Large Margin DAGs for Multiclass Classification

Abstract

$N(N-1)/2$

We present a new learning architecture: the Decision Directed Acyclic Graph (DDAG), which is used to combine many two-class classifiers into a multiclass classifiers. For an N -class problem, the DDAG contains $N(N-1)/2$ classifiers, one for each pair of classes. We present a VC analysis of the case when the node classifiers are hyperplanes; the resulting bound on the test error depends on N and on the margin achieved at the nodes, but not on the dimension of the space. This motivates an algorithm, DAGSVM, which operates in a kernel-induced feature space and uses two-class maximal margin hyperplanes at each decision-node of the DDAG. The DAGSVM is substantially faster to train and evaluate than either the standard algorithm or Max Wins, while maintaining comparable accuracy to both of these algorithms.

1 Introduction

N

The problem of multiclass classification, especially for systems like SVMs, doesn't present an easy solution. It is generally simpler to construct classifier theory and algorithms for two mutually-exclusive classes than for N mutually-exclusive classes. We believe constructing N-class SVMs is still an unsolved research problem.

$N - v - r$

The standard method for N-class SVMs is to construct SVMs. The ith SVM will be trained with all of the examples in the ith class with positive labels, and all other examples with negative labels. We refer to SVMs trained in this way as $1-v-r$ SVMs (short for one-versus-rest). The final output of the N $1-v-r$ SVMs is the class that corresponds to the SVM with the highest output value. Unfortunately, there is no bound on the generalization error for the $1-v-r$ SVM, and the training time of the standard method scales linearly with N.

$1 - v - (N-1)/2$

Another method for constructing N-class classifiers from SVMs is derived from previous research into combining two-class classifiers. Knerr suggested constructing all possible two-class classifiers from a training set of N classes, each classifier being trained on only two out of N classes. There would thus be $K = N(N-1)/2$ classifiers. When applied to SVMs, we refer to this as $1-v-1$ SVMs (short for one-versus-one).

Knerr suggested combining these two-class classifiers with an “AND” gate. Friedman

$1-v-1$
suggested a Max Wins algorithm: each $1-v-1$ classifier casts one vote for its preferred class, and the final result is the class with the most votes. Friedman shows circumstances in which this algorithm is Bayes optimal. KreBel applies the Max Wins algorithm to Support Vector Machines with excellent results.
$N-v-1$
A significant disadvantage of the $1-v-1$ approach, however, is that, unless the individual classifiers are carefully regularized (as in SVMs), the overall N-class classifier system will tend to overfit. The “AND” combination method and the Max Wins combination method do not have bounds on the generalization error. Finally, the size of the $1-v-1$ classifier may grow superlinearly with N, and hence, may be slow to evaluate on large problems.

2 Decision DAGs

2
A Directed Acyclic Graph (DAG) is a graph whose edges have an orientation and no cycles. A Rooted DAG has a unique node such that it is the only node which has no arcs pointing into it. A Rooted Binary DAG has nodes which have either 0 or 2 arcs leaving them. We will use Rooted Binary DAGs in order to define a class of functions to be used in classification tasks. The class of functions computed by Rooted Binary DAGs is formally defined as follows.
$(j+1)'(N-1)/2.1\}\}$
Definition 1 Decision DAGs (DDAGs). Given a space X and a set of boolean functions $\mathcal{F}=\{f:X\to\{0,1\}\}$, the class $DDAG(\mathcal{F})$ of Decision DAGs on N classes over $\mathcal{F}$ are functions which can be implemented using a rooted binary DAGs with N leaves labeled by the classes where each of the $K=N(N-1)/2$ internal nodes is labeled with an element of $\mathcal{F}$. The nodes are arranged in a triangle with the single root node at the top, two nodes in the second layer and so on until the final layer of N leaves. The i-th node in layer $j<N$ is connected to the i-th and $(i+1)$-st node in the $(j+1)$-st layer.
To evaluate a particular DDAG G on input $x\in X$, starting at the root node, the binary function at node is evaluated. The node is then exited via the left edge, if the binary function is zero; or the right edge, if the binary function is one. The next node's binary function is then evaluated. The value of the decision function $D(x)$ is the value associated with the final leaf node. The path taken through the DDAG is known as the *evaluation path*. The input x reaches a node of the graph, if that node is on the

evaluation path for x. We refer to the decision node distinguishing classes i and j as the ij-node. Assuming that the number of a leaf is its class, this node is the i-th node in the $(N-j+i)$-th layer provided . Similarly the j-nodes are those nodes involving class , that is, the internal nodes on the two diagonals containing the leaf labeled by j.

The DDAG is equivalent to operating on a list, where each node eliminates one class from the list. The list is initialized with a list of all classes. A test point is evaluated against the decision node that corresponds to the first and last elements of the list. If the node prefers one of the two classes, the other class is eliminated from the list, and the DDAG proceeds to test the first and last elements of the new list. The DDAG terminates when only one class remains in the list. Thus, for a problem with N classes, $N-1$ decision nodes will be evaluated in order to derive an answer.

The current state of the list is the total state of the system. Therefore, since a list state is reachable in more than one possible path through the system, the decision graph the algorithm traverses is a DAG, not simply a tree.

Decision DAGs naturally generalize the class of Decision Trees, allowing for a more efficient representation of redundancies and repetitions that can occur in different branches of the tree, by allowing the merging of different decision paths. The class of functions implemented is the same as that of Generalized Decision Trees, but this particular representation presents both computational and learning-theoretical advantages.

3 Analysis of Generalization

In this paper we study DDAGs where the node-classifiers are hyperplanes. We define a Perceptron DDAG to be a DDAG with a perceptron at every node. Let w be the (unit) weight vector correctly splitting the i and j classes at the ij-node with threshold θ. We define the margin of the ij-node to be $\gamma = \min_{c(x)=i,j} \{|\langle w, x\rangle - \theta|\}$, where $c(x)$ is the class associated to training example x. Note that, in this definition, we only take into account examples with class labels equal to i or j.

Theorem 1 Suppose we are able to classify a random m sample of labeled examples using a Perceptron DDAG on N classes containing K decision nodes with margins γ_i at node i, then we can bound the generalization error with probability greater than $1-\delta$ to be less than

$$\frac{130R^2}{m}\Big(D'\log(4em)\log(4m)+\log\frac{2(2m)^K}{\delta}\Big),$$

where $D' = \sum_{i=1}^{K}\frac{1}{\gamma_i^2}$ and R is the radius of a ball containing the distribution's support.

Theorem 1 implies that we can control the capacity of DDAGs by enlarging their margin. Note that, in some situations, this bound may be pessimistic: the DDAG partitions the input space into polytopic regions, each of which is mapped to a leaf node and assigned to a specific class. Intuitively, the only margins that should matter are the ones relative to the boundaries of the cell where a given training point is assigned, whereas the bound in Theorem 1 depends on all the margins in the graph.

By the above observations, we would expect that a DDAG whose j-node margins are large would be accurate at identifying class j, even when other nodes do not have large margins. Theorem 2 substantiates this by showing that the appropriate bound depends only on the j-node margins, but first we introduce the notation,

$\epsilon_j(G) = P\{x : \text{(x in class j and x is misclassified by G) or x is misclassified as class j by G}\}$

.

Theorem 2 Suppose we are able to correctly distinguish class j from the other classes in a random m-sample with a DDAG G over N classes containing K decision nodes with margins γ_i at node i, then with probability $1-\delta$,

$$\epsilon_j(G) \le \frac{130R^2}{m}\Big(D'\log(4em)\log(4m)+\log\frac{2(2m)^{N-1}}{\delta}\Big),$$

$j < j\; j + i)$

where $D' = \sum_{i\in j-nodes}\frac{1}{\gamma_i^2}$, and R is the radius of a ball containing the support of the distribution.