Investigating Breast Tumor Data

Abstract

This study aims to answer the question "Can we use Machine Learning Models to accurately predict if a breast tumor mass is malignant or benign?", specifically using the "Breast Cancer Wisconsin (Diagnostic) Data Set" collected by Dr. William H. Wolberg.

Using Machine Learning Classification Models, such as Decision Tree and Random Forest, **four** models were created in order to predict whether or not a tumor is malignant or benign. Two different models were created for each type of classifier (Decision Tree vs Random Forest), the difference between the two models is the impurity measure used (Gini vs Entropy). The Impurity Measure is used to determine the splitting of the data.

All Machine Learning Models created had an accuracy score of over 94%, as well as very high AUC scores. Both of these mean that the models performed very well.

Motivation

“With 281,550 new cases of breast cancer estimated to be reported in 2021, Breast Cancer is the most commonly diagnosed cancer among American Women... Over 43,600 women are expected to die in 2021 from breast cancer”

U.S. Breast Cancer Statistics (breastcancer.org)

With the substantial amount of breast cancer in human lives today, it is the responsibility of our society that we find a way to mitigate the amount of death caused by breast cancer.

The early diagnosis of cancer is one of the only ways to have a better chance of having successful treatment. As such, it is imperative that we identify easier/faster/and more efficient ways to identify if a woman has a malignant breast cancer tumor.

Dataset

- Breast Cancer Wisconsin (Diagnostic) Data Set:
  - http://archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29
  - Created by Dr. William H. Wolberg, Physician at University of Wisconsin Hospital in Madison, Wisconsin.
  - Dataset was creating using a graphical computer program (Xcyt) to analyze fluid samples taken from patients with solid breast masses.
  - Using Xcyt, nuclei were isolated, then analyzed to find the mean, standard error, and worst-case for 10 features between all nuclei in a breast mass
- Features:
  - 2 categorical features:
    - Sample ID Number, Diagnosis (Benign or Malignant)
  - Mean, Standard Error, and Worst-Case were found for the 10 following features:
    - Radius, Texture, Perimeter, Area, Smoothness, Compactness, Concavity, Concave Points, Symmetry, Fractal Dimension
  - Total: 32 features, 569 observations

Data Preparation and Cleaning

- Checked for null values
  - No null values were found
- Structural Errors:
  - Importing the csv into pandas caused one observation to be used as the column names.
  - Added/Changed the column names
  - Added the observation used as column name to the dataframe.
  - 569 total observations.
- Check for Unwanted Outliers:
  - Used Box Plot to check for outliers in all 30 quantitative features, none were found.
- Removed Columns:
  - Removed the Sample ID Number column from the DataFrame
- Changed Structure:
  - Split DataFrame into two dataframes, one for input data (quantitative features), one for output data (Categorical features)
- Split the data into training and testing sets.
  - Used train_test_split library to split the data into X_train, y_train and X_test and y_test

Research Question

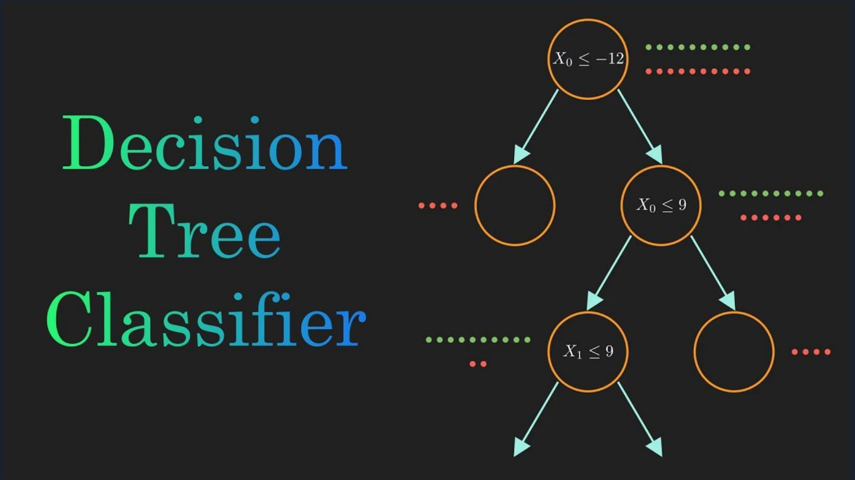
Today we'll be looking at this breast tumor data to do some investigation.

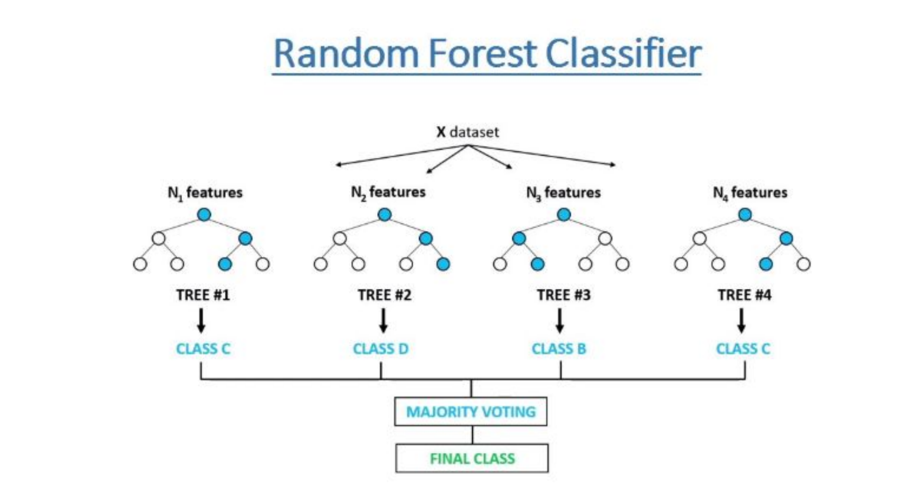
Research Question: Can we use Machine Learning Models to accurately predict if a breast mass is malignant or benign?

With breast cancer being the second most common cancer among women in the United States, it is imperative that we detect the cancer as soon as possible. Building tools that will correctly and efficiently classify and identify malignant tumors will help doctors identify breast cancer in its early stages.

Methods

- Based on the data and problem we wish to solve, this problem is a classification problem.
  - The data is labelled, so it makes sense to use a classification algorithm.
- Classification methods we will use are:



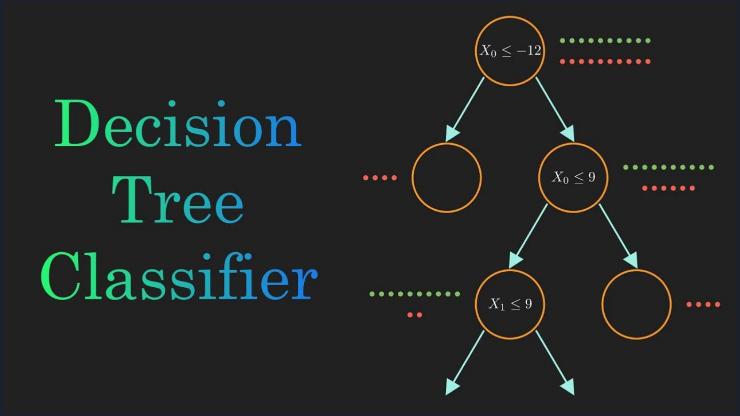


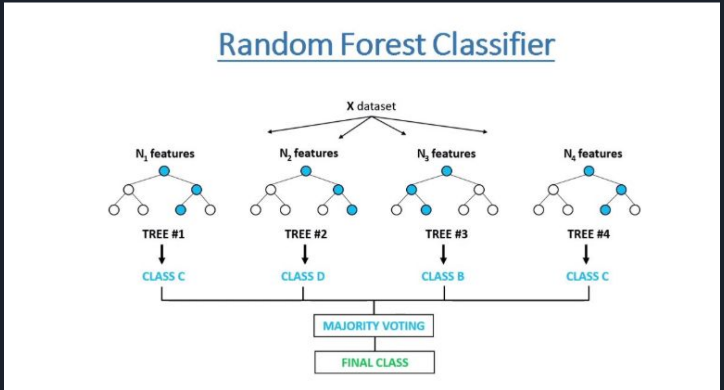
More info on these methods in the next slide.

Research Methods

Decision Tree Classifier:

- Uses a tree like structure to represent a decision path that leads to final classification at leaf nodes
- Based on input data, each decision will divide the input space into as pure as possible subgroups until classification is reached.
    - Example Decision: “If radius_mean is greater than X”

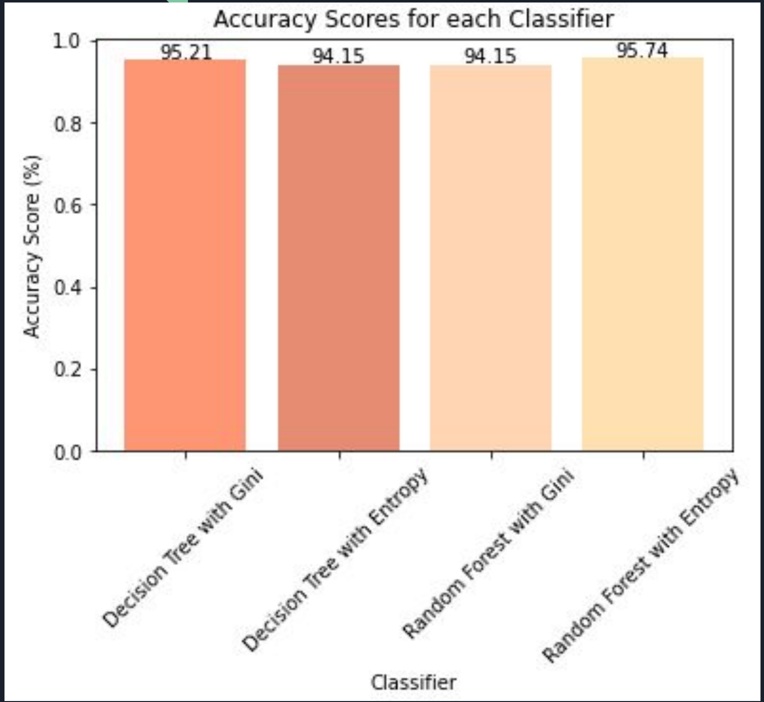




Random Forest Classifier:

- Similar to Decision Tree Classifier, but instead of building one tree, we build many trees
- After the model is built, when new data (new row) comes in, each trees will make an attempt on classification.
- Each tree gets a single “vote” as to what the class (malignant or benign) the data belongs to.
- The class that has the most votes, “wins” the classification

Findings

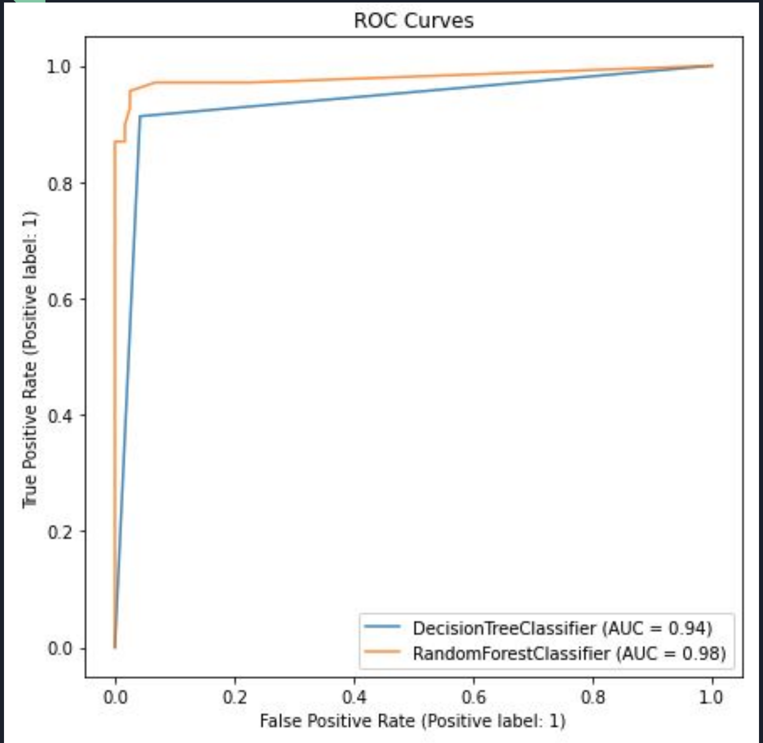


I decided to take my research one step further, and not only compare the two classification models, but also make changes to the 'criterion' parameter (Gini vs Entropy) and compare those models as well.

The 'criterion' parameter changes the impurity measure used when dividing the input space into subgroups in a classification problem. The impurity measure is used to measure the impurity of the new subgroups.

Measuring the accuracy scores of our models, we see that all models have over a **90% accuracy score**, with the Random Forest with Entropy Impurity Measure Model being the most accurate.

Findings (ROC Curves)



- Here we plot Receiver Operating Characteristic (ROC) curves for two of our created models: **Decision Tree with Entropy criterion, and Random Forest with Entropy Criterion**
- We can see from the curve, that the **false positive rate is low**, and the **true positive rate is high**, which means our models are performing well.
- Area Under the Curve (AUC) is another measure of model performance, which also has very high scores.
- Our Random Forest Classifier has a slightly higher AUC, which means it performs slightly better than our Decision Tree Classifier.

Limitations

Before this classifier is used in the public domain, there are some limitations that need to be addressed:

- Data was collected specifically using a Computer Vision Program called *Xcyt*. To use the classifier accurately, data should be collected using the similar, if not the same, program.

- Use of *Xcyt* requires user input, and thus user training.
  - Users must use the mouse pointer to draw approximate boundaries of cell nuclei, this may also bring user error into the data collection phase.

- Training set is only 569 observations, we would want to have more observations if we want our classifier to be more accurate.

Conclusions

Research Question: Can we use Machine Learning Models to accurately predict if a breast mass is malignant or benign?

Answer: Using Data Science techniques and principles, we can confidently say that the answer to our research question is “Yes, we can use Machine Learning Models to accurately predict if a breast mass is malignant or benign”.

All of the models generated in this study had an accuracy score of greater than 90%.

Acknowledgements

I’d like to thank the following people:

- My parents, for bringing me into a place where I can seek endless knowledge on the subjects I care about.
- My girlfriend, for the constant love and support and for constructive criticism on this presentation.
- The instructors for our course, “Python for Data Science” for giving us the skills needed to conduct interesting research!

References

DataSet Citation:

Wolberg, William H, et al. “Wisconsin Diagnostic Breast Cancer.” *UCI Machine Learning Repository*, 1 Nov. 1995, archive.ics.uci.edu/ml/datasets/breast+cancer+wisconsin+%28diagnostic%29.

The DataSet summary greatly helped me in understanding the data and what could be done.

Final Project Notebook

June 27, 2021

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import plot_roc_curve

breastCancerData = pd.read_csv('./DataSets/breastcancerwisconsin/
↪breast-cancer-wisconsin.data', na_values=["?"])
wdbc = pd.read_csv('./DataSets/breastcancerwisconsin/wdbc.data')
```

0.0.1 Research Question: Can we Classification ML Models to accurately predict if a rbeast cancer tumor is malignant or benign?

Data Science Steps: 1) Acquire 2) Prepare 3) Analyze 4) Report 5) Act

We will use classification ML model(s) to determine whether or not a tumor is malignant (cancerous) or benign (non-cancerous)

https://towardsdatascience.com/building-a-simple-machine-learning-model-on-breast-cancer-data-eca4b3b99fa3

DataSets Used: * UCI Machine Learning Repository for breast cancer dataset. * wdbc: * Created by Dr. Wolberg, physician at the University of Wisconsin Hospital in Madison, Wisconsin. * Dataset was created by using fluid samples taken from patients with solid breast masses, also used graphical computer program to perform analysis of cytological features based on a digital scan. * Program called Xcyt was used to calculate feature values of all nuclei within the tumor. Mean, standard error, and extreme values of these features are computed, resulting in 30 nuclear features for each sample * breast-cancer-wisconsin * Also created by Dr. William H Wolberg

WDBC feature list: * ID * Diagnosis * Mean, stderr, and max extreme was retrieved for the following features: * Radius * Texture * Perimeter * Area * Smoothness * Compactness * Concavity * Concave Points * Symmetry * Fractal Dimension * NOTE: Extreme was calcculated by the mean of the three largest values * Total: 32 Features

Breast-cancer-wisconsin feature list: * Sample Code Number * Class: (2 for benign, 4 for malignant) * The following features have been scaled 1-10 for intensity * Clump Thickness * Uniformity of Cell Size * Uniformity of Cell Shape * Marginal Adhesion * Single Epithelial Cell

Size * Bare Nuclei * Bland Chromatin * Normal Nucleoli * Mitoses * Total: 11 Features * Can we use this data? Not sure.

0.0.2 Classification Algorithms:

- NOTE: All ML Classification is supervised
- kNN
- Decision Trees <–
  - Greedy Induction Algorithm
  - Feature Scaling must be used if ML algorithms use Eucledian Distance in their computations, do we need to do this?
- Naive Bayes

0.0.3 Things I've done to the data:

breastCancerData: * Added column names to the breast-cancer-wisconsin.data dataframe * Fixed the dataframe to include the one row that was read as column names initially, breastCancerData should have a total of 699 rows before removing anything * Removed null values from the breast-cancer-wisconsin.data dataframe

wdbc: * Added column names * Fixed the dataframe to include the one row that was read as column names initially, there should be a total of 569 rows * Deleted "ID number" column since we won't need it

1 WDBC SECTION

1.0.1 Data Cleaning

[2]:
```
#Adding Columns
wdbc.columns = ['ID number', 'Diagnosis', 'radius_mean', 'texture_mean',␣
 ↪'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean',␣
 ↪'concavity_mean', 'concave_points_mean', 'symmetry_mean',␣
 ↪'fractal_dimension_mean', 'radius_se', 'texture_se', 'perimeter_se',␣
 ↪'area_se', 'smoothness_se', 'compactness_se', 'concavity_se',␣
 ↪'concave_points_se', 'symmetry_se', 'fractal_dimension_se', 'radius_worst',␣
 ↪'texture_worst', 'perimeter_worst', 'area_worst', 'smoothness_worst',␣
 ↪'compactness_worst', 'concavity_worst', 'concave_points_worst',␣
 ↪'symmetry_worst', 'fractal_dimension_worst']
```

[3]:
```
wdbc.loc[0]
```

[3]:
```
ID number                    842517
Diagnosis                         M
radius_mean                   20.57
texture_mean                  17.77
perimeter_mean                132.9
area_mean                      1326
smoothness_mean             0.08474
compactness_mean            0.07864
```

```
concavity_mean              0.0869
concave_points_mean        0.07017
symmetry_mean               0.1812
fractal_dimension_mean     0.05667
radius_se                   0.5435
texture_se                  0.7339
perimeter_se                 3.398
area_se                      74.08
smoothness_se             0.005225
compactness_se             0.01308
concavity_se                0.0186
concave_points_se           0.0134
symmetry_se                0.01389
fractal_dimension_se      0.003532
radius_worst                 24.99
texture_worst                23.41
perimeter_worst              158.8
area_worst                    1956
smoothness_worst            0.1238
compactness_worst           0.1866
concavity_worst             0.2416
concave_points_worst         0.186
symmetry_worst               0.275
fractal_dimension_worst    0.08902
Name: 0, dtype: object
```

[4]:
```
#Adding top column
wdbc.loc[len(wdbc.index)] = [842302, 'M', 17.99, 10.38, 122.8, 1001, 011.84, 0.
 ↪2776, 0.3001, 0.1471, 0.2419, 0.07871, 1.095, 0.9053, 8.589, 153.4, 0.
 ↪006399, 0.04904, 0.05373, 0.01587, 0.03003, 0.006193, 25.38, 17.33, 184.6,␣
 ↪2019, 0.1622, 0.6656, 0.7119, 0.2654, 0.4601, 0.1189]
```

[5]:
```
wdbcCopy = wdbc.copy()
```

[6]:
```
#Deleted ID number column since we won't need it
del wdbcCopy['ID number']
```

[7]:
```
wdbc.info()
#See that there are no nulls
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 569 entries, 0 to 568
Data columns (total 32 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   ID number                 569 non-null    int64
 1   Diagnosis                 569 non-null    object
 2   radius_mean               569 non-null    float64
```

```
 3   texture_mean             569 non-null    float64
 4   perimeter_mean           569 non-null    float64
 5   area_mean                569 non-null    float64
 6   smoothness_mean          569 non-null    float64
 7   compactness_mean         569 non-null    float64
 8   concavity_mean           569 non-null    float64
 9   concave_points_mean      569 non-null    float64
 10  symmetry_mean            569 non-null    float64
 11  fractal_dimension_mean   569 non-null    float64
 12  radius_se                569 non-null    float64
 13  texture_se               569 non-null    float64
 14  perimeter_se             569 non-null    float64
 15  area_se                  569 non-null    float64
 16  smoothness_se            569 non-null    float64
 17  compactness_se           569 non-null    float64
 18  concavity_se             569 non-null    float64
 19  concave_points_se        569 non-null    float64
 20  symmetry_se              569 non-null    float64
 21  fractal_dimension_se     569 non-null    float64
 22  radius_worst             569 non-null    float64
 23  texture_worst            569 non-null    float64
 24  perimeter_worst          569 non-null    float64
 25  area_worst               569 non-null    float64
 26  smoothness_worst         569 non-null    float64
 27  compactness_worst        569 non-null    float64
 28  concavity_worst          569 non-null    float64
 29  concave_points_worst     569 non-null    float64
 30  symmetry_worst           569 non-null    float64
 31  fractal_dimension_worst  569 non-null    float64
dtypes: float64(30), int64(1), object(1)
memory usage: 146.7+ KB
```

1.0.2 Data Exploration

[8]:
```
wdbcCopy.head()
```

[8]:
```
  Diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
0         M        20.57         17.77          132.90     1326.0
1         M        19.69         21.25          130.00     1203.0
2         M        11.42         20.38           77.58      386.1
3         M        20.29         14.34          135.10     1297.0
4         M        12.45         15.70           82.57      477.1

   smoothness_mean  compactness_mean  concavity_mean  concave_points_mean  \
0          0.08474           0.07864          0.0869              0.07017
1          0.10960           0.15990          0.1974              0.12790
2          0.14250           0.28390          0.2414              0.10520
```

```
3          0.10030            0.13280          0.1980                0.10430
4          0.12780            0.17000          0.1578                0.08089

   symmetry_mean  …  radius_worst  texture_worst  perimeter_worst  \
0         0.1812  …         24.99          23.41           158.80
1         0.2069  …         23.57          25.53           152.50
2         0.2597  …         14.91          26.50            98.87
3         0.1809  …         22.54          16.67           152.20
4         0.2087  …         15.47          23.75           103.40

   area_worst  smoothness_worst  compactness_worst  concavity_worst  \
0      1956.0            0.1238             0.1866           0.2416
1      1709.0            0.1444             0.4245           0.4504
2       567.7            0.2098             0.8663           0.6869
3      1575.0            0.1374             0.2050           0.4000
4       741.6            0.1791             0.5249           0.5355

   concave_points_worst  symmetry_worst  fractal_dimension_worst
0                0.1860          0.2750                  0.08902
1                0.2430          0.3613                  0.08758
2                0.2575          0.6638                  0.17300
3                0.1625          0.2364                  0.07678
4                0.1741          0.3985                  0.12440

[5 rows x 31 columns]
```

[9]:
```
wdbcCopy.shape
```

[9]: (569, 31)

[10]:
```
#Number of Malignant Tumors in our DataSet
len(wdbcCopy[wdbcCopy['Diagnosis'] == 'M'].index)
```

[10]: 212

[11]:
```
#Number of Benign Tumors in our DataSet
len(wdbcCopy[wdbcCopy['Diagnosis'] == 'B'].index)
```

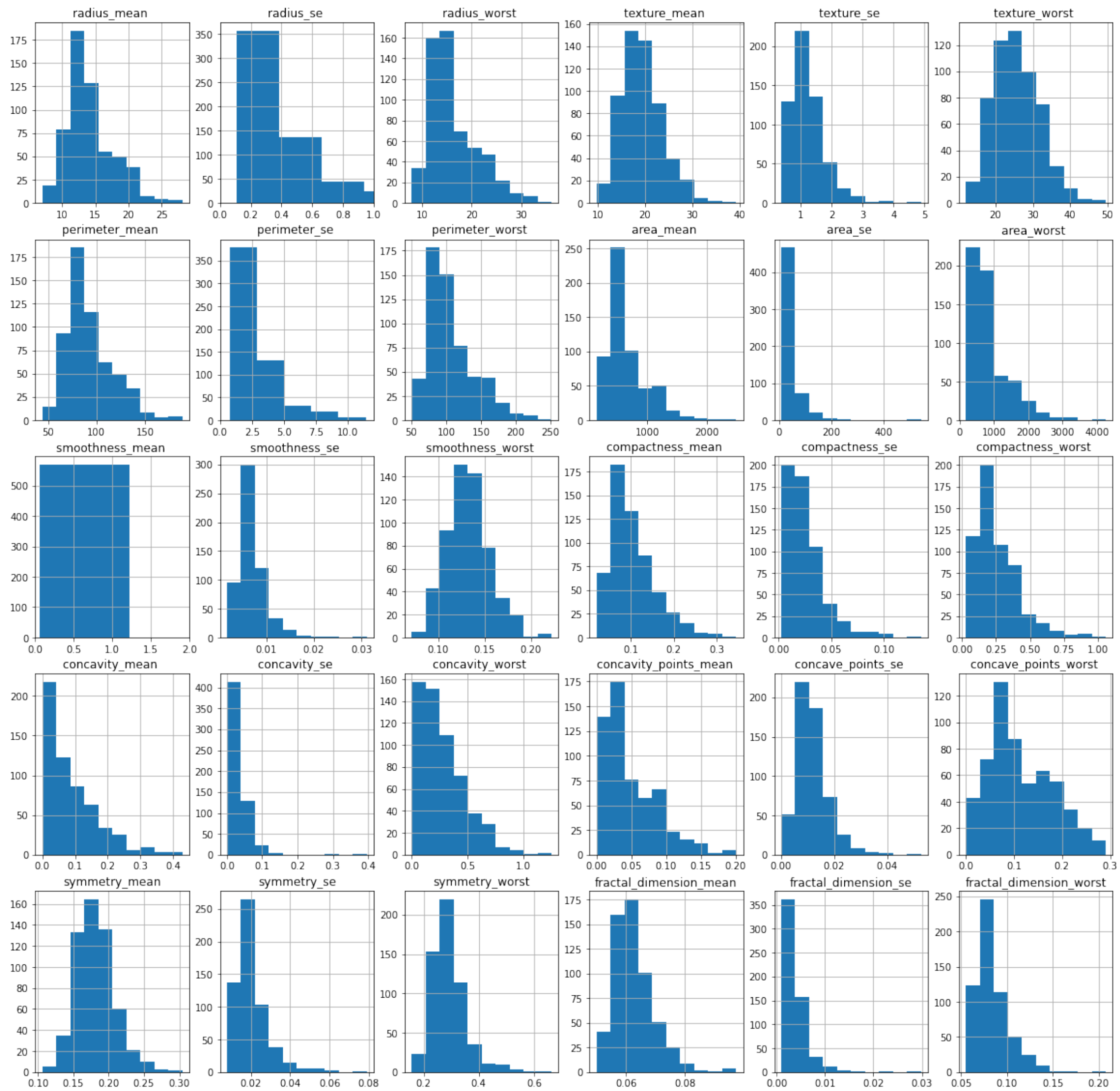
[11]: 357

[12]:
```
fig, axs = plt.subplots(5, 6, figsize=(20,20))
axs[0, 0].hist(wdbcCopy['radius_mean'])
axs[0, 0].grid()
axs[0, 0].title.set_text('radius_mean ')

axs[0, 1].hist(wdbcCopy['radius_se'])
axs[0, 1].grid()
axs[0, 1].title.set_text('radius_se ')
```

```
axs[0, 1].set_xlim(0, 1)

axs[0, 2].hist(wdbcCopy['radius_worst'])
axs[0, 2].grid()
axs[0, 2].title.set_text('radius_worst ')

axs[0, 3].hist(wdbcCopy['texture_mean'])
axs[0, 3].grid()
axs[0, 3].title.set_text('texture_mean ')

axs[0, 4].hist(wdbcCopy['texture_se'])
axs[0, 4].grid()
axs[0, 4].title.set_text('texture_se ')

axs[0, 5].hist(wdbcCopy['texture_worst'])
axs[0, 5].grid()
axs[0, 5].title.set_text('texture_worst ')

axs[1, 0].hist(wdbcCopy['perimeter_mean'])
axs[1, 0].grid()
axs[1, 0].title.set_text('perimeter_mean ')

axs[1, 1].hist(wdbcCopy['perimeter_se'])
axs[1, 1].grid()
axs[1, 1].title.set_text('perimeter_se ')
axs[1, 1].set_xlim(0, 12)

axs[1, 2].hist(wdbcCopy['perimeter_worst'])
axs[1, 2].grid()
axs[1, 2].title.set_text('perimeter_worst ')

axs[1, 3].hist(wdbcCopy['area_mean'])
axs[1, 3].grid()
axs[1, 3].title.set_text('area_mean ')

axs[1, 4].hist(wdbcCopy['area_se'])
axs[1, 4].grid()
axs[1, 4].title.set_text('area_se ')

axs[1, 5].hist(wdbcCopy['area_worst'])
axs[1, 5].grid()
axs[1, 5].title.set_text('area_worst ')

axs[2, 0].hist(wdbcCopy['smoothness_mean'])
axs[2, 0].grid()
axs[2, 0].title.set_text('smoothness_mean ')
axs[2, 0].set_xlim(0, 2)
```

```
axs[2, 1].hist(wdbcCopy['smoothness_se'])
axs[2, 1].grid()
axs[2, 1].title.set_text('smoothness_se ')

axs[2, 2].hist(wdbcCopy['smoothness_worst'])
axs[2, 2].grid()
axs[2, 2].title.set_text('smoothness_worst ')

axs[2, 3].hist(wdbcCopy['compactness_mean'])
axs[2, 3].grid()
axs[2, 3].title.set_text('compactness_mean ')

axs[2, 4].hist(wdbcCopy['compactness_se'])
axs[2, 4].grid()
axs[2, 4].title.set_text('compactness_se ')

axs[2, 5].hist(wdbcCopy['compactness_worst'])
axs[2, 5].grid()
axs[2, 5].title.set_text('compactness_worst ')

axs[3, 0].hist(wdbcCopy['concavity_mean'])
axs[3, 0].grid()
axs[3, 0].title.set_text('concavity_mean ')

axs[3, 1].hist(wdbcCopy['concavity_se'])
axs[3, 1].grid()
axs[3, 1].title.set_text('concavity_se ')

axs[3, 2].hist(wdbcCopy['concavity_worst'])
axs[3, 2].grid()
axs[3, 2].title.set_text('concavity_worst ')

axs[3, 3].hist(wdbcCopy['concave_points_mean'])
axs[3, 3].grid()
axs[3, 3].title.set_text('concavity_points_mean ')

axs[3, 4].hist(wdbcCopy['concave_points_se'])
axs[3, 4].grid()
axs[3, 4].title.set_text('concave_points_se ')

axs[3, 5].hist(wdbcCopy['concave_points_worst'])
axs[3, 5].grid()
axs[3, 5].title.set_text('concave_points_worst ')

axs[4, 0].hist(wdbcCopy['symmetry_mean'])
axs[4, 0].grid()
```

```
axs[4, 0].title.set_text('symmetry_mean')

axs[4, 1].hist(wdbcCopy['symmetry_se'])
axs[4, 1].grid()
axs[4, 1].title.set_text('symmetry_se')

axs[4, 2].hist(wdbcCopy['symmetry_worst'])
axs[4, 2].grid()
axs[4, 2].title.set_text('symmetry_worst')

axs[4, 3].hist(wdbcCopy['fractal_dimension_mean'])
axs[4, 3].grid()
axs[4, 3].title.set_text('fractal_dimension_mean')

axs[4, 4].hist(wdbcCopy['fractal_dimension_se'])
axs[4, 4].grid()
axs[4, 4].title.set_text('fractal_dimension_se')

axs[4, 5].hist(wdbcCopy['fractal_dimension_worst'])
axs[4, 5].grid()
axs[4, 5].title.set_text('fractal_dimension_worst')

plt.grid(True)
plt.show()
```



[13]:
```
#Converting into a classification task
copiedData = wdbcCopy.copy()
copiedData['diagnosis_label'] = (copiedData['Diagnosis'] == 'M') * 1
```

[14]:
```
#copiedData.head()
copiedData[copiedData['Diagnosis'] == 'B']
```

[14]:
```
   Diagnosis  radius_mean  texture_mean  perimeter_mean  area_mean  \
18         B       13.540         14.36           87.46      566.3
19         B       13.080         15.71           85.63      520.0
20         B        9.504         12.44           60.34      273.9
36         B       13.030         18.42           82.61      523.8
45         B        8.196         16.84           51.71      201.9
```

```
..        …        …          …           …       …
557       B   14.590      22.68       96.39   657.1
558       B   11.510      23.93       74.52   403.5
559       B   14.050      27.15       91.38   600.4
560       B   11.200      29.37       70.67   386.0
567       B    7.760      24.54       47.92   181.0

     smoothness_mean  compactness_mean  concavity_mean  concave_points_mean  \
18           0.09779           0.08129         0.06664             0.047810
19           0.10750           0.12700         0.04568             0.031100
20           0.10240           0.06492         0.02956             0.020760
36           0.08983           0.03766         0.02562             0.029230
45           0.08600           0.05943         0.01588             0.005917
..               …                 …               …                    …
557          0.08473           0.13300         0.10290             0.037360
558          0.09261           0.10210         0.11120             0.041050
559          0.09929           0.11260         0.04462             0.043040
560          0.07449           0.03558         0.00000             0.000000
567          0.05263           0.04362         0.00000             0.000000

     symmetry_mean  …  texture_worst  perimeter_worst  area_worst  \
18          0.1885  …          19.26            99.70       711.2
19          0.1967  …          20.49            96.09       630.5
20          0.1815  …          15.66            65.13       314.9
36          0.1467  …          22.81            84.46       545.9
45          0.1769  …          21.96            57.26       242.2
..             …  …            …               …          …
557         0.1454  …          27.27           105.90       733.5
558         0.1388  …          37.16            82.28       474.2
559         0.1537  …          33.17           100.20       706.7
560         0.1060  …          38.30            75.19       439.6
567         0.1587  …          30.37            59.16       268.6

     smoothness_worst  compactness_worst  concavity_worst  \
18            0.14400            0.17730          0.23900
19            0.13120            0.27760          0.18900
20            0.13240            0.11480          0.08867
36            0.09701            0.04619          0.04833
45            0.12970            0.13570          0.06880
..                …                  …                …
557           0.10260            0.31710          0.36620
558           0.12980            0.25170          0.36300
559           0.12410            0.22640          0.13260
560           0.09267            0.05494          0.00000
567           0.08996            0.06444          0.00000

     concave_points_worst  symmetry_worst  fractal_dimension_worst  \
```

```
18               0.12880        0.2977              0.07259
19               0.07283        0.3184              0.08183
20               0.06227        0.2450              0.07773
36               0.05013        0.1987              0.06169
45               0.02564        0.3105              0.07409
..                   …             …                 …
557              0.11050        0.2258              0.08004
558              0.09653        0.2112              0.08732
559              0.10480        0.2250              0.08321
560              0.00000        0.1566              0.05905
567              0.00000        0.2871              0.07039

     diagnosis_label
18                 0
19                 0
20                 0
36                 0
45                 0
..               …
557                0
558                0
559                0
560                0
567                0

[357 rows x 32 columns]
```

[15]:
```
y = copiedData[['diagnosis_label']].copy()
```

[16]:
```
y.head()
```

[16]:
```
   diagnosis_label
0                1
1                1
2                1
3                1
4                1
```

[17]:
```
copiedData['Diagnosis'].head()
```

[17]:
```
0    M
1    M
2    M
3    M
4    M
Name: Diagnosis, dtype: object
```

[18]:
```
X = copiedData.copy()
del X['Diagnosis']
del X['diagnosis_label']
```

[19]:
```
X.columns
```

[19]:
```
Index(['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean',
       'smoothness_mean', 'compactness_mean', 'concavity_mean',
       'concave_points_mean', 'symmetry_mean', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimeter_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'concave_points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_worst', 'texture_worst',
       'perimeter_worst', 'area_worst', 'smoothness_worst',
       'compactness_worst', 'concavity_worst', 'concave_points_worst',
       'symmetry_worst', 'fractal_dimension_worst'],
      dtype='object')
```

[20]:
```
y.columns
```

[20]:
```
Index(['diagnosis_label'], dtype='object')
```

Now we have our y (our label/output), and x (our data/input)

1.0.3 Decision Tree Classifier with Gini Impurity Measure

[21]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
↪random_state=69)
```

[22]:
```
accuracyScores = []
accuracyScoreLabels = ['Decision Tree with Gini', 'Decision Tree with Entropy',␣
↪'Random Forest with Gini', 'Random Forest with Entropy']
```

[23]:
```
#Basic DecisionTreeClassifier, criterion = gini
giniDecisionTree = DecisionTreeClassifier(random_state=69)
giniDecisionTree.fit(X_train, y_train)
```

[23]:
```
DecisionTreeClassifier(random_state=69)
```

[24]:
```
#Testing our model
giniDecisionTreePredictions = giniDecisionTree.predict(X_test)
```

[25]:
```
accuracyScores.append(accuracy_score(y_true = y_test, y_pred =␣
↪giniDecisionTreePredictions))
```

1.0.4 Decision Tree Classifier with Entropy Impurity Measure

[26]:
```
#entropy DecisionTree
entropyDecisionTree = DecisionTreeClassifier(criterion = 'entropy',␣
↪random_state=69)
entropyDecisionTree.fit(X_train, y_train)
```

[26]:
```
DecisionTreeClassifier(criterion='entropy', random_state=69)
```

[27]:
```
entropyDecisionTreePredictions = entropyDecisionTree.predict(X_test)
```

[28]:
```
accuracyScores.append(accuracy_score(y_true = y_test, y_pred =␣
↪entropyDecisionTreePredictions))
```

1.0.5 Random Forest Classifier with Gini Impurity Measure

[29]:
```
giniRandomForest = RandomForestClassifier(criterion='gini', random_state=69,␣
↪n_estimators=10)
giniRandomForest.fit(X_train, y_train['diagnosis_label'])
```

[29]:
```
RandomForestClassifier(n_estimators=10, random_state=69)
```

[30]:
```
giniRandomForestPredictions = giniRandomForest.predict(X_test)
```

[31]:
```
giniRandomForestPredictions[:10]
```

[31]:
```
array([0, 1, 1, 1, 0, 1, 0, 0, 0, 1])
```

[32]:
```
accuracyScores.append(accuracy_score(y_true=y_test,␣
↪y_pred=giniRandomForestPredictions))
```

1.0.6 Random Forest Classifier with Entropy Impurity Measure

[33]:
```
entropyRandomForest = RandomForestClassifier(criterion='entropy',␣
↪random_state=69, n_estimators=10)
entropyRandomForest.fit(X_train, y_train['diagnosis_label'])
```

[33]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10, random_state=69)
```

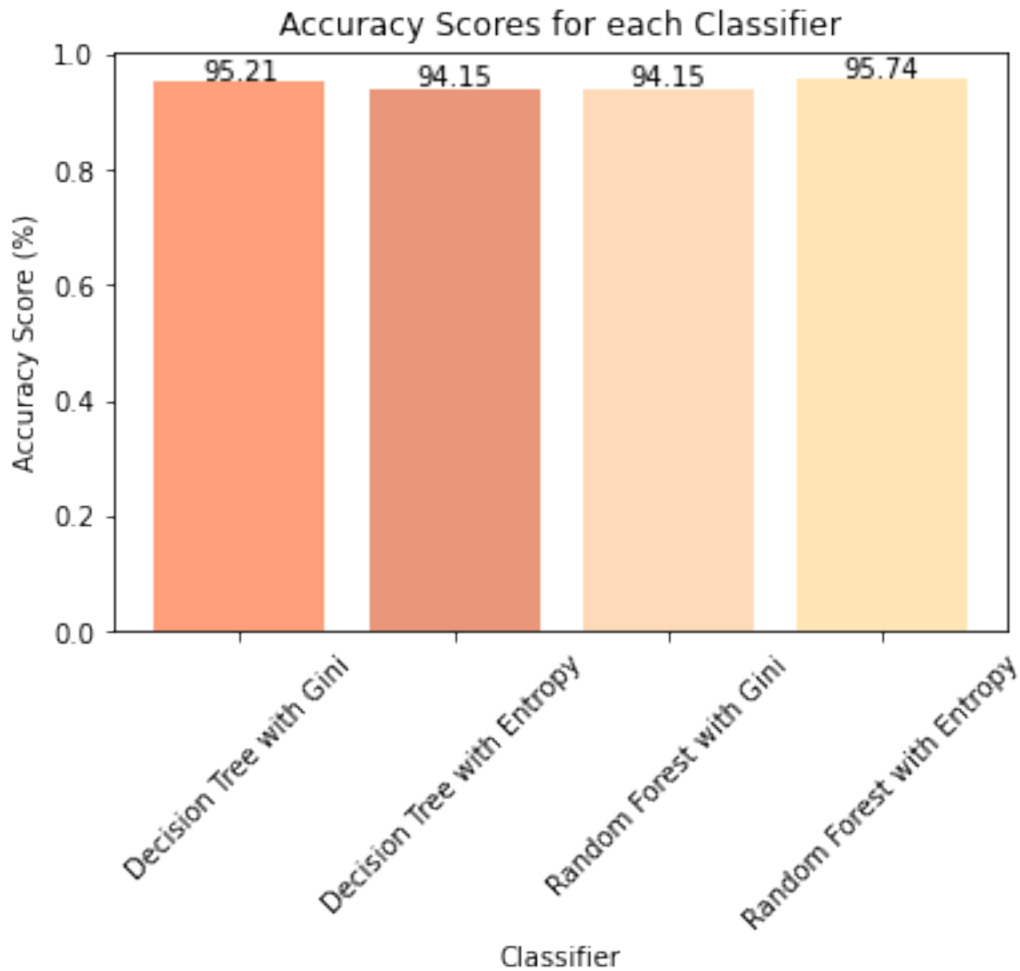
[34]:
```
entropyRandomForestPredictions = entropyRandomForest.predict(X_test)
```

[35]:
```
entropyRandomForestPredictions.ptp()
```

[35]:
```
1
```

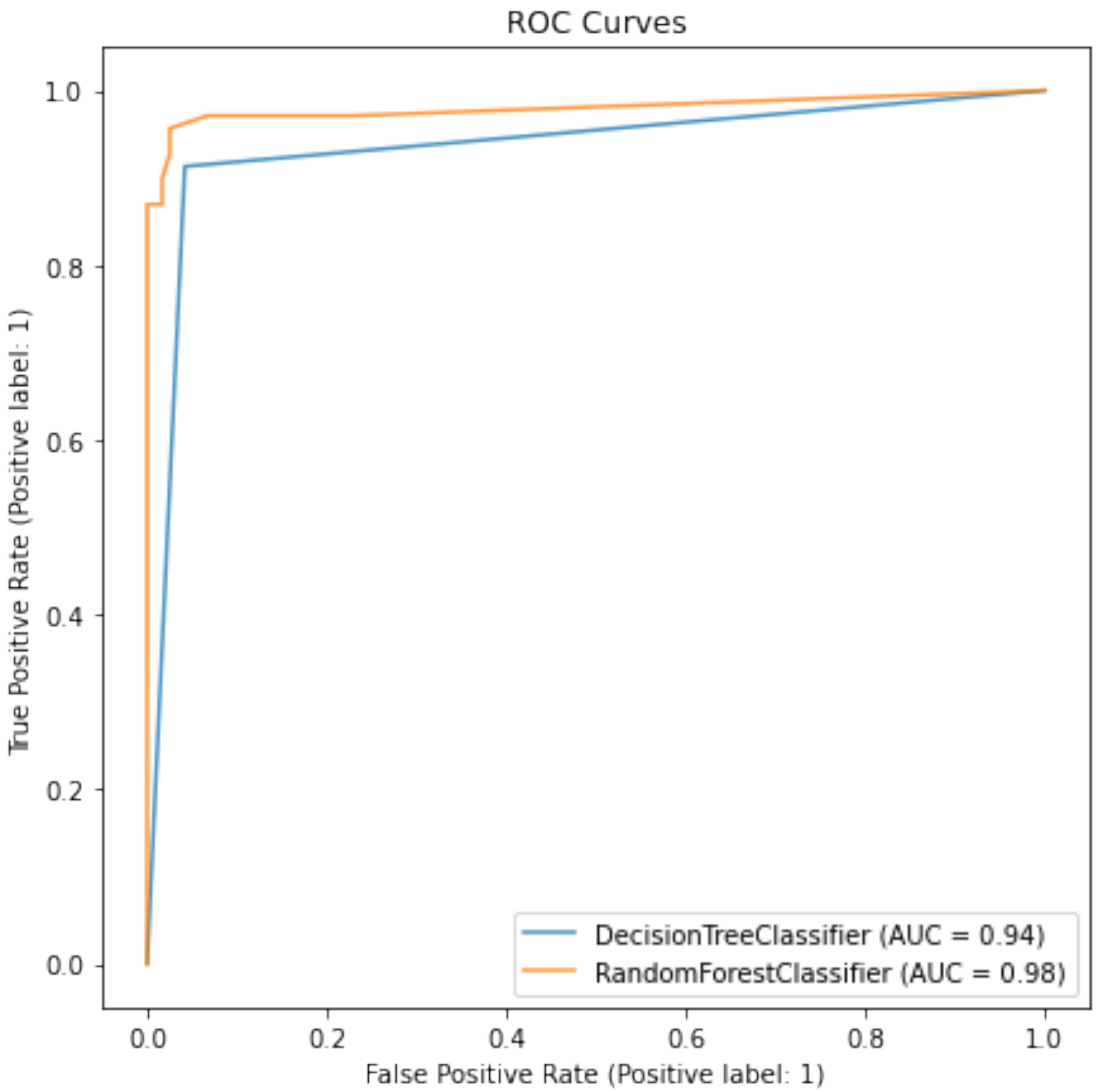
[36]:
```
accuracyScores.append(accuracy_score(y_true=y_test,␣
↪y_pred=entropyRandomForestPredictions))
```

[37]:
```
fig, ax = plt.subplots()

ax.bar(accuracyScoreLabels, accuracyScores, color = ['lightsalmon',␣
↪'darksalmon', 'peachpuff', 'moccasin'])
ax.set_title('Accuracy Scores for each Classifier')
ax.set_xlabel('Classifier')
ax.set_ylabel('Accuracy Score (%)')
plt.xticks(rotation=45)

xlocs, xlabs = plt.xticks()
for index, value in enumerate(accuracyScores):
    plt.text(xlocs[index] - 0.18, value + 0.0035, str(round(value*100, 2)),␣
↪color='black')

plt.rcParams['text.color'] = 'black'
plt.rcParams['axes.labelcolor'] = 'black'
plt.rcParams['xtick.color'] = 'black'
plt.rcParams['ytick.color'] = 'black'
plt.show()
```



1.0.7 ROC Curve to Visualize Model Performance

[46]:
```
fig, ax = plt.subplots(figsize=(7, 7))
entropyDecisionTree_disp = plot_roc_curve(entropyDecisionTree, X_test, y_test,␣
 ↪ax=ax,  alpha=0.8)
#giniDecisionTree_disp.plot(ax=ax, alpha=0.8)
giniRandomForest_disp = plot_roc_curve(giniRandomForest, X_test, y_test, ax=ax,␣
 ↪alpha=0.8)
plt.title("ROC Curves")
plt.show()
#entropyRandomForest_disp = plot_roc_curve(entropyRandomForest, X_test, y_test,␣
 ↪ax=ax, alpha=0.8, pos_label=1)
```


ROC Curves
True Positive Rate (Positive label: 1)
False Positive Rate (Positive label: 1)
0.0
0.2
0.4
0.6
0.8
1.0
DecisionTreeClassifier (AUC = 0.94)
RandomForestClassifier (AUC = 0.98)