1 Algorithms for Machine Learning

Suppose you want to classify mails as spam/not-spam. Your data points are email messages, and say each email can be classified as a positive instance (it is spam), or a negative instance (it is not). It is unlikely that you'll get more than one copy of the same message: you really want to "learn" the concept of spam-ness, so that you can classify future emails correctly.

Emails are too unstructured, so you may represent them as more structured objects. One simple way is to use feature vectors: say you have a long bit vector indexed by words and features (e.g., "money", "bank", "click here", poor grammar, mis-spellings, known-sender) and the i^{th} bit is set if the i^{th} feature occurs in your email. (You could keep track of counts of words, and more sophisticated features too.) Now this vector is given a single-bit label: 1 (spam) or -1 (not spam). These positively- and negatively-labeled vectors are sitting in $\mathbb{R}^d$, and ideally we want to find something about the structure of the positive and negative clouds-of-points so that future emails (represented by vectors in the same space) can be correctly classified.

2 Linear Separability

How do we represent the concept of "spam-ness"? Given the geometric representation of the input, it is reasonable to represent the concept geometrically as well. Here is perhaps the simplest formalization: we are given a set S of n pairs $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$, where each $\mathbf{x}_i$ is a *data point* and y_i is a *label*, where you should think of $\mathbf{x}_i$ as the vector representation of the actual data point, and the label $y_i = 1$ as "$\mathbf{x}_i$ is a positive instance of the concept we are trying to learn" and $y_i = -1$ as "$\mathbf{x}_i$ is a negative instance".[1]

The goal is to find a vector $\mathbf{w} \in \mathbb{R}^d$ such that for all i, $\mathbf{w} \cdot \mathbf{x}_i \geq 1$ if $y_i = 1$, and $\mathbf{w} \cdot \mathbf{x}_i \leq -1$ if $y_i = -1$. If such a vector exists then the data is said to be *linearly separable*, and the hyperplane $\mathbf{w} \cdot \mathbf{x} = 0$ is called the *separating hyperplane*.[2] Let us write down a formal definition (using a slick compact representation of the requirements).

Definition 1 *The* Linear Separator *problem takes as input a set S of labeled points $(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{-1, 1\}$. The desired output is a vector $\mathbf{w}$ such that for all i,*

$$y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1.$$

(Check that this is indeed equivalent to what we wanted before!) Note that we don't have any requirements on the length of $\mathbf{w}$: so if you found some $\mathbf{w}'$ such that $y_i(\mathbf{w}' \cdot \mathbf{x}_i) \geq 0.00001$ for all i, then you could set $\mathbf{w} = 10^5\,\mathbf{w}'$ and solve the Linear Separator problem.

[1] If $y_i = 1$ then x_i is called a *positive example*; if $y_i = -1$ then x_i is called a *negative example*.

[2] Of course, such a perfect separating hyperplane may not exist. What to do then? You can try choosing more/better features. You may consider still sticking with a linear classifier but allowing some errors, e.g., using support vector machines. You may consider higher-dimensional mappings called kernels. Consult a machine-learning courses close to you.

2.1 Attempt #1: Using a Big Hammer

How to find such a vector $\mathbf{w}$? One way, given what we have learned already, is to use linear programming. The variables are the coefficients of $\mathbf{w}$, namely $w_1, w_2, \ldots, w_d$. The constraints are

$$w \cdot x_i = \sum_{j=1}^{d} w_j x_{ij} \geq 1 \qquad \forall i \text{ s.t. } y_i = 1$$
$$w \cdot x_i = \sum_{j=1}^{d} w_j x_{ij} \leq -1 \qquad \forall i \text{ s.t. } y_i = -1$$

These are all linear constraints, and we just want to find any feasible solution $\mathbf{w} \in \mathbb{R}^d$ given these constraints.

2.2 Attempt #2: A Different Algorithm

Here's an algorithm that works well when there exists a solution $\mathbf{w}$ with small length: the algorithm runs in time proportional to $\|\mathbf{w}\|^2$. (Here $\|\mathbf{w}\|$ is the Euclidean length of the vector $\mathbf{w} = (w_1, w_2, \ldots, w_d)$; recall this is $\sqrt{\sum_i w_i^2} = \sqrt{\mathbf{w} \cdot \mathbf{w}}$.)

2.2.1 The Margin of the Data Set

Let us define the *margin* of the data set. For simplicity, assume that all data points are inside the unit ball, i.e., $\|x_i\| \leq 1$ for all i. As the name suggests, the margin tries to capture the distance of the points from the boundary.

Suppose $\mathbf{w}^*$ is a linear separator for the data set S having minimum (Euclidean) length. Then the margin of S is

$$\gamma(S) := 1/\|\mathbf{w}^*\|.$$

Why this? A little thought shows that given any hyperplane $\mathbf{w}$ and a point $\mathbf{x}_i$, the Euclidean distance of the point from the hyperplane $\mathbf{w} \cdot \mathbf{x} = 0$ is

$$\frac{\mathbf{w} \cdot \mathbf{x}_i}{\|\mathbf{w}\|}$$

Now if $\mathbf{w} \cdot \mathbf{x}_i \geq 1$, then is at least

$$\text{the distance of } \mathbf{x}_i \text{ from the hyperplane } \geq \frac{1}{\|\mathbf{w}\|}.$$

Hence, the smaller the length of $\mathbf{w}$, the larger the distance of the point $\mathbf{x}_i$ from the boundary. Choosing the vector $\mathbf{w}$ to be the one with smallest length (namely the vector $\mathbf{w}^*$), we get the distance of the points from the separating hyperplane to be at least $1/\|\mathbf{w}^*\| = \gamma(S)$.

What does small length $\|\mathbf{w}^*\|$ (or equivalently, large margin $\gamma(S)$) mean? If the data points are "far away" from the separating hyperplane $\mathbf{w}^* \cdot x = 0$, tilting the hyperplane slightly should not cause any points $\mathbf{x}_i$ to become misclassified, i.e., to go from the positive to the negative side (or vice versa). And this should make our problem (of finding any such separating hyperplane) easier. Indeed, this is what happens with our next algorithm, called the *Perceptron algorithm*.

2.2.2 The Perceptron Algorithm

The algorithm is very easy to state:

Start with $\mathbf{w} = 0$.
If there exists $(\mathbf{x}_i, y_i)$ for which $\mathbf{w}$ is incorrect,
    If $\mathbf{w} \cdot \mathbf{x}_i < 1$ and $y_i = 1$ then $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_i$.
    If $\mathbf{w} \cdot \mathbf{x}_i > -1$ and $y_i = -1$ then $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_i$.

We can restate it slightly more compactly as follows:

> Start with $\mathbf{w} = 0$.
> If there exists $(\mathbf{x}_i, y_i)$ for which $\mathbf{w}$ is incorrect, set $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$.

In words, if the current vector $\mathbf{w}$ makes a mistake on $\mathbf{x}_i$, we try to fix it by just adding $y_i\mathbf{x}_i$ to it. Intuitively this makes sense: if $\mathbf{w}'$ denotes $\mathbf{w} + y_i\mathbf{x}_i$, then note that

$$\mathbf{w}' \cdot \mathbf{x}_i = \mathbf{w} \cdot \mathbf{x}_i + y_i(\mathbf{x}_i \cdot \mathbf{x}_i) = \mathbf{w} \cdot \mathbf{x}_i + y_i\|\mathbf{x}_i\|^2.$$

We have reduced the error: e.g., if we were incorrect on $\mathbf{x}_i$ for which $y_i = 1$, we had $\mathbf{w} \cdot \mathbf{x}_i \leq 0$, and our update step adds in $y_i\|\mathbf{x}_i\|^2 > 0$, thereby reducing the error.

This makes intuitive sense — but it is not a proof. Maybe we fixed $\mathbf{x}_i$ a bit but messed up other $\mathbf{x}_j$s? How do we analyze this?

Theorem 2 *The Perceptron algorithm updates the vector* $\mathbf{w}$ *at most* $3/\gamma(S)^2 = 3\|\mathbf{w}^*\|^2$ *times.*

Proof: We will track the quantity $\Phi = \mathbf{w}^* \cdot \mathbf{w}$. (Recall that $\mathbf{w}^*$ was the vector with least length, one that showed the margin to be $\gamma(S)$.) Initially this quantity is 0 since $\mathbf{w} = 0$.

Now consider an update in which we add in $y_i\mathbf{x}_i$ to $\mathbf{w}$, and let $\mathbf{w}' := \mathbf{w} + y_i\mathbf{x}_i$. How does the squared length of $\mathbf{w}$ change?

$$\begin{aligned}\|\mathbf{w}'\|^2 = \mathbf{w}' \cdot \mathbf{w}' &= (\mathbf{w} + y_i\mathbf{x}_i) \cdot (\mathbf{w} + y_i\mathbf{x}_i) \\ &= \mathbf{w} \cdot \mathbf{w} + 2y_i(\mathbf{w} \cdot \mathbf{x}_i) + y_i^2(\mathbf{x}_i \cdot \mathbf{x}_i) \\ &= \|w\|^2 + 2(\text{something } < 1) + 1(\text{something } \leq 1).\end{aligned}$$

So on each update the new squared-length of $\mathbf{w}$ is at most the old squared-length plus 3.

What about the change in $\mathbf{w}^* \cdot \mathbf{w}$?

$$(\mathbf{w}^* \cdot \mathbf{w}') = \mathbf{w}^* \cdot \mathbf{w} + y_i(\mathbf{w}^* \cdot \mathbf{x}_i) \geq (\mathbf{w}^* \cdot \mathbf{w}) + 1.$$

The last inequality uses the fact that $y_i(\mathbf{w}^* \cdot \mathbf{x}_i) \geq 1$, since $\mathbf{w}^*$ is a solution to our problem. So upon each update the increase in this dot-product is at least 1.

Finally, suppose we do M updates. Using the above facts, we get that the final vector $\mathbf{w}_f$ satisfies:

$$\begin{aligned}\|\mathbf{w}_f\|^2 &\leq 3M \\ \mathbf{w}^* \cdot \mathbf{w}_f &\geq M.\end{aligned}$$

Putting these together,

$$M \leq \mathbf{w}^* \cdot \mathbf{w}_f \leq \|\mathbf{w}^*\| \, \|\mathbf{w}_f\| \leq \|\mathbf{w}^*\| \times (3M)^{1/2}$$

Simplifying, $M \leq 3\|\mathbf{w}^*\|^2 = 3/\gamma(S)^2$. (BTW, the inequality $\mathbf{x} \cdot \mathbf{y} \leq \|\mathbf{x}\| \, \|\mathbf{y}\|$ is the familiar Cauchy-Schwarz inequality.) ■

Nice and easy, and the running time on data that is linearly separable degrades nicely as large margin decreases.

What happens with the data is not linearly-separable? Then the algorithm may never terminate. For such data, you can look into other techniques, for instance, support-vector machines; they also have a nice theory behind it and work well in practice. (Citations later, see Wikipedia for now.)

We can also view Perceptron as an online algorithm, where the data points from S are arriving one-by-one, and we want to predict the labels. When $(\mathbf{x}_i, y_i)$ arrives, we initially just see $\mathbf{x}_i$, and want to predict the label y_i. Suppose we predict $\widetilde{y}_i := sign(\mathbf{w} \cdot \mathbf{x}_i)$. If we make a mistake (if $y_i \neq \widetilde{y}_i$), we repeatedly add in $y_i \mathbf{x}_i$ to $\mathbf{w}$ until $y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$. The analysis above shows that the total number of mistakes — the number of i for which $\widetilde{y}_i \neq y_i$ — is at most $3(1/\gamma(S))^2$, where $\gamma(S)$ is the margin of the dataset.[3]

3 The Mistake Bound Model

In this section, we consider other online prediction problems: at each time-step we first make a prediction of some sort, *then* we see the real answer — we want to minimize the total number of wrong predictions we make.

The specific problem we consider is that of *learning from expert advice*. Say you want to predict the stock market. Every morning you predict whether the market will go up or down (just that one bit of information), and every evening you find our whether you were correct or now. Of course, you don't predict these things in a vacuum: you listen to the predictions from N "experts". Say, the WSJ, the radio, the TV, various blogs, all tell you their predictions. (These experts are not necessarily experts in any sense, they just happen to have an opinion.) Based on all their up/down predictions, you predict up/down — call this bit $\widetilde{y}_i$. Then you find out the real y_i. You want to minimize the mistakes, the number of i for which you predict wrong, i.e., for which $\widetilde{y}_i \neq y_i$.

How well can you do? The yardstick we'll use in this case is the number of mistakes made by best expert in hindsight. (Basically, you don't want to have regrets of the form *"I could have just followed to expert blah's advice every day, and done much better than I did by being smart."*)

[Now follow PowerPoint slides.]

[3] This is a loose estimate, the actual number of mistakes may be lower. E.g., we may get the sign correct even if $|\mathbf{w} \cdot \mathbf{x}_i| < 1$; also, some mistakes might cause multiple updates to $\mathbf{w}$ which all contribute to the total count of $3(1/\gamma(S))^2$, even we only make a single mistake.