A Mistake Bound for Perceptron Training

This section is based on the article:

We consider linear separators with threshold 0, that is, the separating hyperplane goes through the origin of the coordinate system.

First we rewrite the perceptron training algorithm; essentially it is the same training rule that we discussed earlier. Let $x = (x_1, \ldots, x_n)$ denote the next training instance, with label $y$. We predict the label of $x$ as $\text{sgn}(\sum_j w_j x_j)$. If this misclassifies $x$, we adjust the weights $w_i$ as follows, for all $i$: $w_i := w_i + \Delta w_i$ with $\Delta w_i := yx_i$. (Here the learning rate is set to 1, but we can instead multiply all $x_i$ with a scaling factor.) The initial weights are $w_i = 0$ for all $i$. Let $|x|$ denote the Euclidean length of vector $x$.

The following was shown by Block and Novikoff already in 1962: If all training instances have length at most $R$, and there is a separator with margin $\gamma$, then the perceptron training rule makes at most $(R/\gamma)^2$ mistakes. Here, the margin is the distance between two parallel hyperplanes that separate positive from negative training instances (and no training instances are between them). It is remarkable that this mistake bound does not depend on the dimension of the space, but only on the “separatedness” of the training set.

For the proof we need some basic analytical geometry. We denote the inner product of two vectors $u, v$ by $u \cdot v = \sum_j u_j v_j$. Margin $\gamma$ means the existence of some unit vector $u$ such that $y(u \cdot x) \geq \gamma$ holds for all training instances $(x, y)$, where $y$ denotes the label of $x$. Suppose that the weight vector $w$ misclassifies the instance $(x, y)$. Let $w'$ be the adjusted weight vector according to the training rule. Since $w' = w + yx$, we have $w' \cdot u = w \cdot u + y(u \cdot x) \geq w \cdot u + \gamma$. Since the initial weights were 0, after $k$ mistakes our final weight vector $w''$ fulfills $w'' \cdot u \geq k\gamma$. 

1
Furthermore, a mistake means that $\text{sgn}(w \cdot x)$ differs from $y$, hence $y(w \cdot x) \leq 0$. The length of the weight vector cannot increase too much in the event of a mistake: $|w'|^2 = |w + yx|^2 = |w|^2 + 2y(w \cdot x) + |x|^2 \leq |w|^2 + R^2$. This implies $|w''|^2 \leq kR^2$. Finally we put things together: $\sqrt{kR} \geq |w''| \geq w'' \cdot u \geq k\gamma$, thus $k \leq (R/\gamma)^2$.

More About Numerical Attributes

Nearest Neighbor Learning

This learning approach is suitable for instance sets $X$ where instances are vectors of numerical (i.e., real-valued) attributes, for example, technical or medical parameters, characteristic sizes, coordinates. It applies a very simple classification rule: Fix some number $k$. For any new $x \in X$, take the most common classification value among the $k$ training examples in $D$ closest to $x$. In nearest neighbor learning we are mainly interested in classification values rather than an explicit hypothesis, but of course, the classification rule as such constructs a hypothesis from $D$. As every learning method, this method has an inductive bias: The learner believes that nearby instances say something about the value of $x$, that is, the target concept does not exhibit too many local changes. If this assumption on the target concept is true, then nearest neighbor classification is quite robust against noise.

The nearest neighbor rule looks extremely simple, but one should notice some inherent problems. First of all, what is “nearest”? The distance between instances of $X$ can be defined in many different ways. While there is a natural distance function in geometric spaces, the “proper” definition of distance in parameter spaces is not so clear. The given attributes are, in general, of very different nature. We may multiply the attribute variables by arbitrary factors, i.e., stretch the coordinate axes arbitrarily and independently. It is very easy to construct examples where these arbitrary factors lead to totally different classification results. Even worse, the values of irrelevant attributes may influence the classification, although they should not do so. This phenomenon is called the curse of dimensionality. It is necessary to do some preprocessing where the attributes get suitable factors according to their relevance. Good stretch factors may be determined by cross validation: Take a vector of stretch factors that minimizes the classification error on a validation set. This requires testing of many vectors.

Some computational issues need discussion as well. Nearest neighbor learning is called a “lazy” learning method, as it does not output a hypoth-
esis in explicit form. Instead, much computation is needed at classification time. This is unpleasant because training is done only once, while the classifier shall be applied many times. (In all previous learning methods we had the opposite situation: Training was computationally more or less expensive, but classification was fast and simple.) In order to apply the nearest neighbor rule to a new \( x \), we must efficiently find the \( k \) nearest instances in \( D \). A naive method is to compute the distances between \( x \) and all members of \( D \), and to take the \( k \) smallest. But this is time-consuming if the training set was large. Instead, we may invoke clever data structures where the instances of \( D \) are partially sorted, so that we need not search the whole \( D \) every time.

Finally, which \( k \) should we take? This depends on various circumstances. As a rule of thumb, larger \( k \) are advisable in the following cases: the training set is large, we want classifications to be robust against noise in the data, the target concept depends probabilistically on the attribute values. On the other hand, the larger \( k \) is, the more computation is needed.

Similarly as for all learning approaches, there exist many refinements of the method. For example, we may assign weights to the training examples that decrease with the distance to \( x \), and classify \( x \) by weighted majority vote.

**Discretization of a Numerical Attribute**

No lecture notes provided, because this section is based on:


3