**Assignment 2**

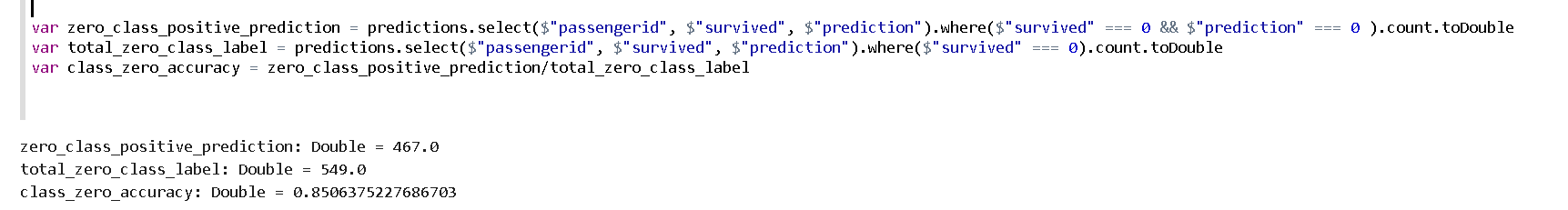
1. Using the model built in class for titanic dataset, calculate the accuracy of our model for passengers who survived and those didn't survive separately, i.e., accuracy for both possible outcomes.

// show accuracy for Label Class Zero - Not Survived

var zero\_class\_positive\_prediction = predictions.select($"passengerid", $"survived", $"prediction").where($"survived" === 0 && $"prediction" === 0 ).count.toDouble

var total\_zero\_class\_label = predictions.select($"passengerid", $"survived", $"prediction").where($"survived" === 0).count.toDouble

var class\_zero\_accuracy = zero\_class\_positive\_prediction/total\_zero\_class\_label

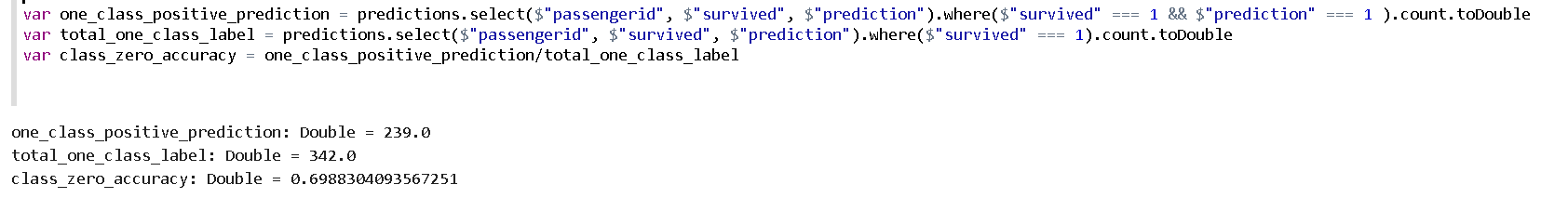


// show accuracy for Label Class One - Survived

var one\_class\_positive\_prediction = predictions.select($"passengerid", $"survived", $"prediction").where($"survived" === 1 && $"prediction" === 1 ).count.toDouble

var total\_one\_class\_label = predictions.select($"passengerid", $"survived", $"prediction").where($"survived" === 1).count.toDouble

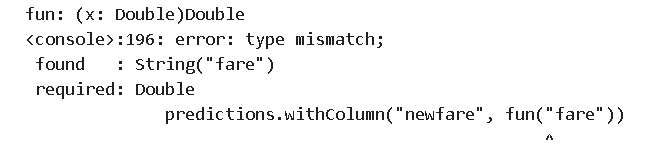
var class\_zero\_accuracy = one\_class\_positive\_prediction/total\_one\_class\_label



1. The 'fare' field in titanic dataset has continuous values in the range of zero to 512. This usually doesn't work well with most machine learning algorithms. Lets scale the fare values in the range of 0 to 1 and create a separate field (column) that has the scaled version of fare

def fun(x:Double) : Double = x / 512

predictions.withColumn("newfare", fun("fare"))



Note: I’m not able to debug the above error, even if I try to explicitly cast the variable to **DoubleType**. I keep getting type mismatch error.

1. We indexed the gender values in class exercise to zero or one (0 for male, 1 for female). A better practice is to convert such categorical fields to indicator fields. Create two additional fields in our titanic dataframe as:
   * IsMale: value would be 1 if the passenger was male otherwise 0
   * IsFemale: value would be 1 if the passenger was female otherwise 0

// isMale Function

val ismale = udf((gender: Double) =>

{

if (gender == 0) 1

else 0

})

// isFemale Function

val isfemale = udf((gender: Double) =>

{

if (gender == 1) 1

else 0

})

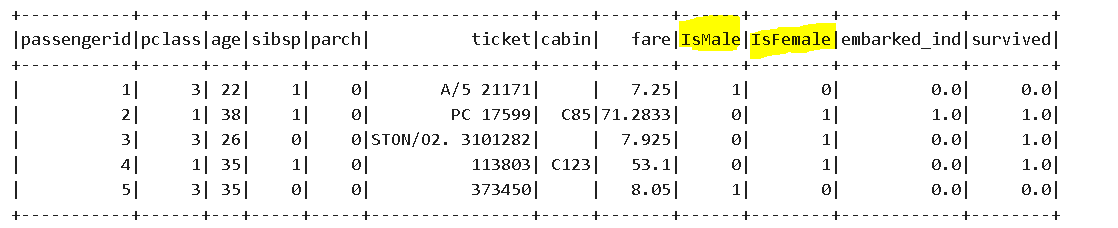
// Add new columns with isMale and isFemale

val df2 = predictions.withColumn("IsMale", ismale($"gender\_ind"))

val df3 = df2.withColumn("IsFemale", isfemale($"gender\_ind"))

var df4 = df3.select($"passengerid",$"pclass",$"age",$"sibsp",$"parch",$"ticket",$"cabin",$"fare",$"IsMale",$"IsFemale",$"embarked\_ind", $"survived")

df4.show(5)



1. Build the model again with changes in 2 & 3 above, build the same logistic regression model and calculate the accuracy

// Define new features array

val features = Array("pclass", "age" , "sibsp", "parch", "fare", "IsMale","IsFemale","embarked\_ind")

// put all the features into one vector called "features"

val t\_newvec = new VectorAssembler().setInputCols(features).setOutputCol("features").transform(df4)

t\_newvec.show(10)

/// Modeling

val model = (new LogisticRegression()).setLabelCol("survived").fit(t\_newvec)

// score the model

val newpredictions = model.transform(t\_newvec)

// show some predicted results

newpredictions.select($"passengerid", $"IsMale",$"IsFemale", $"survived", $"prediction", $"features", $"probability")

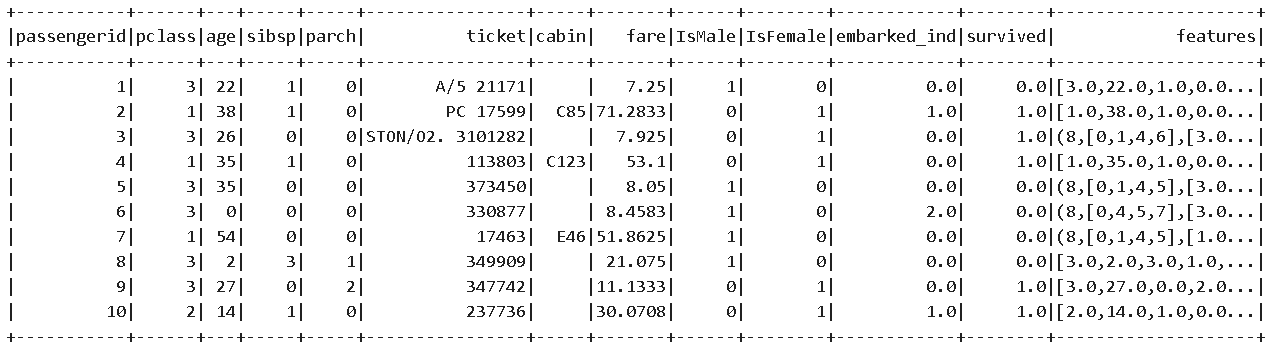
// calculate accuracy

val newaccuracy = newpredictions.filter($"survived" === $"prediction").count.toDouble/t.count

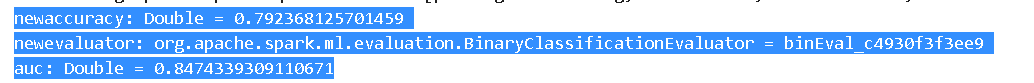
// evaluation metrics

val newevaluator = new BinaryClassificationEvaluator().setLabelCol("survived").setMetricName("areaUnderROC")

val auc = newevaluator.evaluate(predictions)



Accuracy value of 79% as shown below



1. We built and scored our model on the same data (t\_vec) which is not ideal. Split the dataset to use 70% of data for training and 30% for scoring. Calculate the accuracy on your test/scored data.

* Dataset was split between training and testing as 621 and 270 respectively with accuracy value of 56% to 23%

// split data between training and test

val splits = t\_newvec.randomSplit(Array(0.7, 0.3), seed = 11L)

val train = splits(0).cache()

val test = splits(1)

// Modelling using training and testing data

val trn\_model = (new LogisticRegression()).setLabelCol("survived").fit(train)

val test\_model = (new LogisticRegression()).setLabelCol("survived").fit(test)

// score the model

val trn\_predictions = trn\_model.transform(train).count

val test\_predictions = test\_model.transform(test).count

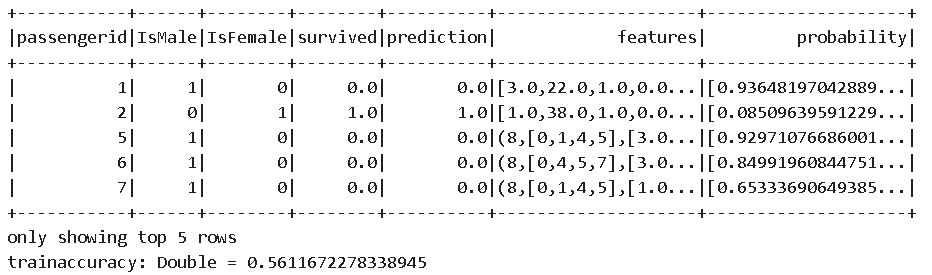
/// Modeling ---- training data the model using training data

val model = (new LogisticRegression()).setLabelCol("survived").fit(train)

val trainpredictions = model.transform(train)

trainpredictions.select($"passengerid", $"IsMale",$"IsFemale", $"survived", $"prediction", $"features", $"probability").show(5)

val trainaccuracy = trainpredictions.filter($"survived" === $"prediction").count.toDouble/t.count



/// Modeling ---- testing data the model using training data

val model = (new LogisticRegression()).setLabelCol("survived").fit(test)

val testpredictions = model.transform(test)

testpredictions.select($"passengerid", $"IsMale",$"IsFemale", $"survived", $"prediction", $"features", $"probability").show(5)

val testaccuracy = testpredictions.filter($"survived" === $"prediction").count.toDouble/t.count

