Supervised Learning - Assignment 1

Xiaobing Li

Sunday, Feb 1, 2015

Abstract

In the first part of this project, I used a small set of the data from the MINST dataset and applied five learning algorithms to recognize hand-written digits in the dataset. In the second part of the project, the spam dataset was used. The algorithms were adopted from different R packages. The error rates, training and test times of different learning algorithms were studied and compared.

Introduction

The purpose of this project is to explore some techniques in supervised learning. Five learning algorithms were tested in this project. They were: 1.Decision trees with some form of pruning, 2. Neural networks, 3. Boosting, 4. Support Vector Machines, 5. k-nearest neighbors.

I chose two datasets for this project. The first one was the MNIST dataset, which included thousands of hand-written digits. I tried to use machine learning algorithms to recognize these digits. Image recognition is an interesting topic in the real world. Digit recognition will be a good, simply start for me to explore the feasibility and efficiency of different learning algorithms. Moreover, this dataset is studied extensively. A lot of examples/papers can be found online, which would be very helpful.

The second dataset I used was the UCI spam dataset. People use it to develop algorithms to classify spam or non-spam emails, so this is also a good start for me to study maching learning algorithms. I use Google mail, yahoo mail and Microsoft hotmail for a long time. They all have their own spam filters. Google's spam filter is the best of them, but it still makes a lot of mistakes. I feel excited if I can figure out how a spam filter works and design my own models. Similar strategy also can be used for classifying news, hot topics, etc. So it worths to learn it.

Part I Digit Recognizer

Data Source The MNIST("Modified National Institute of Standards and Technology") dataset is used in this part. I downloaded the data files from Kaggle competition- Digit Recognizer at https://www.kaggle.com/c/digit-recognizer/data. The goal in this competition is to take an image of a handwritten single digit, and determine what that digit is.

The MNIST dataset is classic within the Machine Learning community that has been extensively studied. The original dataset could be found on http://yann.lecun.com/exdb/mnist/index.html, having a training set of 60,000 images and a test set of 10,000 images. Each image has 28X28 pixels with gray levels from 0-254.

The data downloaded from Kaggle has been converted into .csv files with each row representing one image. Since the test data on Kaggle have no label provided, I use only the data in train.csv. For saving the running time, I took only 1000 images as a first try. By splitting this 1000-image dataset into a train subset with 75% images and a test subset with 25% images, I can train my models and do cross-validations.

Data Preparation

Generally, for object recognition, we have to compute feature vectors from the images. The simplest way to contruct a feature vector is to use raw pixels. Obviously raw pixel is not a good feature vector considering the distorations of hand-written digits, but I decided to use them anyway to simplify the questions.

Because the image data have been saved in CSV format, I simply loaded the train.csv into R as a data frame. The train dataset included 42000 observations and 785 variables. The first variable is 'label', which tells the actual digit number. The next 784 variables are the gray levels of each pixel in the images. The original images are two-dimensional and have been flattened into a vector.

To make our life easier, I only took the first 1000 images (1000 rows) for my study. This subset included all 10 digits, in relatively balanced frequencies (Table 1).

```
##
##   0   1   2   3   4   5   6   7   8   9
## 107  96 124  90 102  89  97 105  93  97
```

Table 1. *The numbers of each digit in this subset of the data.*

To visualize the image data, I showed first 20 images in the dataset with their labels. The images have been normalized and well-centered.

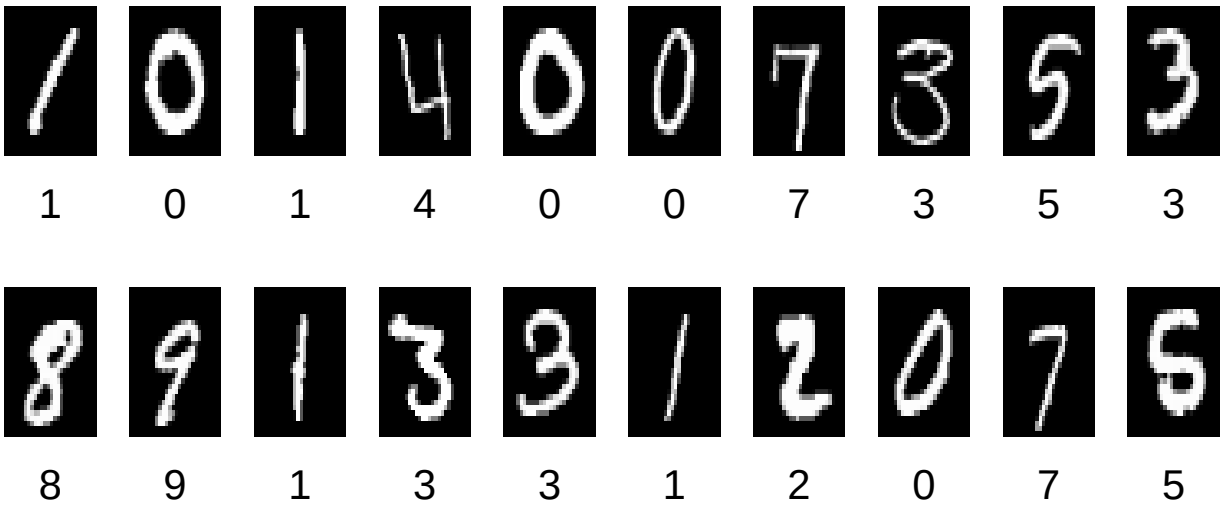

Figure 1. *First 20 images in the dataset. Each image has 28X28 pixels with 255 gray levels. They were centered and their sizes were adjusted.*

Data Splitting The dataset was randomly splitted into a train set and a test set. The train set included about 75% of the images and the test set included about 25% of the images. For studying the algorithm performance as a function of training size, I created another 4 datasets from the train set. They included 20%, 40%, 60% and 80% of the images in the train set, respectively. By plotting the algorithm performance to these five data sets (four plus the whole train set), I can get a general idea that how the size of a train set influences the performance of a learning algorithm.

Data training and testing

Now the dataset is prepared, so I start to train and test using learning algorithms. For decision tree, I used the rpart packages; for neural networks, I used the nnet package; for boosting, I used the adabag package; for SVM, I used the kernlab package and for kNN, I used the class package. Although the caret package wrapped in all these algorithms, for some reasons, the training process using the caret package is much longer than using individual packages, so I decided to use individual packages.

Decision trees with pruning The name 'rpart' means Recursive Partitioning and Regression Trees. When the method parameter is set to 'class', it can work on classification problems. rpart fitting uses a set of parameters to control the growth of the decision tree. To demonstrate the pruning process, I set complexity parameter(cp) to 0.0 to achieve a complete tree. Even though, the method itself has other default settings to limit the tree growth, such as minimum number of observations in a node, maximum depth of any node, etc. It also does internal cross-validations. So, setting cp to 0.0 may or may not give an overfitting, but I would like to try and use my test dataset to prune it.

After training, I first got a tree with 83 splits, which was hard to be shown with labels, so I simply showed it with bare branches (Figure 2a).

By checking the cptable, I found the tree achieved its minimum xerror (which is an error measurement from internal cross-validations) at around size 69. Actually, the xerror has very limited decrease after size 45 (Figure 2b).

Then I started to prune this tree. I calculated error rates of predictions based on the test data using tree sizes from 32 to 83. It turned out that the tree with size 69 gave the best error rate on my test data (Figure 2c).

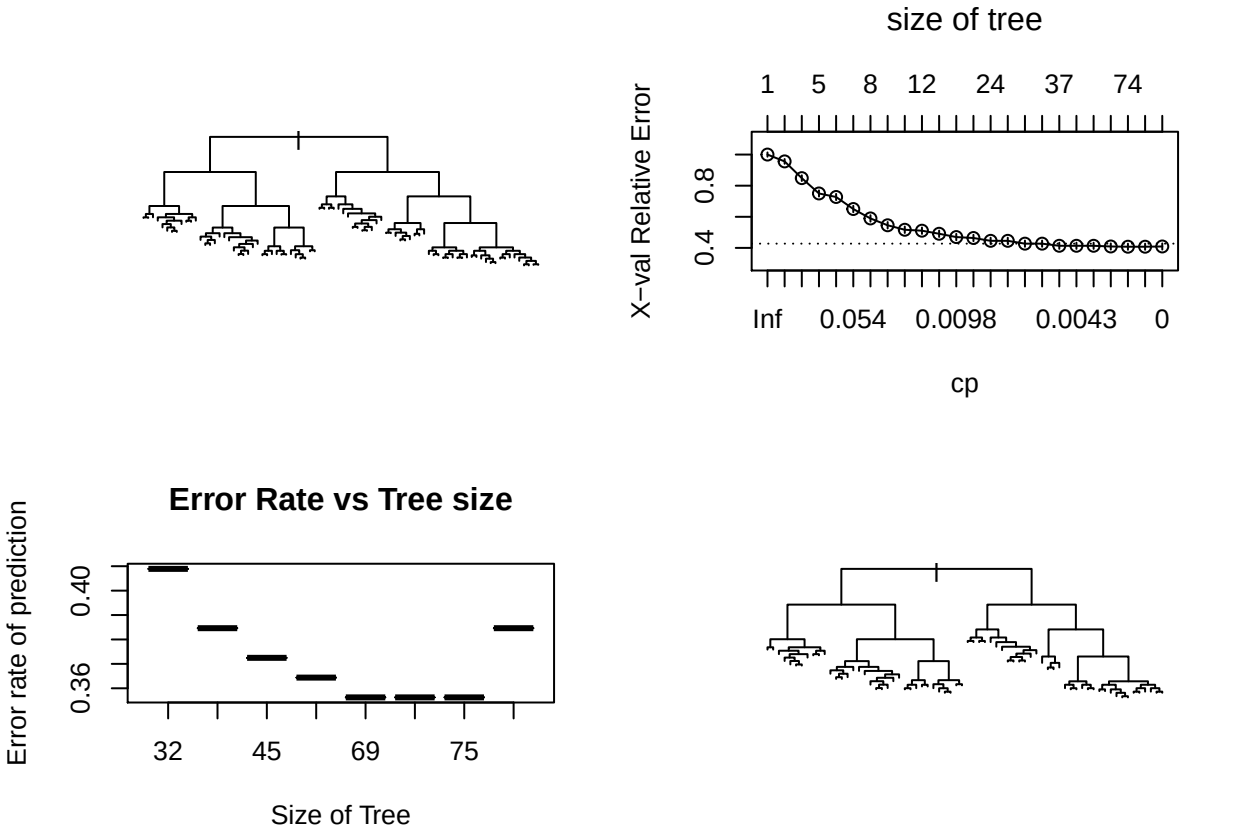

Figure 2 *Decision tree.* ***a*** *tree before pruning.* ***b*** *Xerror vs CP.* ***c*** *The tree size is plotted to the error rate of prediction.* ***d*** *tree after pruning.*

The optimized pruned tree is shown in Figure 2d. The tree is only slightly different with the first one. The best error rate is 35.6%. It took 2.88s for the training and 0.11s for testing.

Next, I trained the model on different sizes of training datasets from train1 to train4. I found while the training size is larger, the predicton accuracy is better. The result is plotted on Figure 3.

Figure 3 *Decision tree. The performance is better with increased training size.*

Neural networks The package nnet provides methods for using feed-forward neuronal networks with a single hidden layer. In this training process, the number of units in the hidden layer is set to 15, the maximum iteration is set to 50, 100, 150 and the weight decay is set to 0.1. This algorithm is very time-consuming so I cannot try too many different parameters. Another option is to use the train function in the caret package. It will automatically tune some parameters.

The training is slow and when the maximum iteration is longer, the error rate is lower. This result is presented in Figure 4.

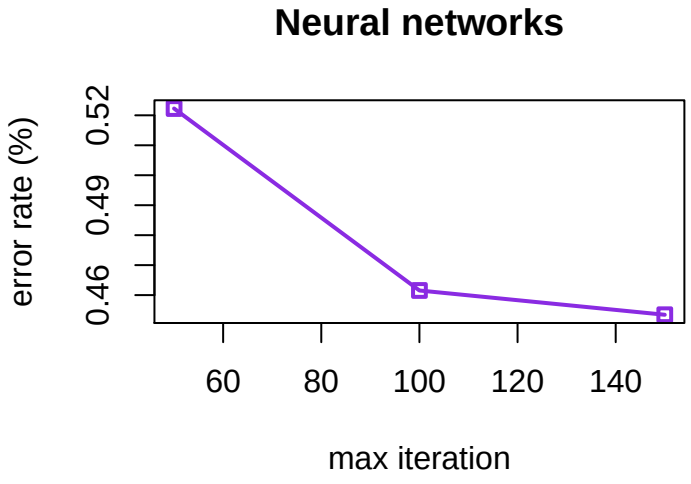

Figure 4 *The error rate from Neuronal networks decreases with the maximum iterations.*

Boosting For boosting, I chose Freund and Schapires's Adaboost.M1 algorithm. The weak classifiler used in this algorithm is CART (rpart). I used the same parameters as I used in the decision tree above and used 10 trees in the algorithm. For pruning purpose, I chose a series of cp (complexity parameter) values then

compared the model error rates. The relationship between the tree size and the error rate is plotted in Figure 5a.

I then chose the best parameters and tested the model with different sizes of train data. Not surprisely, with a bigger training size, the performance of the model is getting better (Figure 5b).

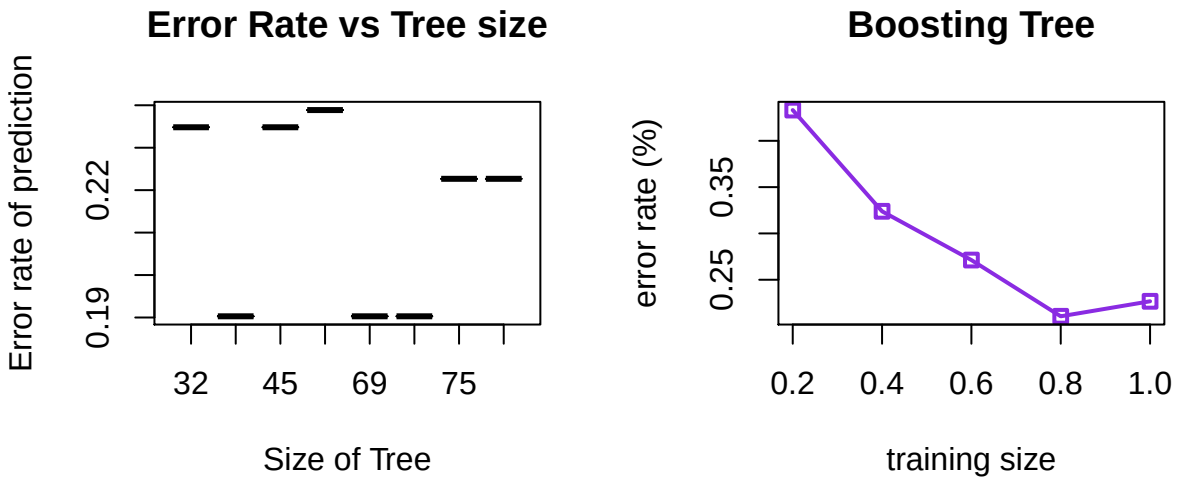

Figure 5 a *The relationship between the tree size and the error rate.* ***b*** *The error rate of model decreases with the training data size.*

Support Vector Machines For Support Vector Machines, I used the kernlab package and tested two kernals. One was a linear kernal and another one was a Gaussian kernal. For both kernals, I measured the error rate of prediction as a function of training size. The results are plotted on Figure 6. For this dataset, the Gaussian kernal was slightly better than the linear kernal when the data size was small. However, with bigger data sizes, the performances of two kernals were close. It may be data-dependant.

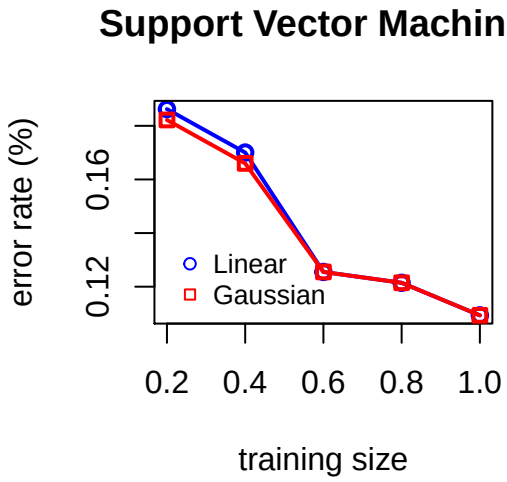

Figure 6 *The error rates are plotted to the training sizes in two different kernals.*

k-nearest neighbors The knn algorithm in the class package is a single k nearest neighbour classification method using Euclidean distance. The knn.cv method in the same package uses leave-one-out cross validation. I tested knn in different k numbers and used the best model on different sizes of datasets. I also compared the knn.cv method to the knn method. The results were presented in Figure 7.

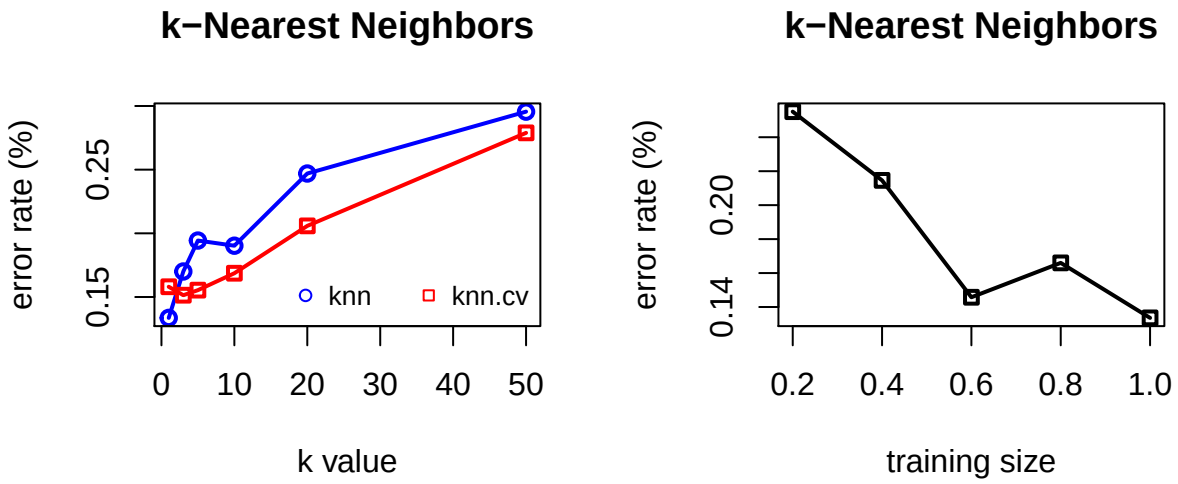

Figure 7 *k-nearest neighbors.* ***a*** *the error rates with different k values in two kernals.* ***b*** *the error rates with different training sizes.*

Results and Comparisons

Until now, I have studied five learning algorithms elaborately on the digit dataset. Here I will compare these five algorithms in terms of training time, testing time and error rate.

I used the optimized parameters tested above to measure the training time, testing time and error rate in different learning algorithms. These results are shown in Figure 8.

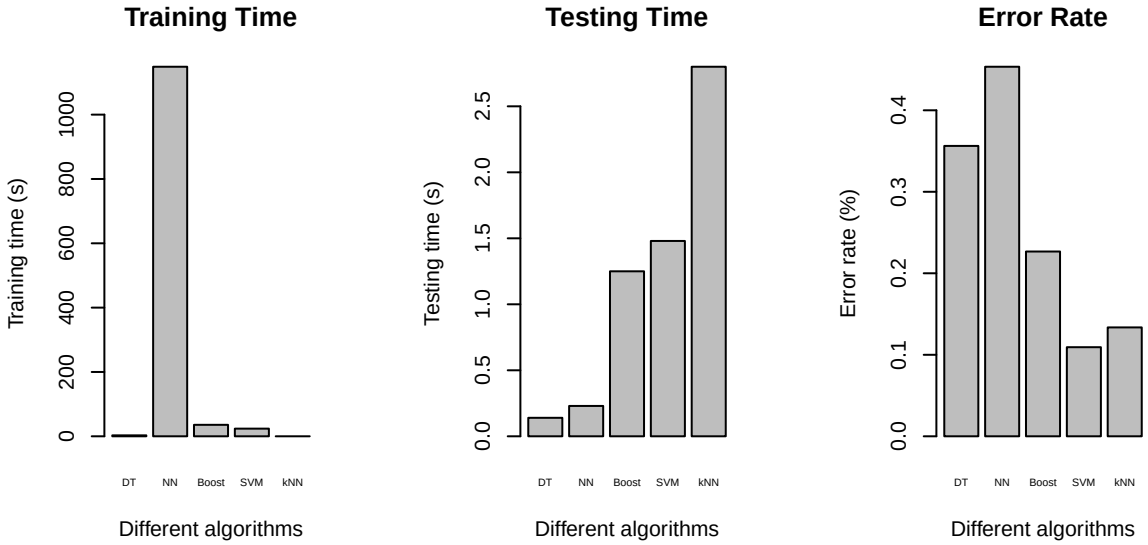

Figure 8 *Comparisons across different learning algorithms.*

From the figure, I can conclude :
1. the neural network takes much longer time to train (1148.97s) but has the worst performance(45.3% error rate).
2. the decision tree without boosting is easy to train and test, but the performance is not optimal (35.6% error rate).
3. SVM model gives the best performance in terms of the error rate (10.9%) and the training time is short (23.79s). It takes slightly longer testing time (1.48s) on the test set than other models except kNN.
4. Because kNN is a lazy model, it need not to be trained. Overall kNN spends the least time to solve the problem (training + testing time =2.8s) to achieve a quite good performance (13.3 error rate).
5. the Boosting model has better performance than the pruned decision tree (22.7% error rate vs 35.6%).

Because I don't have enough time to tune the models thoroughly, these comparisons may not be fair enough. For example, if I run the neural network model for more iterations, it may give better performance. However, this study gives me a general idea how well these algorithms work on the MNIST dataset.

Part II Spam or no-Spam classification

Data Source The UCI Spambase data set is used in this part. The R default 'datasets' package inlcuded this dataset. The original dataset can be found at https://archive.ics.uci.edu/ml/datasets/Spambase. It has 4601 instances and 57 attributes. This dataset also has been extensively studied. I will use the same strategy to study this dataset as I did to the MNIST dataset.

Data Preparation

I can simply load the dataset by using data(spam) command in R studio. At first, I checked the loaded dataset. This dataset included 4601 observations and 58 variables. The variable 1-54 indicate whether a particular word or character frequenctly occurred in the email. The variable 55-57 measure the length of sequences of consecutive capital letters. The last variable is a tag, which marks the email as spam (1) or no spam (0).

Data Splitting The dataset was randomly splitted into a train set and a test set. The train set included about 75% of the images and the test set included about 25% of the images. For studying the algorithm performance as a function of training size, I created another 4 datasets from the train set. They included 20%, 40%, 60% and 80% of the images in the train set, respectively. By plotting the algorithm performance to these five data sets (four plus the whole train set), I can get a general idea that how the size of a train set influences the performance of a learning algorithm.

Data training and testing

I used the same packages for this dataset. Because this dataset has less variables and only two classes, I expect the training process will be much shorter.

Decision trees with pruning To demonstrate the pruning process, I set complexity parameter(cp) to 0.0 to achieve a complete tree and use my test dataset to prune it.

After training, I first got a tree with 114 splits, which was hard to be shown with labels, so I simply showed it with bare branches (Figure 1a).

By checking the cptable, I found the tree achieved its minimum xerror (which is an error measurement from internal cross-validations) at around size 89. Actually, the xerror has very limited decrease after size 48 (Figure 1b).

Then I started to prune this tree. I calculated error rates of predictions based on the test data using tree sizes from 34 to 114. It turned out that the tree with size 73 gave the best error rate on my test data (Figure 1c).

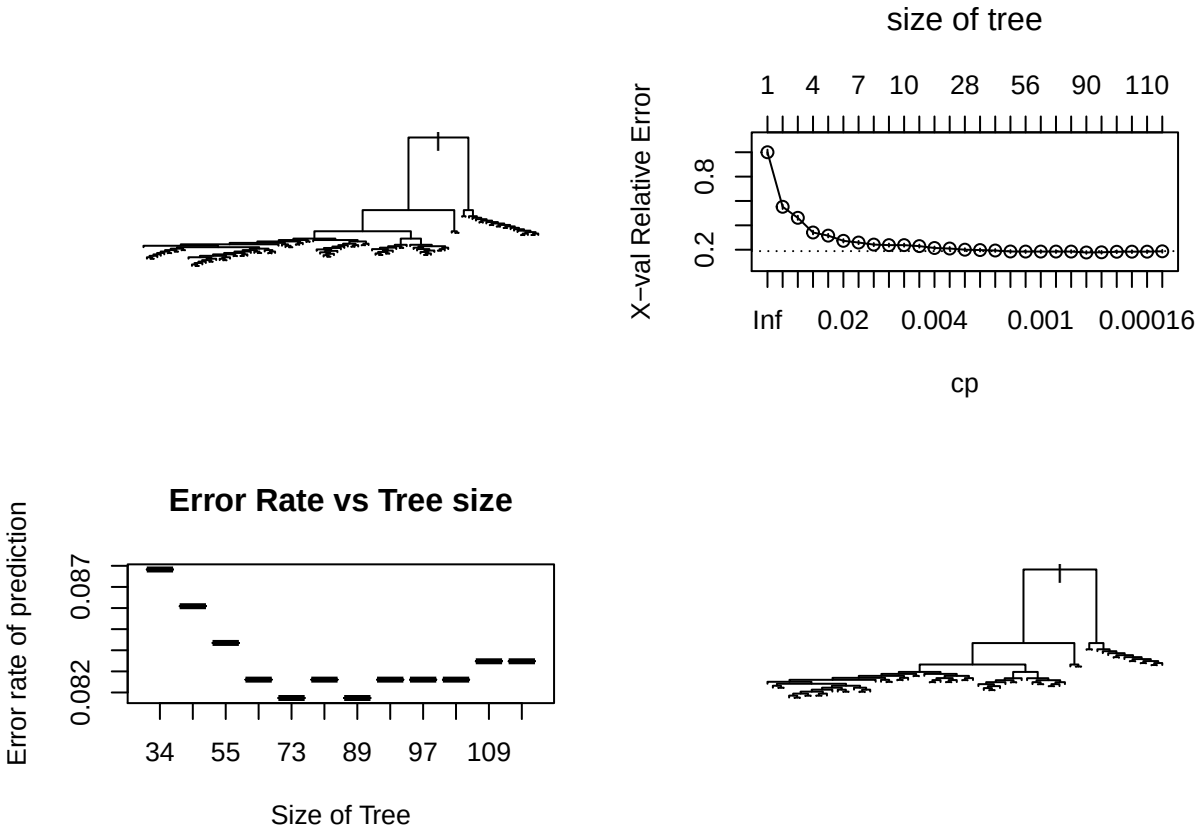

Figure 1 *Decision tree.* ***a*** *tree before pruning.* ***b*** *Xerror vs CP.* ***c*** *The tree size is plotted to the error rate of prediction.* ***d*** *tree after pruning.*

The optimized pruned tree is shown in Figure 1d. The tree is only slightly different with the first one. The best error rate is 8.17%. It took 1.37s for the training and 0.01s for testing.

Next, I trained the model on different sizes of training datasets from train1 to train4. I found while the training size is larger, the predicton accuracy is better. The result is plotted on Figure 2.

Figure 2 *Decision tree. The performance is better with increased training size.*

Neural networks This dataset only has two classes, so I set the number of units in the hidden layer to 10, the maximum iteration is set to 100, 200, 300, 400, 500 and the weight decay is set to 0.1. This process is much faster than in the MNIST dataset.

At this time, increasing the maximum iteration didn't always lower the error rate. 200 iterations gave the best error rate at 5.65%. The model did converge at around 600 iterations, but that didn't make the lowest error rate. This result is presented in Figure 3.

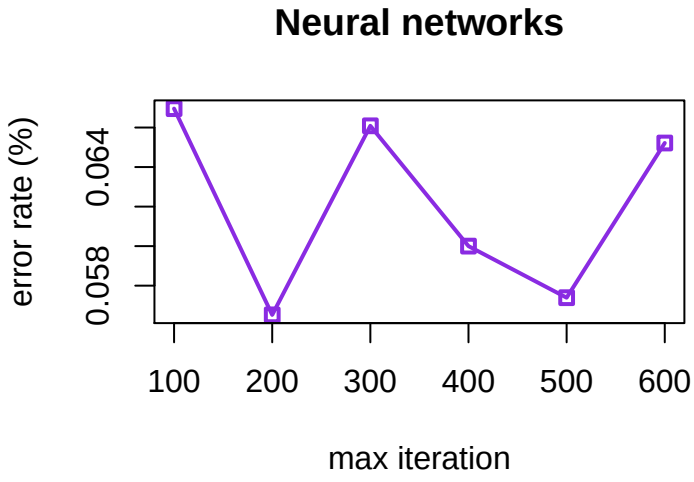

Figure 3 *The error rate from Neuronal networks changing with the maximum iterations.*

Boosting I used the same parameters as I used in the decision tree above and used 10 trees in the algorithm. For pruning purpose, I chose a series of cp (complexity parameter) values then compared the model error rates. The relationship between the tree size and the error rate is plotted in Figure 4a.

At this time, the best error rate is achieved at the tree size 109. I used this pruned tree to test different sizes of train data. With a bigger training size, the performance of the model is getting better (Figure 4b).

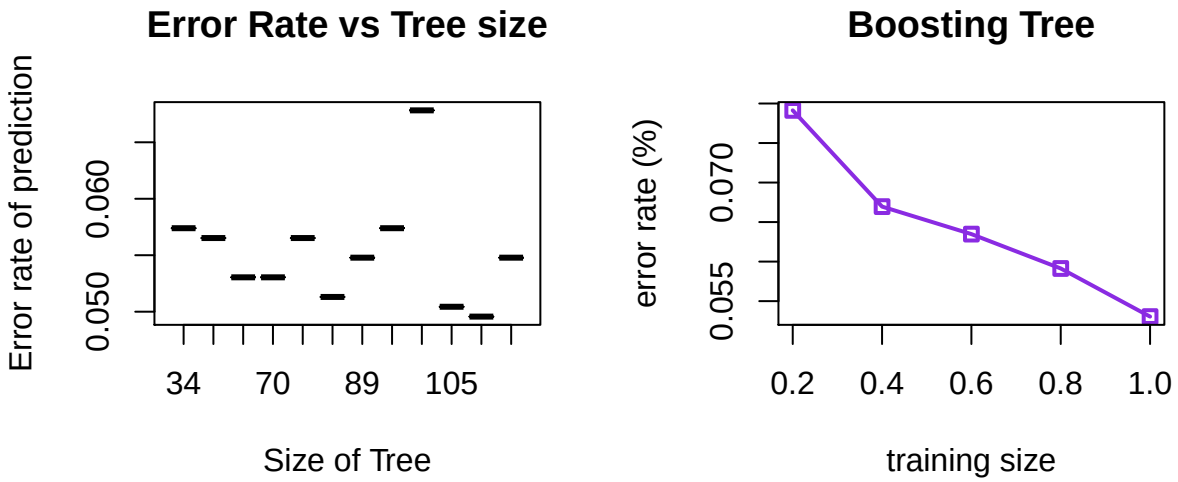

Figure 4 ***a*** *The relationship between the tree size and the error rate.* ***b*** *The error rate of model decreases with the training data size.*

Support Vector Machines For Support Vector Machines, I used the kernlab package and tested two kernals. One was a linear kernal and another one was a Gaussian kernal. For both kernals, I measured the error rate of prediction as a function of training size. The results are plotted on Figure 5. For this dataset, the Gaussian kernal was slightly better than the linear kernal but not always.

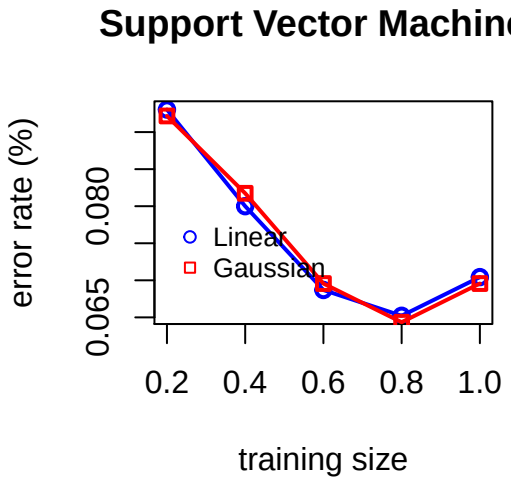

Figure 5 *The error rates are plotted to the training sizes in two different kernals.*

k-nearest neighbors I tested knn in different k numbers and used the best model on different sizes of datasets. I also compared the knn.cv method to the knn method. The results were presented in Figure 6. Still when k=1, the models gave the best error rates.

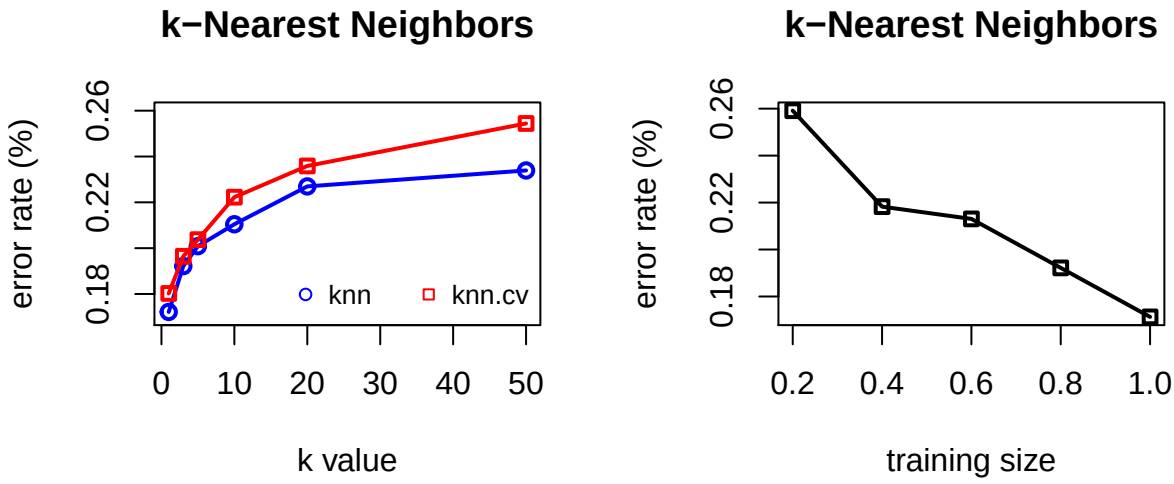

Figure 6 *k-nearest neighbors.* ***a*** *the error rates with different k values in two kernals.* ***b*** *the error rates with different training sizes.*

Results and Comparisons

I have studied five learning algorithms elaborately on the spam dataset. Here I will compare these five algorithms in terms of training time, testing time and error rate.

I used the optimized parameters tested above to measure the training time, testing time and error rate in different learning algorithms. These results are shown in Figure 7.

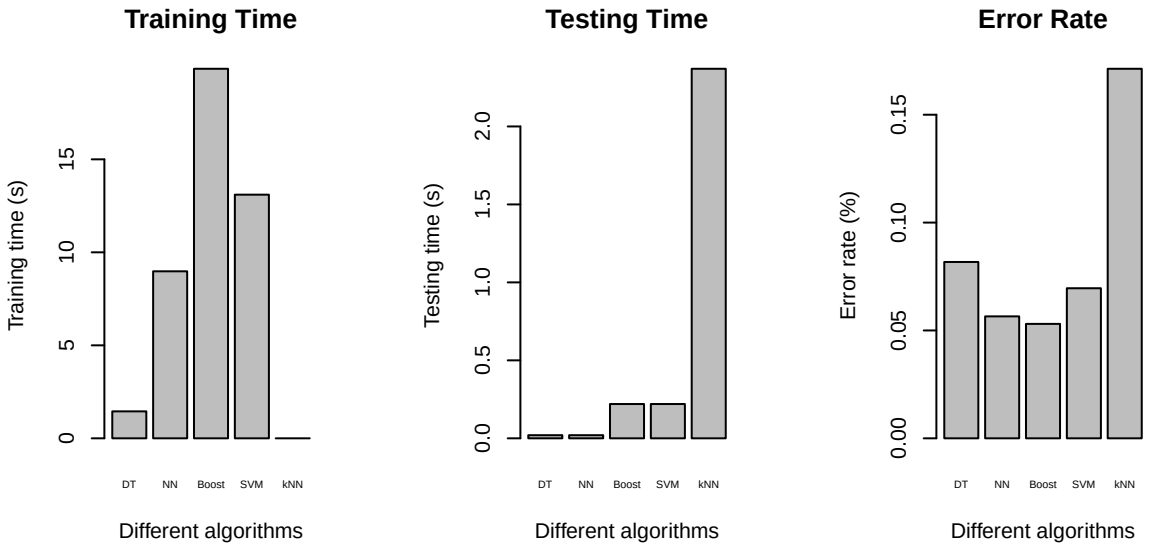

Figure 7 *Comparisons across different learning algorithms.*

From the figure, I conclude :
1. the boosting tree takes a long time to train (19.83s) but has the best performance(5.3% error rate).
2. the decision tree without boosting is fast to train and test, but the performance is so-so (8.17% error rate).
3. the training on SVM model takes the second long time with a moderate performance in terms of the error rate (6.96%).

4. Because kNN is a lazy model, it need not to be trained. Overall kNN spends a short time to solve the problem (training + testing time =2.37s), but its performance is the worst(17.1% error rate).
5. the neural network model surprises me this time. The training time is as short as 8.98s and the performace is the second good (5.65%).

Overall, the neural network did a good job for this task. It indicates that when the classes in the dataset are limited, a neural network can work pretty fast and accurate. For the digit recognizer task, since it has too many variable, a neural network model can be very slow. In that case, a SVM model may be a good choice.

References

1. rpart package: Author(s): Terry Therneau [aut], Beth Atkinson [aut], Brian Ripley [aut, trl, cre]

2. nnet package: Author(s): Brian Ripley ripley@stats.ox.ac.uk.URL: http://www.stats.ox.ac.uk/pub/MASS4/

3. adabag package: Author(s): Alfaro, Esteban; Gamez, Matias and Garcia, Noelia

4. kernlab package: Author(s): Alexandros Karatzoglou, Alex Smola, Kurt Hornik

5. class package: Author(s): Brian Ripley [aut, cre, cph], William Venables [cph]