Analisi esplorativa e classificazione di transazioni fraudolente

Questo Notebook contiene del codice per scaricare un dataset di transazioni finanziarie, alcune delle quali sono fraudolente.

Esegui il codice presente, completa il codice mancante e rispondi alle domande.

Puoi importare questo Notebook nel tuo ambiente di sviluppo preferito: Databricks come svolto in classe (ricorda: prima di eseguire il codice dovrai creare un cluster) o IDE locale (in questo caso, assicurati che le librerie che verranno importate siano installate).

```
!pip install pyspark
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: pyspark in /usr/local/lib/python3.10/dist-packages (3.4.0)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
```

```
#Importazione delle librerie
import requests
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import pyspark
from pyspark.sql import *
from pyspark.sql.types import *
from pyspark.sql.functions import *
from pyspark import SparkContext, SparkConf

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import BinaryClassificationEvaluator
```

```
#Download del dataset
def get_data(dataset_url, dest, chunk_size=1024):
  response = requests.get(dataset_url,  verify= False, stream=True)
  if response.status_code == 200:
    with open(dest, "wb") as file:
      for block in response.iter_content(chunk_size=chunk_size):
        if block:
          file.write(block)

DATASET_URL = "https://learning.prorob.it/ITS_2023/LF/datasets/transactions_10000.csv"
DATASET_FOLDER = "/tmp"
DATASET_FILE = DATASET_FOLDER + "/" + DATASET_URL.split("/")[-1]
```

```
print("Retrieving dataset from URL: {} ...".format(DATASET_URL))
get_data(DATASET_URL, DATASET_FILE)
print("Dataset successfully retrieved and stored at: {}".format(DATASET_FILE))
```

```
Retrieving dataset from URL: https://learning.prorob.it/ITS_2023/LF/datasets/transactions_10000.csv ...
/usr/local/lib/python3.10/dist-packages/urllib3/connectionpool.py:1045: InsecureRequestWarning: Unverified HTTPS request is being m
  warnings.warn(
Dataset successfully retrieved and stored at: /tmp/transactions_10000.csv
```

1. Carica il dataset come Spark DataFrame

```
#Creo una sessione Spark
spark = SparkSession.builder.getOrCreate()

#Percorso del file del dataset
DATASET_FILE = "/tmp/transactions_10000.csv"

#Leggo il file CSV e crea un DataFrame
df = spark.read.csv(DATASET_FILE, header=True, inferSchema=True)

#Visualizzo lo schema del DataFrame
df.printSchema()

#DataFrame creato
```

```
root
 |-- type: string (nullable = true)
 |-- amount: double (nullable = true)
```

```
 |-- nameOrig: string (nullable = true)
 |-- oldbalanceOrig: double (nullable = true)
 |-- newbalanceOrig: double (nullable = true)
 |-- nameDest: string (nullable = true)
 |-- oldbalanceDest: double (nullable = true)
 |-- newbalanceDest: double (nullable = true)
 |-- isFraud: integer (nullable = true)
```

2. Analisi esplorativa

2.1 Mostra le prime 5 righe del dataset

```
#Le prime 5 righe del DataSet
df.show(5)
```

```
+--------+--------+-----------+-------------+-------------+-----------+-------------+-------------+-------+
|    type|  amount|   nameOrig|oldbalanceOrig|newbalanceOrig|   nameDest|oldbalanceDest|newbalanceDest|isFraud|
+--------+--------+-----------+-------------+-------------+-----------+-------------+-------------+-------+
| PAYMENT| 9839.64|C1231006815|     170136.0|     160296.36|M1979787155|          0.0|          0.0|      0|
| PAYMENT| 1864.28|C1666544295|      21249.0|      19384.72|M2044282225|          0.0|          0.0|      0|
|TRANSFER|   181.0|C1305486145|        181.0|          0.0| C553264065|          0.0|          0.0|      1|
|CASH_OUT|   181.0| C840083671|        181.0|          0.0|  C38997010|      21182.0|          0.0|      1|
| PAYMENT|11668.14|C2048537720|      41554.0|      29885.86|M1230701703|          0.0|          0.0|      0|
+--------+--------+-----------+-------------+-------------+-----------+-------------+-------------+-------+
only showing top 5 rows
```

2.2 Quante righe e colonne ha il dataset in totale?

```
#Numero totale delle righe
row_count = df.count()
print("Numero totale delle righe: {}".format(row_count))

#Numero totale delle colonne
col_count = len(df.columns)
print("Numero totale delle colonne: {}".format(col_count))

#Num. totale righe -- > 10000
#Num. totale colonne -- > 9
```

```
Numero totale delle righe: 10000
Numero totale delle colonne: 9
```

2.3 Quante transazioni fraudolente ci sono?

```
#Operazione per trovare il totale delle transazioni fraudolente

#Filtro le righe con isFraud = 1 (transazioni fraudolente)
fraud_transactions_count = df.filter(col("isFraud") == 1).count()

#Stampo il numero totale di transazioni fraudolente
print("Numero totale di transazioni fraudolente: {}".format(fraud_transactions_count))

#Num. totale di transazioni fraudolente -- > 68
```

```
Numero totale di transazioni fraudolente: 68
```

2.4 Quante transazioni non fraudolente ci sono?

```
#Operazione per trovare il totale delle transazioni non fraudolente

#Filtro le righe con isFraud = 0 (transazioni non fraudolente)
non_fraud_transactions_count = df.filter(col("isFraud") == 0).count()

#Stampo il numero totale di transazioni non fraudolente
print("Numero totale di transazioni non fraudolente: {}".format(non_fraud_transactions_count))

#Num. totale di transazioni non fraudolente -- > 9932
```

```
Numero totale di transazioni non fraudolente: 9932
```

2.5 Qual è la percentuale di transazioni fraudolente sul totale?

```
#Numero totale di transazioni
total_transactions_count = df.count()

#Il numero totale delle transazioni fraudolente lo abbiamo già (fraud_transactions_count)

#Calcolo della percentuale di transazioni fraudolente
fraud_percentage = (fraud_transactions_count / total_transactions_count) * 100

#Stampo della percentuale di transazioni fraudolente
print("Percentuale di transazioni fraudolente: {:.2f}%".format(fraud_percentage))

#Perc. delle transazioni fraudolente sul tot. delle transazioni -- > 0.68%
```

```
Percentuale di transazioni fraudolente: 0.68%
```

2.6 Il dataset è bilanciato?

```
#Per verificare se il DataSet è bilanciato, si può confrontare il numero di transazioni fraudolente con il numero di transazioni non
#fraudolente nel DataFrame. Se i due numeri sono simili, il DataSet può essere considerato bilanciato.

#Si può farlo attraverso questa operazione di confronto:
if fraud_transactions_count == non_fraud_transactions_count:
    print("Il dataset è bilanciato.")
else:
    print("Il dataset non è bilanciato.")

#Il DataSet NON è bilanciato in quanto i due valori delle transazioni (68 per le fraudolenti e 9932 per quelle non fraudolenti non
#sono, per niente, simili ma molto distanti tra di loro)
```

```
Il dataset non è bilanciato.
```

2.7 Ci sono valori nulli? Se sì, rimuovili

```
#Verifico se ci sono valori nulli
null_counts = df.select([count(when(col(c).isNull(), c)).alias(c) for c in df.columns]).toPandas().transpose()

#Stampo dei valori nulli per ogni colonna
print("Valori nulli nel DataFrame:")
print(null_counts)

#No, nel DataFrame non risultano valori nulli quindi non è necessaria l'oprazione della loro rimozione
```

```
Valori nulli nel DataFrame:
                0
type            0
amount          0
nameOrig        0
oldbalanceOrig  0
newbalanceOrig  0
nameDest        0
oldbalanceDest  0
newbalanceDest  0
isFraud         0
```

2.8 Qual è l'importo medio di ogni transazione?

```
#Calcolo dell'importo medio di ogni transazione
average_amount_per_transaction = df.agg(avg("amount").alias("average_amount"))

#Stampo dell'importo medio di ogni transazione
print("Importo medio di ogni transazione:")
average_amount_per_transaction.show()

#L'importo medio di ogni transazione è  -- > 103546.68994900018
```

```
Importo medio di ogni transazione:
+------------------+
|    average_amount|
+------------------+
|103546.68994900018|
+------------------+
```

2.9 Qual è la deviazione standard rispetto alla media di ogni transazione?

```
#Calcolo della deviazione standard rispetto alla media di ogni transazione
stddev_amount_per_transaction = df.select(stddev("amount").alias("stddev_amount"))

#Stampo della deviazione standard rispetto alla media di ogni transazione
stddev_amount_per_transaction.show()
```

```
+-----------------+
|    stddev_amount|
+-----------------+
|266307.1952950504|
+-----------------+
```

2.10 Calcola l'importo medio di ogni transazione

```
#Calcolo dell'importo medio per ogni tipo di transazione
average_amount_per_transaction = df.groupBy("type").agg(avg("amount").alias("average_amount"))

#Stampo dell'importo medio di ogni transazione
print("Importo medio di ogni transazione:")
average_amount_per_transaction.show()


#L'importo medio per ogni tipo di transazione:
    #TRANSFER -- > 471654.5189142232
    #CASH_IN -- > 168041.4924679321
    #CASH_OUT -- >173507.34354277063
    #PAYMENT -- > 7891.430867337604
    #DEBIT -- > 3576.593720930232

    Importo medio di ogni transazione:
    +--------+------------------+
    |    type|    average_amount|
    +--------+------------------+
    |TRANSFER| 471654.5189142232|
    | CASH_IN| 168041.4924679321|
    |CASH_OUT|173507.34354277063|
    | PAYMENT| 7891.430867337604|
    |   DEBIT| 3576.593720930232|
    +--------+------------------+
```

2.11 Calcola il numero medio di transazioni fatte per ogni utente

```
#Calcolo del numero medio di transazioni per ogni utente
average_transactions_per_user = df.groupBy("nameOrig").agg(count("*").alias("transaction_count")).groupBy().avg("transaction_count").alia

#Stampo del numero medio di transazioni per ogni utente
average_transactions_per_user.show()

#Ogni utente ha fatto, in media, una sola transazione
```

```
+----------------------+
|avg(transaction_count)|
+----------------------+
|                   1.0|
+----------------------+
```

3: Classificazione di transazioni fraudolente

```
####Trasformazione delle variabili categoriche in one-hot encoding e assemblaggio delle features in un unico vettore
def to_numerical(df, numerical_features, categorical_features, target_variable):

    """
    Args:
        - df: the input dataframe
        - numerical_features: the list of column names in `df` corresponding to numerical features
        - categorical_features: the list of column names in `df` corresponding to categorical features
        - target_variable: the column name in `df` corresponding to the target variable

    Return:
```

```
        - transformer: the pipeline of transformation fit to `df` (for future usage)
        - df_transformed: the dataframe transformed according to the pipeline
    """

    from pyspark.ml import Pipeline
    from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler


    #1. Create a list of indexers, i.e., one for each categorical feature
    indexers = [StringIndexer(inputCol=c, outputCol="{0}_indexed".format(c), handleInvalid="keep") for c in categorical_features]

    #2. Create the one-hot encoder for the list of features just indexed (this encoder will keep any unseen label in the future)
    encoder = OneHotEncoder(inputCols=[indexer.getOutputCol() for indexer in indexers],
                                    outputCols=["{0}_encoded".format(indexer.getOutputCol()) for indexer in indexers],
                                    handleInvalid="keep")

    #3. Indexing the target column (i.e., transform it into 0/1) and rename it as "label"
    #Note that by default StringIndexer will assign the value `0` to the most frequent label, which in the case of `deposit` is `no`
    #As such, this nicely resembles the idea of having `deposit = 0` if no deposit is subscribed, or `deposit = 1` otherwise.
    label_indexer = StringIndexer(inputCol = target_variable, outputCol = "label")

    #4. Assemble all the features (both one-hot-encoded categorical and numerical) into a single vector
    assembler = VectorAssembler(inputCols=encoder.getOutputCols() + numerical_features, outputCol="features")

    #5. Populate the stages of the pipeline
    stages = indexers + [encoder] + [label_indexer] + [assembler]

    #6. Setup the pipeline with the stages above
    pipeline = Pipeline(stages=stages)

    #7. Transform the input dataframe accordingly
    transformer = pipeline.fit(df)
    df_transformed = transformer.transform(df)

    #8. Eventually, return both the transformed dataframe and the transformer object for future transformations
    return transformer, df_transformed

NUMERICAL_FEATURES = ["amount",
                      "oldbalanceOrig",
                      "newbalanceOrig",
                      "oldbalanceDest",
                      "newbalanceDest"
                      ]
CATEGORICAL_FEATURES = ["type",
                        "nameOrig"
                        ]
TARGET_VARIABLE = "isFraud"

#Transform the training set and get back both the transformer and the new dataset
transformer, df_transformed = to_numerical(df, NUMERICAL_FEATURES, CATEGORICAL_FEATURES, TARGET_VARIABLE)
```

3.1 Dividi il dataset appena trasformato in train e test, in modo random e con un rapporto di 70-30

```
#Definisco il rapporto di divisione (70% per il train, 30% per il test)
train_ratio = 0.7
test_ratio = 0.3

#Divido il DataSet in train e test in modo randomico
train_data, test_data = df_transformed.randomSplit([train_ratio, test_ratio], seed=42)

#Stampo il numero di righe nel train set e nel test set
print("Numero di righe nel train set:", train_data.count())
print("Numero di righe nel test set:", test_data.count())

#Numero di righe nel train set -- > 7104
#Numero di righe nel test set -- > 2896
```

```
Numero di righe nel train set: 7104
Numero di righe nel test set: 2896
```

3.2 Definisci e addestra un modello LogisticRegression sul train set utilizzando la colonna "features" come X e la colonna "isFraud" come y

```
from pyspark.ml.feature import VectorAssembler
#Definisco le due colonne "features" come X e "isFraud" come y

#Rinomino la colonna "features" in "X"
train_data = train_data.withColumnRenamed("features", "X")
```

```
#Creo la colonna "y" dal valore della colonna "isFraud"
train_data = train_data.withColumn("y", train_data["isFraud"])

#Creo un VectorAssembler per raccogliere le feature in una colonna "features"
assembler = VectorAssembler(inputCols=["X"], outputCol="features")

#Trasformo il DataFrame di addestramento utilizzando il VectorAssembler
train_data = assembler.transform(train_data)



#Ora definisco ed addestro il modello LogisticRegression

#Definizione
lr = LogisticRegression(featuresCol="features", labelCol="y")

#Addestramento sul train set
lr_model = lr.fit(train_data)

#Stampo i coefficienti del modello
print("Coefficients:", lr_model.coefficients)

#Stampo l'intercetta del modello
print("Intercept:", lr_model.intercept)
```

```
Coefficients: [-0.1571475377188016,-0.36942462019306577,0.8112727578382884,0.09293589392473871,-0.09448467517240523,0.0,0.0,0.0,-0.
Intercept: -19.64789180514869
```

3.3 Fai predizioni sul test set con il modello appena addestrato

```
#Effettuo le predizioni sul test set utilizzando il modello appena addestrato
predictions = lr_model.transform(test_data)

#Visualizzazione delle predizioni
predictions.select("label", "prediction", "probability").show()
```

```
+-----+----------+--------------------+
|label|prediction|         probability|
+-----+----------+--------------------+
|  0.0|       0.0|[0.99999999903198...|
|  0.0|       0.0|[0.99999999853376...|
|  0.0|       0.0|[0.99999999919156...|
|  0.0|       0.0|[0.99999999838855...|
|  0.0|       0.0|[0.99999999903538...|
|  0.0|       0.0|[0.99999999812694...|
|  0.0|       0.0|[0.99999999845224...|
|  0.0|       0.0|[0.99999999906338...|
|  0.0|       0.0|[0.99999999860789...|
|  0.0|       0.0|[0.99999999900097...|
|  0.0|       0.0|[0.99999999910541...|
|  0.0|       0.0|[0.99999999818659...|
|  0.0|       0.0|[0.99999999824777...|
|  0.0|       0.0|[0.99999999892330...|
|  0.0|       0.0|[0.99999999867704...|
|  0.0|       0.0|[0.99999999799197...|
|  0.0|       0.0|[0.99999999862123...|
|  0.0|       0.0|[0.99999999853002...|
|  0.0|       0.0|[0.99999999834969...|
|  0.0|       0.0|[0.99999999948118...|
+-----+----------+--------------------+
only showing top 20 rows
```

3.4 Bonus: definisci un BinaryClassificationEvaluator e mostra la metrica areaUnderROC sulle predizioni appena fatte

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

#Creo un oggetto BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(labelCol="isFraud", rawPredictionCol="probability", metricName="areaUnderROC")

#Calcolo l'area under ROC curve sul DataFrame con le predizioni
auc = evaluator.evaluate(predictions)

#Stampo l'area under ROC curve
print("Area under ROC curve:", auc)
```

```
Area under ROC curve: 0.8740753582986592
```

```
#Conversione del file notebook in pdf
#!apt-get install texlive-xetex -y
#!pip install nbconvert


#!jupyter nbconvert --to pdf ERavagnan_Esame_notebook.ipynb
```

✓ 0 s data/ora di completamento: 11:09