Monira Khan

CS4641: Machine Learning

February 2nd, 2019

**Supervised Learning**

The purpose of this report is to analyze and describe five different supervised learning techniques: decision tree learning, neural networks, boosting, support vector machines, and k-nearest neighbors. I used the python library scikit-learn in order to easily implement all of these classifiers. The two datasets that all classifiers were run on are listed below.

* **Breast cancer dataset:**

This dataset consists 699 instances of data obtained from breast cancer databases from the University of Wisconsin hospitals. The instances are in chronological order due to the periodical reporting of clinical cases. For the purposes of this assignment, the data is shuffled before split into training and testing subsets. The first attribute of the instance is a unique id number which I kept out of the training and testing subsets because it does not benefit the classifier–in fact this attribute hinders the classifier’s performance. Each attribute afterward correlates to an aspect of a breast tissue cell. The aspects are: clump thickness, uniformity of the cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitoses. The classification of the instance is either benign or malignant.

I think this dataset is interesting and extremely important because finding patterns and learning/analyzing this data could save lives by predicting cancers in patients before they get too dangerous. I believe that machine learning and computer science as a whole should be used for the greater good of helping the world, and studying cancer data is a very good start. I also found the relationship between the aspects of the cell and severity of cancer itself interesting.

* **Chess game dataset:**

This dataset follows a chess game of king and rook team against a king and pawn team. The database was generated and described by Alen Sharprio and supplied by the Turing Institute in Glasgow. The dataset is multivariate and categorical, but in order for the skit-learn library to classify the data, I had to convert the dataset into an integer dataset by setting the letter to a corresponding number. This dataset is composed of three thousand ninety-six instances with 36 attributes. The classification of the data is either white can win or white cannot win. The white team is the king and rook and the black team is the king and pawn team. The white is deemed to lose if the black pawn can safely advance. Each instance is a board-descriptions for this chess endgame. The 36 attributes correspond to the board. There is one board position per line.

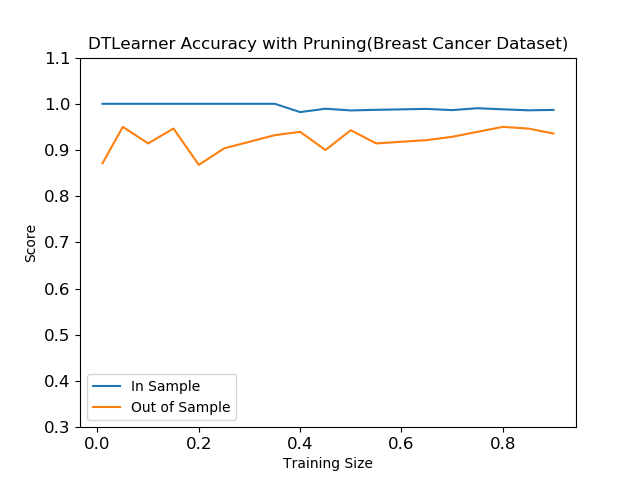
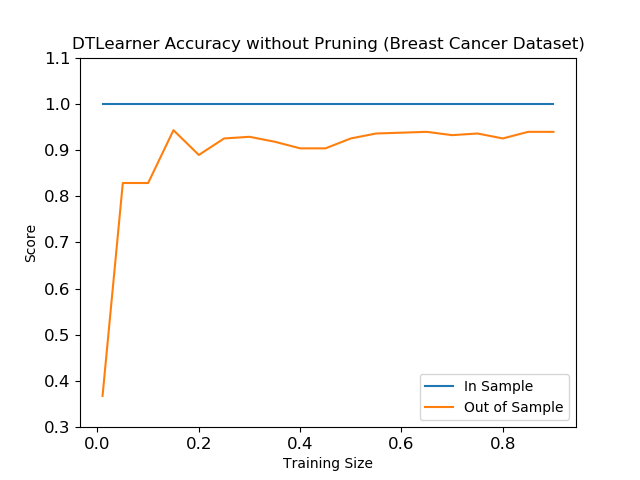
I personally found this database interesting because I have always been interested in the learning that took place when playing against a CPU on chess. Because I used to always play against the chess CPU, when I saw this dataset, I thought it would be very interesting to run classifiers on this type of data, because it gets me closer to creating my own machine learning chess bot.

All five algorithms’ hyperparameters were tuned with GridSearchCV in order to achieve optimal performance. With Grid Search, you feed the method a list of hyperparameters which it creates a grid and compares all of the algorithms performances according to the all the parameter combinations exhaustively. This results in an estimator especially tuned for the certain dataset.

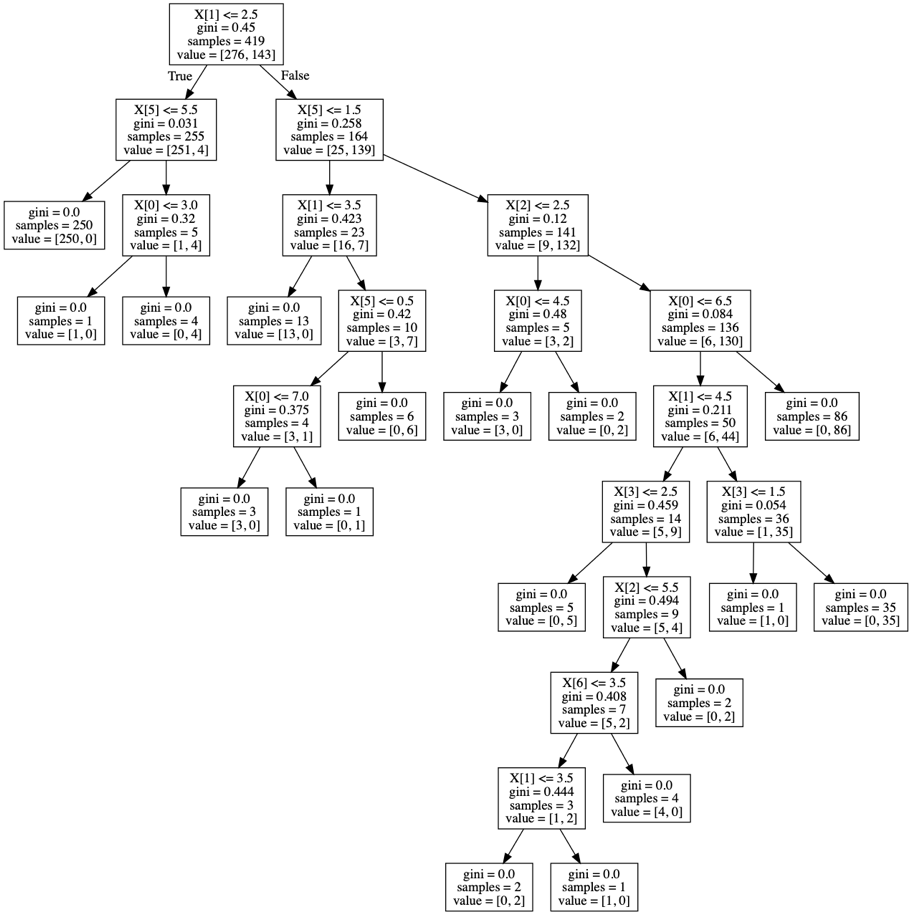
**Decision Trees:** Decision trees are a non-parametric classifier that attempts to classify data points by rules it infers from its attributes. The rules work like a game of 21-questions where asking one question creates a rule–canceling out many options that the X can be classified as. The algorithm is logarithmic in the number of data points used to train the tree by this method of splitting up the data at every node. A disadvantage of decision trees, though, is overfitting to the training data by creating overly complex trees, so the classifier cannot generalize to the testing data.

There were two ways that I have pruned the decision tree–using GridSearchCV and setting the max depth of the tree. Using the GridSearchCV is a form of pre-pruning the classifier. The model selection object takes parameters–max depth and max features–to compare and search for the optimized combination of the parameters for the training set. By choosing the best calibration of the parameters, the object is essentially pre-pruning the decision tree to be best fit for the data. This method resulted in the following learning curves for the

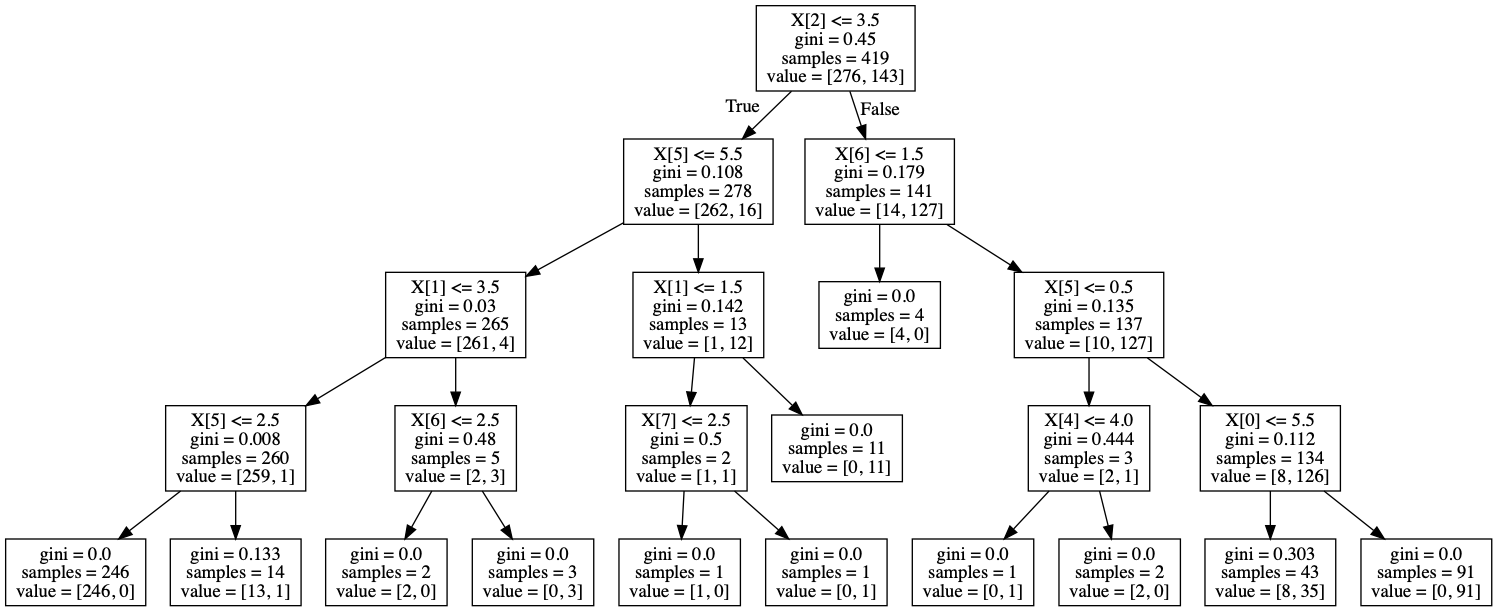
*breast cancer dataset*:



In the graph, without pruning, it can be seen that the Decision Tree Classifier’s accuracy is at a constant 100% because of **overfitting.** The learner is creating branches for every particular node and molding itself for the training data. Because of this, I expected the test data score to be much worse. To my surprise, the unpruned learner still performed well on the testing data–meaning the decision tree was able to generalize despite the fact that the classifier has completely trusted the training data. A reason why this is happening could be that the breast cancer dataset is easy for the decision tree to learn–the dataset only has 9 attributes making it a rather simple dataset. In order to prune the classifier, I lowered threshold for maximum depth of the tree and max number of features that the classifier can look when looking for the best split and let GridSearchCV choose the best combination of the parameters. This change forces the decision tree to be less focused on the training dataset and generalize well to the testing dataset. I expected the training score to suffer but the testing score to perform better because the graph should be able to generalize better. As can be seen in the graph above, the training score did indeed lower, but not by much, and the testing score is significant better than the unpruned tree with less data to train on. This makes me think that thee pruning did in fact work to help the classifier not conform completely and generalize better, but the dataset seems way to easy for the learner to learn. The impact of the pruning can also be seen in the visualization of the trees in Figure 1 and Figure 2. The pruned tree does not expand to the extent the the unpruned tree since the unpruned tree is delimited and interestingly the pruned tree is perfectly balanced.

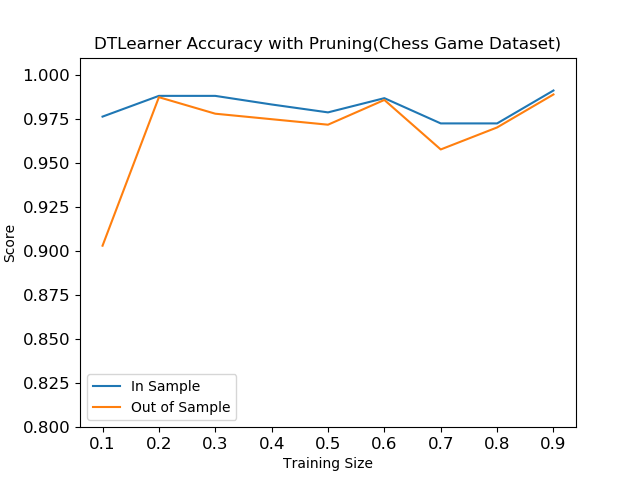
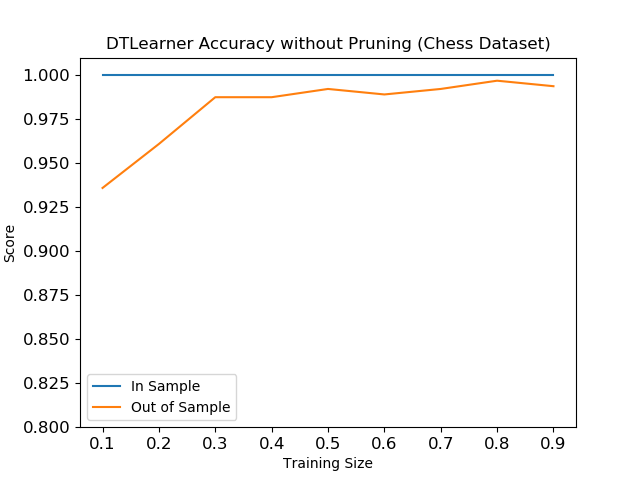


**Figure 1: Unpruned Decision Tree**



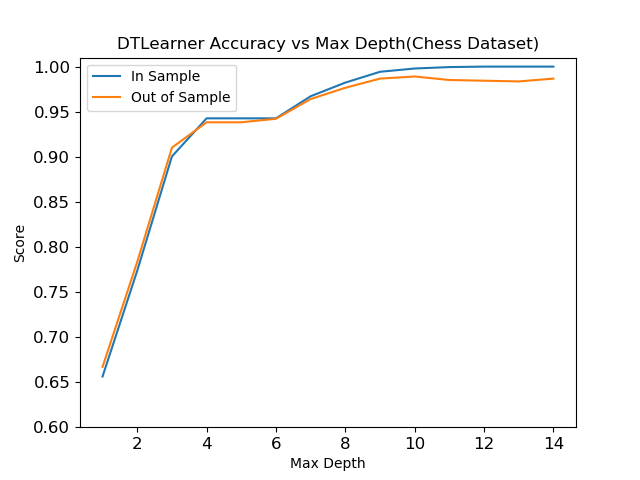
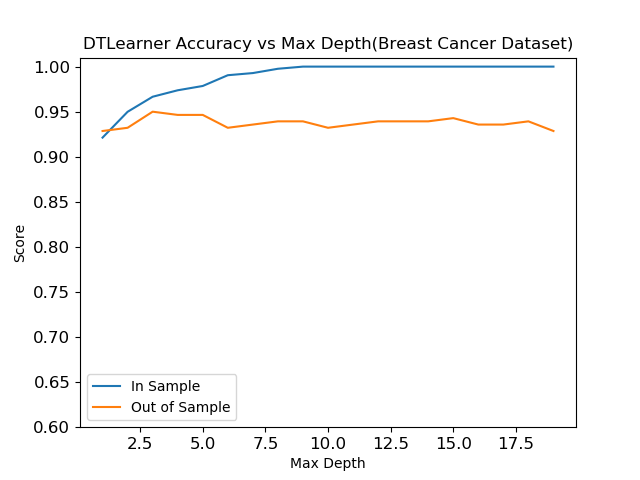
**Figure 2: Pruned Decision Tree**

Moving on to the chess game dataset, I ran the decision tree on the chess game dataset and produced the following learning curves:



With this dataset, you can see similar patterns to the breast cancer dataset’s learning curve. To my surprise, the unpruned decision tree still performed extremely well with the testing data despite the fact that the chess game dataset is way more complex than the breast cancer dataset. Although, the unpruned tree is still experiencing overfitting as can be seen by how the training accuracy is constantly at 100% while the testing accuracy is not going as well. In order to prune the classifier, I again lowered the max depth and max features for the decision tree. In the pruned tree, you can see that the training and testing score eventually converge.

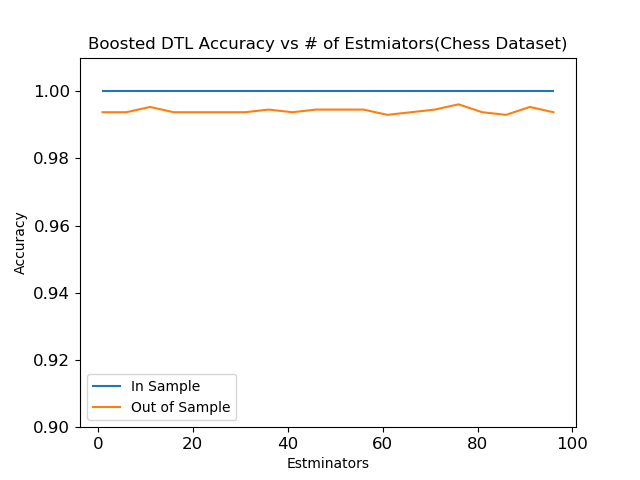
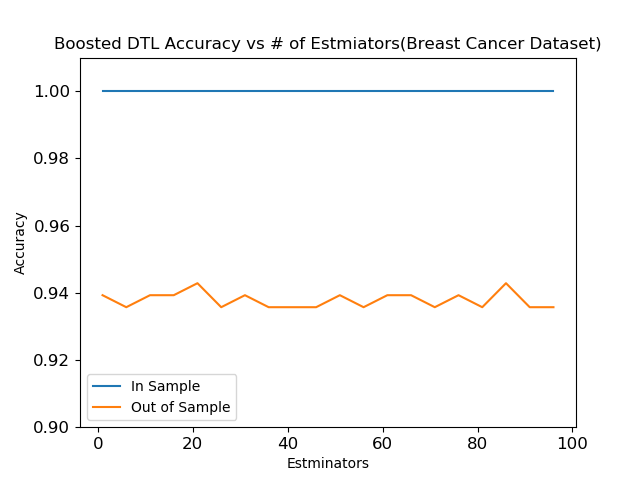
For experimental purposes, I also graphed the learning curve of the decision tree against max depth for both datasets, producing the following graphs:



It is interesting how max depth impacts the datasets in different ways: for the breast cancer dataset, the high the max depth goes, the training score is increasing, but the testing score is stabilizing–overfitting is visibly increasing–, while for the chess dataset, as max depth increases both training and testing increase together with no sign of overfitting. Although, I suspect if the max depth were to increase even more, the classifier would eventually overfit to the training data and testing score would decrease.

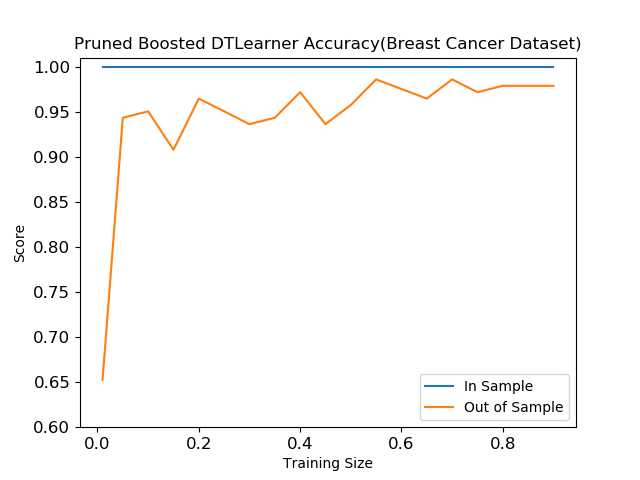
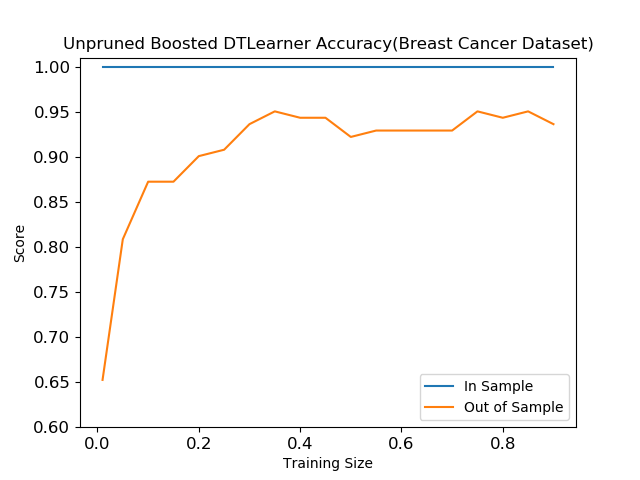
**Adaptive Boosting:** This method of learning fits one decision tree to the training set and then fits additional copies of the estimator to the dataset, but *adapts* to the data by adjusting the decision trees are that incorrectly classifying the data.

For the first experiment, I wanted to see if the number of estimators impacted the accuracy of the classifier. I graphed the accuracy of the classifier against number of estimators:

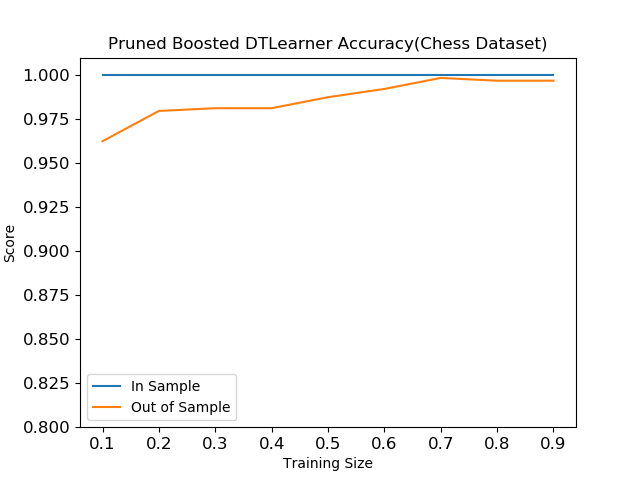
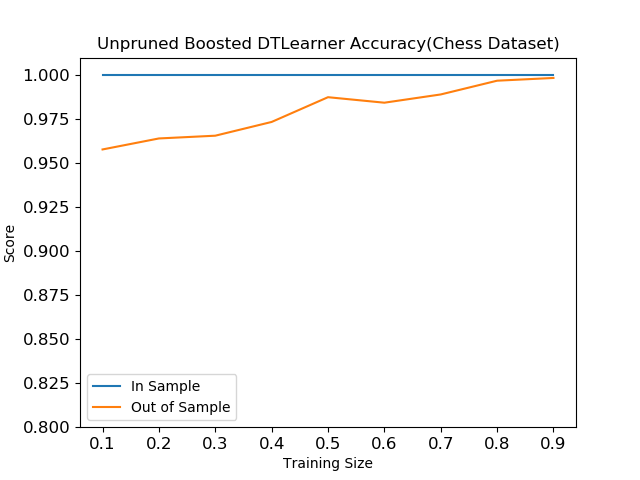


Interestingly, the number of estimators does not seem to help or even impact the testing score for the classifier. Another interesting thing to note is that the classifier is working extremely well on the chess dataset rather than the breast cancer dataset, despite the fact that the breast cancer dataset is more trivial than the chess one.

In order to prune the adaptive boosting learner, I lowered the number of estimators as the chosen hyperparameter. Using GridSearchCV, I lowered the number of estimators that the classifier could use and let the algorithm find the best number of estimators.

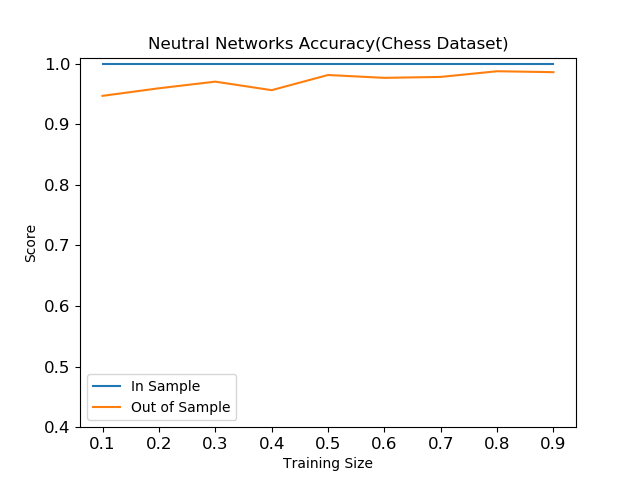
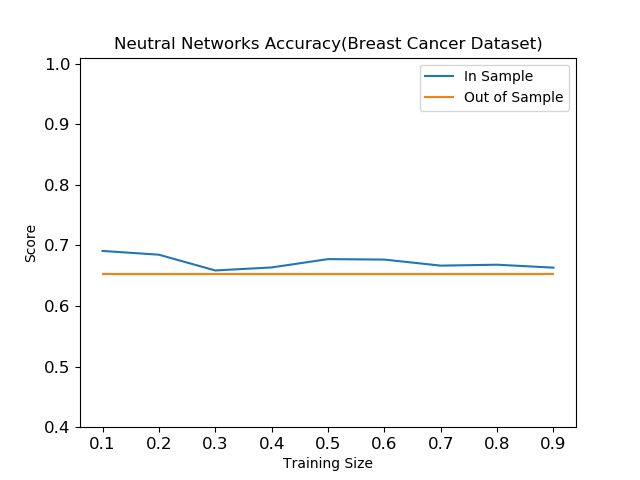


As seen from the graphs above, the pruned learner for the breast cancer dataset generally scored better for the testing data meaning that the classifier generalized better despite the training score being a consistent 100%.



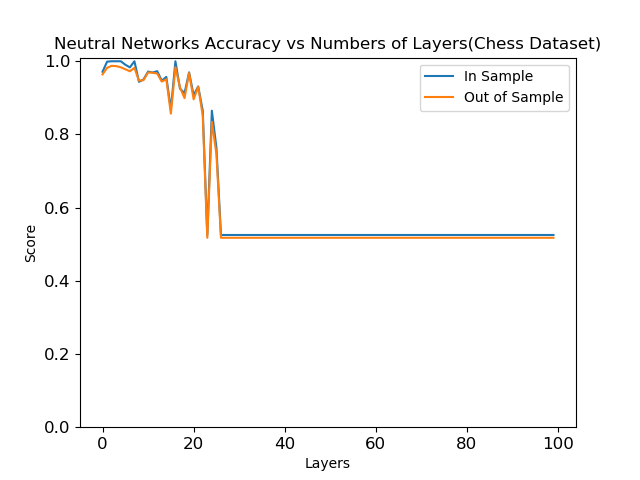
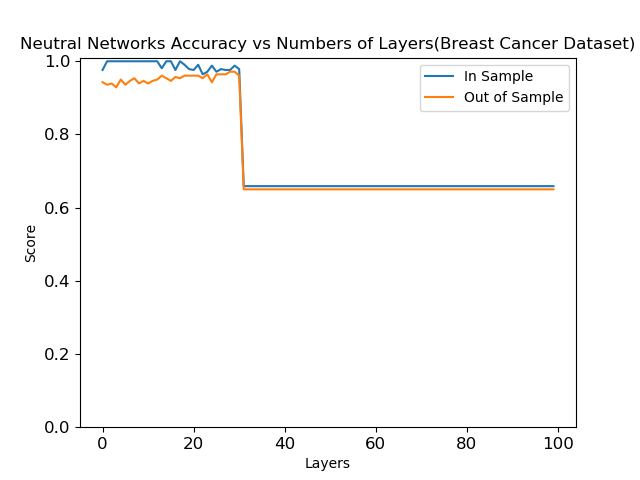
To prune for the chess dataset, I lowered the max depth and estimators to 50 while the unpruned dataset had max depth and estimators set to 150. Seeing no impact, I lowered the max depth to 10 for the pruned tree. It seems like pruning had no impact. This came to my surprise because the chess dataset is more complex than the breast cancer dataset, yet the unpruned boosted decision tree learner generalized very well. This may because of the nature of adaptive boosting–have any number of copies of the estimators will create a very powerful learner and almost eradicate overfitting.

**Neural Networks**: For the purposes of this assignment, I used scikit’s multilayer perceptron, MLPClassifier, which trains based on backpropagation–a batch gradient descent. The hyperparameters that can be tuned are number of layers, the activation function for the hidden layer, and the solver for weight optimization. I inputted a dictionary containing all of these hyperparameters and their possible values for GridSearchCV to find the best combination of them all to optimize the learner’s performance on the each of the training datasets.



For the breast cancer dataset, the neural networks learner did not perform the best. The GridSearchCV algorithm’s best classifier for the dataset has 181 layers. I have experimented and noticed that lowering the number of hidden layers from 200 to 100, the GridSearchCV algorithm selects 81 layers and performs exactly the same. The optimized activation function for the breast cancer dataset is logistic–the sigmoid function. The solver for the weight optimization that is best for the dataset is ‘adam’ which is a stochastic gradient based optimizer. For the chess dataset, the GridSearchCV algorithm chose 121 layers out for the range 200 layers. The optimized activation function for the chess dataset is tanh–the hyperbolic tan function that returns tanh(x)–and the best solver is ‘lbfgs’.

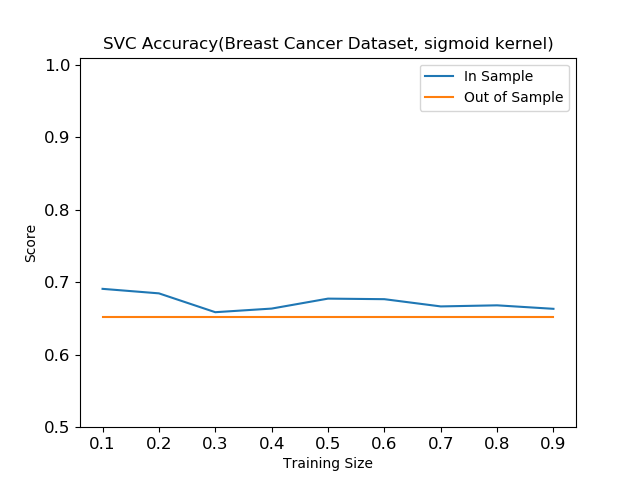
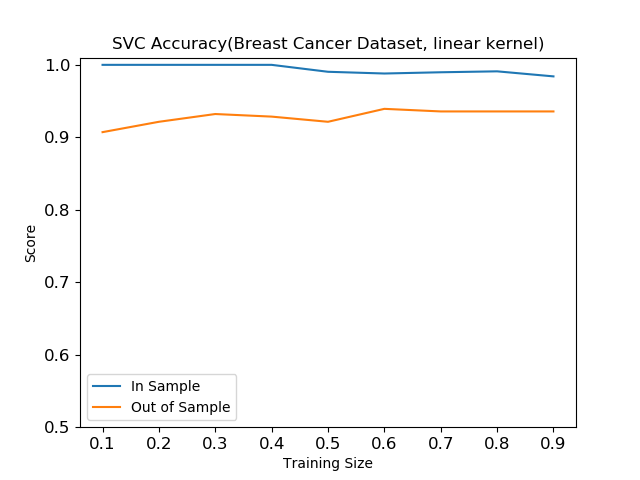
For the next experiment, I focused on the hyperparameter number of layers, producing the graphs below:



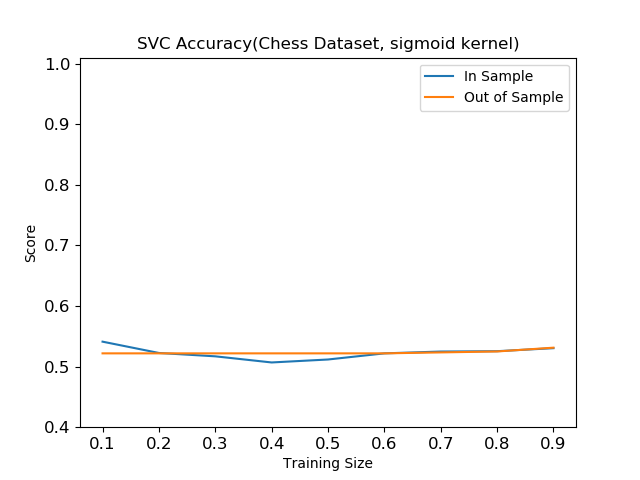
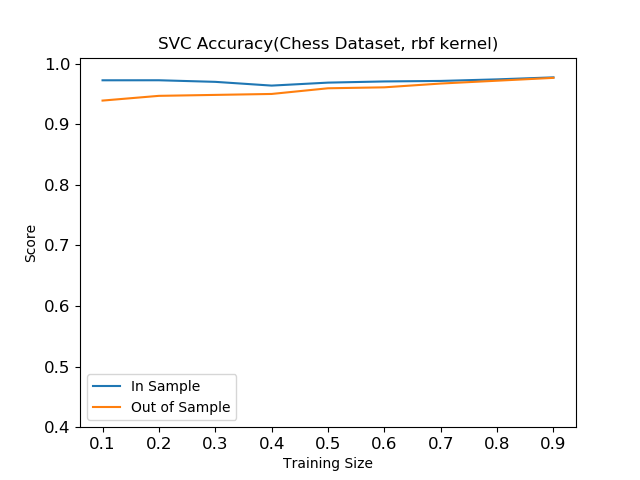
The impact of the number of layers is pretty much the same for both datasets. It seems like the classifier does extremely well with around 15 layers, but as it increases it begins to hurt the learner’s performance on both the training and testing set in almost the same way. In the breast cancer dataset, you can see how the testing score and training score eventually converge. It’s at that point where overfitting is minimized. After the scores converge though, on both datasets, the accuracy plummets, showing that too many layers do not help but harms the learner.

**Support Vector Machines:** SVM is a set of learning algorithms, but for the purposes of this assignment, we are focusing on the classification methods. This algorithm categorizes data by a specifically chosen kernel function that will map spaces of the data to a category. Using GridSearchCV, one can find the optimal kernel for each dataset.

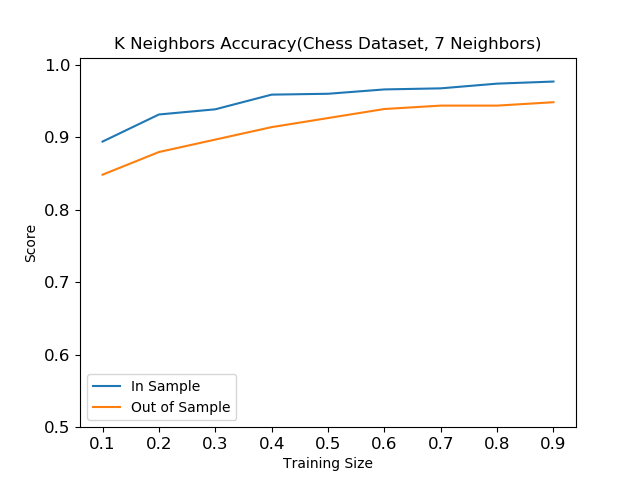
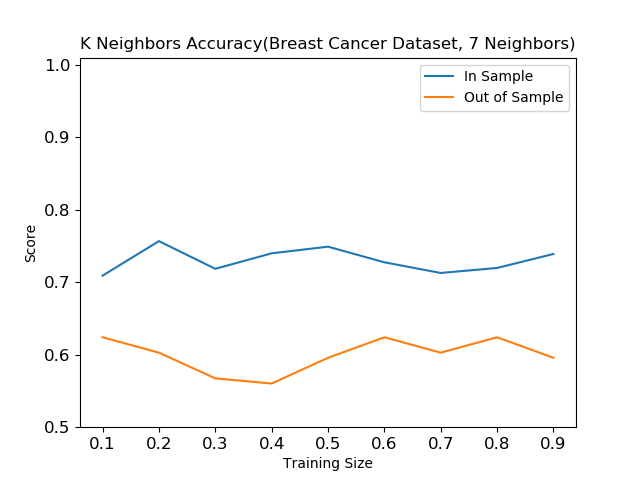
For the breast cancer dataset, the optimal kernel was the linear function. I also graphed the classifier on the sigmoid function in order to compare performances. As seen from the graphs below, the linear kernel did in fact perform extremely well compared to the non-optimal option for the dataset. This makes sense because the breast cancer dataset is quite trivial and the linear kernel is the simplest kernel. This choice suggests that there is a linear relationship between the data attributes in the breast cancer dataset. Although the testing accuracy never quite reached the training accuracy, the accuracy is still above 90% the entire learning curve.



For the chess dataset, the optimal kernel was the rbf function. Rbf stands for radial basis function suggesting that there is a circular branching relationship between the data attributes. For comparison's sake, I graph the classifier’s performance with the rbf function and with the sigmoid function. As can be seen in the graphs below, the rbf kernel performed extremely better than the sigmoid kernel. For the rbr kernel, you can see the testing and training learning curves converge around 90% training size. This is where overfitting is minimized, which is surprising because there is where it trains on the most data, therefore the chances of overfitting is maximized.:

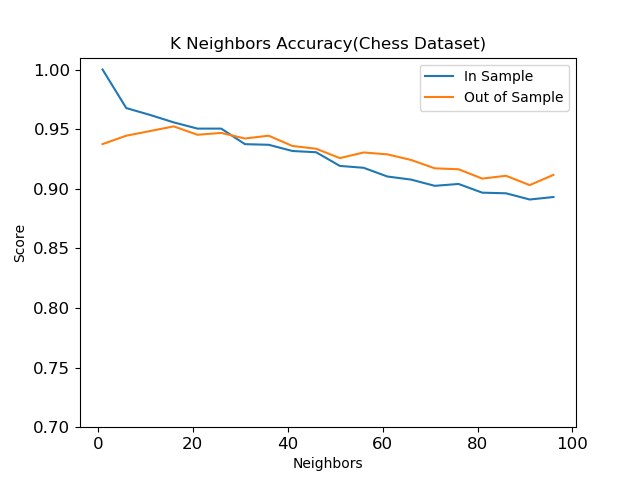
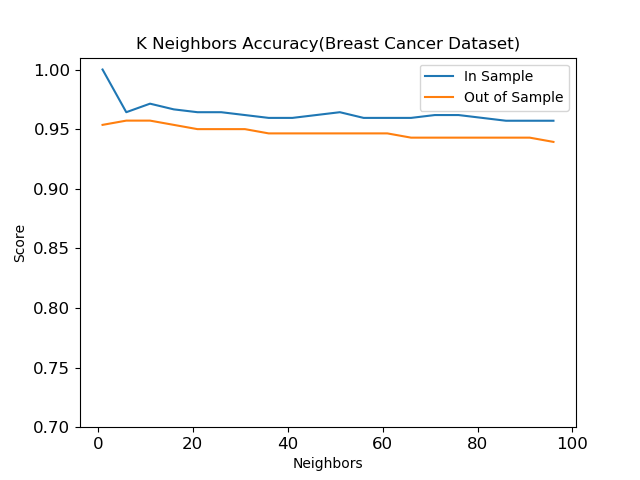


**K-Nearest Neighbors:** The K-nearest algorithm finds a predefined number of data points nearest to a new data point and creates a label from that information. K-nearest can be used for classification and regression–for the purposes of this assignment, KNearestClassifier from scikit was used. The hyperparameter that was used is the number of neighbors. The following graphs were produced, graphing the learning curve with the optimal amount of neighbors for each dataset:



For both algorithms, out of a range of 50 neighbors, 7 neighbors performed optimally on the training set. As can be seen in the graphs above, the algorithm has impacted both datasets differently. For the breast cancer dataset, the algorithm is very much overfitting since the testing and training learning curves are sloped in opposing directions while in the chess dataset, overfitting is less severe and the learning curves are both heading in the same direction. K Nearest Neighbors agrees more with the chess game dataset.

I also wanted to see the impact of increasing the number of neighbors on the accuracy of the learners, producing the following graphs:



The number of neighbors ceases to help accuracy for both datasets around 7 neighbors, as GridSearchCV suggested. For the breast cancer dataset, the score seems to just stabilize, while for the chess game dataset, the score is getting worse. To my surprise, for the chess dataset, the training score crosses the testing score at about 37 and consistently performed better.

**Conclusion:** Most of the experimenting with the algorithms did not perform the ways that I would have predicted. After analyzing the graphs that have been reported in this paper, I see that overall the chess dataset was easier for the classifiers to learn rather than the breast cancer dataset. This is very counterintuitive since the breast cancer dataset is the more trivial dataset.

For judging how well the algorithms performed against each other, I would rate it on how well the classifier generalized to the testing data. Overall, it seems from the graphs listed in this report, that Neural Networks generalized the best to new data. Though, Neural Networks also was one of the classifiers that took the longest to run. In a situation where time is valued over accuracy, one should not choose neural networks and go with another classifier.

According to the table below, for the breast cancer dataset, k-nearest neighbor has the best testing score for the optimized learner with 60% training data. This came as a surprise because according to the performance graphs, k-nearest did not look the best for this dataset.

For the chess game dataset, adaptive boosting worked the best at almost 100% testing accuracy, with Neural Networks as a close second.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Pruned Decision Tree | Adaptive Boosting | Neural Networks | SVM | K-Nearest Neighbor |
| Breast Cancer Dataset | 90.7% | 95.7% | 95% | 94.2% | 96% |
| Chess Game Dataset | 94.9% | 99.2% | 98% | 96.7% | 94.2% |

**Testing Scores from All Classifiers with .60 Training Data**