Part1

January 3, 2023

Initiate Spark

[1]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Part1').getOrCreate()
```

Import Required Modules

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

Import Data

Read Master csv

[3]:
```
masterdf = spark.read.csv('master3.csv', inferSchema=True, header=True)
masterdf.limit(5).toPandas()
```

[3]:
```
                     Id  Class     db    mov   push    dd   call   lea  align  \
0  01kcPWA9K2BOxQeS5Rju      1    105     89     81    81     53    36     28
1  04EjIdbPV5e1XroFOpiN      1   3871   9764   5927  2092   2900  1230    660
2  05EeG39MTRrI6VY21DPd      1   2561   2415    915   592    461   157    295
3  05rJTUWYAKNegBk2wE8X      1  19684  32566  28620  2650  10686  5819   1333
4  0AnoOZDNbPXIr2MRBSCJ      1   3162   2624    781   120    462   344    177

    pop  …  topic  options  pt  pmxcd  lprect  vcmpss  invalid  memotected  \
0    19  …    NaN      NaN NaN    NaN     NaN     NaN      NaN         NaN
1  1527  …    NaN      NaN NaN    NaN     NaN     NaN      NaN         NaN
2   376  …    NaN      NaN NaN    NaN     NaN     NaN      NaN         NaN
3  6384  …    NaN      NaN NaN    NaN     NaN     NaN      NaN         NaN
4   539  …    NaN      NaN NaN    NaN     NaN     NaN      NaN         NaN

   start  bmp
0    NaN  NaN
1    NaN  NaN
2    NaN  NaN
3    NaN  NaN
4    NaN  NaN
```

```
[5 rows x 721 columns]
```

Remove all-null rows and replace null values with 0

[4]:
```
masterdf.count()
```

[4]:
```
10868
```

[5]:
```
masterdf = masterdf.na.drop("all")
masterdf = masterdf.na.fill(0)
```

[6]:
```
masterdf.count()
```

[6]:
```
10868
```

[7]:
```
masterdf.limit(5).toPandas()
```

[7]:
```
                     Id  Class     db    mov   push    dd   call   lea  align  \
0  01kcPWA9K2BOxQeS5Rju      1    105     89     81    81     53    36     28
1  04EjIdbPV5e1XroFOpiN      1   3871   9764   5927  2092   2900  1230    660
2  05EeG39MTRrI6VY21DPd      1   2561   2415    915   592    461   157    295
3  05rJTUWYAKNegBk2wE8X      1  19684  32566  28620  2650  10686  5819   1333
4  0AnoOZDNbPXIr2MRBSCJ      1   3162   2624    781   120    462   344    177

    pop  …  topic  options  pt  pmxcd  lprect  vcmpss  invalid  memotected  \
0    19  …      0        0   0      0       0       0        0           0
1  1527  …      0        0   0      0       0       0        0           0
2   376  …      0        0   0      0       0       0        0           0
3  6384  …      0        0   0      0       0       0        0           0
4   539  …      0        0   0      0       0       0        0           0

   start  bmp
0      0    0
1      0    0
2      0    0
3      0    0
4      0    0

[5 rows x 721 columns]
```

Required Modules

[8]:
```
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

[9]:

```
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 (Treesize = 10)

[10]:

```
start_time = time.time()


# Add Parameters:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
```

```
                .addGrid(classifier.numTrees, [10])
                .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 size 10 : {accuracy}')
acc.append(accuracy)


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

```
Feature Importance Scores (add up to 1)
[1.82456616e-02 1.16921894e-02 3.82247845e-03 1.90304845e-02
 7.52574665e-03 1.20591241e-02 2.09116029e-02 6.18522802e-03
 2.07975354e-02 1.29674613e-02 1.23129037e-02 3.12214977e-02
 1.77772861e-02 1.26845239e-02 4.70766862e-03 9.08709356e-03
 4.42593399e-02 4.23537649e-02 5.39047532e-03 2.94927787e-02
 2.10393403e-02 4.11902984e-02 8.20118308e-03 5.40164797e-03
 3.73194559e-02 9.38544643e-03 1.44670994e-02 1.96130554e-02
 8.06311993e-03 9.31372364e-03 2.95013555e-03 1.07681888e-02
 1.92945841e-02 1.91072840e-03 1.52722968e-02 2.58385899e-03
 1.16968096e-02 1.64437876e-02 1.10919385e-02 1.56316669e-03
 1.92925184e-03 5.86418729e-03 2.36990243e-02 1.78890036e-02
 5.26856107e-02 3.47739239e-02 4.45590919e-03 1.96714660e-02
 7.64494161e-04 0.00000000e+00 3.78319649e-03 1.36496436e-03
 4.20998179e-02 3.63647145e-03 3.53648185e-04 3.75211722e-03
 1.36889084e-02 1.41749108e-03 1.79819650e-02 1.05266894e-03
 1.03535714e-03 4.02793946e-04 8.37450326e-04 5.00904911e-04
 4.94803725e-05 3.25568719e-04 4.85093219e-03 5.29860080e-04
 6.16034563e-03 4.24294194e-05 8.25214192e-05 0.00000000e+00
```

```
2.84794089e-04 0.00000000e+00 3.22206616e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 1.63999750e-03 3.17739769e-04
0.00000000e+00 4.89026516e-05 1.92543210e-03 4.15199453e-03
1.66775162e-04 1.46647874e-04 9.21325530e-04 1.75588314e-05
0.00000000e+00 2.86701845e-03 4.19771287e-04 1.62854962e-03
3.65786210e-04 0.00000000e+00 9.21874663e-04 5.66496025e-05
2.33983596e-03 1.06625151e-04 5.87719945e-04 3.47768852e-03
8.94629987e-03 3.68613162e-04 2.19920208e-03 5.39480351e-04
9.52192292e-03 0.00000000e+00 0.00000000e+00 1.73698851e-03
7.75163380e-04 3.27327802e-05 4.13257972e-03 0.00000000e+00
3.07387150e-04 0.00000000e+00 5.81838533e-05 6.62956722e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 2.18741237e-04 4.10000512e-03
3.43214876e-05 1.70027713e-05 0.00000000e+00 5.35091130e-04
1.88305622e-04 5.92360679e-04 4.56400074e-04 1.64282364e-05
4.89578189e-03 0.00000000e+00 1.28896253e-04 0.00000000e+00
1.51009867e-04 1.19892636e-04 0.00000000e+00 0.00000000e+00
1.27186245e-04 0.00000000e+00 0.00000000e+00 9.70175174e-05
1.58796353e-03 3.36349273e-05 1.00963158e-04 1.63367845e-04
3.12270854e-03 1.47031242e-04 0.00000000e+00 0.00000000e+00
2.76468949e-03 4.64699402e-05 0.00000000e+00 0.00000000e+00
0.00000000e+00 2.31792429e-05 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 5.99242325e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 3.65781883e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
1.06148300e-03 2.22109200e-05 3.31132041e-06 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 1.53491762e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 9.00823539e-05
0.00000000e+00 0.00000000e+00 4.53407236e-05 0.00000000e+00
0.00000000e+00 6.54633044e-05 6.39664142e-05 2.85369997e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 6.60485009e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
1.81362894e-05 0.00000000e+00 0.00000000e+00 0.00000000e+00
7.55700234e-05 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 2.17592489e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 3.19758699e-05
2.86113038e-05 0.00000000e+00 3.42695485e-05 2.59573933e-04
0.00000000e+00 1.17435240e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 2.26703618e-05 9.94900898e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 3.09069046e-03 1.89824493e-03
5.62937911e-03 8.40505992e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 9.21284836e-05 0.00000000e+00
```

```
0.00000000e+00 1.21810539e-04 0.00000000e+00 0.00000000e+00
1.43655542e-04 0.00000000e+00 0.00000000e+00 3.22417646e-05
0.00000000e+00 4.86734033e-03 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.34134578e-05
0.00000000e+00 2.70267133e-03 0.00000000e+00 2.55036477e-04
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.12681012e-04
0.00000000e+00 0.00000000e+00 0.00000000e+00 6.38997438e-04
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 4.39467108e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 5.24933003e-06 1.01852900e-02
6.83438070e-04 3.11747009e-03 0.00000000e+00 4.69378546e-05
0.00000000e+00 0.00000000e+00 6.14280231e-05 0.00000000e+00
0.00000000e+00 1.22239618e-02 4.19613205e-04 3.92533274e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
3.15442606e-05 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 7.10943571e-05 5.10132402e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
2.51422569e-05 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 8.36880636e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
4.05917071e-07 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 2.04237494e-07 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 3.22384006e-04 6.95475917e-05
0.00000000e+00 3.65877730e-05 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 3.41123931e-05
0.00000000e+00 0.00000000e+00 2.56864566e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
```

```
0.00000000e+00 6.77168546e-05 1.51550179e-04 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 7.12137108e-05 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 6.65333013e-05 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 1.85433943e-05 0.00000000e+00 9.48835880e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 6.33654187e-05 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 1.83285850e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 1.19546172e-04 1.04195508e-03
0.00000000e+00 3.33074901e-04 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 4.98939049e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
2.46223137e-05 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 1.55409130e-04
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 1.61414521e-07 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 3.29881017e-05
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
```

```
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 6.20312551e-05 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
 0.00000000e+00 0.00000000e+00 0.00000000e+00]
Random Forest with tree size 10 : 97.91212215643502
--- 44.63968014717102 seconds ---
------------------
```

Random Forest (Treesize = 20)

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



# Add Parameters:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
               .addGrid(classifier.numTrees, [20])
               .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 size 20 : {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 size 20 : 98.06793393580556
--- 40.03208589553833 seconds ---
------------------
```

Random Forest (Treesize = 30)

[12]:

```
start_time = time.time()



# Add Parameters:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
               .addGrid(classifier.numTrees, [30])
               .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 size 30 : {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 size 30 : 98.13025864755376
--- 43.3293354511261 seconds ---
------------------
```

Random Forest (Treesize = 40)

[13]:

```
start_time = time.time()



# Add Parameters:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
               .addGrid(classifier.numTrees, [40])
               .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 size 40 : {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 size 40 : 98.09909629167966
--- 46.58205437660217 seconds ---
------------------
```

Random Forest (Treesize = 50)

[14]:
```
start_time = time.time()


# Add Parameters:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
               .addGrid(classifier.numTrees, [50])
               .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 size 50 : {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 size 50 : 98.13025864755376
--- 48.537452697753906 seconds ---
-------------------
```

All in one (no time calculation)

[15]:
```
# Add Parameters:
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
               .addGrid(classifier.numTrees, [10, 20, 30, 40, 50])
```

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

```
Random Forest best accuracy : 98.13025864755376
Best Num Trees: 50
Max Bins (default): 32
Max Depth (default): 10
```

Summary

[16]:

```
for i in range(5):
    print(f'{(i+1)*10}-Tree Forest: {acc[i]} | {timetook[i]}')
```

```
10-Tree Forest: 97.91212215643502 | 44.63968014717102
20-Tree Forest: 98.06793393580556 | 40.03208589553833
30-Tree Forest: 98.13025864755376 | 43.3293354511261
40-Tree Forest: 98.09909629167966 | 46.58205437660217
50-Tree Forest: 98.13025864755376 | 48.537452697753906
```

So the best model is 30-Tree Forest

Precision, Recall, F-measure, True Positive Rate, False Positive Rate:

Random Forest (Treesize = 30) Metric Parameters

[17]:

```
import sklearn

# Add Parameters:
```

```
classifier = RandomForestClassifier()
paramGrid = (ParamGridBuilder() \
               .addGrid(classifier.maxDepth, [10])
               .addGrid(classifier.numTrees, [30])
               .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)


predictions = fitModel.transform(test)

y_true = predictions.select(['label']).collect()
y_pred = predictions.select(['prediction']).collect()

from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       887
         1.0       1.00      1.00      1.00       725
         2.0       0.93      1.00      0.96       463
         3.0       0.98      0.93      0.95       362
         4.0       1.00      0.98      0.99       267
         5.0       0.98      0.98      0.98       246
         6.0       1.00      0.90      0.95       136
         7.0       0.88      1.00      0.94       107
         8.0       1.00      0.81      0.90        16

    accuracy                           0.98      3209
   macro avg       0.98      0.95      0.96      3209
weighted avg       0.98      0.98      0.98      3209
```

[18]:

```
preds_and_labels = predictions.select(['prediction','label'])
```

[19]:

```
preds_and_labels.show()
```

```
+----------+-----+
|prediction|label|
+----------+-----+
|       7.0|  0.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
|       2.0|  2.0|
+----------+-----+
only showing top 20 rows
```

[20]:

```
a = [[0 for i in range(8)] for j in range(8)]
for i in range(8):
    for j in range(8):
        a[i][j] = preds_and_labels.filter(preds_and_labels.prediction == i).
  ↪filter(preds_and_labels.label == j).count()
```

[21]:

```
import pandas as pd
pd.DataFrame(a)
```

[21]:

```
     0    1    2    3    4    5    6    7
0  885    0    0    0    0    0    1    0
1    0  722    0    0    0    0    0    0
2    0    3  461   23    3    2    0    0
3    0    0    1  335    2    1    2    0
4    0    0    0    0  261    0    0    0
5    1    0    1    1    1  242    1    0
6    0    0    0    0    0    0  123    0
7    1    0    0    3    0    1    9  107
```

[22]:

```
false_positive = 0
true_positive = 0
for i in range (8):
    for j in range (8):
        if i != j:
            false_positive += a[i][j]
        else:
            true_positive += a[i][j]
print(f'False Positive: {false_positive}')
print(f'True Positive: {true_positive}')
```

```
False Positive: 57
True Positive: 3136
```