**KENNESAW STATE UNIVERSITY**

COLLEGE OF COMPUTING AND SOFTWARE ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE

CS7265-Big Data Analytics

Section W01-Spring 2022

PROJECT

Aswani I Chandran

Ignatius Jyosthna L

Kavitha Muppa

Bhargav Saripudi

Lakshmi Prasanna Pedarla

[achandr2@students.kennesaw.edu](mailto:achandr2@students.kennesaw.edu)

[ilingare@students.kennesaw.edu](mailto:ilingare@students.kennesaw.edu)

[kmuppa@students.kennesaw.edu](mailto:kmuppa@students.kennesaw.edu)

[bsaripud@students.kennesaw.edu](mailto:bsaripud@students.kennesaw.edu)

lpedarla@students.kennesaw.edu

05/05/2022

**SOURCE-CODE**

**Decision Tree & Random Forest**

*//Covid-19 dataset*

import org.apache.spark.ml.linalg.\_

import org.apache.spark.ml.regression.\_

import org.apache.spark.ml.classification.{DecisionTreeClassifier,RandomForestClassifier, RandomForestClassificationModel}

import org.apache.spark.ml.feature.{VectorAssembler, VectorIndexer}

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

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.ml.tuning.{ParamGridBuilder,TrainValidationSplit}

import org.apache.spark.ml.{PipelineModel, Pipeline}

import org.apache.spark.sql.types.{StructType,StructField,StringType,IntegerType};

*\*\*\*\*\*// Read the data//\*\*\*\**

val data = spark.read.option("header","true").option("inferSchema","true").format("csv").load("C:/Spark/Cleaned-Data.csv")

*// check data//*

Data.count

data.printSchema

*//drop insignificant columns*

val df=data.drop("Gender\_Female","Gender\_Male","Gender\_Transgender","Contact\_Dont-Know","Contact\_No","Contact\_Yes","Country")

df.printSchema

*//prepare data*

val datafinal = df.withColumn("Fever",$"Fever".cast(IntegerType)).withColumn("Tiredness",$"Tiredness".cast(IntegerType)).withColumn("Dry-Cough",$"Dry-Cough".cast(IntegerType)).withColumn("Difficulty-in-Breathing",$"Difficulty-in-Breathing".cast(IntegerType)).withColumn("Sore-Throat",$"Sore-Throat".cast(IntegerType)).withColumn("None\_Sympton",$"None\_Sympton".cast(IntegerType)).withColumn("Pains",$"Pains".cast(IntegerType)).withColumn("Nasal-Congestion",$"Nasal-Congestion".cast(IntegerType)).withColumn("Runny-Nose",$"Runny-Nose".cast(IntegerType)).withColumn("Diarrhea",$"Diarrhea".cast(IntegerType)).withColumn("None\_Experiencing",$"None\_Experiencing".cast(IntegerType)).withColumn("Age\_0-9",$"Age\_0-9".cast(IntegerType)).withColumn("Age\_10-19",$"Age\_10-19".cast(IntegerType)).withColumn("Age\_20-24",$"Age\_20-24".cast(IntegerType)).withColumn("Age\_25-59",$"Age\_25-59".cast(IntegerType)).withColumn("Age\_60+",$"Age\_60+".cast(IntegerType)).withColumn("Severity\_Mild",$"Severity\_Mild".cast(IntegerType)).withColumn("Severity\_Moderate",$"Severity\_Moderate".cast(IntegerType)).withColumn("Severity\_None",$"Severity\_None".cast(IntegerType)).withColumn("Severity\_Severe",$"Severity\_Severe".cast(IntegerType))

val data = datafinal.toDF()

val Severity = Array("Severity\_Mild","Severity\_Moderate","Severity\_None","Severity\_Severe")

*// Assemble the four target variables*

val Severityassembler = new VectorAssembler(). setInputCols(Severity). setOutputCol("severity")

//One-hot encoding of data

val unhotUDF = udf((vec:Vector)=>if(vec.toArray.sum==0)0.0 else vec.toArray.indexOf(1).toDouble)

*//prepare the final data*

val withseverity = Severityassembler.transform(data).drop(Severity:\_\*). withColumn("severity", unhotUDF($"severity"))

*//Split in to 80% train, 10% cross validation,10% test*

val Array(trainData, cvData, testData) = withseverity.randomSplit(Array(0.8,0.1, 0.1))

trainData.cache()

cvData.cache()

testData.cache()

val inputCols = trainData.columns.filter(\_ != "severity")

val assembler = new VectorAssembler().setInputCols(inputCols).setOutputCol("featureVector")

val assembledTrainData = assembler.transform(trainData)

assembledTrainData.select("featureVector").show(truncate = false)

val assembledCvData = assembler.transform(cvData)

assembledCvData.select("featureVector").show(truncate = false)

\*\*\*\*\*\*// first simple decision tree

\*\*\*\*\*\*//Target--> severity

val classifier = new DecisionTreeClassifier().setSeed(2345).setLabelCol("severity").setFeaturesCol("featureVector").setPredictionCol("prediction")

*// train the model*

val model = classifier.fit(assembledTrainData)

*// show the tree*

println(model.toDebugString)

*// explore the model*

model.featureImportances.toArray.zip(inputCols).sorted.reverse.take(10).foreach(println)

*// make predictions*

val predictions = model.transform(assembledCvData)

predictions.select("severity", "prediction", "probability").show(false)

*//Evaluate the model*

val evaluator = new MulticlassClassificationEvaluator().setLabelCol("severity").setPredictionCol("prediction")

val accuracy = evaluator.setMetricName("accuracy").evaluate(predictions)

val f1 = evaluator.setMetricName("f1").evaluate(predictions)

println(accuracy)

println(f1)

val weightedPrecision = evaluator.setMetricName("weightedPrecision").evaluate(predictions)

println(weightedPrecision)

val predictionRDD = predictions.select("prediction", "severity").as[(Double,Double)].rdd

val multiclassMetrics = new MulticlassMetrics(predictionRDD)

println(multiclassMetrics.confusionMatrix)

*// Create confusion Matrix*

val confusionMatrix = predictions.groupBy("severity").pivot("prediction", (0 to 3)).count().na.fill(0.0).orderBy("severity")

confusionMatrix.show()

val assembler = new VectorAssembler().setInputCols(inputCols).setOutputCol("featureVector")

val indexer = new VectorIndexer().setMaxCategories(40).setInputCol("featureVector").setOutputCol("indexedVector")

\*\*\*\*\*\*// Random Forest Classifier//\*\*\*\*\*\*\*\*

val classifier = new RandomForestClassifier().setSeed(123456).setLabelCol("severity").setFeaturesCol("indexedVector").setPredictionCol("prediction").setImpurity("entropy").setMaxDepth(20).setMaxBins(300)

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

*// create param grid*

val paramGrid = new ParamGridBuilder().addGrid(classifier.impurity, Seq("gini", "entropy")).addGrid(classifier.maxDepth, Seq(1, 20)).addGrid(classifier.maxBins, Seq(40, 300)).addGrid(classifier.minInfoGain, Seq(0.0, 0.05)).build()

val multiclassEval = new MulticlassClassificationEvaluator().setLabelCol("severity").setPredictionCol("prediction").setMetricName("accuracy")

val validator = new TrainValidationSplit().setSeed(12345).setEstimator(pipeline).setEvaluator(multiclassEval).setEstimatorParamMaps(paramGrid).setTrainRatio(0.9)

val validatorModel = validator.fit(trainData)

val bestModel = validatorModel.bestModel

val forestModel = bestModel.asInstanceOf[PipelineModel].stages.last.asInstanceOf[RandomForestClassificationModel]

println(forestModel.extractParamMap)

println(forestModel.getNumTrees)

forestModel.featureImportances.toArray.zip(inputCols).sorted.reverse.foreach(println)

val testAccuracy = multiclassEval.evaluate(bestModel.transform(testData))

println(testAccuracy)

//91.7%

**Logistic Regression**

import org.apache.spark.ml.linalg.\_

import org.apache.spark.ml.regression.\_

import org.apache.spark.ml.classification.LogisticRegression

import org.apache.spark.ml.{PipelineModel, Pipeline}

import org.apache.spark.mllib.evaluation.MulticlassMetrics

import org.apache.spark.ml.tuning.{ParamGridBuilder, TrainValidationSplit}

import org.apache.spark.ml.evaluation.RegressionEvaluator

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

import org.apache.spark.sql.types.{StructType,StructField,StringType,IntegerType};

import org.apache.spark.ml.feature.{VectorAssembler, VectorIndexer}

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

*//Read data*

val data = spark.read.option("header","true").option("inferSchema","true").format("csv").load("C:/Spark/Cleaned-Data.csv")

*//Check data*

data.printSchema

val df=data.drop("Gender\_Female","Gender\_Male","Gender\_Transgender","Contact\_Dont-Know","Contact\_No","Contact\_Yes","Country")

df.show

val datafinal = df.withColumn("Fever",$"Fever".cast(IntegerType)).withColumn("Tiredness",$"Tiredness".cast(IntegerType)).withColumn("Dry-Cough",$"Dry-Cough".cast(IntegerType)).withColumn("Difficulty-in-Breathing",$"Difficulty-in-Breathing".cast(IntegerType)).withColumn("Sore-Throat",$"Sore-Throat".cast(IntegerType)).withColumn("None\_Sympton",$"None\_Sympton".cast(IntegerType)).withColumn("Pains",$"Pains".cast(IntegerType)).withColumn("Nasal-Congestion",$"Nasal-Congestion".cast(IntegerType)).withColumn("Runny-Nose",$"Runny-Nose".cast(IntegerType)).withColumn("Diarrhea",$"Diarrhea".cast(IntegerType)).withColumn("None\_Experiencing",$"None\_Experiencing".cast(IntegerType)).withColumn("Age\_0-9",$"Age\_0-9".cast(IntegerType)).withColumn("Age\_10-19",$"Age\_10-19".cast(IntegerType)).withColumn("Age\_20-24",$"Age\_20-24".cast(IntegerType)).withColumn("Age\_25-59",$"Age\_25-59".cast(IntegerType)).withColumn("Age\_60+",$"Age\_60+".cast(IntegerType)).withColumn("Severity\_Mild",$"Severity\_Mild".cast(IntegerType)).withColumn("Severity\_Moderate",$"Severity\_Moderate".cast(IntegerType)).withColumn("Severity\_None",$"Severity\_None".cast(IntegerType)).withColumn("Severity\_Severe",$"Severity\_Severe".cast(IntegerType))

val data = datafinal.toDF()

val Severity = Array("Severity\_Mild","Severity\_Moderate","Severity\_None","Severity\_Severe")

*//Assemble the target variables*

val Severityassembler = new VectorAssembler(). setInputCols(Severity). setOutputCol("severity")

*//One-hot encoding*

val unhotUDF = udf((vec:Vector)=>if(vec.toArray.sum==0)0.0 else vec.toArray.indexOf(1).toDouble)

*//Set final data*

val withseverity = Severityassembler.transform(data).drop(Severity:\_\*). withColumn("severity", unhotUDF($"severity"))

val seed = 5043

*//Split in to 80% train, 10% cross validation,10% test*

val Array(trainData,testData) = withseverity.randomSplit(Array(0.8,0.2))

trainData.cache()

testData.cache()

val inputCols = trainData.columns.filter(\_ != "severity")

val assembler = new VectorAssembler().setInputCols(inputCols).setOutputCol("featureVector")

val indexer = new VectorIndexer().setMaxCategories(40).setInputCol("featureVector").setOutputCol("indexedVector")

**\*\*\*\*\*\*// Logistic Regression Classifier//\*\*\*\*\*\*\*\***

val classifier = new LogisticRegression().setLabelCol("severity").setFeaturesCol("indexedVector").setPredictionCol("prediction").setMaxIter(100)

.setRegParam(0.02)

.setElasticNetParam(1)

val classifier = new LogisticRegression().setLabelCol("severity").setFeaturesCol("indexedVector").setPredictionCol("prediction").setMaxIter(100)

val stages = Array(assembler, indexer, classifier)

val pipeline = new Pipeline().setStages(stages)

val multiclassEval = new MulticlassClassificationEvaluator().setLabelCol("severity").setPredictionCol("prediction").setMetricName("accuracy")

*//Tuning hyper parameter*

val paramGrid = new ParamGridBuilder().addGrid(classifier.regParam, Seq(0.1, 0.01)).addGrid(classifier.elasticNetParam, Seq(0.0,0.5,0.8,1.0)).addGrid(classifier.fitIntercept).build()

*// 80% of the data will be used for training and the remaining 20% for validation.*

val validator = new TrainValidationSplit()

.setEstimator(pipeline)

.setEvaluator(multiclassEval)

.setEstimatorParamMaps(paramGrid)

.setTrainRatio(0.8)

val validatorModel = validator.fit(trainData)

val paramsAndMetrics = validatorModel.validationMetrics.zip(validatorModel.getEstimatorParamMaps).sortBy(-\_.\_1)

paramsAndMetrics.foreach { case (metric, params) =>

println(metric)

println(params)

println()}

val bestModel = validatorModel.bestModel

println(bestModel.asInstanceOf[PipelineModel].stages.last.extractParamMap)

println(validatorModel.validationMetrics.max)

val cvAccuracy = multiclassEval.evaluate(bestModel.transform(cvData))

println(cvAccuracy)

val testAccuracy = multiclassEval.evaluate(bestModel.transform(testData))

println(testAccuracy)

val trainAccuracy = multiclassEval.evaluate(bestModel.transform(trainData))

println(trainAccuracy)

val stages = Array(assembler, indexer, classifier)

val pipelineModel = pipeline.fit(trainData)

val pipelinePredictionDf = pipelineModel.transform(testData)

pipelinePredictionDf.show(10)

*//Evaluate the model*

val evaluator = new MulticlassClassificationEvaluator().setLabelCol("severity").setPredictionCol("prediction")

val accuracy = evaluator.setMetricName("accuracy").evaluate(pipelinePredictionDf)

val f1 = evaluator.setMetricName("f1").evaluate(pipelinePredictionDf)

println(accuracy)

println(f1)

val weightedPrecision = evaluator.setMetricName("weightedPrecision").evaluate(pipelinePredictionDf)

println(weightedPrecision)

val weightedRecall =evaluator.setMetricName("weightedRecall").evaluate(pipelinePredictionDf)