LOAN DEFAULT PREDICTION

Project on Big Data analytics with Spark

Project date: 12.09.2023

Team members:

- Cindurasri. T. L
- Jeyasneha.S
- Senthamarai Kannan
- Nallasamy.K
- Lakshminarayanan.B

PROBLEM STATEMENT:

To Build a logistic regression model in big data environment for a bank to predict whether a client will default on a loan or not.

ABSTRACT:

Banks provide loan to clients in exchange for the promise of repayment. Some might default on the loans; unable to repay them due to some reason. The bank maintains insurance to reduce their risk of loss in the event of default. The insured amount may cover all or just some part of the loan amount.

For this assignment, the bank wants to predict which client will default on their loans based on their financial information.

PROCESSING:

This process involves the following steps to build a logistic regression model:

Step 1:

Removing all unwanted columns to enhance the quality, efficiency, and effectiveness of machine learning tasks. It helps you focus on the most relevant information and can lead to better insights and model performance.

The unwanted columns are:

* CODE_GENDER
* NAME_CONTRACT_TYPE
* CNT_CHILDREN
* DAYS_BIRTH
* OWN_CAR_AGE
* FLAG_MOBIL
* FLAG_EMP_PHONE
* FLAG_WORK_PHONE
* FLAG_CONT_MOBILE
* FLAG_PHONE
* CNT_FAM_MEMBERS
* REGION_RATING_CLIENT
* AMT_ANNUITY
* DAYS_EMPLOYED

Total no of columns remaining are 16 columns

Among that 'TARGET' is the target column of this prediction

Step 2:

This step involves data pre-processing. It helps to improve data quality, addresses issues such as missing values and outliers. Proper data pre-processing can ultimately lead to more accurate and reliable results.

- Summing all the 'FLAG_DOCUMENT' columns to a single column which provides the information that how many documents does the client has provided.

- Shifting the target column to the end of the table

- Next step is Label encoding which is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine learning.

- As the above process was performed in python notebook. So, we have converted the pre-processing result to csv file

Step 3:

Loading the csv file to the shared file, so that it can be used in cloudera interface.

Step 4:

Loading the data to the Hadoop file.

Step 5:

Now comes the main machine learning part which is performed in ‘spark-shell’ and we have used scalar language to build the model.

The steps are as follows:

- Import the needed libraries.
- Transform each qualitative data in the data set into a double numeric value.
- Read data into memory in - lazy loading.
- Prepare data for the logistic regression algorithm.
- Split data into training (60%) and test (40%).
- Train the model.
- Evaluate model on training examples and compute training error.

CODE AND THE RESULT:

1.Loading the data to the Hadoop directory.

- hadoop fs -copyFromLocal defaultvv.csv

```
[cloudera@quickstart windows]$ hadoop fs -copyFromLocal defaultvv.csv
[cloudera@quickstart windows]$ hadoop fs -ls
Found 14 items
-rw-r--r--   1 cloudera cloudera       3893 2023-09-10 21:29 Qualitative_Bankrup
tcy.txt
-rw-r--r--   1 cloudera cloudera   20210380 2023-09-11 01:15 default.csv
-rw-r--r--   1 cloudera cloudera   14060160 2023-09-12 00:35 defaultvv.csv
drwxr-xr-x   - cloudera cloudera          0 2023-09-04 02:14 jpr
drwxr-xr-x   - cloudera cloudera          0 2023-08-29 21:00 jprfiles
-rw-r--r--   1 cloudera cloudera   43637043 2023-09-10 23:25 loan.csv
```

2. Import the needed libraries

- import org.apache.spark.mllib.evaluation.MulticlassMetrics
  import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, LogisticRegressionModel}
  import org.apache.spark.mllib.regression.LabeledPoint

  import org.apache.spark.mllib.linalg.{Vector, Vectors}

```
scala> import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, Logis
ticRegressionModel}
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.{Vector, Vectors}
import org.apache.spark.mllib.evaluation.MulticlassMetrics
import org.apache.spark.mllib.classification.{LogisticRegressionWithLBFGS, Logis
ticRegressionModel}
import org.apache.spark.mllib.regression.LabeledPoint
import org.apache.spark.mllib.linalg.{Vector, Vectors}
```

3. Transform each qualitative data in the data set into a double numeric value

- def getDoubleValue (input:String) : Double = {
  var result: Double = 0.0
  if (input == "0") result = 0.0
  if (input == "1") result = 1.0
  if (input == "2") result = 2.0
  if (input == "3") result = 3.0
  if (input == "4") result = 4.0
  if (input == "5") result = 5.0
  if (input == "6") result = 6.0
  if (input == "7") result = 7.0
  return result
  }

```
scala> def getDoubleValue( input:String ) : Double = {
    var result:Double = 0.0
    if (input == "0")  result = 0.0
    if (input == "1")  result = 1.0
    if (input == "2")  result = 2.0
    if (input == "3")  result = 3.0
    if (input == "4")  result = 4.0
    if (input == "5")  result = 5.0
    if (input == "6")  result = 6.0
    if (input == "7")  result = 7.0
    return result
   }
getDoubleValue: (input: String)Double
```

4. Read data into memory in - lazy loading

- val data = sc.textFile("/user/cloudera/defaultvv.csv")
  data.count()

```
scala> val data = sc.textFile("/user/cloudera/defaultvv.csv")
data: org.apache.spark.rdd.RDD[String] = /user/cloudera/defaultvv.csv MapPartiti
onsRDD[1] at textFile at <console>:31

scala> data.count()
res0: Long = 307511
```

5. Prepare data for the logistic regression algorithm

- val parsedData = data.map{line =>
  val parts = line.split(",")
  LabeledPoint(getDoubleValue(parts(15)), Vectors.dense(parts.slice(0,15).map(x =>getDoubleValue(x))))
  }
  println(parsedData.take(10).mkString("\n"))

```
scala> val parsedData = data.map{line =>
val parts = line.split(",")
LabeledPoint(getDoubleValue(parts(15)), Vectors.dense(parts.slice(0,15).map(x =>
getDoubleValue(x))))
}
println(parsedData.take(10).mkString("\n"))
(1.0,[0.0,1.0,0.0,0.0,0.0,7.0,4.0,3.0,1.0,0.0,1.0,0.0,0.0,5.0,0.0])
(0.0,[0.0,0.0,0.0,0.0,0.0,4.0,1.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,1.0])
(0.0,[1.0,1.0,0.0,0.0,0.0,7.0,4.0,3.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0])
(0.0,[0.0,1.0,0.0,0.0,0.0,7.0,4.0,0.0,1.0,0.0,1.0,0.0,0.0,5.0,1.0])
(0.0,[0.0,1.0,0.0,0.0,0.0,7.0,4.0,3.0,1.0,0.0,1.0,0.0,0.0,0.0,1.0])
(0.0,[0.0,1.0,0.0,0.0,0.0,4.0,4.0,1.0,1.0,0.0,1.0,0.0,0.0,0.0,1.0])
(0.0,[1.0,1.0,0.0,0.0,0.0,1.0,1.0,1.0,1.0,0.0,1.0,0.0,0.0,5.0,1.0])
(0.0,[1.0,1.0,0.0,0.0,0.0,4.0,1.0,1.0,1.0,0.0,2.0,0.0,0.0,0.0,1.0])
(0.0,[0.0,1.0,0.0,0.0,0.0,3.0,4.0,1.0,1.0,0.0,1.0,0.0,0.0,0.0,1.0])
(0.0,[0.0,1.0,0.0,0.0,0.0,7.0,4.0,3.0,1.0,0.0,1.0,0.0,0.0,0.0,0.0])
parsedData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPo
int] = MapPartitionsRDD[2] at map at <console>:35
```

6. Split data into training (60%) and test (40%)

- val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)
  val trainingData = splits(0)
  val testData = splits(1)

```
scala> val splits = parsedData.randomSplit(Array(0.6, 0.4), seed = 11L)
val trainingData = splits(0)
val testData = splits(1)
splits: Array[org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.Labeled
Point]] = Array(MapPartitionsRDD[3] at randomSplit at <console>:37, MapPartition
sRDD[4] at randomSplit at <console>:37)
trainingData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.Labeled
Point] = MapPartitionsRDD[3] at randomSplit at <console>:37
testData: org.apache.spark.rdd.RDD[org.apache.spark.mllib.regression.LabeledPoin
t] = MapPartitionsRDD[4] at randomSplit at <console>:37
```

7. Train the model

- val model = new
  LogisticRegressionWithLBFGS().setNumClasses(2).run(trainingData)

```
scala> val model = new LogisticRegressionWithLBFGS().setNumClasses(2).run(traini
ngData)
23/09/12 00:49:17 WARN classification.LogisticRegressionWithLBFGS: The input dat
a is not directly cached, which may hurt performance if its parent RDDs are also
 uncached.
23/09/12 00:49:26 WARN netlib.BLAS: Failed to load implementation from: com.gith
ub.fommil.netlib.NativeSystemBLAS
23/09/12 00:49:26 WARN netlib.BLAS: Failed to load implementation from: com.gith
ub.fommil.netlib.NativeRefBLAS
23/09/12 00:49:30 WARN classification.LogisticRegressionWithLBFGS: The input dat
a was not directly cached, which may hurt performance if its parent RDDs are als
o uncached.
model: org.apache.spark.mllib.classification.LogisticRegressionModel = org.apach
e.spark.mllib.classification.LogisticRegressionModel: intercept = 0.0, numFeatur
es = 15, numClasses = 2, threshold = 0.5
```

8. Evaluate model on training examples and compute training error

- val labelAndPreds = testData.map{ point =>
  val prediction = model.predict(point.features)
  (point.label, prediction)
  }
  val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / testData.count
  println("Test Error = " + testErr)

```
scala> val labelAndPreds = testData.map{ point =>
val prediction = model.predict(point.features)
(point.label, prediction)
}
val testErr = labelAndPreds.filter(r => r._1 != r._2).count.toDouble / testData.
count
println("Test Error = " + testErr)
Test Error = 0.08075414830178895
labelAndPreds: org.apache.spark.rdd.RDD[(Double, Double)] = MapPartitionsRDD[30]
 at map at <console>:45
testErr: Double = 0.08075414830178895
```

CONCLUSION:

In conclusion, our project built an accurate logistic regression model for predicting which client will default on their loans based on their financial information.

We achieved a test error rate of 0.080, which is a promising result and demonstrates the effectiveness of our model.

----------THANK YOU----------