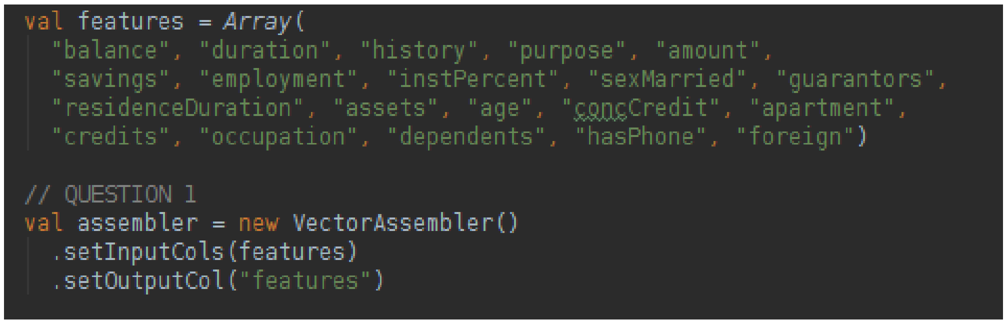
Ceng790 Big Data Analytics
Assignment 3
Random Forest Classifier

Egemen Berk Galatali
e2099018@ceng.metu.edu.tr

Part I

1. Use a VectorAssembler to transform and return a new dataframe with all of the feature columns in a vector column.

   We extract a feature vector for each row in our data. Every column except creditability constitues the feature vector. We specify which column values should be in the vector while creating VectorAssembler object(see features Array). Therefore, our model will take feature vector for each row, later we will add label to those vectors.



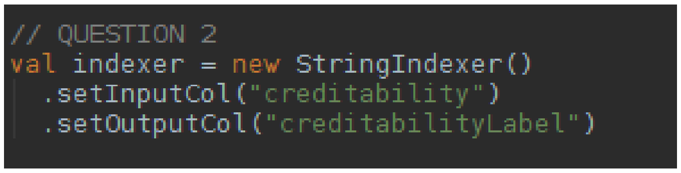
```
val features = Array(
  "balance", "duration", "history", "purpose", "amount",
  "savings", "employment", "instPercent", "sexMarried", "guarantors",
  "residenceDuration", "assets", "age", "concCredit", "apartment",
  "credits", "occupation", "dependents", "hasPhone", "foreign")

// QUESTION 1
val assembler = new VectorAssembler()
  .setInputCols(features)
  .setOutputCol("features")
```

Figure 1: Vector Assembler

2. Use a StringIndexer to return a Dataframe with the creditability column added as a label

   We create label based on creditability. This operation creates a new column. Its value is 1 if the creditability is 1 and 0 otherwise. By doing this we actually create a pair that is (featureVector, label) that will be later used by the machine learning classifier to train the model.



```
// QUESTION 2
val indexer = new StringIndexer()
  .setInputCol("creditability")
  .setOutputCol("creditabilityLabel")
```

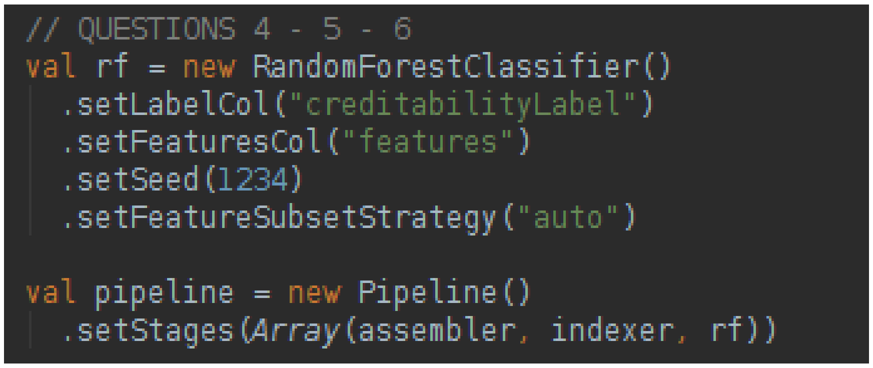
Figure 2: String Indexer

3. Use randomSplit function to split the data into three sets: 80% of the data is used to train (and tune) the model, 20% will be used for testing.

   We split the data(creditDF) by calling randomSplit(Array(0.8, 0.2)) into training and test Datasets.

4. Train the model and optimize hyperparameters

   We create a randomForest Classifier and evaluator below.

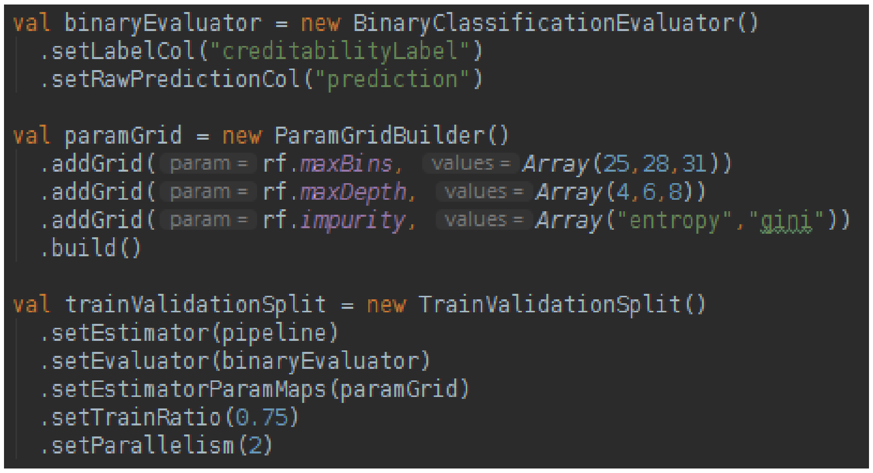


```
// QUESTIONS 4 - 5 - 6
val rf = new RandomForestClassifier()
  .setLabelCol("creditabilityLabel")
  .setFeaturesCol("features")
  .setSeed(1234)
  .setFeatureSubsetStrategy("auto")

val pipeline = new Pipeline()
  .setStages(Array(assembler, indexer, rf))
```

Figure 3: Model and Evaluator

We create binaryEvaluator object to later evaluate the performance our model. We create ParamGrid object as specified in the assignment for pipeline in order to do hyper-parameter tuning. There are total 18 different combinations for a new model to be trained with. Training of each model with different hyper-parameters and testing is done on the 80% of our whole data that was created before. Later, we do prediction on 20% of the data and get the accuracy score.



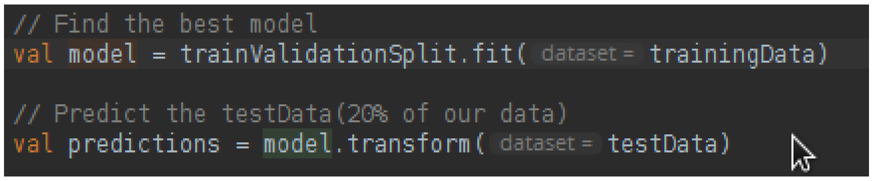
```
val binaryEvaluator = new BinaryClassificationEvaluator()
  .setLabelCol("creditabilityLabel")
  .setRawPredictionCol("prediction")

val paramGrid = new ParamGridBuilder()
  .addGrid( param = rf.maxBins,  values = Array(25,28,31))
  .addGrid( param = rf.maxDepth,  values = Array(4,6,8))
  .addGrid( param = rf.impurity,  values = Array("entropy","gini"))
  .build()

val trainValidationSplit = new TrainValidationSplit()
  .setEstimator(pipeline)
  .setEvaluator(binaryEvaluator)
  .setEstimatorParamMaps(paramGrid)
  .setTrainRatio(0.75)
  .setParallelism(2)
```

Figure 4: Train and Tune

At the end we get the best model out of these computations. Then we try to predict testData that is 20% of our whole data with our best model.



```
// Find the best model
val model = trainValidationSplit.fit( dataset = trainingData)

// Predict the testData(20% of our data)
val predictions = model.transform( dataset = testData)
```

Figure 5: Predict

5. Stats

I have obtained accuracy results between 0.55-0.69 on training set and 0.62-0.67 on test set. I got the highest results with parameters as follows: maxDepth=8, maxBins=25 and 28 and impurity=entropy. I have also used Cross-Validation as opposed to Train-Validation because in the spark's document it is stated as the following that:

> Train-Validation Split only evaluates each combination of parameters once, as opposed to k times in the case of CrossValidator. It is, therefore, less expensive, but will not produce as reliable results when the training dataset is not sufficiently large [1]

So since our dataset is a small one I thought the model could be improved. I tested with different fold numbers(3,5,10). But results did not change so much. With Cross-Validation results remained above 0.65 and below 0.72 and it executed slower compared Train-Validation split

[1] Train-Validation Split