# CS4641: Supervised Learning

Paras Jain

February 2016

## 1 Datasets

### 1.1 Political party classification of Congress

#### 1.1.1 Structure and Source

This dataset[1] is data used to attempt to classify members of the House of Representatives in 1984 as either a Democrat or Republican. Features in this dataset include the individual votes for 16 bills (16 attributes). Each instance contains one binary class, either Democrat or Republican. There are 435 instances total. Data for this was sourced from the Congressional Quarterly Almanac. This dataset is incomplete and has missing values (congressman was not present/abstained from voting).

#### 1.1.2 Motivation and Interest

Political party classification is interesting because it is a presidential election year. The presidential primary this year includes candidates who are outside traditional party lines so it will be interesting to examine how a candidate's voting record aligns with their party membership.

This dataset is interesting from a machine learning perspective as it is a small dataset yet is higher variance with more noise. Each of the attributes are discrete and binary (yes or no). This dataset is clean and fast to train on. Having strictly binary attributes makes this dataset perhaps well suited for some algorithms like decision trees. The number of features is of moderate size and is smaller than the Hypothyroidism diagnosis dataset yet still contains enough features for the algorithms to have a chance at extracting some signal. The limited number of instances will make this dataset interesting to compare to the second dataset which has an order of magnitude more instances. This dataset also contains missing values - this will allow analysis into how algorithms perform on partial datasets. Political party classification is interesting as it is an election year. It is interesting to see how closely partly lines influence votes. One issue, however, is that this data is strictly from 1984 - this may impact external validity by preventing extrapolating this data to current candidates. Moreover, the attributes in this dataset are bills voted on in 1984 which would only apply to

[1] Congressional Voting Records Data Set from UCI Machine Learning Repository. `https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records`

congressmen at that time. Mis-classifications would be interesting to study in order to see which members of Congress are outliers from their party.

## 1.2 Spambase

### 1.2.1 Structure and Source

This dataset[2] is data used to attempt to classify emails as spam or non-spam. There are 4601 instances and 57 attributes. The labeled class is binary (spam/not-spam). Attributes include frequency of key words and various interesting properties of the text (e.g. length of sequences of capital letters, character frequencies, etc.). Attributes are real continuous values. The spambase dataset is complete (all attributes have values for each instance).

### 1.2.2 Motivation and Interest

This dataset, compared to the political party dataset, is interesting as it is larger and nosier. Variance is higher for this dataset as compared to the political party classification dataset. Attributes are primarily continuous variables which should allow comparison between discrete and continuous values.

There are many more attributes which will may potentially allow for studying the impact of the curse of dimensionality. Manual inspection of the dataset shows some of the attributes may not be particularly informative of class so it will be interesting to examine how algorithms perform given a number of sparse attributes.

Spambase will also allow for studying the scalability of various machine learning algorithms over a larger number of training instances. With the increasing scale of data available, highly scalable machine learning algorithms may see more usefulness.

## 2 Analysis of Algorithms

Minimal cleaning of the data was needed. Data was imported into Weka and classified for each of the tasks. Data was split into separate test/train sets using Weka's *RemovePercentage* filter.

## 2.1 Decision Tree

The J48 algorithm was chosen as it allows for both non-pruned and pruned decision trees. In order to control the level of pruning, a confidence parameter can be modified. J48 is very similar to ID3 in that it uses information gain as the way to select which attributes to classify with first.

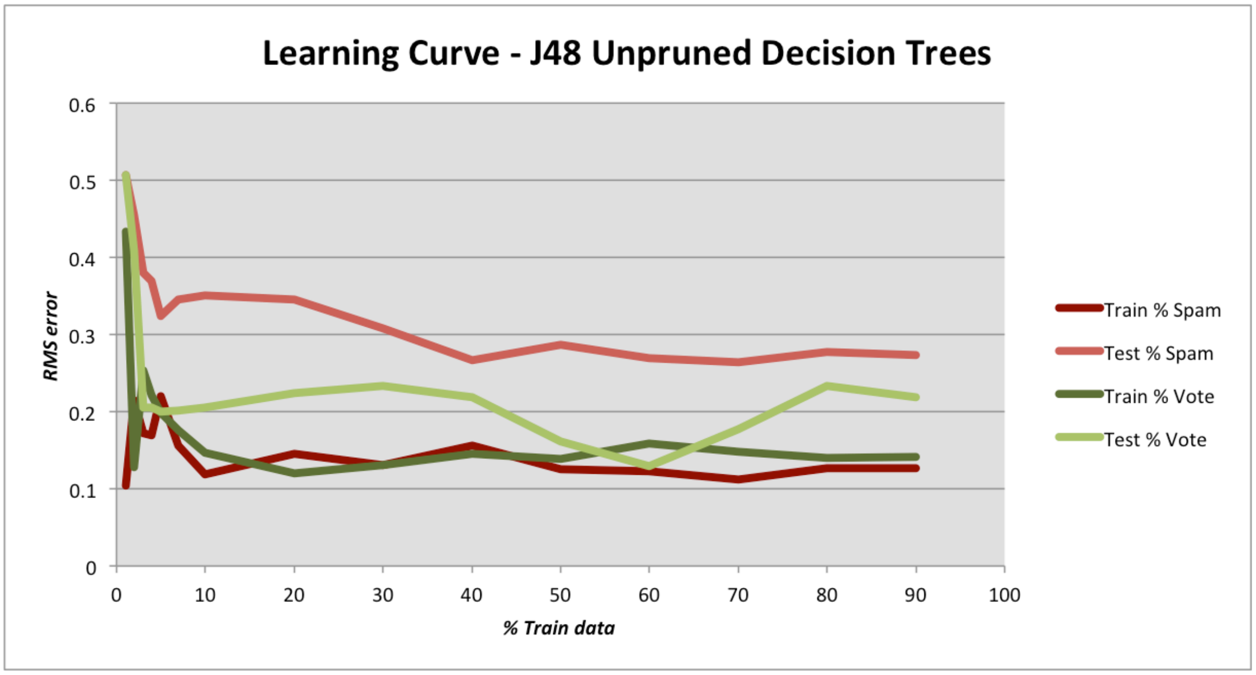
[2]Spambase Data Set from UCI Machine Learning Repository. `https://archive.ics.uci.edu/ml/datasets/Spambase`

Table 1: Voting classification performance with a 70%-30% Train-Test split using J48

| Pruning | Pruning Confidence | Training % | Testing % | Leaves | Tree Size | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|---|---|
| Yes | 0.0625 | 95.41% | 95.15% | 2 | 3 | 0.05 | 0.01 |
| Yes | 0.1250 | 95.41% | 96.15% | 2 | 3 | 0.03 | 0.00 |
| Yes | 0.2500 | 96.39% | 96.15% | 5 | 9 | 0.03 | 0.00 |
| Yes | 0.5000 | 96.39% | 96.15% | 5 | 9 | 0.04 | 0.00 |
| No | — | 97.05% | 96.15% | 8 | 15 | 0.03 | 0.01 |

Table 2: Spambase performance with a 70%-30% Train-Test split using J48

| Pruning | Pruning Confidence | Training % | Testing % | Leaves | Tree Size | Train Time (s) | Test Time (s) |
|---|---|---|---|---|---|---|---|
| Yes | 0.0625 | 95.56% | 92.17% | 50 | 99 | 0.57 | 0.02 |
| Yes | 0.1250 | 96.62% | 92.53% | 70 | 139 | 0.82 | 0.02 |
| Yes | 0.2500 | 97.27% | 92.68% | 85 | 169 | 0.75 | 0.02 |
| Yes | 0.5000 | 98.60% | 92.82% | 139 | 277 | 0.67 | 0.03 |
| No | — | 98.60% | 92.90% | 154 | 307 | 0.73 | 0.03 |



The voting dataset is perhaps ideal for a decision tree since it is composed of purely discrete binary attributes - yay or nay on a particular bill. With quite a small decision tree, J48 was able to achieve quite high testing accuracy. Given the small number of instances, it appears that J48 reached perhaps the maximum performance on this particular random test-train split as all algorithms seem to hit a maximum performance around 96.15%. The process of maximizing information gain interesting tells one that the Physician Fee Freeze bill was the most divisive across party lines with almost all democrats voting no and almost all republicans voting yes. The J48 algorithm placed that attribute at the root. Given the structure of the dataset with purely binary attributes, I expected decision trees to be a natural fit and it appears it did fit the data well. Interestingly, pruning did not have much of an effect on the algorithm. Manually inspecting the generated decision trees, it appears that this is because items closer to leaves provide relatively low information gain (e.g. Anti-satellite testing ban bill). Pruning confidence did not change results to a great degree - this is likely because the trees are already relatively small so there is little point to pruning.

Overfitting is apparent in the spambase dataset - as pruning increased, training error decreased without decreasing testing error substantially. I suspect that a larger test set may even reveal improved testing performance as this dataset is a poor sample of email spam (spambase only contains spam data from HP). It is interesting to see attributes like frequency of the dollar sign, the word 'remove' and exclamation marks were close to the root and therefore indicative of spam.

Decision trees are a very fast technique to use, especially at runtime - regardless of the number of training examples, decision trees only take a few hundredths of a second to run on a modern laptop. Decision trees are obviously a class of eager learners but it was surprisingly scalable with regard to number of instances. In addition, it seems that J48's use of information gain and pruning lessens the impact of the curse of dimensionality.

Analyzing the learning curve, it is interesting to note that the spambase dataset could benefit from more training samples. The test-train gap is large and training on more data should continue to decrease test error. The vote dataset converges to almost touching and this indicates that more training samples may not help as much as with the spambase classification dataset.

## 2.2 Boosting

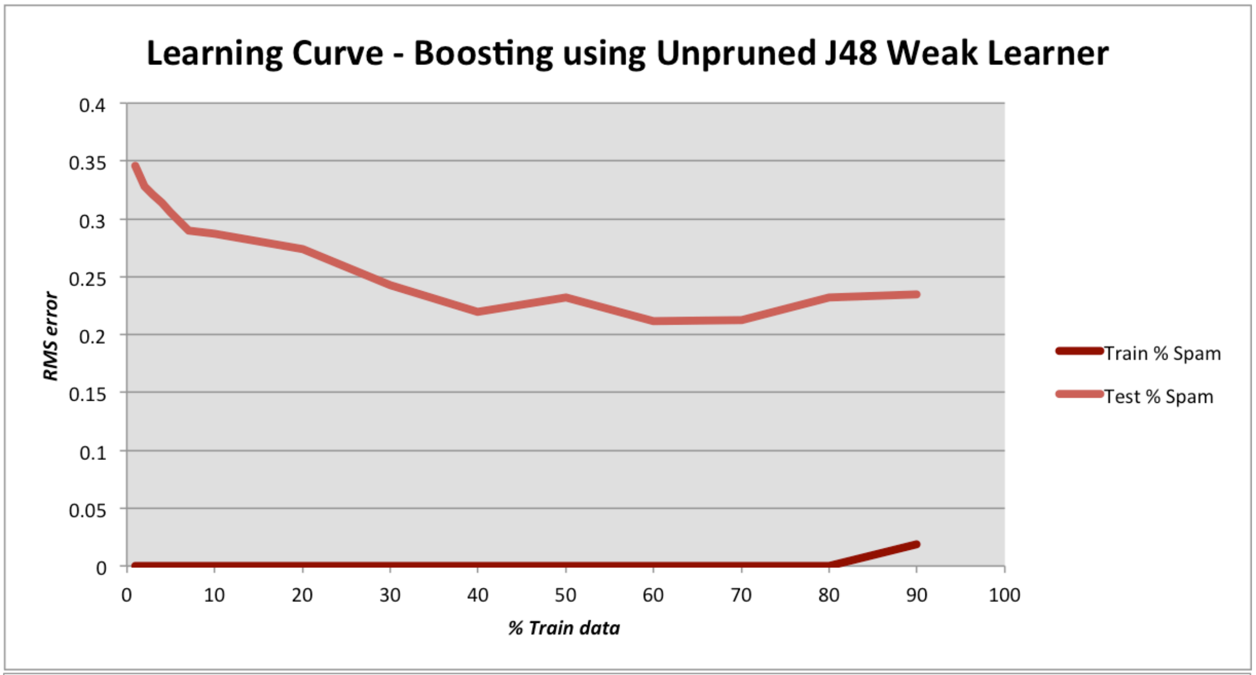
AdaBoost is used as a boosting algorithm, which has already been implemented in the Weka machine learning tool. Decision trees are used as the weak learner (specifically J48). As shown in class, boosting tends not to overfit. In addition to the number of boosting iterations run, pruning parameters were varied for the weak learner (J48).

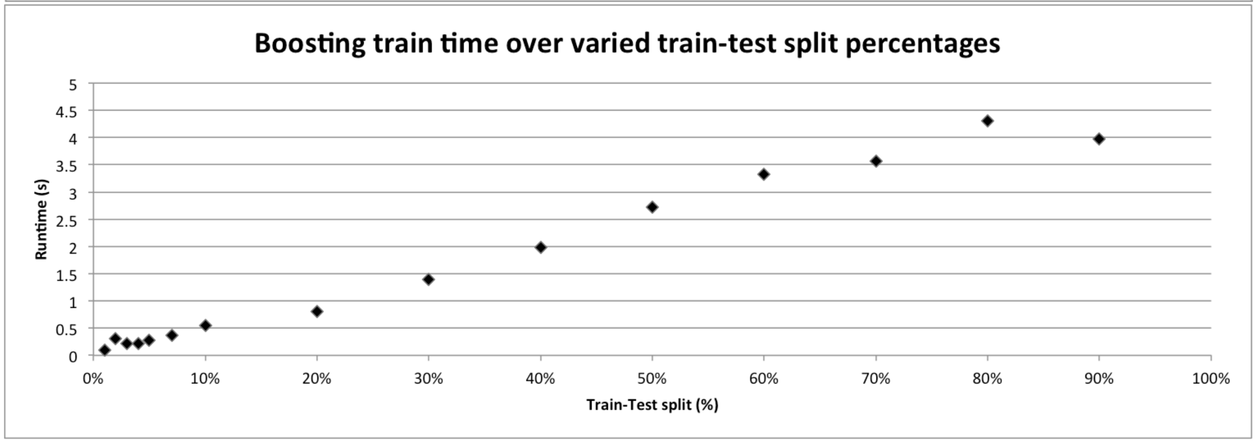
Table 3: Voting classification performance with a 70%-30% Train-Test split using AdaBoost

| **Pruning** | **Pruning Confidence** | **Boost Iterations** | **Iterations Performed** | **Training %** | **Test %** | **Train Time (s)** | **Test Time (s)** |
|---|---|---|---|---|---|---|---|
| Yes | 0.125 | 10 | 9 | 97.70% | 96.92% | 0.12 | 0.00 |
| Yes | 0.125 | 50 | 9 | 97.70% | 96.92% | 0.17 | 0.00 |
| Yes | 0.125 | 100 | 9 | 97.70% | 96.92% | 0.13 | 0.00 |
| Yes | 0.250 | 10 | 10 | 98.69% | 96.92% | 0.12 | 0.00 |
| Yes | 0.250 | 50 | 18 | 98.69% | 95.38% | 0.17 | 0.00 |
| Yes | 0.250 | 100 | 18 | 98.69% | 95.38% | 0.18 | 0.00 |
| Yes | 0.500 | 10 | 10 | 98.69% | 95.38% | 0.18 | 0.00 |
| Yes | 0.500 | 50 | 50 | 100.00% | 94.62% | 0.38 | 0.01 |
| Yes | 0.500 | 100 | 72 | 100.00% | 94.62% | 0.38 | 0.01 |
| No | — | 10 | 10 | 100.00% | 96.15% | 0.11 | 0.00 |
| No | — | 50 | 19 | 100.00% | 95.38% | 0.18 | 0.01 |
| No | — | 100 | 19 | 100.00% | 95.38% | 0.17 | 0.01 |

Table 4: Spambase performance with a 70%-30% Train-Test split using AdaBoost

| **Pruning** | **Pruning Confidence** | **Boost Iterations** | **Training %** | **Test %** | **Train Time (s)** | **Test Time (s)** |
|---|---|---|---|---|---|---|
| Yes | 0.125 | 10 | 100% | 94.42% | 5.03 | 0.04 |
| Yes | 0.125 | 50 | 100% | 95.94% | 22.90 | 0.16 |
| Yes | 0.125 | 100 | 100% | 96.38% | 45.20 | 0.38 |
| Yes | 0.250 | 10 | 100% | 94.93% | 5.54 | 0.04 |
| Yes | 0.250 | 50 | 100% | 95.94% | 24.61 | 0.28 |
| Yes | 0.250 | 100 | 100% | 95.80% | 47.05 | 0.40 |
| Yes | 0.500 | 10 | 100% | 95.00% | 5.00 | 0.03 |
| Yes | 0.500 | 50 | 100% | 96.16% | 23.22 | 0.18 |
| Yes | 0.500 | 100 | 100% | 96.30% | 45.45 | 0.38 |
| No | — | 10 | 100% | 95.00% | 3.71 | 0.03 |
| No | — | 50 | 100% | 96.23% | 25.20 | 0.19 |
| No | — | 100 | 100% | 96.16% | 38.91 | 0.48 |





It's immediately apparent that boosting is an eager learner as it takes more time than a decision tree to train but quickly evaluates the model at runtime. Training time does not scale as well with regard to number of instances as does decision trees. Still, boosting was a relatively fast machine learning technique overall. Runtime was nearly linear with regard to the train-test split percentage ($runtime \propto training\ data$, $k \approx 5$).

Interestingly, AdaBoost decided to run fewer iterations than passed as a parameter for the political party classification dataset. This is the result of early stopping, a technique used by AdaBoost to avoid overfitting the data. As boosting does not tend to overfit, early stopping is used to guarantee consistency. Pruning confidence appears to have an effect on how early AdaBoost stops - pruning hit the stopping threshold after 9 iterations with a confidence of 0.125 compared to 72 for a pruning threshold of 0.5. This indicates that the more the weak learner overfits, the more iterations are required to reach convergence.

Runtime is directly proportional with the number of boosting iterations. This makes intuitive sense. Performance of both algorithms both increased. For the political party classification dataset, performance only increased a small amount. For spambase, however, performance increased considerably (around 4%!). The higher number of attributes combined with AdaBoost's technique of widening the decision boundary allowed for the increased performance. Boosting was the best technique out of those tested for spambase.

Boosting did not show much evidence of overfitting - test performance increased nearly monotonically with regard to the number of iterations. The learning curve demonstrates that boosting can take advantage of more training data in order to decrease test performance. Boosting is able to fit the training data very well and it appears that the data does generalize well to the test set. As more training instances were used, training error increased marginally.

## 2.3 Support Vector Machines

Two different kernel functions were tested in Weka for SVM - a polynomial kernel and the RBF[3] kernel. These two kernels are used by SVM for the kernel trick in order to represent higher order functions.

Table 5: Voting classification performance with a 70%-30% Train-Test split using SVM

| **Kernel** | **Exponent** | **Gamma** | **Training %** | **Test %** | **Train Time (s)** | **Test Time (s)** |
|---|---|---|---|---|---|---|
| Polynomial | 1 | — | 95.74% | 96.15% | 0.40 | 0.02 |
| Polynomial | 2 | — | 99.34% | 96.15% | 0.47 | 0.02 |
| Polynomial | 3 | — | 99.34% | 94.62% | 0.57 | 0.10 |
| Polynomial | 5 | — | 99.34% | 96.15% | 0.56 | 0.03 |
| Polynomial | 10 | — | 92.79% | 88.46% | 0.18 | 0.12 |
| RBF | — | 0.001 | 89.51% | 90.77% | 0.25 | 0.04 |
| RBF | — | 0.010 | 94.75% | 93.85% | 0.16 | 0.02 |
| RBF | — | 0.100 | 95.74% | 96.15% | 0.17 | 0.03 |
| RBF | — | 1.000 | 98.69% | 94.62% | 0.33 | 0.08 |

[3]Radial Basis Function

Table 6: Spambase classification performance with a 70%-30% Train-Test split using SVM

| **Kernel** | **Exponent** | **Gamma** | **Training %** | **Test %** | **Train Time (s)** | **Test Time (s)** |
|---|---|---|---|---|---|---|
| Polynomial | 1 | — | 90.16% | 90.65% | 1.23 | 0.12 |
| Polynomial | 2 | — | 82.74% | 81.96% | 29.06 | 2.30 |
| Polynomial | 3 | — | 72.03% | 71.45% | 41.01 | 2.13 |
| Polynomial | 5 | — | 63.55% | 62.83% | 24.89 | 2.48 |
| Polynomial | 10 | — | 62.12% | 61.52% | 22.51 | 2.76 |
| RBF | — | 0.001 | 60.48% | 60.87% | 77.86 | 1.64 |
| RBF | — | 0.010 | 72.59% | 73.04% | 53.75 | 1.76 |
| RBF | — | 0.100 | 86.96% | 86.67% | 33.56 | 1.68 |
| RBF | — | 1.000 | 92.52% | 92.90% | 11.67 | 1.18 |

It is immediately apparent that the SVM may be overfitting the training data. In the Political Party classification experiment, test performance drops as the polynomial kernel goes from an exponent of 5 to an exponent of 10. For the RBF kernel, test performance decreases at a gamma value of $\gamma = 1.00$. For the spambase classification experiment, test performance drops consistently as exponent increases for the polynomial. Interestingly, increasing the gamma value for the RBF kernel increases both train and test performance. This indicates that as an SVM gains too much expressive power (for the polynomial kernel at an exponent of 10), it overfits the training data by producing a very complex decision boundary.

SVMs appear to be slower than other techniques for larger datasets, excluding neural networks. This is interesting as SVMs should consider instances at the edge of the margin and therefore runtime should be more-or-less invariant to a larger number of instances. Increased runtime may be due to the more complex decision boundary for spam, thereby requiring more instances to train the SVM.

SVMs are an eager learning technique - this is especially evident in the spambase experiment where train times where much longer than test times.

## 2.4 *k*-nearest neighbors

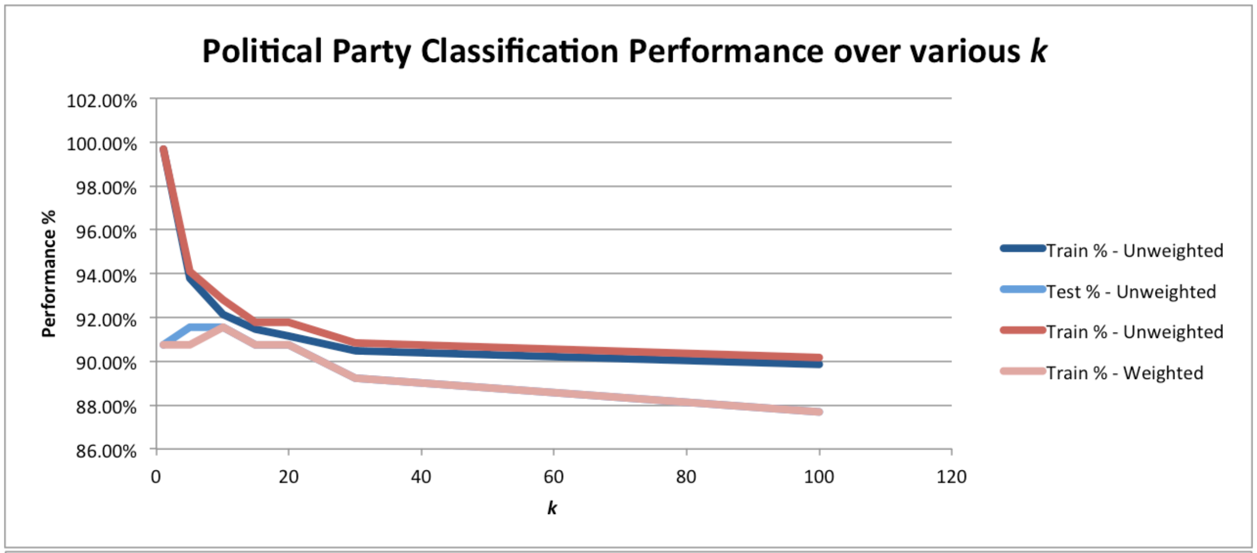
$k$-NN was tested using a variety of nearest neighbor counts. Neighbors were weighted by $1 - D_i$ where $D_i$ is the distance to the neighbor.

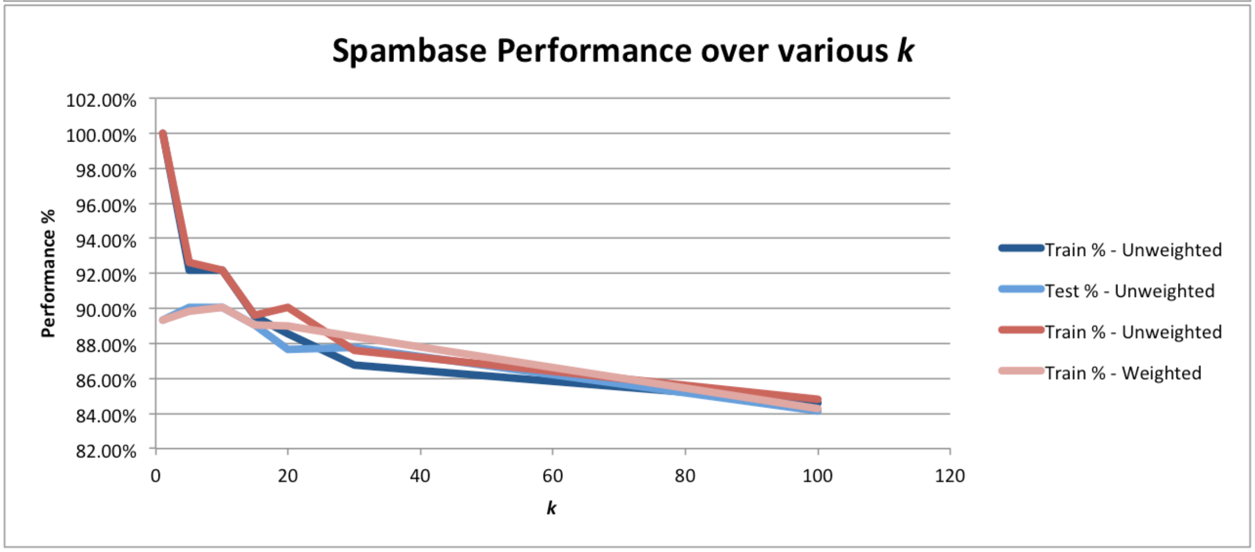
Table 7: Political classification performance with a 70%-30% Train-Test split using $k$-NN

| **Weighted?** | **# Nearest Neighbors** | **Train %** | **Test %** | **Train Time (s)** | **Test Time (s)** |
|---|---|---|---|---|---|
| No | 1 | 99.67% | 90.77% | 0.02 | 0.31 |
| No | 5 | 93.77% | 91.54% | 0.10 | 0.21 |
| No | 10 | 92.13% | 91.54% | 0.02 | 0.35 |
| No | 15 | 91.48% | 90.77% | 0.06 | 0.37 |
| No | 20 | 91.15% | 90.77% | 0.04 | 0.40 |
| No | 30 | 90.49% | 89.23% | 0.19 | 0.35 |
| No | 100 | 89.84% | 87.69% | 0.07 | 0.36 |
| Yes | 1 | 99.67% | 90.77% | 0.02 | 0.37 |
| Yes | 5 | 94.10% | 90.77% | 0.14 | 0.37 |
| Yes | 10 | 92.79% | 91.54% | 0.01 | 0.38 |
| Yes | 15 | 91.80% | 90.77% | 0.04 | 0.32 |
| Yes | 20 | 91.80% | 90.77% | 0.01 | 0.43 |
| Yes | 30 | 90.82% | 89.23% | 0.10 | 0.28 |
| Yes | 100 | 90.16% | 87.69% | 0.06 | 0.41 |

Table 8: Spambase classification performance with a 70%-30% Train-Test split using $k$-NN

| **Weighted?** | **# Nearest Neighbors** | **Train %** | **Test %** | **Train Time (s)** | **Test Time (s)** |
|---|---|---|---|---|---|
| No | 1 | 100.00% | 89.35% | 0.04 | 1.77 |
| No | 5 | 92.18% | 90.07% | 0.04 | 3.33 |
| No | 10 | 92.18% | 90.07% | 0.04 | 3.33 |
| No | 15 | 89.57% | 89.06% | 0.37 | 11.60 |
| No | 20 | 88.58% | 87.68% | 0.37 | 11.90 |
| No | 30 | 86.74% | 87.75% | 0.36 | 12.46 |
| No | 100 | 84.60% | 84.13% | 0.30 | 13.88 |
| Yes | 1 | 100.00% | 89.35% | 0.26 | 7.21 |
| Yes | 5 | 92.61% | 89.86% | 0.11 | 8.71 |
| Yes | 10 | 92.18% | 90.07% | 0.22 | 9.94 |
| Yes | 15 | 89.60% | 89.06% | 0.07 | 10.15 |
| Yes | 20 | 90.07% | 88.99% | 0.17 | 10.53 |
| Yes | 30 | 87.58% | 88.41% | 0.16 | 12.66 |
| Yes | 100 | 84.79% | 84.28% | 0.18 | 13.21 |





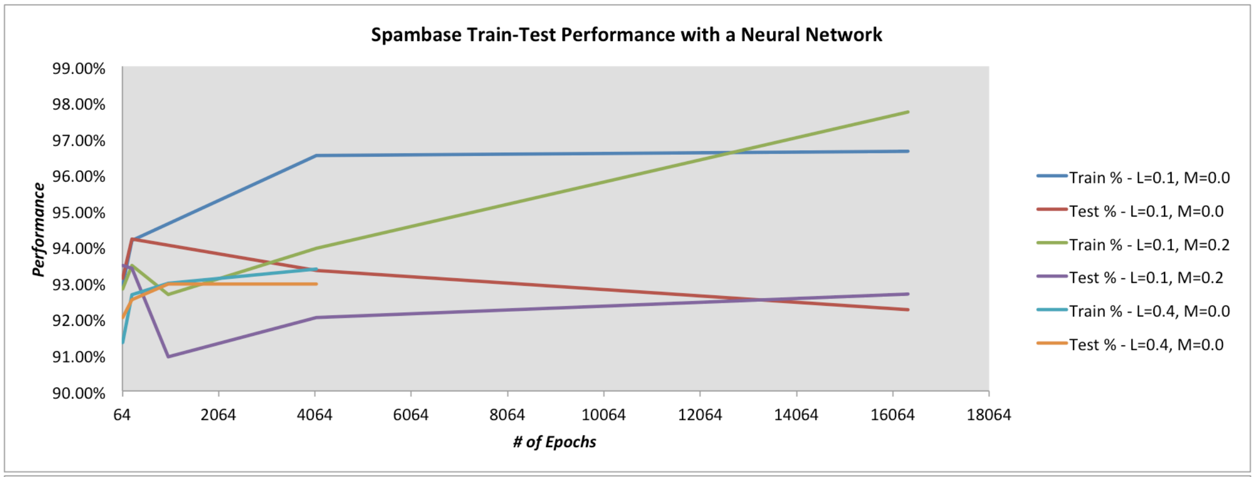
It is immediately apparent that $k$-Nearest Neighbor is a lazy learning technique as test runtime is around an order of magnitude longer than train runtime. The algorithm is not particularly scalable to large datasets - test runtime for the spam dataset is quite a bit slower than for the voting dataset (however, train time is not significantly impacted). It did not appear that weighting impacted runtime significantly.

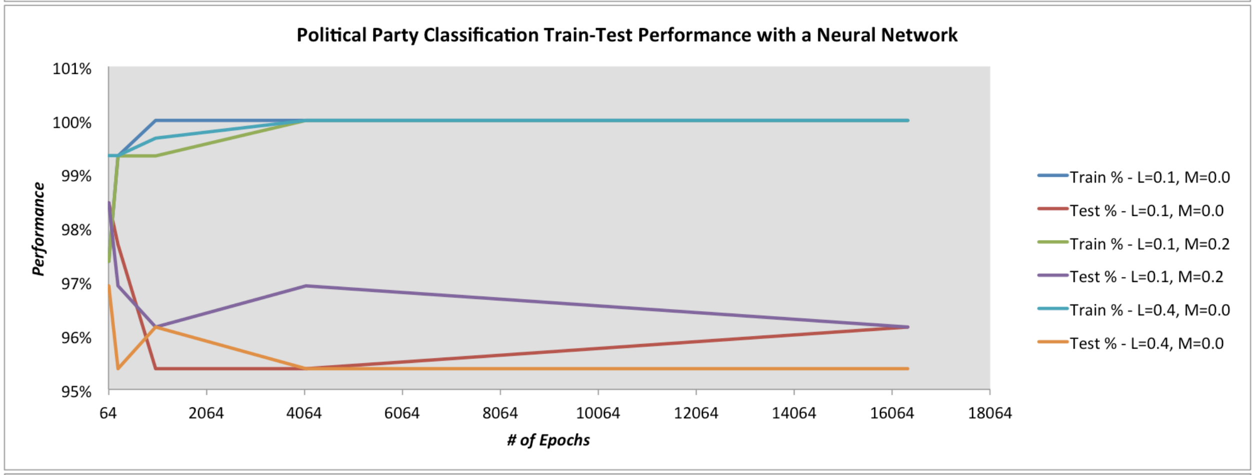
For both datasets, $k$-NN did not produce fantastic results but it did still reach a reasonable test performance. Performance was quite sensitive to parameter tuning, primarily the number of neighbors $k$. It appeared that for both datasets, $k = [5, 10]$ contained the optimal $k$ value for the data. Weighting made the results less sensitive to tuning the value of the parameter $k$. Intuitively, this makes sense - if $k$ is too small, performance will be very sensitive to noise and outliers. If $k$ is too large, then the model loses the ability to model local behavior of the underlying function and may begin mixing votes from multiple classes. Also, it is intuitive that weighting would make the model less sensitive to a particular choice of $k$ as farther train instances are given less weight by $k$-NN.

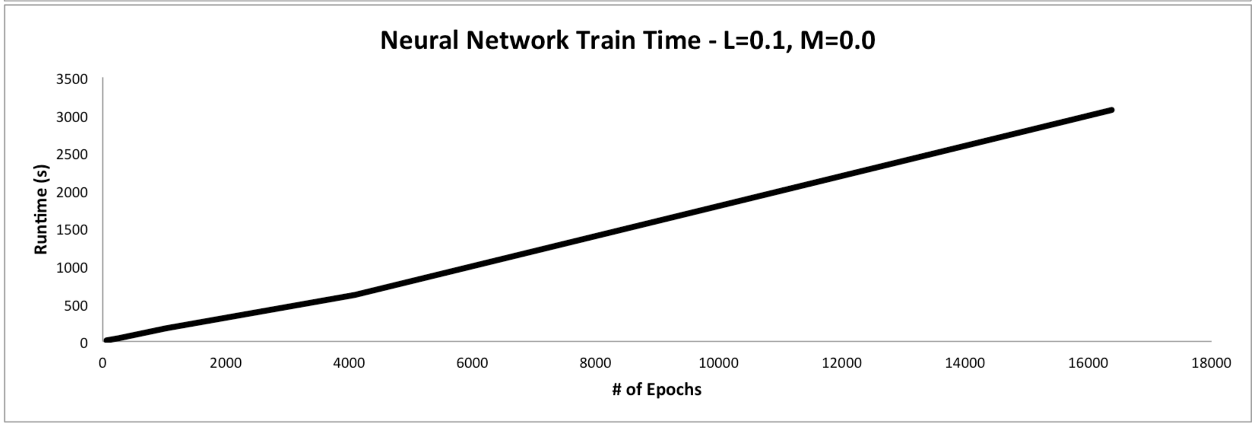
I suspect that $k$-NN performed poorly on the spambase dataset as there are a number of attributes that aren't very important. This causes two instances that should be close to each other to potentially end up distant due to an extra attribute.

## 2.5 Neural networks

Total runtime for this neural network task was 3 hours. GNU Parallel was used to speed up computation. This experiment varies learning rate and momentum to examine impact on performance.







Neural networks were the slowest technique to train at a high number of epochs. Runtime for neural networks are directly proportional to the number of epochs. In the runtime versus number of epochs run, the relationship is almost nearly linear ($runtime \propto epochs$, $k \approx 0.18$).

Neural networks are extremely sensitive to initial parameters for learning rate and momentum - for political party classification, performance varied by almost a full percent between a learning rate of 0.1 and 0.4. Training a neural networks is an optimization problem over a non-convex function so it makes sense that too high of a learning rate would fail to converge at a minimum.

The performance graph for political party classification clearly shows train error decreasing over more training epochs while test error decreases to convergence at about 4000 epochs. This suggests that neural networks are overfitting the simple political party dataset.

For spambase, train error decreases over more training epochs while test error performs differently for various learning rates. With a learning rate $L = 0.1$ and momentum $M = 0.2$, test error continued to decrease up until around 16,000 epochs - if more training epochs were used, test error should continue to decrease. However, with a learning rate $L = 0.1$ and momentum $M = 0.0$, test error increased after hitting a minimum at around 200 epochs. In this case, it appears that the neural network is overfitting the data. Larger learning rates converge more quickly but as both the voting dataset and spambase dataset show, can have suboptimal performance.

It appears that neural networks require more training data than other methods - neural nets overfit the political party classification dataset much more quickly than the spambase dataset. It appears that neural networks perform well using real valued attributes as those found in the spambase dataset. However, the large number of extraneous attributes made training require a large number of attributes to converge.

Neural networks are very clearly a form of an eager learner - significant time is spent at train time to produce a model which is very quick to evaluate at test time. Due to the large amount of time taken to train neural networks, I wasn't able to study more granular parameter tuning.