Generalizing Binary Classifiers to the Multiclass Case

Dain, Eliyahu  
eliyahud@post.tau.ac.il

Zach, Rotem  
rotemzac@post.tau.ac.il

March 3, 2013

Abstract

The course described several ways to solve binary classification problems. This paper describes the evaluation of several different ways to extend binary classifiers to the multiclass case. The focus is on empirical experiments rather than rigorous analysis of the different algorithms.

For the empirical experiments, AdaBoost was used as the binary classifier. The MNIST handwritten digit was the data set. Several different multiclass algorithms were tested.

1 Introduction

The binary classification problem can be described as such [6]: let $X$ be a set, $D$ a distribution over $X$ and $H$ a hypothesis class. Suppose that $f: X \rightarrow \{-1, 1\}$ is the unknown target function.

Then the binary classification problem is:

Given a set $S = \{(x, f(x))\}$ of random examples drawn i.i.d. according to the distribution $D$, find a hypothesis $h \in H$ such that error $(h, f) \leq \varepsilon$ with probability $1 - \delta$ where:

$$\text{error} (h) = \mathbb{E}_{x \sim D} [I_{h(x) \neq f(x)}]$$

Two different ways to extend this to a multiclass problem are:

1. Monolabel – If $f: X \rightarrow \{1, 2, \ldots, k\}$, this is a $k$-class monolabel problem.

In this case the error function is the same as in the binary problem:

$$\text{error} (h) = \mathbb{E}_{x \sim D} [I_{h(x) \neq f(x)}]$$

2. Multilabel – If $f: X \rightarrow \{-1, 1\}^k$, this is a $k$-label multilabel problem.

The error function is:

$$\text{error} (h) = \mathbb{E}_{x \sim D} \left[ \sum_{i=1}^{k} I_{h(x)_i \neq f(x)_i} \right]$$

There are several algorithms that solve multiclass classification and they can be divided into two main groups:

**Aggregated** These algorithms reduce the problem to multiple binary classification problems. Included in this group are: one versus all, one versus one, error-correcting output codes and filter trees.
These algorithms are specifically designed for multiclass classification. Included in this group are versions of SVMs, a version of AdaBoost and decision trees. This paper concentrates on the monolabel case and implementation of several aggregated algorithms. We will compare the algorithms performances on the MNIST handwritten digit data set [5].

2 Aggregated Algorithms

Let \( S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \subseteq X \times \{1, 2, \ldots, k\} \) be the labeled training samples.

2.1 One versus All

In one-versus-all [6] (OVA), \( k \)-binary classifiers \( f_i : X \rightarrow [-1, 1] \), \( 1 \leq i \leq k \) are learned independently. When learning \( f_i \), the training set is partitioned into two sets, one being the samples labeled \( i \) and the other all the rest.

Formally the input to the binary classifier \( f_i \) is \( S_i = \{(x_j, y^i_j)\} \) where:

\[
y^i_j = \begin{cases} 
1 & y_j = i \\
-1 & \text{otherwise}
\end{cases}
\]

This classifier classifies \( i \) versus all other labels.

The hypothesis \( h_{OVA} \) returned by this algorithm is:

\[
h_{OVA}(x) = \arg\max_{1 \leq i \leq k} \{f_i(x)\}
\]

Note that it is not enough for the algorithm that the binary classifier will return a value in \([-1, 1]\). A value in the range \([-1, 1]\) is required: as the value is further from 0, more confidence is placed in its prediction.

2.2 One versus One

In one-versus-one [6] (OVO), \( \binom{k}{2} \) binary classifiers \( h_{i,\ell} : X \rightarrow \{-1, 1\} \), \( 1 \leq i < \ell \leq k \) are learned independently. When learning \( h_{i,\ell} \) the training set is partitioned into two sets, one being the samples labeled \( i \) and the other the samples labeled \( \ell \).

Formally the input to the binary classifier \( h_{i,\ell} \) is \( S_{i,\ell} = \{(x,i) \mid (x, i) \in S\} \cup \{(x,-1) \mid (x, \ell) \in S\} \).

Note that \( S_{\ell,i} = \{(x,-y) \mid (x,y) \in S_{i,\ell}\} \).

The hypothesis \( h_{OVO} \) returned by this algorithm is a majority vote among the binary classifiers:

\[
h_{OVO}(x) = \arg\max_{1 \leq i \leq k} |\{\ell \mid h_{i,\ell}(x) = 1\}|
\]

2.3 Error Correcting Output Code

Error correcting output code [3] (ECOC) is a generalization of the previous two algorithms. In this algorithm, a matrix \( M \in \{-1, 0, 1\}^{k \times m} \) is created, consisting of \( k \) rows and a variable number of columns, \( m \) (depending on the chosen coding scheme).

Each row \( M_i \) corresponds to a class, and each column corresponds to a binary classifier.
Denote the $\ell$'th binary classifier as $C_\ell$. Its input is all the classes whose values in $M$ are either $-1$ or $1$. Formally:

$$S_\ell = \{ (x_j, 1) \mid M_{\ell,y_j} = 1 \} \cup \{ (x_j, -1) \mid M_{\ell,y_j} = -1 \}$$

When classifying a sample $x$, a vector $v(x)$ is calculated:

$$v(x) = (C_1(x), C_2(x), \ldots, C_m(x))$$

The algorithm returns the class whose row has the minimal Hamming distance from $v(x)$:

$$h_{\xi\text{COC}}(x) = \arg\min_{1 \leq i \leq k} \text{HD}(M_i, v(x))$$

Adding the third value, $0$, allows greater generalization abilities and enables implementing OVA and OVO. OVA can be described as a square $k$ matrix where the $i$'th column has the $i$'th element as $1$, and the others as $-1$. Similarly, OVO can be described as a $k \times \binom{k}{2}$ matrix, where each row has a single $1$ and a single $-1$, and the other elements are $0$.

When choosing $M$, two properties should be satisfied:

1. **Row separation.**
   
   Each row $M_i$ should have a well separated Hamming distance from the values of the other rows. If the minimal Hamming distance of two rows in the matrix is $d$, the error correcting code will be able to fix $\left\lfloor \frac{d-1}{2} \right\rfloor$ binary classification errors.

2. **Column separation.**
   
   Each column $C_l$ should have a well separated Hamming distance from the values of the other columns and from the complements of the other columns. The codes ability to fix errors relies on these errors to be independent. If two columns have a small Hamming distance then there will be a dependency in their errors.

### 2.3.1 Naïve Error Correcting Output Code

The naïve way to find such a matrix $M$ would suggest using an exhaustive code to create the matrix [3]. This code is of length $2^{k-1} - 1$ and is constructed as follows:

- **Row 1** consists of $2^{k-1} - 1$ ones.
- **Row 2** consists of $2^{k-2}$ minus ones followed by $2^{k-2} - 1$ ones.
- **In general**, the $i$'th row consists of runs of $2^{k-i}$ minus ones and ones.

From the fact that the first column is the bit representation of $2^{k-1}$ (when $-1$ counts as $0$ in this representation), and the $i$'th column is the bit representation of $2^{k-1} + (i - 1)$, one can deduce an easier way of calculating this matrix.

The advantage of this coding scheme is that the minimal Hamming distance between two rows is $2^{k-3}$, and no two columns are equal or complementary. The disadvantage of this scheme is that the code size is exponential in the number of classes $k$.

An example of an exhaustive coding scheme could be found in table 1.
<table>
<thead>
<tr>
<th>$f_1$</th>
<th>$f_2$</th>
<th>$f_3$</th>
<th>$f_4$</th>
<th>$f_5$</th>
<th>$f_6$</th>
<th>$f_7$</th>
<th>$f_8$</th>
<th>$f_9$</th>
<th>$f_{10}$</th>
<th>$f_{11}$</th>
<th>$f_{12}$</th>
<th>$f_{13}$</th>
<th>$f_{14}$</th>
<th>$f_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 1: The exhaustive coding scheme matrix for $k = 5$

### 2.3.2 Other Error Correcting Output Codes

Allwein, Schapire, Singer and Kaelbling [1] suggest two different randomized approaches for choosing the matrix $M$:

1. **Dense Random.**
   - This type of code has a length of $\lceil 10 \log_2 (k) \rceil$, and each entry is chosen uniformly between $\{-1, 1\}$. 10,000 different matrices are created, and the matrix with the largest Hamming minimum distance between its rows, and with no two identical or complimentary columns is chosen.

2. **Sparse Random.**
   - This type of code has a length of $\lceil 15 \log_2 (k) \rceil$, and each entry is chosen with probabilities $\Pr[0] = \frac{1}{2}$ and $\Pr[-1] = \Pr[1] = \frac{1}{4}$. As before, this is done 10,000 times with the same considerations as before for choosing the best matrix, but additional care needs to be taken so that the chosen matrix does not have any row or column that consist of pure 0’s.

### 2.4 Filter Trees

Filter Trees [2] are refined divide and conquer trees. Let $T_{\mathcal{FT}}$ be a binary tree whose leaves are the classes $\{1, 2, \ldots, k\}$. The tree $T_{\mathcal{FT}}$ is created going bottom-up, as demonstrated in figure 1.

Let $n$ be a node and denote all the classes in the sub tree of node $n$ as classes $(n)$. Going bottom up, a binary classifier that chooses between classes $(n_{\text{left}})$ and classes $(n_{\text{right}})$ is constructed for each node $n$.

The key trick of this algorithm is how the training sets are chosen. A sample $(x, y)$ is chosen for the training set of a node $n$ if and only if $y \in$ classes $(n_{\text{left}}) \cup$ classes $(n_{\text{right}})$, and all the nodes on the path between leaf $(y)$ and $n$ predicted the label of $x$ correctly.

The evaluation of this tree is done top-down, as described in algorithm 1.

### 2.5 All Pairs Filter Trees

All Pairs Filter Trees [2] are a variation of filter trees. Let $T_{\mathcal{APFT}}$ be a binary tree whose leaves are the classes $\{1, 2, \ldots, k\}$.

As in OVO, when creating the tree a binary classifier between each two classes is created. Unlike regular filter trees, the samples for each class are all the samples in the training set.

The tree is evaluated bottom-up, as described in algorithm 2.
Algorithm 1 Filter tree evaluation

function EvaluateFilterTree(Filter tree $T_{F_T}$, Sample $s$)

Start with $n \leftarrow \text{Root}(n)$

repeat

   if $\text{Classify}(n, s) = -1$ then

      $n \leftarrow \text{Left}(n)$

   else

      $n \leftarrow \text{Right}(n)$

   end if

until $n$ is a leaf

return $\text{Class}(n)$

end function
Algorithm 2 All pairs filter tree evaluation

function EvaluateAllPairsFilterTree(Filter tree $T_{APFT}$, Sample $s$)
    for $\ell \in \text{Leaves}(T_{APFT})$
        Define $\ell$.winner $\leftarrow \text{Class}(\ell)$
    end for
    for $n \in \text{Nodes}(T_{APFT})$
        if $\text{Classify}(n, s) = 1$
            Define $n$.winner $\leftarrow \text{Left}(n).winner$
        else
            Define $n$.winner $\leftarrow \text{Right}(n).winner$
        end if
    end for
    return $\text{Root}(T_{APFT}).winner$
end function

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of classifiers</th>
<th>Number of evaluations</th>
</tr>
</thead>
<tbody>
<tr>
<td>One versus All</td>
<td>$k$</td>
<td>$k$</td>
</tr>
<tr>
<td>One versus One</td>
<td>$\binom{k}{2}$</td>
<td>$\binom{k}{2}$</td>
</tr>
<tr>
<td>Exhaustive ECOC</td>
<td>$2^k - 1$</td>
<td>$2^k - 1$</td>
</tr>
<tr>
<td>Dense ECOC</td>
<td>$10 \log_2 (k)$</td>
<td>$10 \log_2 (k)$</td>
</tr>
<tr>
<td>Sparse ECOC</td>
<td>$15 \log_2 (k)$</td>
<td>$15 \log_2 (k)$</td>
</tr>
<tr>
<td>Filter Trees</td>
<td>$k - 1$</td>
<td>$\approx \log (k)$</td>
</tr>
<tr>
<td>All Pairs Filter Trees</td>
<td>$\binom{k}{2}$</td>
<td>$k - 1$</td>
</tr>
</tbody>
</table>

Table 2: A complexity comparison of different aggregated multiclass algorithms

2.6 Complexity Comparison

Table 2 summarizes the number of binary classifiers needed for each algorithm.

The column “Number of classifiers” describes the total number of binary classifiers that have to be trained during the learning phase of the multiclass classifier.

The column “Number of evaluations” describes the total number of binary classifier evaluations done in a classification of a single multiclass sample. Each binary classifier is counted once.

3 Results

3.1 Data Set

The data set that we used is the MNIST handwritten digit data set [5]. It is composed of images of handwritten digits that have been size normalized and centered in a $28 \times 28$ grid.

The original data set is composed of a 60,000 image training set and a 10,000 image testing set. Seeing that the partition of the images into training and testing sets was not random, both sets were merged to a set of 70,000 images.
3.2 Binary Classifiers

For the binary classifiers, the AdaBoost algorithm [4] was used. The weak classifiers (as the input for the AdaBoost classifiers) used threshold for the gray scale of single pixels.

For example, a weak classifier $WC^+\theta$ with positive threshold $\theta$, would classify a pixel whose gray scale value is $p$ as follows:

$$h_{WC^+\theta} (p) = \begin{cases} 1 & x \geq \theta \\ -1 & x < \theta \end{cases}$$

A weak classifier $WC^-\theta$ with negative threshold $\theta$, would classify a pixel whose gray scale value is $p$ as follows:

$$h_{WC^-\theta} (p) = \begin{cases} 1 & x \leq \theta \\ -1 & x > \theta \end{cases}$$

3.3 Experiments

Seven different algorithms were tested:

1. One versus All (OVA)
2. One versus One (OVO)
3. Error Correcting Output Code using an exhaustive coding scheme (EECOC)
4. ECOC using a dense random coding scheme (DECOC)
5. ECOC using a sparse random coding scheme (SECOC)
6. Filter Trees (FT)
7. All Pairs Filter Trees (APFT)

The testing was done using 10-fold cross validation on the merged set of 70,000 images, each time choosing a random subset consisting of 100, 250, 500, 1,000, 2,500, 5,000, 10,000, 20,000 or 30,000 images (with the exception of EECOC which was trained with up to 11,000 samples). The AdaBoost binary classifier was run through 30 iterations.

Figure 2 shows a graph of the error rate of each algorithm when trained with varying sizes of training sets.

4 Conclusions

In this paper we experimentally compared seven algorithms for reducing multiclass classification to binary classification.

All the algorithms managed to learn even with very small training sets, less than 500 images. There were significant improvements as the size of the training set increases, but only up to 10,000 images. With training sets larger than that, the error rates of the algorithms are only marginally improved.

Of all the algorithms tested, One versus One had the smallest error rate, with All Pairs Filter Trees and Sparse Random Error Correcting Output Code close behind. Seeing as One versus One has a larger computational complexity than the other two algorithms, it seems that All
Figure 2: The error rates of the experiments
Pairs Filter Trees and Sparse Random Error Correcting Output Code are preferable when the number of classes \(k\) is large.

The other three algorithms (One versus All, Dense Random Error Correcting Output Code and Filter Trees) consistently lagged behind. Exhaustive Error Correcting Output Code was very computationally intensive and we weren’t able to test it with large enough training sets to make reasonable conclusions regarding its performance.

When compared to the algorithms stated in the MNIST web site [5], even the best algorithms we tested were far from state of the art.

It would be interesting to check whether these conclusions are still valid when testing other data sets and different binary classifiers.

Sources and Data

The sources and data sets attached to this paper can be found at the website: http://www.cs.tau.ac.il/~eliyahud/ml.

References


[3] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-

