Complément au TP2 : calcul de la régression logisitique

Propos introductif :
Dans la mesure où le script R que nous avions lancé pour le calcul de la régression logistique n'a pas pu se terminer dans un temps raisonnable, nous avons décidé de l'exécuter en python sur Spark (à travers la stack Big Data de l'entreprise où l'un de nous est en stage, en l'occurrence Stéphane URENA au sein de Chronopost (vous pouvez vérifier auprès du service des stages)).

Protocole :
Après récupération des 2 CSV train.csv et test.csv issus de la partition exécutée dans notre script R (de manière à travailler avec les mêmes données), nous avons importé ces données sur une VM incluant Spark.

Exécution du code :

```
[surena@lyn1e338-hdp surena]$ python lr.py
/usr/lib64/python2.7/site-packages/scipy/__init__.py:110: UserWarning: Numpy 1.8.2 or above is recommended for this version of scipy (detected version 1.7.1)
  UserWarning)
RuntimeError: module compiled against API version 9 but this version of numpy is 7
2018-12-03 11:23:53 WARN  NativeCodeLoader:62 - Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
------------------
<class 'pyspark.sql.dataframe.DataFrame'>
------------------
Train set count: 10882
Test set count: 4664
New set count: 15546
root
 |-- ID: integer (nullable = true)
 |-- Prod: integer (nullable = true)
 |-- Uprice: double (nullable = true)
 |-- Insp: integer (nullable = true)

None
root
 |-- ID: integer (nullable = true)
 |-- Prod: integer (nullable = true)
 |-- Uprice: double (nullable = true)
 |-- Insp: integer (nullable = true)

---------------
<class 'pyspark.sql.dataframe.DataFrame'>
---------------
+----+----+----------------+----+
|  ID|Prod|          Uprice|Insp|
+----+----+----------------+----+
| 756|3197|           11.45|   1|
| 259|1060|20.3915119363395|   1|
|1662|3131| 37.734693877551|   1|
|1962|4089|51.3681592039801|   1|
|5089|1088|2.47024021669399|   1|
+----+----+----------------+----+
only showing top 5 rows
```

Illustration 1: Lancement du Script Python

```
-------------Preparing Data for Machine Learning---------------
-----------PIPELINES------------
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- ID: integer (nullable = true)
 |-- Prod: integer (nullable = true)
 |-- Uprice: double (nullable = true)
 |-- Insp: integer (nullable = true)

Train set count: 10927
Test set count: 4619
--------------- fitting model -----------------
2018-12-03 11:24:03 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
2018-12-03 11:24:03 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
+-----+--------------------+----------+--------------------+
|label|       rawPrediction|prediction|         probability|
+-----+--------------------+----------+--------------------+
|  0.0|[2.78296963433772...|       0.0|[0.94174856713185...|
|  0.0|[3.35321149377891...|       0.0|[0.96620984304698...|
|  0.0|[2.79160184914983...|       0.0|[0.94222031310471...|
|  0.0|[3.30550064489822...|       0.0|[0.96461703386953...|
|  0.0|[3.30523763713631...|       0.0|[0.96460805605148...|
|  0.0|[2.68577663388141...|       0.0|[0.93618212033173...|
|  0.0|[3.02849753745320...|       0.0|[0.95384507266018...|
|  0.0|[2.74057170113674...|       0.0|[0.93937866118053...|
|  0.0|[2.75834301561455...|       0.0|[0.94038280629286...|
|  0.0|[2.85959397231016...|       0.0|[0.94581249387054...|
+-----+--------------------+----------+--------------------+
only showing top 10 rows
```

Illustration 2: Exécution de l'algorithme

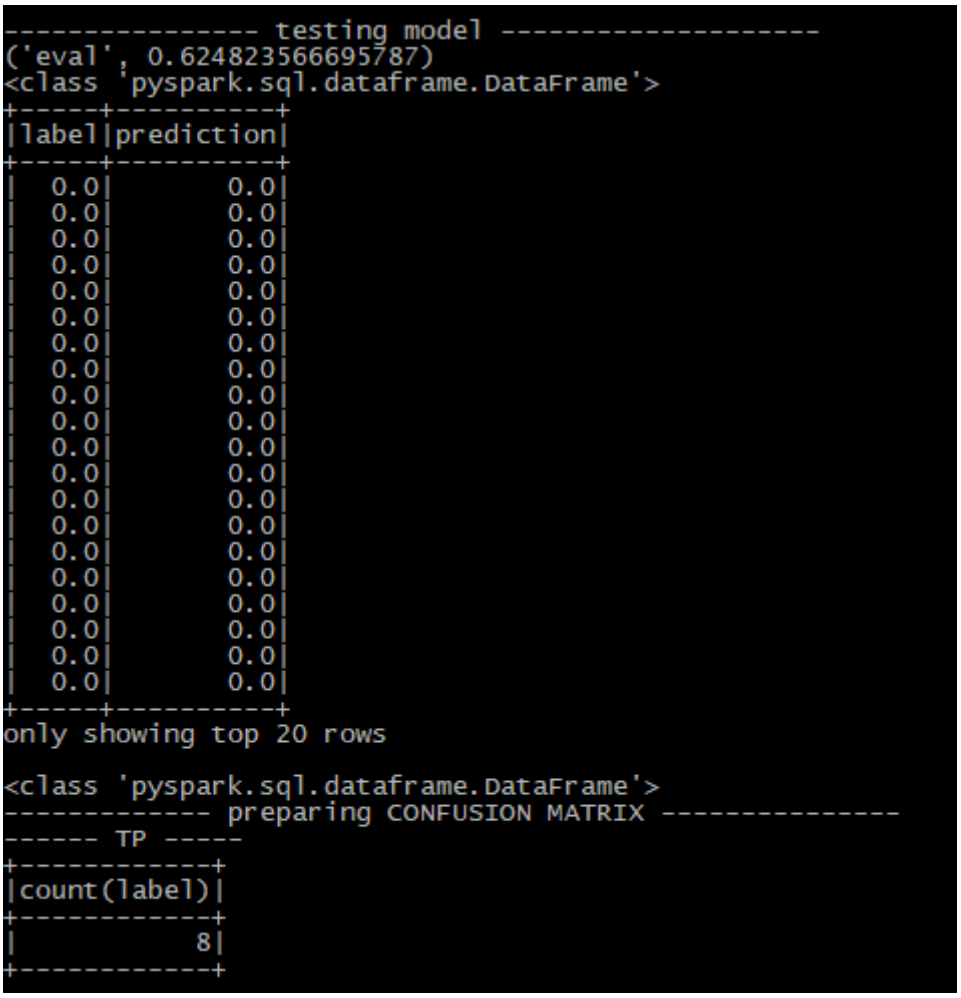

Illustration 3: prédictions

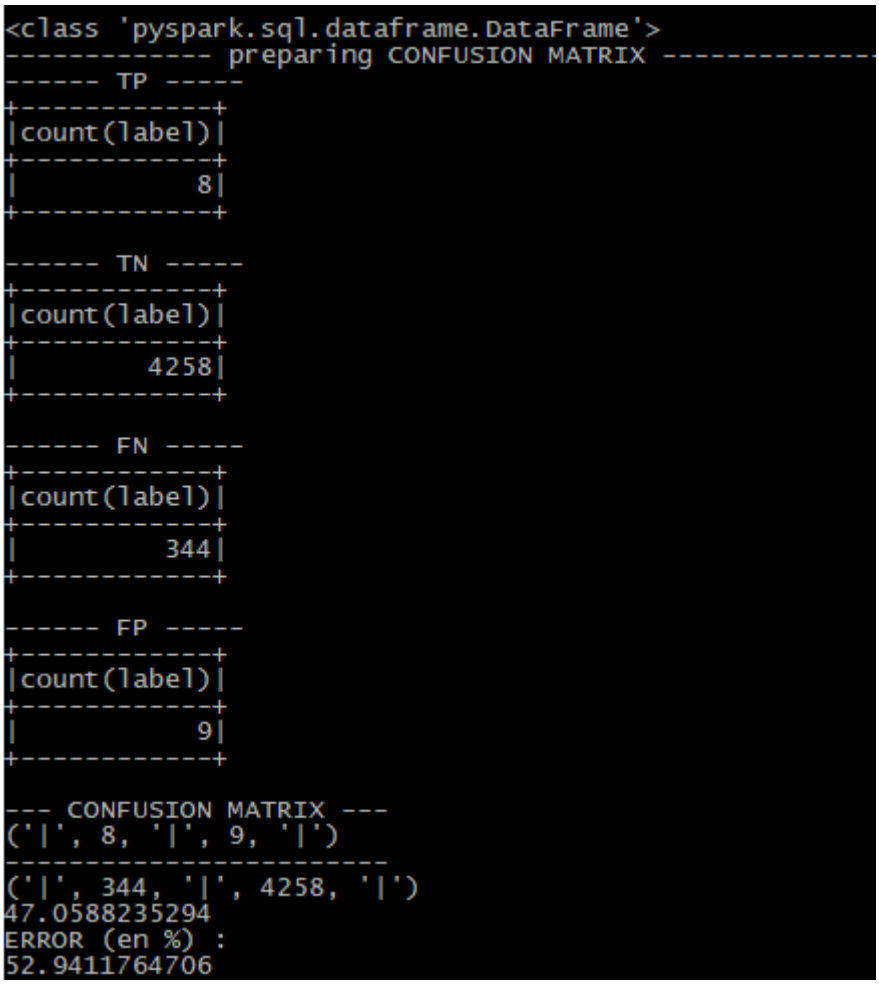

Illustration 4: matrice de confusion et résultats

Résultats :

Le code en annexe a été utilisé pour obtenir nos résultats. Après calcul de la matrice de confusion, nous avons obtenu une précision de 47 %, autrement dit une erreur de 52 % (particulièrement élevée), ce qui nous a amené à conclure que la régression logistique n'était pas une bonne approche pour l'exercice proposé (puisque l'algorithme a une performance proche d'une approche naïve telle que l'affectation aléatoire de labels).

Annexe : script python de LR

```
from pyspark.ml import Pipeline
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler
spark = SparkSession.builder.appName("Logit").getOrCreate()


df_train = spark.read.csv('../../data/surena/train.csv', header=True, inferSchema=True)
df_test = spark.read.csv('../../data/surena/test.csv', header=True, inferSchema=True)
print("------------------")
print(type(df_test))
print("------------------")
print("Train set count: "+str(df_train.count()))
print("Test set count: "+str(df_test.count()))
df=df_train.union(df_test)

print("New set count: "+str(df.count()))

print(df.printSchema())

df = df.select('ID', 'Prod', 'Uprice', 'Insp')
cols = df.columns
df.printSchema()

print("---------------")
print(type(df))
print("---------------")
df.show(5)


# Preparing Data for Machine Learning
print("-------------Preparing Data for Machine Learning----------------")

stages = []
label_stringIdx = StringIndexer(inputCol='Insp', outputCol='label')
stages += [label_stringIdx]

numericCols = ['ID','Prod','Uprice']
assembler = VectorAssembler(inputCols=numericCols, outputCol="features")
stages += [assembler]

#Pipelines
print("-----------PIPELINES------------")

pipeline=Pipeline(stages=stages)
pipelineModel=pipeline.fit(df)
df=pipelineModel.transform(df)
selectedCols = ['label','features'] + cols
df=df.select(selectedCols)
df.printSchema()
```

```
train,test=df.randomSplit([0.7,0.3],15)
print("Train set count: "+str(train.count()))
print("Test set count: "+str(test.count()))


#Machine Learning : Logistic Regression Model
print("---------------- fitting model -----------------")

from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol='features',labelCol='label', maxIter=10)
lrModel=lr.fit(train)

predictions=lrModel.transform(test)
predictions.select('label','rawPrediction','prediction','probability').show(10)

#evaluate LR
print("---------------- testing model --------------------")


from pyspark.ml.evaluation import BinaryClassificationEvaluator
trainingSummary = lrModel.summary
eval=BinaryClassificationEvaluator()
print('eval',eval.evaluate(predictions))
print(type(predictions))

tab=predictions.select('label','prediction')
tab.show()
print(type(tab))

# CONFUSION MATRIX

print("------------- preparing CONFUSION MATRIX ---------------")
tab.createOrReplaceTempView("table")
sql1=spark.sql("SELECT count(label) FROM table WHERE label=1 and prediction=1")
print("------ TP -----")
sql1.show()
sql2=spark.sql("SELECT count(label) FROM table WHERE label=0 and prediction=0")
print("------ TN -----")
sql2.show()
sql3=spark.sql("SELECT count(label) FROM table WHERE label=1 and prediction=0")
print("------ FN -----")
sql3.show()
sql4=spark.sql("SELECT count(label) FROM table WHERE label=0 and prediction=1")
print("------ FP -----")
sql4.show()

tp=sql1.first()['count(label)']
tn=sql2.first()['count(label)']
fn=sql3.first()['count(label)']
fp=sql4.first()['count(label)']
```

```
print("--- CONFUSION MATRIX ---")
print("|",tp,"|",fp,"|")
print("-----------------------")
print("|",fn,"|",tn,"|")


tp=float(tp)
tn=float(tn)
fn=float(fn)
fp=float(fp)

precision= float(tp/(tp+fp)*100)
print(precision)
error = float(100 - precision)
print("ERROR (en %) : ")
print(error)
```