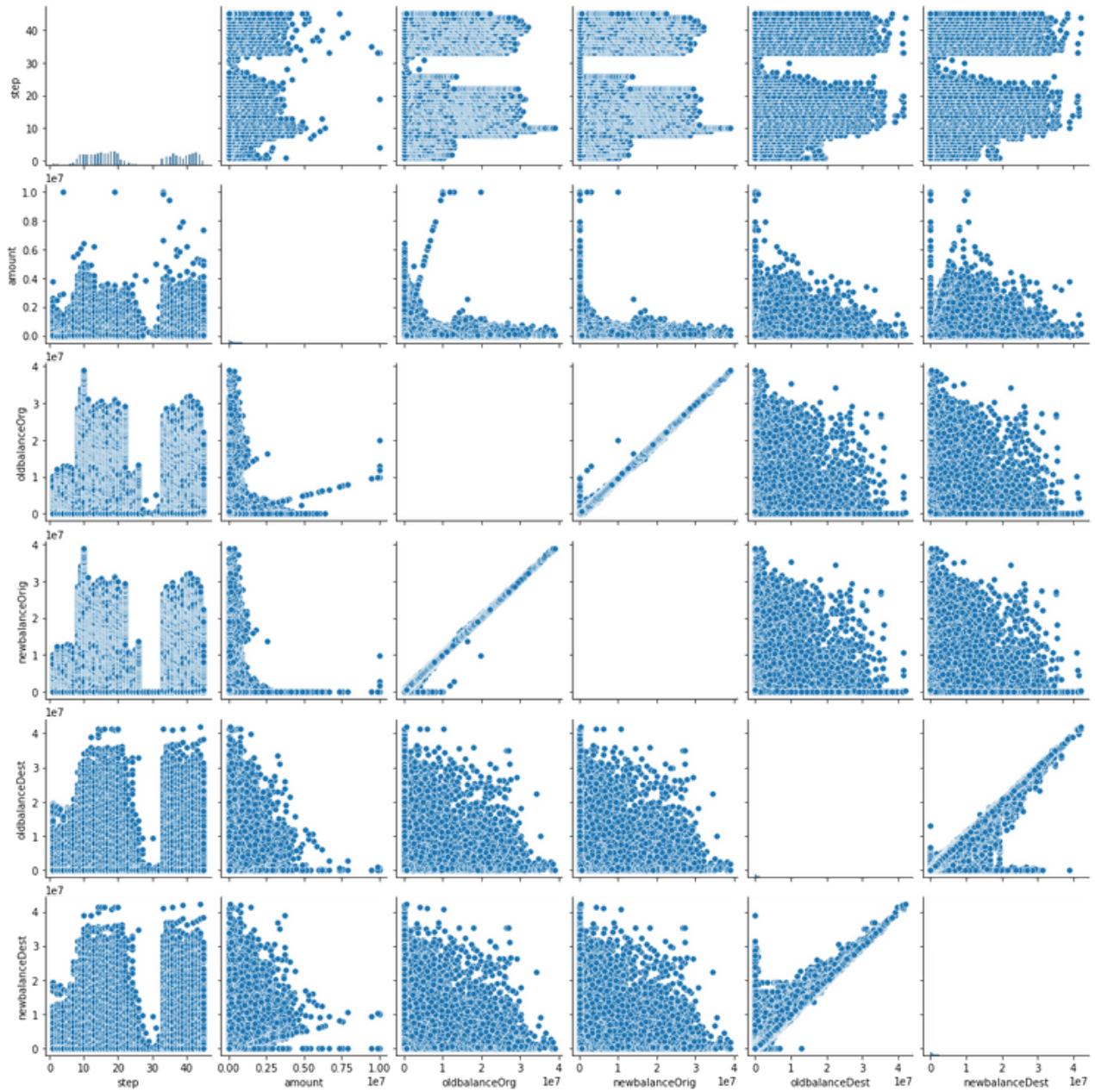
```
plt.figure(figsize=(15,6))
plt.pie(fraud_count['count'], labels=['is fraud' , 'is not fraud'], autopct='%.2f')
plt.title('The distribution's percentage of each class', fontsize=15, fontweight='bold')
plt.show()
```

The distribution's percentage of each class

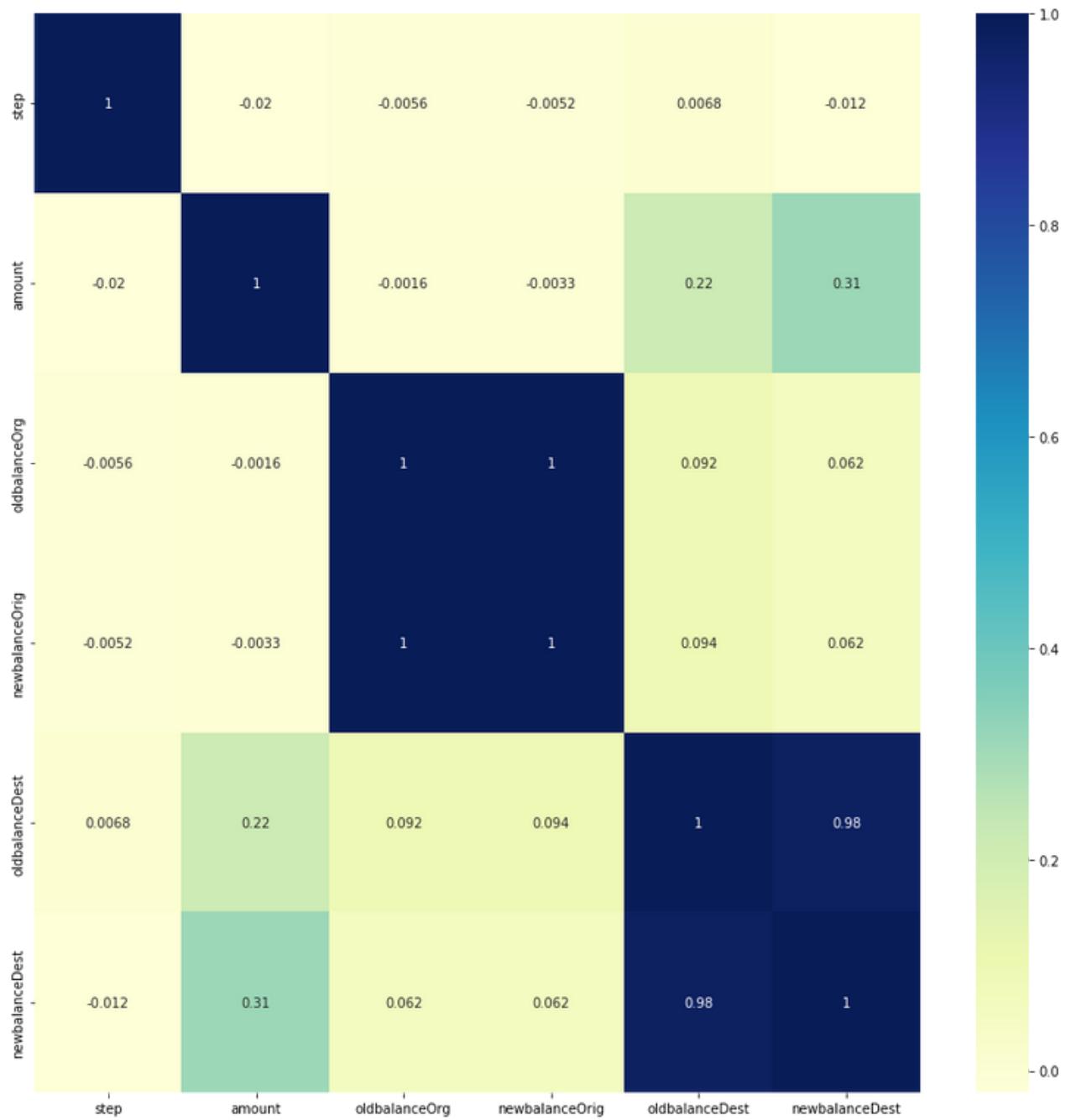


Les transactions frauduleuses ne constituent que 0.13% de notre data ce qui montre qu'elle est "imbalanced"

La figure suivante montre la corrélation entre chacun des variable de notre jeu de données :



On affiche aussi la matrice de corrélation entre toutes les variables du dataset :



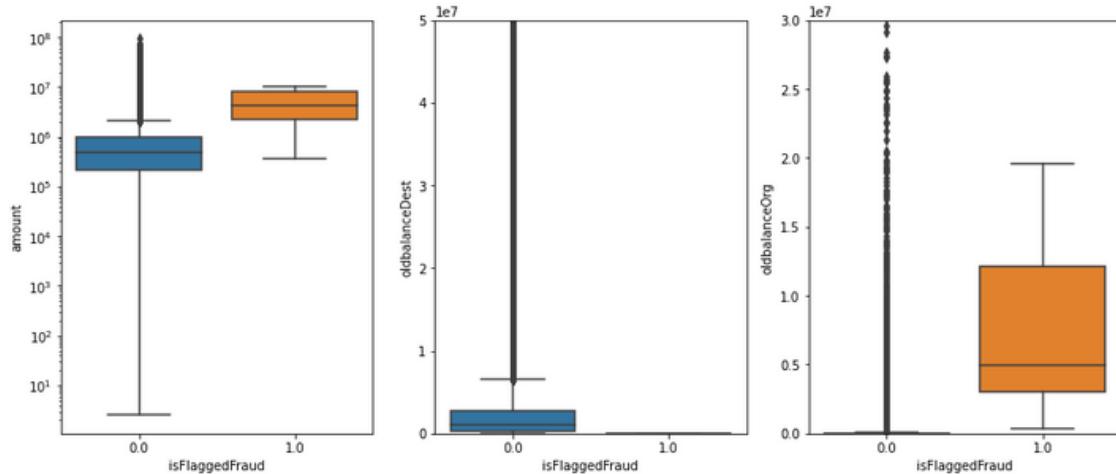
On voit qu'il y a une forte corrélation entre les colonnes oldbalanceOrig et newbalanceOrig d'un coté et entre oldbalanceDest newbalanceDest d'un autre coté ce qui est bien évident .

On fait finalement une comparaison entre les transactions de type TRANSFER détectées par le système traditionnel et celles non détectées grâce à des boxplots comme suit : on voit clairement que seules les transactions de grandes sommes à partir d'un compte vide vers un compte contenant une large somme d'argent qui sont détectés par le système traditionnel ce qui prouve sa limitation .

```
transferdata = data.filter(f.col('type')=='TRANSFER').toPandas()

fig, axs = plt.subplots(1, 3, figsize=(15, 6), squeeze=False)
a = sns.boxplot(x = 'isFlaggedFraud', y = 'amount', data = transferdata, ax=axs[0][0])
axs[0][0].set_yscale('log')
b = sns.boxplot(x = 'isFlaggedFraud', y = 'oldbalanceDest', data = transferdata , ax=axs[0][1])
axs[0][1].set(ylim=(0, 0.5e8))
c = sns.boxplot(x = 'isFlaggedFraud', y = 'oldbalanceOrg', data=transferdata, ax=axs[0][2])
axs[0][2].set(ylim=(0, 3e7))

plt.show()
```



On s'assure après qu'on n'a ni valeur manquante ni ligne dupliquée pour commencer notre prétraitement :

Count null values

```
data.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in data.columns]).show()
+-----+-----+-----+-----+-----+-----+-----+-----+
|step|type|amount|nameOrig|oldbalanceOrg|newbalanceOrig|nameDest|oldbalanceDest|newbalanceDest|isFraud|isFlaggedFraud|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0| 0|
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

Count duplicate records

```
data.groupBy(data.columns).count().where(f.col('count') > 1).select(f.sum('count')).show()
+-----+
|sum(count)|
+-----+
|    null|
+-----+
```

1. Prétraitemet des données

On élimine les colonnes qui ne sont pas importantes et puis on supprime tous les transactions qui ne sont pas de type TRANSFER et CASH_OUT (puisque ce sont les seuls à être détectées comme fraudes)

```
cols = ("step","isFlaggedFraud","nameOrig","nameDest")
data = data.drop(*cols)

data.select('type').distinct().collect()
[Row(type='TRANSFER'),
 Row(type='CASH_IN'),
 Row(type='CASH_OUT'),
 Row(type='PAYMENT'),
 Row(type='DEBIT')]

data = data.filter((data.type=="TRANSFER") | (data.type=="CASH_OUT"))
```

Après on vectorise la colonne de type de transactions pour la convertir en colonne numérique

```
from pyspark.ml.feature import StringIndexer
type_indexer = StringIndexer(inputCol="type", outputCol="numeric_type")
data = type_indexer.fit(data).transform(data)
data = data.drop('type')
data.show(5)

+-----+-----+-----+-----+-----+
| amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|isFraud|numeric_type|
+-----+-----+-----+-----+-----+
| 181.0|     181.0|       0.0|      0.0|      0.0|    1.0|      1.0|
| 181.0|     181.0|       0.0|  21182.0|       0.0|    1.0|      0.0|
| 229133.94|   15325.0|       0.0|    5083.0|  51513.44|    0.0|      0.0|
| 215310.3|     705.0|       0.0|  22425.0|       0.0|    0.0|      1.0|
| 311685.88|   10835.0|       0.0|    6267.0|  2719173.0|    0.0|      1.0|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

on prépare les données en rassemblant les colonnes "Features" ou "inputs" , et après on les normalise grâce à MinMaxScaler , on obtient ce qui suit :

```
+-----+-----+-----+-----+-----+-----+-----+
| amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|isFraud|numeric_type|      features|
scaledFeatures|
+-----+-----+-----+-----+-----+-----+-----+
| 181.0|     181.0|       0.0|      0.0|      0.0|    1.0|      1.0| 1.0|[181.0,181.0,0.0,...|[1.9
5790991277889...|
| 181.0|     181.0|       0.0|  21182.0|       0.0|    1.0|      0.0|[181.0,181.0,0.0,...|[1.9
5790991277889...|
| 229133.94|   15325.0|       0.0|    5083.0|  51513.44|    0.0|      0.0|[229133.9375,1532...|[0.0
0247858346732...|
| 215310.3|     705.0|       0.0|  22425.0|       0.0|    0.0|      1.0|[215310.296875,70...|[0.0
0232905063301...|
+-----+-----+-----+-----+-----+-----+-----+
only showing top 4 rows
```

on divise nos données entre 90% pour "train data" et 10% pour "test data "

```
(train, test )= df.randomSplit([0.9, 0.1], seed=2022)

train.show(5)

+-----+-----+-----+-----+-----+-----+
|amount|oldbalanceOrg|newbalanceOrig|oldbalanceDest|newbalanceDest|isFraud|numeric_type|      features|
+-----+-----+-----+-----+-----+-----+
| 0.37|    25032.0|   25031.63|       0.0|       0.0|     0.0|[0.37000000476837...
| 1.58|        0.0|       0.0| 197938.48|   70090.2|     0.0|[1.58000004291534...
| 1.65|        0.0|       0.0| 6653022.5|  6653024.5|     0.0|[1.64999997615814...
| 1.9|   18399.0|   18397.1| 1.1306069E7| 1.1306071E7|     0.0|[1.89999997615814...
| 2.05|        0.0|       0.0| 243703.05|  243705.1|     0.0|[2.04999995231628...
+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

train.groupBy('isFraud').count().show(truncate = False)

+-----+-----+
|isFraud|count |
+-----+-----+
|1.0    |7403   |
|0.0    |2486403|
+-----+-----+
```

3. Prédiction et évaluation des différents algorithmes

- **Logistic regression**

La régression logistique est un modèle statistique permettant d'étudier les relations entre un ensemble de variables qualitatives X_i et une variable qualitative Y . Il s'agit d'un modèle linéaire généralisé utilisant une fonction logistique comme fonction de lien.

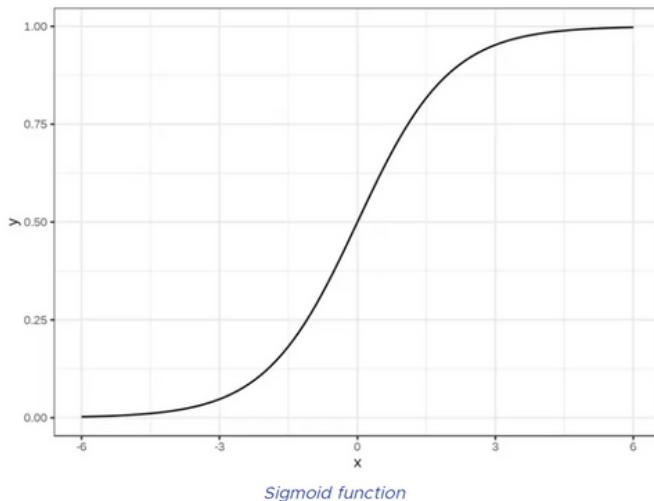
Un modèle de régression logistique permet aussi de prédire la probabilité qu'un événement arrive (valeur de 1) ou non (valeur de 0) à partir de l'optimisation des coefficients de régression. Ce résultat varie toujours entre 0 et 1. Lorsque la valeur prédictive est supérieure à un seuil, l'événement est susceptible de se produire, alors que lorsque cette valeur est inférieure au même seuil, il ne l'est pas.

Mathématiquement, comment ça se traduit/ça s'écrit ?

Considérons une entrée $X = x_1 \ x_2 \ x_3 \dots \ x_n$, la régression logistique a pour objectif de trouver une fonction h telle que nous puissions calculer :

$$y = \{1 \text{ si } hX \geq \text{seuil}, \ 0 \text{ si } hX < \text{seuil}\}$$

La fonction qui remplit le mieux ces conditions est la fonction sigmoïde, définie sur R à valeurs dans [0,1].



Tout le problème de classification par régression logistique apparaît alors comme un simple problème d'optimisation où, à partir de données, nous essayons d'obtenir le meilleur jeu de paramètre Θ permettant à notre courbe sigmoïde de coller au mieux aux données. C'est dans cette étape qu'intervient notre apprentissage automatique.
Une fois cette étape effectuée, voici un aperçu du résultat qu'on peut obtenir:

Logistic Regression

```
: from pyspark.ml.classification import LogisticRegression
logistic = LogisticRegression(labelCol = "isFraud", featuresCol = "features")
lrModel = logistic.fit(train)

: prediction_LR = lrModel.transform(test)

: prediction_LR.groupBy("isFraud", "prediction").count().show()
+-----+-----+-----+
|isFraud|prediction| count|
+-----+-----+-----+
|  1.0|      0.0|   430|
|  0.0|      0.0|275753|
|  1.0|      1.0|   380|
|  0.0|      1.0|    40|
+-----+-----+-----+
```

Evaluation

```

tp = prediction_LR[(prediction_LR.isFraud == 1) & (prediction_LR.prediction == 1)].count()
tn = prediction_LR[(prediction_LR.isFraud == 0) & (prediction_LR.prediction == 0)].count()
fp = prediction_LR[(prediction_LR.isFraud == 0) & (prediction_LR.prediction == 1)].count()
fn = prediction_LR[(prediction_LR.isFraud == 1) & (prediction_LR.prediction == 0)].count()
recall_LR = tp/(tp+fn)
precision_LR = tp/(tp+fp)
f1_score_LR = 2*(recall_LR*precision_LR)/(recall_LR+precision_LR)
print("Recall : ", recall_LR)
print("Precision : ", precision_LR)
print("F1 Score : ", f1_score_LR)

Recall : 0.4691358024691358
Precision : 0.9047619047619048
F1 Score : 0.6178861788617886

```

```

from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="isFraud")
areaUnderROC_LR = evaluator.evaluate(prediction_LR, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_LR)

Area under ROC = 0.9831340807452497

areaUnderPR_LR = evaluator.evaluate(prediction_LR, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_LR)

Area under PR = 0.6665939046903049

```

• Naive Bayes

Naive Bayes Classifier est un algorithme populaire en Machine Learning. C'est un algorithme du Supervised Learning utilisé pour la classification. Il est particulièrement utile pour les problématiques de classification de texte. Le naive Bayes classifier se base sur le théorème de Bayes. Ce dernier est un classique de la théorie des probabilités. Ce théorème est fondé sur les probabilités conditionnelles.

Naive bayes

```

from pyspark.ml.classification import NaiveBayes
nb = NaiveBayes(labelCol="isFraud", featuresCol="features", smoothing=1.0, modelType="multinomial")
nb_model = nb.fit(train)

prediction_NB = nb_model.transform(test)

prediction_NB.groupBy("isFraud", "prediction").count().show()

+-----+-----+-----+
|isFraud|prediction| count|
+-----+-----+-----+
| 1.0 | 0.0 | 110 |
| 0.0 | 0.0 | 208676 |
| 1.0 | 1.0 | 700 |
| 0.0 | 1.0 | 67117 |
+-----+-----+-----+

```

Evaluation

```

tp = prediction_NB[(prediction_NB.isFraud == 1) & (prediction_NB.prediction == 1)].count()
tn = prediction_NB[(prediction_NB.isFraud == 0) & (prediction_NB.prediction == 0)].count()
fp = prediction_NB[(prediction_NB.isFraud == 0) & (prediction_NB.prediction == 1)].count()
fn = prediction_NB[(prediction_NB.isFraud == 1) & (prediction_NB.prediction == 0)].count()
recall_NB = tp/(tp+fn)
precision_NB = tp/(tp+fp)
f1_score_NB = 2*(recall_NB*precision_NB)/(recall_NB+precision_NB)
print("Recall : ",recall_NB)
print("Precision : ", precision_NB)
print("F1 Score : ", f1_score_NB)

Recall : 0.8641975308641975
Precision : 0.01032189568987127
F1 Score : 0.02040013405802381

```

```

areaUnderROC_NB = evaluator.evaluate(prediction_NB, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_NB)

```

```
Area under ROC = 0.5626975644150937
```

```

areaUnderPR_NB = evaluator.evaluate(prediction_NB, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_NB)

```

```
Area under PR = 0.008746362879704742
```

• Random forest

Un random forest est constitué d'un ensemble d'arbres de décision indépendants.

Chaque arbre dispose d'une vision parcellaire du problème du fait d'un double tirage aléatoire :

- un tirage aléatoire avec remplacement sur les observations (les lignes de votre base de données). Ce processus s'appelle le tree bagging,
- un tirage aléatoire sur les variables (les colonnes de votre base de données). Ce processus s'appelle le feature sampling.

A la fin, tous ces arbres de décisions indépendants sont assemblés. La prédiction faite par le random forest pour des données inconnues est alors la moyenne (ou le vote, dans le cas d'un problème de classification) de tous les arbres.

L'idée de base de cet algorithme est assez intuitive. A titre d'exemple, si votre banque vous refuse votre demande de crédit, il y a fort à parier que vous irez consulter une ou plusieurs autres banques. Effectivement, un seul avis ne suffit pas en général pour prendre la meilleure décision.

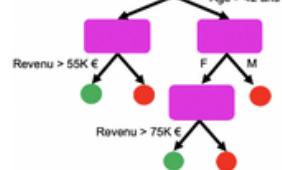
Le random forest fonctionne sur ce même principe : plutôt que d'avoir un estimateur complexe capable de tout faire, le random forest utilise plusieurs estimateurs simples (de moins bonne qualité individuelle). Chaque estimateur a une vision parcellaire du problème. Ensuite, l'ensemble de ces estimateurs est réuni pour obtenir la vision globale du problème. C'est l'assemblage de tous ces estimateurs qui rend extrêmement performante la prédiction.

① Construction de l'arbre N°1 du random forest

Feature sampling

Id client	Age	Revenu	Sexe	Marie	Propriétaire	CSP	Apport personnel
id0001	31	32K	M	1	0	Ouvriers qualifiés	0
id0002	71	42K	M	1	1	Non déterminé	45 000 €
id0003	53	72K	M	0	1	Employés qualifiés	0 €
id0004	45	105K	F	0	1	Cadres	115 000 €
id0005	32	55K	M	1	0	Employés qualifiés	45 000 €
id0006	35	58K	F	0	1	Cadres	25 000 €
id0007	66	35K	M	0	1	Non déterminé	15 000 €
id0008	47	112K	F	1	0	Cadres	245 000 €
id0009	51	85K	F	1	0	Cadres	65 000 €
id0010	52	80K	M	1	1	Cadres	85 000 €

Id Client	Age	Revenue	Sexe
id0002	71	42K	M
id0003	53	72K	M
id0004	45	105K	F
id0005	32	55K	M
id0006	35	58K	F
id0007	66	35K	M

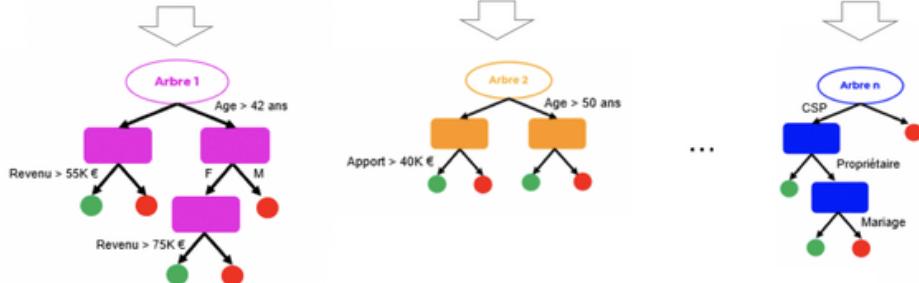


② Mise en œuvre d'une forêt d'arbres

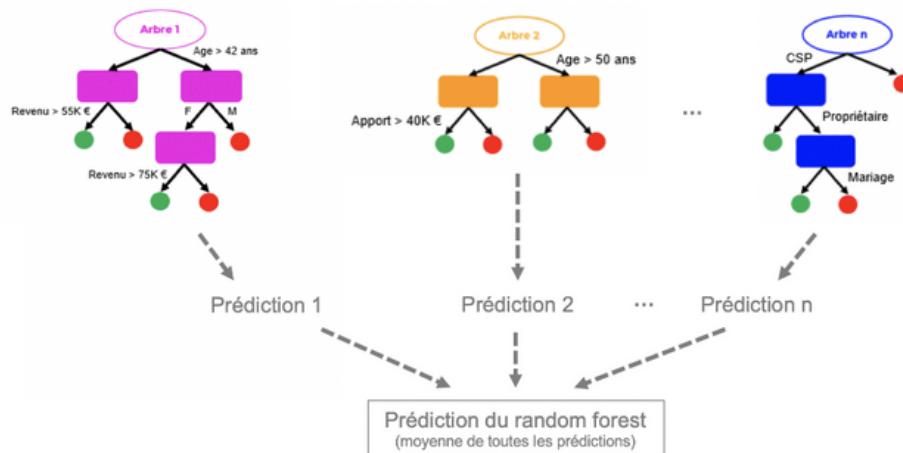
Id client	Age	Revenue	Sexe
id0002	71	42K	M
id0003	53	72K	M
id0004	45	105K	F
id0005	32	55K	M
id0006	35	58K	F
id0007	66	35K	M

Id client	Age	CSP	Apport personnel
id0001	31	Ouvriers qualifiés	0
id0007	66	Non déterminé	15 000 €
id0008	47	Cadres	245 000 €
id0009	51	Cadres	65 000 €
id0010	52	Cadres	85 000 €

Id client	Marie	Propriétaire	CSP
id0001	1	0	Ouvriers qualifiés
id0003	0	1	Employés qualifiés
id0005	1	0	Employés qualifiés
id0007	0	1	Non déterminé
id0009	1	0	Cadres



3 Prédiction du random forest



Random forest

```
: from pyspark.ml.classification import RandomForestClassifier  
  
# Train a RandomForest model.  
rf = RandomForestClassifier(labelCol="isFraud", featuresCol="features", numTrees=10)  
  
rf_model = rf.fit(train)  
  
: # Make predictions.  
prediction_RF = rf_model.transform(test)
```

Evaluation

```
tp = prediction_RF[(prediction_RF.isFraud == 1) & (prediction_RF.prediction == 1)].count()  
tn = prediction_RF[(prediction_RF.isFraud == 0) & (prediction_RF.prediction == 0)].count()  
fp = prediction_RF[(prediction_RF.isFraud == 0) & (prediction_RF.prediction == 1)].count()  
fn = prediction_RF[(prediction_RF.isFraud == 1) & (prediction_RF.prediction == 0)].count()  
recall_RF = tp/(tp+fn)  
precision_RF = tp/(tp+fp)  
f1_score_RF = 2*(recall_RF*precision_RF)/(recall_RF+precision_RF)  
print("Recall : ", recall_RF)  
print("Precision : ", precision_RF)  
print("F1 Score : ", f1_score_RF)  
  
Recall : 0.5679012345679012  
Precision : 0.9956709956709957  
F1 Score : 0.7232704402515723
```

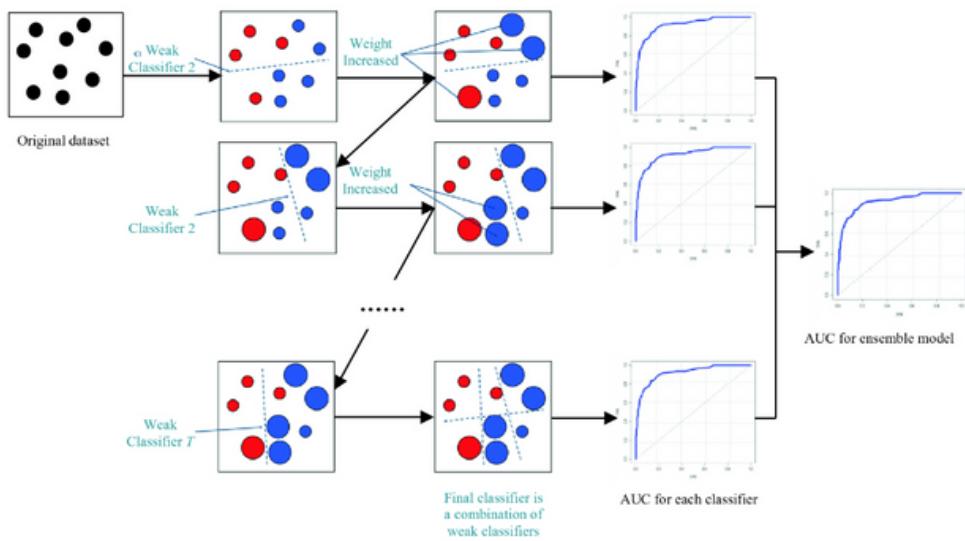
```
areaUnderROC_LR = evaluator.evaluate(prediction_RF, {evaluator.metricName: "areaUnderROC"})  
print("Area under ROC = %s" % areaUnderROC_LR)  
  
Area under ROC = 0.921505064654637
```

```
# Area under precision-recall curve  
areaUnderPR_RF = evaluator.evaluate(prediction_RF, {evaluator.metricName: "areaUnderPR"})  
print("Area under PR = %s" % areaUnderPR_RF)  
  
Area under PR = 0.7501002010064396
```

• Gradient boosted tree

C'est une méthode qui permet de transformer les apprenants faibles en apprenants forts. Dans le paysage du boosting, chaque arbre s'adapte à la version modifiée du premier ensemble de données. Vous pouvez expliquer l'algorithme du boosting de gradient en établissant l'algorithme Ada Boost. Il commence par former des arbres de décision. Chaque observation de cette procédure se voit attribuer un poids égal.

Après avoir analysé le premier arbre, les spécialistes des données augmentent le poids de chaque observation qu'ils trouvent compliquée à classer. En revanche, ils diminuent le poids de celles dont la classification n'est pas un problème. Par conséquent, vous remarquerez que le deuxième arbre pousse sur les données pondérées. L'idée originale est d'apporter des améliorations aux prévisions du premier arbre.



Gradient-Boosted Tree Classifier

```
from pyspark.ml.classification import GBTClassifier

# Train a GBT model.
gbt = GBTClassifier(labelCol="isFraud", featuresCol="features", maxIter=10)

# Train model.
gbt_model = gbt.fit(train)

prediction_GBT = gbt_model.transform(test)
```

Evaluation

```
tp = prediction_GBT[(prediction_GBT.isFraud == 1) & (prediction_GBT.prediction == 1)].count()
tn = prediction_GBT[(prediction_GBT.isFraud == 0) & (prediction_GBT.prediction == 0)].count()
fp = prediction_GBT[(prediction_GBT.isFraud == 0) & (prediction_GBT.prediction == 1)].count()
fn = prediction_GBT[(prediction_GBT.isFraud == 1) & (prediction_GBT.prediction == 0)].count()
recall_GBT = tp/(tp+fn)
precision_GBT = tp/(tp+fp)
f1_score_GBT = 2*(recall_GBT*precision_GBT)/(recall_GBT+precision_GBT)
print("Recall : ",recall_GBT)
print("Precision : ", precision_GBT)
print("F1 Score : ", f1_score_GBT)

Recall :  0.6691358024691358
Precision :  0.9713261648745519
F1 Score :  0.7923976608187135
```

```
#Area under ROC curve
# Area under ROC curve
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="isFraud")
areaUnderROC_GBT = evaluator.evaluate(prediction_GBT, {evaluator.metricName: "areaUnderROC"})
print("Area under ROC = %s" % areaUnderROC_GBT)
```

Area under ROC = 0.9895590708060567

```
# Area under precision-recall curve
areaUnderPR_GBT = evaluator.evaluate(prediction_GBT, {evaluator.metricName: "areaUnderPR"})
print("Area under PR = %s" % areaUnderPR_GBT)
```

Area under PR = 0.8230876904767537

4.Comparaison et choix du modèle

Dans la table ci dessous ; on représente les différentes métriques d'évaluation de chacun des algorithmes qu'on a déjà implémenter : Recall, Precision, F1 Score, Area Under ROC, Area Under PR

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100

on peut choisir le meilleur d'entre eux, selon les critères dont on a besoin.

- Selon Recall

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594

- Selon Precision

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746

- Selon F1 Score

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746

- Selon Area Under ROC

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746

- Selon Area Under PR

	Recall	Precision	F1 Score	Area Under ROC	Area Under PR
Gradient-Boosted Tree	0.669136	0.971326	0.792398	0.989559	0.823088
Random forest	0.567901	0.995671	0.723270	0.921505	0.750100
Logistic Regression	0.469136	0.904762	0.617886	0.921505	0.666594
Naive Bayes	0.864198	0.010322	0.020400	0.562698	0.008746

Choix de l'algorithme pour notre problème :

Random Forest : Puisqu'on s'est basé sur la métrique "Precision" aka combien de transactions frauduleuses on a pu détecter grâce à notre modèle et il s'avère que l'algorithme Random forest est l'algorithme le plus satisfaisant de ce critère avec 99% de précision .

Deploiment par Streamlit

on passe maintenant à la partie déploiement. Pour ce faire, on a décidé d'utiliser Streamlit qui permet de déployer les modèles de machine Learning et deep learning déjà implémentés .

```

1 import streamlit as st
2 import ktrain
3 import pickle
4 import joblib
5 st.title('Fraud detection')
6 st.subheader('Predict if it is fraud or not')
7 step=st.text_input('Enter step')
8 Type=st.text_input('Enter Type')
9 amount=st.text_input('Enter amount')
10 nameOrig=st.text_input('Enter nameOrig')
11 oldbalanceOrig=st.text_input('Enter oldbalanceOrig')
12 newbalanceOrig=st.text_input('Enter newbalanceOrig')
13 nameDest=st.text_input('Enter nameDest')
14 oldbalanceDest=st.text_input('Enter oldbalanceDest')
15 newbalanceDest=st.text_input('Enter newbalanceDest')
16 isFlaggedFraud=st.text_input('Enter isFlaggedFraud')
17 if st.button("Predict"):
18     model=joblib.load('modelgbt1.pkl')
19     data = list(map([step,Type,amount, nameOrig,oldbalanceOrig,newbalanceOrig], [nameDest,oldbalanceDest,newbalanceDest], [isFlaggedFraud]))
20     result, probs = model.predict(data)
21     if result==0:
22         st.success('notfraud')
23     elif result==1:
24         st.error('fraud')
25

```

on arrive donc à mettre en place l'interface graphique suivante:

Fraud detection

Predict if it is fraud or not

Enter step

Enter Type

Enter amount

Enter nameOrig

Enter oldbalanceOrig

Enter newbalanceOrig

Enter nameDest

Enter oldbalanceDest

Enter newbalanceDest

Enter isFlaggedFraud



test

Predict if it is fraud or not

Enter step

Enter Type

Enter amount

Enter nameOrig

Enter oldbalanceOrig

Enter newbalanceOrig

Enter nameDest

Enter oldbalanceDest

Enter newbalanceDest

Enter isFlaggedFraud

notfraud

Conclusion et Perspectives

Finalement , La structure de base pour un système de détection de fraude en apprentissage automatique est comme suit :

- **Chargement des données** : Les données sont d'abord introduites dans le modèle. La précision du modèle est déterminée par la quantité de données utilisées pour l'entraîner ; plus on utilise de données, plus le modèle est performant. Vous devrez introduire des volumes de données de plus en plus importants dans votre modèle pour détecter les escroqueries liées à une seule entreprise. Cela permettra d'entraîner votre modèle à détecter précisément les comportements frauduleux propres à votre entreprise.
- **Extraction de caractéristiques** : Il s'agit essentiellement d'extraire des informations de chaque fil impliqué dans un processus de transaction. Par exemple, le lieu d'où la transaction est effectuée, l'identité du client, le mode de paiement et le réseau utilisé pour la transaction.
- **Formation de l'algorithme** : Une fois que vous avez construit un algorithme d'apprentissage automatique pour la détection de la fraude, vous devez l'entraîner en lui fournissant des données de clients afin qu'il puisse apprendre à distinguer les transactions "frauduleuses" des transactions "réelles".
- **Création du modèle** : Vous disposerez d'un modèle capable de détecter les transactions "frauduleuses" et "non frauduleuses" , une fois que vous aurez entraîné votre algorithme de détection des fraudes sur un ensemble de données spécifique. L'avantage de la détection des fraudes à l'aide d'algorithmes d'apprentissage automatique est qu'elle s'améliore avec le temps lorsqu'elle est exposée à davantage de données.

La détection des fraudes à l'aide de services d'apprentissage automatique permet de réduire la charge manuelle globale des analystes et d'améliorer le rendement global du travail. Elle permet aux analystes de travailler plus rapidement et plus précisément en leur donnant accès à des données et à des informations, ce qui réduit le temps consacré à l'analyse manuelle.

La création d'un modèle entraîné avec une quantité suffisante de données peut aider l'algorithme d'apprentissage automatique de détection des fraudes à différencier les clients réels des clients frauduleux.