Multiclass Classification

Yossi Keshet

March 24, 2014

Assume that we have a training set of examples $S = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$, where the instances $\boldsymbol{x}_i \in \mathbb{R}^d$ are vectors of length d and the labels are from a set of k possible classes, $y_i \in \mathcal{Y} = \{1, 2, \ldots, k\}$.

There are generally two techniques to build a multiclass classifier: (i) reduction of the multiclass problem to several binary classification problems; and (ii) design of a multiclass classifier.

1 Binary reductions

1.1 One-against-All (OA)

The One-against-All (OA) method is based on a reduction of the multiclass problem into k binary problems, each of which discriminates between one class to all the rest of the classes, that is, in the first binary problem we assign the label $+1$ to all examples labeled 1 and the label -1 to all examples labeled 2, 3, ..., k. We call the resulted weight vector $\boldsymbol{w}^1$. In the second binary problem we assign the label $+1$ to all examples labeled 2 and the label -1 to all examples labeled 1, 3, 4, ..., k. We call the resulted weight vector $\boldsymbol{w}^2$, and so on. We end up with k weight vectors. Given a new instance $\boldsymbol{x}$ we predict the label that gets the highest confidence

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} \boldsymbol{w}^y \cdot \boldsymbol{x} \ .$$

1.2 All Pairs (AP)

A different reduction is the All-Pairs (AP) approach in which all pairs of classes are compared to each other. In the first binary problem we assign the label $+1$ to all examples labeled 1 and the label -1 to all examples labeled 2 and get the weight vector $\boldsymbol{w}^{12}$. In the second binary problem we assign the label $+1$ to all examples labeled 1 and the label -1 to all examples labeled 3 and get the weight vector $\boldsymbol{w}^{13}$ and so on. We end up with $(k-1)k/2$ weight vectors. Given a new instance $\boldsymbol{x}$ we predict the label that gets the highest confidence in the following way

$$\hat{y} = \operatorname*{argmax}_{y \in \mathcal{Y}} \sum_{r \neq y} \boldsymbol{w}^{yr} \cdot \boldsymbol{x}$$

1.3 Error Correction Output Codes (ECOC)

The following section is taken from Allwein et al. (2000).
The two approaches OA and AP have been unified under the framework of Error Correction Output Codes (ECOC). Their idea is to associate each class $r \in \mathcal{Y}$ with a row of a coding matrix $M \in \{-1, 0, +1\}^{k \times l}$ for some l (there are l binary problems). The binary learning algorithm is then run once for each column of the matrix on the induced binary problem in which the label of each example labeled y is mapped to $M(y, s)$. This yield l hypotheses f_s. Given an example $\boldsymbol{x}$, we then predict the label y for which row y of matrix M is "closest" to $(f_1(\boldsymbol{x}), \ldots, f_l(\boldsymbol{x}))$. Some of the entries $M(r, s)$ may be zero, indicating that we don't care how hypothesis f_s categorizes examples with label r.

Thus, the scheme for learning multiclass problems using a binary margin-based learning algorithm A works as follows. We begin with a given coding matrix $\boldsymbol{M}$. For $s = 1, \ldots, l$, the learning algorithm A is provided with labeled data of the form $(\boldsymbol{x}_i, M(y_i, s))$ for all examples i in the training set but omitting all examples for which $M(y_i, s) = 0$. The algorithm A uses this data to generate a hypothesis f_s.

For example, for the one-against-all approach, $\boldsymbol{M}$ is a $k \times k$ matrix in which all diagonal elements are $+1$ and all other elements are -1. For all-pairs approach, $\boldsymbol{M}$ is $k \times \binom{k}{2}$ matrix in which each column corresponds to a distinct pair (r_1, r_2). For this column, $\boldsymbol{M}$ has $+1$ in row r_1, -1 in row r_2 and zeros in all other rows.

We now describe the how prediction is done. Let $\boldsymbol{M}(r)$ denote row r of $\boldsymbol{M}$ and let $\boldsymbol{f}(\boldsymbol{x})$ be the vector of predictions on an instance $\boldsymbol{x}$:

$$\boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \ldots, f_l(\boldsymbol{x}))$$

Given the predictions of the $\boldsymbol{f}(s)$'s on a test point $\boldsymbol{x}$, which of the k labels in $\mathcal{Y}$ should be predicted? The basic idea of both methods is to predict with the label r whose row $\boldsymbol{M}(r)$ is "closest" to the predictions $\boldsymbol{f}(\boldsymbol{x})$. In other words, predict the label r that minimizes $d(\boldsymbol{M}(r), \boldsymbol{f}(\boldsymbol{x}))$ for some distance d. This formulation begs the question, however, of how we measure distance between the two vectors.

One way of doing this is to count up the number of positions s in which the sign of the prediction $f_s(\boldsymbol{x})$ differs from the matrix entry $M(r, s)$. Formally, this means our distance measure is

$$d_H(\boldsymbol{M}(r), \boldsymbol{f}(\boldsymbol{x})) = \sum_{s=1}^{l} \left(\frac{1 - \text{sign}(M(r, s)\, f_s(\boldsymbol{x}))}{2} \right) \tag{1}$$

where $\text{sign}(z)$ is $+1$ if $z > 0$, -1 if $z < 0$, and 0 if $z = 0$. Note that if either $M(r, s)$ or $f_s(\boldsymbol{x})$ is zero then that component contributes $1/2$ to the sum. For an instance $\boldsymbol{x}$ and a matrix $\boldsymbol{M}$, the predicted label is

$$\hat{y} = \underset{y \in \mathcal{Y}}{\text{argmin}}\, d_H(\boldsymbol{M}(r), \boldsymbol{f}(\boldsymbol{x}))$$

This prediction is called *Hamming decoding.*

A disadvantage of this method is that it ignores entirely the magnitude of the predictions which can often be an indication of a level of "confidence." Our second method for combining predictions takes this potentially useful information into account, as well as the relevant loss function L which is ignored with Hamming decoding. The idea is to choose the label r that is most consistent with the predictions $f_s(\boldsymbol{x})$ in the sense that, if example $\boldsymbol{x}$ were labeled r, the total loss on example $(\boldsymbol{x}, r)$ would be minimized over choices of $r \in \mathcal{Y}$. Formally, this means that our distance measure is the total loss on a proposed example $(\boldsymbol{x}, r)$:

$$d_L(\boldsymbol{M}(r), \boldsymbol{f}(\boldsymbol{x})) = \sum_{s=1}^{l} L(M(r,s)\, f_s(\boldsymbol{x})) \tag{2}$$

and the prediction:

$$\hat{y} = \operatorname*{argmin}_{y \in \mathcal{Y}} d_L(\boldsymbol{M}(r), \boldsymbol{f}(\boldsymbol{x}))$$

This prediction is called *loss-based decoding.* The loss is $L(z) = \max\{0, 1 - z\}$ for SVM, $L(z) = \exp(-z)$ for boosting, and $L(z) = \log(1 + \exp(-z))$ for logistic regression.

Example. Let the output of the binary classifier be

$$\boldsymbol{f}(\boldsymbol{x}) = [0.5, -7, -1, -2, -10, -12, 9]$$

The sign of the binary classifier is

$$\text{sign}(\boldsymbol{f}(\boldsymbol{x})) = [+1, -1, -1, -1, -1, -1, +1]$$

Let the coding matrix be

$$\boldsymbol{M} = \begin{bmatrix} -1 & 0 & -1 & -1 & +1 & -1 & -1 \\ +1 & -1 & 0 & +1 & +1 & +1 & -1 \\ +1 & 0 & -1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & 0 & -1 & -1 & +1 \end{bmatrix}$$

The Hamming distance of $\text{sign}(\boldsymbol{f}(\boldsymbol{x}))$ to the row of $\boldsymbol{M}$ according to Eq. (1) is

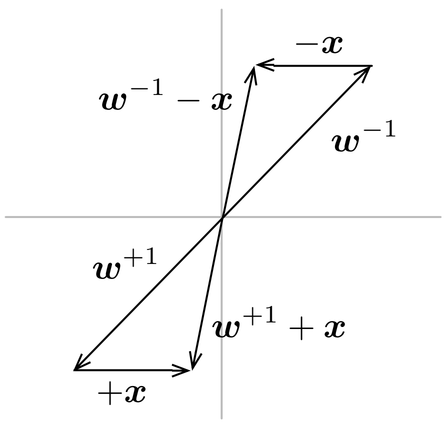
$$\begin{bmatrix} -1 & 0 & -1 & -1 & +1 & -1 & -1 \\ +1 & -1 & 0 & +1 & +1 & +1 & -1 \\ +1 & 0 & -1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & 0 & -1 & -1 & +1 \end{bmatrix} \qquad \begin{array}{l} D = 3.5 \\ D = 4.5 \\ D = 1.5 \\ D = 2.5 \end{array}$$

Hence the prediction here is **class 3** (the minimal Hamming distance).

Assume the loss function is the exponential loss $L(z) = \exp(-z)$. The loss-based distances according to Eq. (2) are

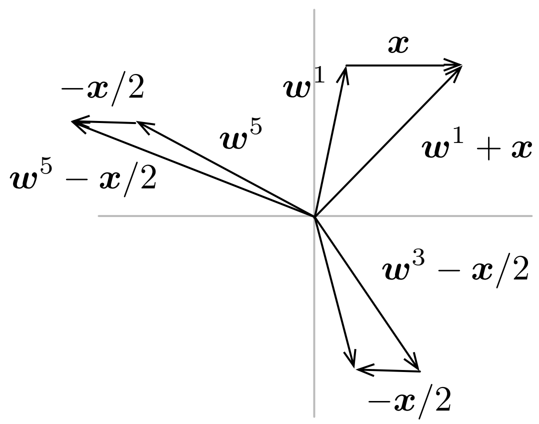
$$\begin{bmatrix} -1 & 0 & -1 & -1 & +1 & -1 & -1 \\ +1 & -1 & 0 & +1 & +1 & +1 & -1 \\ +1 & 0 & -1 & -1 & -1 & +1 & +1 \\ -1 & -1 & +1 & 0 & -1 & -1 & +1 \end{bmatrix} \qquad \begin{array}{l} D = \exp(-1 * 0.5) + \exp(0 * -7) + \ldots = 30133.0 \\ D = 192,893.0 \\ D = 162,757.0 \\ D = 5.4 \end{array}$$

Hence the prediction in this case is **class 4** (the minimal loss-based distance).



$-\boldsymbol{x}$
$\boldsymbol{w}^{-1} - \boldsymbol{x}$
$\boldsymbol{w}^{-1}$
$\boldsymbol{w}^{+1}$
$\boldsymbol{w}^{+1} + \boldsymbol{x}$
$+\boldsymbol{x}$

(a) A subfigure



$\boldsymbol{x}$
$-\boldsymbol{x}/2$
$\boldsymbol{w}^1$
$\boldsymbol{w}^5$
$\boldsymbol{w}^1 + \boldsymbol{x}$
$\boldsymbol{w}^5 - \boldsymbol{x}/2$
$\boldsymbol{w}^3 - \boldsymbol{x}/2$
$-\boldsymbol{x}/2$

(b) A subfigure

Figure 1: A figure with two subfigures

2 Multiclass

2.1 Multiclass Perceptron

The multiclass perceptron should be regarded as direct extension of the binary Perceptron. In the binary Perceptron, where $y_i \in \{-1, +1\}$, the update rule for example $(\boldsymbol{x}_i, y_i)$ which was wrongly classified is

$$\begin{aligned}
&\hat{y} = \text{sign}(\boldsymbol{w} \cdot \boldsymbol{x}_i) \\
&\text{if } \hat{y} \neq y_i : \\
&\qquad \boldsymbol{w} \leftarrow \boldsymbol{w} + y_i \boldsymbol{x}_i
\end{aligned}$$

The binary case can be extended as follows. We have two classes -1 and $+1$, and two weight vectors $\boldsymbol{w}^{-1}$ and $\boldsymbol{w}^{+1}$, where $\boldsymbol{w}^{-1} = -\boldsymbol{w}^{+1}$. The update rule can be:

$$\begin{aligned}
&\hat{y} = \underset{y \in \{-1,+1\}}{\text{argmax}}\ \boldsymbol{w}^y \cdot \boldsymbol{x} \\
&\text{if } \hat{y} \neq y_i : \\
&\qquad \boldsymbol{w}^y \leftarrow \boldsymbol{w}^y + \boldsymbol{x}_i \\
&\qquad \boldsymbol{w}^{\hat{y}} \leftarrow \boldsymbol{w}^{\hat{y}} - \boldsymbol{x}_i
\end{aligned}$$

The update rule is depicted in Figure 1a. The reader can verify that both algorithms are mathematically the same up to factor of 2 between the weight vectors $\boldsymbol{w}^{-1} = -\boldsymbol{w}^{+1}$ and $\boldsymbol{w}$.

The extension of the binary Perceptron to multiclass is straightforward now. Define the set of weight vectors $\{\boldsymbol{w}^1, \ldots, \boldsymbol{w}^k\}$ to be the weights corresponding to classes. The *multiclass Perceptron* algorithm may be:

$$\begin{aligned}
&\hat{y} = \underset{y \in \{1,\ldots,k\}}{\text{argmax}}\ \boldsymbol{w}^y \cdot \boldsymbol{x} \\
&\text{if } \hat{y} \neq y_i : \\
&\qquad \boldsymbol{w}^r \leftarrow \boldsymbol{w}^r + \tau_r \boldsymbol{x}_i
\end{aligned}$$

where

$$\tau_r = \begin{cases} +1 & \text{if } r = y_i \\ -1 & \text{if } r = \hat{y} \\ 0 & \text{otherwise} \end{cases} .$$

Define the set of all classes which have higher score from the true label:

$$E_y = \{r \neq y \mid \boldsymbol{w}^r \cdot \boldsymbol{x} > \boldsymbol{w}^y \cdot \boldsymbol{x}\}$$

Another option for an update might be

$$\tau_r = \begin{cases} +1 & \text{if } r = y_i \\ -1/|E_{y_i}| & \text{if } r \in E_{y_i} \\ 0 & \text{otherwise} \end{cases} .$$

The first option updates only two classes: the correct one and the predicted one. The second option updates all classes that have higher score from the correct one. Example of such update is depicted in Figure 1b. Similar to the binary case, a mistake bound can be derived for the multiclass Perceptron algorithm. Finally, note that other options for τ_r are available.

2.2 Multiclass SVM

Let $Py|\boldsymbol{x})$ be the (true) probability of the class y given that the instance is $\boldsymbol{x}$. The Bayesian decision rule for multiclass prediction is

$$\hat{y} = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} \; P(y|\boldsymbol{x})$$

The performance are measured in terms of probability of error $P(\hat{y} \neq y)$ Let our model be of the following form

$$\hat{y} = \underset{y \in \{1,\ldots,k\}}{\operatorname{argmax}} \, \boldsymbol{w}^y \cdot \boldsymbol{x}.$$

We would like to find the weight vectors $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k\}$ such that the probability of error $\hat{y} \neq y$ is minimized. Namely

$$\begin{aligned} \{\boldsymbol{w}^r\} &= \underset{\{\boldsymbol{w}^r\}}{\operatorname{argmin}} \, P(\hat{y} \neq y) && (3) \\ &= \underset{\{\boldsymbol{w}^r\}}{\operatorname{argmin}} \, E\Big[\delta(\hat{y} \neq y)\Big] && (4) \end{aligned}$$

where E is the expectation with respect to the probability P, and δ is Kronecker delta function. Remember that $\hat{y}$ depends on the set of parameters $\{\boldsymbol{w}^r\}$. Now we replace the expectation in the last equation with average and add regularization to get

$$\{\boldsymbol{w}^r\} = \underset{\{\boldsymbol{w}^r\}}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \delta(\hat{y} \neq y) + \frac{\lambda}{2} \sum_{r=1}^{k} \|\boldsymbol{w}^r\|^2$$

Support vector machine works by upper-bounding the loss $\delta(\hat{y} \neq y)$ by a surrogate hinge loss

$$\begin{aligned} \delta(\hat{y} \neq y) &= \delta(\hat{y} \neq y) - \boldsymbol{w}^y \cdot \boldsymbol{x} + \boldsymbol{w}^y \cdot \boldsymbol{x} && (5) \\ &\leq \delta(\hat{y} \neq y) - \boldsymbol{w}^y \cdot \boldsymbol{x} + \max_{\hat{y}} \left[\boldsymbol{w}^{\hat{y}} \cdot \boldsymbol{x} \right] && (6) \\ &\leq \max_{\hat{y}'} \left[\delta(\hat{y}' \neq y) - \boldsymbol{w}^y \cdot \boldsymbol{x} + \boldsymbol{w}^{\hat{y}'} \cdot \boldsymbol{x} \right]. && (7) \end{aligned}$$

The multiclass SVM optimization problem:

$$\{\boldsymbol{w}^r\} = \operatorname*{argmin}_{\{\boldsymbol{w}^r\}} \frac{1}{m} \sum_{i=1}^{m} \max_{\hat{y}'} \left[\delta(\hat{y}' \neq y_i) - \boldsymbol{w}^{y_i} \cdot \boldsymbol{x}_i + \boldsymbol{w}^{\hat{y}'} \cdot \boldsymbol{x}_i \right] + \frac{\lambda}{2} \sum_{r=1}^{k} \|\boldsymbol{w}^r\|^2$$

Gradient descent will result with the following algorithm:

$$\hat{y}' = \operatorname*{argmax}_{y \in \{1,\ldots,k\}} \left[\delta(\hat{y}' \neq y) + \boldsymbol{w}^{\hat{y}'} \cdot \boldsymbol{x} \right]$$

$$\text{if } \left[\delta(\hat{y}' \neq y) - \boldsymbol{w}^y \cdot \boldsymbol{x} + \boldsymbol{w}^{\hat{y}'} \cdot \boldsymbol{x} \right] > 0 :$$

$$\begin{aligned} &\boldsymbol{w}^{y_i} \leftarrow (1 - \lambda\eta_t)\boldsymbol{w}^{y_i} + \eta_t \boldsymbol{x}_i \\ &\boldsymbol{w}^{\hat{y}'} \leftarrow (1 - \lambda\eta_t)\boldsymbol{w}^{\hat{y}'} - \eta_t \boldsymbol{x}_i \\ &\boldsymbol{w}^{y} \leftarrow (1 - \lambda\eta_t)\boldsymbol{w}^{y} \qquad \text{for } y \neq \hat{y}', y_i \end{aligned}$$

3 Which method is better for your application?

A full theoretical comparison of all the methods is given in Daniely et al. (2012). The complexity of OA, multiclass Perceptron and multiclass SVM is the same and equals dk. The complexity of ECOC with a code of length l and code-distance δ is at most $\tilde{O}(dl)$ and at least $d\delta/2$. The complexity of AP is dk^2. The hypothesis class of multiclass Perceptron and multiclass SVM essentially contains the hypothesis classes of OA, hence might result with better performance. The hypothesis of AP contains the hypothesis class of multiclass Perceptron and multiclass SVM and hence also that of OA.

Example. assume that there are $k = 3$ classes, and they are 3 subsets sectors of equal angle of the unit circle (see Figure 2). Clearly, by taking $\boldsymbol{w}^r$ to point to the middle of each sector (dashed arrows in the illustration), we get zero error. In contrast, no linear separator can split the three labels into two groups without error.

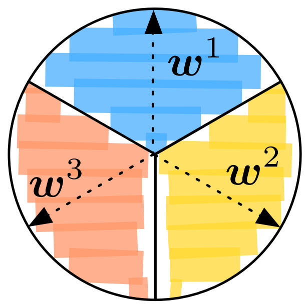


Figure 2: Multiclass SVM hypothesis class is reach enough to separate those three classes.

References

E. L. Allwein, R.E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

A. Daniely, S. Sabato, and S. Shalev-Schwartz. Multiclass learning approaches: A theoretical comparison with implications. In *Advances in Neural Information Processing Systems (NIPS) 25*, 2012.