CS 7641: Supervised Learning

Stacey Wieseneck
swieseneck3@gatech.edu

1 CLASSIFICATION PROBLEMS

For this assignment my two classification problems are:

1. Classifying high or low heating load on a building based on factors of relative compactness, surface area, wall area, roof area, overall height, orientation, glazing area, and glazing area distribution.
2. Classifying high or low cooling load on a building based on factors of relative compactness, surface area, wall area, roof area, overall height, orientation, glazing area, and glazing area distribution.

The reason for choosing this data is largely from having a background in construction and manufacturing. It's a much more commercialized industry than people realize, and a lot of developments are industry driven with new products promoting enhanced heating and cooling load coming from a corporate supplier. This data is non-trivial in that it will show if there are simple design characteristics, ones that do not make a difference for any company's bottom line, that can still affect heating and cooling load. Also my electricity bill has been insane lately, would love to know at a high level design principles to look for in buildings where I'm responsible for the power.

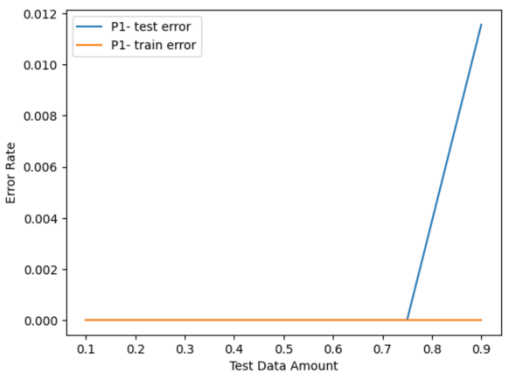
The link to my code is in my *ReadMe.txt* file. However for good measure here is the url: https://github.com/staceywies/Nothing-to-see-here/tree/main/Supervised%20Learning.

2 RESULTS

The following table outlines the training and testing error rates I obtained utilizing the variable learning algorithms. For all values below, the test data was 25% of the data available.

2.1 Decision Trees

Starting with decision trees as my learner, both problems we the same thing. Basically this data is very, very well suited for a decision tree. The error rates for almost all splits of test/train data stays static at 0. For both problems we see that as the training data becomes only 10% of the data set overfitting occurs as the training data has less error than the testing data. Overfitting occurs a bit sooner when looking at the second problem (P2). I'm almost thankful for this showing up in the data, before seeing this the consistent error value of 0 had me worries about the integrity of the decision tree.



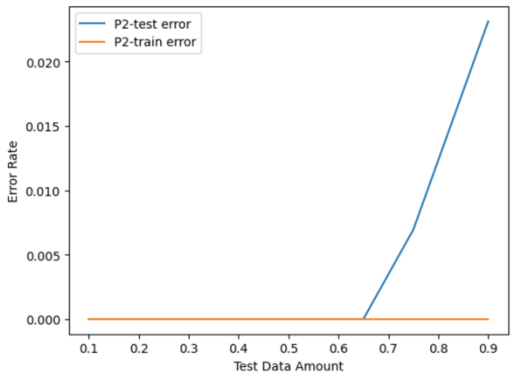


Figure 1 and 2: Test and Train Error on decision tree learner for both problems with vary test/train data amounts.

Looking at the number of iterations run, the data starts to get interesting. Iterations do decrease a bit even as the amount of testing data decreases slightly. While this isn't a huge increase it is insight into the data set as this would not be expected behavior. This tells me that the data splits are much more clear when we're working with a larger amount of data, as the data fed into the training model decreases, the bests splits are harder to decipher. Since I'm using old code from a previous class my method for determining which variable and value to split on is a bit archaic, decided through correlation coefficient with the output values and the split value determined by the mean of all the values. A lot of assumptions are made with this split and I would expect a much more reasonable and expected curve for iterations and time if I were to use GINI index or other entropy or entropy adjacent methods that are less biased to the volume of data present. This could also have to do with the pruning method at play, where I am not building out to set leaf amounts, I am building until I am left with a group of datapoints that have a common output variable.

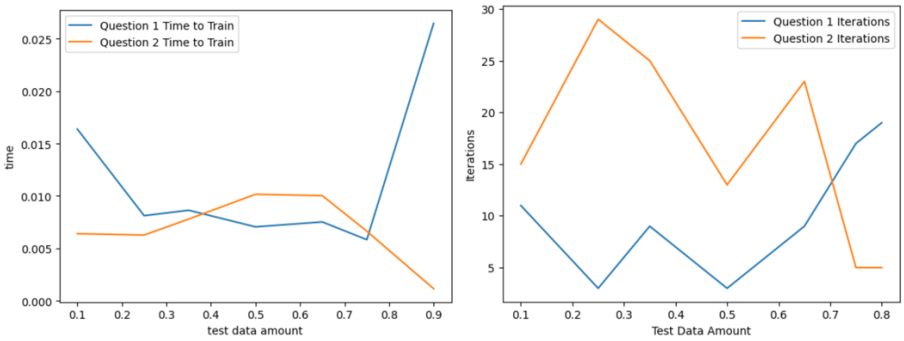


Figure 3 and 4: Classification questions 1 and 2 and their associated time to train and required iterations on the decision tree learner.

2.2 Neural Networks

Looking at my neural networks model next. This one behaved much more like one would expect to see. I approached this with a very simple neural network, no hidden layer and optimized the weights iteratively using the gradient approach. The activation function was to "fire" for values greater than or equal to 0.5. For both problems I started by looking at the error rate for a 0.25 rate of test data to confirm the learner was indeed decreasing error. We see the error rate does decrease eventually, after a few dozen iterations on the gradient, however both plateau at a significant level of error, either unable to reduce error further or unable in the set amount of iterations without the system crashing. This shows the simple neural network model is not suitable to account for all of the variable and their interactions with such a simple model.

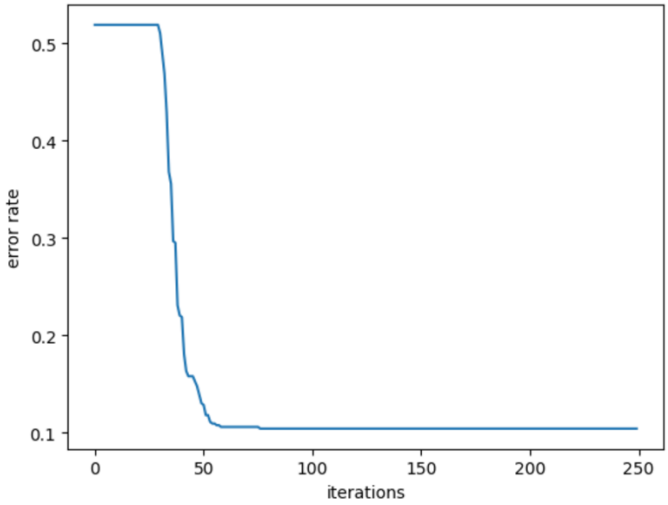


Figure 5: Error through iterations of gradient descent on a simple layer neural network for classification question 1.

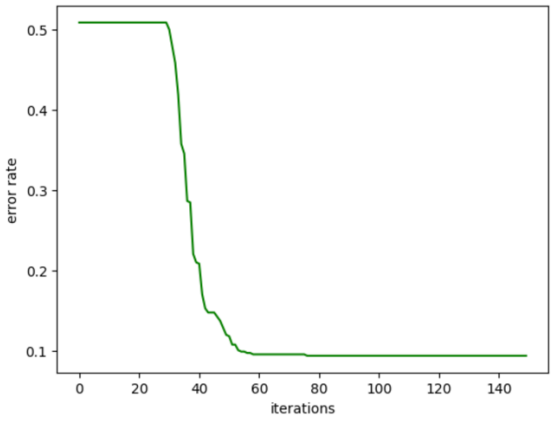
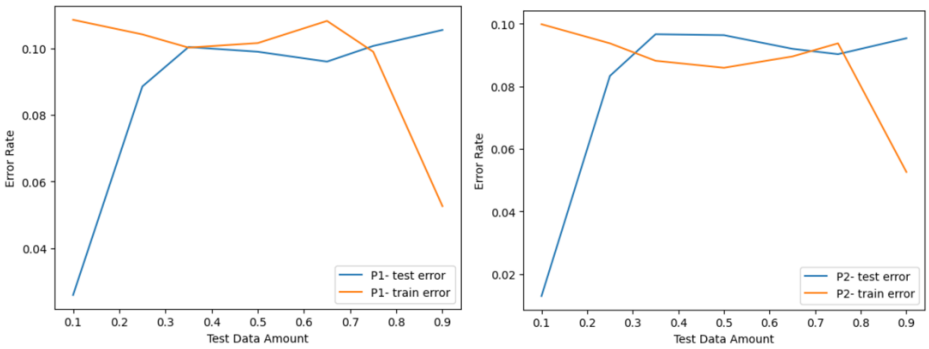
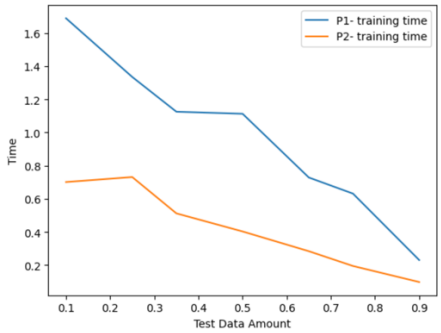


Figure 6: Error through iterations of gradient descent on a simple layer neural network for classification question 2.

For test and train error rates in both classification problems, my first and top thought is that there are not many anomalies in the data. The performance of the test and train data mirror each other very, very closely, deviating by at most 0.10. Of course, as expected the training time decreased with the amount of data fed into it as expected.



Figures 7 and 8: Error rate of test and train data for problem 1 (Figure 7) and problem 2 (Figure 8) for various quantities of training data

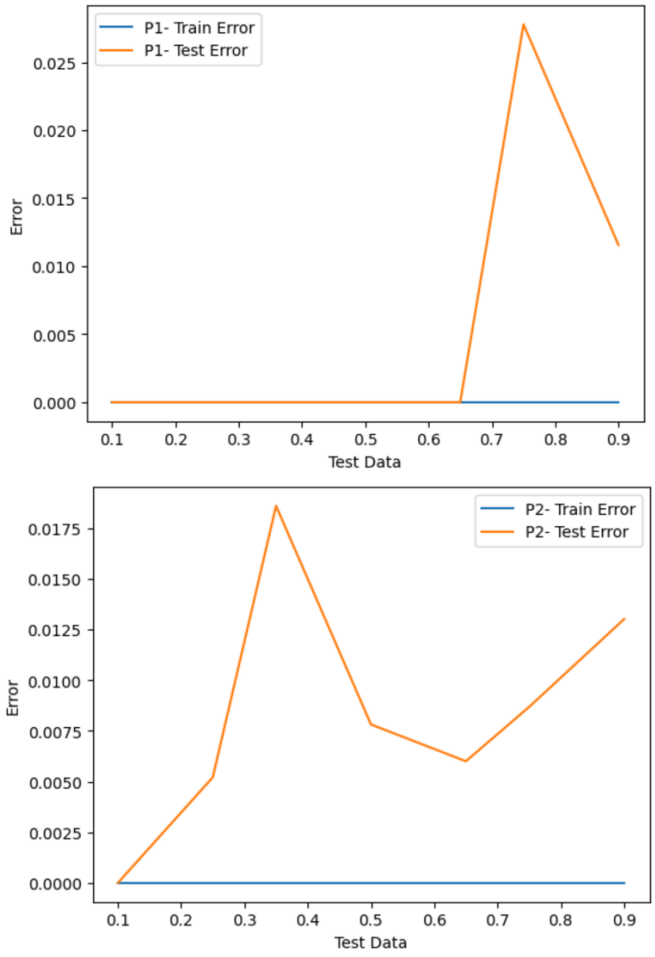


Figures 9: Training time (sec) for Neural Network.

My favorite aspects of neural networks is looking at the final weights given for each model which is as follows in order for the variables: relative compactness, surface area, wall area, roof area, overall height, orientation, glazing area, and glazing area distribution. For problem 1: (0.9982589, 0.9988857, -0.36501914, 0.40280413, 0.61608837, 0.99390616, 0.99651781, 0.99982589, 0.99129452).
And for problem 2: (0.9982589, 0.9988857, -0.36501914, 0.40280413, 0.61608837, 0.99390616, 0.99651781, 0.99982589, 0.99129452). This is very direct insight into the factors that are the most directly connected to the variable being classified, heating load for problem 1 and cooling load for problem 2. However, as much as I want this to be meaningful data it would be more important to normalize the data, have it within a set range for all of the variables before I could truly claim that a nominal comparison of the weight values is valid.

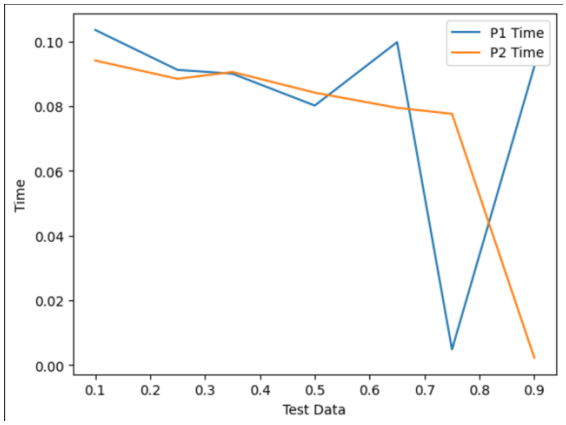
2.3 Boosting

Covering Boosting next. The pre-built scikit solution was used. This means not only was boosting incorporated but the underlying decision tree algorithm is much more "polished" to say the least so going into this type of learning I fully expected a better result than the already strong result I saw with my decision tree learner.



Figures 10 and 11: Test (orange) and train (blue) error classification problems 1 (figure on the right) and problem 2 (figure on the left).

As expected this performed superbly within it's training data set. As stated, going into this learner this was expected because it's boosting built on decision trees which we already know the data is very well suited for. As we can see there's much more error in the test data than we saw with decision trees telling us that the model is overfitted. Insufficient pruning and a greater need of controlling leaf size and tree depth is the likely cause.

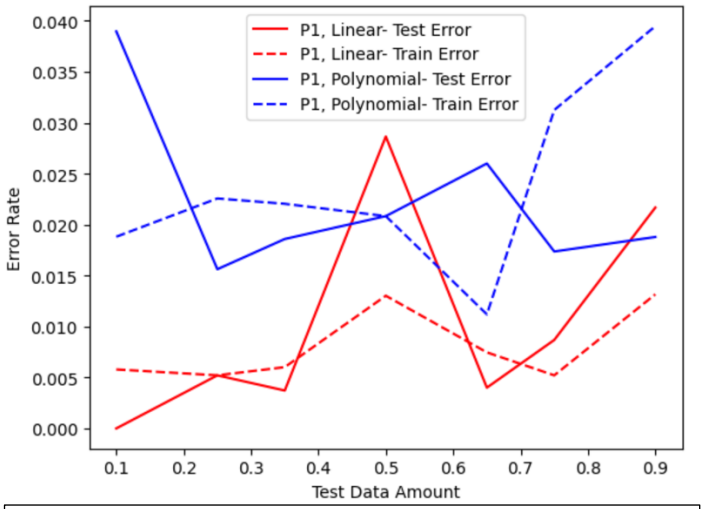


Figures 12: Time to train for P1 (blue) and P2 (orange).

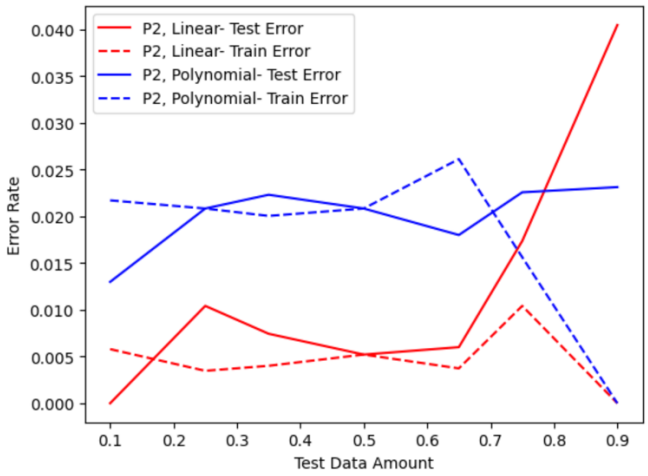
Data around time to train for boosting was a bit strange. Of course as the training data set gets smaller time to train on it should also decrease and we see that expected change with P2 classification problem, however looking at P1 data we have a sudden spike in the time to train. This is similar to what we saw looking at decision trees, something in this data set is harder to classify in smaller bits.

2.4 Support Vector Machines

Covering the Support Vector Machines (SVM's) next. Overall, SVM's had a much lower error rate than work with the decision tree or Neural Network. It stayed consistently below 0.5% error rate. However looking at the test/train error rate, even on such a small scale there were some interesting takeaways here. First the training error was significantly larger than the test error for problem 1, specifically for a polynomial kernel. While this is not expected generally, it makes since that this would occur on the low end of the amount of data being fed into the model and I read this to mean it needs more data to work with than 10% of the set available. For Problem 2 with a linear kernel, error values behave much more like I would expect them to with testing error growing as training error decreases, illustrating the smaller the training sample, generally the less representative it is of the test data.



Figures 13: Test and train error for classification problem 1, both for linear and polynomial (order 8) kernel.

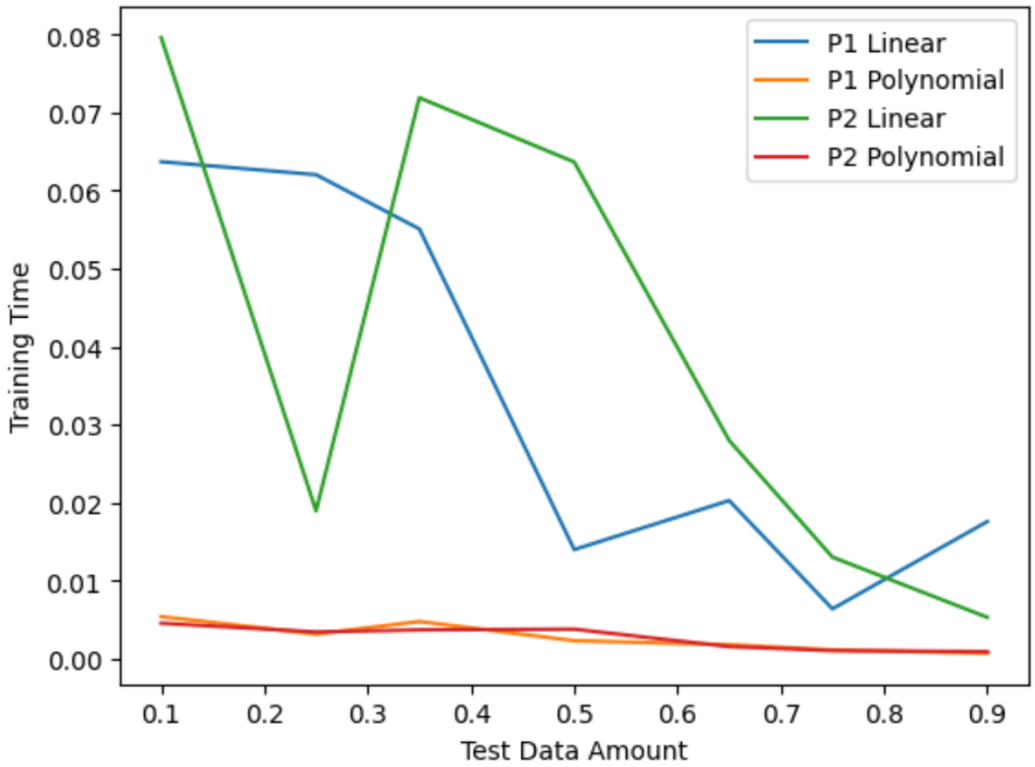


Figures 14: Test and train error for classification problem 2, both for linear and polynomial (order 8) kernel.

The truly interesting part of this learner is the difference that the kernel makes on its performance. Generally in machine learning, higher order functions are thought to model the data better, create a better representation, despite having a risk of overfitting. However, for this model on both of my questions, a polynomial kernel (of degree 8) had consistently more error compared to it's linear kernel counter part. This is perhaps my favorite insight into the data across all of the machine learning models. A kernel at its most basic function in SVM's is a distance measure. Seeing that a linear kernel is performing better than a polynumeric one tells me very clearly that there is a linear relationship between the points and their values. Looking at the variables relative compactness, surface area, wall area, roof area,

overall height, orientation, glazing area, and glazing area distribution- this makes a lot of sense. Basic heat transfer principles are generally linear relationships with factors like area. Seeing this reflected in the two different kernel models is exciting.

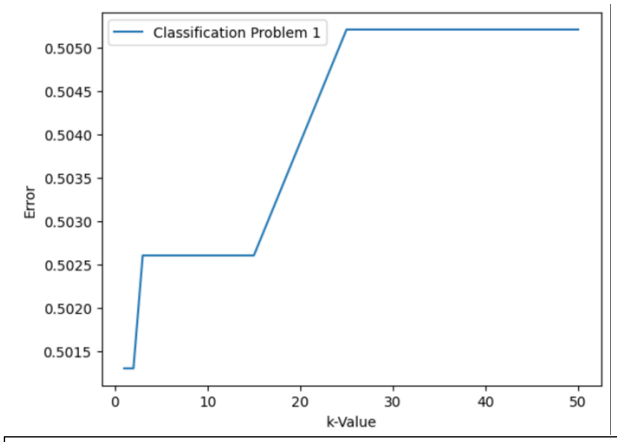
Additional, training time on models performed as expected generally. Polynomial kernels took less time to train than linear. And general time decreased as the amount of data that it was training on decreased as well.



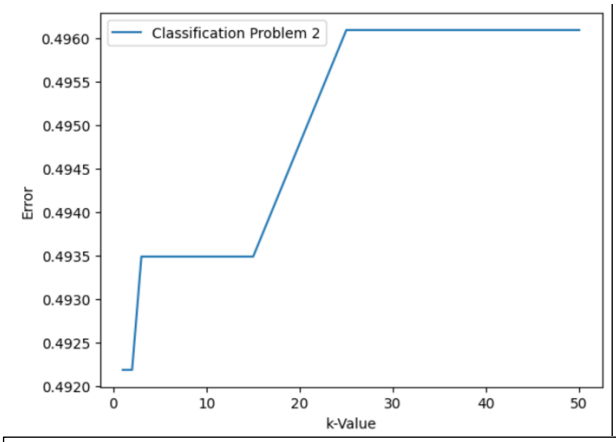
Figures 14: Model training times for both classification problems and both kernels (linear and polynomial order 8)

2.5 k-Nearest Neighbors

k-Nearest Neighbors wins the award for the most surprising results. The data did not model well with this learner, a consistent error rate of around 50% even for varying k's. As you can see from the following figures, smaller k values did fare better.

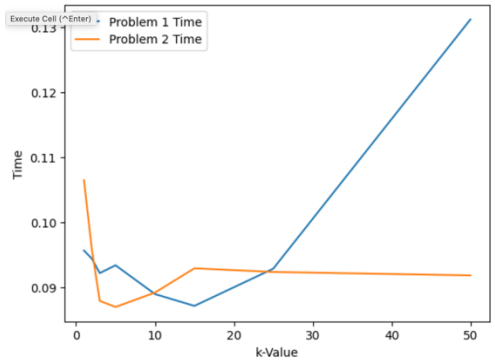


Figures 15: Error rates for kNN, classification problem 1 with varying k values.



Figures 16: Error rates for kNN, classification problem 2 with varying k values.

Given these results and even comparing them to other more successful learners the error rate is a bit of a surprise, but also good information on the data. A data point is *not* similar to it's closest surrounding data points, with the caveat being *at least not in the way 'closeness' was measured this time around*.



Figures 16: Time measurements for processing KNN algorithm with varying K's.

The time data is not what I would expect either, I would expect larger groups of data to take longer to process, meaning longer times at higher K's as is shown for problem 1, however, KNN is such a quick algorithm that looking at the activity on such a microscopic level feels arbitrary to its performance.

3 ANALYSIS

A lot the reasons behind why I got the results that I did has to do with the models used and more importantly in the way that they were developed and implemented. Perhaps the best example is with the decision tree learner, the impact that the use of entropy or the GINI index in place of my remedial splitting based on correlation and mean is huge. The graphs and results would look much different with different logic supporting the learner. While my data set is more simplistic as illustrated by how well it was learned on with my existing decision tree algorithm, I'm still curious how it entropy would affect its results.

Thinking about improvements I would make, first and foremost would be incorporating entropy into my decision tree learner. On my neural network, the data is clear that it needs a layered network, I would go back and add hidden layers to see the impact it has on the model, maybe even try my hand at backpropagation with a sigmoid function in place of my discrete decision function. For SVM's I would incorporate different Kernels, perhaps iterating through different degrees of polynomials and seeing how that affects the resulting model and error. Despite seeing better results with the linear kernel than polynomic, the fact that my data has 8 input variables makes me wonder about different kernels still and I would iterate through Gaussian, Sigmoid, etc.

For KNN, I would want to weight the values that I find in my algorithm as opposed to simply taking the mean. I would hope that this would make a difference but I suspect it's more likely that the distance measure I used does not accurately capture the data. Comparing my KNN results with my SVM results this makes a lot of sense. The kernel acting as a measure of similarity worked best with a linear set up. Meanwhile for KNN I used linalog to measure the distance between two points and the result was unclear classification with a high error rate. This is good insight into the way different data points relate to each other going forward.

Across the board the results were timely, not much time spend waiting at the computer for processing to be complete. While this may be a result of picking a data set that is clear and concise to what I was looking for, I will take it to mean the code is clean and clear. Iteration wise, I was disappointed in my neural network performance. Looking back at figures 5 and 6, there is a long lag time before the learner begins to move around and improve error. Assuming this is due to it starting near a local minima/maxima different initial weights and different learning rates could help. If I were given more time, I could see myself iterating through different values for both initial weight as well as learning rate to find which one begins to move error the soonest. For this assignment I did look at different training rates in a less formal weight. Dealing with 8 variables, targeting values of -1,1 lead me to a much smaller learning rate than I initially put. The pros and cons to this are a bit

frustrating- unfortunately it requires many more iterations and takes a significant amount of time to "move" the error around. But, at higher error rates the gradient and weights quickly became too large to manage, spiraling out of control. Given that the neural network error also flattens at a less than impressive error rate I would guess that it hit another local minima/maxima and would want to incorporate annealing into the optimization to prevent this issue.

The best results could be any of the models I choose depending on what we want to measure "best" on. Lowest error? Decision trees. Insight into data? Neural network (presence of hidden layers), SVM's (linear kernel performance), or *k*-NN (not how to classify this data set). For the sake of the assignment I will say the decision tree lead to the best model, quick, simple, and pruned. However, my favorite model and favorite take away is with neural networks, a new topic for me. Looking at my model in hindsight, it needs hidden layers. From a data point of view, I'm seeing that different variables are combining to form almost "subvariables" in the form of hidden layers with their own firing thresholds. It paints a really great picture of complex this data is. Very different than the beautiful and simple picture decision trees paint.