Mital Bhanushali – SEC01 (NUID 002919890)

Big Data System Engineering with Scala  
Spring 2023   
Assignment: Spark #2



**-List of Tasks Implemented**

* Exploratory Data Analysis
* // Load the training and test datasets  
  val *trainDF* = *spark*.read.format("csv")  
   .option("header", "true")  
   .option("inferSchema", "true")  
   .load("D:/Scala/Spark\_ML/src/test/resources/train.csv")  
    
  val *testDF* = *spark*.read.format("csv")  
   .option("header", "true")  
   .option("inferSchema", "true")  
   .load("D:/Scala/Spark\_ML/src/test/resources/test.csv")  
    
  // Perform exploratory data analysis on the training dataset  
  *trainDF*.printSchema()  
  *trainDF*.show(20)  
    
  // Calculate the mean age  
  val *meanAge* = *trainDF*.select(*mean*("age")).head().getDouble(0).round.toInt  
  val *meanAgeTest* = *testDF*.select(*mean*("age")).head().getDouble(0).round.toInt  
    
  // Fill null values in "age" column with the mean age  
  val *trainData* = *trainDF*.na.fill(*meanAge*, Seq("age"))  
  *println*("TrainData Details:")  
  *trainData*.show(20)  
    
  val *testData* = *testDF*.na.fill(*meanAgeTest*, Seq("age"))  
  *println*("testData Details:")  
  *testData*.show(20)  
    
    
  *println*("Total No Of Rows:" + *trainData*.count())  
  *trainData*.describe().show()  
  //trainDF.describe("Age", "SibSp", "Parch", "Fare").show()  
    
    
  //count number of passengers based on sex  
  *println*("Gender wise count of Passengers")  
  *trainData*.groupBy("Sex").count().show(false)  
    
  //count sum of total ticket fare by class type  
  *println*("Total fare by class")  
  *trainData*.groupBy("Pclass").sum("Fare").show(false)  
    
  //trainData.groupBy("Sex","Age")  
  val *children* = *sum*(*when*(*col*("age") < 19, 1).otherwise(0)).as("children\_count")  
  val *adults* = *sum*(*when*(*col*("age").between(19, 60), 1).otherwise(0)).as("adults\_count")  
  val *seniorcitizen* = *sum*(*when*(*col*("age") > 60, 1).otherwise(0)).as("seniorcitizen\_count")  
    
  *println*("children " + *children*)  
  //No of passengers grouped by age  
  *println*("Total no of passengers grouped by age")  
  val *result* = *trainData*.select(*children*, *adults*, *seniorcitizen*)  
  *result*.show()

//no of passengers grouped by age and gender  
*trainData*.groupBy("sex").agg(*children*, *adults*, *seniorcitizen*).show()

* Feature Engineering
* val *trainData1* = *trainData* .withColumn("FamilySize", *col*("SibSp") + *col*("Parch") + 1)  
   .withColumn("IsAlone", *when*(*col*("FamilySize") === 1, 1).otherwise(0))  
   .drop("Embarked", "Fare","Cabin","Ticket")  
    
  val *testData1* = *testData* .withColumn("FamilySize", *col*("SibSp") + *col*("Parch") + 1)  
   .withColumn("IsAlone", *when*(*col*("FamilySize") === 1, 1).otherwise(0))  
   .drop( "Embarked", "Fare", "Cabin", "Ticket")  
    
    
  *trainData1*.show(20)  
    
  // encode categorical variables as numerical  
  val *indexer1* = new StringIndexer().setInputCol("Sex").setOutputCol("SexIndex")  
  val *indexed1* = *indexer1*.fit(*trainData1*).transform(*trainData1*)  
    
  // encode categorical variables as numerical for the test data  
  val *indexer2* = new StringIndexer().setInputCol("Sex").setOutputCol("SexIndex")  
  val *indexed2* = *indexer2*.fit(*testData1*).transform(*testData1*)  
    
  *println*("Index1 " )  
  *indexed1*.show(5)  
    
  *trainData1*.show(20)  
  // assemble the features  
  val *assembler* = new VectorAssembler()  
   .setInputCols(*Array*("PassengerId","Pclass", "SexIndex", "Age", "FamilySize"))  
   .setOutputCol("features")  
  val *featureDF* = *assembler*.transform(*indexed1*).select(*col*("Survived"), *col*("features"))  
  val *testFeatures* = *assembler*.transform(*indexed2*).select(*col*("SexIndex"),*col*("Pclass"),*col*("PassengerId"), *col*("features"))
* Prediction

//If we want to use same data for training and testingwe need to split it into some ratio like below:  
 //Since there is no columna named survived in test data, I preferred to divide train data into 80-20 ratio to  
 // test the accuracy of model  
 // split the data into training and validation sets  
 val *Array*(*trainingData*, *validationData*) = *featureDF*.randomSplit(*Array*(0.8, 0.2), seed = 123)  
  
 // create a logistic regression model  
 val *lr* = new LogisticRegression().setLabelCol("Survived").setFeaturesCol("features")  
  
 // fit the model to the training data  
 val *model* = *lr*.fit(*trainingData*)  
  
 // make predictions on the validation data  
 val *predictions* = *model*.transform(*validationData*)  
  
 // evaluate the model using binary classification metrics  
 val *evaluator* = new BinaryClassificationEvaluator().setLabelCol("Survived").setRawPredictionCol("prediction")  
 val *areaUnderROC* = *evaluator*.evaluate(*predictions*)  
  
 //val predictions2 = model.transform(testData)  
  
 // print the area under ROC  
 *println*("AreaUnderROC using BinaryClassificationEvaluator = " + *areaUnderROC*)  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
  
 val *testPredictions* = *model*.transform(*testFeatures*)  
 *testPredictions*.show(5)  
 val *evaluator1* = new BinaryClassificationEvaluator().setLabelCol("Survived").setRawPredictionCol("testPredictions")  
 // Survived does not exist in test data so we can't evaluate accuracy, else it could have been done as below:  
 //val areaUnderROC1 = evaluator.evaluate(testPredictions)  
 //println("Area under ROC1 = " + areaUnderROC1)  
  
 *testPredictions*.select(*col*("SexIndex"), *col*("Pclass"),*col*("PassengerId"), *col*("prediction").cast("Int").alias("Survived")).show(5)  
  
  
 val *evaluator2* = new MulticlassClassificationEvaluator().setLabelCol("Survived").setPredictionCol("prediction")  
 val *accuracy2* = *evaluator2*.evaluate(*predictions*)  
  
 *println*("accuracy using MulticlassClassificationEvaluator = " + *accuracy2*)  
  
  
}

**-Findings and analysis and Result**

Text

Description automatically generated

Got 78% accuracy.