Yukt Mitash
Professor Hrolenok
CS 4641
February 11, 2018

Assignment 1: Supervised Learning

This paper will discuss the performance of the following machine learning algorithms: Decision Trees, Neural Networks, Boosted Decision Trees, Support Vector Machines, and kNN (k-Nearest Neighbors). This paper will observe their performance under a variety of circumstances across two vastly different data sets. I will be using the Scikit learn library to implement all of these algorithms. For all the algorithms performed, the data will be split as such: 85% training and 15% testing.

Data Set 1 Spotify Song Attributes:

This data was collected by someone using Spotify's API. Essentially, it contains 2017 songs with the following information on each song: *Acousticness* (confidence measure of 0.0 to 1.0), *Danceability* (confidence measure of 0.0 to 1.0), *duration* (in milliseconds), *energy* (confidence measure of 0.0 to 1.0), *instrumentalness* (confidence measure of 0.0 to 1.0), *key* (an estimated overall key with integers mapping to keys using standard pitch class notation), *liveness* (confidence measure of 0.0 to 1.0 of whether or not the track is live), *loudness* (measured in decibels averaged across the track form -60 to 0), *speechiness* (the amount of speech used in the track measure form 0.0 to 1.0), *valence* (measure from 0/0 to 1.0 of positivity), *tempo* (estimated in Beats per Minute), *modality* (major is represented by 1 and minor by 0), and *time_signature* (a measure of meter). In addition, each entry in the data contains a classification value of 1 or 0, respectively pertaining to whether the user liked the song or not. All algorithms for this data set will focus on learning/predicting whether or not the user liked the song based on the 13 above characteristics. All data pertaining to a given song is determined by the Spotify API which one can learn about here: https://developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/. The data set is found on kaggle.com and you can learn about that here: https://www.kaggle.com/geomack/spotifyclassification. This data set is very interesting because it examines the makings of how a single person can come to like or dislike a song unconsciously because of 10 different features. The purpose of using this many features is to see how the algorithms perform classification when given many attributes to view the data by.

Data Set 2 Graduate College Admissions:

This data was posted on kaggle and includes the following features: *GRE Score, TOEFL Score, University Rating* (rated from 1-5*), SOP* (rated from 1-5), *LOR* (rated from 1-5), *CGPA, Research* (1 or 0 respectively pertaining the whether or not an applicant had done research), and *Chance of Admit* (represented as a decimal). The size of the dataset is 500. To turn this into classification problem from a regression problem, the algorithms must be testing/predicting a binary output (admit or not). To do this, I modified the data set so that all chances of admission greater than or equal to 0.725 were represented by a 1 and all chances of admission below that were represented by a 0. Rather than predicting whether or not an applicant with certain scores will be admitted or not (a difficult task), algorithms will be predicting whether or not an applicant has a chance of being admitted that is greater than or equal to 0.75. In addition to this, I wanted to change the structure of this problem from the last one to examine how the five algorithms perform under different conditions. To do this, I decided to only look at three factors: CGPA, GRE score, and University Rating. This allows more variety across problem sets. In addition, it gives us the chance to watch how the training times, training scores, learning scores, and confusion matrices change when a smaller number of features are given. Learn more about the data here: https://www.kaggle.com/mohansacharya/graduate-admissions.

Decision Trees

Overview: Essentially, a decision tree is a supervised learning will continuously split a set of test data according to the "best" attribute at each turn. It will continue to do this until all of the data is divided into two groups based on the "target" attribute. In the following cases the target attributes are whether or not a certain song is liked and whether or not a person with a certain profile has a acceptance chance greater than or equal to 0.75. The "best" attribute is generally the one that most accurately splits the data by the classification attribute. The nodes used to split the training data are then applied to the test data. Because decision trees are designed to keep splitting the data until they have classified everything according to the classifying attribute, it will always have perfect or near perfect accuracy with training data. As the tree gets bigger it becomes more and more accurate with training data, but less accurate with testing data that it has not seen because the tree is becoming much less generalized. This undoubtedly results in overfitting, a phenomenon in which an ML algorithm becomes overly specified to the training data and as a result less accurate on new data. This can be fixed with pruning. Pruning is when, after growing the tree, the algorithm gets rid of nodes that produce data that is too specific, thereby reducing the complexity of the tree. I implemented pruning by "borrowing" some code from stackoverflow. Essentially, I traverse the tree and find which nodes have a minimum class count below n and remove their children. I implemented this twice on the first data set: once with an n value of 100 and again with one of 2000 to show the dangers of over-pruning. For the second data set I used n values of 40 and 300 to reflect the smaller dataset. Below are pictures/graphs.

Decision Trees W/O Pruning
Data Set 2:

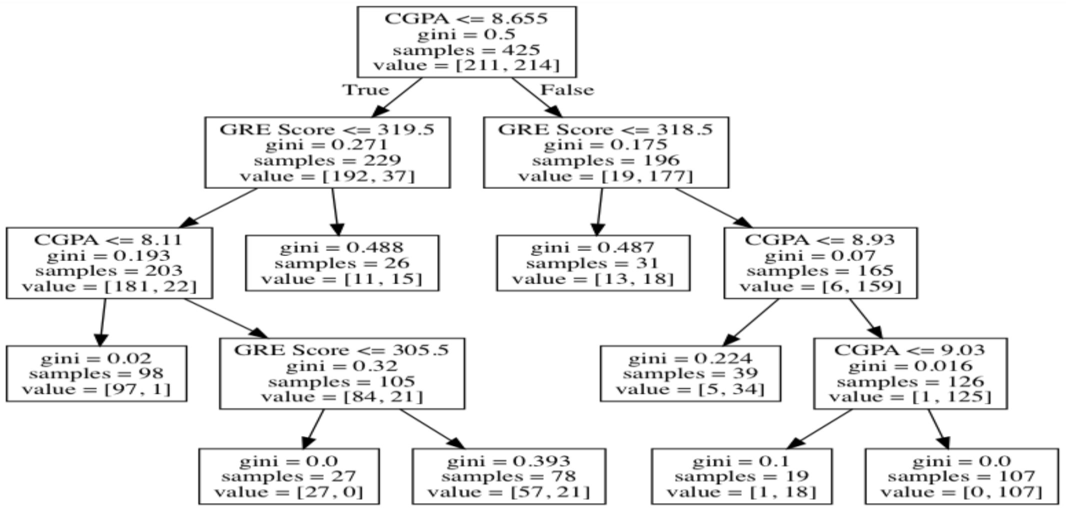


Figure 1

```
Analysis of Decision Tree Without Pruning:
Testing Score: 0.8266666666666667 and Training Score: 0.8776470588235294
Training Error: 0.12235294117647055
Testing Error: 0.17333333333333334
Confusion Matrix:
[[34  7]
 [ 6 28]]
Total:  75
Correct Positives  34
Correct Negatives  28
False Positives:  7
False Negatives:  6
Training time of Decision Tree w/o pruning:  0.01657271385192871 seconds
```

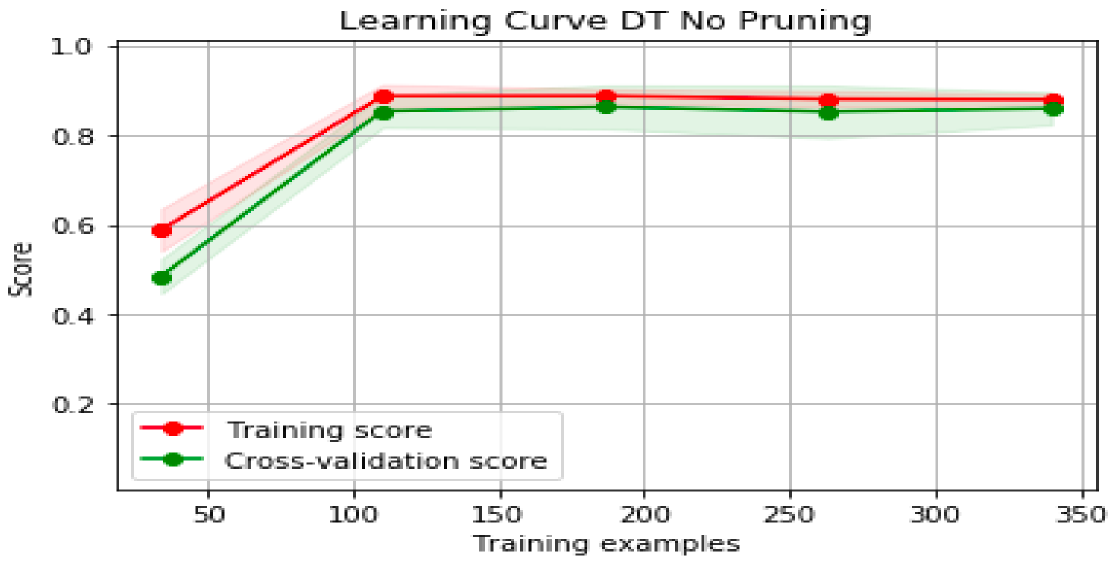
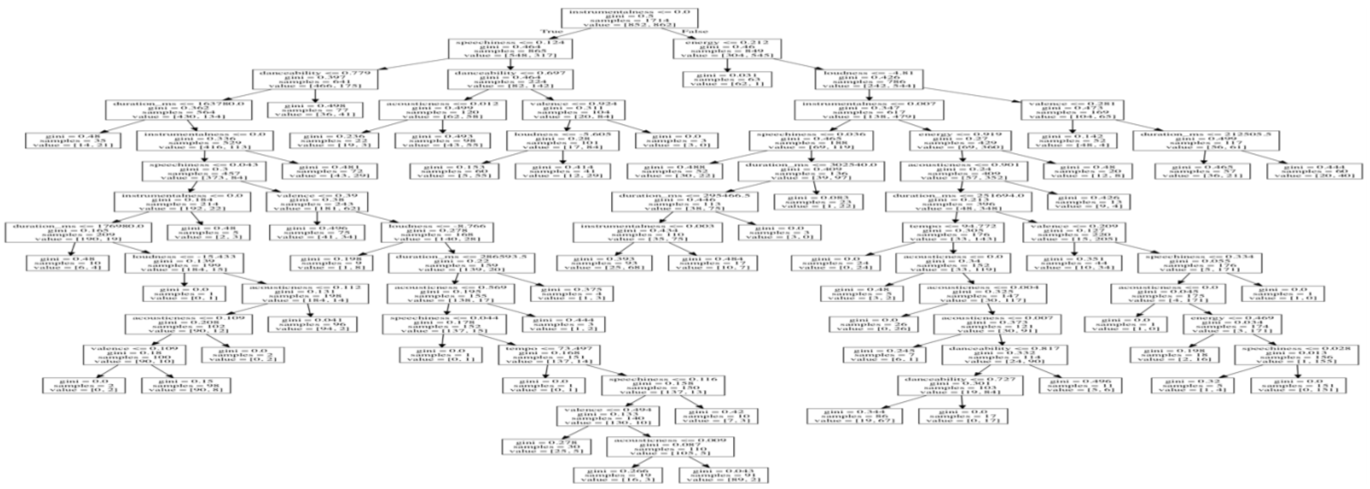


Figure 2

Data Set 1:



Analysis of Decision Tree Without Pruning:
Testing Score: 0.6732673267326733 and Training Score: 0.7893815635939323
Training Error: 0.2106184364060677
Testing Error: 0.3267326732673267
Confusion Matrix

```
[[ 89  56]
 [ 43 115]]
```

Total: 303
Correct Positives 89
Correct Negatives 115
False Positives: 56
False Negatives: 43
Training time of Decision Tree w/o pruning: 0.0322012901306152 34

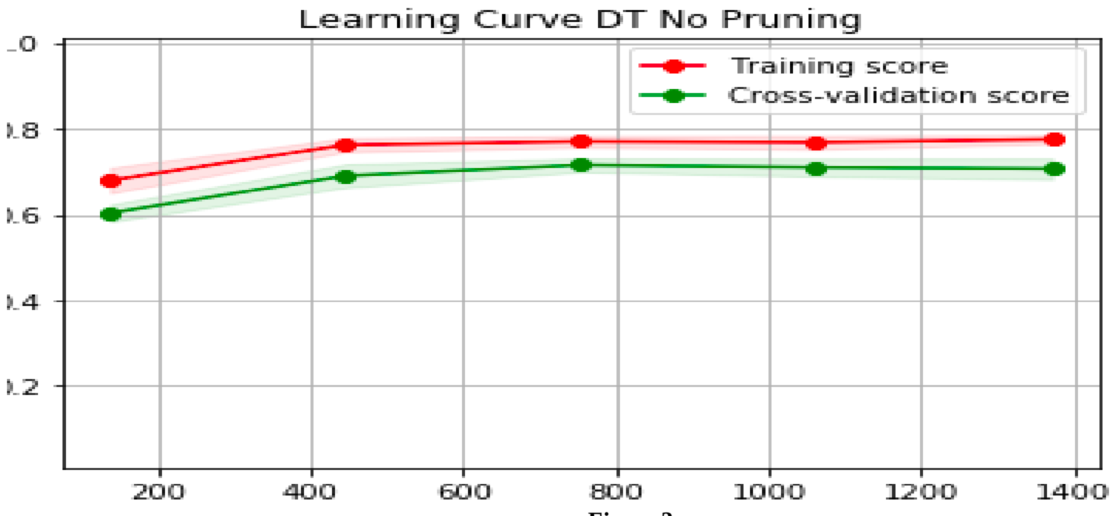


Figure 3

Written Analysis:

Overfitting seems more apparent in the tree for the second data set than for the first data set. The tree looks very large and possibly over-specified to the training data. One indicator of overfitting is that as the training data set increases, the training score will also increase, but the testing score will drop marginally. This is not overly apparent in Figure 3, but there seems to be some degree of it. Another indicator is a training score that is much higher than the testing score. There is some degree of this in data set 2 with respective rounded training and testing scores at 0.88 and 0.83, but it is much more apparent with data set 1 with respective rounded training and test scores at 0.79 and 0.67. This shows that the Decision Tree is definitely specified to the training data, which makes sense given the shape of the tree. It is somewhat surprising that the data set two shows a much higher accuracy with the Decision tree algorithm when it is given much fewer attributes to partition the data with. Perhaps this is due to the nature of the problem. The confusion matrices show nothing notable.

Decision Trees W/ Pruning

For the pruning algorithm the threshold separating classes will be 100 and 30 for data set 1 and data set 2 respectively. Therefore, nodes that split fewer than 100 and 30 data respectively will be removed.

Data Set 2:

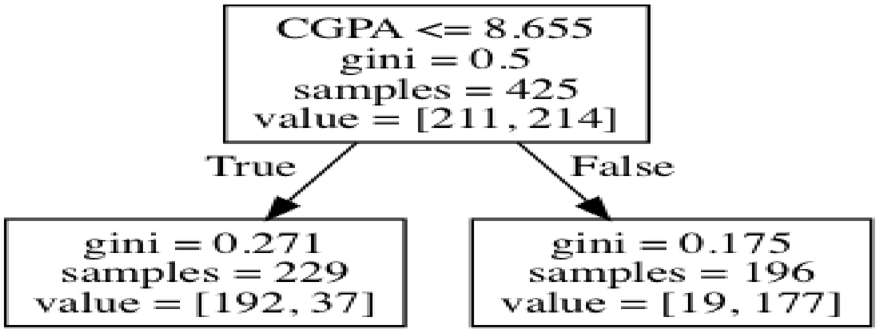


Figure 4

Analysis of Decision Tree With Pruning:
Testing Score: 0.84 and Training Score: 0.8682352941176471
Training Error: 0.1317647058823529
Testing Error: 0.16000000000000003
Confusion Matrix
[[36 5]
 [7 27]]
Total: 75
Correct Positives 36
Correct Negatives 27
False Positives: 5
False Negatives: 7
Training time of Decision Tree w/o pruning: 0.029604196548461914

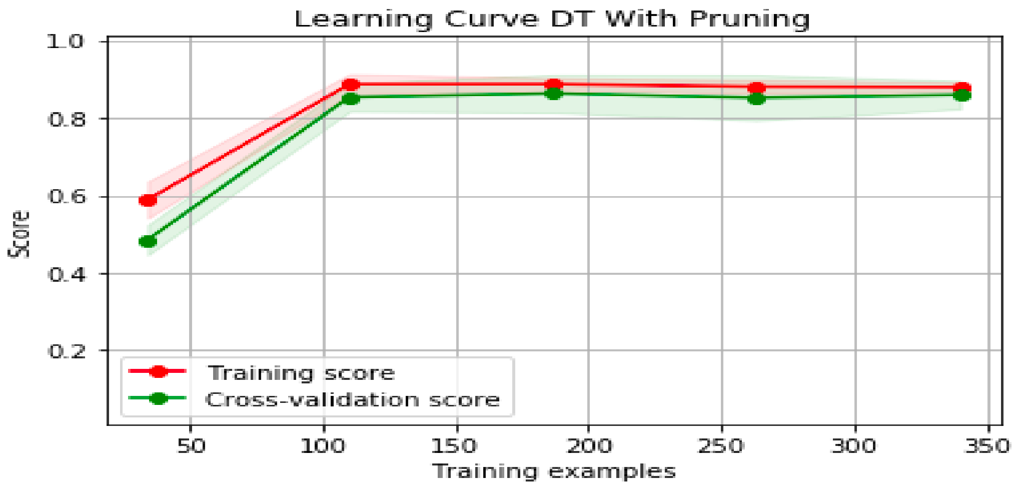


Figure 5

Data Set 1:

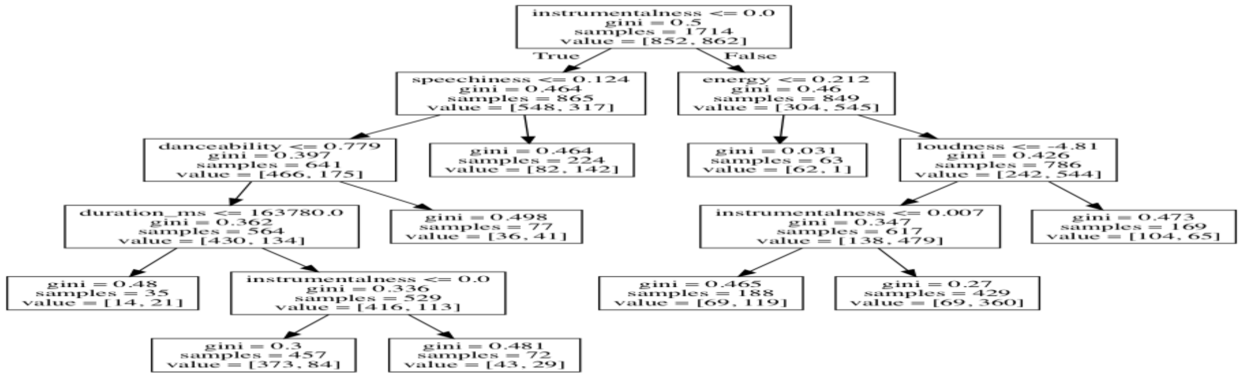


Figure 6

Analysis of Decision Tree With Pruning:
Testing Score: 0.7161716171617162 and Training Score: 0.7380396732788798
Training Error: 0.26196032672112024
Testing Error: 0.2838283828382838
Confusion Matrix
[[92 53]
 [33 125]]
Total: 303
Correct Positives 92
Correct Negatives 125
False Positives: 53
False Negatives: 33

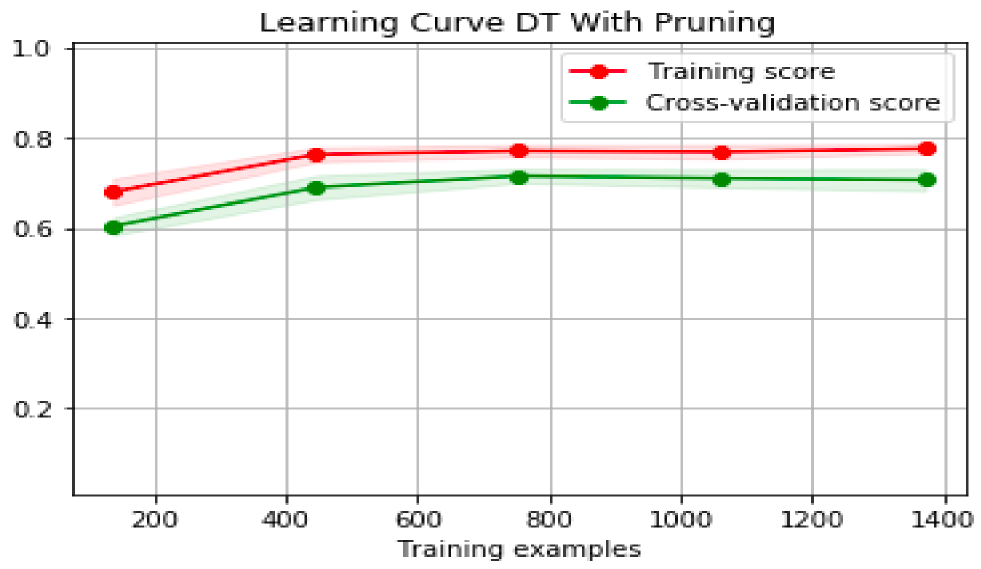


Figure 7

Written Analysis

It is visually apparent that pruning generalizes a decision tree. The number of partition nodes was decreased aggressi vely for each decision tree. In fact, for data set 2, it was reduced to the point which the only splitting factor was CGPA. One woul d think this generalizes the splitting too much, but it increased the testing score from 0.83 to 0.84. In both cases the training score decreased, as one would expect happens when you remove nodes from the tree. For data set 1 it went from 0.79 to 0.74 and for da ta set 2 it went from 0.88 to 0.87. This is evidence of the algorithm become less specified to the training data and more generalize d. Unfortunately, the learning curve does not change as much as one would expect. Ideally it would show the cross-validation sco re increasing at a faster rate and the training score increasing at a slower rate. Again, the confusion matrix shows nothing interesti ng. It is worth noting that the difference in training times is 0.03 with data set 1 taking longer. Data set 1 has roughly four times a s much data.

Decision Trees With Aggressive Pruning

For the pruning algorithm the threshold separating classes will be 300 and 2000 for data set 1 and data set 2 respectively. Therefore, nodes that split fewer than 300 and 2000 data respectively will be removed. Doing this had essentially the same effect on each data set, so we will only be looking at Data Set 2.

Data Set 2:

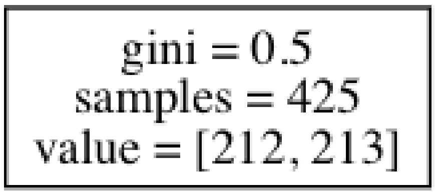


Figure 8

Analysis of Decision Tree With Too Much Pruning:
Testing Score: 0.4666666666666667 and Training Score: 0.5011764705882353
Training Error: 0.49882352941176467
Testing Error: 0.5333333333333333
Confusion Matrix
[[0 40]
 [0 35]]
Total: 75
Correct Positives 0
Correct Negatives 35
False Positives: 40
False Negatives: 0

Written Analysis:

Essentially, over aggressive pruning will get rid of all nodes in the tree and randomly divide the data, causing a Testing Score of roughly 0.47.

Neural Networks

Overview: A neural network is a supervised learning algorithm that uses a series of neurons and layers with weights and biases to make a prediction based on some data. Essentially, a neural network starts off with random weights and biases and uses those to determine an output based on an input training data. If the output is off, the neural network will back propagate and adjust the weights and biases. This is done repeatedly on the training data. We will implement this using skLearn's Neural Network. Here, I have compared the performance of the network for both data sets based on the number of hidden layers that the network has. A neural network will be created and tested with varying number of layers. The optimal number of layers will be determined based on the testing score and further analysis shall be done on the neural network with the optimal layers. Below are some graphs.

Data Set 1:

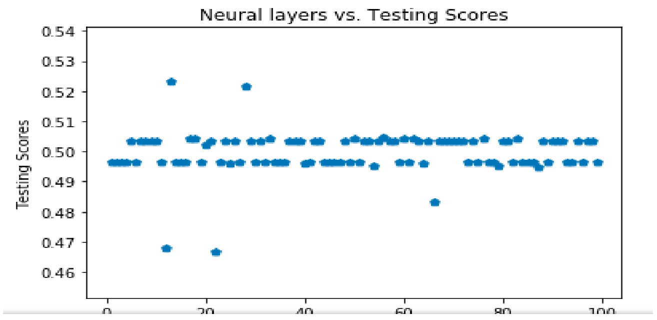


Figure 9

Best Score: 0.5233372228704785
Optimal Layers: 13

Analysis of Neural Network With 13 layers:
Testing Score: 0.5643564356435643 and Training Score: 0.5233372228704785
Training Error: 0.47666277712952154
Testing Error: 0.4356435643564357
Confusion Matrix
[[75 71]
 [61 96]]

Total: 303
Correct Positives 75
Correct Negatives 96
False Positives: 71
False Negatives: 61
Training Time of Neural Net With 13 layers: 0.34138917922973633

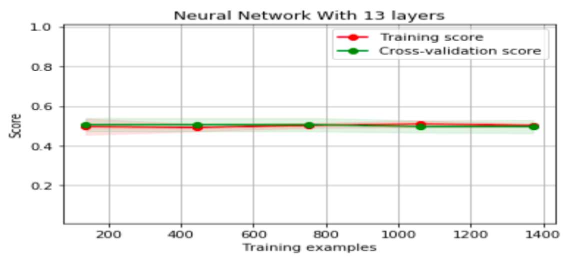


Figure 10

Data Set 2:

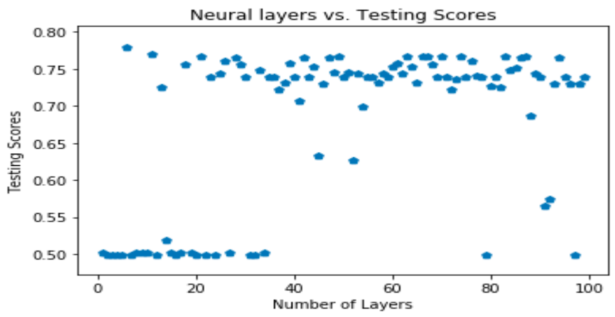


Figure 11

Best Score: 0.7788235294117647
Optimal Layers: 6

Testing Score: 0.6933333333333334 and Training Score: 0.72
Training Error: 0.28
Testing Error: 0.30666666666666664
Confusion Matrix
[[20 20]
 [3 32]]
Total: 75
Correct Positives 20
Correct Negatives 32
False Positives: 20
False Negatives: 3
Training Time of Neural Net With 6 layers: 0.30724191665649414

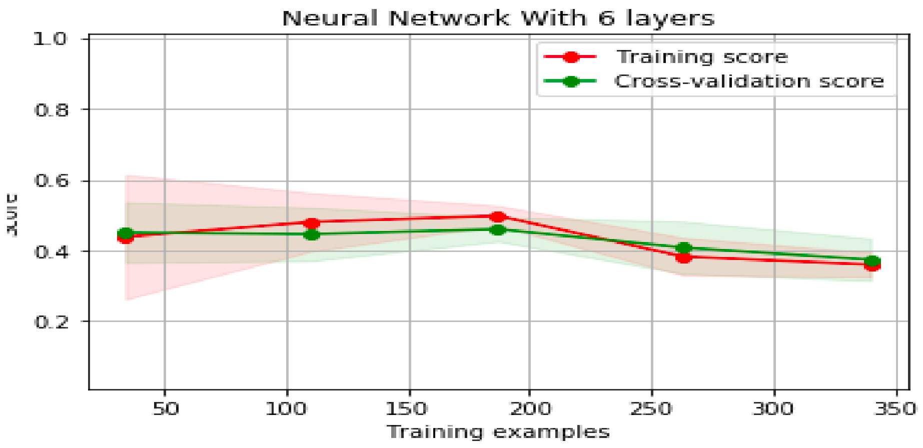


Figure 12

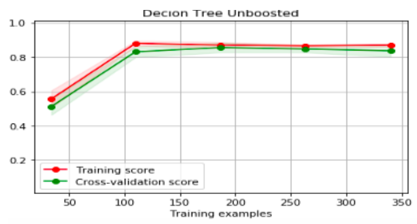
Written Analysis

Although neither graph shows a clear relationship between the number of layers and Neural Network performance, it seems that the best performance generally occurs at a mid-size number relative to the inputs/outputs. This makes sense as a large number of layers will likely cause overfitting because the model would become over-adjusted to the training data and the weights and biases would be very niche. Both confusion matrices showed a higher likelihood of a correct response when the correct response was negative. In both cases, the learning curves showed that the training scores and cross-validation scores were almost identical. This is evidence of little overfitting. Both the learning curves show that this algorithm does not require very much data to properly train. Finally, unlike other algorithms, data set 2 performed significantly better than data set 1 (0.72 and 0.52) with rounding. This shows that neural networks perform a lot better with fewer inputs. Also given the largely mathematical nature of neural networks as opposed to decision trees (I am not saying decision trees do not operate on a mathematical basis. I am only saying that the Neural Network algorithm is likely better at determining relationships that directly involve numbers due to the weights and biases system), Neural Networks are likely better at solving problems with a clearer numbers-based relationship like graduate admissions than problems that are subjective like enjoying music.

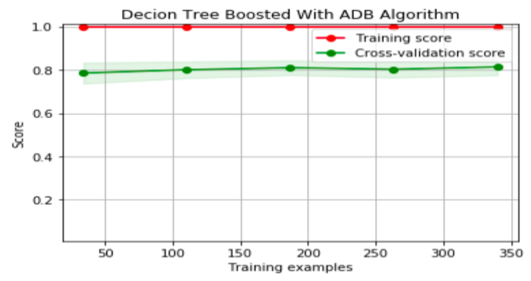
Boosted Decision Trees

Overview: The biggest problem with decision trees is their inflexibility in classifying data they were not trained with. Boosting algorithms aim to make decision trees more adaptable. The two algorithms we will analyze are Random Forests and AdaBoost. Random Forest creates bootstrapped subsets of the original data set and uses a subset of the parameters for each bootstrapped set to create a tree. This results in many trees. When classifying a datum, the datum is run through each tree and whichever classification receives the most "votes" is the output. AdaBoost uses a similar approach. It uses all the parameter in isolation and determines how good each parameter is at classifying the training data. Based on this each parameter is given a weight. The aggregate of the outputs and weights of each single classifying parameter determines the output. Let's use the SkLearn library to examine these algorithms. Below is the data.

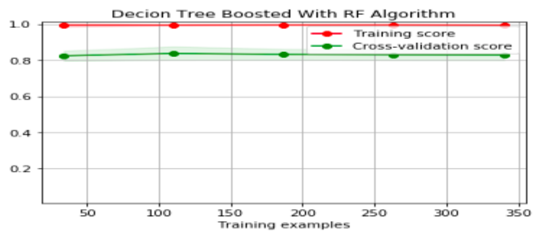
Data Set 2:



Testing Score: 0.8666666666666667 and Training Score: 0.8635294117647059
Training Error: 0.13647058823529412
Testing Error: 0.1333333333333333
Training time of unboosted Decision Tree: 0.003609895706176758

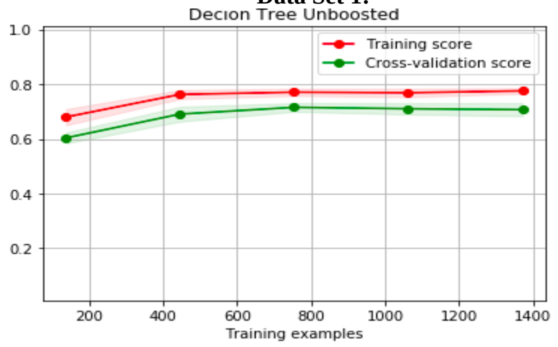


Testing Score: 0.76 and Training Score: 1.0
Training Error: 0.0
Testing Error: 0.24
Training time of Decision Tree with ADA Algorithm Boosting: 0.005786895751953125

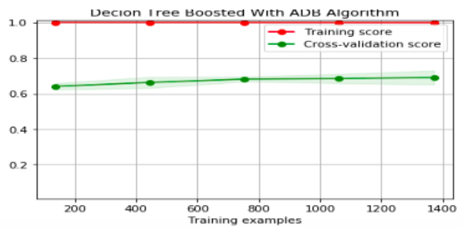


Testing Score: 0.84 and Training Score: 0.9929411764705882
Training Error: 0.007058823529411784
Testing Error: 0.16000000000000003
Training time of Decision Tree with RF Algorithm Boosting: 0.09227180480957031

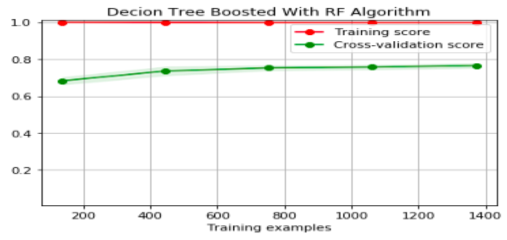
Data Set 1:



Testing Score: 0.6732673267326733 and Training Score: 0.7893815635939323
Training Error: 0.2106184364060677
Testing Error: 0.3267326732673267
Training time of unboosted Decision Tree: 0.020224571228027344



Testing Score: 0.6864686468646864 and Training Score: 0.9982497082847142
Training Error: 0.0017502917152858455
Testing Error: 0.31353135313531355
Training time of Decision Tree with ADA Algorithm Boosting: 0.16762995719909668



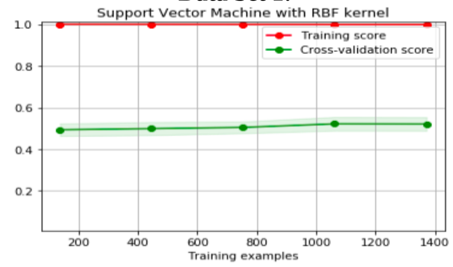
Testing Score: 0.7623762376237624 and Training Score: 0.9970828471411902
Training Error: 0.0029171528588097795
Testing Error: 0.2376237623762376
Training time of Decision Tree with RF Algorithm Boosting: 0.10079216957092285

Written Analysis: It is interesting to note that both boosting algorithms made the decision tree for data set #2 perform worst. In both cases, the RF algorithm was the better algorithm. All boosting reduced the number of training samples required for the algorithm to reach peak performance as evidenced by the learning curves of boosting compared to the upboosted learning curve. The Random Forest algorithm caused a ten percent boost in the accuracy of data set #1. This is very impressive and likely due to the fact that the second data set has many parameters. This allowed the algorithm to create many subtrees and create a more accurate estimate. Finally, it is clear that boosting works best for problems with a large number of features. In fact, both algorithms reduced performance for data set #2. This is due to the fact that both of these algorithms create subtrees by examining subsets of the parameters. This will obviously work better with many parameters. It is important to note that all boosting algorithms cause a greater accuracy with training data. This likely lead to overfitting with data set #2.

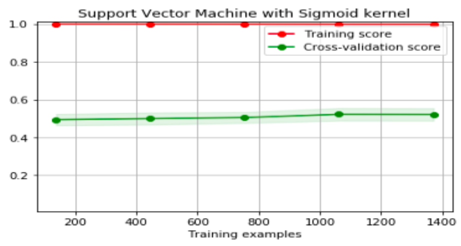
Support Vector Machines

Overview: SVM is a supervised learning algorithm that creates a vector of weights in a plane to divide all data. The vector equation will return a value greater than 1 for certain data and less than 1 for other, thus classifying the data. SVM aims to use weights that maximize the differences between each class. Essentially SVM uses an algorithm to best map data to a certain class. The function is based off of a kernel. The two kernels we will look at are the RBF kernel and the Sigmoid kernel.

Data Set 1:

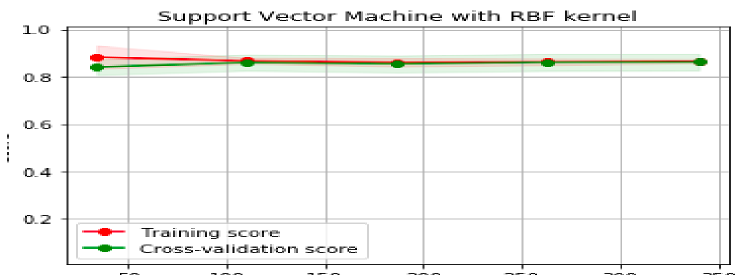


Testing Score: 0.5775577557755776 and Training Score: 0.9982497082847142
Training Error: 0.0017502917152858455
Testing Error: 0.42244224422442245
Training time of SVM with RBF kernel: 0.19920706748962402

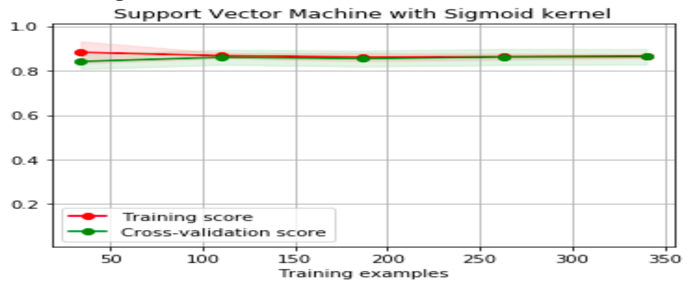


Testing Score: 0.5214521452145214 and Training Score: 1.0
Training Error: 0.4970828471411902
Testing Error: 0.47854785478547857
Training time of SVM with Sigmoid Kernel: 0.129317045211792

Data Set 2:



Testing Score: 0.8533333333333334 and Training Score: 0.8635294117647059
Training Error: 0.13647058823529412
Testing Error: 0.1466666666666666
Training time of SVM with RBF kernel: 0.011116981506347656



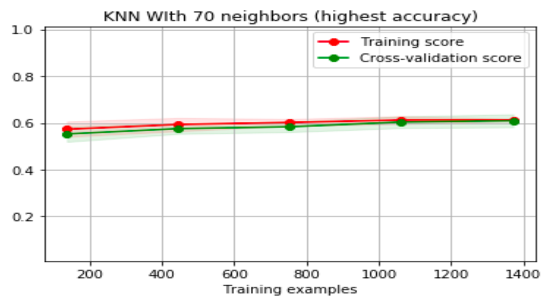
Testing Score: 0.4666666666666667 and Training Score: 0.5011764705882353
Training Error: 0.49882352941176467
Testing Error: 0.5333333333333333
Training time of SVM with Sigmoid Kernel: 0.0069010257720947266

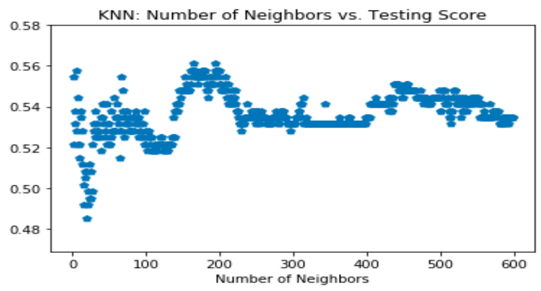
Written Analysis: For data set 1, the RBF kernel proved to be more accurate. With data set 1, both kernels had an enormous amo unt of overfitting to the training data, as evidenced by the training score of 1.0. The RBF kernel took much longer to train. Also, SVMs seem to reach their highest accuracy point with less data than other algorithms. It seems that the RBF kernel performs muc h better with more parameters.

K-Nearest Neighbors

Overview: The k-nearest neighbor algorithm is a "lazy algorithm" that determines its output based on examining the k-nearest da ta points and making an informed decision based on these. We try to find the optimal k for each problem and then further test the algorithm under those conditions.

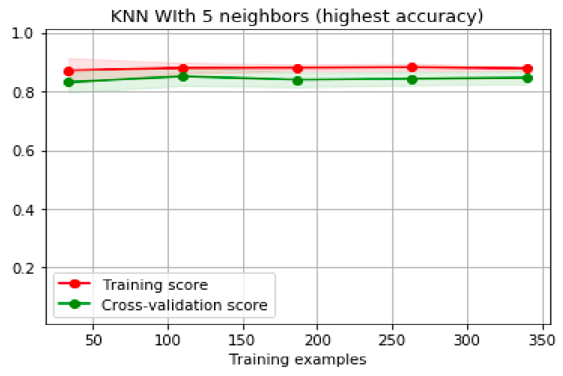
Data Set 1:

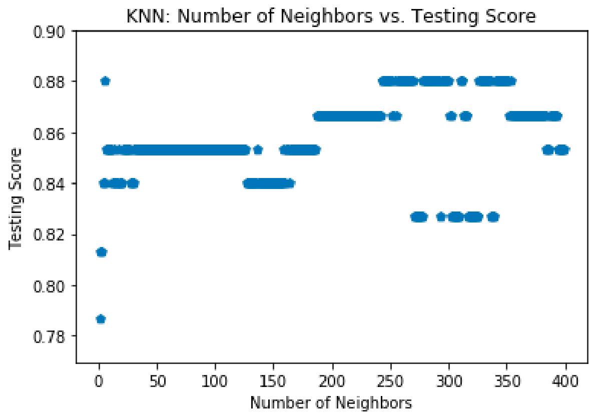




Analysis of knn with 70 neighbors, (highest accuracy):
Testing Score: 0.5247524752475248 and Training Score: 0.6207701283547258
Training Error: 0.3792298716452742
Testing Error: 0.4752475247524752

Data Set 2:





Testing Score: 0.88 and Training Score: 0.8729411764705882
Training Error: 0.12705882352941178
Testing Error: 0.12

Written Analysis: There does not seem to be a clear relationship between the number of neighbors and testing score, but the optimal value of k seems to be dependent on the number of parameters. Once again, KNN is most accurate on data set #2 by quite a bit. This could be due to two things. because data set #2 has fewer parameters, it is much less likely for the algorithm to over fit the data. This causes the algorithm to be more general. Secondly, this problem can be very easily deduced to an equation. Determining someone's chanced of admission based on some numbers is very predictable. Since KNN uses a weights, it is likely that the algorithm could have uncovered the relationship.

Conclusion: When I began, my initial contention was that neural networks would be the "best" algorithm and that problem # 1 would have better testing scores due to the fact that all algorithms were given more data to reason with. I learned that there is no "best" algorithm. Each algorithm's performance varies depending on the nature of the data. For example, both boosting algorithms caused a significant improvement in the decision tree's accuracy for data set 1 and a worsening in the accuracy for data set 2. This is because the boosting algorithms use the parameters of the problem to make many trees with which to decide on each data point with. On the other hand, Neural Networks caused excellent performance with Data Set 2 and abysmal performance with Data Set 1. This is likely due to the fact that neural networks create a system of weight and biases, so they work excellent on a problem that has somewhat of a clear mathematical relationship and not so well on a problem that can be influenced by many factors, not all of which are easily quantifiable. I found that Data Set 2 performed better in every instance. This is due to the fact that having too many parameters can often cause overffiting, especially with the decision tree algorithm. Also, predicting college admissions based on scores is much easier than predicting whether or not someone will like the song. There are hundreds of other factors the affect song preference that the Spotify API simply cannot capture like mood, time of say, etc. In conclusion, I was amazed at how accurately these algorithms can predict trends that humans struggle to or only think of subconsciously. Alos, it was interesting seeing how much the performace of each algorithm varies from data set to data set.