In [5]:
```
from pyspark.ml.feature import VectorAssembler
spark = SparkSession.builder.getOrCreate()
from pyspark.ml.feature import StandardScaler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import OneHotEncoderEstimator
from pyspark.ml import Pipeline
import time
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.tuning import ParamGridBuilder
from pyspark.ml.tuning import CrossValidator
import numpy
```

In [110…
```
f = spark.read.load('/data/students/bigdata_internet/lab4/log_tcp_complete_classes.
                    , sep =' ',
                     format = 'csv', header = True, inferSchema=True )
```

1.1. How many columns/features does the file have?

In [7]:
```
num_col=len(f.columns)
print('number of columns',num_col)
```

```
number of columns 207
```

1.2. How many TCP connections are there in the log?

In [8]:
```
num_log=f.count()
print('number of TCP logs: ', num_log)
```

```
23/02/08 13:03:54 WARN util.Utils: Truncated the string representation of a plan s
ince it was too large. This behavior can be adjusted by setting 'spark.debug.maxTo
StringFields' in SparkEnv.conf.
number of TCP logs:  100000
```

In [9]:
```
classes = f.select('class:207')
dist_classes= classes.distinct()
```

2.0.1 How many classes are there in the file?

In [10]:
```
num_classes = dist_classes.count()
print(num_classes)
```

```
[Stage 5:===========================================>            (154 + 2) / 200]
10
```

2.0.2 Can you list all of them? 2.0.3 How many connections per web service are present in the DataFrame?

In [11]:
```
connection = f.groupBy('class:207').count()
connection.show()
```

```
+--------------+-----+
|     class:207|count|
+--------------+-----+
|  class:google|10000|
|  class:amazon|10000|
|class:instagram|10000|
| class:facebook|10000|
|  class:netflix|10000|
|     class:ebay|10000|
|  class:spotify|10000|
| class:linkedin|10000|
|  class:youtube|10000|
|     class:bing|10000|
+--------------+-----+
```

In [12]:

```
new_df = f.select('c_pkts_all:3','s_pkts_all:17','c_bytes_all:9',\
                  's_bytes_all:23','durat:31','c_rtt_std:48',\
                  's_rtt_std:55','c_first:32','s_first:33','class:207')
new_df.show(1)
```

```
+------------+-------------+-------------+--------------+--------+------------+---
---------+----------+----------+------------+
|c_pkts_all:3|s_pkts_all:17|c_bytes_all:9|s_bytes_all:23|durat:31|c_rtt_std:48|s_r
tt_std:55|c_first:32|s_first:33|   class:207|
+------------+-------------+-------------+--------------+--------+------------+---
---------+----------+----------+------------+
|          21|           28|          167|         35450|1005.178|         0.0|
0.0|   613.296|   695.518|class:google|
+------------+-------------+-------------+--------------+--------+------------+---
---------+----------+----------+------------+
only showing top 1 row
```

2.1 Select features, 2.2 Read and split the data

2.1.1. Does it make sense to use the IP addresses + ports (`#31#c_ip:1` , `c_port:2` , `s_ip:15` , `s_port:16`) as features?
No,it does not make sense. A computer works with numbers not strings as ips,ports,etc. When features are categorical, the distance metric is not meaningful, for example, the distance between two ports does not have a mathematical meaning because those are not real measurements. Therefore, when using categorical features, it's common to do some kind of processing/transformations.

2.1.2. Would it be fair to use the Fully Qualified Domain Name (FQDN, `fqdn:127` , for instance `www.google.com`) for the classification? No, you should process the string to get only the part of our interest.

In [13]:

```
train,test=new_df.randomSplit([0.7,0.3],100)
print('Number of train elements:',train.count())
print('Number of test elements:',test.count())
print('Total number:',train.count()+test.count())
```

```
Number of train elements: 70047
```

```
Number of test elements: 29953
```

```
[Stage 24:=============================>                           (1 + 1) / 2]
Total number: 100000
```

2.3 Pre-process the dataset

In [14]:
```
feat_cols = ['c_pkts_all:3','s_pkts_all:17','c_bytes_all:9',\
             's_bytes_all:23','durat:31','c_rtt_std:48',\
                  's_rtt_std:55','c_first:32','s_first:33']
```

In [84]:
```
vector_assembler = VectorAssembler(inputCols = feat_cols, outputCol = 'features')
transformedDF = vector_assembler.transform(train)
```

In [85]:
```
scaler = StandardScaler(inputCol='features',\
                        outputCol="scaledFeatures",\
                        withStd=True, withMean=True)
scalerModel = scaler.fit(transformedDF)
scaledDF = scalerModel.transform(transformedDF)
```

In [83]:
```
indexer = StringIndexer(inputCol="class:207",outputCol="label")
indexerModel = indexer.fit(scaledDF)
indexedDF=indexerModel.transform(scaledDF)
```

In [86]:
```
encoder = OneHotEncoderEstimator(inputCols=["label"], \
                                 outputCols=["labelOneHot"])
model = encoder.fit(indexedDF)
encodedDF = model.transform(indexedDF)
```

In [87]:
```
pipeline=Pipeline(stages=[vector_assembler,scaler,indexer,encoder])
model= pipeline.fit(train)
processed = model.transform(train)
training = processed.select('scaledFeatures', 'label')
```

RANDOM FOREST

2.4 Train at least two different models, 2.5 Evaluate the performance of the models

In [37]:
```
start_time = time.time()
rf = RandomForestClassifier(labelCol="label", featuresCol='scaledFeatures',\
                            maxDepth = 28, numTrees = 20,\
                            impurity='entropy', maxBins=100 )
rfModel1  =rf.fit(training)
finalDF1=rfModel1.transform(training)
end_time = time.time()
print("time execution:",(end_time - start_time))
```

```
time execution: 177.33818316459656
```

2.4.1 How much does it take to train the model (time in seconds), for the different algorithm and parameters? time execution: 177.33818316459656 s

In [38]:
```
predictions = finalDF1
ev1 = MulticlassClassificationEvaluator(labelCol='label',\
                                        predictionCol="prediction",\
                                        metricName='accuracy')
accuracy_1=ev1.evaluate(predictions)
print('The accuracy is: ',accuracy_1)
```

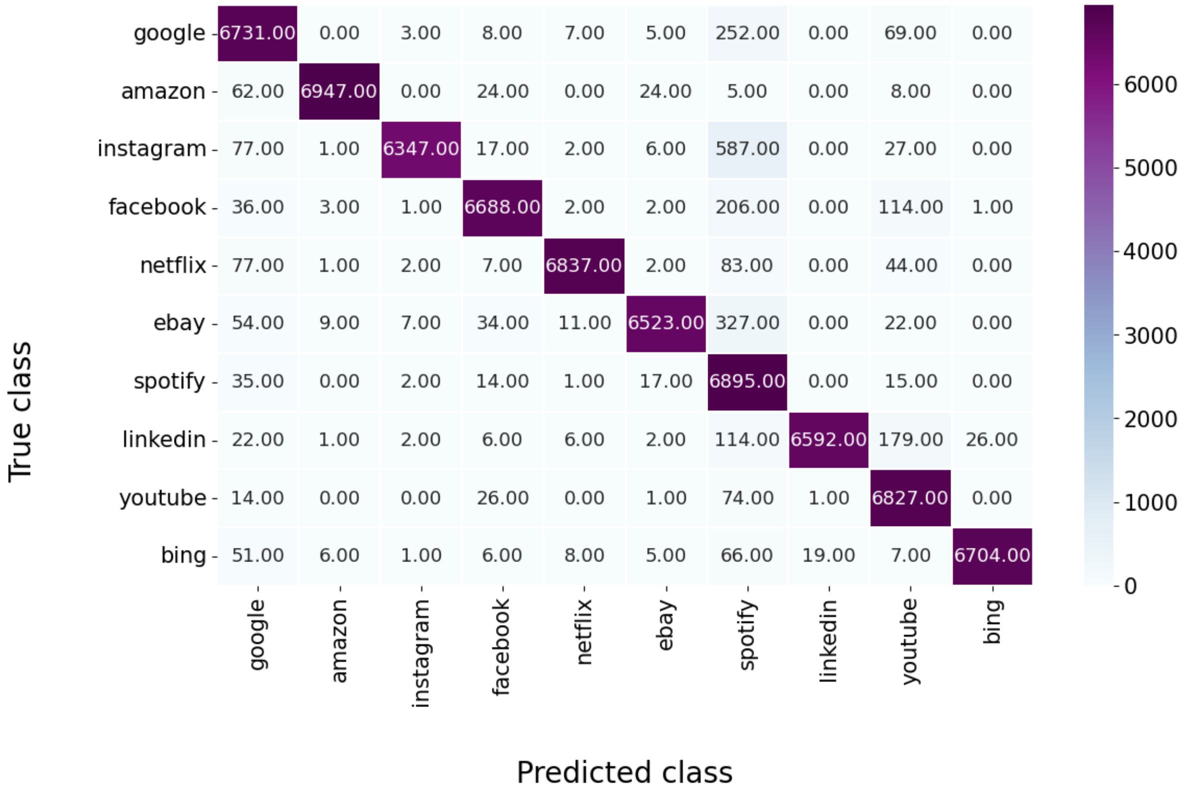
```
[Stage 588:============================>                          (1 + 1) / 2]
The accuracy is:  0.9577997630162605
```

DECISION TREE

In [39]:
```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
from pyspark.sql.types import FloatType
from pyspark.mllib.evaluation import MulticlassMetrics
from pyspark.sql.functions import col
predictions = finalDF1

preds_and_labels = predictions.select(['prediction','label'])\
.withColumn('label', col('label').cast(FloatType())).orderBy('prediction')
predictionAndLabels = preds_and_labels.select(['prediction','label']).rdd.map(tuple
metrics = MulticlassMetrics(predictionAndLabels)

cm = metrics.confusionMatrix().toArray()
#REPLACE NAME OF YOUR DATAFRAME HERE
classes =  f.select("class:207").distinct().rdd.map(lambda r:r[0]).collect()
classes = [el.replace("class:", "") for el in classes]
fig, ax = plt.subplots(figsize =(12, 8))
fontsize = 15
ax = sns.heatmap(cm, xticklabels=classes, yticklabels=classes,linewidth = 0.2,\
                 cmap="BuPu",\
                 annot = True, fmt = ".2f",annot_kws={"fontsize":fontsize-2})
cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=fontsize)
ax.figure.axes[-1].yaxis.label.set_size(fontsize+5)
ax.figure.axes[-1].yaxis.set_label_coords(3,.5)
ax.set_xticklabels(classes, fontsize=fontsize, rotation = 90)
ax.set_yticklabels(classes, fontsize=fontsize, rotation = 0)
ax.set_ylabel("True class", fontsize = fontsize + 5)
ax.set_xlabel("Predicted class", fontsize = fontsize + 5)
ax.yaxis.set_label_coords(-.22,.3)
ax.xaxis.set_label_coords(.5, -.3)
plt.tight_layout()
plt.show()
```



In [40]:
```
start_time = time.time()
dt = DecisionTreeClassifier(labelCol="label",\
                            featuresCol="scaledFeatures", impurity='entropy',\
                            maxDepth=28, maxBins=100).fit(training)
finalDF2=dt.transform(training)
end_time = time.time()
print("time execution:",(end_time - start_time))
```

```
time execution: 34.41801929473877
```

2.4.1 How much does it take to train the model (time in seconds), for the different algorithm and parameters? time execution: 34.41801929473877 s

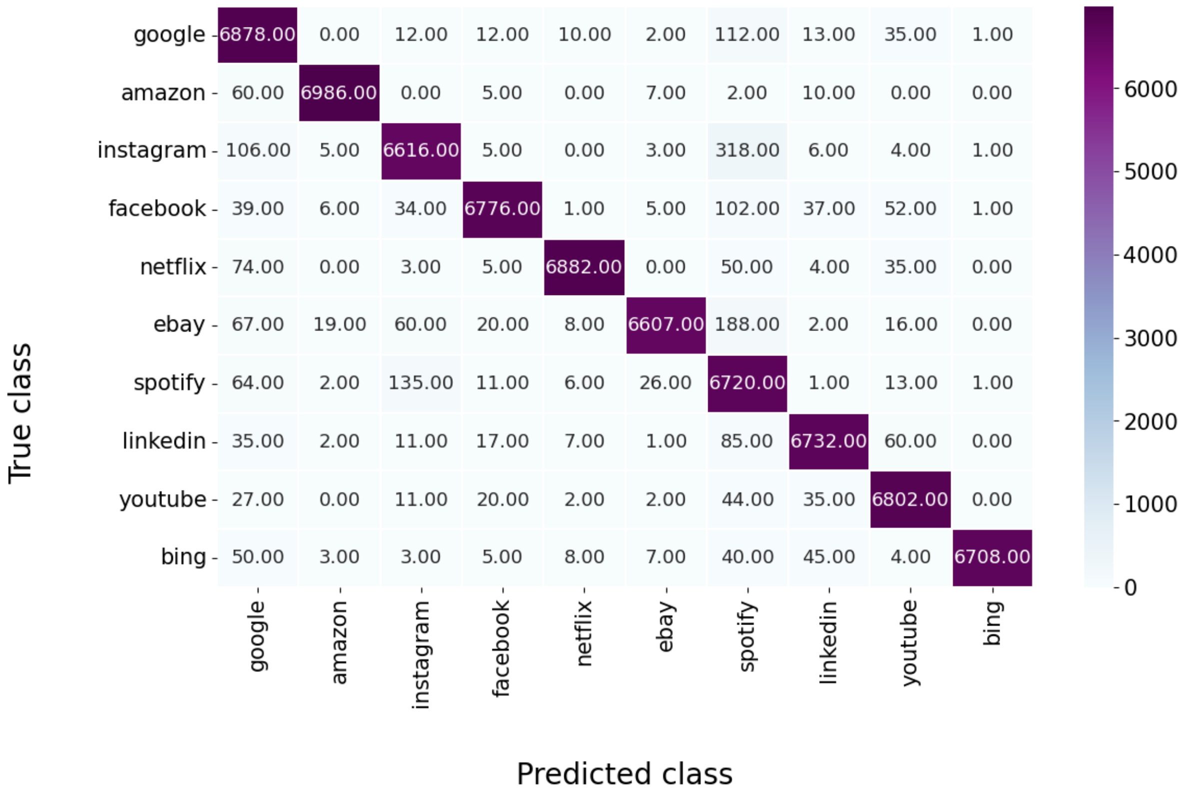
In [44]:
```
ev2 = MulticlassClassificationEvaluator(labelCol='label',metricName='accuracy')
accuracy_2=ev2.evaluate(finalDF2)
print(accuracy_2)
```

```
[Stage 665:============================>                              (1 + 1) / 2]
0.9665938584093537
```

In [45]:
```
preds_and_labels = finalDF2.select(['prediction','label'])\
.withColumn('label', col('label').cast(FloatType())).orderBy('prediction')
predictionAndLabels = preds_and_labels.select(['prediction','label']).rdd.map(tuple
metrics = MulticlassMetrics(predictionAndLabels)

cm = metrics.confusionMatrix().toArray()
#REPLACE NAME OF YOUR DATAFRAME HERE
classes =  f.select("class:207").distinct().rdd.map(lambda r:r[0]).collect()
classes = [el.replace("class:", "") for el in classes]
fig, ax = plt.subplots(figsize =(12, 8))
fontsize = 15
ax = sns.heatmap(cm, xticklabels=classes, yticklabels=classes,linewidth = 0.2,\
                 cmap="BuPu",\
                 annot = True, fmt = ".2f",annot_kws={"fontsize":fontsize-2})
```

```
cbar = ax.collections[0].colorbar
cbar.ax.tick_params(labelsize=fontsize)
ax.figure.axes[-1].yaxis.label.set_size(fontsize+5)
ax.figure.axes[-1].yaxis.set_label_coords(3,.5)
ax.set_xticklabels(classes, fontsize=fontsize, rotation = 90)
ax.set_yticklabels(classes, fontsize=fontsize, rotation = 0)
ax.set_ylabel("True class", fontsize = fontsize + 5)
ax.set_xlabel("Predicted class", fontsize = fontsize + 5)
ax.yaxis.set_label_coords(-.22,.3)
ax.xaxis.set_label_coords(.5, -.3)
plt.tight_layout()
plt.show()
```



In [36]:
```
processed.select("label","class:207").distinct().show()
training.groupBy("label").count().show()
```

```
+-----+---------------+
|label|      class:207|
+-----+---------------+
|  0.0|  class:netflix|
|  3.0|   class:google|
|  8.0|  class:youtube|
|  9.0|class:instagram|
|  5.0| class:linkedin|
|  4.0|  class:spotify|
|  7.0| class:facebook|
|  6.0|     class:ebay|
|  1.0|     class:bing|
|  2.0|   class:amazon|
+-----+---------------+

+-----+-----+
|label|count|
+-----+-----+
|  8.0| 6960|
|  0.0| 7063|
|  7.0| 7019|
|  1.0| 7038|
|  4.0| 7010|
|  3.0| 7026|
|  2.0| 6994|
|  6.0| 7016|
|  5.0| 7023|
|  9.0| 7003|
+-----+-----+
```

2.4.1 How much does it take to train the model (time in seconds), for the different algorithm and parameters?
When working with simialr parameters, it is possible to see that the decision tree takes less time to train the model.

2.5.1 Comment your results: which classes are easier to classify? Which get confused the most?
In order to get the accuracy for the single classes we can take the ratio between the values on the diagonal and the total umber of samples (in the trainin set) for the corresponding class. I calculated all the accuracies and reported the ones of interest, related to the classes that are more easily confused and that are easier to classify.
RANDOM FOREST
From this analysis, we can say that the most confused classes are Instagram (90.6%), Ebay (93.1%) and Bing (95.3%) . The covariance matrix shows that Instagram and Ebay are mostly confused with Spotify while Bing is sometimes confused with Google and Spotify. The easiset classes to classify in this case are Amazon with an accuracy of 99.3%, Linkedin with 98.9% and Youtube with 98.1%.
DECISION TREE
From this analysis, we can say that the most confused classes are : Ebay (94.2%), Instagram (94.4%) and, Bing (95.3%). In this case Instagram and Ebay are mostly confused with Spotify and Bing is sometimes confused with with Spotify and Linkedin with Google. The easiset classes to classify with the decision tree are are Amazon with an accuracy of 98.9%, Facebook with 97.4% and google with 97.6%.

2.5.2 Which classifier performs better? Why do you think it is the case? In this case, the decision tree works better because, using similar parameters, it is faster (34 s versus 177 s) and, overall, the total accuracy is higher (96.6% versus 95,7%).

2.6 Tune the parameters of the models

In [89]:
```
#RandomForest
rf_DF1 = RandomForestClassifier(labelCol="label",\
                                featuresCol="scaledFeatures")
param_grid1 = ParamGridBuilder().addGrid(rf_DF1.numTrees, [20,28,32]).\
addGrid(rf_DF1.maxDepth, [10, 15, 20])\
.addGrid(rf_DF1.impurity,["Gini","Entropy"]).\
addGrid(rf_DF1.maxBins,[32,100]).build()
evaluator1= MulticlassClassificationEvaluator(labelCol="label",\
                                              predictionCol="prediction",\
                                              metricName='accuracy')
rf_cv=CrossValidator(estimator=rf_DF1,evaluator=evaluator1,\
 estimatorParamMaps=param_grid1, numFolds=3)
rf_cvModel1=rf_cv.fit(training)
finalDF_rf1=rf_cvModel1.transform(training)
```

In [90]:
```
print(rf_cvModel1.avgMetrics)
```

```
[0.678231899533231, 0.6827360917189406, 0.691276175666157, 0.6980872957013098, 0.7
637746146171019, 0.7700278187968235, 0.7716514717734597, 0.7790923195909516, 0.782
1041240422955, 0.7888416214236611, 0.78347176551589, 0.7927126267443605, 0.6831995
118395364, 0.6846734261534884, 0.6959932333107404, 0.7011614822381951, 0.768099493
0568876, 0.7715416730214235, 0.7738236312407352, 0.7826750261993238, 0.78765778729
79236, 0.7937397624730198, 0.7866158426087675, 0.7962640213517505, 0.6836429993590
224, 0.6856150939717083, 0.6960223413546244, 0.7001207632183191, 0.768384132774336
8, 0.7758238857597703, 0.7768925406798848, 0.7828177279844313, 0.7876135277928324,
0.7946957601408136, 0.7901271338298295, 0.7974075865624699]
```

In [91]:
```
rf_cvModel1.bestModel
```

Out[91]:
```
RandomForestClassificationModel (uid=RandomForestClassifier_4388ca2d061d) with 32
trees
```

In [92]:
```
rf_cvModel1.getEstimatorParamMaps()[numpy.argmax(rf_cvModel1.avgMetrics)]
```

Out[92]:
```
{Param(parent='RandomForestClassifier_4388ca2d061d', name='numTrees', doc='Number
of trees to train (>= 1).'): 32,
 Param(parent='RandomForestClassifier_4388ca2d061d', name='maxDepth', doc='Maximum
depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 interna
l node + 2 leaf nodes.'): 20,
 Param(parent='RandomForestClassifier_4388ca2d061d', name='impurity', doc='Criteri
on used for information gain calculation (case-insensitive). Supported options: en
tropy, gini'): 'Entropy',
 Param(parent='RandomForestClassifier_4388ca2d061d', name='maxBins', doc='Max numb
er of bins for discretizing continuous features.  Must be >=2 and >= number of cat
egories for any categorical feature.'): 100}
```

In [93]:
```
print("Accuracy on training is ", evaluator1.evaluate(finalDF_rf1))
```

```
[Stage 22795:==========================>                          (1 + 1) / 2]
Accuracy on training is  0.9566291204476994
```

In [94]:
```
# DecisionTree
evaluator2= MulticlassClassificationEvaluator(labelCol="label",\
                                              predictionCol="prediction",\
                                              metricName='accuracy')
dt2 = DecisionTreeClassifier(labelCol="label",featuresCol="scaledFeatures")
paramGrid2 = ParamGridBuilder().addGrid(dt2.maxDepth, [20,28,30]).\
                            addGrid(dt2.impurity, ["Gini","Entropy"]).\
                            addGrid(rf_DF1.maxBins,[32,100]).build()
cv2=CrossValidator(estimator=dt2,\
                   evaluator=evaluator2,estimatorParamMaps=paramGrid2, numFolds=3)
cvModel2=cv2.fit(training)
finalDF2=cvModel2.transform(training)
cvModel2.getEstimatorParamMaps()[numpy.argmax(cvModel2.avgMetrics)]
```

Out[94]:
```
{Param(parent='DecisionTreeClassifier_463489e1f7ae', name='maxDepth', doc='Maximum
depth of the tree. (>= 0) E.g., depth 0 means 1 leaf node; depth 1 means 1 interna
l node + 2 leaf nodes.'): 20,
 Param(parent='DecisionTreeClassifier_463489e1f7ae', name='impurity', doc='Criteri
on used for information gain calculation (case-insensitive). Supported options: en
tropy, gini'): 'Entropy',
 Param(parent='RandomForestClassifier_4388ca2d061d', name='maxBins', doc='Max numb
er of bins for discretizing continuous features.  Must be >=2 and >= number of cat
egories for any categorical feature.'): 32}
```

In [95]:
```
print(cvModel2.bestModel)
```

```
DecisionTreeClassificationModel (uid=DecisionTreeClassifier_463489e1f7ae) of depth
20 with 23455 nodes
```

In [96]:
```
print(cvModel2.avgMetrics)
```

```
[0.7164199488818832, 0.7164199488818832, 0.7217022942421363, 0.7217022942421363,
0.7165775172837046, 0.7165775172837046, 0.7212155604021462, 0.7212155604021462, 0.
7163062361328075, 0.7163062361328075, 0.7212298622570968, 0.7212298622570968]
```

In [97]:
```
print("Accuracy on training is ", evaluator2.evaluate(finalDF2))
```

```
[Stage 24991:===========================>                          (1 + 1) / 2]
Accuracy on training is  0.9478064727968364
```

2.6.1 Report the accuracy results for all the parameters you tried. What can you conclude? The models returned with bestModel are the one associated to the best parameter setting. The parameters settings can be seen from the result of the avgMetrics obtained with the cross validation. Also in this case the accuracy value obtained through Decision Tree is slightly worse than the one of Random Forest but they are still very similar. The performance of the random forest probably improved with respect to the previous analysis because we are now using more trees (32 versus 28). The performance metrics for all the parameters I tried are given as the output of print('---'.avgMetrics)

2.7 Return the best possible model and estimate its performance on new data

In [108…
```
#random forest best model
test_data=model.transform(test)
rf_classifier = RandomForestClassifier(labelCol="label",\
 featuresCol="scaledFeatures",numTrees=32,\
```

```
 impurity='entropy', maxDepth=20,maxBins=100).fit(training)
rf_predictions_fin=rf_classifier.transform(test_data)
```

In [105…

```
accuracy_f1=MulticlassClassificationEvaluator(labelCol='label',\
 metricName='accuracy').evaluate(rf_predictions_fin)
print("Global accuracy Random Forest:",accuracy_f1)
```

```
[Stage 25395:===========================>                          (1 + 1) / 2]
Global accuracy Random Forest: 0.8067973157947451
```

In [106…

```
#Decision tree best model
dt_classifier=DecisionTreeClassifier(labelCol="label",\
 featuresCol="scaledFeatures",impurity='entropy',\
 maxDepth=20,maxBins=32).fit(training)
dt_predictions_fin=dt_classifier.transform(test_data)
```

In [107…

```
accuracy_f2=MulticlassClassificationEvaluator(labelCol='label',\
 metricName='accuracy')\
 .evaluate(dt_predictions_fin)
print("Global accuracy Decision Tree:",accuracy_f2)
```

```
[Stage 25443:>                                                    (0 + 2) / 2]
Global accuracy Decision Tree: 0.7424965779721564
```

2.7.1 Report the expected results performance and comment on the results obtained. The best models evaluated in the previous point have been used here to evaluate the performance on the test. As we can see the the performance of the Random forest on the test is better. In general, random forest is a better if we have a large amount of data or a complex problem. Instead for simpler problems and smaller datasets decision tree might be good a option because it is easier to interpret and faster to train.

In [ ]: