**Department of Electrical and Computer Engineering**

Homework Assignment No. 10:

**HW No. 10: Nonparametric Classifiers, ROC Curves and AUC**

submitted to:

Professor Joseph Picone

ECE 8527: Introduction to Pattern Recognition and Machine Learning

Temple University

College of Engineering

1947 North 12th Street

Philadelphia, Pennsylvania 19122

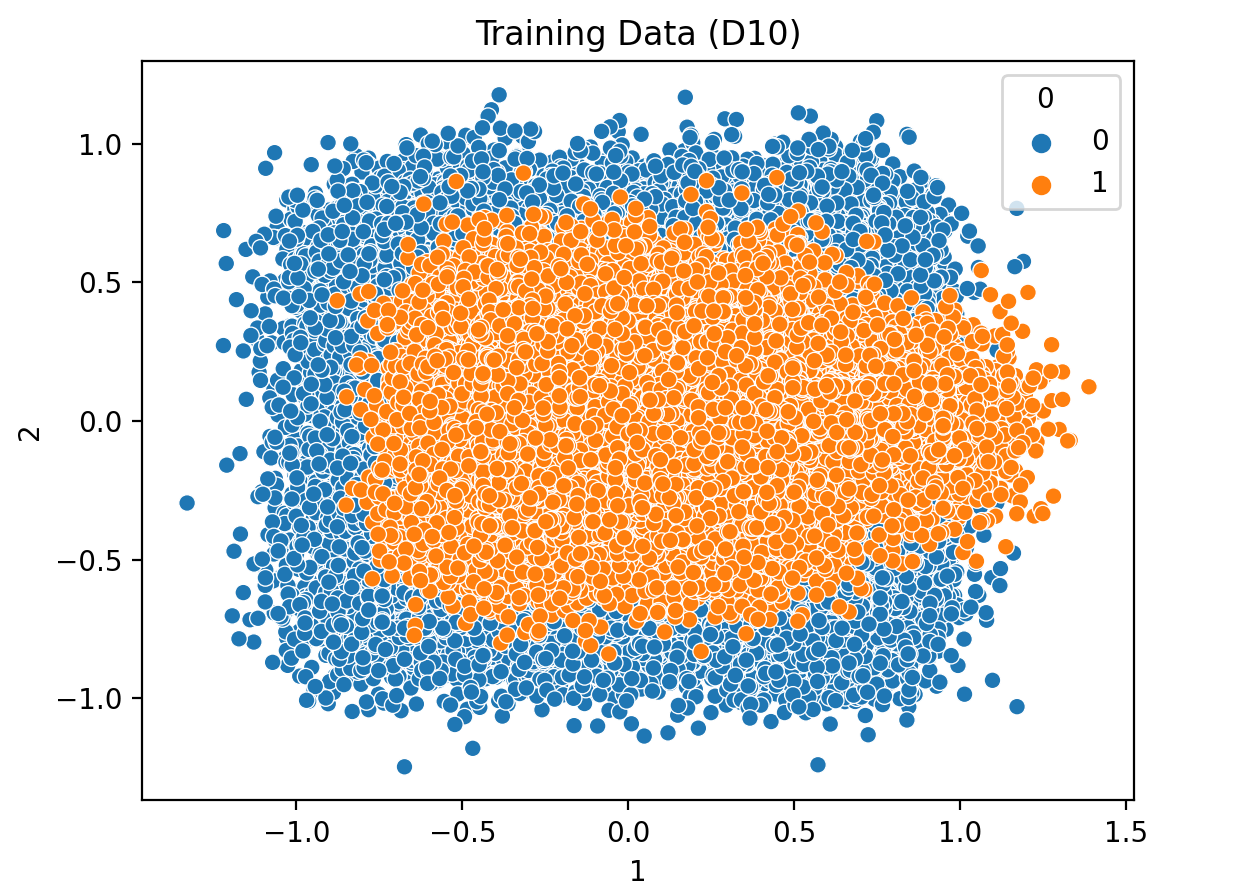
March 30th, 2023

prepared by:

Gavin Koma  
Email: gavintkoma@temple.edu

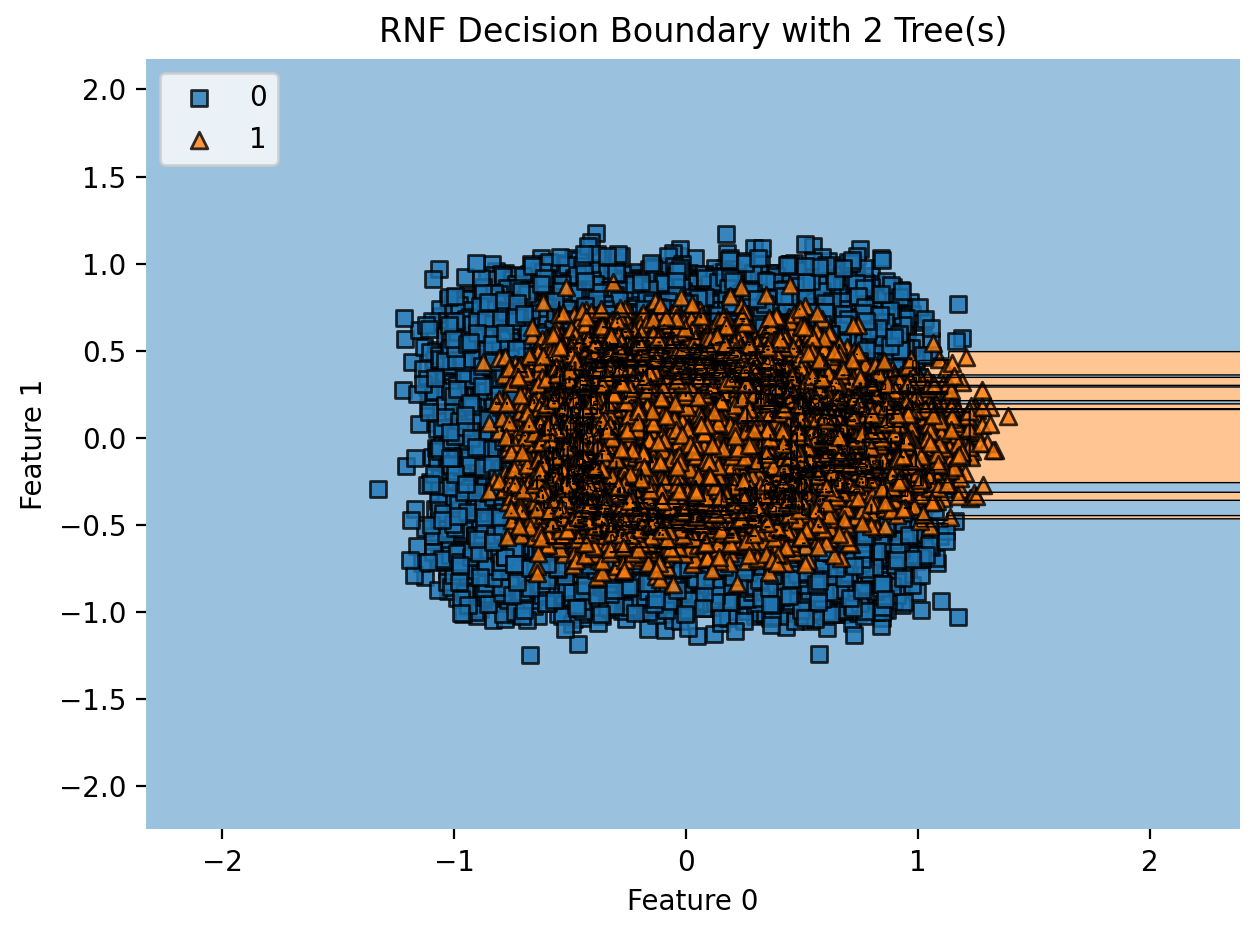
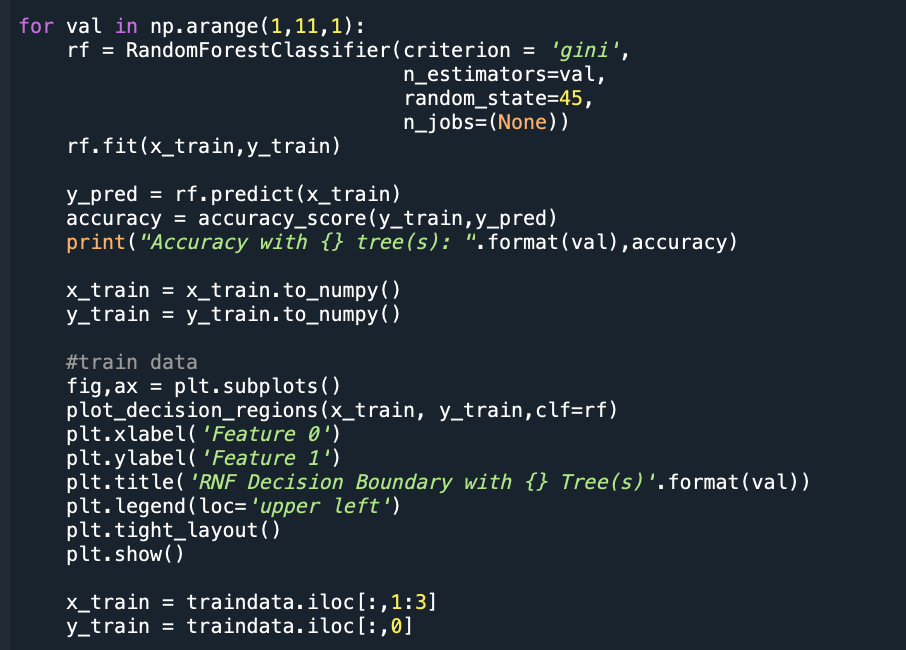
# Task 1

Task 1 of this assignment requires us to use Dataset 10 and implement a Random Forests (RNF) and Support Vector Machines (SVM) using standard Python Packages. We are then to plot the performance on only the eval set as a function of the number of decision tress and as a number of support vectors. To conclude this task, we are to include a table that shows the performance on /train, /dev, and /eval. To begin, this task, I first implement standard Python libraries and will create one graph that plots the original data and another that will plot the RNF decision boundary. These include os (for directory management), pandas and numpy (for basic mathematical computations), and sklearn (for a variety of algorithmic implementations). I will begin by working with the /train data to train my model for the RNF implementation and assess the corresponding ROC curve and their error rates. We obtain the following graph from plotting the training data:

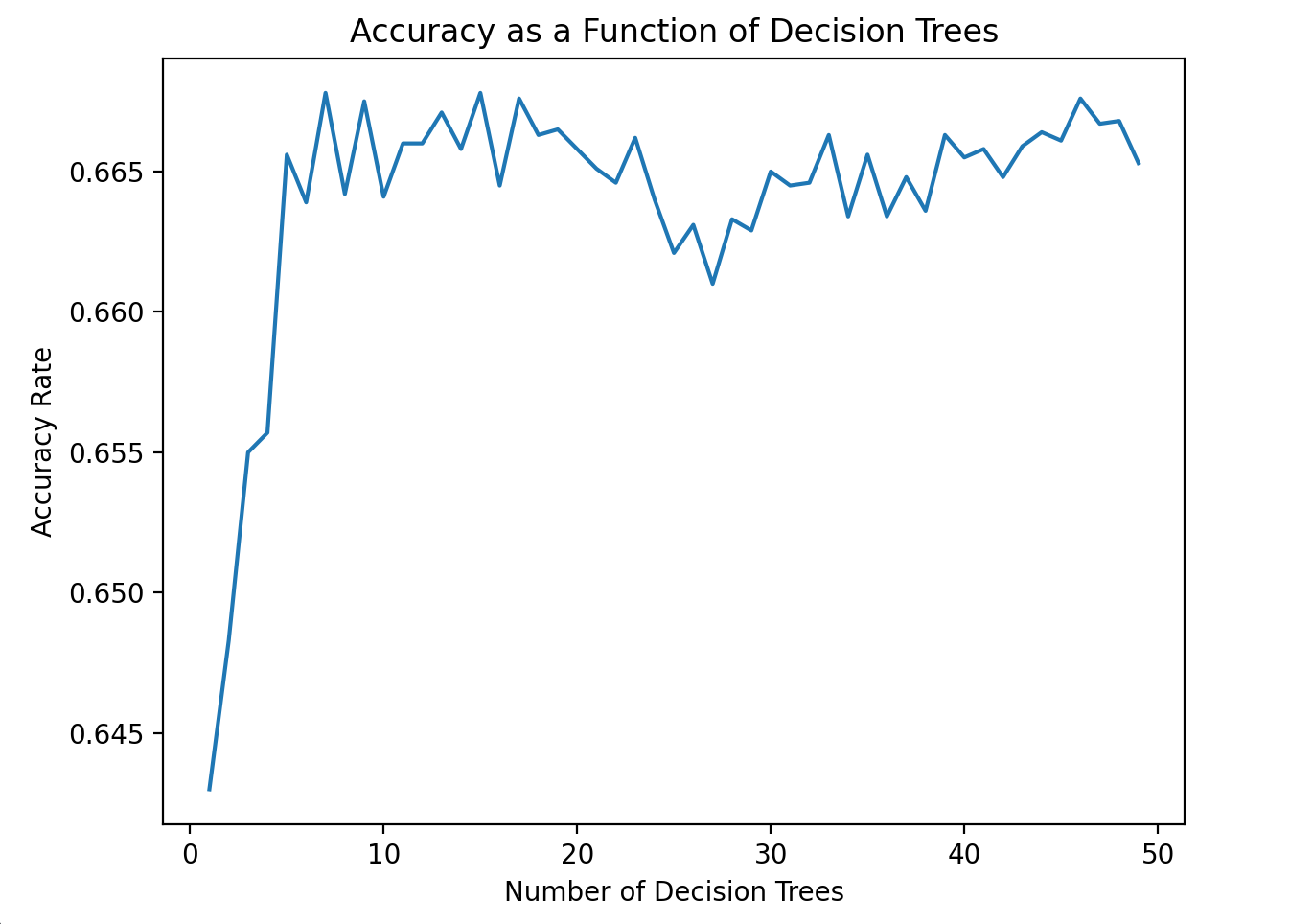


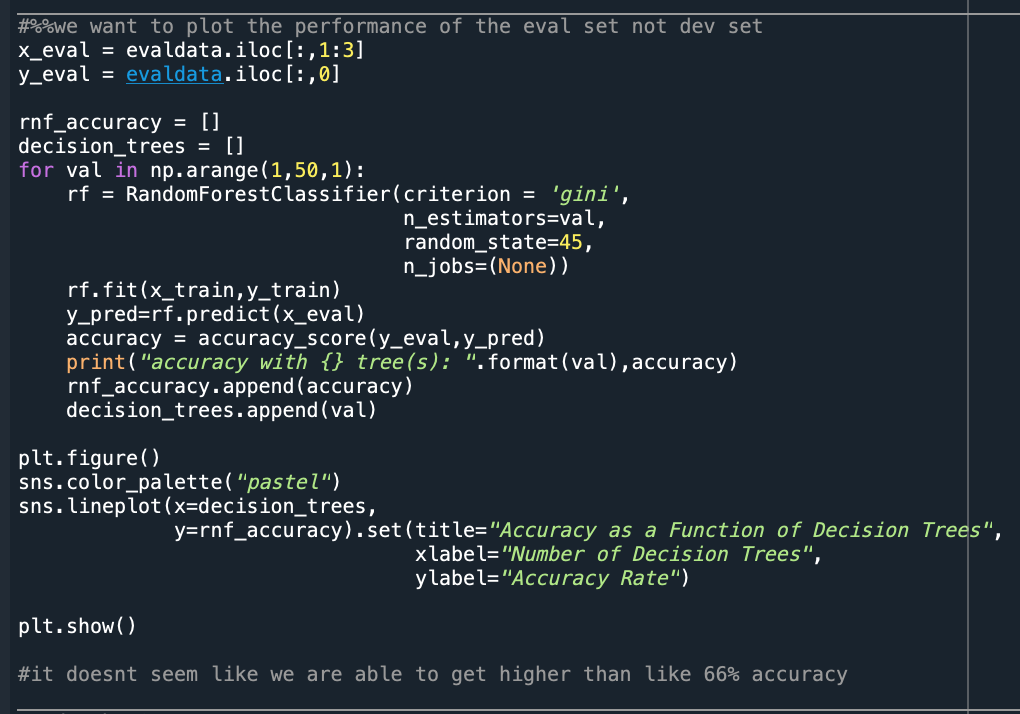
From here, it is essential to determine what hyperparameters to choose for our RNF algorithm. I have chosen to follow the same parameters that are used in imld.py which has ‘gini’ for criterion, random\_state set to 45, and n\_jobs having a Boolean value of False. Unfortunately, I am unsure how many decision trees we should include in our algorithm. RNF algorithms are incredibly prone to overfitting and the chances for this increase as we increase the number of decision tress included in our search.

I decided to create a for-loop that runs a RNF algorithm with n number of decision trees where n is a value provided from a list of 10 numbers ranging from 1 to 10. As expected, we are able to get an accuracy of 0.99243 with 10 decision trees. I am inclined to say that although this accuracy is fantastic at sorting through training data, I worry that it will be unable to generalize for our values in the /dev data and /eval data. For this reason, I am going to do a bit of pruning and instead choose to use a random forest with only 2 trees. 2 trees still give a great accuracy of 0.96187. I am further inclined to say that choosing the accuracy score that is less (but still above 90%) will not be detrimental for our purposes and will potentially allow us to get a higher accuracy when using other data values.

Here is the corresponding code and its output decision boundary:

After successfully implementing the RNF algorithm, our next goal is to plot performance as a function of the number of decision trees *on the eval set*. To do so, I perform a similar action as above. I call my data, create a for-loop, set my hyperparameters, and append my accuracy values to a list to be plotted after. I am able to obtain the following plot with this corresponding code:





Here, we see a peak accuracy of 0.6678 with 7 decision trees included in our RNF algorithm. I understand that normally we are able to obtain an incredibly high accuracy rate with RNF. I am now curious if the lack of data points is resulting in a poor accuracy or if there is notably more overlap than seen in the training data.

The next step of this Task is to implement a Suppert Vector Machine of varying types

# Task 2

# Conclusion