

Reductions

Want to turn an algorithm for solving a simple problem into an algorithm for solving a more complex problem.

Simple example: multiclass classification with a binary classifier. Several straightforward approaches (OVA/OVR and AVA/OVO/AP) and several non-straightforward approaches (ECOC, SECOC, etc.).

The general requirements of a reduction are:

1. A mapping from examples for the hard problem to examples for the easy problem.
2. A method for solving the hard problem based on a solution to the easy problem.
3. An error bound that guarantees that good performance on the easy problem implies good performance on the hard problem.

One of the easiest reductions is from multiclass classification to binary classification using the “one versus all” approach.

In multiclass classification, we are given inputs $x \in \mathcal{X}$ and outputs $y \in \{1, 2, 3, \dots, K\}$. The multiclass classification loss for a hypothesis h is $\mathbf{1}[h(x) \neq y]$, where $\mathbf{1}[\cdot]$ is an indicator function that takes value 1 when \cdot is true and value 0 otherwise. In the OVA reduction, we train K -many binary classifiers h_1, \dots, h_K . Classifier h_k attempts to distinguish between class k and *not* class k . In particular, the reduction works as:

1. Map a multiclass example (x, y) to K -many binary examples, one for each classifier h_k . Classifier h_k is fed the binary pair $(x, y = k)$.
2. Given binary classifiers h_1, \dots, h_K and a new test point \hat{x} , compute $y_1 = h_1(\hat{x}), \dots, y_K = h_K(\hat{x})$. Choose at random any of the “positive” y s. (If there are none, choose at random.)
3. It turns out that the loss on the multiclass problem is at most $4\sqrt{K}$ times the loss on the binary problems. This is pretty bad.

We can do better by using the all-versus-all reduction. Here, we train $\binom{K}{2}$ classifiers, where classifier h_{ij} (for $1 \leq i < j \leq K$) distinguishes only between classes i and j . The reason the error bound for OVA was so bad is because just one classifier screwing up can completely change our decisions. AVA has the advantage that this is no longer true. The reduction works as follows:

1. Map a multiclass example (x, y) to K -many binary examples. In particular, for each class pair $i < j$, we feed $(x, +1)$ to h_{ij} if $i = y$; we feed $(x, -1)$ to h_{ij} if $j = y$; and we ignore classifier h_{ij} if neither $i = y$ nor $j = y$.
2. Given binary classifiers h_{ij} and a new test point \hat{x} , maintain a *score vector* S of length K . Compute $h_{ij}(\hat{x})$ for every pair i, j . If h_{ij} returns “true” (i.e., it believes the correct class is i), then increment S_i . If it returns “false”, then increment S_j . Return the class with the highest score (breaking ties randomly).
3. The error bound is much better here. The loss on the multiclass problem is at most *twice* the average loss on the binary problems. This is quite good (i.e., there is no dependence on K).

There are other methods for doing multiclass-to-binary reductions:

1. Build a classification tree. Gets an error bound of that depends on $\log_2 K$.
2. Use an error correcting output code (such as a Hadamard code). If the codewords have distance d apart, then we get an error bound that depends on $2K/d$. Thus the Hadamard code gets an error bound of 4 because $d = K/2$.
3. Many more options...

There are also other problems we can solve with reductions:

1. Weighted binary classification to binary classification
2. Regression to binary classification
3. Binary classification to regression
4. Original regression
5. Ranking
6. Structured prediction
7. Reinforcement learning
8. Etc...

For a fairly substantial bibliography, see <http://hunch.net/~jl/projects/reductions/reductions.html>.