Tutorial 5 - Natasha Ng, Lucia Kim, Jeffrey Smith, James Tong

1 Theoretical Questions

1. What are the two key factors an ensemble must have? - James
One important key factor is diversity within the ensemble, if many of the individual models are highly correlated or identical to each other, the ensemble method won't be as effective. This is essential to have as it allows the ensemble to capture different patterns in the data. Another key factor important to have is the individual accuracy of the base models. If many of the models have low accuracy, regardless of how diverse the ensemble is, the end result won't be as reliable. It's essential that each individual model makes meaningful contributions such that the ensemble can give better predictions.

2. Bagging changes two things in a dataset. What are they? - James
The process of bootstrap sampling in bagging ensembles helps bring diversity to the dataset. With random sampling and replacement done for each subset, this allows the possibility of duplicates and omissions of some samples. Another change will be uneven weightings of instances between subsets. Because some samples will be shown more frequently than others or omitted entirely, the training data could be more skewed for certain instances, influencing the learning behavior of the model.

3. What are the main differences between random forest (RF), bagging, and XGBoost? - James
The main difference between Random Forests and other ensemble methods is that they can only be used with decision tree classifiers, whereas bagging and XGBoost can be used with a diverse range of algorithms. The XGBoost ensemble method is unique as it builds a collection of weak learners sequentially, with every iteration improving upon the weaknesses of the one before. Lastly, Bagging involves training the same base classifier on different subsets of the dataset. This helps reduce variance by averaging the predictions of the individual models. Hence, it produces more stable and robust predictions in the final result.

4. Which of the “methods for constructing ensembles” do RF and XGBoost use? -Lucia
RF uses the averaging method for constructing ensembles. RF aims to reduce variance by finding the average of very deep decision trees. The randomness of RF is introduced through both the rows and the columns of the data. Each tree trains on a bootstrapped subset of the samples, and each split selects a feature from a subset of the features remaining. As the trees are randomly and independently generated, the average of the prediction reduces overfitting,

XGBoost uses the boosting method. XGBoost aims to reduce bias by reducing the errors of the previous model. XGBoost uses regularised regression trees as the base classifier. Each tree in the XGBoost ensemble tries to decrease the training error by predicting the residuals of the previous tree. The final prediction is the sum of the individual predictions.

5. Will variable importance in RF always give you the “correct” answer? Why? - Lucia

Variable importance in RF can be used to identify which features were important for the model. They do not explain the dependency between the features and the target variable. It cannot be used to determine how much the feature is able to predict the target in general. The variable importance in RF can be incorrect when many of the variables in the dataset are correlated. This means that the variables can be used interchangeably when selecting the next split. High correlation in the features masks and merges the true importance of the individual feature. The model using the best variable importance may not be the most accurate model.

6. Between RF and bagging, what will the effect be of having a data set with a larger or smaller number of instances? - Natasha
In both Random Forest and Bagging, having a larger number of instances in the data set generally leads to better performance as more data usually means a better representation of the underlying distribution in the data. With Random Forest, having more data means it can potentially create more decision trees, leading to a more diverse set of trees that improves the model's performance. With Bagging, it can create more diverse bootstrap samples, leading to a more diverse set of base learners that improves the performance of the ensemble. However, Bagging may be more sensitive to the size of the data set because it relies on bootstrapping and sampling with replacement. If the data set is small, the chances of repeated instances in the bootstrap sample is higher, leading to less diverse base learners and poorer model performance.

7. How does the number of features (large versus small) affect the performance of the random forest and bagging? - Natasha
For random forests, a large number of features will not affect the performance too much as it randomly selects a subset of features at each split in each tree. This feature sampling means that each tree is independent and prevents them from being too similar. However, if there are many irrelevant features in the dataset, random forests may not be able to identify the most important features, which can lead to decreased performance. Meanwhile, having a small number of features might limit the diversity of trees in the random forest, which may reduce the model's performance. With a small number of features, the random forest may not be able to capture the underlying patterns in the data, leading to underfitting and poorer performance.

Bagging handles a large number of features well because it creates several base learners based on random subsets of the features. With a large number of features, bagging can create diverse base learners that can improve the model's performance. Meanwhile, having a small number of features can limit the diversity of the base learners in the ensemble, and the complexity of the data may not be captured by the model, thus leading to decreased performance.

2 Practical Question

```
from ucimlrepo import fetch_ucirepo
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay


from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier

from sklearn.metrics import roc_curve
from sklearn.metrics import auc
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_recall_fscore_support

import sklearn.datasets;
data = sklearn.datasets.load_breast_cancer()
```

Data Imbalance Check

There are 212 malacious and 357 benign samples in the dataset. The data is relatively balanced.

In [ ]:

```
data, target = sklearn.datasets.load_breast_cancer(return_X_y = True, as_frame = True)
print(target.value_counts())
x_train, x_test, y_train, y_test = train_test_split(data, target, random_state = 512)
```

```
target
1    357
0    212
Name: count, dtype: int64
```

Useful functions and values for final evaluation

In [ ]:

```
#initializing useful lists
accuracies = {"Test": {}, "Train": {}}
precisions = {"Test": {}, "Train": {}}
recall = {"Test": {}, "Train": {}}
f_score = {"Test": {}, "Train": {}}

def set_precisions_recall_f_score(y_test, test_predictions, y_train, train_predictions, name):
```

```
    test_precision, test_recall, test_f_score, _ = precision_recall_fscore_support(y_test,
test_predictions)
    train_precision, train_recall, train_f_score, _ = precision_recall_fscore_support(y_train,
train_predictions)

    # test_accuracy =

    # print(name, 1 - test_precision, 1 - test_recall, 1 - test_f_score)
    # print(name, 1 - train_precision, 1 - train_recall, 1 - train_f_score)

    precisions["Test"][name] = test_precision
    precisions["Train"][name] = train_precision

    recall["Test"][name] = test_recall
    recall["Train"][name] = train_recall

    f_score["Test"][name] = test_f_score
    f_score["Train"][name] = train_f_score

def set_accuracies(name, train_accuracy, test_accuracy):
    accuracies["Train"][name] = train_accuracy
    accuracies["Test"][name] = test_accuracy
```

Classifiers

In [ ]:

```
# Jeff's changes: DecisionTreeClassifier


dTree = DecisionTreeClassifier(random_state = 512)

dTree.fit(x_train, y_train)

test_score = dTree.score(x_test, y_test)
train_score = dTree.score(x_train, y_train)
print(f"A Decision Tree classifier running on the breast cancer dataset from sklearn gives an accuracy
of{train_score * 100: .3f}% on the training data and an accuracy of{test_score * 100: .3f}% on the
testing data.")

test_predictions = dTree.predict(x_test)
train_predictions = dTree.predict(x_train)

name = "Decision Tree Classifier"
set_accuracies(name, train_score, test_score)
set_precisions_recall_f_score(y_test, test_predictions, y_train, train_predictions, name)
```

A Decision Tree classifier running on the breast cancer dataset from sklearn gives an accuracy of 100.000% on the training data and an accuracy of 90.210% on the testing data.

In [ ]:

```
#requires installing xgboost using pip install xgboost to work.
from xgboost import XGBClassifier

data, target = sklearn.datasets.load_breast_cancer(return_X_y = True, as_frame = True)

x_train, x_test, y_train, y_test = train_test_split(data, target, random_state = 512)

xgbClassifier = XGBClassifier(random_state = 42)
xgbClassifier.fit(x_train.to_numpy(), y_train.to_numpy()) # looking up in xgboost docs, it can't be a pandas dataframe, but can be a numpy array.

test_score = xgbClassifier.score(x_test, y_test)
train_score = xgbClassifier.score(x_train, y_train)
print(f"An XGBoost classifier running on the breast cancer dataset from sklearn gives an accuracy of{train_score * 100: .3f}% on the training data and an accuracy of{test_score * 100: .3f}% on the testing data.")



test_predictions = xgbClassifier.predict(x_test)
train_predictions = xgbClassifier.predict(x_train)


name = "XGBoost"
set_accuracies(name, train_score, test_score)
set_precisions_recall_f_score(y_test, test_predictions, y_train, train_predictions, name)
```

An XGBoost classifier running on the breast cancer dataset from sklearn gives an accuracy of 100.000% on the training data and an accuracy of 94.406% on the testing data.

In [ ]:

```
#James's Changes: Bagging Classifier
data, target = sklearn.datasets.load_breast_cancer(return_X_y = True, as_frame = True)
x_train, x_test, y_train, y_test = train_test_split(data, target, random_state = 512)

bagging = BaggingClassifier(random_state = 512)
bagging.fit(x_train, y_train)

test_score = bagging.score(x_test, y_test)
train_score = bagging.score(x_train, y_train)
print(f"A Bagging classifier running on the breast cancer dataset from sklearn gives an accuracy of{train_score * 100: .3f}% on the training data and an accuracy of{test_score * 100: .3f}% on the testing data.")



test_predictions = bagging.predict(x_test)
train_predictions = bagging.predict(x_train)

name = "Bagging"
```

```
set_accuracies(name, train_score, test_score)
set_precisions_recall_f_score(y_test, test_predictions, y_train, train_predictions, name)
```

A Bagging classifier running on the breast cancer dataset from sklearn gives an accuracy of 99.296% on the training data and an accuracy of 93.007% on the testing data.

In [ ]:

```
# Lucia's changes

data, target = sklearn.datasets.load_breast_cancer(return_X_y = True, as_frame = True)
x_train, x_test, y_train, y_test = train_test_split(data, target, random_state = 512)


adaboost = AdaBoostClassifier(random_state=512)
adaboost.fit(x_train, y_train)

test_score = adaboost.score(x_test, y_test)
train_score = adaboost.score(x_train, y_train)
print(f"An Adaboost classifier running on the breast cancer dataset from sklearn gives an accuracy of{train_score * 100: .3f}% on the training data and an accuracy of{test_score * 100: .3f}% on the testing data.")

test_predictions = adaboost.predict(x_test)
train_predictions = adaboost.predict(x_train)

name = "AdaBoost"
set_accuracies(name, train_score, test_score)
set_precisions_recall_f_score(y_test, test_predictions, y_train, train_predictions, name)
```

```
C:\Users\Admin\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\sklearn\ensemble\_weight_boosting.py:519: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

An Adaboost classifier running on the breast cancer dataset from sklearn gives an accuracy of 100.000% on the training data and an accuracy of 94.406% on the testing data.

In [ ]:

```
# Natasha

data, target = sklearn.datasets.load_breast_cancer(return_X_y = True, as_frame = True)
x_train, x_test, y_train, y_test = train_test_split(data, target, test_size = 0.2, random_state = 5)

random_forest = RandomForestClassifier(n_estimators = 20, criterion = 'entropy', random_state = 51)
random_forest.fit(x_train, y_train)

test_score = random_forest.score(x_test, y_test)
train_score = random_forest.score(x_train, y_train)
```

```
print(f"A Random Forest classifier running on the breast cancer dataset from sklearn gives an accuracy of{train_score * 100: .3f}% on the training data and an accuracy of{test_score * 100: .3f}% on the testing data.")
```

```
test_predictions = random_forest.predict(x_test)
train_predictions = random_forest.predict(x_train)

name = "Random Forest Classifier"
set_accuracies(name, train_score, test_score)
set_precisions_recall_f_score(y_test, test_predictions, y_train, train_predictions, name)
```

A Random Forest classifier running on the breast cancer dataset from sklearn gives an accuracy of 99.780% on the training data and an accuracy of 97.368% on the testing data.

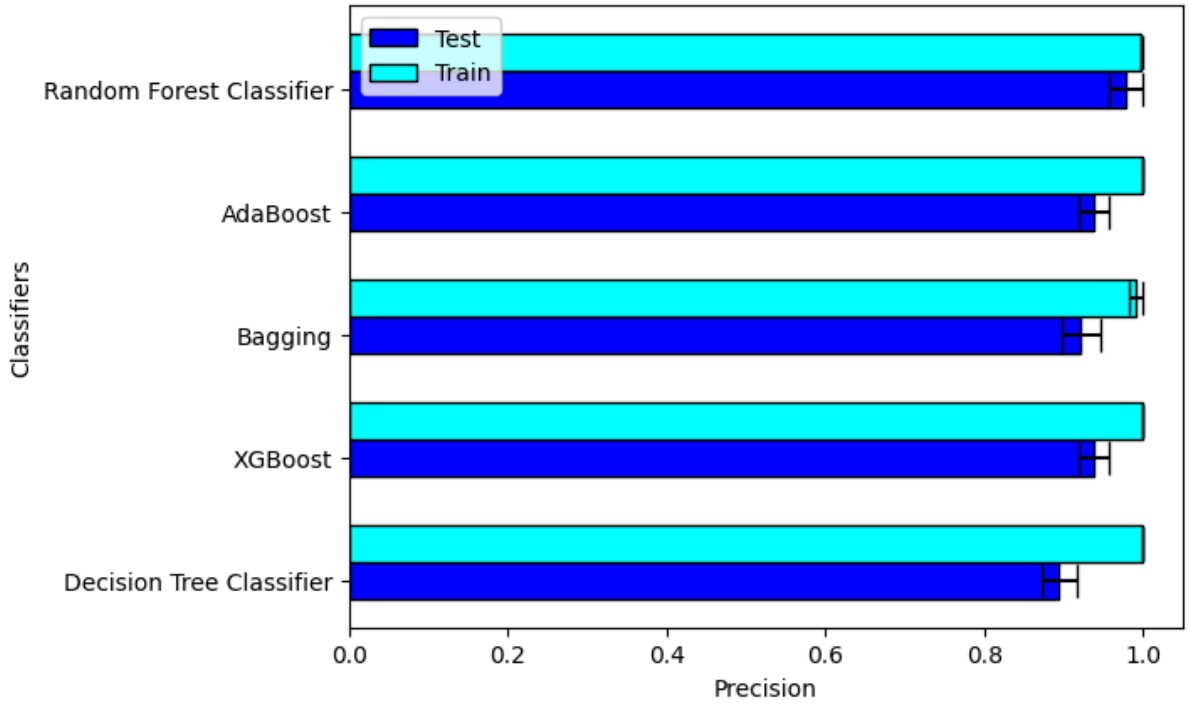
Benchmarks

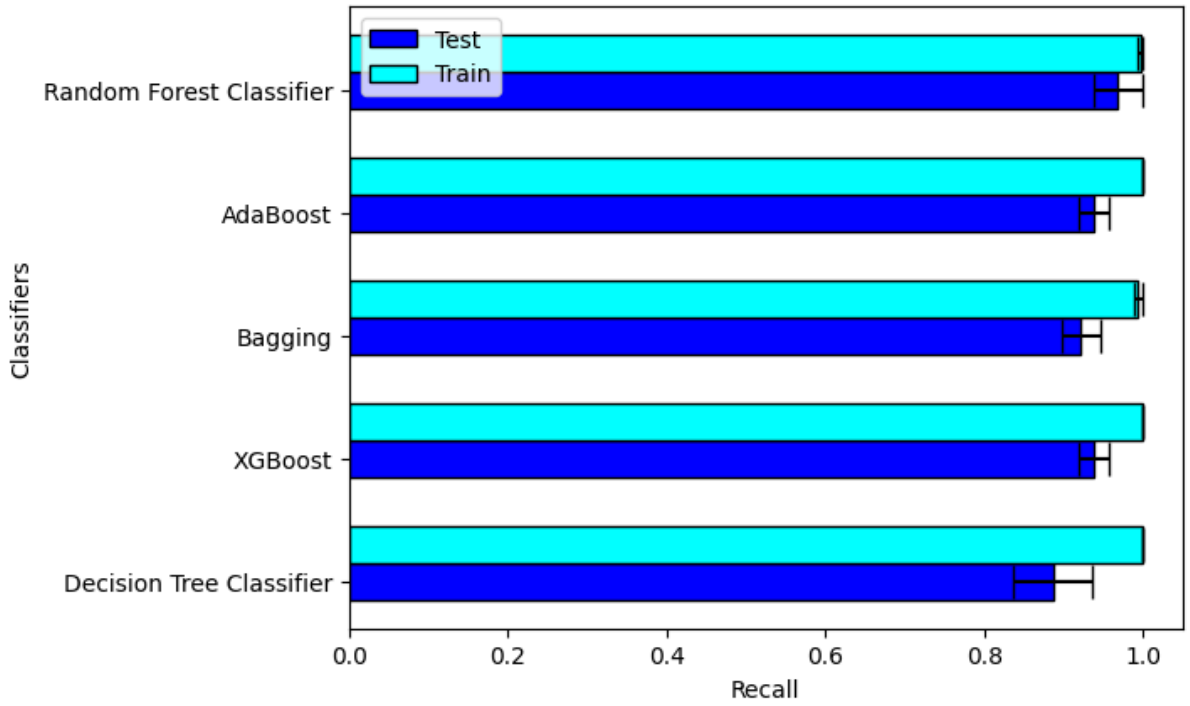
Comparing tables that compare the Accuracy, Precision and Recall of the different models.

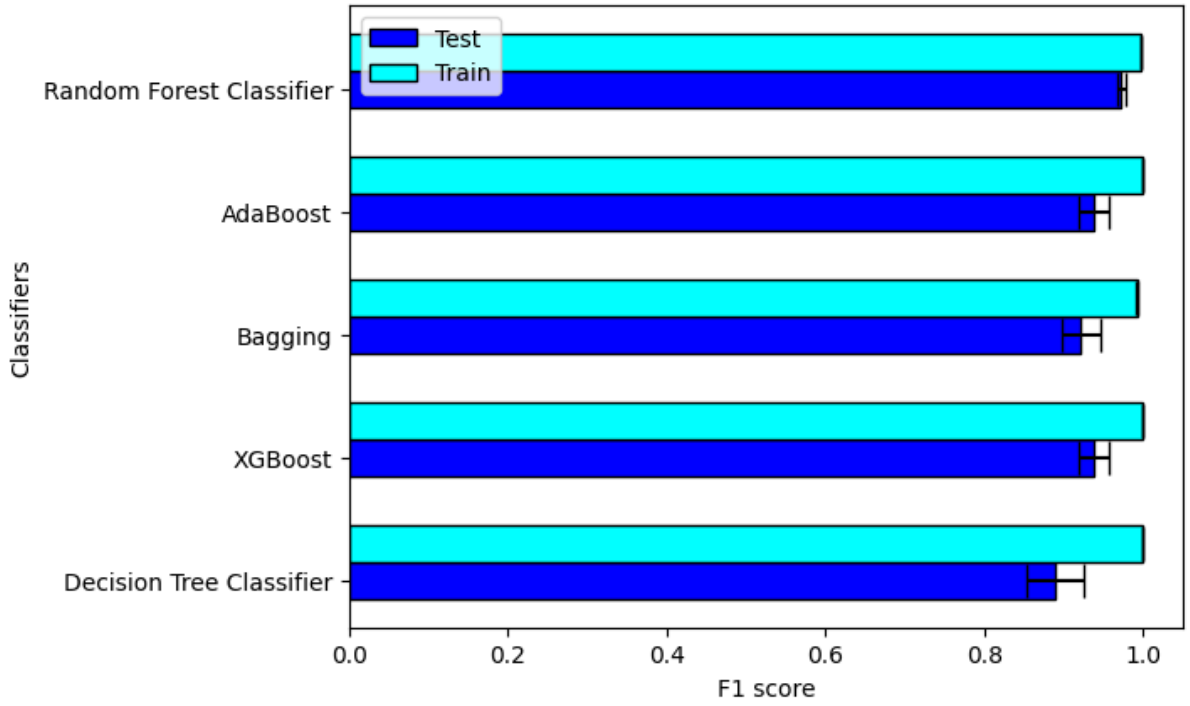
In [ ]:

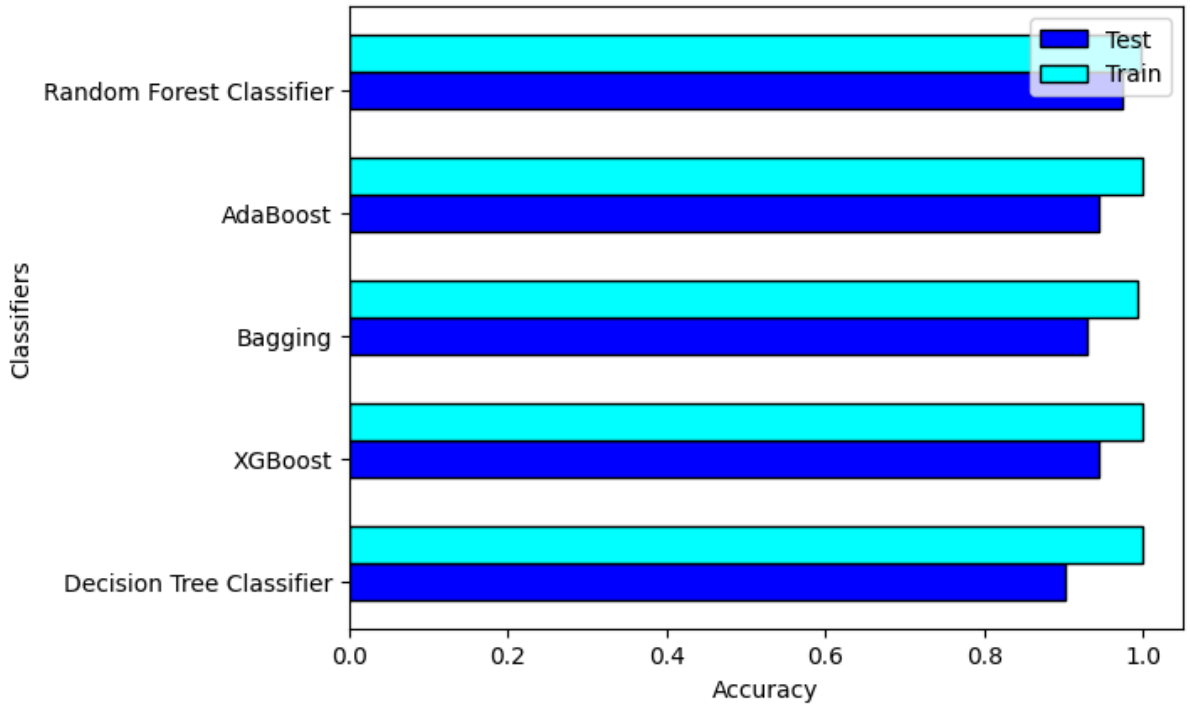
```
def draw_table_w_conf_int(data, statistic):
    bar_width = 0.3

    classifiers = list(data["Test"].keys())
    test_results = list(data["Test"].values())
    train_results = list(data["Train"].values())
    test_exp = []
    test_err = []
    for test_result in test_results:
        test_exp.append(test_result[0] + (test_result[1] - test_result[0]) / 2)
        test_err.append(abs(test_result[1] - test_result[0]) / 2)

    train_exp = []
    train_err = []
    for train_result in train_results:
        train_exp.append(train_result[0] + (train_result[1] - train_result[0]) / 2)
        train_err.append(abs(train_result[1] - train_result[0]) / 2)

    r1 = np.arange(len(test_exp))
    r2 = [x + bar_width for x in r1]

    plt.barh(r1, test_exp, height = bar_width, color = 'blue', edgecolor = 'black', xerr=test_err, capsize=7, label="Test")

    plt.barh(r2, train_exp, height = bar_width, color = 'cyan', edgecolor = 'black', xerr=train_err, capsize=7, label='Train')
```

```
    plt.yticks([r for r in range(len(test_exp))], classifiers)

    plt.xlabel(f"{statistic}")
    plt.ylabel('Classifiers')

    plt.legend()
    plt.show()


def draw_table(data, statistic):
    bar_width = 0.3

    classifiers = list(data["Test"].keys())
    test_results = list(data["Test"].values())
    train_results = list(data["Train"].values())


    r1 = np.arange(len(classifiers))
    r2 = [x + bar_width for x in r1]

    plt.barh(r1, test_results, height = bar_width, color = 'blue', edgecolor = 'black', capsize=7,
label="Test")

    plt.barh(r2, train_results, height = bar_width, color = 'cyan', edgecolor = 'black', capsize=7,
label='Train')

    plt.yticks([r for r in range(len(classifiers))], classifiers)

    plt.xlabel(f"{statistic}")
    plt.ylabel('Classifiers')

    plt.legend()
    plt.show()


draw_table_w_conf_int(precisions, "Precision")

draw_table_w_conf_int(recall, "Recall")

draw_table_w_conf_int(f_score, "F1 score")

draw_table(accuracies, "Accuracy")
```


Test
Train
Random Forest Classifier
AdaBoost
Bagging
XGBoost
Decision Tree Classifier
Classifiers
0.0
0.2
0.4
0.6
0.8
1.0
Precision


Test
Train
Random Forest Classifier
AdaBoost
Bagging
XGBoost
Decision Tree Classifier
Classifiers
0.0
0.2
0.4
0.6
0.8
1.0
Recall





ROC Curve

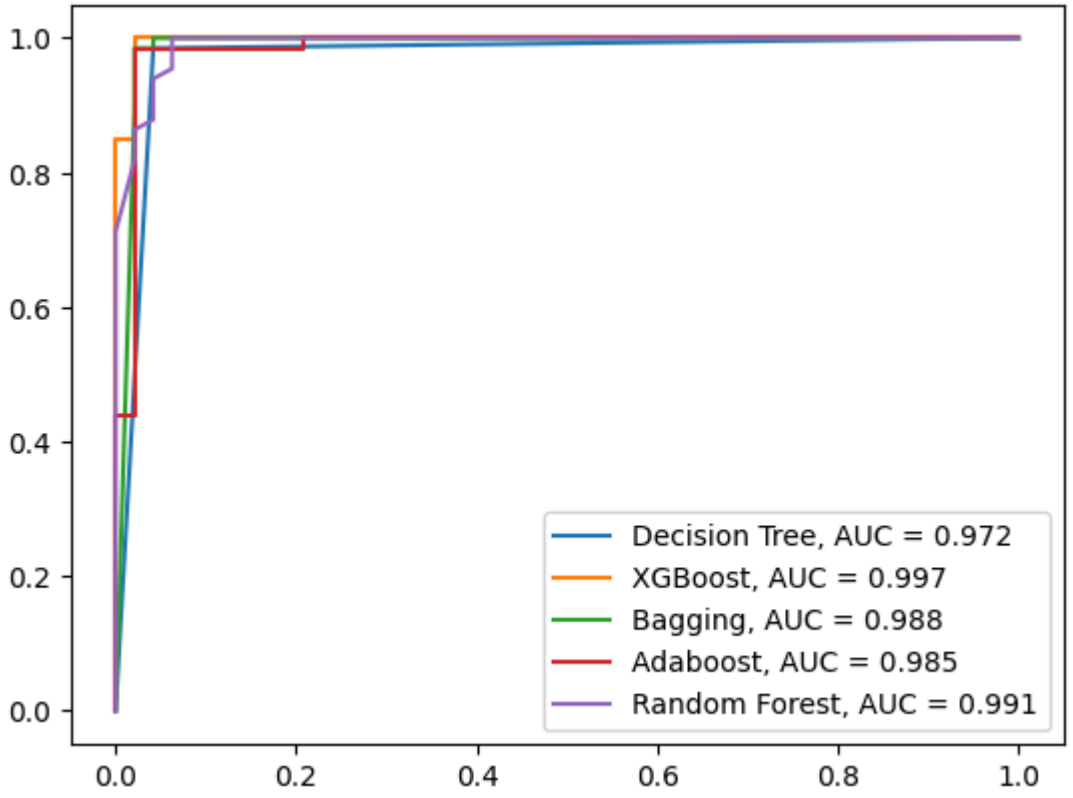
In [ ]:

```
plt.figure(0).clf()

classifiers = {
    "Decision Tree": dTree,
    "XGBoost": xgbClassifier,
    "Bagging": bagging,
```

```
    "Adaboost": adaboost,
    "Random Forest": random_forest
}

for name, classifier in classifiers.items():
    y_pred_probs = classifier.predict_proba(x_test)[:, 1]
    fpr, tpr, _ = roc_curve(y_test, y_pred_probs)
    auc = roc_auc_score(y_test, y_pred_probs)
    plt.plot(fpr, tpr, label = f"{name}, AUC = {auc:.3f}")


plt.legend()
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x171c1116d20>
```



Contribution Section:
Lucia Q1: 4&5, Q2: Adaboost, Roc curve
Jeff Q2: Decision Tree, XGBoost, benchmark graphs
Natasha Q1: 6&7, Q2: Random Forest
James Q1: 1~3, Q2: Bagging