Kaley Findley

CS 4641 Project 1

Supervised Learning

To best illustrate the advantages and disadvantages of the following supervised learning algorithms: k-Nearest Neighbors (kNN), Decision Tree (DecTree), Boosting, Neural Networks (NN), and Support Vector Machines (SVM), I chose two data sets that are very different from each other. They differ from each other in terms of size and classification type. The first data set describes a person in terms of education, residence, occupation, gender, age, etc., to determine whether or not they make more than a 50k salary. This data set is a binary classification problem and has a large number of instances. I chose the income data set to see how modifying parameters of different algorithms behave over large data sets. The other data set I chose was about the chemical makeup of wine and how that determines the overall quality of the wine on a scale of 1 to 9. The wine data set was much smaller in scale than the income data set. Due to the fact it is not a binary classification problem and it's smaller in sample size, my hypothesis was that the algorithms that behaved well for the income data set, would behave very differently from the wine data set. This way I can get a thorough representation of the advantages and disadvantages of each algorithm.

My overall approach to this problem consisted of three steps. The first step was simply organizing my data into varying training sets and a consistent test set that was large enough in comparison to the training set to avoid overfitting. The income data set had roughly 40k instances in the overall training set and came with a separate test set of about 16k. I initially split the total 40k instances into 5 partitions starting at 8k to 40k. However, when I began running tests on these massive training sets, I found that it was extremely costly and inefficient to spend 10 hours (for NN) on one test and not result in any better results than smaller sample sizes. So I decided to partition the training set into 5 partitions starting at 5k to 25k instances. Wine data was one file of around 4k instances, so I used the final 1k as the testing set and I partitioned the remaining 3k into 5 parts consisting of 500, 1000, 2000, 2500, and 3000.

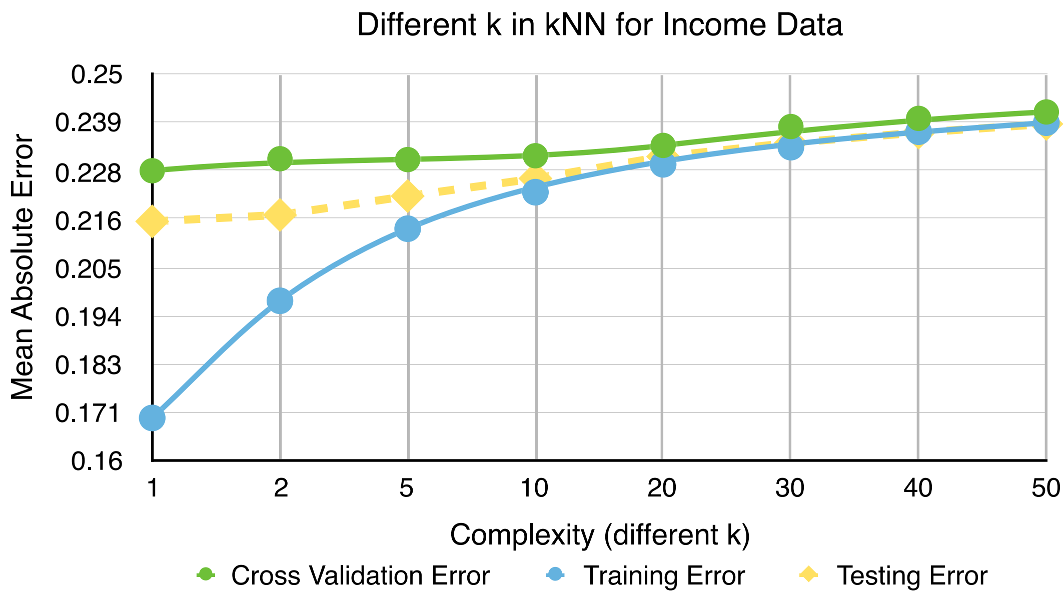
I used Weka to run my tests and I saved the result buffers and models for later analysis. For each algorithm, I selected the best parameters and I assumed they were independent of each other. In other words, I would change one parameter and leave the rest as their default values in Weka. This was the easiest way for me to visualize how to test certain parameters. So I took the training set and ran cross validation on it with 10 folds, then reevaluated that model on the overall training set to get the training error, then reevaluated the model again on the test set to see the 'real world' performance of the model I just generated. When evaluating different parameters, I used the smallest training set (complexity curves) and later made a learning curve on the best model, which I will explain in more detail later in the analysis.

I illustrated the results by plotting the mean absolute error of each test I ran on each training set partition. For example, for different k values in kNN in wine data, I plotted mean absolute error vs. the value of k for 500 samples for cross-validation (CV) error, training error, and testing error. I then analyzed these graphs to determine the point of overfitting when the CV error was at its minimum when training error approached zero. For the best models for each algorithm I plotted the average ROC area to determine the average performance of the model in terms of true-positives and false-positives.

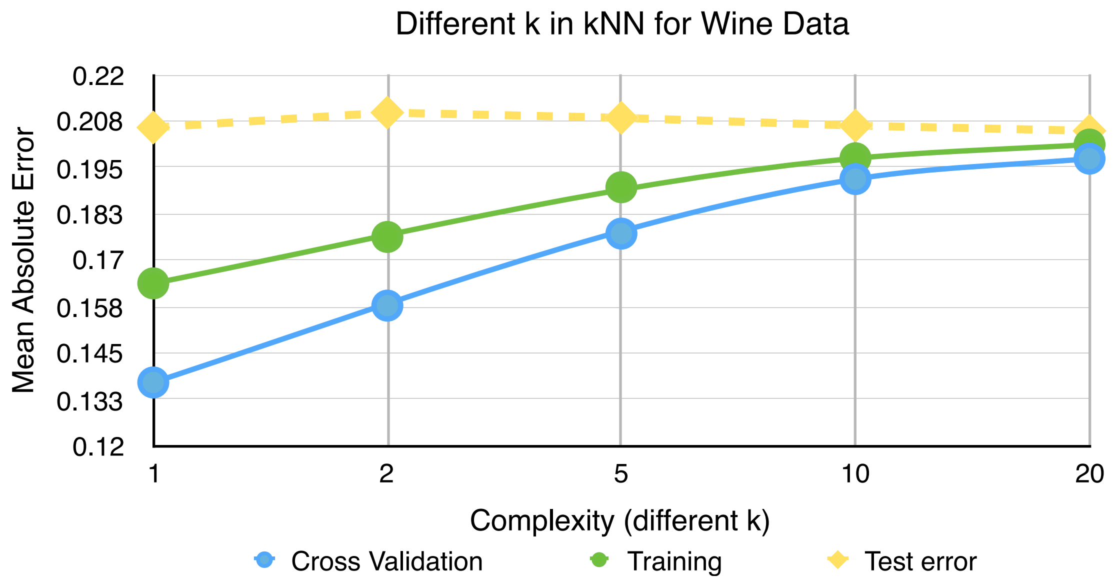
To pick the best model, if the result graphs did not show the overfitting point, I continued adding more data points to find that absolute minimum of the CV data before it began to overfit. My hypothesis was that if the CV error was as low as it could be then this would be the most accurate prediction of the 'real world' performance of the algorithm. I understood that the CV model is optimistic, but I believed it would be the best way to get an educated generalization of the data. I will go into more detail of the result of this process in terms of the algorithms later.

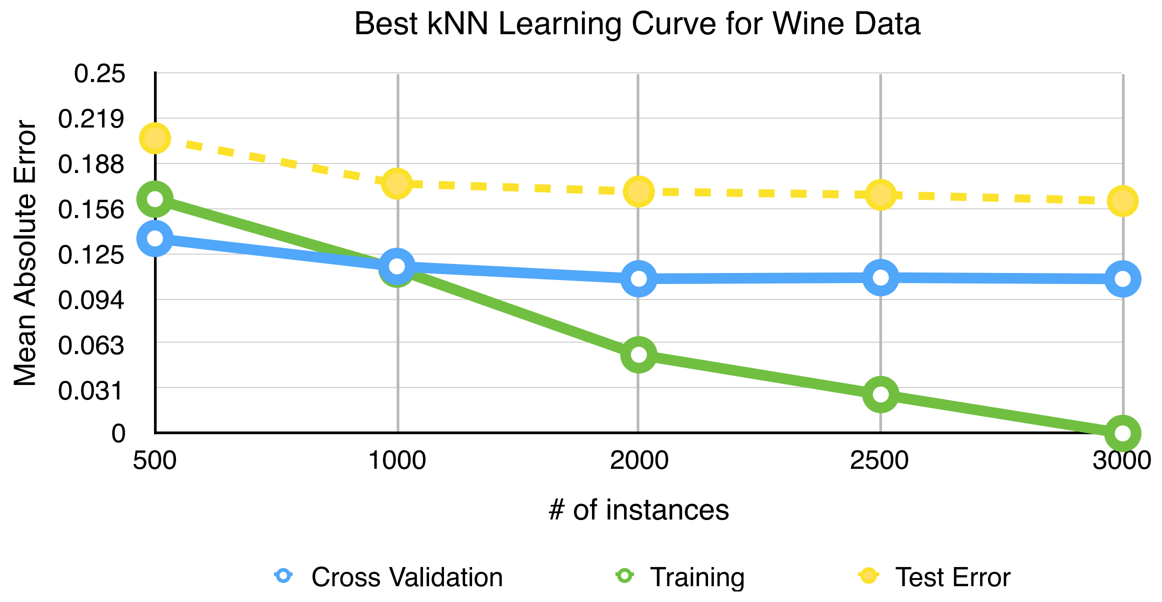
For the kNN algorithm, I chose to manipulate the values of k (number of nearest neighbors) and the weighting scheme. In my understanding and from initial exploratory tests I found that these parameters would be the most 'telling' parameters. The other parameters affected the overall result minimally or not at all. For DecTree I decided to use the decision tree with information gain along with pruning for both sets of data. Information gain tells us how important a given attribute of the feature vectors is. It is used to define a preferred sequence of attributes to investigate to most rapidly narrow down the outcome. I also decided to manipulate the minimum weight of the tree. I predicted that it would lead to better results rather than manipulating the max depth or weight because that would lead to limiting the depth and accuracy of the tree. For boosting, I manipulated the type of classifier and the number of iterations. For wine data, the classifiers compatible with the data were SVM, DecTree, and NN. For income data, the compatible classifiers were kNN, Decision Stump, NN, and DecTree. In regard to the iteration number, my initial hypothesis was when the number increased the accuracy of the predictions would increase. For NN, I manipulated the learning rate, training time, and hidden layers (only for income data). Since income was a larger data set, changing the hidden layers would be potentially more interesting than spending time evaluating it on wine data. My hypothesis for NN was when the learning rate increased the time to run the algorithm would decrease. Some NN tests on large data sets could potentially take 8-10 hours to complete. For SVM on income data I changed the kernels, cost, and the gamma value. I only manipulated the kernels on wine data, which led me to try manipulating some of the other minor parameters for income data. That way I can see if those more insignificant parameters actually held more weight on larger data sets.

The first algorithm I began to test was kNN and I started manipulating the k values. I tested 1, 2, 5, 10, 20, and 30 (and 40, 50 for income data). I found that when k increased the mean absolute error would increase dramatically. The concept of kNN is that if you increase k then you increase the complexity of the line that partitions the data set, thus leading to more accurate predictions. However in the case of income and wine data, they must be linearly separable, therefore increasing the k value would increase the complexity of the separating line, leading to a biased prediction on the training set. I found that when k was 1 it led to the lowest mean absolute error over CV error, training error, and testing error as you can see in the graphs below.



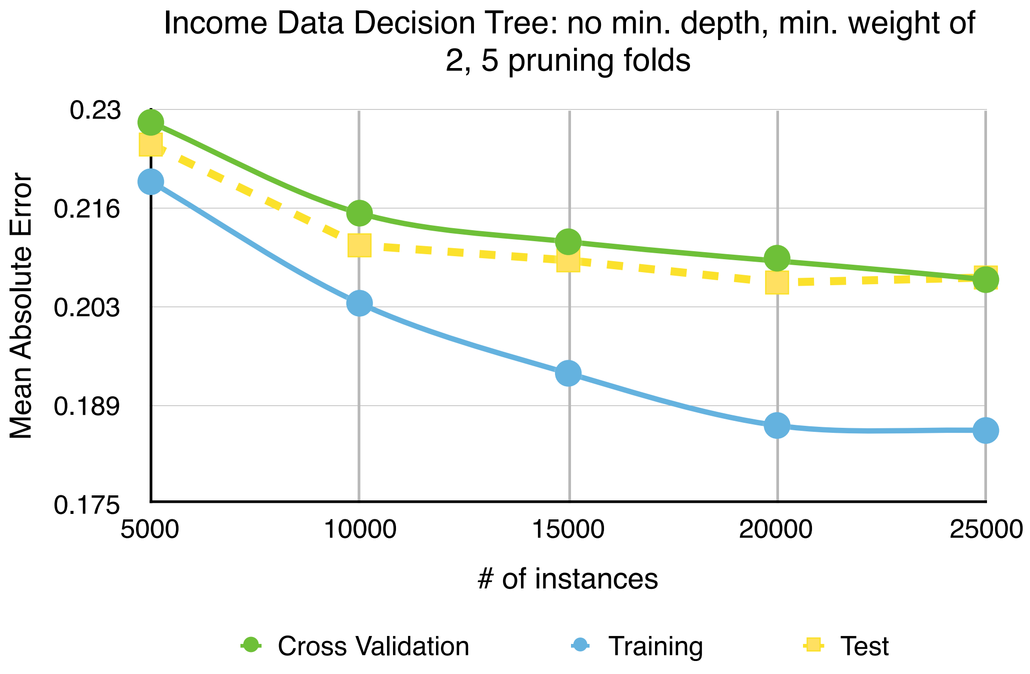
I then started to manipulate the weighting scheme for kNN. The two options for which are 1/distance and 1-distance. I have found that the 1/distance weighting scheme performed better than the other scheme. The 1/distance scheme has a greater complexity than 1-distance and it made the data points closer to the line separating the data more important. So it makes sense to assume that with 1 nearest neighbor the 1/distance scheme would make the model more accurate in predictions. You can see this trend in the best kNN model of k=1 and with 1/distance weighting in the graph below.



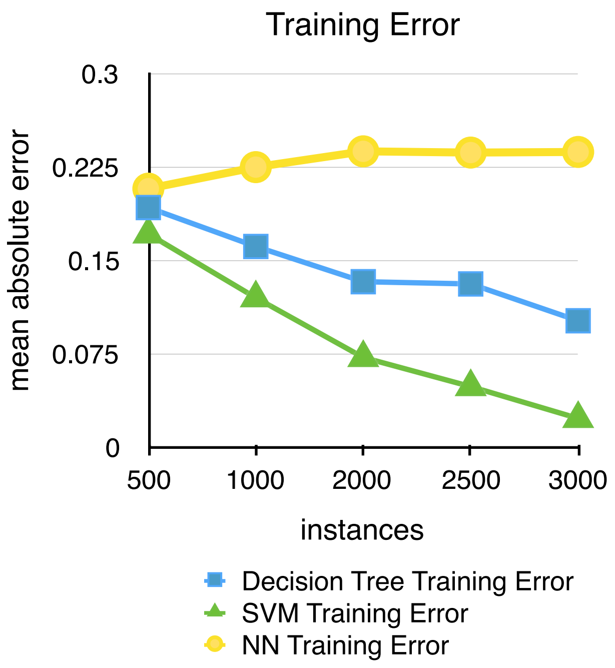


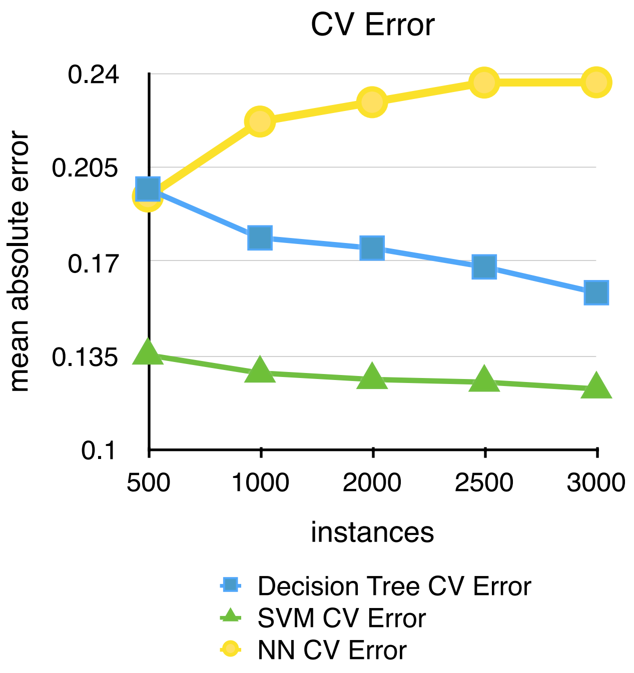
The next algorithm I evaluated was Decision Tree. For wine data I only manipulated the minimum weight of the tree. I tested the performance of the pruning folds and maximum depth parameters with income data. For wine data I have found that increasing the minimum weight the mean absolute error increased greatly when it was greater than 1. Any values less than 1 yielded a constant or relatively minimal error result. For wine data, the best model for decision tree, based on the parameter I tested was a model with a minimum weight of 1.

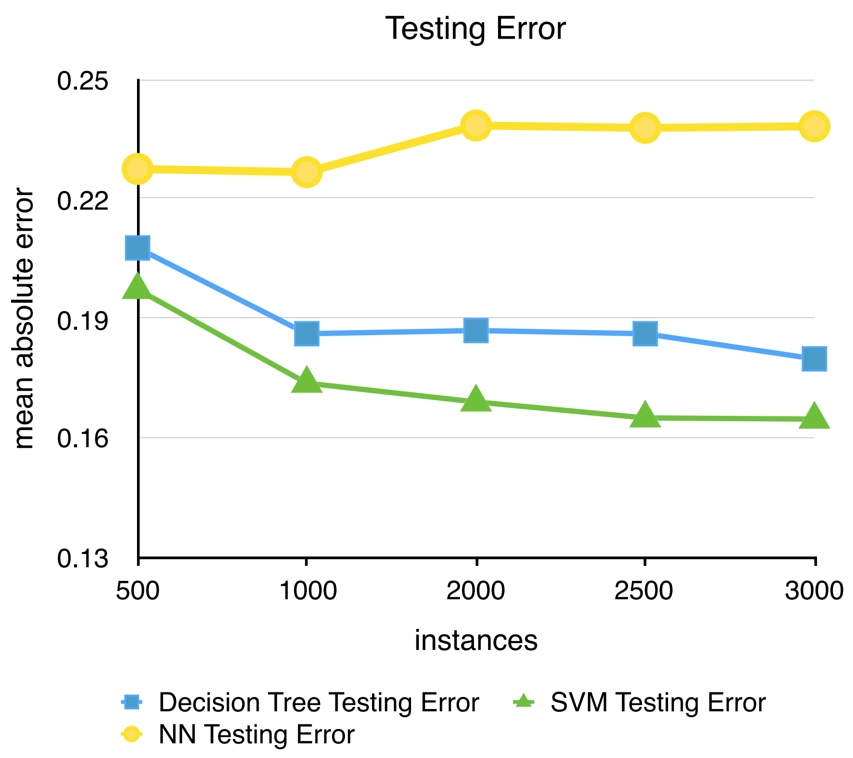
However, for income data that was a different story. With income data I found that adjusting the maximum depth of the tree was not applicable for the data we have. The CV, training, and testing error did decreased initially from 1 to 2, but then remained constant for any following values. In addition to the minimum weight parameter, the pruning folds affected the outcome of the test significantly. The optimal amount of pruning folds (before overfitting) was five. When the optimal pruning fold value was applied with a minimum weight of 2, it yielded an overall optimal generalization of the data, based on our performance metric and scope of our experiment.



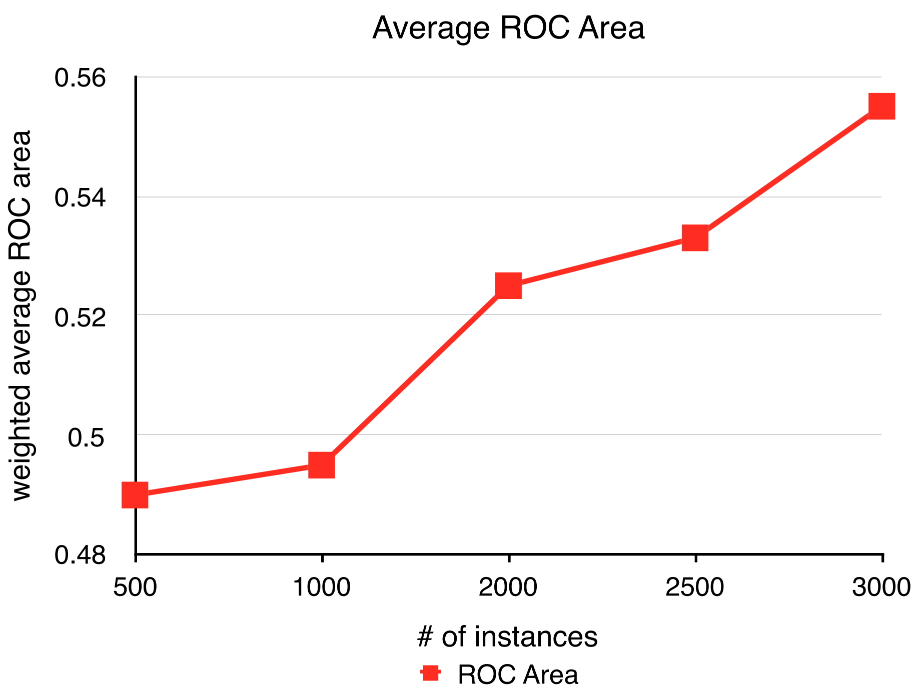
The next algorithm I analyzed is Boosting. When attempting to run boosting on wine data with the default settings, it notified me that some of the classifiers are not compatible for boosting with the data set. For wine data, the only classifiers compatible are DecTree, SVM, and NN. DecTree performed the second best behind SVM. The reason it wasn't the best because wine data is not a binary classification problem, which is what dec tree is good at predicting. SVM performed relatively the best in terms of my decided performance metric of mean absolute error. It produced the lowest CV, training, and testing error. However when I say it performed 'well' the correctly classified instances was only around 60% for CV and only about 43% for test (for 3000 samples). When looking at the average ROC curve, it became apparent that SVM was not the optimal choice. It had the lowest averages over the increasing amount of sample size. But to keep in line with my performance metric, SVM was the optimal choice and will be more apparent in the graphs below.



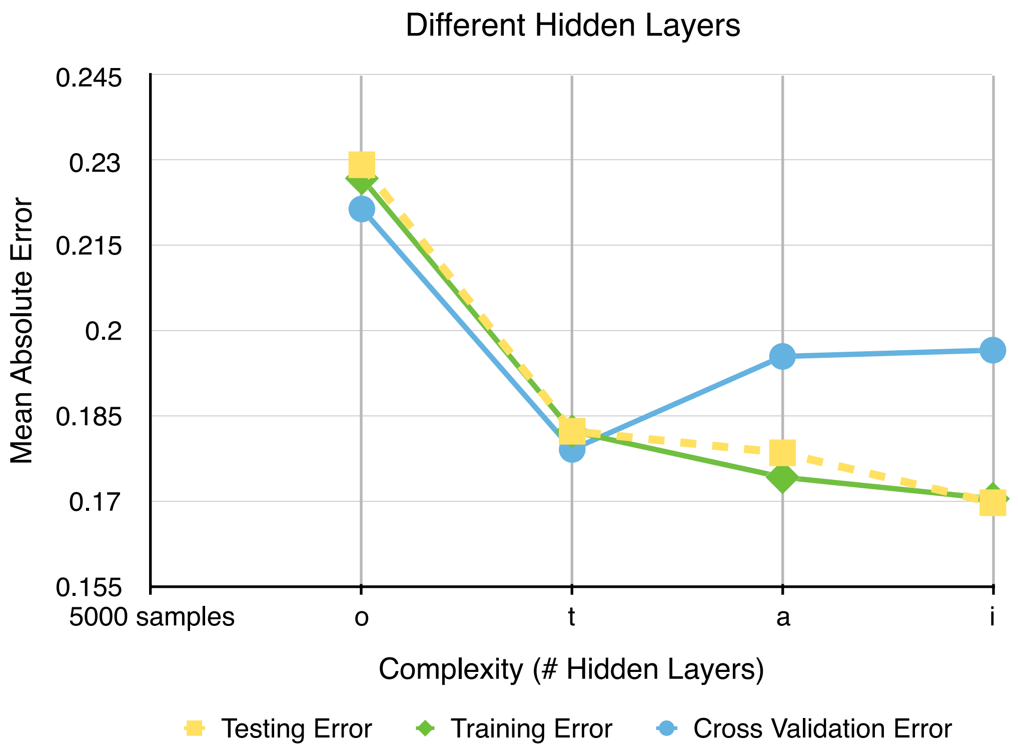




The SVM is in green and it obvious that it performed the best in terms of reducing mean absolute error. Also it is not apparent that it is overfitting in the 3000 samples provided. The CV error is still decreasing while the training error is approaching zero. For an additional point of analysis, you can see that the testing error is still decreasing as well, which is preferred.

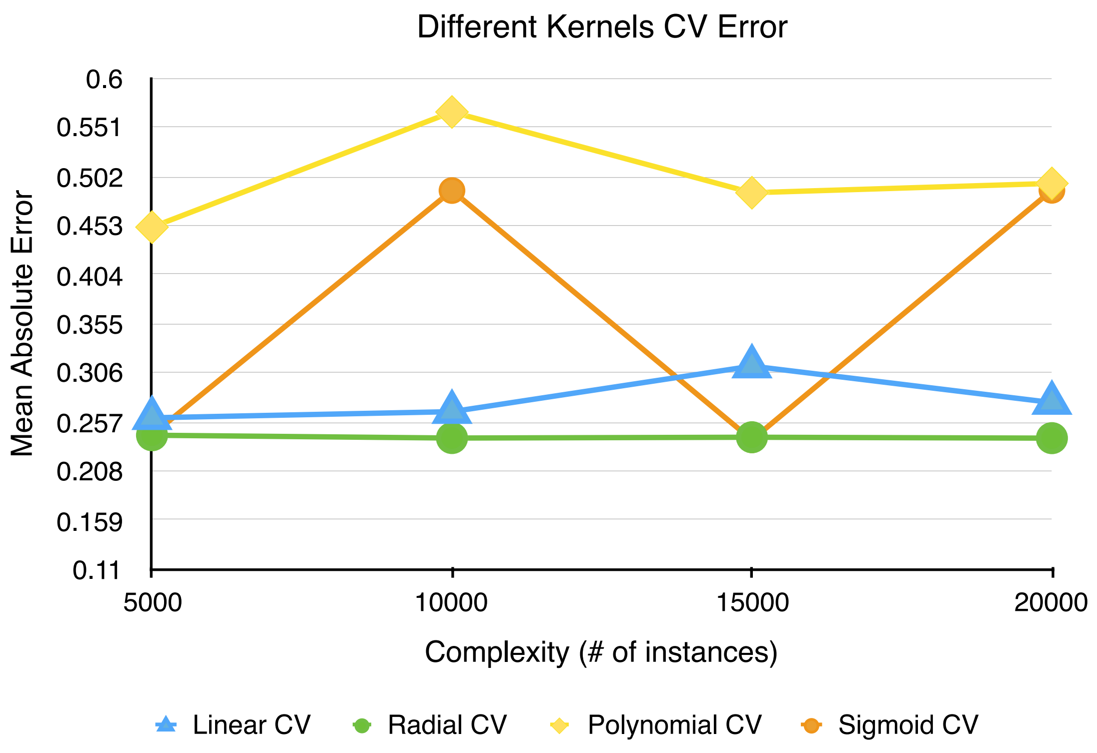


The next algorithm is Neural Network. This algorithm was by far the most time consuming algorithm to run. This is due to the different layers of neural networks and the time it takes to execute each one of those layers. For income data, I tested training time, learning rate, and different hidden layers. The optimal point of training time was 500 epochs. This was the point right before the overfitting began. The optimal learning rate was 0.3. The learning rate complexity curve illustrated some weird behavior. When you set learning rate to 0.6 the error spiked by over a tenth of error. The mean absolute error never goes above 1 so for an error to jump by at least a tenth, something is not working efficiently. The complexity curve for the



different hidden layers clearly shows the point of overfitting, which is why I chose 't' as the best hidden layer parameter value. Overall for income data, in terms of percentage of incorrectly predicted instances, neural network performed the best. In the largest training set of 25k, it only missed about 12% of the total testing instances.

The last algorithm I evaluated was Support Vector Machine. For income data I tested the performance of the parameters: cost, kernels, gamma. The effects of cost and gamma were very minimal if effective at all. The transition from 0.75 to 1 showed a decrease in training, but no significant change in CV or testing error. I originally was only going to test the different kernels for income data, but for the sake of being thorough, I decided to explore other options and see if there were other significant factors that can come into play for SVM. After my many tests and graph analysis, I have concluded that the main parameter influencing SVM outcome, is the kernel used. The kernels tested on income data were linear, polynomial, radial basis, and sigmoid. One of the best kernels was linear, which is expected since due to previous algorithms like kNN, it has shown that my data can be linearly separated well. However, the best kernel was radial basis. It stayed the most consistent in terms of error output. The other kernels had random spikes over the different iterations of training data. This strange behavior is shown in the CV error graph below. Overall for wine data, SVM was the best algorithm for the data structure.



In conclusion, after analyzing all of my data across both data sets, I realized the way I chose the best model and how I decided which parameter performed the best led to overfitting the model to the training set. I chose the 'best' parameters and models based on the lowest CV error, which I initially thought was avoiding overfitting because I understood that CV gave a generalized model to base your predictions on how the model would perform in the 'real world' aka test set. Cross validation splits the training set into 'folds' which basically is splitting a percentage of the training data into a smaller training set and using the rest as a test set and evaluating the training set in regard to that sub test set. It repeats this process for how many folds you specify (in my case it was 10). The goal of CV is to generate a model that is an optimistic expectation on how the algorithm will perform when introduced to data it has not seen before. It shows how might the algorithm will generalize to 'real-world' applications. So I assumed this was a proper decision metric. However, the lower the CV error got the test set error wouldn't always decrease with it, which is an apparent sign that I am overfitting in regard to the training set.

In the future I would choose the 'best' performing parameters and model based on not only CV error but also test error. I would use the CV to determine the overfitting point in terms of the training set, but not solely rely on it to avoid overfitting to testing set as well. I would take into account how it actually performed to the sample test data I set aside. If time were not an issue, I would explore manipulating for parameters such as some of the weighting choices for decision tree or the combination of using different classifiers with optimized parameters in boosting. I would test the 'best' configurations I found in my analysis over much larger data sets. With the wine data sets, there were times where my data was not big enough to find that crucial point of overfitting. I could have found a more optimal solution if I had a greater sample space. There was an assumption I mentioned earlier about how the parameters are independent of each other. After developing ‘optimal’ models, I believe that assumption is not necessarily true. Based on the situation of the data or the status of the other parameters, you could create a completely different model that behaves differently than you might expect. For future experiments regarding learning algorithms I now understand what to look for in regard to overfitting and to assure that I don’t become too biased to the training set like I did this time.