# Hand Written Digit Recognition and Its Improvement

Bi Ran(U087272L)

February 26, 2010

**Abstract**

Handwritten digits recognition is classic problem of machine learning. The objective is to recognize images of single handwritten digits(0-9). In this paper, we apply C4.5 decision tree algorithm, boosting algorithm on C4.5, support vector machine(SVM) algorithm on this problem. The later part of the paper gives a way to improvement the classification accuracy by adding a few extra attributes into data set.

## 1 Introduction

Handwritten digit recognition recognition is a classification problem. The performance of classification algorithm heavily depends on the feature of data set. In this paper, Semeion Handwritten Digit Data Set is used for training and testing. The first part of the paper will show the result of different algorithms. It will be shown that the performance of decision tree is poor by itself, but much better after being boosted; SVM gives the best performance in these algorithms. The second part will try to improve performance of classifiers by adding extra features into data set. It will be shown that performance of all three algorithms has significant improvement on the modified data set, and SVM with 2 degree polynomial kernel achieves error rate 3.7037%, which is the lowest of all three algorithms.

## 2 Background and Motivation

Handwritten digit recognition has important usage such as recognizing zip code, online recognition on computer tablets. Algorithms like support vector machine and multi layer perceptron are popular in this field. Currently, Lenet and its boosted version achieves error rate less than 1% on MNIST data set. SVM also gives one of the lowest error rate among many classification algorithms.

## 3 Data set

Semeion Handwritten Digit Data Set is used in this paper. The data set was created by Tactile Srl, Brescia, Italy (`http://www.tattile.it`) and donated in 1994 to Semeion Research Center of Sciences of Communication, Rome, Italy (`http://www.semeion.it`), for machine learning research. 1593 handwritten digits from around 80 persons were scanned, stretched in a rectangular box 16x16 in a gray scale of 256 values.Then each pixel of each image was scaled into a boolean (1/0) value using a fixed threshold. This data set contains no missing values. Each person wrote on a paper all the digits from 0 to 9, twice. The commitment was to write the digit the first time in the normal way (trying to write each digit accurately) and the second time in a fast way (with no accuracy).

This data set consists of 1593 records (rows) and 256 attributes (columns). Each record represents a handwritten digit, originally scanned with a resolution of 256 grays scale. Each pixel of the each original scanned image was first stretched, and after scaled between 0 and 1 (setting to 0 every pixel whose value was under the value 127 of the grey scale (127 included) and setting to 1 each pixel whose original value in the grey scale was over 127). Finally, each binary image was scaled again into a 16x16 square box (the final 256 binary attributes).

The order of the examples in the data set is shuffled before training and testing because the original order of the data set is not really random.

## 4 Evaluation

The data set contains 1593 instances, which not very big. In this paper, cross validation is used to perform testing.

By applying cross validation on different fold number using C4.5 algorithm, we can see that the error rate almost keeps still after 4 fold. 4 fold cross validation is very likely to provide accurate estimation of the real error rate on unseen data. To be safe, 10 fold cross validation is used in this paper in all cases of testing.

| Fold Number | Error Rate |
|---|---|
| 2 | 29.0019% |
| 4 | 25.6121% |
| 6 | 24.231 % |
| 8 | 24.6704% |
| 10 | 23.9799% |
| 12 | 24.8588% |
| 14 | 24.5449% |

# 5 Testing

## 5.1 C4.5

Decision tree is a popular algorithm in data mining and machine learning. It represents a function in form of a decision tree, each node denote an attribute, and each edge represent the condition which lead to the next node. Leaves represent labels, which are the result of classification. Decision tree is efficient and simple. The result tree can be visualized, and pretty human readable. There are many decision tree algorithms, and we only use C4.5 in this paper, which is an improved version of ID3 algorithm. C4.5 algorithm follows a simple inductive bias, which is "smaller trees are preferred", which means that smaller trees tend to perform better than bigger trees on unseen data. This inductive bias is motivated by Occan's Razor principle, the primal principle in machine learning. A free implementation of C4.5 algorithm in java is available in J48 package.

Decision tree has the problem of over fitting. C4.5 use some adjustable parameters to perform pruning, limit the size of the tree, and therefore reduce over fitting. To achieve good performance, we perform parameter search on two parameters: $C$ and *minObj*. C is the confident factor, which is a probability threshold of the probability of the actual error rate is worse than the pessimistic estimation based on some reasonable assumption. The pessimistic estimation of the error rate for each subtree is used when do the subtree replacement and subtree raising pruning in C4.5 algorithm. So smaller confidence factor implies more pessimistic estimation, which will cause more drastic pruning. The *minObj* variable limit the minimum number of examples each leaf could have. This constrain can help reduce the effect of noise in the data, especially then the data set is large.

| C\ minObj | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| 1e-3 | 24.6077 % | 24.3566 % | 24.1682 % | 24.6704 % |
| 1e-2 | 24.4193 % | 24.4821 % | 24.231 % | 24.5449 % |
| 1e-1 | 24.6077 % | 24.231 % | 24.231 % | 24.6077 % |
| 0.5 | 24.2938 % | 24.0427 % | 24.1055 % | 24.8588 % |
| 1 | 24.6077 % | 23.7916 % | 24.1682 % | 24.796 % |

The result in the table above shows that he error rate of C4.5 algorithm is lowest when $C = 1$ and $minObj = 2$, which gives error rate of 23.7916%. This implies that not much pruning is needed to achieve good performance for C4.5 on this data set. Maybe it is because the size of data set is small, and noise in the data set is very little.

This result is very poor compared to many other popular algorithms like support vector machine(will see it later). This is because the limitation of decision tree itself. Although a decision tree with no size limit can be proved to form any function, C4.5 algorithm prefers smaller trees, which potentially reduce the number candidate hypothesis. So sometimes the result decision tree may not have a very high accuracy.

### 5.2 Boosting

Boosting is an algorithm which aims to boost week classifier into a stronger classifier. The boosting algorithm used here is AdaBoosting M1, which is generalization of AdaBoosting algorithm on multiple class classification problem. It requires over 50% accuracy of "week classifier" on the data set in order to "boost" it into a stronger classifier.

The boosting algorithm constructs the classifier by several iterations, each iteration obtains a classifier by a weighted data set, and the output of the final classifier is obtained by weighted sum of the results of all classifiers. After each iteration, data set is reweighted, and misclassified instances has a larger weight so that they are paid more attention on in the next iteration. The weight of each classifier is decided by their accuracy. Boosting works well with decision tree in practice, and it often does not suffer from over fitting problem.

Following result is obtained by applying boosting on C4.5 algorithm($C = 1$ and $minObj = 2$) with different number of iterations.

| Number of Iterations | Error rate |
|---|---|
| 2 | 23.7916% |
| 4 | 17.7024% |
| 8 | 13.4965% |
| 16 | 9.6673% |
| 32 | 7.7841% |

From the table, we can see that the error rate drops rapidly when the number of iteration grows. Although higher accuracy can be achieved by further increase the number of iterations, finally the error rate would stop decreasing or start to increase.

Boosting algorithm constructs a classifier by combining several classifiers, which intuitively increase the complexity of the classifier. However, according to Occan's Razor principle, simpler classifiers tend to generalize better on unseen data. Boosting algorithm seems a contradict with Occan's Razor principle.

One possible explanation is that the complexity of classifier is not measured by the number of classifiers, but by the "margin" of the classifier. In Boosting algorithm, each classifier vote to a class by its weight, and the class with the highest weight is the result of classification. The "margin" of an example is defined by the difference between weight of correct class, and the weight of the sum of all weights of incorrect class. The larger the margin is , the more confident the classification for this instance is. The boosting algorithm tend to maximize margin of instances, and convergence with some large margin distribution. Sometimes we can observe that when running boosting algorithm, the testing error keeps on decreases even after the error rate of training set is 0. This is because when the error rate of testing data is zero, the margin distribution may not have converged yet, and as margin increases, the error rate on testing data set decreases.

## 5.3 Support Vector Machine

Support vector machine can be seen as a generalized version of linear classifier. To classify data set which are not linear separatable, SVM project each example into a higher dimensional feature space, and find a hyper plane, which separate different classes. The margin of hyper plane is defined as the distance of the plane to the closest example. Hyper plane with large margin is preferred in SVM. All calculation of feature vector with high dimension can be done by kernel function, which reduce the computation time dramatically. Different kernel function can be used for different feature space. Although new kernels are being purposed by researchers, four basic kernels are post popular:(ref: A Practical Guide to Support Vector Classification)

- linear: $K(x_i, x_j) = x_i^T x_j$
- polynomial: $K(x_i, x_j) = (\gamma x_i^T x_j + r)^d$, $\gamma > 0$
- radial basis function (RBF): $K(x_i, x_j) = \exp(-\gamma + \|x_i - x_j\|^2)$, $\gamma > 0$
- sigmoid: $K(x_i, x_j) = tanh(\gamma x_i^T x_j + r)$

According to Keerthi and Lin (2003) and (Lin and Lin, 2003), linear kernel is a special case of RBF kernel, and sigmoid kernel behaves like RBF kernel with certain parameters. So in this paper, we only use RBF kernels.

There are many kinds of SVMs based on different error function it aims to minimize. In this paper, we simply use C-svm, which minimize function: subject to:

$$\frac{1}{2}\|\vec{\beta}\|^2 + C\sum_{i=1}^{n}\xi_i$$

subject to:

$$y_i(\vec{\beta}.\vec{x_i} + \beta 0) \geq 1 - \xi_i$$

$$\forall i, \xi_i \geq 0$$

*libsvm* package(`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`) implemented by Chih-Chung Chang and Chih-Jen Lin is used for training and testing purpose in this paper.

For C-SVM using RBF kernel, parameters $C$ and $\gamma$ are adjustable. We follow the way proposed in (guide.pdf) to perform feature scaling parameter search to achieve a good performance. We use tools provided in *libsvm* package to perform the model selection automatically.
As shown in Figure 1, the SVM tends to perform best when $C = 2.0$ and $\gamma = 0.0078125$, and the error rate is 4.2059%.

# 6 Improvement

## 6.1 Motivation

One observation is that for decision tree and SVM, the order of attributes is not important for training and classification. In other word, if I apply some fixed permutation on the attributes of each example in the data set, then decision tree and SVM will generate almost the same model as the model generated by original data set. However, the order of pixel is important in the process of human being recognizing a hand written digit. If the rearrange the pixel of an image randomly, a human is very unlikely to recognize what's in the image. Moreover, some properties of the data set are harder to learn for certain classifiers(e.g. decision tree are not good at representing XOR function). This motivate the preprocessing of data. More specifically, we try to add some attributes which provide those information which are hard to learn by machine learning algorithms.

## 6.2 Adding Features

34 extra features are added into each examples.The modified data set has 290 features for each example. To provide information about the position of pixels, we choose to add 16 extra attributes $r[16]$, one for each row, which describes how many "black segments" are there in the row. In image of 1, most of $r[i]$ can be expected to be 1, and some can be 2, depends on the way of writing. In image of 0, most $r[i]$ can be expected to be 2. Similarly, 16 more attributes $c[16]$ is added for each column, which means the number of 'black segments" in each column.
The variance of the number of black pixels in each row, and the variance of the number of black pixels in each column are also added into each example.

### 6.3 Testing

| Algorithm | Original Data | Modified Data |
|---|---|---|
| C4.5($C = 1, minObj = 2$) | 23.7916% | 18.5813% |
| Boosted C4.5(16 iterations) | 9.6673% | 6.0891% |
| SVM($C = 2, \gamma = 0.0078125$) | 4.2059% | 3.2643% |

In the testing, From the result, we can see improvements on all algorithms are significant. All algorithm's error rate on modified data drop down to the $\frac{3}{4}$ of the original data.
34 attributes is a small number compared to original 256 attributes, so the increase in time cost is trivial. So adding extra information tend to help improve accuracy for these algorithms.

## 7 Conclusion

By applying several machine learning algorithms on Semeion Handwritten Digit Data Set, we can observe that C4.5 algorithm performs poorly, while boosted C4.5 has a much better performance. C-SVM with RBF kernel performs very well on this data set. By adding extra information into the data set, the error rate for all these algorithm drops significantly, and SVM with polynomial d-2 kernel achieves error rate of 3.2643%.

## 8 References