I have previously taken the class. I used different data sets but wanted to add this warning in case anything is triggered in the plagiarism detector.

Assignment 1 – CS7641

Alan Borlie (aborle3)

Data Sets

For this project, I choose to use Python and the Sklearn library for all machine learning experiments. I automated as much of the process as I could, as I looked at 7 separate data sets before deciding of the two data sets discussed below, so this approach allowed me to quickly look at multiple sets to begin to gauge if they were interesting. I also leveraged the Mathplotlib library for all graphs displayed in the following sections.

The first data set I selected was for classifying beans, and it contained over 13,500 records. This dataset came from the UCI Machine Learning Repository. This data set was made up of 16 numeric features relating to the size and shape of the various beans. The beans could be classified into one of 7 classes (there was an 8th class, however there were only 4 examples for the entire data set, so I removed that class and its results from the list. The data was somewhat unbalanced, as the lowest class was only represented in about 4% of the results whereas the most prevalent was in about 26% of the results.

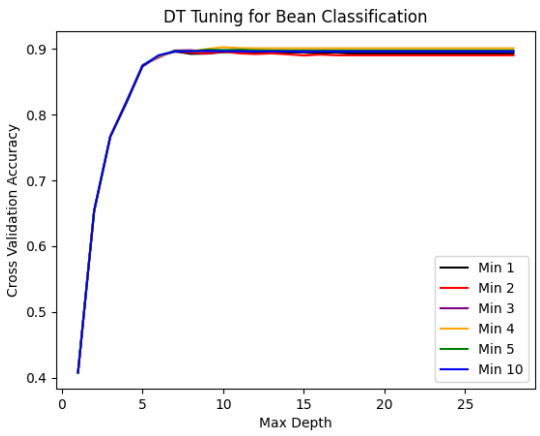
The second data set was designed to help predict phishing attacks and contained over 14,000 records. This dataset came from Kaggle. The dataset was made up of 26 numeric features that were integer values in the set of -1, 0 and 1. This data set had only two classifications, phishing or not phishing. This data set was almost perfectly balanced, with only 5 more examples of phishing than non-phishing.

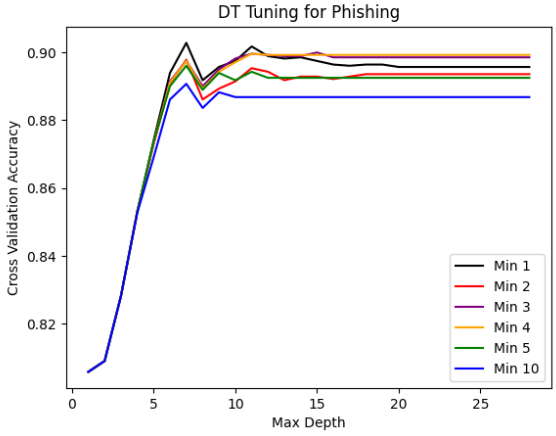
While I will go into further discussions in the following sections, I did find these data sets an interesting complement to each other. The first major reason was that even though they are both categorical in nature, one is binary, whereas the other one has 7 possible classifications. The tuning processes were also different for each set, as scaling had to be considered for the bean classification data set for some of the algorithms whereas the phishing dataset didn't need this additional step. There was a variety in which dataset worked better on which algorithm, which I will get into further when discussing the various algorithms and how they performed for each set.

On a final note I leveraged the accuracy score method of the Sklearn library for comparing all of the algorithms. While looking at the confusion matrices for each of the algorithms, I didn't see anything that would tell me that accuracy was not a good metric to use to measure the scores.

Decision Trees

Tuning the Model

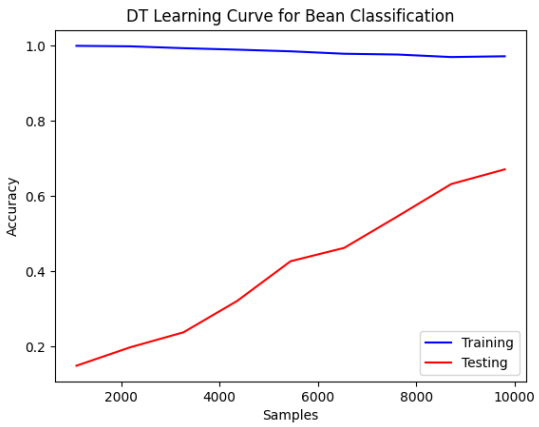


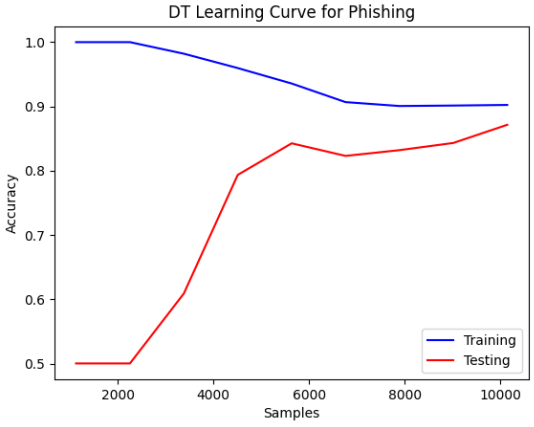


For the decision tree algorithm, I selected max depth, to handle the pruning requirement, and minimum samples split hyper parameters for tuning the decision tree. For the bean classification problem, there wasn't much difference between the values selected for the minimum samples split and the depth of the tree, as most runs of the algorithm produced the same graph, with the best pruned tree having a depth of 9 with a minimum sample split per leaf of 4. This tree scored 90% accuracy with cross validation testing.

For the phishing problem, there was a bit more difference in accuracy when changing the minimum sample split hyperparameter, as the best tree generated was nearly 2% more accurate with cross validation testing. This tree had a depth of 6 and a split of 1. The graphs above show the accuracy based on the selected hyper parameters.

Learning rate





Looking at the learning curves for the two data sets, we can see that neither set was able to converge with the number of examples provided. This does indicate that both data sets could have benefited from more examples. This was more evident with the bean classification data set, as it was much farther from

converging when compared to the phishing data set. This is most likely due to having multiple classifications compared to the binary classification for the phishing example.

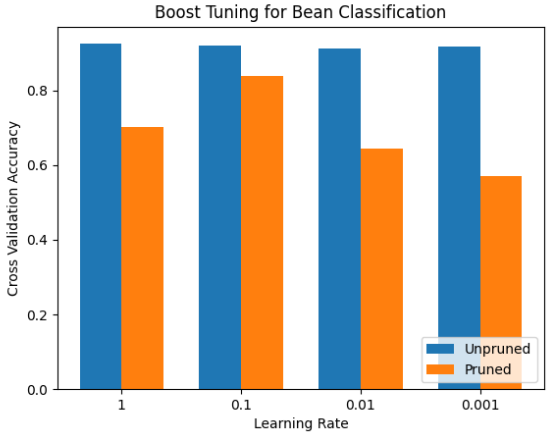
Results

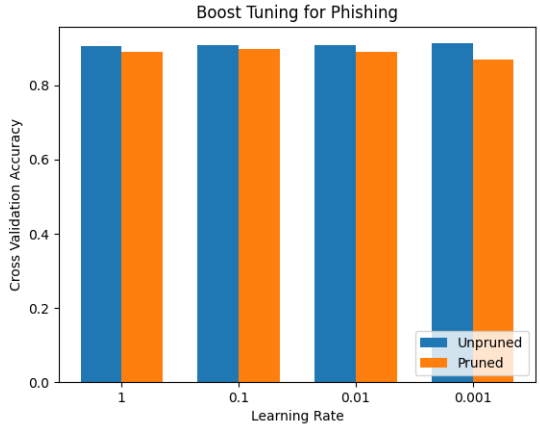
When training the bean classification decision tree with the optimal model, I was able to obtain a 95% training accuracy and 89.98% testing accuracy. Similarly, I was able to obtain a 90% training accuracy and 89.75% testing accuracy for the phishing data set.

I also generated the unpruned tree for both examples to see how many nodes were saved from pruning the tree. For the beans data set, the unpruned tree contained 387 nodes whereas the pruned version only contains 195. This is a very significant decrease in the complexity of the decision tree. Similarly, the unpruned tree for the phishing dataset contained 593 nodes, whereas the pruned tree only contained 91 nodes. In both instances, the effect on the accuracy was minimal and based on the testing accuracy scores, there didn't appear to be any overfitting of the trained model.

Boosting

Tuning the Model



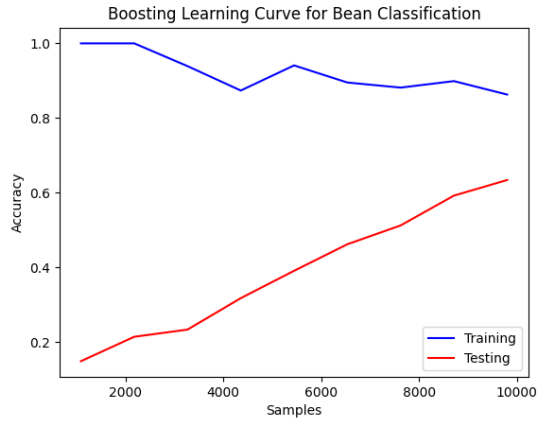


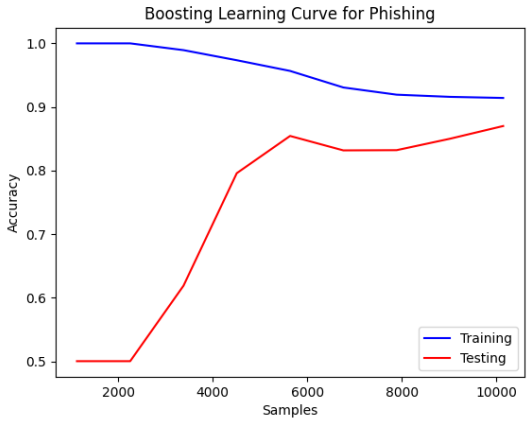
For boosting, I took the trees that were generated as part of the decision tree section and attempted to improve their performance. I selected multiple learning rates as one of the hyperparameters. I decided to use a second, more pruned tree as a second hyperparameter to see if the algorithm would do better with a more aggressively pruned tree.

For the bean classification problem, there was a huge difference between the pruned tree and the unpruned tree in regards to cross validation accuracy scores as seen in the bar graph above. The overall best result was for original tree. With a learning rate value of 1 this algorithm was able to achieve a 92.4% accuracy. When looking at the pruned tree, which had a max depth of only 2, I was still able to obtain a 83.9% accuracy with a learning rate of 0.1.

For the phishing problem tree, there was less difference between the pruned and unpruned trees. The unpruned tree was able to obtain an accuracy of 91.3% with a learning rate of 0.001. The pruned tree, which was limited to a depth of 2, was able to obtain an accuracy of 89.8% with a learning rate of 0.1.

Learning rate





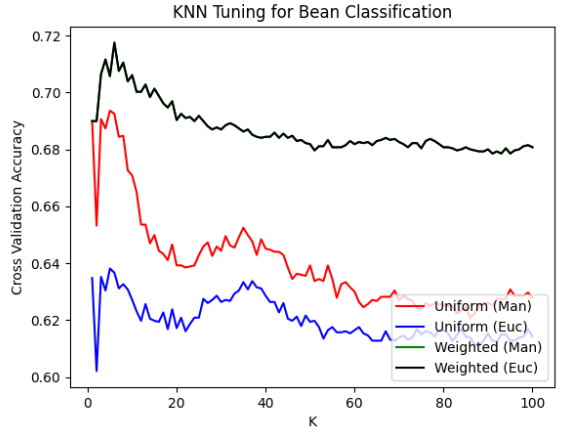
For the learning curves, the graphs are nearly identical to the decision tree learning curves. This is to be expected as I used a more aggressively pruned tree, but due to how boosting functions it was able to achieve the same results using a much smaller tree as it was able to consistently get better on the problems it struggled with.

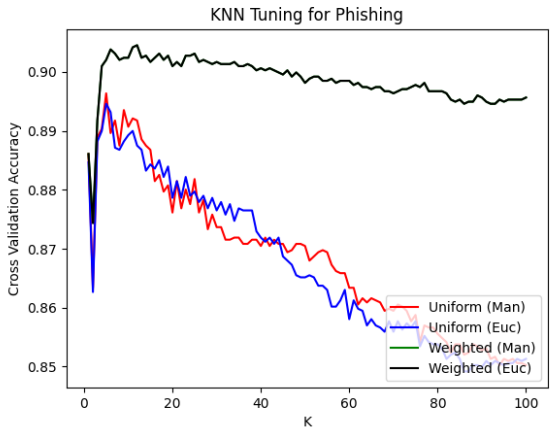
Results

The aggressively pruned bean classification tree was able to achieve 89% training accuracy and 88% testing accuracy. This is slightly less than the base decision tree was able to obtain, and likely could have been improved with more testing examples or a slightly less aggressively pruned tree. The phishing dataset had a 92% training accuracy and a 90% testing accuracy. This was a slight decrease in training accuracy but a slight increase in testing accuracy over the base decision tree. This did show that for this data set boosting was able to improve accuracy and reduce overfitting.

K Nearest Neighbor

Tuning the Model





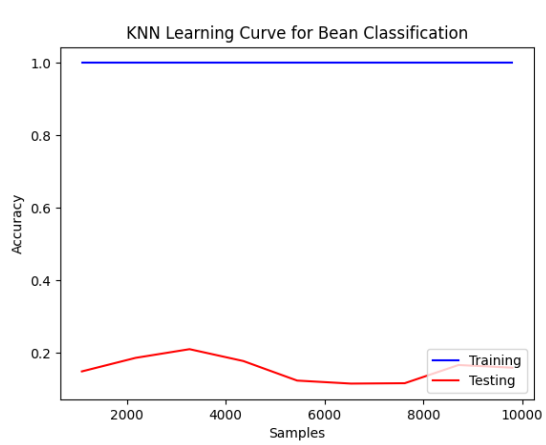
For the k-nearest neighbor algorithm, I looked at 3 separate hyperparameter values. The first was the value to use for k. The second was the distance formula, checking Manhattan and Euclidean distances. The final one was the weights for k, using both uniform as well as distance. For both algorithms, we can

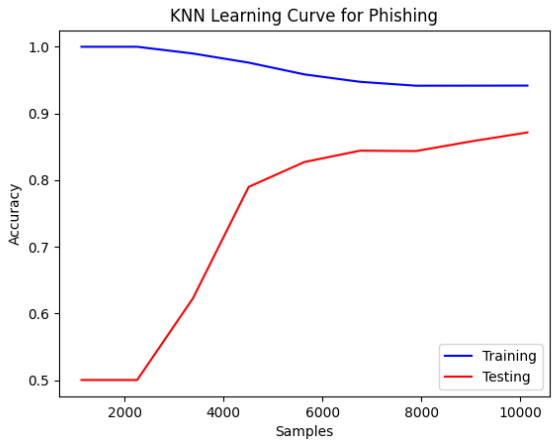
see that as we increase K, the accuracy is increased and overfitting is reduced in the cross validation accuracy score. This occurs until an apex is reached, then from that point forward increases in K begin to decrease the accuracy of the algorithm.

For the bean classification dataset, the adding distance-based weights to the values of k greatly improved the accuracy. This indicates there were more clusters of points near each other. There wasn't much difference between the Euclidean and Manhattan distance formulas, the best value of k was 6. This only produced an accuracy of 71%, the lowest of all algorithms.

The phishing dataset did much better, as the best accuracy obtained was 90% with a value of 12 for k. In this instance, there was only a negligible increase when using the weighted distance values over the uniform ones, indicating there probably was a bit more of a uniform spread to the data. Also, there was negligible difference between the two different distance formulas.

Learning rate





The learning rates show that the bean classification problem would have greatly benefitted from an increase in training examples, as it was not even close to converging. The Phishing dataset did better, and with a bit more examples looks like it was on pace to converge. I did find this behavior to be a bit strange as there are less features in the bean classification data set when compared to the phishing data set. Due to the curse of dimensionality, I would have expected the phishing dataset to have required more data.

Results

For the bean classification problem, the training accuracy was 100%, where as the testing accuracy was only 73.5%. This points to an issue with overfitting, and most likely goes back to not having enough training data for this particular problem. This was the best rate I could get based on the tuning parameters for this problem.

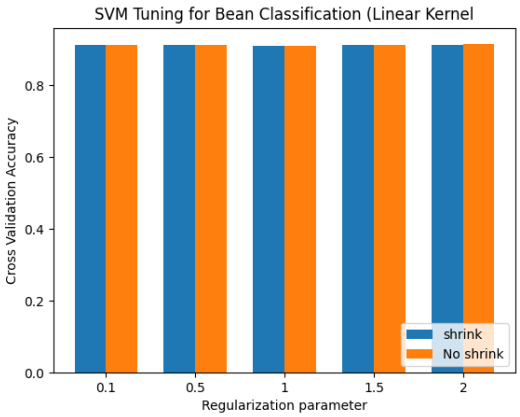
For the phishing data set I was able to obtain 95% training accuracy and nearly 91% testing accuracy. This dataset worked much better for the K-nearest neighbor algorithm.

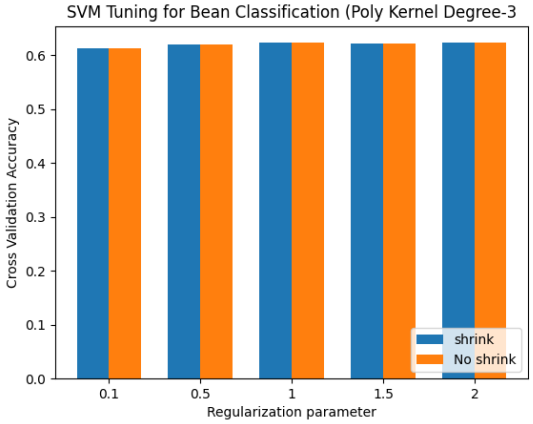
Support Vector Machines

Tuning the Model

For the support vector machine hyperparameters, I selected the regularization parameter and the shrinking hyperparameter. While the shrinking hyperparameter had minimal, if any impact on the overall accuracy score, it had a huge impact on the wall clock performance of the algorithm, often reducing runtimes by 90% or greater for the bean classification data set and 25% or greater for the phishing data set. For both data sets, I used the linear and polynomial kernels, with multiple degrees for the polynomial kernel.

Bean Classification Data Set

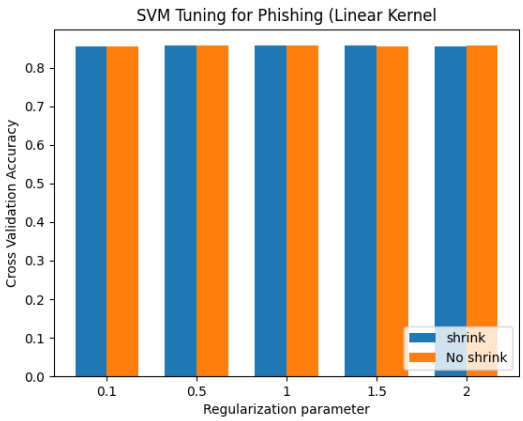


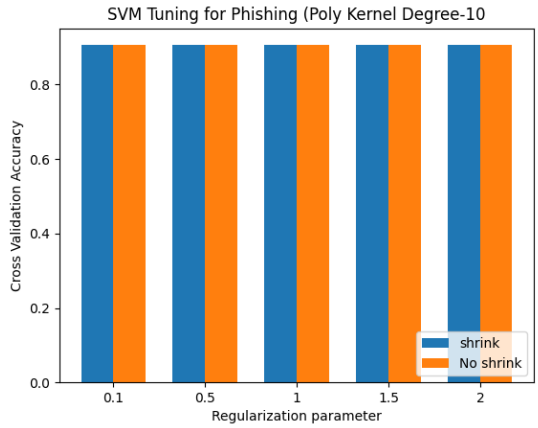


I only am displaying the graph for the 3rd degree polynomial as it performed best. The table below reflects the best cross validation for all the kernel functions used. This dataset did best with the linear kernel. The polynomial kernel consistently performed worse as the degree was increased. This is to be expected as the degree is exponentially increasing an inefficient kernel.

	Linear	**Poly (3rd)**	**Poly (4th)**	**Poly (5th)**	**Poly (10th)**
C Value:	2	1.5	1.5	1.5	1.5
Shrinking	No Shrink	Shrink	Shrink	Shrink	Shrink
Accuracy	91.477%	62.269%	61.865%	60.983%	57.457%

Phishing Data Set

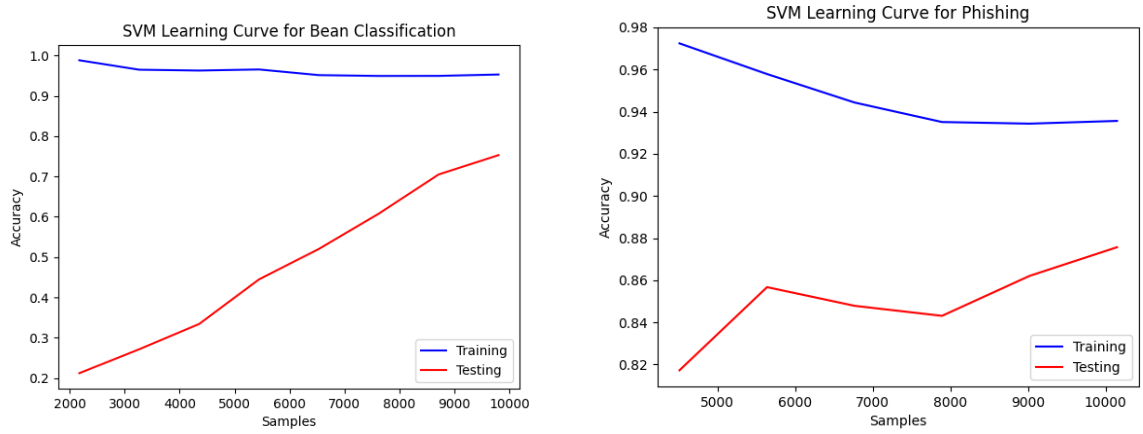




For the phishing data set, I am displaying the 10th degree polynomial as it performed the best. In this instance, the polynomial kernel was the more accurate kernel. Because of this, as I increased the degree of the kernel, the accuracy improved due to the exponential nature of the polynomial kernel function. The table below summarizes the finding for the phishing data set.

	Linear	Poly (3rd)	Poly (4th)	Poly (5th)	Poly (10th)
C Value:	2	2	1.5	2	2
Shrinking	No Shrink	Shrink	Shrink	Shrink	Shrink
Accuracy	85.700%	89.603%	89.744%	90.134%	90.667%

Learning rate



For the bean classification data set, I generated the learning curve above using the best tuned model, which was a linear kernel with a regularization parameter of 2 and no shrinking heuristic applied. This model was only able to get to 70% accuracy on the testing set, so there is still some overfitting occurring, due to the high training accuracy. The phishing data was using a polynomial kernel to the 10th degree, a regularization parameter of 2 and the shrinking heuristic. This was able to get to 88% accuracy with the testing data. As with the other data sets, both would have benefitted from more training data.

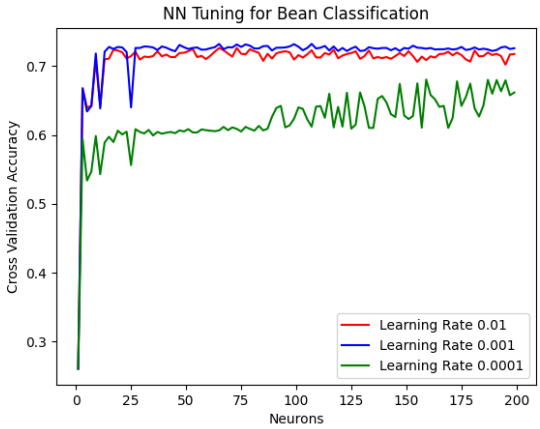
Results

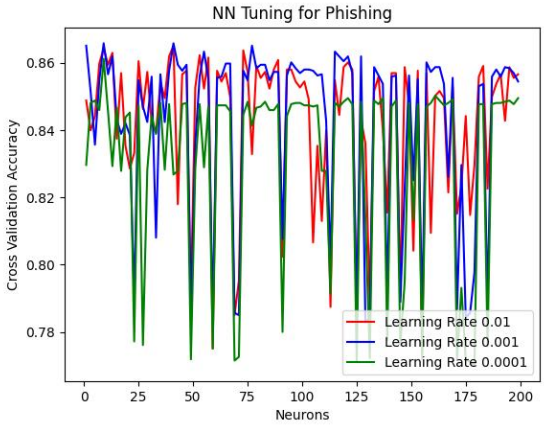
The bean classification data set was able to score a 91% for both the training and testing set. The phishing set was able to score 94% for the training set and just under 91% for the testing set. Both sets performed very well and neither one had an overfitting issue.

The shrinking heuristic was very vital overall in keeping the execution times lower for this particular algorithm without affecting the accuracy of the output. While tuning the algorithm for a linear kernel, the execution times were between 580 and 775 seconds per cross validation run for the bean classification data. When the shrinking heuristic was used, the time was cut down to between 55 and 78 seconds per cross-validation run. This makes sense as shrinking is attempting to remove elements to make the problem a smaller one, thus taking less time to solve.

Neural Networks

Tuning the Model



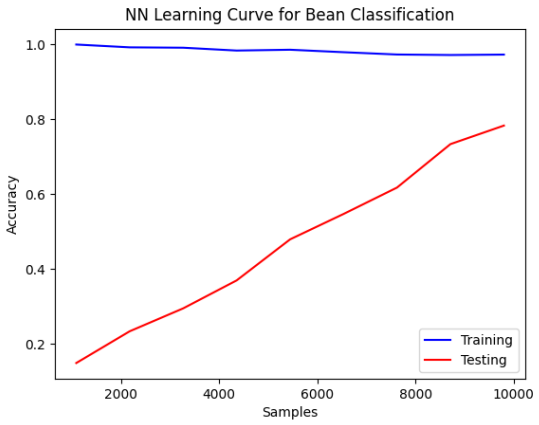


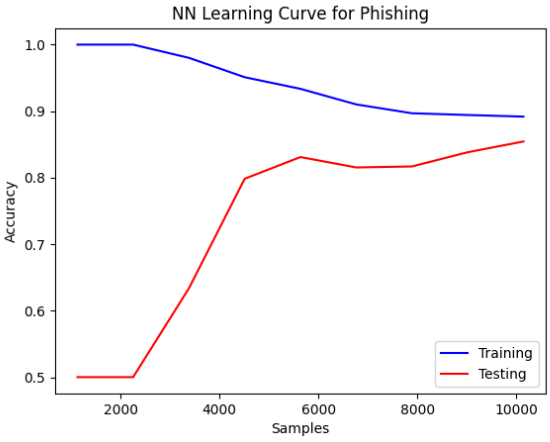
For the neural network, I looked at the hidden layer sizes (neurons) and the learning rate hyperparameters. I didn't notice any major improvements when adding layers, so I stuck with these 2 hyperparameters for the analysis of both datasets. For both datasets, I did eventually reach a point of diminishing returns where adding neurons was not causing any improvements and was decreasing accuracy at some points. The learning rates did behave as expected, as the larger learning rates tended to get close to the optimal accuracy fastest, but lacked the finesse to get to the highest levels that the smaller learning rates were able to reach.

For the Bean classification data set, I did need to apply a scaler object to get any results. I used the standard scaler option in Sklearn. Without scaling, I was only able to get a cross validation accuracy of 26%. By applying scaling, I was able to get an accuracy of 73% overall with a learning rate of 0.001 and a hidden layer size of 65.

For the phishing dataset, there was a lot of oscillation in the cross-validation accuracy. I was able to get an accuracy of 86% with a hidden layer size of 9 and a learning rate of 0.0001. This data set did not require scaling.

Learning rate





The learning rate once again showed that the bean classification data set could use some more samples to learn from as it was not able to converge. The phishing dataset almost converged, as the there was only a 4% difference between the training and testing accuracy at the end of the chart.

Results

Overall, the bean classification data set performed very well with the neural network, with a training accuracy of 94.7% and a testing accuracy of 92.6%. The phishing dataset did not perform as well as it had an 87% for both the testing and training accuracy.

Conclusion

Below is a summary of the training accuracy, testing accuracy and wall clock time of the algorithm that will be used in the final discussion.

Bean Classification Summary

	Decision Tree	**Boosting**	**K-Nearest Neighbor**	**Support Vector Mach.**	**Neural Networks**
Training Accuracy	95.29%	89.45%	100%	91.66	94.74%
Testing Accuracy	89.98%	88.04%	73.50%	91.35	92.60%
Wall Clock Time	0.0574 seconds	54.7621 seconds	0.3417 seconds	18.8853 seconds	3.9082 seconds

Phishing Summary

	Decision Tree	**Boosting**	**K-Nearest Neighbor**	**Support Vector Mach.**	**Neural Networks**
Training Accuracy	90.48%	92.12%	95.20%	94.00%	87.15%
Testing Accuracy	89.75%	90.45%	90.87%	90.68%	87.31%
Wall Clock Time	0.0159 seconds	12.6410 seconds	1.1992 seconds	0.5936 seconds	2.4648 seconds

Overall, the two datasets that were selected were different enough in their behavior to be considered interesting for the purpose of this assignment. A common theme among the learning curves generated for the algorithms was that both data sets would have benefitted from about 15-20% more data. This would have allowed the curves to converge on the graph for nearly all algorithms on both data sets.

It would be tough for me to make a generic recommendation as to what algorithm to use going forward. I think it would greatly depend on the use case. For example, if I was being asked to write a program for classifying beans and I was told that accuracy was the most important thing, I would choose to use either a neural network or a support vector machine, as they were the most accurate, but with added computational costs. If I was asked the same question but told that a slightly lower accuracy would be

acceptable to keep computing costs down, I would select a decision tree as it was the least computationally expensive of all the options.

Overall, for these datasets, the computational times were relatively close, however, if the sets were scaled to much larger sizes, I can see that support vector machines, boosting and neural networks would all begin to have much higher costs associated with them that would lead to decisions needing to be made over how accurate the models need to be for the problem being solved.

Finally, in discussing the complexity, I really didn't have a hard time implementing any of these particular algorithm, however I definitely spent more time tuning some than others. I was able to tune my decision tree, boosted tree and k-nearest neighbor algorithms quickly, though I did spend a bit of time with the bean classification k-nearest neighbor algorithm to try and improve it. I spent most of my time tuning the neural network and the support vector machine. The added complexity was the longer run times for these algorithms, meaning there was a lot of down time waiting for experiments to complete. I also spent a lot of time setting up a pipeline, as I tried multiple data sets before finding these ones that I considered interesting.