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.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml import Pipeline
from helpers.helper_functions import translate_to_file_string
from pyspark.mllib.evaluation import MulticlassMetrics
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("HeartDiseaseAnalSVM")
       .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)
print(df.printSchema())

#Pandas df for visualization
dfp = df.toPandas()
```

```
root
 |-- age: integer (nullable = true)
 |-- sex: string (nullable = true)
 |-- cp: integer (nullable = true)
 |-- trestbps: integer (nullable = true)
 |-- chol: integer (nullable = true)
 |-- fbs: integer (nullable = true)
 |-- restecg: integer (nullable = true)
 |-- thalach: integer (nullable = true)
 |-- exang: integer (nullable = true)
 |-- oldpeak: double (nullable = true)
 |-- slope: integer (nullable = true)
 |-- ca: integer (nullable = true)
 |-- thal: integer (nullable = true)
 |-- target: string (nullable = true)

None
```

In [5]:

```
#transform labels
labelIndexer = StringIndexer().setInputCol("target").setOutputCol("
sexIndexer = StringIndexer().setInputCol("sex").setOutputCol("sex_n
```

In [6]:

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

In [7]:

```
#vector assembler
assembler =  VectorAssembler(outputCol="features", inputCols=list(f
```

In [8]:

```
#Build feauture Indexer
featureIndexer = VectorIndexer(inputCol="features",outputCol="index
```

In [9]:

```
#Convert Indexed labels back to original labels
predConverter = IndexToString(inputCol="prediction",outputCol="pred
```

In [10]:

```
lsvc = LinearSVC(labelCol="label",aggregationDepth=2, featuresCol="
```

In [11]:

```
# build a Parameter Grip for testing
paramGrid = ParamGridBuilder().addGrid(lsvc.maxIter, [100])\
                                 .addGrid(lsvc.regParam, [0.1, 0.00
                                 .addGrid(lsvc.standardization, [Tr
                                 .build()
```

In [12]:

```
#split data for testing

splits = df.randomSplit([0.8, 0.2 ], 1234)
train = splits[0]
test = splits[1]
```

In [13]:

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

In [15]:

```
#build evaluator
evaluator =  BinaryClassificationEvaluator(labelCol="label",rawPred
```

In [16]:

```
#Cross validator
cvSVM = CrossValidator(estimator=pipeline, evaluator=evaluator,esti
```

In [17]:

```
#train model
cvSVMModel = cvSVM.fit(train)
```

In [18]:
```
#Find out the best model
linearSVCModel = cvSVMModel.bestModel.stages[4] # the stage at inde
print("Best Params: \n", linearSVCModel.explainParams())
print("Param Map: \n", linearSVCModel.extractParamMap())
```

```
Best Params: 
 aggregationDepth: suggested depth for treeAggregate (>= 2). (defa
ult: 2, current: 2)
featuresCol: features column name. (default: features, current: fe
atures)
fitIntercept: whether to fit an intercept term. (default: True)
labelCol: label column name. (default: label, current: label)
maxIter: max number of iterations (>= 0). (default: 100, current: 
100)
predictionCol: prediction column name. (default: prediction)
rawPredictionCol: raw prediction (a.k.a. confidence) column name. 
(default: rawPrediction)
regParam: regularization parameter (>= 0). (default: 0.0, current: 
0.1)
standardization: whether to standardize the training features befo
re fitting the model. (default: True, current: True)
threshold: The threshold in binary classification applied to the l
inear model prediction.  This threshold can be any real number, wh
ere Inf will make all predictions 0.0 and -Inf will make all predi
ctions 1.0. (default: 0.0)
tol: the convergence tolerance for iterative algorithms (>= 0). (d
efault: 1e-06)
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='LinearSVC_e7ca3d79b100', name='aggregationDepth', 
doc='suggested depth for treeAggregate (>= 2).'): 2, Param(parent=
'LinearSVC_e7ca3d79b100', name='featuresCol', doc='features column 
name.'): 'features', Param(parent='LinearSVC_e7ca3d79b100', name='
fitIntercept', doc='whether to fit an intercept term.'): True, Par
am(parent='LinearSVC_e7ca3d79b100', name='labelCol', doc='label co
lumn name.'): 'label', Param(parent='LinearSVC_e7ca3d79b100', name
='maxIter', doc='max number of iterations (>= 0).'): 100, Param(pa
rent='LinearSVC_e7ca3d79b100', name='predictionCol', doc='predicti
on column name.'): 'prediction', Param(parent='LinearSVC_e7ca3d79b
100', name='rawPredictionCol', doc='raw prediction (a.k.a. confide
nce) column name.'): 'rawPrediction', Param(parent='LinearSVC_e7ca
3d79b100', name='regParam', doc='regularization parameter (>= 0).'
): 0.1, Param(parent='LinearSVC_e7ca3d79b100', name='standardizati
on', doc='whether to standardize the training features before fitt
ing the model.'): True, Param(parent='LinearSVC_e7ca3d79b100', nam
e='threshold', doc='The threshold in binary classification applied 
to the linear model prediction.  This threshold can be any real nu
mber, where Inf will make all predictions 0.0 and -Inf will make a
ll predictions 1.0.'): 0.0, Param(parent='LinearSVC_e7ca3d79b100', 
name='tol', doc='the convergence tolerance for iterative algorithm
s (>= 0).'): 1e-06}
```

In [19]:

```
#test model
predictions = cvSVMModel.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|prediction|predictedLabel|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----
+---+----+------+-----+-------+--------------------+--------------
------+--------------------+----------+--------------+
| 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...|[2.30823258493557...|       0.0|             y|
| 35|  m|  0|     120| 198|  0|      1|    130|    1|    1.6|    1
|  0|   3|     n|  1.0|    0.0|[35.0,0.0,120.0,1...|[35.0,0.0,120.
0,1...|[-1.1022514592139...|       1.0|             n|
| 41|  f|  1|     130| 204|  0|      0|    172|    0|    1.4|    2
|  0|   2|     y|  0.0|    1.0|[41.0,1.0,130.0,2...|[41.0,1.0,130.
0,2...|[1.61240998576064...|       0.0|             y|
| 41|  m|  1|     110| 235|  0|      1|    153|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|[41.0,1.0,110.0,2...|[41.0,1.0,110.
0,2...|[1.35696388514107...|       0.0|             y|
| 41|  m|  1|     120| 157|  0|      1|    182|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|[41.0,1.0,120.0,1...|[41.0,1.0,120.
0,1...|[1.65615246968859...|       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...|[1.08371455597593...|       0.0|             y|
| 41|  m|  2|     112| 250|  0|      1|    179|    0|    0.0|    2
|  0|   2|     y|  0.0|    0.0|[41.0,2.0,112.0,2...|[41.0,2.0,112.
0,2...|[2.17222055617881...|       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...|[1.33684823504480...|       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...|[2.35318889127026...|       0.0|             y|
| 44|  m|  0|     110| 197|  0|      0|    177|    0|    0.0|    2
|  1|   2|     n|  1.0|    0.0|(13,[0,2,3,6,9,10...|(13,[0,2,3,6,9
,10...|[0.68520957487296...|       0.0|             y|
| 44|  m|  1|     130| 219|  0|      0|    188|    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,...|[1.53929202695927...|       0.0|             y|
| 45|  f|  0|     138| 236|  0|      0|    152|    1|    0.2|    1
|  0|   2|     y|  0.0|    1.0|[45.0,0.0,138.0,2...|[45.0,0.0,138.
0,2...|[0.35341041791256...|       0.0|             y|
| 45|  f|  1|     112| 160|  0|      1|    138|    0|    0.0|    1
|  0|   2|     y|  0.0|    1.0|[45.0,1.0,112.0,1...|[45.0,1.0,112.
0,1...|[1.51665765123809...|       0.0|             y|
| 45|  f|  1|     130| 234|  0|      0|    175|    0|    0.6|    1
|  0|   2|     y|  0.0|    1.0|[45.0,1.0,130.0,2...|[45.0,1.0,130.
```

```
0,2...|[1.60813546753669...|       0.0|             y|
| 45|  m|  3|     110| 264|  0|       1|    132|    0|    1.2|    1
|  0|    3|     n|  1.0|    0.0|[45.0,3.0,110.0,2...|[45.0,3.0,110.
0,2...|[1.03632897133690...|       0.0|             y|
| 46|  f|  2|     142| 177|  0|       0|    160|    1|    1.4|    0
|  0|    2|     y|  0.0|    1.0|[46.0,2.0,142.0,1...|[46.0,2.0,142.
0,1...|[0.81102055355660...|       0.0|             y|
| 47|  m|  0|     110| 275|  0|       0|    118|    1|    1.0|    1
|  1|    2|     n|  1.0|    0.0|[47.0,0.0,110.0,2...|[47.0,0.0,110.
0,2...|[-1.1325815773057...|       1.0|             n|
| 47|  m|  0|     112| 204|  0|       1|    143|    0|    0.1|    2
|  0|    2|     y|  0.0|    0.0|[47.0,0.0,112.0,2...|[47.0,0.0,112.
0,2...|[0.66439585440506...|       0.0|             y|
| 48|  f|  2|     130| 275|  0|       1|    139|    0|    0.2|    2
|  0|    2|     y|  0.0|    1.0|[48.0,2.0,130.0,2...|[48.0,2.0,130.
0,2...|[2.08972397268438...|       0.0|             y|
| 49|  f|  1|     134| 271|  0|       1|    162|    0|    0.0|    1
|  0|    2|     y|  0.0|    1.0|[49.0,1.0,134.0,2...|[49.0,1.0,134.
0,2...|[1.68369320955668...|       0.0|             y|
+---+---+---+--------+----+---+-------+-------+-----+-------+-----
+---+----+------+-----+-------+--------------------+--------------
------+--------------------+----------+--------------+
only showing top 20 rows
```

In [20]:

```
accuracy = evaluator.evaluate(predictions)
print("Test Error = " ,(1.0 - accuracy))
```

```
Test Error =  0.09717607973421916
```

In [21]:

```
#confusion matrix

predictionAndLabels = predictions.select("prediction", "label").rdd
metrics =  MulticlassMetrics(predictionAndLabels)
```

In [22]:

```
confusion = metrics.confusionMatrix()
print("Confusion matrix: \n" , confusion)
```

```
Confusion matrix:
 DenseMatrix([[40.,  3.],
             [ 8., 20.]])
```

In [23]:

```
##statistics per label

labels = predictionAndLabels.map(lambda x: x[1]).distinct().collect
print(labels)
for label in  labels:
  print("Class %f precision = %f\n" % (label , metrics.precision(la
  print("Class %f recall = %f\n" % (label, metrics.recall(label)))
  print("Class %f F1 score = %f\n" % (label, metrics.fMeasure( labe
```

```
[0.0, 1.0]
Class 0.000000 precision = 0.833333

Class 0.000000 recall = 0.930233

Class 0.000000 F1 score = 0.879121

Class 1.000000 precision = 0.869565

Class 1.000000 recall = 0.714286

Class 1.000000 F1 score = 0.784314
```

In [24]:

```
#weighted stats
print("Weighted precision = %s\n" % metrics.weightedPrecision)
print("Weighted recall = %s\n" % metrics.weightedRecall)
print("Weighted false positive rate = %s\n" % metrics.weightedFalse
```

```
Weighted precision = 0.8476219636660544

Weighted recall = 0.8450704225352113

Weighted false positive rate = 0.20055215010996208
```

In [25]:

```
#summary of stats
print(f"Recall = {metrics.recall(1.0)}")
print(f"Precision = {metrics.precision(1.0)}")
print(f"Accuracy = {metrics.accuracy}")
print(f"F1 = {metrics.fMeasure(1.0)}")
```

```
Recall = 0.7142857142857143
Precision = 0.8695652173913043
Accuracy = 0.8450704225352113
F1 = 0.7843137254901961
```

In [26]:

```
spark.stop()
```