ML Project 1: Supervised Learning

Charlie McKnight
CS 4641
2/5/2016

Introduction

The purpose of this paper is to analyze the performance of various machine learning algorithms on two different data sets with a focus on limiting training time and reducing complexity of models while still achieving a high classification accuracy. Weka was used to perform the analysis of the algorithms. The algorithms which I analyzed were Neural Networks (Multilayer Perceptron), KNN (IBK), SVM (SMO), Decision Trees (J48), Boosted Decision Trees (AdaBoostM1, J48/ Decision Stumps)

Classification Problems

Phishing Website Data

The first dataset contains a variety of attributes about websites and a classification as either a phishing website or not. There are 30 attributes in total including values such as being listed on google, and having a registered domain. All of the attributes are boolean values, some having a third state indicating an unknown or missing field. In total there are 11055 data points. 4890 (~44%) of the data points are non-phishing websites. This leaves 6157 (~56%) data points representing phishing websites. This data set originally stuck out to me due to it having been collect from real world data meaning that noise in the data would be realistic and the results collected are in theory actually useful. The second reason it stood out was a relatively high number of attributes for a relatively low number of datapoints. This is especially true when compared to the second data set.

1996 Census Data

The second dataset is a subset of the 1994 Census data. This is also a binary classification problem, the goal of which is to identify whether a person makes >$50,000 a year or <$50,000 a year. There are 14 other attributes including age, education, race, and marital status. There is also a large variety of attribute types, ranging from booleans such a sex to continuous data such as hours worked per week. This is a larger data set with 48,842 data points. Another consideration is the breakdown. 11,687(~24) percent of the data represents incomes larger than 50,000. This leaves a large majority of 37,155 (76%) with an income less

than 50,000. I liked this data set because it is also real world data and I was interested to see the accuracy that could be achieved.

Comparison

These two datasets play off of each other well for a variety of reasons. The first reason is the size of the datasets. The Census Data is roughly 4 times the size of the Website Data but also has half the number of attributes. Another large difference that is interesting to note is the makeup of the attributes. The Phishing data's attributes are all boolean values while the Census data has a large range of attribute types. These difference make the datasets not only interesting in and of themselves but also interesting to compare.

Learning Curves

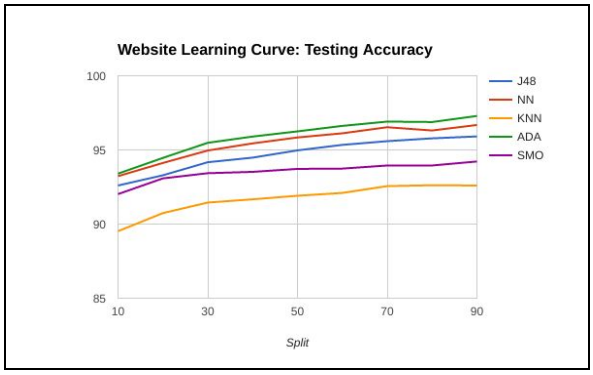
Different to how many would have approached the problem. I started with doing learning curves first. The reason for this being that in my preliminary exploration of my data, I realized that with the computing power I had running all of the models on too large of a training sample set would mean that it would take weeks just to train all of the models I needed. Thus I decided to start with learning curves so that I could decide what an appropriate training / testing split would be.

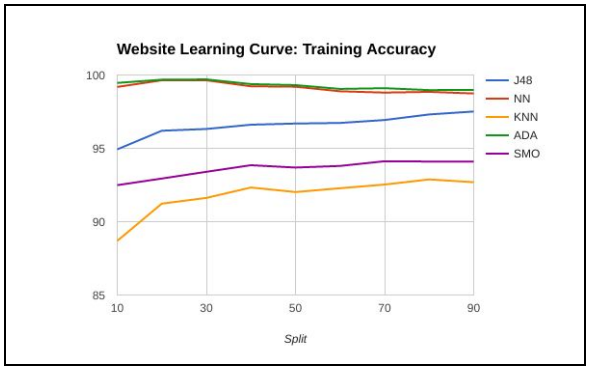
I ran the five algorithms on the two datasets with with training sizes starting at 10% and going to 90% with an increase of 10% each time. Each algorithm was run 10 times with the data divided randomly each time. The accuracies were then averaged.

The exact parameters of each model are listed below

1. Decision Tree (J48) - Pruning, confidence factor = .25, num_of_folds = 3
2. Boosting (adaboost-m1) - 10 iterations, Decision tree from above
3. Neural Network (multilayer perceptron) - hidden_layers = a, LR = 0.3, momentum = .2, training time = 500
4. SVM (SMO) - c=1, RBFKernel
5. KNN (IBK)- k=101

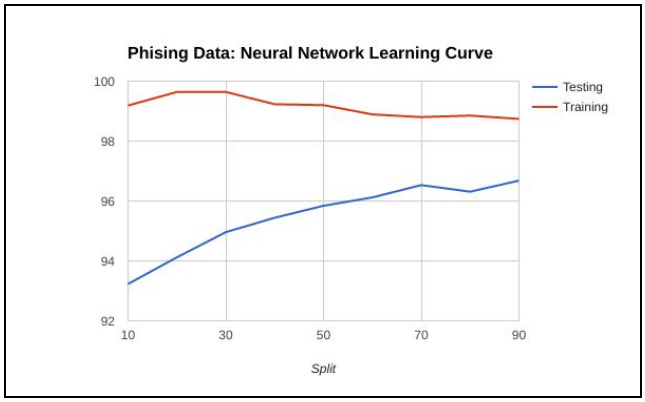
Website Learning Curves





As one would expect when testing on the training data the accuracies tended to go down as the split became larger for Neural Networks as it was harder to fit the data without increasing the complexity of the model. Also as one would expect, KNN saw the largest increase in accuracy as the training set became larger.

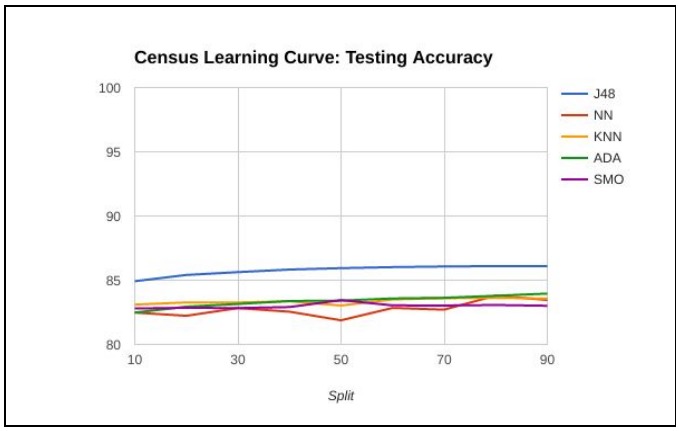
The Testing Accuracy Graph tells a different story, with every graph showing improvement as the split of the data grows larger following a logarithmic pattern. However, many of the models continue to show relatively large improvement even at the edge of the graph. This seems to indicate that collecting more data would improve the performance of the models and that when experimenting I should choose a large percent for the training data.

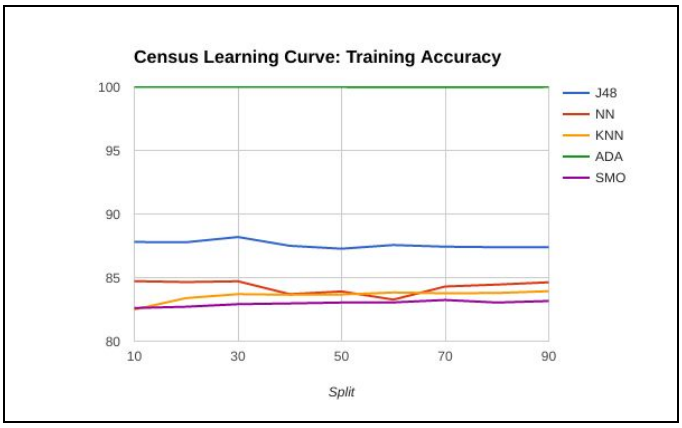


This is confirmed by looking at the Neural Network Learning Curve when pulled out on its own. This graph indicates High Variance since the Training and Testing curves are so far apart and the Testing Curve seems to be continuing its upward trajectory.

Since the models seems to universally perform better with better data I decided to segment into 70% training data and 30% testing data. The reason I went with 30% testing data is to avoid bad performance simply because as the testing set becomes smaller noise begins to play a larger role just as it does in training. Training time for this dataset did not factor into my decision on the split since this data set is considerably smaller than the next data set and the longest training time (Neural Network: 90% split) was under two minutes.

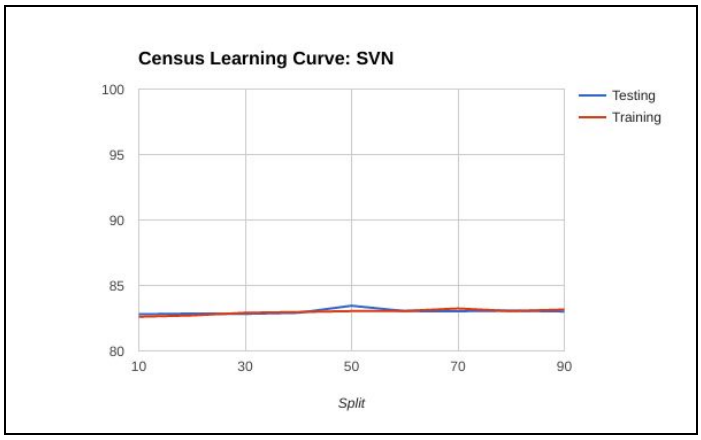
Census Learning Curves





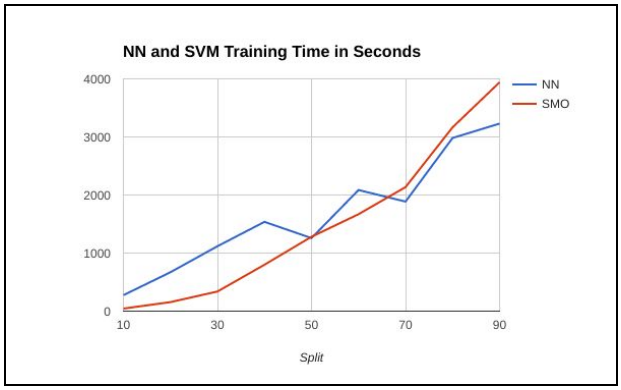
The Census Learning Curves off the bat seem convey a different message than the Website Learning Curves. While the Website Curves seemed in indicate a change over time, the learning curves of the census data do not show much change at all either in Testing or Training Accuracy.

This seems to indicate high bias. Since we have high bias, collecting more data would not help us to better fit the the underlying model. In addition we do not see much of a curve at all not matter how small the data gets. This is shown even better when we pull out the SVN Learning Curve.



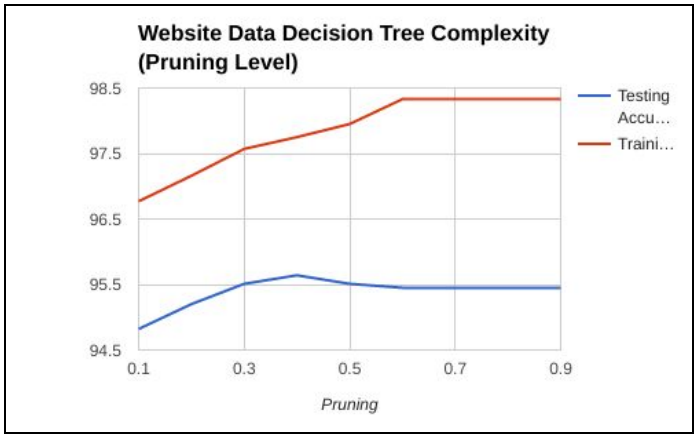
The fact that the curves are on top of each other shows that we are underfitting the data and that more data will not improve our results. Instead focusing on increasing the complexity of the models is the best course of action. This also seems to indicate that potentially we are missing some informative attributes which will limit our success in the long run.

It is also worth noting at this point that the one exception to the general trend of high bias in the data is with the Boosted Decision Trees. The learning curves for the Boosted Decision Trees indicate large overfitting even in the largest data sets with the training accuracy being 100% across the board.



Unlike the Website Learning Data which was relatively small, The Census data is large and training time will prove to be a large issue particularly with Neural Networks and SVMs.

As we can see in the training time graph, training one instance of an SVM or NN with 90% of the data can take over an hour with cross validation. Once we start performing Cross Validation on our training set and increase the complexity of the model (as we saw was necessary previously) the training times will quickly pass to unreasonable levels with the computing power and time frame given.



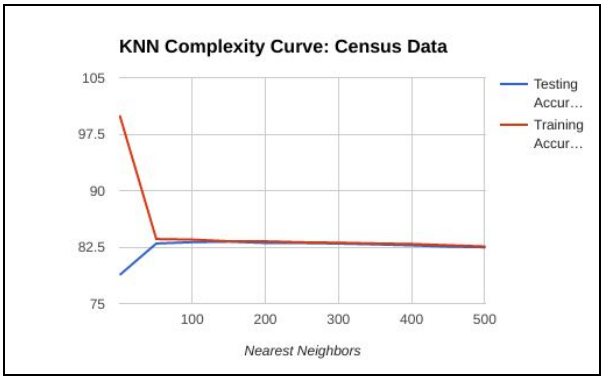
For this Census Data Set I decided to make the training set be 30% of the data and the testing set to be the remaining 70% of the data. The major reasons for this the fact that increasing the amount of data did not indicate an increase in performance and the training times passed to infeasible levels.

Complexity Curves

Complexity curves were calculated by tracking performance on the training data as well as the cross validation accuracy. For cross-validation I chose a split of 10 which struck a fine line between training time and limiting the effects of noisy data. I chose to cross validate because it allows you to “use” more of your training set and limiting the effects of a bad split in the data.

KNN

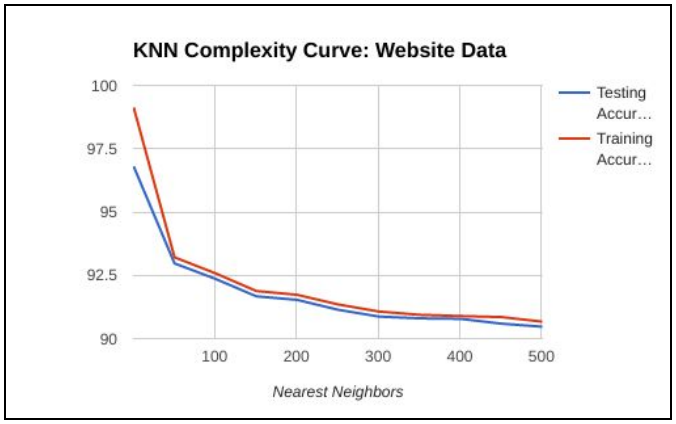
Census Data



To change the complexity of KNN I went with the obvious choice of changing the K. Obviously this is not a traditional complexity graph in that a larger K is usually more likely to underfit the data. As one can see from the graph a small K yields a sort of overfitting in which it responded too highly to its neighbors. This caused the accuracy to drop. However, when K was increased to around 50 we see a sort of local max and then a slow descent as too many neighbors are included.

Due to the variety of types of attributes in the Census data the distance calculation could have proved to be an important parameter. However, when experimenting, the algorithm chosen did not seem to provide much of a change in accuracy. I attribute this to the large diversity in types of attributes. In addition many of the attributes do not have a meaningful distance calculation. One example of this would be country of origin. Perhaps a custom written distance function could improve performance but that is outside the scope of this paper.

Phishing Data

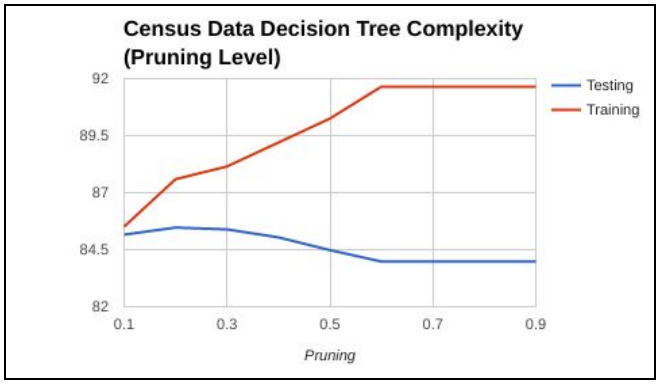


The Phishing Data KNN Complexity Curve is a more interesting graph for a variety of reasons. As one can see at first glance the curves mimic each other to a very high extent. In addition as the number of neighbors increases the accuracy of both graphs fall off very quickly. In fact one nearest neighbor results in the highest accuracy.

For the Phishing data, since all attributes are booleans, the distance algorithm has no effect so sticking with a euclidean distance algorithm is what I stuck with. This makes sense since the distance between attributes is constant.

Decision Tree

Census Data



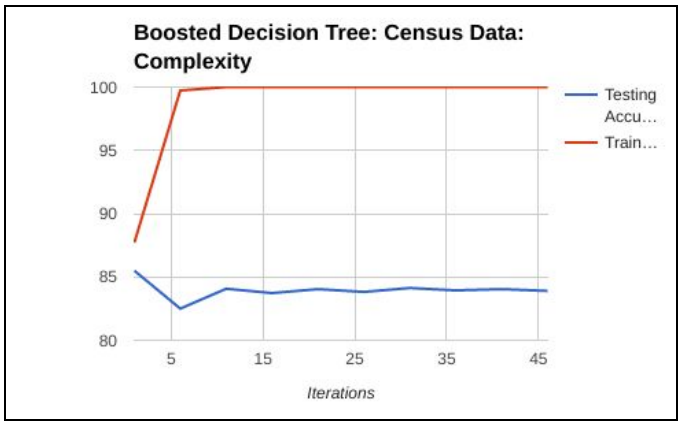
For the decision tree's complexity graph I decided to graph the confidence factor. A higher confidence factor indicates less pruning. We can see from the graph above that the curves begin to diverge at a confidence level of 0.2. This is where our tree optimally fits the data. Any less pruning and the trees begin to overfit and less and we are losing useful data.

Phishing Data

For the Phishing data we can see a similar curve however the point of divergence is at a slightly higher confidence value. This occurs at a confidence factor of 0.3.

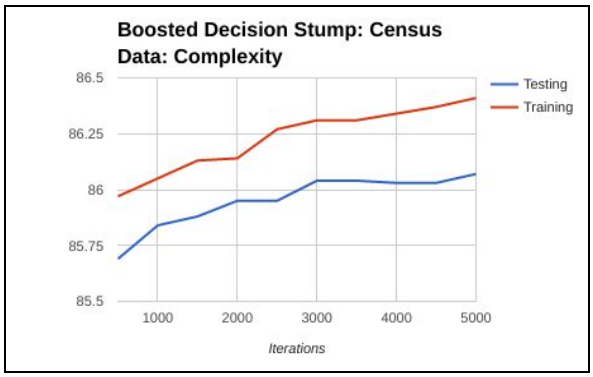
Boosted Decision Tree

Census Data



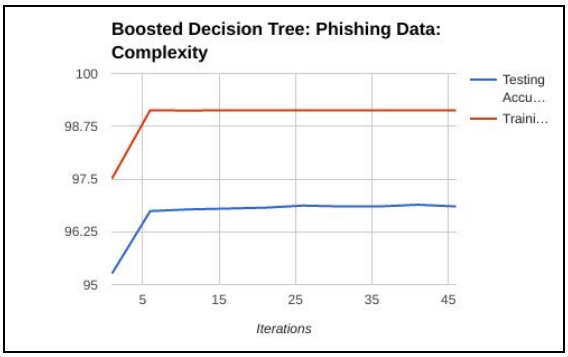
For boosting I started out with the standard J48 Decision Tree and boosted off of that. To change the complexity I changed the number of iterations. As we can see from the graph the idea lnumber of iterations is one. This means that any boosting of the J48 Decision Tree results in overfitting. To compensate I turned the pruning up on the underlying tree to very high. This resulted in a graph that mimicked the original.

Having decided that a J48 decision tree was too complex of a model to boost of I resorted to switching the underlying model to a less complex one. In this case I chose a Decision Stump to keep it in the Decision Tree Family.



The Decision stump solved the problem of overfitting that the tree had. It also improved the accuracy from maxing out at 85% to an accuracy of over 86%. This graph seem to diverge at 3000 iterations an astronomical number when compared to the one iteration of the Decision Tree.

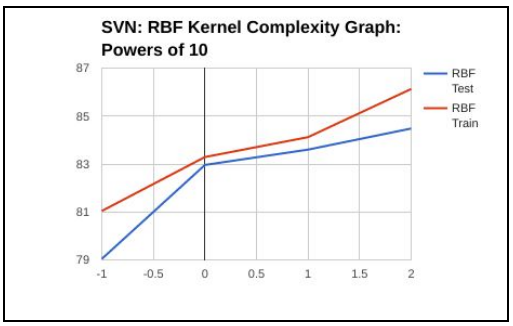
Phishing Data

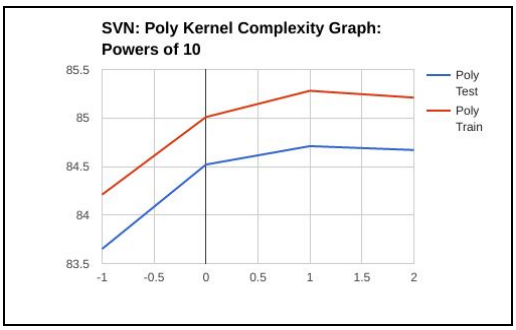


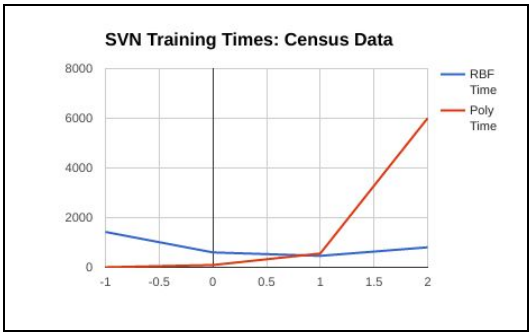
The Phishing data did not have the same problem of immediate overfitting that the Census Data Experienced. In fact the accuracy seemed to top out at around 5 iterations and remained relatively constant from then on. Since Occam's razor holds especially true in machine learning I decided to stick with 5 iterations.

SVM

Census Data





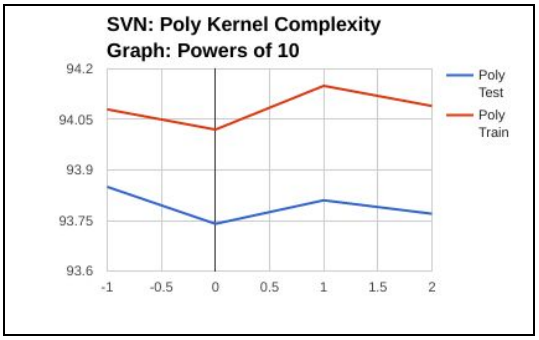


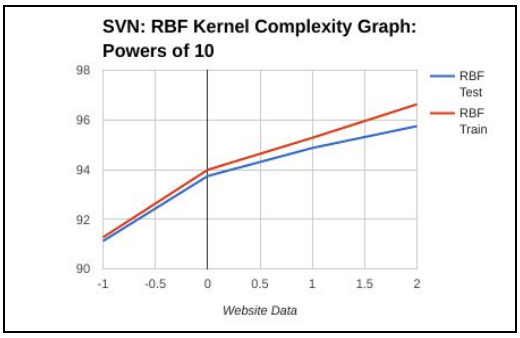
For the SVM complexity curves I decided to experiment with two different kernels, a polynomial Kernel and RBF Kernel and adjusting the complexity in powers of 10. As one can see from the graphs the polynomial Kernel performs at a higher level initially however seems to reach peak accuracy at a c value of 10. The RBF Kernel on the other hand starts at a lower accuracy which continues to increase eventually passing the Polynomial Kernel.

This seems to indicate that data is partially linearly separable with a polynomial kernel but that there is a limit to how much it can improve. The RBF Kernel on the other hand seems to perform oorly when allowing for a higher tolerance in the margin. However as the tolerance is made more strict RBF is able to find a better separation than the polykernel.

Unlike with the previous 3 models SVM (and later Neural Networks) training times are a concern with SVM's. As one can see the training time for the polynomial kernel appear to increase exponentially along with the complexity. The accuracy however, does not continue to increase. The RBF Kernel does not experience the same increase in training time.

Phishing Data

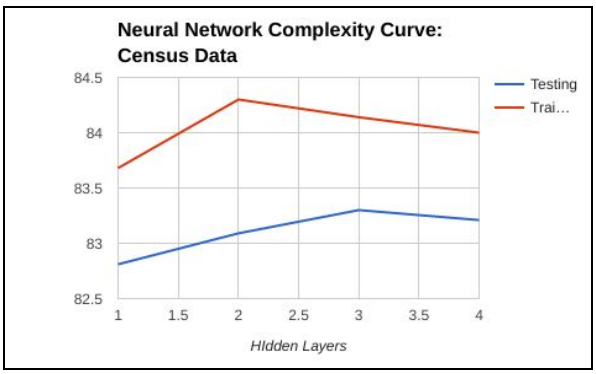


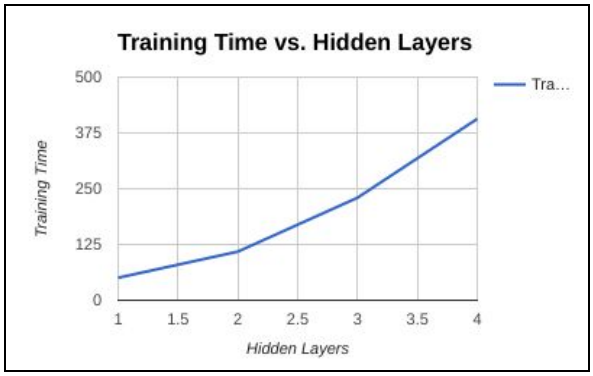


The SVMs on the Phishing Data seem to indicate that a polynomial kernel is not a good choice and that the RBF Kernel outperforms it fairly handily. This seems to be because RBF deals in locality and as we previously saw with KNN locality is a very good predictor for this dataset.

Neural Network

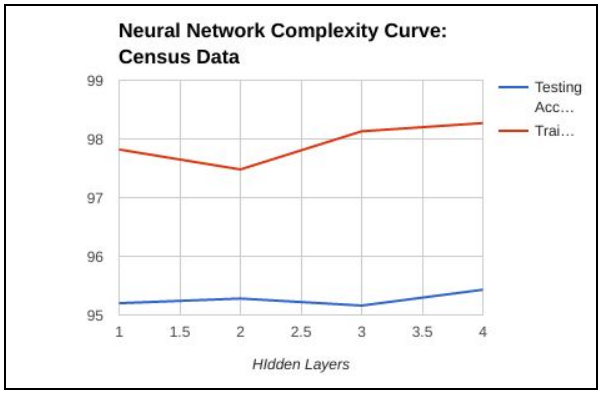
Census Data





To increase the complexity of neural networks I experimented with adding more hidden layers. This proved to provide a marginal improvement but at a high cost. As you can see from the graph the training time seems to be increasing exponentially with the number of hidden layers. Also tested but not displayed are a variety of momentums and nodes per layer. These graphs are taken out for brevity. Ideal values were momentum=0.5, nodes=8

Phishing Data



The phishing data follows a similar trend. The number of hidden layers does not yield a large increase in accuracy. Though also does not seem to indicate overfitting to a large extent due to the Training accuracy not growing at a large rate. I chose to have 2 hidden layers since this seemed to strike a good balance between training time and correct categorization. Ideal momentum was found to be momentum=0.6, nodes=19, data not shown for brevity.

Comparison Of Algorithms

To analyse the final results I trained the algorithms one last time a testing on the testing data that I had previously split upon.

Census Data

Model on Census Data	Accuracy on Testing Data	Training Time
KNN: k=51, distance_weighting	83.55	0
Decision Tree: confidence_level=0.2	85.86	1
Boosted Decision Stump: i=3000	86.26	58
SVM: RBFkernel c = 100	85.09	92.78
Neural Network: m=0.5 n=8 hl=3	84.15	124

After tuning the algorithms it appears that The Decision Tree is the stand out model for this dataset. When boosting is applied it reaches a correct categorization accuracy of 86.26%. I attribute this success to the fact that many of the attributes in the census data are discrete and unrelated such as country of origin. Decision Trees perform well on these types of attributes. However, this is not a staggeringly good performance since simply categorizing everyone as

having an income below $50,000 would yield an accuracy of roughly 75%. I attribute the relatively large error to untracked attributes which contribute noise to the data.

It is also important to note that the more complex and time intensive algorithms actually performed worse that the simpler algorithms even with a large amount of data. This is evidence for the statement that a simpler model can and often is the better choice.

Phishing Data

Model on Phishing Data	Accuracy on Testing Data	Training Time
KNN: k=1, distance_weighting	96.89	0
Decision Tree: confidence_level=0.3	96.35	1
Boosted Decision Tree: i=5	96.92	1
SVM: RBFkernel c = 100	96.14	28
Neural Network: m=0.6 n=19 hl=2	96.11	192

As opposed to The Census Data, The Phishing Data was able to achieve spectacular results, Improving on the default split of around 55% to well over 96% accuracy. Interesting enough almost every model was able to achieve roughly the same accuracy once tuned, with KNN and Boosted Decision Trees being the highest by a slight margin. Since all achieved relatively the same accuracy the simplest model is usually the best. For that reason I would choose KNN which has no training time and a tiny calculation time given our K.

In real world application of the data it could also be advantageous to look not only at the algorithms but also the ROC curve for each algorithm to see which algorithm could be tuned to have the highest accuracy while limiting false positives or false negatives. For my data however since I had no preference one way or the other I chose to have the default margin.

Conclusion

Both datasets were different on paper. One had a large amount of data and a small number of attributes. One had a relatively small amount of data and a large number of attributes. The types of attributes were also different. Even the results surprised me with Phishing data having an incredible accuracy with its more limited amount of data and the census data being more difficult to learn. However, the common theme that ran between both datasets was that the simplest model is often times the most effective. By using the simpler models we achieved high accuracy and incredibly fast learning times. This result could open the door for attempting machine learning problems locally on low power machines, such as smartphones in my own personal projects.