Pranshav Thakkar

GT username: pthakkar7

Supervised Learning

For this assignment, we were asked to formulate two different, interesting classification problems based on two distinct sets of data. I created my classification problems based on two datasets that I acquired from UCI's Machine Learning Repository. The first problem concerns the game of tic-tac-toe. The data has 9 features, each representing a position on the nine-block grid that tic-tac-toe is played on. These positions can either be filled with an 'x', an 'o', or be left blank. The problem is to figure out whether 'x' will win or lose when it starts first. The data represents the different scenarios that are possible when 'x' starts first, and the objective is to classify each 'game' with a label of 'positive', for a win by 'x', or a label of 'negative', which indicates a loss by 'x'. The second problem is based on chess, on a situation where white has their king and a rook left, while black has their king and a pawn left. The data has 36 features which represent different states of the board, and the goal of the problem is to determine whether white can win the game or not. The result will be a classification of either 'win', or 'nowin'.

I used the scikit-learn library to use the machine learning algorithms on my two datasets, to solve the classification problems. Both of the datasets had data in the form of letters (text), which scikit-learn does not process when training data, so I iterated through the data and assigned an integer to replace each distinct letter. I believe that the label encoding function that scikit-learn provides has the same overall function.

The tic-tac-toe dataset is a relatively small dataset, with just under 1000 instances, but I believe that it is interesting because it is a complex problem to understand and learn for the machine. The algorithms must understand that the nine features represent nine different places that 'x's and 'o's can be played, and don't necessarily have to be played in all the places. They have to learn that in this case, a player only wins if he gets three in a row, in any configuration, before the other player. It would be interesting to see the results that the various algorithms obtain for solving this problem, since it lets us observe what kind of procedure works best for solving the problem. I think it would be worthwhile to note whether a node-based algorithm like a Decision Tree Classifier or a Neural Net delivers a better accuracy than a margin based algorithm like a Linear SVM or a vote based algorithm like KNN. The chess dataset is larger, with over 3000 instances, so it will be intriguing to see how that affects the performance of the algorithms. The classification problem is similar in that we must determine a win or a loss, but the chess problem has a much larger scope in terms with the number of features and instances as compared to tic-tac-toe. Comparing the algorithms' results when tested on a dataset with more instances could tell us a lot about how much the amount of training data matters in learning.

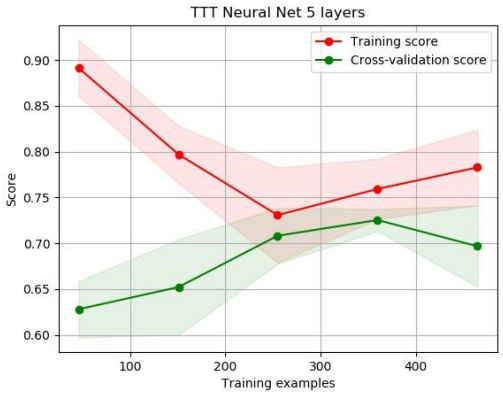
Training/Testing Scores

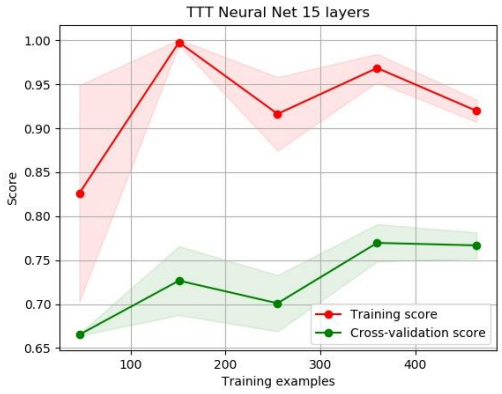
Tic-Tac-Toe Dataset:

1. Decision Tree

Testing accuracy score: 0.8294573643410853

2. Neural Networks

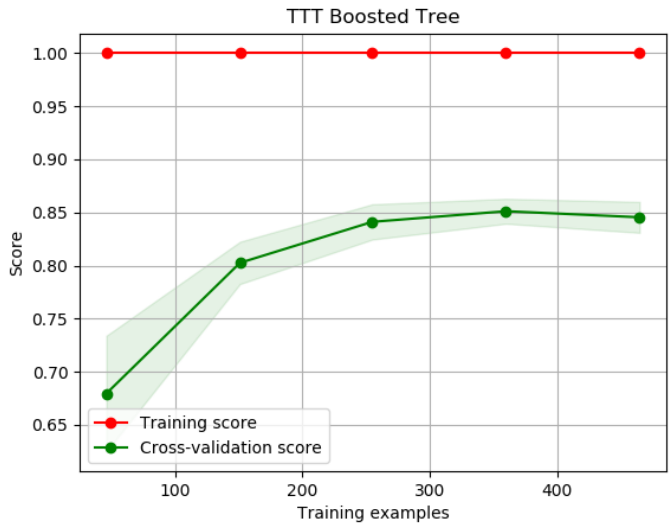




Testing accuracy score: 0.7054263565891473 (5 layers)

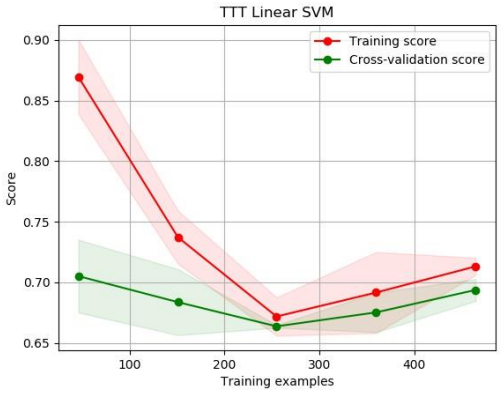
0.7635658914728682 (15 layers)

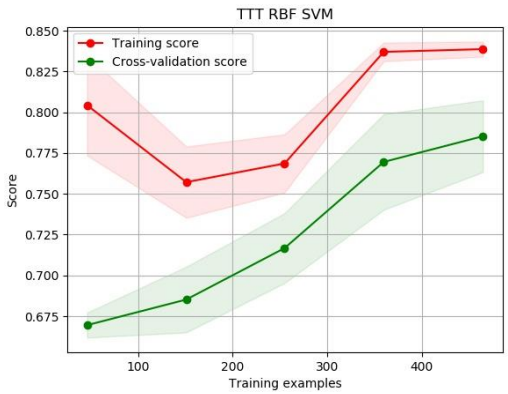
3. Boosted Tree



Testing accuracy score: 0.8062015503875969

4. SVM

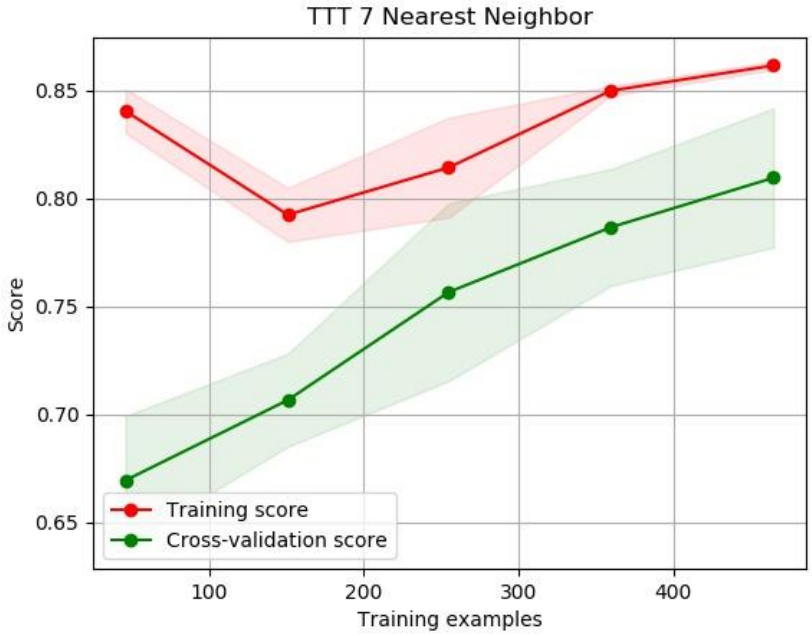




Testing accuracy score: 0.624031007751938 (Linear)

0.7713178294573644 (RBF)

5. KNN



Testing Score accuracy: 0.7635658914728682 (3-NN)

0.7790697674418605 (5-NN)

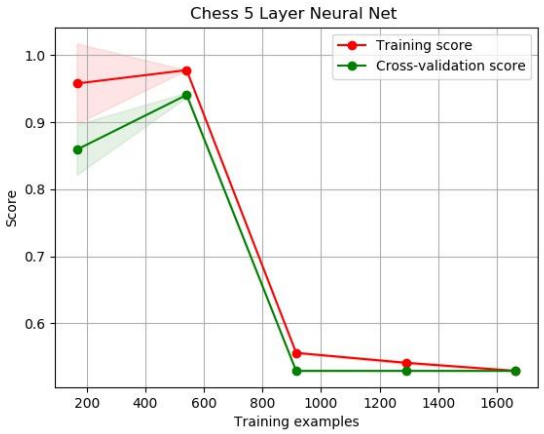
0.7558139534883721(7-NN)

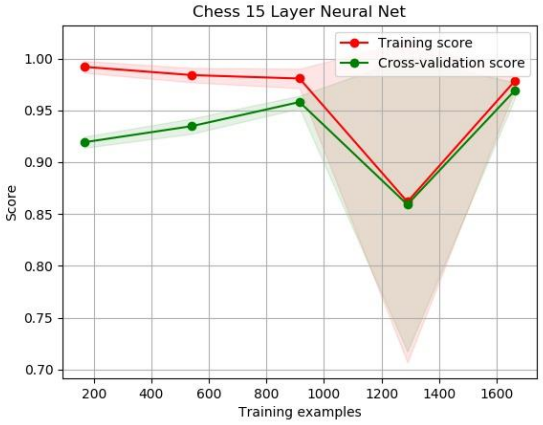
Chess Dataset:

1. Decision Tree

Testing Score accuracy: 0.9841954022988506

2. Neural Net

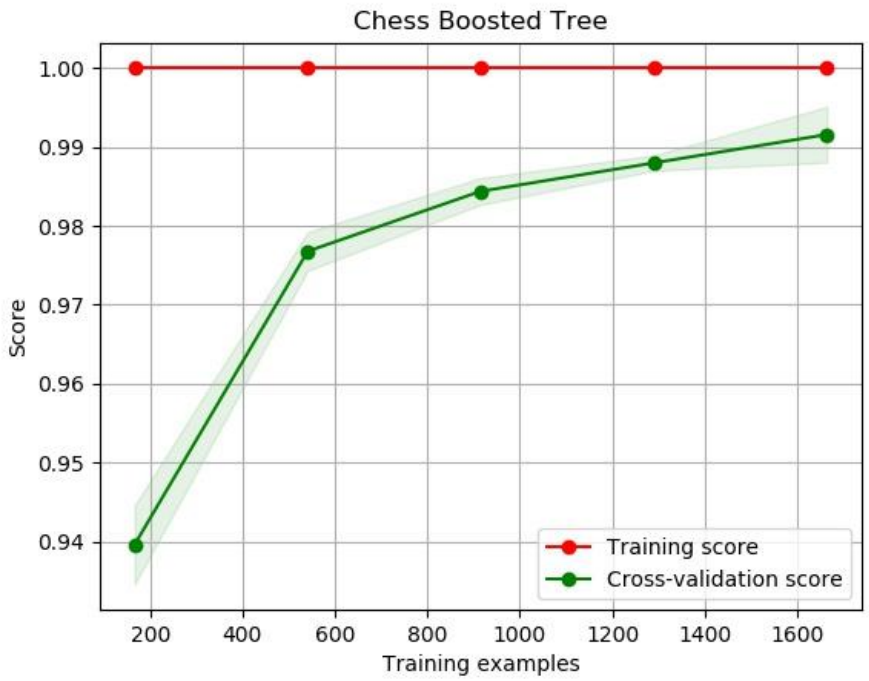




Testing Score accuracy: 0.49712643678160917 (5 Layers)

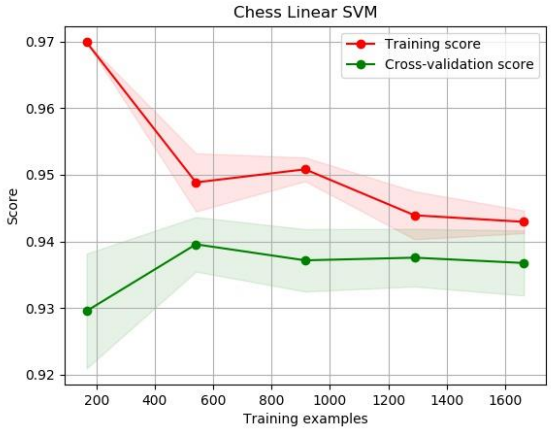
0.9698275862068966 (15 Layers)

3. Boosted Tree



Testing Score accuracy: 0.9870689655172413

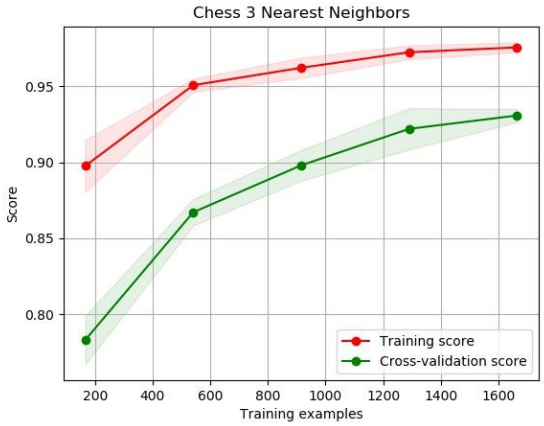
4. SVM

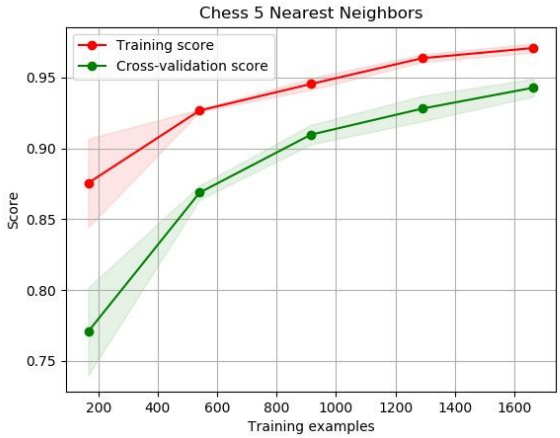


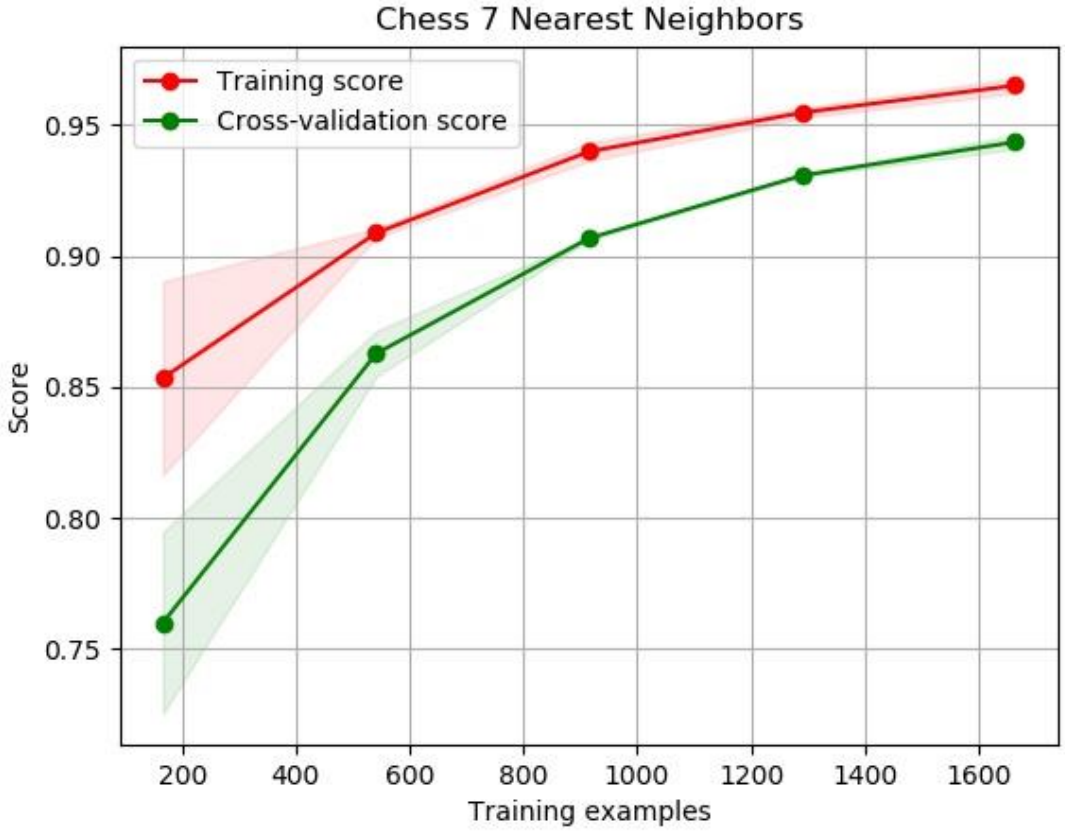
Testing Score accuracy: 0.9568965517241379 (Linear)

0.9454022988505747 (RBF)

5. KNN







Test Score accuracy: 0.9454022988505747 (3-NN)

0.9597701149425287 (5-NN)

0.9511494252873564 (7-NN)

Analysis

For both the problems, Decision Trees and Boosted Decision Trees gave the highest testing accuracy score. I feel that that result makes sense, as the problems I had corresponded well with how Decision Trees work. Making decisions based on the values of the features for both datasets leads you one step closer to the final classification, which is a win or no win. For example, “Is there an ‘x’ in this position or not ?”. The fact that the problems were both binary classifications may have helped produce this outcome, and the fact that the Boosted Tree supports the values of the Decision Tree lowers the possibility of the result being a product of overfitting.

The Linear SVM had the lowest test accuracy score for the tic-tac-toe dataset and there was a large difference between it and the RBF SVM. I believe this to be a result of less data instances being available as the two SVM’s had similar scores for the chess dataset. This result seems to be skewed because of sample size and the RBF SVM could be overfitted for the tic-tac-toe dataset.

K-nearest-neighbors performed relatively well for both problems and produced the median scores for both datasets. Out of the 3, 5, and 7 nearest neighbors, the 5-NN algorithm produced the best test score accuracy for both problems.

Neural Nets with 15 hidden layers performed well in terms of test score accuracy for both datasets while the Neural Nets with 5 hidden layers performed towards the lower end of scores. However, there was not quite a significant difference between the two for the tic-tac-toe dataset, while the 5 hidden layer Net for the chess problem seems to be an outlier. It provided scores close to half of all the other algorithms and had a test score accuracy of around 0.49. I believe that 5 hidden layers may not have been a deep enough Net for the larger dataset, and that the results for the smaller dataset are biased on the number of instances. The best thing to do would be to run the algorithm with many more variants of hidden layers to check whether this assumption is correct.

Again, I believe that the classification problems I designed were well-suited towards Decision Trees, and that is why they had the best performance. I also believe that the size of the datasets was a clear indicator for the difference in results between the problems. If the tic-tac-toe dataset had more instances, we might have seen results that were similar to the chess problem. However, it may just be that the problem itself is harder than the situation with chess, and instances are not the only factor. The fact that only more data can help us understand the problem better (and a little more fine tuning of the parameters) is what makes the problem so interesting.

I think that more time spent experimenting with parameters would help the algorithms perform better. I used a form of cross validation to help me pick the parameters, but I believe that spending more time into a more sophisticated version of cross validation, like Grid Search, to tune the hyper-parameters would have been a good investment of time. That would have helped

me use the best versions of the algorithms possible, and most likely would have resulted in the best performance.

In terms of clock time, most of the algorithms performed very quickly for the tic-tac-toe problem, being that it was a smaller dataset. SVMs and Decision trees gave outputs instantaneously, while Neural Nets and KNN took a few seconds at max. The chess problem took a little longer, which was expected as the dataset was more than three times the size of the former. Here, I noticed that Decision trees and Neural Nets took the least amount of time, being a few seconds each while SVMs started to slow down and took more than 5 seconds. KNNs were again the slowest, taking the same time as SVMs in this case.

For my Decision Trees, I used max depth as the form of pruning and set a max depth of 10 nodes for the base Decision Trees. For my Boosted Trees, I was more aggressive with my pruning and set a max depth of 5 nodes. It seems that the 50 iterations of weak learners that I used for the Boosted Trees compensated for the aggressive pruning, as the performance of the Boosted Trees did not suffer; they were still better than the base Trees. I believe I attempted to vary the max depth value, but I did not record the data which could have been useful in formulating a more concrete argument.

Finally, I believe that Decision Trees were the best algorithm because they performed the best in terms of all score accuracies and also because they seemed to be a good fit for the classification problems that I had created. Neural Nets were the second best algorithm to me and I believe that with more instances, they could have ended up being faster and more accurate than Decision Trees, as they are also well suited towards the problem. Essentially, I believe that is the biggest question to ask whenever we start a machine learning problem: what kind of algorithm would suit the data the best? Because at the end of the day, that is what will make the difference in terms of speed and accuracy: the domain knowledge that you must have that lets you choose the best algorithm to use. After all, there is NO free lunch.