lab4

August 20, 2020

[4]:
```
input_path = "/data/students/bigdata_internet/lab4/log_tcp_complete_classes.txt"
```

[5]:
```
input_df = spark.read.load(input_path,format="csv", header=True,␣
↪inferSchema=True,sep = ' ')
```

[6]:
```
#input_df.printSchema()
```

[7]:
```
def connectionsPerService(classvalue):
    instance = 1
    return int(instance)
```

[8]:
```
def getServicename(classandservie):
    classword,service = classandservie.split(':')
    return service
```

[9]:
```
class_df = input_df.select("class:207").withColumnRenamed("class:
↪207","classvalue")
```

[10]:
```
spark.udf.register("isService", lambda classvalue: getServicename(classvalue))
```

[10]:
```
<function __main__.<lambda>(classvalue)>
```

[11]:
```
service_df = class_df.selectExpr("isService(classvalue) as service")
```

[49]:
```
class_df.count()
```

[49]:
```
100000
```

[13]:
```
service_df.show(4)
```

```
+-------+
|service|
+-------+
| google|
| google|
| google|
| google|
+-------+
```

```
only showing top 4 rows
```

[14]:
```
spark.udf.register("isConnection", lambda service:␣
↪connectionsPerService(service),"integer")
```

[14]:
```
<function __main__.<lambda>(service)>
```

[15]:
```
classvalue_instance_df = service_df.selectExpr("service","isConnection(service)␣
↪as instance")
```

[16]:
```
classvalue_instance_df.show(4)
```

```
+-------+--------+
|service|instance|
+-------+--------+
| google|       1|
| google|       1|
| google|       1|
| google|       1|
+-------+--------+
only showing top 4 rows
```

[17]:
```
total_connections_per_service_df = classvalue_instance_df.groupBy("service").
↪sum("instance")\
.withColumnRenamed("sum(instance)","total instances")
```

[18]:
```
total_connections_per_service_df.show(4)
total_connections_per_service_df.count() # number of different services
```

```
+---------+---------------+
|  service|total instances|
+---------+---------------+
|instagram|          10000|
|  spotify|          10000|
|     bing|          10000|
|   amazon|          10000|
+---------+---------------+
only showing top 4 rows
```

[18]:
```
10
```

[19]:
```
# since Spark only uses numerical values for classification i need to not␣
↪consider the
# cathergorical values
```

[20]:
```
# 2.2
# dataframe with chosen features for classification
df_temp = input_df.select("class:207"\
                          ,"c_pkts_all:3"\
                          ,"s_pkts_all:17"\
                          ,"c_ack_cnt:5"\
                          ,"s_ack_cnt:19"\
                          ,"c_bytes_uniq:7"\
                          ,"s_bytes_uniq:21"\
                          ,"c_pkts_data:8"\
                          ,"s_pkts_data:22"\
                          ,"durat:31"\
                          ,"c_first:32"\
                          ,"s_first:33"\
                          ,"c_first_ack:36"\
                          ,"s_first_ack:37"\
                          ,"c_rtt_avg:45"\
                          ,"s_rtt_avg:52"\
                          ,"c_rtt_std:48"\
                          ,"s_rtt_std:55"\
                          ,"c_rtt_cnt:49"\
                          ,"s_rtt_cnt:56"\
                          ,"c_mss:70"\
                          ,"s_mss:93"\
                          ,"c_mss_max:71"\
                          ,"s_mss_max:94"\
                          ,"http_req_cnt:111"\
                          ,"http_res_cnt:112"\
                          ,"c_rtt_min:46"\
                          ,"s_rtt_min:53"\
                          ,"c_rtt_max:47"\
                          ,"s_rtt_max:54"\
                          ,"c_mss_min:72",\
                         "s_mss_min:95"\
                          ,"c_win_max:73"\
                          ,"s_win_max:96"\
                          ,"c_win_min:74"\
                          ,"s_win_min:97"\
                          ,"c_cwin_max:76"\
                          ,"s_cwin_max:99"\
                          ,"c_cwin_min:77"\
                          ,"s_cwin_min:100"\
                          ,"c_cwin_ini:78"\
                          ,"s_cwin_ini:101"\
                          ,"c_appdataT:123"\
                          ,"s_appdataT:124"\
                          ,"c_appdataB:125"\
```

```
                            ,"s_appdataB:126"\
                            ,"c_last:34"\
                            ,"s_last:35"\
                            ,"c_bytes_retx:11"\
                            ,"s_bytes_retx:25"
                            )\
.withColumnRenamed("class:207","classvalue")\
.withColumnRenamed("c_pkts_all:3","c_pkts_all")\
.withColumnRenamed("s_pkts_all:17","s_pkts_all")\
.withColumnRenamed("c_ack_cnt:5","c_ack_cnt")\
.withColumnRenamed("s_ack_cnt:19","s_ack_cnt")\
.withColumnRenamed("c_bytes_uniq:7","c_bytes_uniq")\
.withColumnRenamed("s_bytes_uniq:21","s_bytes_uniq")\
.withColumnRenamed("c_pkts_data:8","c_pkts_data")\
.withColumnRenamed("s_pkts_data:22","s_pkts_data")\
.withColumnRenamed("durat:31","durat")\
.withColumnRenamed("c_first:32","c_first")\
.withColumnRenamed("s_first:33","s_first")\
.withColumnRenamed("c_first_ack:36","c_first_ack")\
.withColumnRenamed("s_first_ack:37","s_first_ack")\
.withColumnRenamed("c_rtt_avg:45","c_rtt_avg")\
.withColumnRenamed("s_rtt_avg:52","s_rtt_avg")\
.withColumnRenamed("c_rtt_std:48","c_rtt_std")\
.withColumnRenamed("s_rtt_std:55","s_rtt_std")\
.withColumnRenamed("c_rtt_cnt:49","c_rtt_cnt")\
.withColumnRenamed("s_rtt_cnt:56","s_rtt_cnt")\
.withColumnRenamed("c_mss:70","c_mss")\
.withColumnRenamed("s_mss:93","s_mss")\
.withColumnRenamed("c_mss_max:71","c_mss_max")\
.withColumnRenamed("s_mss_max:94","s_mss_max")\
.withColumnRenamed("http_req_cnt:111","http_req_cnt")\
.withColumnRenamed("http_res_cnt:112","http_res_cnt")\
.withColumnRenamed("c_rtt_min:46","c_rtt_min")\
.withColumnRenamed("s_rtt_min:53","s_rtt_min")\
.withColumnRenamed("c_rtt_max:47","c_rtt_max")\
.withColumnRenamed("s_rtt_max:54","s_rtt_max")\
.withColumnRenamed("c_mss_min:72","c_mss_min")\
.withColumnRenamed("s_mss_min:95","s_mss_min")\
.withColumnRenamed("c_win_max:73","c_win_max")\
.withColumnRenamed("s_win_max:96","s_win_max")\
.withColumnRenamed("c_win_min:74","c_win_min")\
.withColumnRenamed("s_win_min:97","s_win_min")\
.withColumnRenamed("c_cwin_max:76","c_cwin_max")\
.withColumnRenamed("s_cwin_max:99","s_cwin_max")\
.withColumnRenamed("c_cwin_min:77","c_cwin_min")\
.withColumnRenamed("s_cwin_min:100","s_cwin_min")\
.withColumnRenamed("c_cwin_ini:78","c_cwin_ini")\
```

```
.withColumnRenamed("s_cwin_ini:101","s_cwin_ini")\
.withColumnRenamed("c_appdataT:123","c_appdataT")\
.withColumnRenamed("s_appdataT:124","s_appdataT")\
.withColumnRenamed("c_appdataB:125","c_appdataB")\
.withColumnRenamed("s_appdataB:126","s_appdataB")\
.withColumnRenamed("c_last:34","c_last")\
.withColumnRenamed("s_last:35","s_last")\
.withColumnRenamed("c_bytes_retx:11","c_bytes_retx")\
.withColumnRenamed("s_bytes_retx:25","s_bytes_retx")
```

[21]:
```
dataframe_for_classification__simple = df_temp.
 ↪selectExpr("isService(classvalue) as service"\
                                                    ,"c_pkts_all"\
                                                    ,"s_pkts_all"\
                                                    ,"c_ack_cnt"\
                                                    ,"s_ack_cnt"\
                                                    ,"c_bytes_uniq"\
                                                    ,"s_bytes_uniq"\
                                                    ,"c_pkts_data"\
                                                    ,"s_pkts_data"\
                                                    ,"durat"\
                                                    ,"c_first"\
                                                    ,"s_first"\
                                                    ,"c_first_ack"\
                                                    ,"s_first_ack"\
                                                    ,"c_rtt_avg"\
                                                    ,"s_rtt_avg"\
                                                    ,"c_rtt_std"\
                                                    ,"s_rtt_std"\
                                                    ,"c_rtt_cnt"\
                                                    ,"s_rtt_cnt"\
                                                    ,"c_mss"\
                                                    ,"s_mss"\
                                                    ,"c_mss_max"\
                                                    ,"s_mss_max"\
                                                    ,"http_req_cnt"\
                                                    ,"http_res_cnt"\
                                                    ,"c_rtt_min"\
                                                    ,"s_rtt_min"\
                                                    ,"c_rtt_max"\
                                                    ,"s_rtt_max"\
                                                    ,"c_mss_min"\
                                                    ,"s_mss_min"\
                                                    ,"c_win_max"\
                                                    ,"s_win_max"\
                                                    ,"c_win_min"\
                                                    ,"s_win_min"\
```

```
                                                    ,"c_cwin_max"\
                                                    ,"s_cwin_max"\
                                                    ,"c_cwin_min"\
                                                    ,"s_cwin_min"\
                                                    ,"c_cwin_ini"\
                                                    ,"s_cwin_ini"\
                                                    ,"c_appdataT"\
                                                    ,"s_appdataT"\
                                                    ,"c_appdataB"\
                                                    ,"s_appdataB"\
                                                    ,"c_last"\
                                                    ,"s_last"\
                                                    ,"c_bytes_retx"\
                                                    ,"s_bytes_retx"
                                                    )
```

[22]:
```
#2.1 Read and split the data after feature selection
training_df,test_df = dataframe_for_classification__simple.randomSplit([0.75,0.
↪25])
```

[23]:
```
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
```

[24]:
```
indexer = StringIndexer(inputCol = "service", outputCol = "service index",␣
↪handleInvalid = "keep")
```

[25]:
```
va = VectorAssembler(inputCols = ["c_pkts_all","s_pkts_all"\
                                  ,"c_ack_cnt","s_ack_cnt"\
                                  ,"c_bytes_uniq","s_bytes_uniq"\
                                  ,"c_pkts_data","s_pkts_data"\
                                  ,"durat","c_first","s_first","c_first_ack"\
                                  ,"s_first_ack","c_rtt_avg"\
                                  ,"s_rtt_avg","c_rtt_std"\
                                  ,"s_rtt_std","c_rtt_cnt"\
                                  ,"s_rtt_cnt","c_mss"\
                                  ,"s_mss","c_mss_max"\
                                  ,"s_mss_max","http_req_cnt"\
                                  ,"http_res_cnt","c_rtt_min"\
                                  ,"s_rtt_min","c_rtt_max"\
                                  ,"s_rtt_max","c_mss_min"\
                                  ,"s_mss_min","c_win_max"\
                                  ,"s_win_max","c_win_min"\
                                  ,"s_win_min","c_cwin_max"\
                                  ,"s_cwin_max","c_cwin_min"\
                                  ,"s_cwin_min","c_cwin_ini"\
                                  ,"s_cwin_ini","c_appdataT"\
                                  ,"s_appdataT","c_appdataB"\
```

```
                                    ,"s_appdataB","c_last"\
                                    ,"s_last","c_bytes_retx"\
                                    ,"s_bytes_retx"]\
                        , outputCol = "features")
```

[26]:
```
from pyspark.ml.feature import StandardScaler
scaler = StandardScaler(inputCol = "features", outputCol = "scaledFeatures",␣
 ↪withStd = True, withMean = True)
```

[27]:
```
featuresColumnToUse = "scaledFeatures"
```

[28]:
```
#2.3 Training two different models
# train a decision tree
from pyspark.ml.classification import DecisionTreeClassifier
```

[29]:
```
decision_tree = DecisionTreeClassifier(labelCol = "service index" , featuresCol␣
 ↪= featuresColumnToUse)
```

[30]:
```
# Creating the pipeline for the decision tree
from pyspark.ml import Pipeline
pipelineTree = Pipeline(stages=[indexer,va, scaler,decision_tree])
```

[31]:
```
# training a random forest classifier
from pyspark.ml.classification import RandomForestClassifier
```

[32]:
```
random_forest = RandomForestClassifier(labelCol = "service index", featuresCol␣
 ↪= featuresColumnToUse, numTrees = 20,maxDepth = 12)
```

[33]:
```
# Creating the Pipeline for the random forest
pipelineRandomForest = Pipeline(stages=[indexer,va,scaler,random_forest])
```

[34]:
```
#2.4 Evaluators
```

[35]:
```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

[36]:
```
# for the decision tree
dt_accuracy_training_eval = MulticlassClassificationEvaluator(labelCol =␣
 ↪"service index", predictionCol = "prediction", metricName = "accuracy")
dt_f1_training_eval = MulticlassClassificationEvaluator(labelCol = "service␣
 ↪index", predictionCol = "prediction", metricName = "f1")
```

[37]:
```
# for the random forest
rf_accuracy_training_eval = MulticlassClassificationEvaluator(labelCol =␣
 ↪"service index", predictionCol = "prediction", metricName = "accuracy")
rf_f1_training_eval = MulticlassClassificationEvaluator(labelCol = "service␣
 ↪index", predictionCol = "prediction", metricName = "f1")
```

[38]:
```
# 2.5 Tuning the parameters
```

[39]:
```
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
```

[51]:
```
paramGridTree = ParamGridBuilder()\
.addGrid(decision_tree.maxDepth, [2,10,12])\
.addGrid(decision_tree.impurity, ["Gini","Entropy"])\
.build()


paramGridRandomForest = ParamGridBuilder()\
.addGrid(random_forest.maxDepth, [2,10,12])\
.addGrid(random_forest.impurity, ["Gini","Entropy"])\
.addGrid(random_forest.numTrees, [15,20,25])\
.build()
```

[52]:
```
cv_dt = CrossValidator(estimator=pipelineTree\
                       ,evaluator=dt_accuracy_training_eval\
                       ,estimatorParamMaps = paramGridTree\
                       ,numFolds=3)

cv_rf = CrossValidator(estimator=pipelineRandomForest\
                       ,evaluator=rf_accuracy_training_eval\
                       ,estimatorParamMaps = paramGridRandomForest\
                       , numFolds=3)
```

[53]:
```
cvModelDecisionTree = cv_dt.fit(training_df)
cvModelRandomForest = cv_rf.fit(training_df)
```

[54]:
```
finalDFDecisionTree=cvModelDecisionTree.transform(training_df)
finalDFRandomForest=cvModelRandomForest.transform(training_df)
```

[55]:
```
import numpy
#cvModelDecisionTree.getEstimatorParamMaps()
```

[56]:
```
# index of the model
numpy.argmax(cvModelDecisionTree.avgMetrics)
```

[56]:
```
5
```

[57]:
```
# accuracy results for all parameters tried
cvModelDecisionTree.avgMetrics
```

[57]:
```
[0.3721507187157899,
 0.372957482420219,
 0.9658794503262684,
```

```
 0.9731691473536938,
 0.9733604175564261,
 0.9778102512864497]
```

[58]:
```
final_dt_testDf = cvModelDecisionTree.transform(test_df)
dt_accuracy_test_eval = MulticlassClassificationEvaluator(labelCol = "service␣
 ↪index", predictionCol = "prediction", metricName = "accuracy")
print("Accuracy of Decision Tree on test set: ",dt_accuracy_test_eval.
 ↪evaluate(final_dt_testDf))
dt_f1_test_eval = MulticlassClassificationEvaluator(labelCol = "service index",␣
 ↪predictionCol = "prediction", metricName = "f1")
print("F1 of Decision Tree on test set: ",dt_f1_test_eval.
 ↪evaluate(final_dt_testDf))
```

```
Accuracy of Decision Tree on test set:  0.9796040313906704
F1 of Decision Tree on test set:  0.979592836831596
```

[59]:
```
cvModelRandomForest.avgMetrics
```

[59]:
```
[0.7086375226799105,
 0.7363698741019857,
 0.7243171030173302,
 0.6894272249373093,
 0.7187216840211644,
 0.7210604181769504,
 0.9701087407272235,
 0.9694822628399986,
 0.9704025690269837,
 0.9699258132118802,
 0.9700215947739583,
 0.969760827531071,
 0.9794842057346562,
 0.9791937325493275,
 0.9792203795436007,
 0.9790713526100359,
 0.9798435980856836,
 0.9793794343888664]
```

[60]:
```
final_rf_testDf = cvModelRandomForest.transform(test_df)
rf_accuracy_test_eval = MulticlassClassificationEvaluator(labelCol = "service␣
 ↪index", predictionCol = "prediction", metricName = "accuracy")
print("Accuracy of Random Forest on test set: ",rf_accuracy_test_eval.
 ↪evaluate(final_rf_testDf))
rf_f1_test_eval = MulticlassClassificationEvaluator(labelCol = "service index",␣
 ↪predictionCol = "prediction", metricName = "f1")
print("F1 of Random Forest on test set: ",rf_f1_test_eval.
 ↪evaluate(final_rf_testDf))
```

```
Accuracy of Random Forest on test set:  0.9815559893239851
F1 of Random Forest on test set:  0.9815464651455406
```