

# Lab Notes: Multiclass Review and an Approach to Ranking

*David S. Rosenberg*

Let's review the multiclass classification framework, and then attempt to use the framework we developed for multiclass classification to address the ranking problem.

## 1 Multiclass Review

- **General [Discrete] Output Space:**  $\mathcal{Y}$  (e.g.  $\mathcal{Y} = \{1, \dots, k\}$  for multiclass)
- **Base Hypothesis Space:**  $\mathcal{H} = \{h : \mathcal{X} \rightarrow \mathbf{R}\}$  ("score functions").
- **Multiclass Hypothesis Space** (for  $k$  classes):

$$\mathcal{F} = \left\{ x \mapsto \arg \max_i h_i(x) \mid h_1, \dots, h_k \in \mathcal{H} \right\}.$$

In words, we have one score function for each class, and we predict the class that has the highest score.

- How to fit the  $h$ 's?
  - One approach is one-vs-all.
  - But we can also train them jointly

We get the example  $(x_i, y_i)$  correct, if

$$h_{y_i}(x_i) > h_j(x_i)$$

for all  $j \neq y_i$ . In words, the score function for the correct class  $y_i$  is higher than the score functions for the incorrect classes. Equivalently,

$$h_{y_i}(x_i) > \max_{j \neq y_i} h_j(x_i).$$

Also equivalently, we want:

$$h_{y_i}(x_i) - \max_{j \neq y_i} h_j(x_i) > 0.$$

This should remind us of margin, which is positive iff our prediction has the right sign. Let  $\ell$  be a margin-based loss (e.g. hinge loss or logistic loss), and try to minimize

$$\frac{1}{n} \sum_{i=1}^n \ell \left( h_{y_i}(x_i) - \max_{j \neq y_i} h_j(x_i) \right)$$

by searching over  $h_1, \dots, h_k \in \mathcal{H}$ . This is a reasonable first approach to multiclass classification.

## 2 Reformulation with Compatibility Scores

The idea here is that, rather than having  $k$  score functions for  $k$  classes, let's have a single function that takes an input  $x$  and a class  $y$  as input, and produces a score measuring how “compatible” the class  $y$  is with the input  $x$ . We will call this a **compatibility score function**

$$h(x, y) \mapsto \mathbf{R}.$$

(NOTE: In  $h(x, y)$ , the second argument  $y$  takes discrete values in  $\{1, \dots, k\}$ .)

The compatibility score approach absorbs the separate score function approach as a special case: Suppose we had  $h_1, \dots, h_k \in \mathcal{H}$  from the previous approach, we can construct an equivalent compatibility function by defining:

$$h(x, y) = h_y(x),$$

for  $y \in \{1, \dots, k\}$ . So then

- $h(x, y)$  classifies  $(x_i, y_i)$  correctly iff

$$h(x_i, y_i) > h(x_i, y) \forall y \neq y_i$$

- Equivalently, if we define

$$m_i = h(x_i, y_i) - \max_{y \neq y_i} h(x_i, y),$$

then classification is correct if  $m_i > 0$ . Generally want  $m_i$  to be large.

### 3 Consider ranking problem

1. For a search query, retrieve 100 results that are keyword matches. (Or use some other fast approach with high recall.)
2. Let  $x$  represent the search query, any information about the user, user browser, user browsing history, etc.
3. Let  $y$  represent a particular webpage. (or item, etc.)
4. Use a compatibility score function  $h(x, y)$  to rank the 100 pages, by plugging each of them in (as different  $y$ 's with  $x$  remaining the same) to  $h(x, y)$  to get 100 scores.

For a given user/query  $x$ , we need to evaluate  $h(x, y)$  for each of the 100 pages we've identified as possibly related:

$$\begin{aligned} h(x = \text{User 104232 with Query } q, y = \text{mathwords.com}) \\ h(x = \text{User 104232 with Query } q, y = \text{mathematica.com}) \\ h(x = \text{User 104232 with Query } q, y = \text{mathcad.com}) \end{aligned}$$

To create the compatibility function  $h(x, y)$ , we will use features that depend jointly on  $x$  and  $y$ , such as:

1. A binary indicator feature for whether or not this user has clicked on page  $y$  before.
2. Page  $y$  has a topic category that's related to query  $q$ .

$h(x, y)$  could be a linear or nonlinear function of these features.

A labeled example would be a triple:

$$(x_i, \{y_{i1}, \dots, y_{i5}\}, y_i^*),$$

where  $y_{i1}, \dots, y_{i5}$  were 5 results shown to the user, and  $y_i^*$  is the result clicked. So loss for this example could be something like

$$\begin{aligned} L(x_i, \{y_{i1}, \dots, y_{i5}\}, y_i^*) &= \ell(m_i) \\ &= \ell \left( h(x_i, y_i^*) - \max_{y \in \{y_{i1}, \dots, y_{i5}\} - \{y_i^*\}} h(x_i, y) \right), \end{aligned}$$

for some margin-based loss function  $\ell$ , such as hinge loss.

NOTE: In practice the approach described above is naive and should be adjusted. One major issue is that the ordering of the search results  $y_{i1}, \dots, y_{yi5}$  has a large influence on which is most likely to be clicked. Burges's paper *From RankNet to LambdaRank to LambdaMART: An Overview* is a good reference for more practical models.