In [1]:
```
from pyspark.sql.types import BooleanType
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import LinearSVC
from pyspark.sql.session import SparkSession
from pyspark.sql.functions import expr
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from helpers.helper_functions import translate_to_file_string
from pyspark.sql import DataFrameReader
from pyspark.sql import SparkSession
from pyspark.ml.feature import IndexToString, Normalizer, StringInd
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.classification import DecisionTreeClassifier
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from helpers.helper_functions import translate_to_file_string
from sklearn.metrics import roc_curve, auc
import seaborn as sns
import pandas as pd
import os
import warnings
import matplotlib.pyplot as plt
warnings.filterwarnings('ignore')
```

In [2]:
```
inputFile = translate_to_file_string("../data/heart_val.csv")
```

In [3]:
```
spark = (SparkSession
       .builder
       .appName("HeartDiseaseAnalRf")
       .getOrCreate())
```

In [4]:
```
# load data file.
# create a DataFrame using an ifered Schema
df = spark.read.option("header", "true") \
       .option("inferSchema", "true") \
       .option("delimiter", ";") \
       .csv(inputFile)
```

In [5]:
```
#remove the outliner
df_filtered=df.filter(df.age > 30)
```

In [6]:
```
#transform labels
labelIndexer = StringIndexer().setInputCol("target").setOutputCol("
sexIndexer = StringIndexer().setInputCol("sex").setOutputCol("sex_n
```

In [7]:
```
#feature columns
featureCols = df.columns.copy()
featureCols.remove("target")
featureCols.remove("sex")
featureCols = featureCols + ["sex_num"]
```

In [8]:
```
#vector assembler of all features
assembler =  VectorAssembler(outputCol="features", inputCols=featur
```

In [9]:
```
#Build feauture Indexer
featureIndexer = VectorIndexer(inputCol="features",outputCol="index
```

In [10]:
```
#Convert Indexed labels back to original labels
predConverter = IndexToString(inputCol="prediction",outputCol="pred
```

In [11]:
```
#create the Random Forest Classification
rf = RandomForestClassifier(labelCol="label", featuresCol="features
                 minInstancesPerNode=1, featureSubsetStrategy='sqrt
```

In [12]:
```
# build a network para grip
paramGrid = (ParamGridBuilder()
             #.addGrid(rf.maxDepth, [2, 5, 10, 20, 30])
               .addGrid(rf.maxDepth, [2, 5, 10])
             #.addGrid(rf.maxBins, [10, 20, 40, 80, 100])
               .addGrid(rf.maxBins, [5, 10, 20, 30])
             #.addGrid(rf.numTrees, [5, 20, 50, 100, 500])
               .addGrid(rf.numTrees, [5, 20, 50])
             .build())
```

In [13]:
```
#split data for testing

splits = df.randomSplit([0.6, 0.4 ], 5756)
train = splits[0]
test = splits[1]
```

In [14]:
```
#Pipelining of all steps
pipeline = Pipeline(stages= [labelIndexer,sexIndexer,  assembler, f
```

In [16]:
```
#build evaluator
evaluator =  BinaryClassificationEvaluator(labelCol="label",rawPred
#evaluator = RegressionEvaluator(labelCol="label", predictionCol="p
```

In [17]:
```
#Cross validator
cvRf = CrossValidator(estimator=pipeline, evaluator=evaluator,estim
```

In [18]:
```
#train model
rfModel = cvRf.fit(train)
```

In [19]:
```
#Find out the best model
rfBestModel = rfModel.bestModel.stages[4] # the stage at index 1 in
print("Best Params: \n", rfBestModel.explainParams())
print("Param Map: \n", rfBestModel.extractParamMap())
#print(cvSVMModel.getEstimatorParamMaps()[np.argmax(cvSVMModel.avgM
```

```
Best Params:
 bootstrap: Whether bootstrap samples are used when building trees
. (default: True)
cacheNodeIds: If false, the algorithm will pass trees to executors
to match instances with nodes. If true, the algorithm will cache n
ode IDs for each instance. Caching can speed up training of deeper
trees. Users can set how often should the cache be checkpointed or
disable it by setting checkpointInterval. (default: False)
checkpointInterval: set checkpoint interval (>= 1) or disable chec
kpoint (-1). E.g. 10 means that the cache will get checkpointed ev
ery 10 iterations. Note: this setting will be ignored if the check
point directory is not set in the SparkContext. (default: 10)
featureSubsetStrategy: The number of features to consider for spli
ts at each tree node. Supported options: 'auto' (choose automatica
lly for task: If numTrees == 1, set to 'all'. If numTrees > 1 (for
est), set to 'sqrt' for classification and to 'onethird' for regre
ssion), 'all' (use all features), 'onethird' (use 1/3 of the featu
res), 'sqrt' (use sqrt(number of features)), 'log2' (use log2(numb
er of features)), 'n' (when n is in the range (0, 1.0], use n * nu
mber of features. When n is in the range (1, number of features),
use n features). default = 'auto' (default: auto, current: sqrt)
featuresCol: features column name. (default: features, current: fe
atures)
impurity: Criterion used for information gain calculation (case-in
sensitive). Supported options: entropy, gini (default: gini, curre
nt: gini)
labelCol: label column name. (default: label, current: label)
leafCol: Leaf indices column name. Predicted leaf index of each in
stance in each tree by preorder. (default: )
maxBins: Max number of bins for discretizing continuous features.
Must be >=2 and >= number of categories for any categorical featur
e. (default: 32, current: 10)
maxDepth: Maximum depth of the tree. (>= 0) E.g., depth 0 means 1
leaf node; depth 1 means 1 internal node + 2 leaf nodes. (default:
5, current: 5)
maxMemoryInMB: Maximum memory in MB allocated to histogram aggrega
tion. If too small, then 1 node will be split per iteration, and i
ts aggregates may exceed this size. (default: 256)
minInfoGain: Minimum information gain for a split to be considered
at a tree node. (default: 0.0)
minInstancesPerNode: Minimum number of instances each child must h
ave after split. If a split causes the left or right child to have
fewer than minInstancesPerNode, the split will be discarded as inv
alid. Should be >= 1. (default: 1, current: 1)
minWeightFractionPerNode: Minimum fraction of the weighted sample
count that each child must have after split. If a split causes the
fraction of the total weight in the left or right child to be less
than minWeightFractionPerNode, the split will be discarded as inva
```

```
lid. Should be in interval [0.0, 0.5). (default: 0.0)
numTrees: Number of trees to train (>= 1). (default: 20, current:
50)
predictionCol: prediction column name. (default: prediction)
probabilityCol: Column name for predicted class conditional probab
ilities. Note: Not all models output well-calibrated probability e
stimates! These probabilities should be treated as confidences, no
t precise probabilities. (default: probability)
rawPredictionCol: raw prediction (a.k.a. confidence) column name.
(default: rawPrediction)
seed: random seed. (default: -3576768138202297541, current: 12345)
subsamplingRate: Fraction of the training data used for learning e
ach decision tree, in range (0, 1]. (default: 1.0, current: 0.95)
thresholds: Thresholds in multi-class classification to adjust the
probability of predicting each class. Array must have length equal
to the number of classes, with values > 0, excepting that at most
one value may be 0. The class with largest value p/t is predicted,
where p is the original probability of that class and t is the cla
ss's threshold. (undefined)
weightCol: weight column name. If this is not set or empty, we tre
at all instance weights as 1.0. (undefined)
Param Map:
 {Param(parent='RandomForestClassifier_07933e31cf5c', name='bootst
rap', doc='Whether bootstrap samples are used when building trees.
'): True, Param(parent='RandomForestClassifier_07933e31cf5c', name
='cacheNodeIds', doc='If false, the algorithm will pass trees to e
xecutors to match instances with nodes. If true, the algorithm wil
l cache node IDs for each instance. Caching can speed up training
of deeper trees. Users can set how often should the cache be check
pointed or disable it by setting checkpointInterval.'): False, Par
am(parent='RandomForestClassifier_07933e31cf5c', name='checkpointI
nterval', doc='set checkpoint interval (>= 1) or disable checkpoin
t (-1). E.g. 10 means that the cache will get checkpointed every 1
0 iterations. Note: this setting will be ignored if the checkpoint
directory is not set in the SparkContext.'): 10, Param(parent='Ran
domForestClassifier_07933e31cf5c', name='featureSubsetStrategy', d
oc="The number of features to consider for splits at each tree nod
e. Supported options: 'auto' (choose automatically for task: If nu
mTrees == 1, set to 'all'. If numTrees > 1 (forest), set to 'sqrt'
for classification and to 'onethird' for regression), 'all' (use a
ll features), 'onethird' (use 1/3 of the features), 'sqrt' (use sq
rt(number of features)), 'log2' (use log2(number of features)), 'n
' (when n is in the range (0, 1.0], use n * number of features. Wh
en n is in the range (1, number of features), use n features). def
ault = 'auto'"): 'sqrt', Param(parent='RandomForestClassifier_0793
3e31cf5c', name='featuresCol', doc='features column name.'): 'feat
ures', Param(parent='RandomForestClassifier_07933e31cf5c', name='i
mpurity', doc='Criterion used for information gain calculation (ca
se-insensitive). Supported options: entropy, gini'): 'gini', Param
(parent='RandomForestClassifier_07933e31cf5c', name='labelCol', do
c='label column name.'): 'label', Param(parent='RandomForestClassi
fier_07933e31cf5c', name='leafCol', doc='Leaf indices column name.
Predicted leaf index of each instance in each tree by preorder.'):
'', Param(parent='RandomForestClassifier_07933e31cf5c', name='maxB
```

```
ins', doc='Max number of bins for discretizing continuous features
.  Must be >=2 and >= number of categories for any categorical fea
ture.'): 10, Param(parent='RandomForestClassifier_07933e31cf5c', n
ame='maxDepth', doc='Maximum depth of the tree. (>= 0) E.g., depth
0 means 1 leaf node; depth 1 means 1 internal node + 2 leaf nodes.

'): 5, Param(parent='RandomForestClassifier_07933e31cf5c', name='m
axMemoryInMB', doc='Maximum memory in MB allocated to histogram ag
gregation. If too small, then 1 node will be split per iteration,
and its aggregates may exceed this size.'): 256, Param(parent='Ran
domForestClassifier_07933e31cf5c', name='minInfoGain', doc='Minimu
m information gain for a split to be considered at a tree node.'):
0.0, Param(parent='RandomForestClassifier_07933e31cf5c', name='min
InstancesPerNode', doc='Minimum number of instances each child mus
t have after split. If a split causes the left or right child to h
ave fewer than minInstancesPerNode, the split will be discarded as
invalid. Should be >= 1.'): 1, Param(parent='RandomForestClassifie
r_07933e31cf5c', name='minWeightFractionPerNode', doc='Minimum fra
ction of the weighted sample count that each child must have after
split. If a split causes the fraction of the total weight in the l
eft or right child to be less than minWeightFractionPerNode, the s
plit will be discarded as invalid. Should be in interval [0.0, 0.5
).'): 0.0, Param(parent='RandomForestClassifier_07933e31cf5c', nam
e='numTrees', doc='Number of trees to train (>= 1).'): 50, Param(p
arent='RandomForestClassifier_07933e31cf5c', name='predictionCol',
doc='prediction column name.'): 'prediction', Param(parent='Random
ForestClassifier_07933e31cf5c', name='probabilityCol', doc='Column
name for predicted class conditional probabilities. Note: Not all
models output well-calibrated probability estimates! These probabi
lities should be treated as confidences, not precise probabilities
.'): 'probability', Param(parent='RandomForestClassifier_07933e31c
f5c', name='rawPredictionCol', doc='raw prediction (a.k.a. confide
nce) column name.'): 'rawPrediction', Param(parent='RandomForestCl
assifier_07933e31cf5c', name='seed', doc='random seed.'): 12345, P
aram(parent='RandomForestClassifier_07933e31cf5c', name='subsampli
ngRate', doc='Fraction of the training data used for learning each
decision tree, in range (0, 1].'): 0.95}
```

In [20]:
```
#test model
predictions = rfModel.transform(test)
predictions.show()
```

```
+---+---+---+--------+----+---+-------+-------+-----+-------+-----
+---+----+------+-----+-------+--------------------+-------------
------+--------------------+--------------------+----------+------
--------+
|age|sex| cp|trestbps|chol|fbs|restecg|thalach|exang|oldpeak|slope
| ca|thal|target|label|sex_num|            features|     indexedFe
atures|       rawPrediction|         probability|prediction|predic
tedLabel|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----
+---+----+------+-----+-------+--------------------+-------------
------+--------------------+--------------------+----------+------
--------+
```

```
| 29|  m|  1|     130| 204|  0|      0|    202|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|(13,[0,1,2,3,6,9,...|(13,[0,1,2,3,6
,9,...|[46.4018604775703...|[0.92803720955140...|       0.0|
y|
| 34|  f|  1|     118| 210|  0|      1|    192|    0|    0.7|    2
|  0|   2|     y|  0.0|    1.0|[34.0,1.0,118.0,2...|[34.0,1.0,118.
0,2...|[47.9306607001698...|[0.95861321400339...|       0.0|
y|
| 34|  m|  3|     118| 182|  0|      0|    174|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|(13,[0,1,2,3,6,9,...|(13,[0,1,2,3,6
,9,...|[44.9436756250607...|[0.89887351250121...|       0.0|
y|
| 35|  m|  1|     122| 192|  0|      1|    174|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|[35.0,1.0,122.0,1...|[35.0,1.0,122.
0,1...|[47.0435649587846...|[0.94087129917569...|       0.0|
y|
| 38|  m|  3|     120| 231|  0|      1|    182|    1|    3.8|    1
|  0|   3|     n|  1.0|    0.0|[38.0,3.0,120.0,2...|[38.0,3.0,120.
0,2...|[32.7243886743886...|[0.65448777348777...|       0.0|
y|
| 39|  f|  2|     138| 220|  0|      1|    152|    0|    0.0|    1
|  0|   2|     y|  0.0|    1.0|[39.0,2.0,138.0,2...|[39.0,2.0,138.
0,2...|[44.8974618267401...|[0.89794923653480...|       0.0|
y|
| 41|  f|  1|     126| 306|  0|      1|    163|    0|    0.0|    2
|  0|   2|     y|  0.0|    1.0|[41.0,1.0,126.0,3...|[41.0,1.0,126.
0,3...|[47.5881431294546...|[0.95176286258909...|       0.0|
y|
| 41|  f|  2|     112| 268|  0|      0|    172|    1|    0.0|    2
|  0|   2|     y|  0.0|    1.0|[41.0,2.0,112.0,2...|[41.0,2.0,112.
0,2...|[45.2132587171169...|[0.90426517434233...|       0.0|
y|
| 41|  m|  1|     135| 203|  0|      1|    132|    0|    0.0|    1
|  0|   1|     y|  0.0|    0.0|[41.0,1.0,135.0,2...|[41.0,1.0,135.
0,2...|[29.5227982173290...|[0.59045596434658...|       0.0|
y|
| 42|  m|  2|     120| 240|  1|      1|    194|    0|    0.8|    0
|  0|   3|     y|  0.0|    0.0|[42.0,2.0,120.0,2...|[42.0,2.0,120.
0,2...|[39.1810490776967...|[0.78362098155393...|       0.0|
y|
| 42|  m|  2|     130| 180|  0|      1|    150|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|[42.0,2.0,130.0,1...|[42.0,2.0,130.
0,1...|[39.3493756433048...|[0.78698751286609...|       0.0|
y|
| 42|  m|  3|     148| 244|  0|      0|    178|    0|    0.8|    2
|  2|   2|     y|  0.0|    0.0|[42.0,3.0,148.0,2...|[42.0,3.0,148.
0,2...|[40.3993376042236...|[0.80798675208447...|       0.0|
y|
| 43|  f|  0|     132| 341|  1|      0|    136|    1|    3.0|    1
|  0|   3|     n|  1.0|    1.0|[43.0,0.0,132.0,3...|[43.0,0.0,132.
0,3...|[6.04144327894327...|[0.12082886557886...|       1.0|
n|
| 43|  m|  0|     120| 177|  0|      0|    120|    1|    2.5|    1
```

```
|  0|   3|     n|  1.0|    0.0|[43.0,0.0,120.0,1...|[43.0,0.0,120.
0,1...|[2.25088082207647...|[0.04501761644152...|       1.0|
n|

| 43|  m|  0|     132| 247|  1|      0|    143|    1|    0.1|    1
|  4|   3|     n|  1.0|    0.0|[43.0,0.0,132.0,2...|[43.0,0.0,132.
0,2...|[9.01827228327228...|[0.18036544566544...|       1.0|
n|
| 43|  m|  0|     150| 247|  0|      1|    171|    0|    1.5|    2
|  0|   2|     y|  0.0|    0.0|[43.0,0.0,150.0,2...|[43.0,0.0,150.
0,2...|[37.3660314166051...|[0.74732062833210...|       0.0|
y|
| 44|  f|  2|     108| 141|  0|      1|    175|    0|    0.6|    1
|  0|   2|     y|  0.0|    1.0|[44.0,2.0,108.0,1...|[44.0,2.0,108.
0,1...|[45.1436922270941...|[0.90287384454188...|       0.0|
y|
| 44|  m|  0|     120| 169|  0|      1|    144|    1|    2.8|    0
|  0|   1|     n|  1.0|    0.0|[44.0,0.0,120.0,1...|[44.0,0.0,120.
0,1...|[10.5787185888848...|[0.21157437177769...|       1.0|
n|
| 44|  m|  1|     120| 220|  0|      1|    170|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|[44.0,1.0,120.0,2...|[44.0,1.0,120.
0,2...|[47.1440940593137...|[0.94288188118627...|       0.0|
y|
| 44|  m|  1|     120| 263|  0|      1|    173|    0|    0.0|    2
|  0|   3|     y|  0.0|    0.0|[44.0,1.0,120.0,2...|[44.0,1.0,120.
0,2...|[41.9186747924277...|[0.83837349584855...|       0.0|
y|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----
+---+----+------+-----+-------+-------------------+--------------
------+-------------------+-------------------+----------+------
--------+
only showing top 20 rows
```

In [21]:
```
accuracy = evaluator.evaluate(predictions)
print("Test Error = " ,(1.0 - accuracy))
```

```
Test Error =  0.10013495276653206
```

In [22]:
```
spark.stop()
```