6 Learning and VC-dimension

6.1 Learning

Learning algorithms are general purpose tools that solve problems from many domains without detailed domain-specific knowledge. They have proven to be very effective in a large number of contexts. The task of a learning algorithm is to learn to classify a set of objects. To illustrate with an example, suppose one wants an algorithm to distinguish among different types of motor vehicles such as cars, trucks, and tractors. Using domain knowledge about motor vehicles, one can create a set of features. Some examples of features are the number of wheels, the power of the engine, the number of doors, and the length of vehicle. If there are $d$ features, each object can be represented as a $d$-dimensional vector, called the feature vector, with each component of the vector giving the value of one feature. The objective is to design a “prediction” algorithm that given a vector will correctly predict the corresponding type of vehicle. Earlier rule-based approaches to this problem used domain knowledge to develop a set of rules such as: if the number of wheels is four, it is a car. Prediction was done by checking the rules.

In the learning approach, the process of developing the prediction rules is not domain-specific; it is automated. In learning, domain expertise is used to decide on the choice of features, reducing the problem to one of classifying feature vectors. Further, a domain expert is called on to classify a set of feature vectors, called training examples, and present these as input to the learning algorithm. The role of the expert ends here.

The learning algorithm takes as input the set of labeled training examples and develops a set of rules that applied to the training vectors gives the correct labels. In the motor vehicle example, the learning algorithm needs no knowledge of this domain at all. It just deals with a set of training vectors in $d$-dimensional space and produces a rule to classify $d$-dimensional space into regions, one region corresponding to each of “car”, “truck”, etc.

The task of the learning algorithm is to output a set of rules that correctly labels all training examples. Of course, for this limited task, one could output the rule “for each training example, use the label that the expert has already supplied”. But, we insist on Occam’s razor principle that states that the rules output by the algorithm, must be more succinct than the table of all labeled training examples. This is akin to developing a scientific theory to explain extensive observations. The theory must be more succinct than just a list of observations.

The general task is not to be correct just on the training examples, but have the learnt rules correctly predict the labels of future examples. Intuitively, if the classifier is trained on sufficiently many training examples, then it seems likely that it would work well on the space of all examples. We will see later that the theory of Vapnik-Chervonenkis dimension (VC-dimension) confirms this intuition. For now, our attention is focussed on getting a
succinct set of rules that correctly classifies the training examples. This is referred to as “learning”.

Throughout this chapter, we assume all the labels are binary. It is not difficult to see that the general problem of classifying into one of several types can be reduced to binary classification. Classifying into car or non-car, tractor or non-tractor, etc. will pin down the type of vehicle. So the teacher’s labels are assumed to be +1 or -1. For an illustration, see Figure 6.1 where examples are in 2-dimensions corresponding to two features. Examples labeled -1 are unfilled circles and those labeled +1 are filled circles. The right hand picture illustrates a rule that the algorithm could come up with, the examples above the line are -1 and those below are +1.

The simplest rule in $d$-dimensional space is the generalization of a line in the plane, namely, a half-space. Does a weighted sum of feature values exceed a threshold? Such a rule may be thought of as being implemented by a threshold gate that takes the feature values as inputs, computes their weighted sum and outputs yes or no depending on whether or not the sum is greater than the threshold. One could also look at a network of interconnected threshold gates called a neural net. Threshold gates are sometimes called perceptrons since one model of human perception is that it is done by a neural net in the brain.
6.2 Linear Separators, the Perceptron Algorithm, and Margins

The problem of learning a half-space or a linear separator consists of \( n \) labeled examples, \( \mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n \), in \( d \)-dimensional space. The task is to find a \( d \)-dimensional vector \( \mathbf{w} \), if one exists, and a threshold \( b \) such that

\[
\begin{align*}
\mathbf{w}^T \mathbf{a}_i &> b \quad \text{for each } \mathbf{a}_i \text{ labelled } +1 \\
\mathbf{w}^T \mathbf{a}_i &< b \quad \text{for each } \mathbf{a}_i \text{ labelled } -1.
\end{align*}
\]  

(6.1)

A vector-threshold pair, \( (\mathbf{w}, b) \), satisfying the inequalities is called a linear separator.

The above formulation is a linear program (LP) in the unknowns \( \mathbf{w} \) and \( b \) that can be solved by a general purpose LP algorithm. Linear programming is solvable in polynomial time but a simpler algorithm called the perceptron learning algorithm can be much faster when there is a feasible solution \( \mathbf{w} \) with a lot of wiggle room or margin, though it is not polynomial time bounded in general.

We begin by adding an extra coordinate to each \( \mathbf{a}_i \) and \( \mathbf{w} \), writing \( \mathbf{\hat{a}}_i = (\mathbf{a}_i, 1) \) and \( \mathbf{\hat{w}} = (\mathbf{w}, -b) \). Suppose \( l_i \) is the \( \pm 1 \) label on \( \mathbf{a}_i \). Then, the inequalities in (6.1) can be rewritten as

\[
(\mathbf{\hat{w}}^T \mathbf{\hat{a}}_i) l_i > 0 \quad 1 \leq i \leq n.
\]

Since the right hand side is zero, we may scale \( \mathbf{\hat{a}}_i \) so that \( |\mathbf{\hat{a}}_i| = 1 \). Adding the extra coordinate increased the dimension by one but now the separator contains the origin. For simplicity of notation, in the rest of this section, we drop the hats and let \( \mathbf{a}_i \) and \( \mathbf{w} \) stand for the corresponding \( \mathbf{\hat{a}}_i \) and \( \mathbf{\hat{w}} \).

The perceptron learning algorithm

The perceptron learning algorithm is simple and elegant. We wish to find a solution \( \mathbf{w} \) to

\[
(\mathbf{w}^T \mathbf{a}_i) l_i > 0 \quad 1 \leq i \leq n
\]

(6.2)

where \( |\mathbf{a}_i| = 1 \). Starting with \( \mathbf{w} = l_1 \mathbf{a}_1 \), pick any \( \mathbf{a}_i \) with \( (\mathbf{w}^T \mathbf{a}_i) l_i \leq 0 \), and replace \( \mathbf{w} \) by \( \mathbf{w} + l_i \mathbf{a}_i \). Repeat until \( (\mathbf{w}^T \mathbf{a}_i) l_i > 0 \) for all \( i \).

The intuition behind the algorithm is that correcting \( \mathbf{w} \) by adding \( \mathbf{a}_i l_i \) causes the new \( (\mathbf{w}^T \mathbf{a}_i) l_i \) to be higher by \( \mathbf{a}_i^T \mathbf{a}_i l_i^2 = |\mathbf{a}_i|^2 \). This is good for this \( \mathbf{a}_i \). But this change may be bad for other \( \mathbf{a}_j \). The proof below shows that this very simple process quickly yields a solution \( \mathbf{w} \) provided there exists a solution with a good margin.

Definition 6.1 For a solution \( \mathbf{w} \) to (6.2), where \( |\mathbf{a}_i| = 1 \) for all examples, the margin is defined to be the minimum distance of the hyperplane \( \{ \mathbf{x} \mid \mathbf{w}^T \mathbf{x} = 0 \} \) to any \( \mathbf{a}_i \), namely,

\[
\min_i \frac{(\mathbf{w}^T \mathbf{a}_i) l_i}{|\mathbf{w}|}.
\]
If we did not require that all $|a_i| = 1$ in (6.2), then one could artificially increase the margin by scaling up the $a_i$. If we did not divide by $|w|$ in the definition of margin, then again, one could artificially increase the margin by scaling $w$ up. The interesting thing is that the number of steps of the algorithm depends only upon the best margin any solution can achieve, not upon $n$ or $d$. In practice, the perceptron learning algorithm works well.

**Theorem 6.1** Suppose there is a solution $w^*$ to (6.2) with margin $\delta > 0$. Then, the perceptron learning algorithm finds some solution $w$ with $(w^T a_i)l_i > 0$ for all $i$ in at most $\frac{1}{\delta^2} - 1$ iterations.

**Proof:** Scale $w^*$ so that $|w^*| = 1$. Consider the cosine of the angle between the current vector $w$ and $w^*$, that is, $\frac{w^T w^*}{|w||w^*|}$. In each step of the algorithm, the numerator of this fraction increases by at least $\delta$ because

\[
(w + a_il_i)^T w^* = w^T w^* + l_i a_i^T w^* \geq w^T w^* + \delta.
\]

On the other hand, the square of the denominator increases by at most one because

\[
|w + a_il_i|^2 = (w + a_il_i)^T (w + a_il_i) = |w|^2 + 2(w^T a_i)l_i + |a_i|^2 l_i^2 \leq |w|^2 + 1
\]

since $w^T a_i l_i \leq 0$, the cross term is nonpositive.

After $t$ iterations, $w^T w^* \geq (t + 1)\delta$ since at the start $w^T w^* = l_1 (a_1^T w^*) \geq \delta$ and at each iteration $w^T w^*$ increases by at least $\delta$. Similarly after $t$ iterations $|w|^2 \leq t + 1$ since at the start $|w| = |a_1| = 1$ and at each iteration $|w|^2$ increases by at most one. Thus, the cosine of the angle between $w$ and $w^*$ is at least $\frac{(t+1)\delta}{\sqrt{t+1}}$ and the cosine cannot exceed one. Now

\[
\frac{(t+1)\delta}{\sqrt{t+1}} \leq 1 \quad \sqrt{t+1} \delta \leq 1 \quad t + 1 \leq \frac{1}{\delta^2} \quad t \leq \frac{1}{\delta^2} - 1
\]
Therefore the algorithm must stop before $\frac{1}{\delta} - 1$ iterations and at termination, $(w^T a_i)l_i > 0$ for all $i$. This proves the theorem.

How strong is the assumption that there is a separator with margin at least $\delta$? Suppose for the moment, the $a_i$ are picked from the uniform density on the surface of the unit hypersphere. We saw in Chapter 2 that for any fixed hyperplane passing through the origin, most of the mass of the unit sphere is within distance $O(1/\sqrt{d})$ of the hyperplane. So, the probability of one fixed hyperplane having a margin of more than $c/\sqrt{d}$ is low. But this does not mean that there is no hyperplane with a larger margin. By the union bound, one can only assert that the probability of some hyperplane having a large margin is at most the probability of a specific one having a large margin times the number of hyperplanes which is infinite. Later we will see using VC-dimension arguments that indeed the probability of some hyperplane having a large margin is low if the examples are selected at random from the hypersphere. So, the assumption of large margin separators existing may not be valid for the simplest random models. But intuitively, if what is to be learnt, like whether something is a car, is not very hard, then, with enough features in the model, there will not be many “near cars” that could be confused with cars nor many “near non-cars”. In a real problem such as this, uniform density is not a valid assumption. In this case, there should be a large margin separator and the theorem would work.

The question arises as to how small margins can be. Suppose the examples $a_1, a_2, \ldots, a_n$ were vectors with $d$ coordinates, each coordinate a 0 or 1 and the decision rule for labeling the examples was the following.

If the first 1 coordinate of the example is odd, label the example +1.
If the first 1 coordinate of the example is even, label the example -1.

This rule can be represented by the decision rule

$$(a_{i1}, a_{i2}, \ldots, a_{in}) \left(1, -\frac{1}{2}, \frac{1}{4}, -\frac{1}{8}, \ldots\right)^T = a_{i1} - \frac{1}{2}a_{i2} + \frac{1}{4}a_{i3} - \frac{1}{8}a_{i4} + \cdots > 0.$$ 

However, the margin in this example can be exponentially small. Indeed, if for an example $a$, the first $d/10$ coordinates are all zero, then the margin is $O(2^{-d/10})$.

**Maximizing the Margin**

In this section, we present an algorithm to find the maximum margin separator. The margin of a solution $w$ to $(w^T a_i)l_i > 0$, $1 \leq i \leq n$, where $|a_i| = 1$ is $\delta = \min_l l_i |(w^T a_i)/|w||$. Since this is not a concave function of $w$, it is difficult to deal with computationally.

Convex optimization techniques in general can only handle the maximization of concave functions or the minimization of convex functions over convex sets. However, by