Part2

January 3, 2023

Initiate Spark

[1]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Part2').getOrCreate()
```

Import Required Modules

[2]:
```
from pyspark.sql.functions import *
import warnings
import time
```

Import Data

Read Master csv

[3]:
```
masterdf = spark.read.csv('master5.csv', inferSchema=True, header=True) #change␣
↪file!!!! Waiting for Alireza
masterdf.limit(5).toPandas()
```

[3]:

	Id	Class	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	\
0	kg1ZDvKAN9CyWamsRIxi.png	2	121	124	129	115	111	115	115	115	
1	avEsqegSVNdCUIAX53cY.png	6	50	53	65	65	53	50	44	48	
2	1TieqSEpxIa6BmcGFsvM.png	1	93	99	86	98	123	105	102	113	
3	cNVK7zn0aZ2o8yvDIpYm.png	1	103	108	114	106	113	118	117	124	
4	bRK4QxagusfH0SyZvwrp.png	1	27	34	32	36	42	33	39	37	

	…	_c1016	_c1017	_c1018	_c1019	_c1020	_c1021	_c1022	_c1023	\
0	…	126	128	126	128	128	129	129	128	
1	…	121	118	108	117	123	116	126	115	
2	…	60	66	55	67	61	54	64	67	
3	…	59	63	62	62	63	63	64	73	
4	…	52	57	62	52	46	51	53	59	

	_c1024	_c1025
0	129	127
1	105	114
2	62	60
3	62	69
4	55	53

```
[5 rows x 1026 columns]
```

Remove all-null rows and replace null values with 0

[4]:
```
masterdf = masterdf.na.fill(0)
```

[5]:
```
masterdf.limit(5).toPandas()
```

[5]:

	Id	Class	_c2	_c3	_c4	_c5	_c6	_c7	_c8	_c9	\
0	kg1ZDvKAN9CyWamsRIxi.png	2	121	124	129	115	111	115	115	115	
1	avEsqegSVNdCUIAX53cY.png	6	50	53	65	65	53	50	44	48	
2	1TieqSEpxIa6BmcGFsvM.png	1	93	99	86	98	123	105	102	113	
3	cNVK7zn0aZ2o8yvDIpYm.png	1	103	108	114	106	113	118	117	124	
4	bRK4QxagusfH0SyZvwrp.png	1	27	34	32	36	42	33	39	37	

	…	_c1016	_c1017	_c1018	_c1019	_c1020	_c1021	_c1022	_c1023	\
0	…	126	128	126	128	128	129	129	128	
1	…	121	118	108	117	123	116	126	115	
2	…	60	66	55	67	61	54	64	67	
3	…	59	63	62	62	63	63	64	73	
4	…	52	57	62	52	46	51	53	59	

	_c1024	_c1025
0	129	127
1	105	114
2	62	60
3	62	69
4	55	53

```
[5 rows x 1026 columns]
```

Required Modules

[6]:
```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import MinMaxScaler
from pyspark.ml.classification import *
from pyspark.ml.evaluation import *
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

from pyspark.sql.types import *
from pyspark.sql.functions import *

import time
```

Preprocessing the Data Frame

[7]:
```
dependent_var = 'Class'


renamed = masterdf.withColumn("label_str", masterdf[dependent_var].
 ↪cast(StringType())) #Rename and change to string type
indexer = StringIndexer(inputCol="label_str", outputCol="label") #Pyspark is␣
 ↪expecting the this naming convention
indexed = indexer.fit(renamed).transform(renamed)


features_list = masterdf.columns[2:] #first col id, second col class, the third␣
 ↪one and more are features
assembler = VectorAssembler(inputCols=features_list, outputCol='features')
final_data = assembler.transform(indexed).select('features','label')


seed = 40 #to get similar results
train_val = 0.7
test_val = 0.3
train, test = final_data.randomSplit([train_val,test_val],seed=seed)

# Set up our classification and evaluation objects
Bin_evaluator = BinaryClassificationEvaluator(rawPredictionCol='prediction')␣
 ↪#labelCol='label'
MC_evaluator = MulticlassClassificationEvaluator(metricName="accuracy") #␣
 ↪redictionCol="prediction",

timetook = []
acc = []
```

Random Forest (Depth = 3)

[8]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [3])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)
```

```
# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 3 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 3 : 71.28060263653484
--- 59.366854429244995 seconds ---
------------------
```

Random Forest (Depth = 4)

[9]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [4])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)
```

```
accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 4 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 4 : 78.78217200251099
--- 52.116631507873535 seconds ---
------------------
```

Random Forest (Depth = 5)

[10]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [5])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 5 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
```

```
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 5 : 82.04645323289391
--- 53.21452593803406 seconds ---
------------------
```

Random Forest (Depth = 6)

[11]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [6])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 6 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 6 : 84.77715003138732
--- 56.8309063911438 seconds ---
------------------
```

Random Forest (Depth = 7)

[12]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [7])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 7 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 7 : 88.38669177652228
--- 62.201735496520996 seconds ---
------------------
```

Random Forest (Depth = 8)

[13]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [8])
               .build())
```

```
# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 8 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 8 : 90.36409290646579
--- 70.10215163230896 seconds ---
------------------
```

All depths in one:

[14]:
```
# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [3, 4, 5, 6, 7, 8])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
```

```
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest Best Accuracy : {accuracy}')



print(f'Best Max Depth: {BestModel.getMaxDepth()}')
print(f'Num Trees (default): {BestModel.getNumTrees}')
print(f'Max Bins (default): {BestModel.getMaxBins()}')
```

```
Random Forest Best Accuracy : 90.36409290646579
Best Max Depth: 8
Num Trees (default): 20
Max Bins (default): 32
```

Summary

[15]:
```
for i in range(6):
    print(f'{(i+3)}-Depth Forest: {acc[i]} | {timetook[i]}')
```

```
3-Depth Forest: 71.28060263653484 | 59.366854429244995
4-Depth Forest: 78.78217200251099 | 52.116631507873535
5-Depth Forest: 82.04645323289391 | 53.21452593803406
6-Depth Forest: 84.77715003138732 | 56.8309063911438
7-Depth Forest: 88.38669177652228 | 62.201735496520996
8-Depth Forest: 90.36409290646579 | 70.10215163230896
```

So the best model is 8-depth Forest

[16]:
```
timetook = []
acc = []
```

Random Forest (Depth = 8 | MaxBins = 4)

[17]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [8])
               .addGrid(classifier.maxBins, [4])
               .build())

# Cross Validator:
```

```
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 8, MaxBins 4 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 8, MaxBins 4 : 88.60640301318267
--- 55.7332079410553 seconds ---
------------------
```

Random Forest (Depth = 8 | MaxBins = 8)

[18]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [8])
               .addGrid(classifier.maxBins, [8])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
```

```
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 8, MaxBins 8 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 8, MaxBins 8 : 90.20715630885122
--- 57.33148121833801 seconds ---
------------------
```

Random Forest (Depth = 8 | MaxBins = 16)

[19]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [8])
               .addGrid(classifier.maxBins, [16])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)
```

```
accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 8, MaxBins 16 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 8, MaxBins 16 : 90.33270558694288
--- 60.968984603881836 seconds ---
------------------
```

Random Forest (Depth = 8 | MaxBins = 32)

[20]:
```
start_time = time.time()


# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [8])
               .addGrid(classifier.maxBins, [32])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest with tree depth 8, MaxBins 32 : {accuracy}')
acc.append(accuracy)


end_time = time.time()
print("--- %s seconds ---" % (end_time - start_time))
```

```
print("------------------")
timetook.append(end_time - start_time)
```

```
Random Forest with tree depth 8, MaxBins 32 : 90.36409290646579
--- 74.39864802360535 seconds ---
------------------
```

All depths in one:

[21]:
```
# Add parameters of your choice here:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [8])
               .addGrid(classifier.maxBins, [4, 8, 16, 32])
               .build())

# Cross Validator:
crossval = CrossValidator(estimator = classifier, estimatorParamMaps =␣
 ↪paramGrid, evaluator = MulticlassClassificationEvaluator(), numFolds=5)

# Fit Model: Run cross-validation, and choose the best set of parameters.
fitModel = crossval.fit(train)

# Retrieve best model from cross val
BestModel = fitModel.bestModel
# print("Feature Importance Scores (add up to 1)")
# featureImportances = BestModel.featureImportances.toArray()
# print(featureImportances)

predictions = fitModel.transform(test)

accuracy = (MC_evaluator.evaluate(predictions))*100
print(f'Random Forest Accuracy: {accuracy}')


print(f'Best Max Bins: {BestModel.getMaxBins()}')
print(f'Best Max Depth: {BestModel.getMaxDepth()}')
print(f'Num Trees (default): {BestModel.getNumTrees}')
```

```
Random Forest Accuracy: 90.33270558694288
Best Max Bins: 16
Best Max Depth: 8
Num Trees (default): 20
```

Summary

[22]:
```
print(f'8-Depth, 4-MaxBin Forest: {acc[0]} | {timetook[0]}')
print(f'8-Depth, 8-MaxBin Forest: {acc[1]} | {timetook[1]}')
```

```
print(f'8-Depth, 16-MaxBin Forest: {acc[2]} | {timetook[2]}')
print(f'8-Depth, 32-MaxBin Forest: {acc[3]} | {timetook[3]}')
```

```
8-Depth, 4-MaxBin Forest: 88.60640301318267 | 55.7332079410553
8-Depth, 8-MaxBin Forest: 90.20715630885122 | 57.33148121833801
8-Depth, 16-MaxBin Forest: 90.33270558694288 | 60.968984603881836
8-Depth, 32-MaxBin Forest: 90.36409290646579 | 74.39864802360535
```

So Maxbins = 8 is enough for our model.