CS 446 Machine Learning Fall 2016 Oct 27, 2016

Multiclass Classification

Professor: Dan Roth *Scribe: C. Cheng*

Overview

- Binary to multiclass
- Multiclass SVM
- Constraint classification

1 Introduction

Multiclass classification, for some aspects, is very simple. There are some interesting issues in multiclass classification that are the stepping stone to more advanced topics in machine learning, such as structure predictions. So far we have been talking about binary classification. We talked about algorithms for linear models, such as Perceptron, Winnow, SVM, etc. The prediction is simple. Given an example x, the output is a single bit predicted by the sign of $w^T x$, where w is the learned linear model.

In many models, more expressive outputs are concerned, such as the multiclass classification task.

2 Task setting

Given a data set $D = \{x_i, y_i\}_1^m$, where $x_i \in \mathbb{R}^n$, $y_i \in \{1, 2, ..., k\}$, the task of multiclass classification is to learn a model that outputs a single class label y given an example x.

3 Binary to multiclass

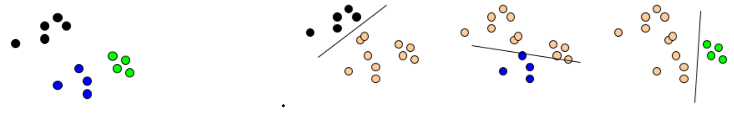
Using a binary classifier as a black box, can we reduce the task of multiclass classification to the binary case? There are multiple ways to decompose the multiclass prediction into multiple binary decisions.

3.1 One-versus-all

Let's assume that our black-box algorithm is a linear classifier, and each class can be separated from all the rest labels. If we do this, we are basically decomposing the task to learning k independent binary classifiers, and we know how to do this.

Formally, let D be the set of training examples. For any label I, take elements of D with label I as positive examples and all other elements of D as negative examples. Then, construct a binary classification problem for I. This is a binary learning problem that we can solve. Since there are k possible labels, we are producing k binary classifiers w_1, w_2,...,w_k. Once we have learned the k classifiers, the decision is made by winner takes all (WTA) strategy, such that $f(x) = \text{argmax}_i w_i^T x$. The "score" $w_i^T x$ (the sigmoid function can be applied on it) can be thought of as the probability that x has label i.

Graphically, consider the data set with three color labels shown in the figure below. Using binary classifiers, we separated the black from the rest, blue from


(a) Data set with 3 labels (b) Decomposed into binary problems

the rest, and green from the rest. Easy.

The only caveat is that when some points with a certain label are not linearly separable from the other, like shown in the figure below, this scheme cannot be used. Basically, we are concerned about the expressivity of this paradigm. It is

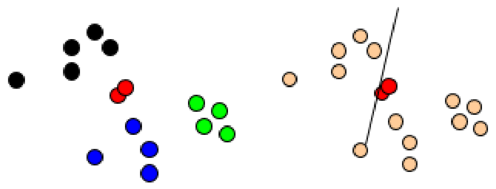


Figure 2: Red points are not linearly separable from other points

not always possible to learn, because it is not always separable in the way we want. Even though it works well and is the commonly used method, there is no theoretical justification for it.

3.2 All-versus-all

Assume that there is a separation between every pair of classes using a binary classifier in the hypothesis space. Then, we can look at all pairs of labels ($\binom{k}{2}$ of them), and for each pair, define a binary learning problem. In each case, for pair (i, j), the positive examples are all examples with label i, and negative examples are those with label j. Now instead of k classifiers as in OvA, we have $\binom{k}{2}$ classifiers.

In this case each label gets k_1 votes, and the decision is more involved, because output of binary classifiers may not cohere. To make a decision, an option is to classify example x to take label i if i wins on x more often than any $j = 1, ..., k$. Alternatively, we can do a tournament. Starting with $n/2$ pairs, continue with the winners and go down iteratively. Either way, there are potential issues.

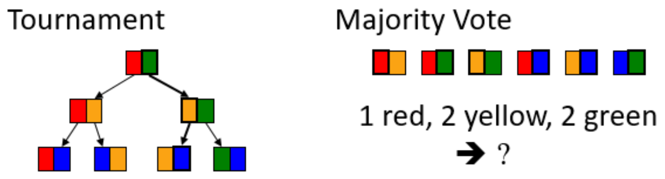


Figure 3: Tournament and majority vote

However, overall, it is a good method.

3.3 One-versus-all VS all-versus-all

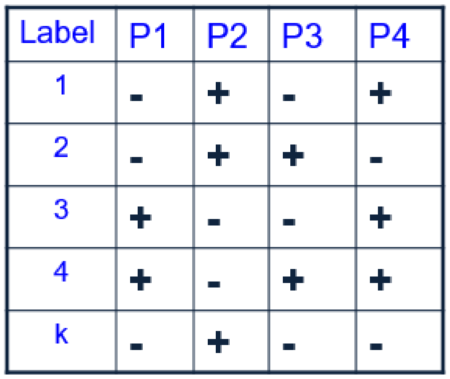
There is a trade-off between OvA and AvA paradigms. Computationally, to apply AvA, a quadratic number of classifiers have to be trained, while OvA requires linear number of classifiers. AvA has more expressivity, but less examples to learn from. In terms of number of examples, AvA has smaller learning problems. And it makes AvA preferable when ran in dual.

4 Error Correcting Codes Decomposition

Other than one-versus-all and all-versus-all, there is another way of thinking about multiclass classification. It is the paradigm called error correcting codes decomposition.

In 1-vs-all, we use k classifiers for k labels. However, we don't really need k classifiers to make k distinctions. Instead of learning k classifiers, we can represent k with $\log_2 k$ binary bits and learn each bit separately. In this way, the number of classifiers is reduced to $\log_2 k$.

The error correcting codes decomposition is basically a relaxed implementation of this idea, because with the scheme just suggested, a wrong label is obtained even with a single incorrectly learned bit. Instead, we will use a few more bits. In the case of $k = 8$, it is sufficient to learn three classifiers, but we will learn a few more. As shown in the table below, we are using four bits.



Label	P1	P2	P3	P4
1	-	+	-	+
2	-	+	+	-
3	+	-	-	+
4	+	-	+	+
k	-	+	-	-

Figure 4: Error correcting codes decomposition

Each row is an encoding of each class. For example, label 1 has the encoding -+-+. The columns are dichotomies of the data, each corresponds to a new classification problem. For example, the column P1 represents a binary classification problem that says + if the label is 3 or 4, and says - otherwise. Notice that with four bits, more labels (16) can be encoded. However, we are choosing to encode only eight, and that gives us a little redundancy. If mistake is made in one of the bits, choose the label that is most consistent with the results. It potentially corrects some of the mistakes made in the classification, and that is why the paradigm is called error correcting codes decomposition.

In the case of 1-vs-all, there are exactly k rows and k columns. Each problem has one +, and all the rest are -'s.

Clearly, we gained the advantage of learning less classifiers. However, the disadvantage is that the dichotomies Pi's are essentially random splits of the labels, and might not be all linearly separable. It is not clear that the Pi's are learnable.

4.1 Problems with decompositions

So far we talked about three decompositions of the label space, namely, one-vs-all, all-vs-all, and error correcting codes. In general, the ability to decompose things dictates how good things are.

In principle, one-vs-all is a good method, but the expressivity is an issue, because we cannot necessarily separate each label from the union of all the others.

Another problem with multiclass classification is that the way we presented here is not scale to very large cases. If there is a case that is exponential in the dimension, one-vs-all or all-vs-all are impossible to do.

And the question is, can we move to other methods that are more global and do not suffer these decomposition issues?

5 Constraint classification

5.1 1-vs-all: learning architecture

As a way to develop our intuition, consider the learning architecture of the one-vs-all decomposition.

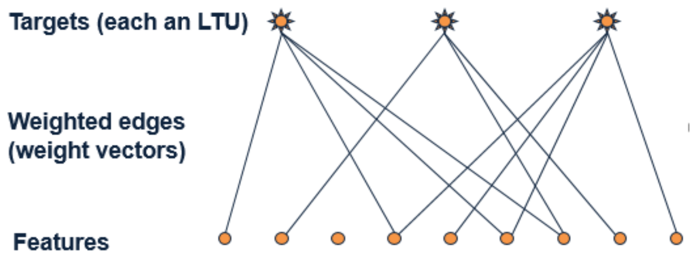


Figure 5: One-vs-all learning architecture

The circles at the bottom are the n input features, and the stars on the top are the k labels. The nk wires represent the weights. The evaluation is done using winner-takes-all strategy. Each set of n weights, corresponding to the ith label, is trained independently given its performance on example x, and independently of the performance of label j on x. Hence, learning is local. Only the final decision is global, because we are doing winner-takes-all.

However, with the same architecture, we can entertain other learning algorithms. We could change the way to learn the weights, while keeping the same architecture. Indeed, we have seen other learning algorithms built on this WTA architecture.

For example, we know that Winnow only learns positive weights, therefore, only learns monotone Boolean functions. We have to extend Winnow in order to learn general Boolean functions. One way to do it is using "balanced Winnow", in which each variable has two weights, w^+ and w^-. Prediction is made by evaluating the "effective weight", i.e., the difference between w^+ and w^-. If a mistake is made, such that,

$$[(w^+ - w^-) \cdot x \geq \theta] \neq y$$

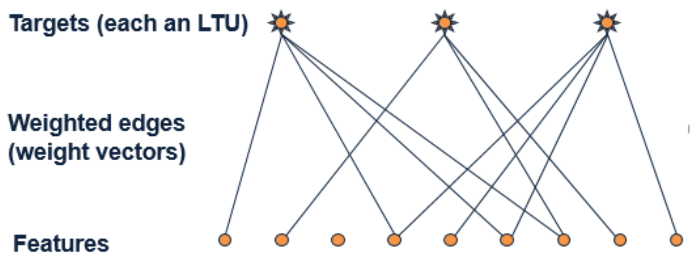


Figure 6: Balanced Winnow

both weight vectors are updated,

$$w_i^+ \leftarrow w_i^+ r^{yx_i} \qquad \text{and} \qquad w_i^- \leftarrow w_i^- r^{-yx_i}$$

We can think of w^+ and w^- as the weight vectors of two labels, positive and negative. The learning rule is basically asking whether $w^+ \cdot x$ is larger than $w^- \cdot x$ or not. Can this be generalized to the case of more than two labels? We need a "global" learning approach.

6 Multiclass SVM

6.1 Multiclass margin

The key idea of SVM is based on the notion of margin. Recall that for a binary classifier, the margin of a hyperplane for a dataset is the distance between the hyperplane and the data point nearest to it. Equivalently, you can think of margin as the smallest distance between a positive example and a negative example.

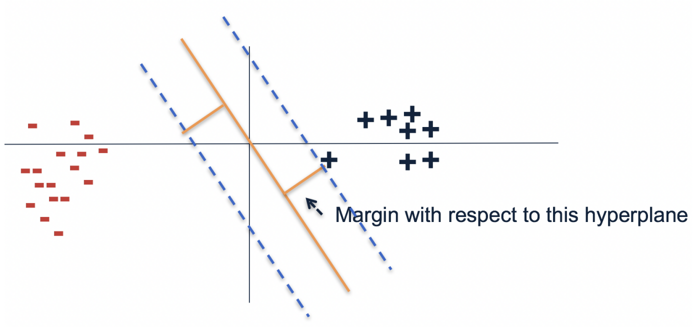


Figure 7: Margin for binary classifiers

If we want to generalize SVM to multiclass case, we have to generalize the notion of margin in the multiclass scenario. Multiclass margin is defined as the score difference between the highest scoring label and the second one. Here the scores are calculated by taking the dot product of the weight vector **w** for each label and **x**. Then, we discard labels with low scores and look at the gap between the top two.

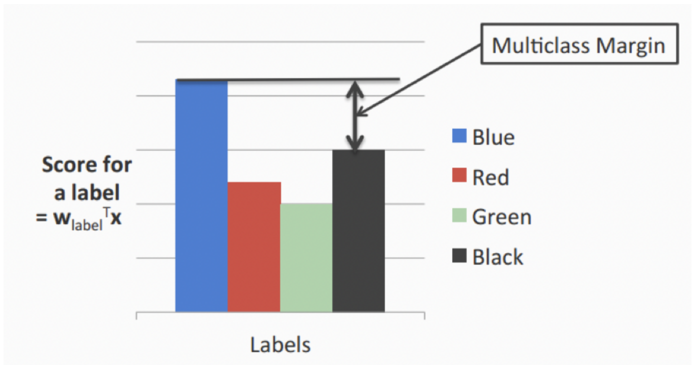


Figure 8: Multiclass margin

6.2 Intuition

In the binary SVM algorithm, our goal is to maximize the margin. Or equivalently, minimize the norm of weights such that the closest points to the hyperplane have scores ± 1.

It took people over twenty years to generalize the SVM algorithm to multiclass case. In the multiclass SVM algorithm, each label has a different weight vector, like one-vs-all, and the goal is to maximize multiclass margin. Or equivalently, minimize the total norm of the weights such that the true label is scored at least 1 more than the second best one.

6.3 Multiclass SVM in the separable case

Recall that in the case of hard binary SVM, the optimization problem is formalized as

$$\begin{aligned} \max_{\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} \\ \text{s.t.} \quad & y_i \mathbf{w}\mathbf{x}_i \geq 1 \end{aligned}$$

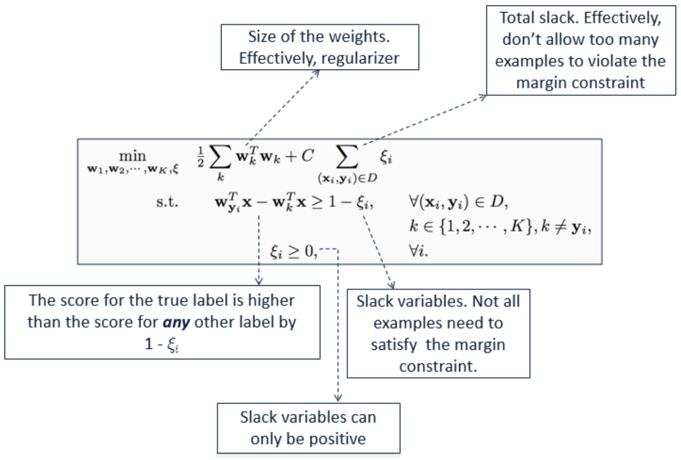
The goal is to minimize the norm of weights subject to a set of constraints (one for each example).
Generalizing it to the multiclass case, we need to minimize the sum of L2-norms of all the weight vectors, subject to the constraints that the score for the true

label is higher than the score for any other label by at least 1. The optimization task can be written as

$$\begin{aligned} \max_{\mathbf{w}_1,\mathbf{w}_2,\ldots,\mathbf{w}_K} \quad & \frac{1}{2}\sum_k \mathbf{w}_k^T\mathbf{w}_k \\ \text{s.t.} \quad & \mathbf{w}_{y_i}^T\mathbf{x}\mathbf{w}_k^T\mathbf{x} \geq 1 \qquad\qquad \forall(\mathbf{x}_i, y_i) \in D, \\ & \qquad\qquad k \in (1, 2, \cdots, K), k \neq y_i \end{aligned}$$

6.4 Multiclass SVM in general case

To apply the multiclass SVM algorithm to general cases, we start with the separable case and add slack variables to the optimization problems. Not all examples need to satisfy the margin constraint.



Size of the weights. Effectively, regularizer
Total slack. Effectively, don't allow too many examples to violate the margin constraint
$\min_{\mathbf{w}_1,\mathbf{w}_2,\cdots,\mathbf{w}_K,\xi} \frac{1}{2}\sum_k \mathbf{w}_k^T\mathbf{w}_k + C\sum_{(\mathbf{x}_i,\mathbf{y}_i)\in D}\xi_i$
s.t. $\mathbf{w}_{\mathbf{y}_i}^T\mathbf{x} - \mathbf{w}_k^T\mathbf{x} \geq 1 - \xi_i, \quad \forall(\mathbf{x}_i,\mathbf{y}_i)\in D,$
$k \in \{1,2,\cdots,K\}, k \neq \mathbf{y}_i,$
$\xi_i \geq 0, \quad \forall i.$
The score for the true label is higher than the score for ***any*** other label by 1 - ξ_i
Slack variables. Not all examples need to satisfy the margin constraint.
Slack variables can only be positive

Figure 9: Multiclass SVM: general case

The constraints ensure that the difference between the scores of any two labels is at least $1 - \xi_i$. The slack variable ξ_i is per example.

6.5 Summary

The multiclass SVM algorithm is generalized from binary SVM. It comes with similar generalization guarantees as the binary SVM, because of the VC dimension idea.
With K labels and inputs in $\mathbb{R}^n$, we have nK weights in all, same as one-vs-all. What has changed is just how we train the weight vectors.
Training optimizes the "global" SVM objective. It can be trained using different optimization methods, essentially with stochastic sub-gradient descent.

Prediction is made with winner-takes-all (WTA) strategy.
The key question is why it works. Or, why is this the natural definition of multiclass margin? A theoretical justification, along with extensions to other algorithms beyond SVM is given by Constraint Classification. It not only applies to multiclass classification problems, but rather, to multi-label problems, ranking problems, etc.

7 Constraint classification

7.1 Intuition

The idea of the constraint classification is as follows. The examples we give the learner are pairs $(\mathbf{x}, y)$, $y \in \{1, \cdots, k\}$. The black box learner (e.g., one vs. all) we described might be thought of as a function of $\mathbf{x}$ only but, actually, we made use of the labels y. The "black box learner" should also be a function of y. How is y being used here? Think about learning one vs. all. The way we define the one vs. all problems takes y into consideration. y decides what to do with the example $\mathbf{x}$. That is, which of the k classifiers should take the example as a positive example, making it a negative to all the others.
How do we make predictions in these cases? We look at $f_y(\mathbf{x}) = \mathbf{w}_y^T\mathbf{x}$ for each label. Then, we predict using the winner-takes-all strategy, such that $y^* = \text{argmax}_{y=1,k} f_y(\mathbf{x})$.
Equivalently, we can say that we predict as follows: Predict y iff

$$\forall y' \in \{1, \cdots, k\}, y' \neq y \qquad (\mathbf{w}_y^T\mathbf{x} - \mathbf{w}_y'^T\mathbf{x}) \geq 0$$

such that label y has to beat all the other labels in order to win.
So far, we only talked about predictions, but did not say how we learn the k weight vectors $\mathbf{w}_y (y = 1, \cdots, k)$. A lot of times in learning, what we do in tests is similar with what happened in training. This is the key principle of learning. So can we train in a way that better fits the way we predict?

7.2 Linear separability for multiclass

We are learning k n-dimensional weight vectors, so we can concatenate the k weight vectors into

$$\mathbf{w} = (w_1, w_2, \cdots, w_k) \in \mathbb{R}^{nk}$$

Now we are working in an nk-dimensional space. In order to deal with the concatenation of the weight vectors, we need to have examples in the same space. Otherwise, we will not be able to compute dot products.
Therefore, we will represent each example $(\mathbf{x}, y)$, as an nk-dimensional vector, $\mathbf{x}_y$, with $\mathbf{x}$ embedded in the yth part of it $(y = 1, 2, \cdots, k)$ and the other

coordinates are 0's. For example, take $k = 4$ and $y = 2$, $\mathbf{x}_y = (0, \mathbf{x}, 0, 0) \in \mathbb{R}^{kn}$. This construction is called Kesler Construction, which was invented in the 1970s and was really used after Zimak re-invented it.
Now we can understand the n-dimensional decision rule, such that, predict y iff

$$\forall y' \in \{1, \cdots, k\}, y' \neq y \quad (\mathbf{w}_y^T - \mathbf{w}_y'^T) \cdot \mathbf{x} \geq 0$$

Equivalently, writing it in the nk-dimensional space, we predict y iff

$$\forall y' \in \{1, \cdots, k\}, y' \neq y \quad \mathbf{w}^T \cdot (\mathbf{x}_y - \mathbf{x}_{y'}) \equiv \mathbf{w}^T \cdot \mathbf{x}_{yy'} \geq 0$$

Here, $\mathbf{x}_{yy'}$ is a vector in the nk-dimensional space. It denotes the difference between $\mathbf{x}_y$ and $\mathbf{x}_{y'}$. Two chunks of $\mathbf{x}_{yy'}$ are nonzero. In the y chunk, we have $\mathbf{x}$. In the y' chunk, we have $-\mathbf{x}$. The rest coordinates are all zeros. And the prediction we make in the n-dimensional space should be consistent with the prediction in the nk-dimensional space. Once we understand this, we have a new paradigm for doing multiclass classification.
If we take all the $\mathbf{x}_{yy'}$ examples as positive, and the negations of them as negative. Then, the set $(\mathbf{x}_{yy'}, +) \equiv (\mathbf{x}_y - \mathbf{x}_{y'}, +)$ is linearly separable from the set $(-\mathbf{x}_{yy'}, -)$ using the linear separator $\mathbf{w} \in \mathbb{R}^{kn}$.
We managed to show that if we have a multiclass classification problem, all we need is that each label is separable from any other label. Then, in the nk-dimensional space, it is separable. Which means for now, we can reduce all the problems we have to a linearly separable case, or a binary problem, in which we have good notion of margin and thus can be solved easily. Actually, we solved the voroni diagram challenge.
Note that this is just a representational trick. We did not say how to learn the weight vectors. We are really going to run the algorithm in n-dimensional space.

7.3 Training and prediction

We first explain via Keslers construction; then show we dont need it.
Given a data set $\{(\mathbf{x}, y)\}$ with m examples in the n-dimensional space, and $y \in \{1, 2, \cdots, k\}$. We create a binary classification task by embedding $\mathbf{x}$'s. If y is the correct label, generate positive examples of the form $(\mathbf{x}_y - \mathbf{x}_{y'}, +)$ and negative examples of the form $(\mathbf{x}_{y'} - \mathbf{x}_y, -)$. Here $\mathbf{x}_y \in \mathbb{R}^{kn}$. Do this for all $y \neq y$. For each example, $k-1$ positive examples and $k-1$ negative examples are generated. Therefore, there are $2m(k-1)$ examples in total. We are actually blowing up the number of examples. Then, use your favorite linear learning algorithm to train a binary classifier.
Given an nk-dimensional weight vector $\mathbf{w}$, which is a concatenation of k n-dimensional vectors, and a new example $\mathbf{x}$, predict $y^* = \text{argmax}_y \mathbf{w}^T \mathbf{x}^y$.

7.4 Details: Kesler construction and multi-class separability

Assume we have four colors in a 2-D space, as shown in the figure below. Blue has label 2, and its score has to be greater than 1, 3, and 4. We map it in the following way. For each example in the 2-D space, there is a set of $k-1$ examples in the transformed space. For the blue example in the 2D space, it is transformed to three examples, each representing $2>1$, $2>3$, and $2>4$.

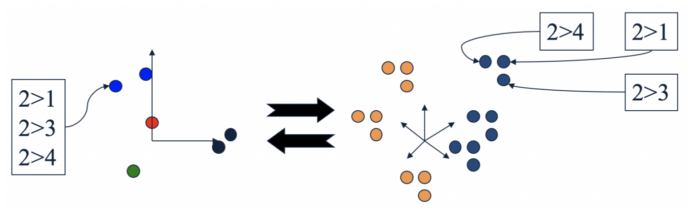


Figure 10: Separability of Kesler construction

Examples in the transformed space are then linearly separable.

7.5 Perceptron in Kesler Construction

If we don't want to run the algorithm in nk-dimensional space, we can actually run it in the n-dimensional space once we understand the notation.
The perceptron update rule applied in the nk-dimensional space is due to a mistake in

$$\mathbf{w}^T \cdot \mathbf{x} \geq 0$$

where $\mathbf{w}$ is the concatenation of the weight vectors.
Or, equivalently to

$$(\mathbf{w}_i^T \mathbf{w}_j^T) \cdot \mathbf{x} \geq 0$$

in the n-dimensional space.
It then implies the following update,

Given example $(\mathbf{x}, i)$ in n-dimensional space with label i, for any (i,j), such that $i, j = 1, \cdots, k, i \neq j$, if

$$(\mathbf{w}_i^T \mathbf{w}_j^T) \cdot \mathbf{x} < 0$$

a mistake is made in the prediction, which is equivalent to $\mathbf{w}^T \cdot \mathbf{x} \geq 0$.
Then, we promote $\mathbf{w}_i$ by

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \mathbf{x}$$

and demote $\mathbf{w}_j$ by

$$\mathbf{w}_j \leftarrow \mathbf{w}_j - \mathbf{x}$$

This is equivalent to what we have to do in the nk-dimensional space for the binary classifier.
Note that this is a generalization of balanced Winnow rule. And we promote $\mathbf{w}_i$ and demote all the other $k-1$ weight vectors $\mathbf{w}_j$'s.
In practice, we will do a more conservative update, because doing $k-1$ updates is costly. Instead, we are doing update once. In case of a mistake, only the weights corresponding to the target node i and that closest node j are updated. Other weight vectors are not being updated.
To be more specific, let $j^* = \text{argmax}_{j=1,\ldots,k}\mathbf{w}_j^T \cdot \mathbf{x}$ be the highest activation among competing labels. If

$$(\mathbf{w}_i^T \mathbf{w}_{j*}^T) \cdot \mathbf{x} < 0$$

a mistake is made in the prediction, then we promote $\mathbf{w}_i$ by

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \mathbf{x}$$

and demote $\mathbf{w}_{j^*}$ by

$$\mathbf{w}_{j^*} \leftarrow \mathbf{w}_{j^*} - \mathbf{x}$$

For practical purposes, all the other weight vectors are farther away. We demote only if they bother us.

7.6 Significance

The hypothesis learned above is more expressive than when the OvA assumption is used. Any linear learning algorithm can be used, and algorithmic-specific properties are maintained (e.g., attribute efficiency if using winnow.)
For example, the multiclass support vector machine can be implemented by learning a hyperplane to separate $P(S)$ with maximal margin.
As a byproduct of the linear separability observation, we get a natural notion of a margin in the multi-class case, inherited from the binary separability in the nk-dimensional space.
Given example $\mathbf{x}_{ij} \in \mathbb{R}^{nk}$,

$$\text{margin}(\mathbf{x}_{ij}, \mathbf{w}) = \min_{ij} \mathbf{w}^T \cdot \mathbf{x}_{ij}$$

Consequently, given $\mathbf{x} \in \mathbb{R}^n$ with label i,

$$\text{margin}(\mathbf{x}, \mathbf{w}) = \min_{j} (\mathbf{w}_i^T - \mathbf{w}_j^T) \cdot \mathbf{x}$$

We get that multiclass margin is indeed the difference between the correct label and its highest competitor.

7.7 Summary

The constraint classification scheme can be generalized to provide a uniform view for multiple types of problems: multi-class, multi-label, category-ranking. It reduces learning to a single binary learning task, and captures theoretical properties of binary algorithm. It has been experimentally verified, and naturally extends Perceptron, SVM, etc.
It is called constraint classification since it does it all by representing labels as a set of constraints or preferences among output labels.