# Code

*Note: To make collaboration easier, this project was done in a Google Colaboratory ipython notebook. Slight modification may be necessary to run in a local Jupyter notebook or standalone python script, such as removing the embedded bash scripts (escaped with “!”). The original notebook in Colab can be seen* [*here*](https://drive.google.com/open?id=1RfhNAQyBca21DsYnIEggmd3kKT8iGoU3).

#If the long url below doesn't work, run the shrorter URL to get the new URL

#!curl "https://drive.google.com/uc?export=download&id=1gYFou8KJGxL-4tG7Ut4ptVmfLjPXRLxw"

!curl "https://doc-04-00-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1/edde69seqvu884lgpj6cdpi3c65vr3c0/1573315200000/12424731692871738664/\*/1gYFou8KJGxL-4tG7Ut4ptVmfLjPXRLxw?e=download" > magic04.csv

!ls

!head magic04.csv

!pip install xgboost

import xgboost as xgb

from sklearn.ensemble import VotingClassifier, BaggingClassifier, RandomForestClassifier

from sklearn.model\_selection import train\_test\_split, KFold

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import roc\_curve, roc\_auc\_score

import pandas as pd

import numpy as np

data = pd.read\_csv("magic04.csv")

X = data.iloc[:, :-1]

Y = data['class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, random\_state =

0, test\_size = 0.3)

# Fit Random Forest Classifier

bestForest = None

bestFalsePositiveRate = 1

for attempt in range(10):

for forestSize in range(1, 50):

ensemble = RandomForestClassifier(n\_estimators = forestSize,

n\_jobs = -1).fit(X\_train, y\_train)

testFalsePositiveRates, testTruePositiveRates, testThresholds =

roc\_curve(y\_test, ensemble.predict(X\_test), pos\_label=1)

if testFalsePositiveRates[1] < bestFalsePositiveRate:

bestForest = ensemble

bestFalsePositiveRate = testFalsePositiveRates[1]

print("The best forest is", bestForest, "with a false positive rate of",

bestFalsePositiveRate)

alex\_preds = bestForest.predict\_proba(X)

# Fit XGBoost Classifier

xg\_reg = xgb.XGBClassifier(objective = 'binary:logistic',

colsample\_bytree = 0.3,

learning\_rate = 0.1,

max\_depth = 5,

alpha = 10,

n\_estimators = 20)

xg\_reg.fit(X\_train, y\_train)

peter\_preds = xg\_reg.predict\_proba(X)

# Fit Bagging Classifier

seed = 17

kfold = KFold(n\_splits = 3, random\_state = seed)

base\_cls = DecisionTreeClassifier()

baggingEns = BaggingClassifier(base\_estimator = base\_cls, n\_estimators = 1000, random\_state = seed, n\_jobs = -1)

bagging = baggingEns.fit(X\_train, y\_train)

jacob\_preds = baggingEns.predict\_proba(X)

#create a dictionary of our models

#estimators=[(‘bag\_ens’, baggingEns), (‘best\_forest’, bestForest),

(‘xgboost\_ens’, xgboostEns)]

#create our voting classifier, inputting our models

#ensemble = VotingClassifier(estimators, voting=’hard’)

ensemble = VotingClassifier(estimators=[('bag\_ens', baggingEns),

('best\_forest', bestForest), ('xgboost\_ense', xg\_reg)],

voting='soft')

#fit model to training data

ensemble.fit(X\_train, y\_train)

allPredictions = ensemble.predict\_proba(X)

allPredictions = pd.DataFrame({'Id': range(len(data)), 'Predicted': allPredictions[:,1]})

allPredictions.to\_csv('submission.csv', index = False)

#If using in Google Colab, run the following line and copy the output to a local file.

!cat submission.csv

# Generate ROC Curves Graphically

import matplotlib.pyplot as plt

plt.figure(0).clf()

# XGBoost

fpr, tpr, thresh = roc\_curve(Y, peter\_preds[:,1])

auc = roc\_auc\_score(Y, peter\_preds[:,1])

plt.plot(fpr,tpr,label= "XGBoost, auc="+str(auc))

# RF

fpr, tpr, thresh = roc\_curve(Y, alex\_preds[:,1])

auc = roc\_auc\_score(Y, alex\_preds[:,1])

plt.plot(fpr,tpr,label= "Random Forest, auc="+str(auc))

# Bagging

fpr, tpr, thresh = roc\_curve(Y, jacob\_preds[:,1])

auc = roc\_auc\_score(Y, jacob\_preds[:,1])

plt.plot(fpr,tpr,label= "Bagging, auc="+str(auc))

# All predictions

allPredictions2 = ensemble.predict\_proba(X)

fpr, tpr, thresh = roc\_curve(Y, allPredictions2[:,1])

auc = roc\_auc\_score(Y, allPredictions2[:,1])

plt.plot(fpr,tpr,label="All Predictions, auc="+str(auc))

x = np.linspace(0, 1)

plt.plot(x, x)

plt.legend(loc=0)

plt.show()

# Explanations

## Random Forest Classifier

An iterative search is made to find the best Random Forest ensemble by creating Random Forests and comparing their accuracies on the test dataset after being trained on the training dataset. Accuracy in my code was measured as the smallest amount of false positives generated by the model, since the description of the data stated that the ROC curve and measuring false positives was the best way to measure the model’s performance. I chose a Random Forest because ever since I implemented my own decision tree in CS 478, I have wanted to use a Random Forest on some data, but hadn’t gotten around to doing it yet. I was also curious to see how well a relatively simple and homogenous ensemble would stack up against the more sophisticated ensembles that other people in the class created.

The number of Decision Trees in the forest is varied between 1 and 50, and this whole range is tested 10 times. This is because some of the hyperparameters that the ensemble sets up for each Decision Tree vary from tree to tree, and in every initialization of the forest. These hyperparameters include things like which subsets of the training data each tree gets, which features each tree considers, and the total depth a tree can reach. My reasoning is that creating many different forests and saving the best one will allow for many different combinations of decision trees to be tested on the training data. The search took several minutes on Google Colab’s runtime, even when I tweaked it to allow the use of all available CPU cores. Interestingly, the final Random Forest only had 2 trees in it but had the best score.

## XGBoost Classifier

XGBoosting is similar to normal Gradient Boosting (which is iteratively adding weak models together to create a stronger model which corrects previous model errors), however, it is specifically used to exploit every bit of memory and hardware resources for tree boosting algorithms. The model is capable of taking in many different parameters, but I used the following for simplicity: objective, colsample\_bytree, learning\_rate, max\_depth, alpha, and n\_estimators. After tinkering around a little with the numbers and comparing the different error rates I settled on the values in the code above.

## Bagging Classifier

Bagging is done through bootstrapping aggregation and is able to decrease errors that come through by decreasing the variance that occurs because of any unstable learners. It uses other classifiers such as Decision Tree and can combine the outputs by voting. The Bagging classifier model Our Bagging classifier has the following parameters: base\_estimator, n\_estimators=10, max\_samples, max\_features, bootstrap, bootstrap\_features, oob\_score, warm\_start, n\_jobs, random\_state, and verbose. For our model, we used the Decision Tree classifier as the base estimator, n\_estimators was 1000, and n\_jobs was -1. Using these parameters listed helped with getting a more accurate prediction.

# Results

The graph demonstrates an interesting comparison between all the ensembles. For some reason, the bagging ensemble was slightly more accurate than the final meta-ensemble. Our theory for this phenomenon is that the gap in accuracy between the bagging ensemble and the other two individual ensembles is so great, the other two models introduce more error than comprehension, even when combined, to the final ensemble. As a result, the final model is actually slightly less accurate than if the bagging model had been used alone. This discrepancy does not appear to be solvable even by fitting the final ensemble after initializing it (the individual models were previously fitted).

Despite this drop in accuracy, our final model performed reasonably well, and at the time of submission we were ranked 6th on the leaderboards (team name is “The Squad”), with a score of 0.98749