SparkScala Final Project -

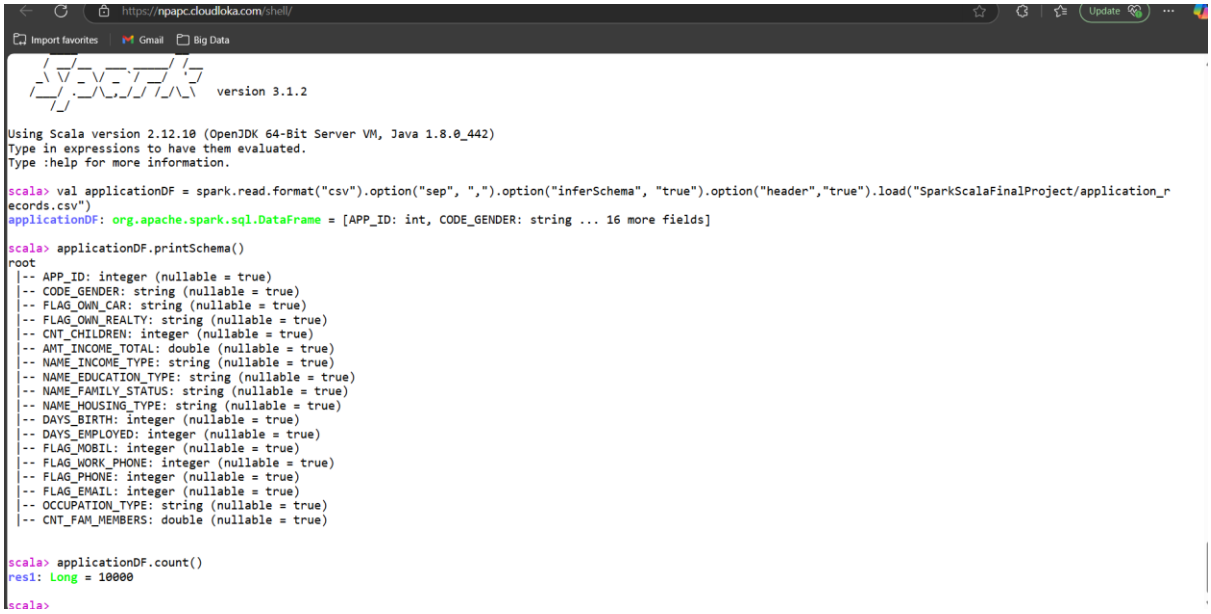
Credit Card Approval Application in Spark

1. Introduction

This document outlines the implementation of a Decision Tree Classifier in Spark ML, including key data processing steps, model training, and evaluation of its performance metrics.

2. Data Preparation

- Loading the dataset into respective data frames and performing the initial checks.



```
https://npapc.cloudloka.com/shell/
Import favorites   Gmail   Big Data
      /  __/__  ___ _____/ /__
     _\ \/ _ \/ _ `/ __/  '_/
    /___/ .__/\_,_/_/ /_/\_\   version 3.1.2
       /_/

Using Scala version 2.12.10 (OpenJDK 64-Bit Server VM, Java 1.8.0_442)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val applicationDF = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header","true").load("SparkScalaFinalProject/application_r
ecords.csv")
applicationDF: org.apache.spark.sql.DataFrame = [APP_ID: int, CODE_GENDER: string ... 16 more fields]

scala> applicationDF.printSchema()
root
 |-- APP_ID: integer (nullable = true)
 |-- CODE_GENDER: string (nullable = true)
 |-- FLAG_OWN_CAR: string (nullable = true)
 |-- FLAG_OWN_REALTY: string (nullable = true)
 |-- CNT_CHILDREN: integer (nullable = true)
 |-- AMT_INCOME_TOTAL: double (nullable = true)
 |-- NAME_INCOME_TYPE: string (nullable = true)
 |-- NAME_EDUCATION_TYPE: string (nullable = true)
 |-- NAME_FAMILY_STATUS: string (nullable = true)
 |-- NAME_HOUSING_TYPE: string (nullable = true)
 |-- DAYS_BIRTH: integer (nullable = true)
 |-- DAYS_EMPLOYED: integer (nullable = true)
 |-- FLAG_MOBIL: integer (nullable = true)
 |-- FLAG_WORK_PHONE: integer (nullable = true)
 |-- FLAG_PHONE: integer (nullable = true)
 |-- FLAG_EMAIL: integer (nullable = true)
 |-- OCCUPATION_TYPE: string (nullable = true)
 |-- CNT_FAM_MEMBERS: double (nullable = true)


scala> applicationDF.count()
res1: Long = 10000

scala>
```

https://npapc.cloudloka.com/shell/

Import favorites | Gmail | Big Data

```
|    min|          5008810|          F|           N|           N|                 0|          28723.5|Commercial associate|     Academic degree|    Civil marriage|
  Co-op apartment|             -24345|            -15713|         1|                 0|                 0|                 0|         Accountants|                1.0
|
|    max|          7999696|          M|           Y|           Y|                 9|        3375000.0|             Working|Secondary / secon...|             Widow|
    With parents|              -7900|               -28|         1|                 1|                 1|                 1|Waiters/barmen staff|               11.0
|
+-------+-----------------+-----------+------------+------------+------------------+-----------------+--------------------+--------------------+------------------+
-----------------+-----------------+-----------+----------+-----------------+-----------------+-----------------+-----------------+-----------------
+


scala> val creditDF = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header","true").load("SparkScalaFinalProject/credit_records.csv
")
creditDF: org.apache.spark.sql.DataFrame = [CR_ID: int, MONTHS_BALANCE: int ... 1 more field]

scala> creditDF.printSchema()
root
 |-- CR_ID: integer (nullable = true)
 |-- MONTHS_BALANCE: integer (nullable = true)
 |-- STATUS: string (nullable = true)


scala> creditDF.count()
res9: Long = 13090

scala> creditDF.show(5, false)
+-------+--------------+------+
|CR_ID  |MONTHS_BALANCE|STATUS|
+-------+--------------+------+
|5010253|-13           |X     |
|5008900|-23           |C     |
|5095383|-3            |C     |
|5126835|-46           |X     |
|5062030|-24           |0     |
+-------+--------------+------+
only showing top 5 rows


scala>
```

https://npapc.cloudloka.com/shell/

Import favorites | Gmail | Big Data

```
scala> val creditDF3 = creditDF2.join(applicationDF2,creditDF2("CR_ID")===applicationDF2("APP_ID"), "inner")
creditDF3: org.apache.spark.sql.DataFrame = [CR_ID: int, MONTHS_BALANCE: int ... 19 more fields]

scala> creditDF3.count()
res15: Long = 13090

scala> creditDF3.printSchema()
<console>:28: error: not found: value ccreditDF3
       ccreditDF3.printSchema()
       ^

scala> creditDF3.printSchema()
root
 |-- CR_ID: integer (nullable = true)
 |-- MONTHS_BALANCE: integer (nullable = true)
 |-- STATUS: string (nullable = true)
 |-- APP_ID: integer (nullable = true)
 |-- CODE_GENDER: string (nullable = true)
 |-- FLAG_OWN_CAR: string (nullable = true)
 |-- FLAG_OWN_REALTY: string (nullable = true)
 |-- CNT_CHILDREN: integer (nullable = true)
 |-- AMT_INCOME_TOTAL: double (nullable = true)
 |-- NAME_INCOME_TYPE: string (nullable = true)
 |-- NAME_EDUCATION_TYPE: string (nullable = true)
 |-- NAME_FAMILY_STATUS: string (nullable = true)
 |-- NAME_HOUSING_TYPE: string (nullable = true)
 |-- DAYS_BIRTH: integer (nullable = true)
 |-- DAYS_EMPLOYED: integer (nullable = true)
 |-- FLAG_MOBIL: integer (nullable = true)
 |-- FLAG_WORK_PHONE: integer (nullable = true)
 |-- FLAG_PHONE: integer (nullable = true)
 |-- FLAG_EMAIL: integer (nullable = true)
 |-- OCCUPATION_TYPE: string (nullable = true)
 |-- CNT_FAM_MEMBERS: double (nullable = true)
```

Connect

3. **Data Transformation**: Applying following transformations to the data

Bucketizer: Binned numerical columns into categorical values based on predefined splits.

https://npapc.cloudloka.com/shell/

Import favorites | Gmail | Big Data

```
scala> val childrenSplits = Array(0.0, 1.0, 2.0, 3.0)
childrenSplits: Array[Double] = Array(0.0, 1.0, 2.0, 3.0)
```

https://npapc.cloudloka.com/shell/

Import favorites | Gmail | Big Data

```
scala> val familySplits = Array(1.0, 2.0, 3.0, 4.0, 5.0)
familySplits: Array[Double] = Array(1.0, 2.0, 3.0, 4.0, 5.0)
```

https://npapc.cloudloka.com/shell/

Import favorites | Gmail | Big Data

```
scala> import org.apache.spark.ml.feature.Bucketizer
import org.apache.spark.ml.feature.Bucketizer

scala> val bucketizer = new Bucketizer().
     |   setInputCols(Array("CNT_CHILDREN", "CNT_FAM_MEMBERS")).
     |   setOutputCols(Array("childnum", "familynum")).
     |   setSplitsArray(Array(childrenSplits, familySplits))
bucketizer: org.apache.spark.ml.feature.Bucketizer = Bucketizer: uid=bucketizer_f7c5e174556c, numInputCols=2, numOutputCols=2
```

QuantileDiscretizer: Divided continuous numeric columns into equal-sized quantile-based bins for:

- Income: Created income categories.
- Age: Transformed days of birth into age groups.
- Employment Duration: Derived employment duration groups.

```
https://npapc.cloudloka.com/shell/
Import favorites   Gmail   Big Data

scala> import org.apache.spark.ml.feature.QuantileDiscretizer
import org.apache.spark.ml.feature.QuantileDiscretizer

scala> val incomeDiscretizer = new QuantileDiscretizer()
incomeDiscretizer: org.apache.spark.ml.feature.QuantileDiscretizer = quantileDiscretizer_ab23f75fe467

scala>

scala>   .setInputCols(Array("AMT_INCOME_TOTAL", "DAYS_BIRTH", "DAYS_EMPLOYED"))
res17: incomeDiscretizer.type = quantileDiscretizer_ab23f75fe467

scala>

scala>   .setOutputCols(Array("income_category", "age_group", "emp_group"))
res18: res17.type = quantileDiscretizer_ab23f75fe467

scala>

scala>   .setNumBuckets(7)
res19: res18.type = quantileDiscretizer_ab23f75fe467
```

StringIndexer: Encoded categorical columns into numerical indices.

```
https://npapc.cloudloka.com/shell/
Import favorites   Gmail   Big Data

scala> import org.apache.spark.ml.feature.StringIndexer
import org.apache.spark.ml.feature.StringIndexer

scala> val stringIndexer = new StringIndexer().
     |
     |   setInputCols(Array("CODE_GENDER", "FLAG_OWN_CAR", "FLAG_OWN_REALTY", "NAME_INCOME_TYPE",
     |
     |                       "NAME_EDUCATION_TYPE", "NAME_FAMILY_STATUS", "NAME_HOUSING_TYPE", "OCCUPATION_TYPE")).
     |
     |   setOutputCols(Array("gender_index", "own_car_index", "own_realty_index", "income_type_index",
     |
     |                        "education_type_index", "family_status_index", "housing_type_index", "occupation_type_index"))
stringIndexer: org.apache.spark.ml.feature.StringIndexer = strIdx_361cdb9e9cd4
```

Vector Assembler: Combined multiple feature columns into a single vector column for model input.

```
https://npapc.cloudloka.com/shell/
Import favorites   Gmail   Big Data

scala> import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.feature.VectorAssembler
```

```
https://npapc.cloudloka.com/shell/
Import favorites   Gmail   Big Data

scala> val assembler = new VectorAssembler().
     |   setInputCols(Array("childnum", "familynum", "income_category", "age_group", "emp_group",
     |                      "gender_index", "own_car_index", "own_realty_index", "income_type_index",
     |                      "education_type_index", "family_status_index", "housing_type_index", "occupation_type_index")).
     |   setOutputCol("features")
assembler: org.apache.spark.ml.feature.VectorAssembler = VectorAssembler: uid=vecAssembler_119b2edb3cc1, handleInvalid=error, numInputCols=13
```

4. Model Building and Training

Algorithm: Decision Tree Classifier from Spark ML.

Split the dataset into training and test sets and trained the model using the training data.

```
https://npapc.cloudloka.com/shell/
Import favorites   Gmail   Big Data

scala> import org.apache.spark.ml.classification.DecisionTreeClassifier
import org.apache.spark.ml.classification.DecisionTreeClassifier

scala> import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.Pipeline

scala> import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator
import org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator

scala

scala> val dtc = new DecisionTreeClassifier().
     |   setLabelCol("label").
     |   setFeaturesCol("features").
     |   setMaxBins(32)
dtc: org.apache.spark.ml.classification.DecisionTreeClassifier = dtc_c46b0ea2860a

scala> val pipeline = new Pipeline().
     |   setStages(Array(bucketizer, incomeDiscretizer, stringIndexer, assembler, dtc))
pipeline: org.apache.spark.ml.Pipeline = pipeline_0327981caf44

scala> val Array(trainingData, testData) = creditDF4.randomSplit(Array(0.7, 0.3))
trainingData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [CR_ID: int, MONTHS_BALANCE: int ... 20 more fields]
testData: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [CR_ID: int, MONTHS_BALANCE: int ... 20 more fields]
```

5. Model Evaluation:

Metric Used:

- Evaluated model accuracy using MulticlassClassificationEvaluator.
- Metric name: accuracy.

```
https://npapc.cloudloka.com/shell/
Import favorites | Gmail | Big Data

scala> val model = pipeline.fit(trainingData)
model: org.apache.spark.ml.PipelineModel = pipeline_0327981caf44

scala> val model = pipeline.fit(trainingData)
model: org.apache.spark.ml.PipelineModel = pipeline_0327981caf44

scala> val predictions = model.transform(testData)
predictions: org.apache.spark.sql.DataFrame = [CR_ID: int, MONTHS_BALANCE: int ... 37 more fields]

scala> val evaluator = new MulticlassClassificationEvaluator()
evaluator: org.apache.spark.ml.evaluation.MulticlassClassificationEvaluator = MulticlassClassificationEvaluator: uid=mcEval_fb6f77e72a74, metricName=f1, metricLabel=0.
0, beta=1.0, eps=1.0E-15

scala>

scala>   .setLabelCol("label")
res20: evaluator.type = MulticlassClassificationEvaluator: uid=mcEval_fb6f77e72a74, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>

scala>   .setPredictionCol("prediction")
res21: res20.type = MulticlassClassificationEvaluator: uid=mcEval_fb6f77e72a74, metricName=f1, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala>

scala>   .setMetricName("accuracy")
res22: res21.type = MulticlassClassificationEvaluator: uid=mcEval_fb6f77e72a74, metricName=accuracy, metricLabel=0.0, beta=1.0, eps=1.0E-15

scala> val accuracy = evaluator.evaluate(predictions)
accuracy: Double = 0.9946126218573628

scala> println(s"Accuracy: $accuracy")
Accuracy: 0.9946126218573628

scala> println(s"Test Error: ${1.0 - accuracy}")
Test Error: 0.005387378142637234
```

6. Results

Accuracy = 0.9946126218573628 (Perfect predictions made by the classifier).

Test Error = 0.005387378142637234

7. Conclusion

The Decision Tree Classifier demonstrated exceptional performance on the Credit Card Approval dataset, achieving an **accuracy of 99.46%** and a **test error of only 0.54%**. These results indicate that the model was able to classify applicants with near-perfect precision, making it highly reliable for predicting 'good' or 'bad' clients.