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 sklearn.metrics import roc_curve, auc
from sklearn.tree import plot_tree
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("HeartDiseaseAnalDT")
       .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 to number values
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
assembler =  VectorAssembler(outputCol="features", inputCols=list(f
```

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]:
```
#Build decistion tree model
dt = DecisionTreeClassifier(labelCol="label", featuresCol="features
```

In [12]:
```
# build a parameter grid for different values
paramGrid = ParamGridBuilder().addGrid(dt.maxDepth, [ 5, 10 ]) \
                              .addGrid(dt.minInfoGain, [0.05, 0.025
                              .addGrid(dt.minInstancesPerNode, [5,
                              .addGrid(dt.maxBins, [5, 6, 9]) \
                              .build()
```

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

splits = df.randomSplit([0.6, 0.4 ], 3455)
train = splits[0]
test = splits[1]
```

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

In [15]:
```
#build evaluator
evaluator =  BinaryClassificationEvaluator(labelCol="label",rawPred
```

In [16]:
```
#Cross validator
cv = CrossValidator(estimator=pipeline, evaluator=evaluator,estimat
```

In [17]:
```
#train model
cvModel = cv.fit(train)
```

In [18]:
```
#Find out the best model
treeModel = cvModel.bestModel.stages[4]
print("Learned classification tree model:\n",treeModel)
print("Best Params: \n", treeModel.explainParams())
```

```
Learned classification tree model:
 DecisionTreeClassificationModel: uid=DecisionTreeClassifier_6d1df
8416897, depth=4, numNodes=13, numClasses=2, numFeatures=13
Best Params:
 cacheNodeIds: If false, the algorithm will pass trees to executor
s to match instances with nodes. If true, the algorithm will cache
node IDs for each instance. Caching can speed up training of deepe
r 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)
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)
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: 5)
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, current: 0.025)
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: 10)
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)
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: 52393225941365585)
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)
```

In [19]:
```
#show tree
plt.figure(figsize=(20,20))
features = df.columns

classes = ["No Disease","Diasease"]

#plot_tree(treeModel,feature_names=features,class_names=classes,fil
#plt.show()
```

```
<Figure size 1440x1440 with 0 Axes>
```

In [20]:
```
#test model
predictions = cvModel.transform(test)
predictions.select("prediction", "label", "target", "features").sho
```

```
+----------+-----+------+--------------------+
|prediction|label|target|            features|
+----------+-----+------+--------------------+
|       0.0|  0.0|     y|[34.0,1.0,118.0,2...|
|       0.0|  0.0|     y|(13,[0,1,2,3,6,9,...|
|       1.0|  1.0|     n|[39.0,0.0,118.0,2...|
|       0.0|  0.0|     y|(13,[0,1,2,3,6,9,...|
|       1.0|  1.0|     n|[40.0,0.0,110.0,1...|
|       0.0|  0.0|     y|[41.0,1.0,105.0,1...|
|       0.0|  0.0|     y|[41.0,1.0,130.0,2...|
|       0.0|  0.0|     y|[41.0,1.0,110.0,2...|
|       0.0|  0.0|     y|[41.0,2.0,112.0,2...|
|       1.0|  0.0|     y|[41.0,2.0,130.0,2...|
|       0.0|  0.0|     y|[42.0,0.0,102.0,2...|
|       0.0|  0.0|     y|[42.0,2.0,120.0,2...|
|       0.0|  1.0|     n|[42.0,0.0,136.0,3...|
|       0.0|  0.0|     y|(13,[0,2,3,5,6,9,...|
|       0.0|  0.0|     y|[42.0,2.0,130.0,1...|
|       0.0|  0.0|     y|[42.0,3.0,148.0,2...|
|       1.0|  1.0|     n|[43.0,0.0,132.0,3...|
|       0.0|  0.0|     y|[43.0,0.0,115.0,3...|
|       1.0|  1.0|     n|[43.0,0.0,120.0,1...|
|       1.0|  1.0|     n|(13,[0,2,3,6,9,10...|
+----------+-----+------+--------------------+
only showing top 20 rows
```

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

```
Test Error =  0.1752653301886793
```

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