Napa Tan Vananupong

Dr. Brian Hrolenok

10 February 2019

Machine Learning CS4641

Assignment 1: Classification Problems

The focus of this assignment was to implement five different supervised learning algorithms on two interesting datasets in order to delve deeper into how to work with classification problems. We were given the freedom to choose any two datasets we wanted to work with that we found interesting and applicable. Both datasets were classification problems, because the value I wanted the algorithms to predict were nominal. This challenged the algorithms to work with distinct feature sets and sizes to predict a classifier.

In my first dataset, I chose the National Consensus data, which I found on Kaggle. I wanted to predict the education level of each individual based on numerous attributes, such as their gender, age, race, marital status, etc. This was an interesting problem to me because it allows us to attempt to predict what would be an extremely useful result if accurate: whether your education level matters in the long run, which, if looked at by a human, could basically indicate if you should actually pursue higher education in order to achieve a better income. Analysis of this dataset would also let us see which features are more important or indicative of the level of education of an individual, whether that would be gender, race, marital status, etc. There are a couple of things about the National Consensus dataset that differentiates it from the second (Letters) dataset: in contrast to the second dataset, the first dataset has considerably more entries but less features. There were approximately 32000 rows of data and 15 features used to predict the final label. Additionally, I would say that this dataset had much more room for error than the second one, because the features were very nonlinear, the dataset was larger, and because what I was trying to predict, in my opinion, was more fickle. I thought this dataset was very interesting to me because I myself am interested in seeing how much money higher education is really worth.

The second dataset I used had more features, but was less fickle: Letters. I decided on this dataset because I wanted to test the classification algorithms ability for modeling something completely different than my first dataset. Whereas consensus data is probably a very popular choice to use for machine learning, I felt that this would be a more unique approach: using the unique attributes of each of the English alphabet's characters to predict which character they belonged to. I felt that this dataset would be a good contrast to the first this dataset was able to strongly predict its target trend so well (due to the fact that the trend is just really a function of the easily quantifiable attributes previously listed) this dataset had less room for error. This dataset is also had a vast collection of data – over 20,000 entries and sixteen features. By comparing the algorithms performances on the first dataset vs this one, we can see which algorithms work better with smaller vs larger datasets and smaller vs larger number of attributes.

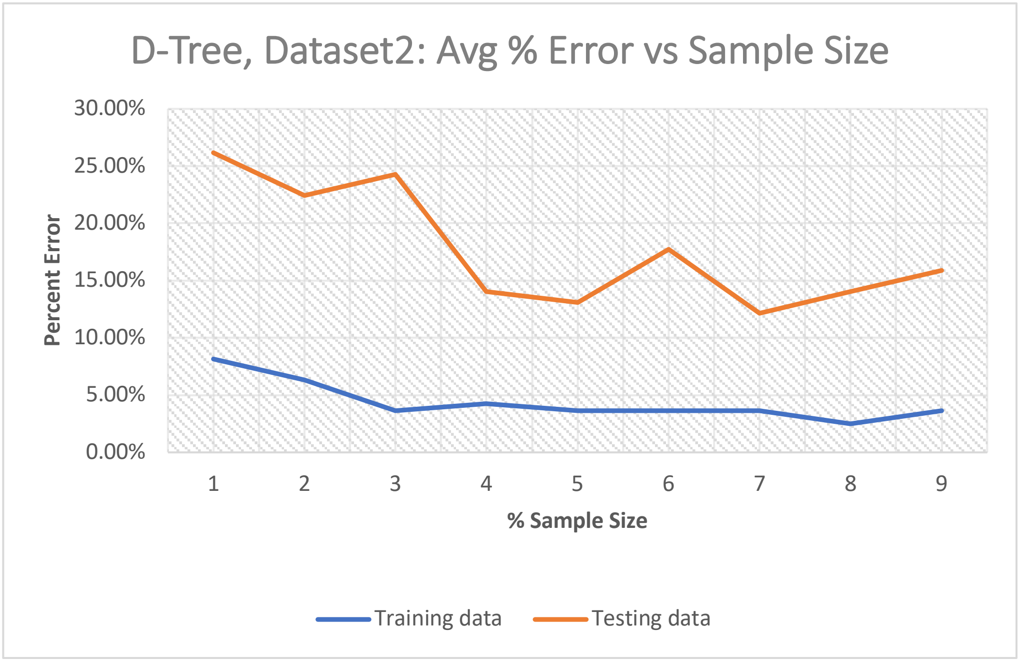
For accuracy, I used three different metrics: Area under ROC curve, Information Retrieval Recall, and the classic Percent Correct. I was interested in these because I felt that they were the best suited accuracy and precision tests for my dataset and classification problems. We want our classifiers to be better than a random or naïve guess, thus we must test their accuracy after the algorithms were run. I chose these tests because I was interested in comparing and contrasting the cost of different mistakes. I also wanted to emphasize the importance and difference between accuracy and precision because it is very possible to have a classifier with a very high accuracy that has a low precision. I felt that the analysis of my dataset 2 (Consensus data) will indeed prove that you can have a classifier with a very high accuracy that is actually quite useless in real life. This is because really need to optimize precision and not merely ensure high accuracy because even impressive looking rates (like my 99%) or more are sometimes not enough to avoid numerous false alarms. There is usually a trade-off between accuracy and precision, as well as recall and precision. If we had cast a wider net, we would have detected more relevant positive cases (and thus have a higher accuracy and recall), but a consequence is that we will also get more false alarms (lower specificity and lower precision). If we classify everything in the positive category, for example, then we would have 100% recall/sensitivity and a bad precision, and most importantly, a useless classifier.

There are several charts throughout this essay where avg. % error does not tell us the full story – where increasing or decreasing a variable does not impact its accuracy as we would expect. Through these charts we can see where the Area under ROC and Precision increases or decreases and use this information to evaluate our algorithms. For the data analysis, I tested accuracy on both training and test sets. I included a chart for every algorithm showing the effect of using a different amount of training and testing data, from a % sample size of 0.1 to 0.9.

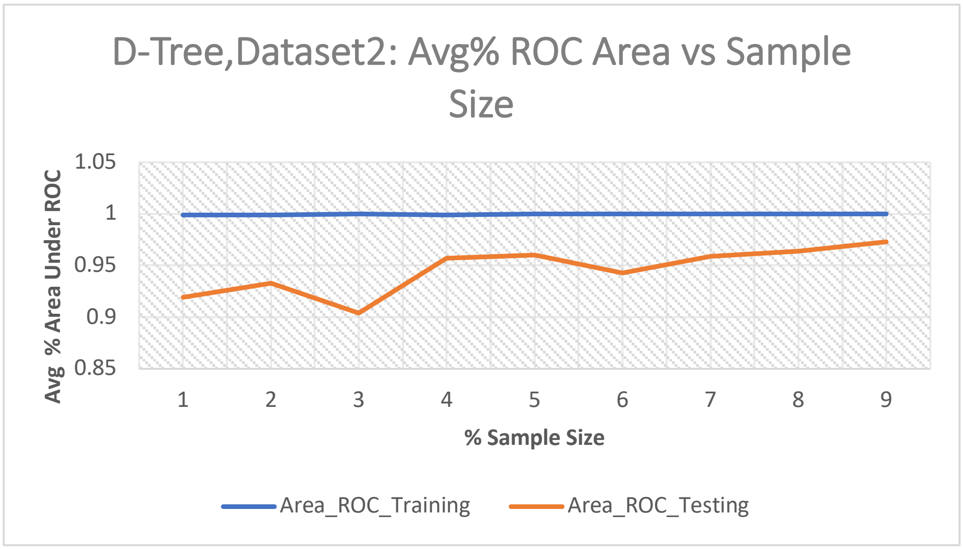
Decision Trees

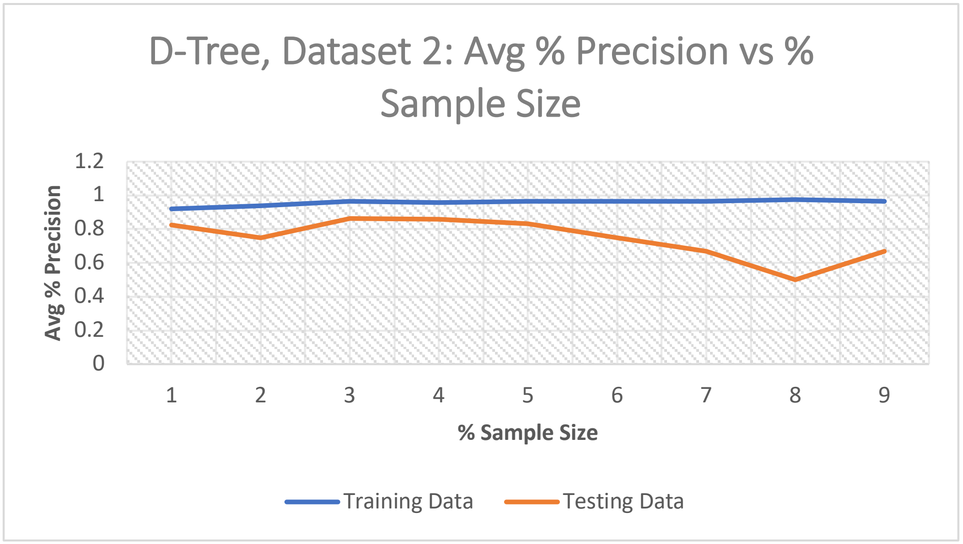
The first algorithm that I implemented was the decision tree algorithm, specifically the Weka built in REPTree, which was an implementation of a fast decision tree learner. I chose the fast decision tree learner because it builds a decision (or regression depending on your problem) tree using the data's gain and variance. It also automatically prunes the tree using reduced error pruning with back fitting. It is also quite speedy because it only sorts values for numeric attributes once, which meant that it was extremely fast for the second dataset. Since it was the first algorithm I implemented, I will also start off by reviewing the performance of the decision trees first. One advantage of decision trees is that it can be used for both classification and regression problems due to its flexible nature. Each branch of the tree represents a possible decision, occurrence, or relation. Decision trees have the advantage of automatically selecting the most discriminatory features and putting them to the root, thus it was expected that the classification of the consensus data and the letter data be more affected by certain features than others, and a decision tree can easily take this into account. However, with all its advantages, Decision trees also have a few major disadvantages, the major ones being: overfitting, high variance, and low bias. Overfitting occurs when the algorithm captures noise in the dataset, and in the charts we can see how there is not a perfectly positive correlation between increasing the train size and better performance. Unlike the other four algorithms I implemented, decision trees benefitted the least from increasing the size of the training set, even though they did still have a positive correlation. To prevent overfitting, I experimented with different hyperparameters – specifically max depth values – to prevent the tree from growing too large and to fit only the training set.

Another way to prevent overfitting is to prune the d-tree after it is generated. I accomplished this by recursively going down the tree and removing



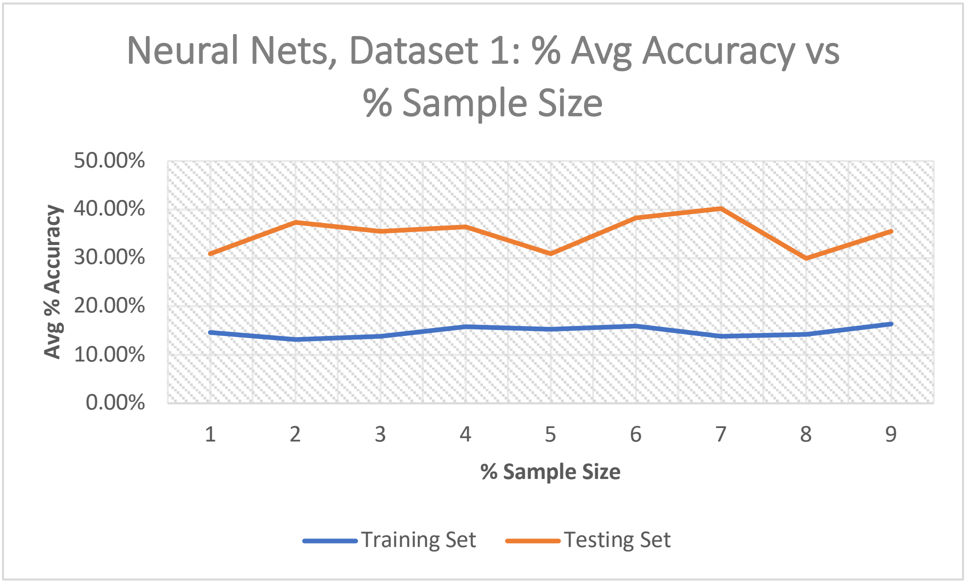
all nodes with less children than a certain amount. What I found with pruning was my results were that expectedly accuracy on the training set went down, but accuracy on the test set saw hardly any improvement.

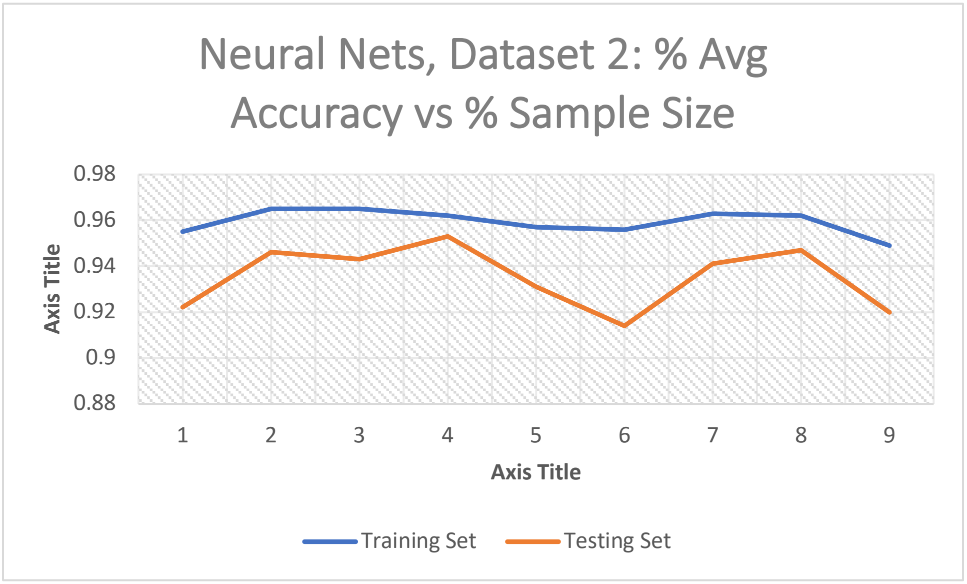


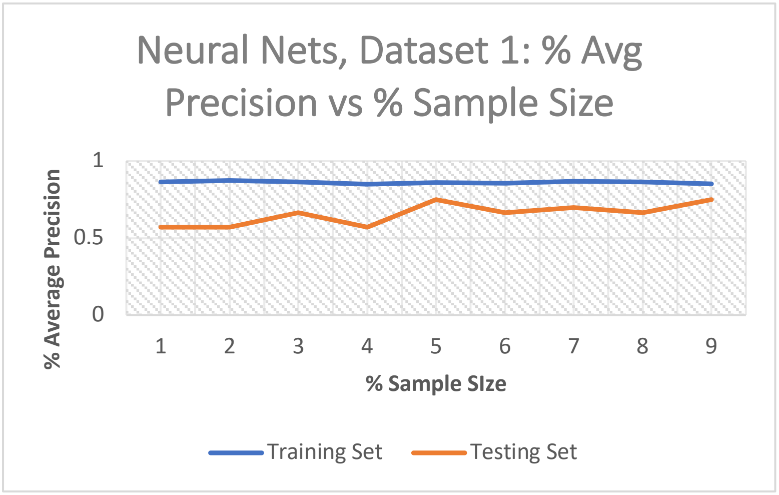


Neural Nets

I implemented neural nets on my datasets by using the Multilayer Perception functions in Weka. With both the datasets, I tested how different training sizes, maximum iterations, and whether or not the dataset was cross-validated impacted the algorithm's accuracy. Neural Networks are good to model with nonlinear data with a large number of inputs – thus it was perfect for my first dataset (consensus). It is very reliable in an approach of tasks that involve many features. They work by splitting the problem of classification into a layered network of simpler elements. With that being said, one weakness of Neural Nets compared to the other algorithms is that their black box nature, that is, we don't really know how the Neural Net came up with the classification, or output. Another big drawback is that they are the most computationally expensive algorithm and sometimes require massive datasets in order to work well. We can see this in the analysis of my results: Neural Net was much better suited for the Consensus dataset because it was much larger and nonlinear, than it was for the Letters dataset which was almost two times smaller and much more linear and predictable. Looking at the plots, it is clear that the Neural Nets accuracy correlated somewhat with the size of the dataset – the Neural Nets benefitted from having a larger training set in both datasets. However, I did not find a strong correlation between the Neural Nets accuracy and increasing or decreasing the learning rate. Varying the maximum iterations did not seem to correlate with the Neural Nets performance either.

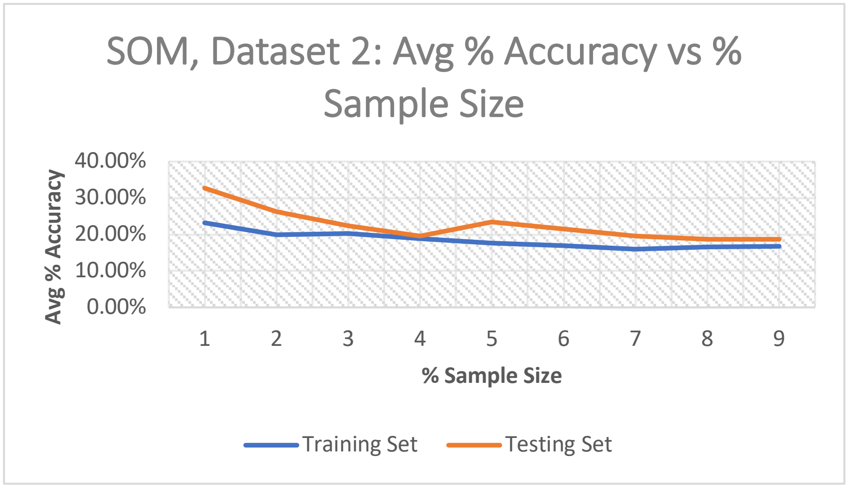

Neural Nets, Dataset 1: % Avg Accuracy vs % Sample Size
Avg % Accuracy
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
1
2
3
4
5
6
7
8
9
% Sample Size
Training Set
Testing Set

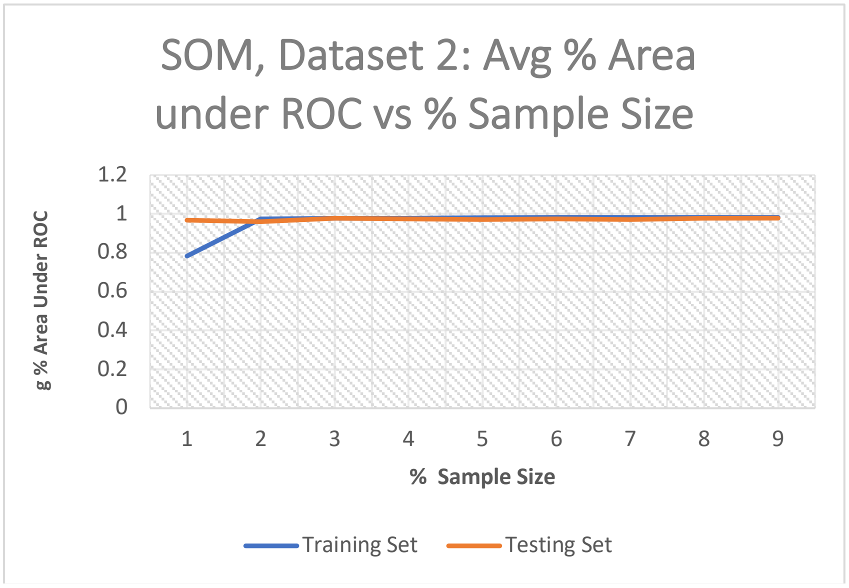

Neural Nets, Dataset 2: % Avg Accuracy vs % Sample Size
Axis Title
0.98
0.96
0.94
0.92
0.9
0.88
1
2
3
4
5
6
7
8
9
Axis Title
Training Set
Testing Set

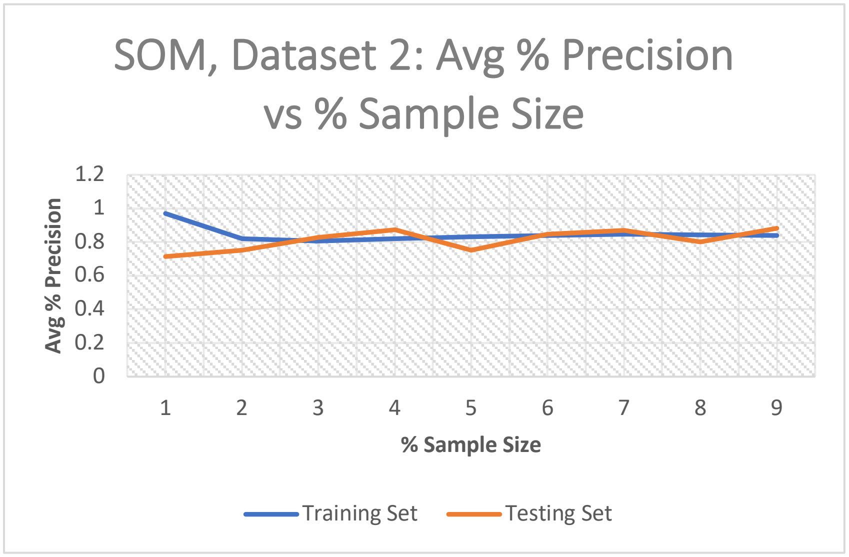

Neural Nets, Dataset 1: % Avg Precision vs % Sample Size
% Average Precision
1
0.5
0
1
2
3
4
5
6
7
8
9
% Sample SIze
Training Set
Testing Set

Support Vector Machines

For the Support Vector Machines, I used the built in SMO function, otherwise known as Sequential Minimal Optimization. I experimented with different sample sizes for both training and testing data and found a strange effect on my results: For both training and testing data of my first dataset, accuracy decreased as sample size increased. The correlation was slight, however, it was still a constant decrease. Average area under ROC did not seem affected and became stagnant after a certain sample size. If this dataset was larger and perhaps more linear, I think that the SOM would have been much more effective at predicting it. Another drawback of SOMs is that they were the second slowest to run, following behind the Neural Nets. Additionally, as we can see from the results, increasing sample sizes for both testing and training did not improve the results at all.

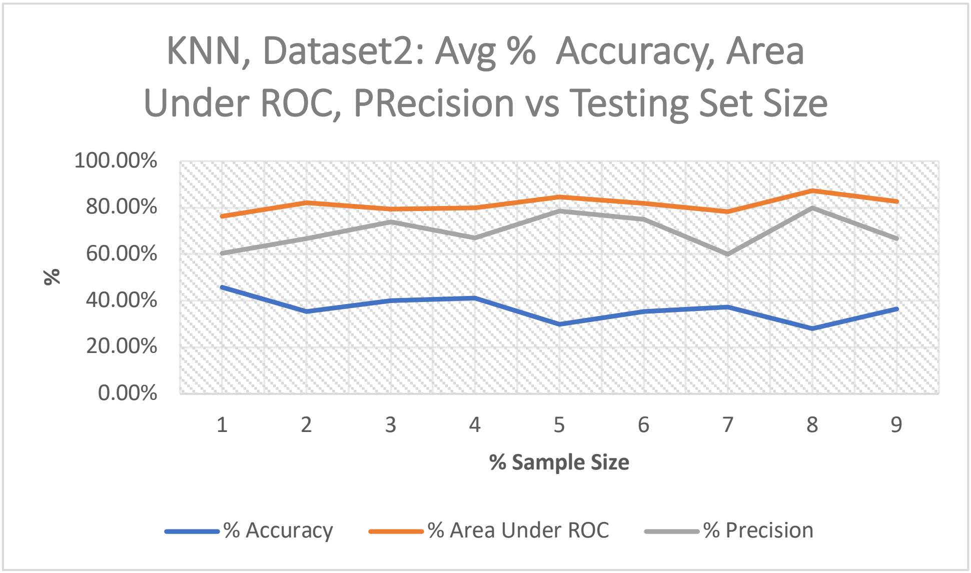






K Nearest Neighbors

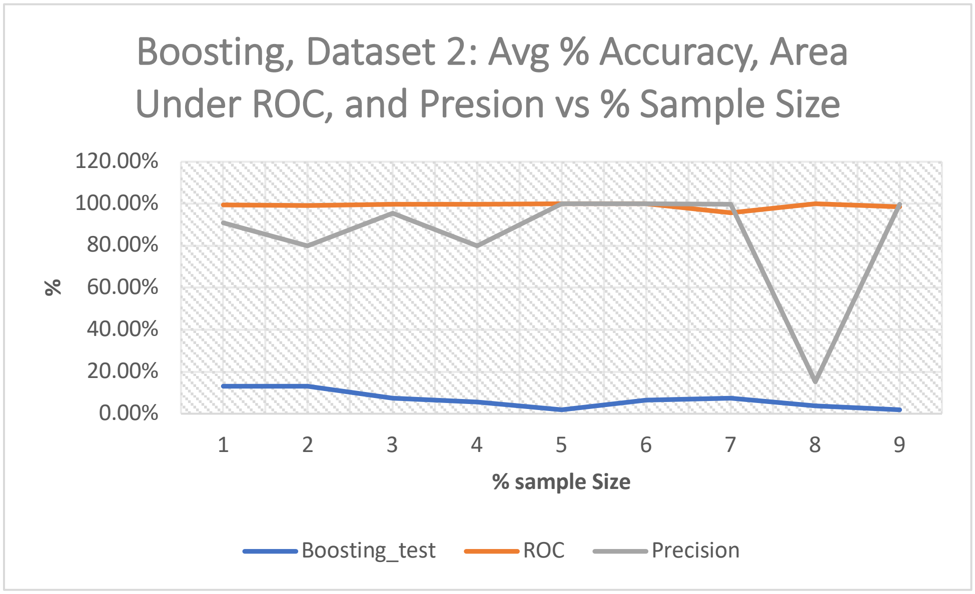
For the KNN algorithm, I used the built-in implementation in Weka. Since KNN algorithms work best with larger datasets, my result's accuracy curves were not surprising. For both datasets, the accuracy correlated slightly with size of training set – the larger the sample sizes, the better the accuracy for both training and testing data. However, the resulting plot shows that the correlation does not seem significant. Another parameter I tested was the number of neighbors. In my second dataset, increasing the number of neighbors led to an increase in accuracy. This is due to the fact that my first dataset was smaller, so when both training size and test size were small, the KNN algorithm struggled with defining accurate boundaries with larger values of K. On the first (much larger) dataset, the results clearly show that increasing K up to a certain point was ideal, however, after it reaches that threshold, the accuracy starts to drop off.



Boosting

To implement the boosting algorithm, I used the built in AdaBoost method in Weka. This is algorithm is unique from the other four because it a one learner ensemble algorithm, meaning that unlike the Neural Nets or Decision Trees, Boosting utilizes multiple learners and attempts to combine them to prune out each learner's weakness. Boosting is an example of an ensemble Machine Learning algorithm, with each predictor implemented to fix the previous predictor's weaknesses. It can be observed from the plots that increasing the training size does not increase % accuracy for either data set, which may be because they already have plenty of examples to train on.

For my second dataset, there is a positive correlation between increasing the training set size and precision, until the random steep drop, which was probably due to error. When it comes to varying the number of boosting stages however, we can observe that increasing the number of boosting stages has a clear increase on accuracy for the second dataset but not the first.



Performance on Training and Testing Sets Summary:

Training sets were approximately 90% of total data

Hyperparameters:

Decision Tree: Max_Depth = 8

Neural Net: Learning Rate = 0.0003

Boosting: N_Estimators = 900

SVM: Kernal = Linear

KNN: N_neighbors_3

Analysis & Conclusion:

Picking an overall best algorithm was difficult because I wanted to consider each algorithm's performance on both training and testing sets as well as their performance on three different metrics: average % accuracy, area under ROC, and average % precision. For both datasets, the best performing algorithm was Boosting. This makes sense because Boosting runs a large number of boosting stages to combine many learners in order to reduce their individual weaknesses as much as possible and built one strong model. To improve my results for the two datasets next time, I think if I had more data it would greatly help. We could also try increasing the number of boosting stages. However, one consequence would be reaching a point of diminishing return by simply increasing the stages. Another approach we might take would be to modify how we combine the learners themselves between stages. Throughout this analysis, I have covered the five of the most popular machine learning algorithms in detail along with providing large amounts of data on how they operate with different parameters and inputs. Hopefully this analysis has shed some light on the strengths, weaknesses, and behaviors of these algorithms applied to classification problems.