**COS 511: Theoretical Machine Learning**

Lecturer: Rob Schapire
Scribe: William Berkeley

Lecture #13
March 24, 2008

In this lecture, we finish our discussion of support vector machines (SVMs) and begin talking about the on-line learning model.

# 1 Finishing Up Support Vector Machines

Previously, we formulated two equivalent ways of finding the output vector of an SVM. The first formulation seeks a vector $\mathbf{v}$ that solves the following constrained maximization problem:

$$\begin{aligned} &\max \delta \\ &\text{s.t. } \|\mathbf{v}\| = 1 \\ &\forall i : y_i(\mathbf{v} \cdot \mathbf{x}_i) \geq \delta. \end{aligned}$$

The second formulation (the "unnormalized" form) eliminates the $\delta$-parameter by dropping the condition that the solution vector $\mathbf{w}$ be normalized:

$$\begin{aligned} &\min \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &\text{s.t. } \forall i : y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1. \end{aligned}$$

We also discussed how to form the dual maximization problem to the above minimization problem using Lagrange multipliers:

$$\begin{aligned} &\max \textstyle\sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ &\text{s.t. } \forall i : \alpha_i \geq 0. \end{aligned}$$

The $\alpha_i$'s are the Lagrange multipliers, and the output vector $\mathbf{w}$ equals $\sum_i \alpha_i y_i \mathbf{x}_i$.

SVM's can also be used in the case when the data is not linearly separable in $\mathbb{R}^n$. To do this, we map the data into a higher dimensional space where it does become linearly separable. For example, in $\mathbb{R}^2$ we could use the mapping $\Psi$:

$$\Psi(\mathbf{x}) = \Psi(x_1, x_2) = (1, x_1, x_2, x_1 x_2, x_1^2, x_2^2).$$

and then use the SVM algorithm to compute a classifier. For $\Psi$, this corresponds to using conic sections as separators in $\mathbb{R}^2$.

However, there are two apparent problems with moving to higher dimensions. The first is statistical: with more dimensions, we expect that more data will be required. As it turns out, SVM's do not suffer from this problem because the VC-dimension is bounded by $1/\delta^2$ and this does not depend on the number of dimensions of the data. The second problem is computational: if we use functions like $\Psi$ that add all terms to degree $d$, then the number of dimensions grows as $O(n^d)$. The number of dimensions quickly becomes so large that manipulating the data is computationally onerous. However, we can also avoid this problem. The key observation is that the only operation performed on the data is the inner product, so if we can choose a mapping into higher dimensions in a clever way we can calculate the inner products without explicitly considering all the dimensions.

For example, let us modify the function $\Psi$ from before:

$$\Psi(\mathbf{x}) = \Psi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2).$$

Notice that the addition of the multiplicative constants does not change what can be represented by hyperplanes under this mapping. Then if we have two vectors $\mathbf{x}$, $\mathbf{u}$,

$$\Psi(\mathbf{x})\cdot\Psi(\mathbf{u}) = 1+2x_1u_1+2x_2u_2+2x_1x_2u_1u_2+x_1^2u_1^2+x_2^2u_2^2 = (1+x_1u_1+x_2u_2)^2 = (1+\mathbf{x}\cdot\mathbf{u})^2.$$

Now, instead of computing an inner product in a higher dimensional space, we may compute an inner product in the original space, add one, and square the result. This leads to big computational savings when mapping into large dimensional spaces. In particular, we need never actually examine all the dimensions of the mapping $\Psi$. $\Psi$ is an example of a *kernel function*: it is the kernel $K(\mathbf{x}, \mathbf{u}) = (1 + \mathbf{x} \cdot \mathbf{u})^d$ when $d = 2$. Another example is the radial basis kernel $K(\mathbf{x}, \mathbf{u}) = \exp(-c\|\mathbf{x} - \mathbf{u}\|^2)$. Kernel functions allow the computation of inner products in some high-dimensional space (which need not be specified) using only computations in the lower dimensional space. To use a kernel function in an SVM algorithm, simply replace every instance of an inner product $\mathbf{x} \cdot \mathbf{u}$ with the kernel function $K(\mathbf{x}, \mathbf{u})$.

We are now finished discussing support vector machine and are ready to move on to a new learning model—the on-line learning model.

## 2 The Online Learning Model

First, we will highlight some key aspects of the on-line learning model and contrast them with the PAC model.

- In the on-line model, the algorithm is presented with an example for which it must make a prediction. It then gets to see the correct labeling of the example and can adjust its behavior for future predictions accordingly. By contrast, in the PAC model the algorithm is given a fixed batch of examples and it computes a fixed hypothesis used in all future tests.
- The on-line model assumes *nothing* about the distribution of examples, as opposed to the statistical assumptions of the PAC model.
- The goal of an on-line learning algorithm is to minimize the number of mistakes made. This is similar to the PAC model, where the algorithm is attempting to minimize the probability of a mistake on a randomly chosen test example.
- As a nice bonus, the algorithms and analyses in the on-line model tend to be very simple and intuitive.

We now consider an instance of the on-line learning model known as learning with expert advice.

## 3 Learning with Expert Advice

Suppose we are trying to predict whether the S&P 500 stock index will increase or decrease on any given day. We could consult the advice of various "experts." In this case, experts

might be financial publications, television news networks' financial analysts, etc. Then we could synthesize the advice of the experts into a prediction. This is an example of learning with expert advice—the learning algorithm has a set of experts that make predictions and the algorithm combines these predictions to form its own prediction. The experts could be people, other learning algorithms, or functions depending on fixed parameters. Ideally, the algorithm will be almost as accurate as the best expert. We formalize the situation as follows:

- There are $N$ experts.
- For each round $t = 1, \ldots, T$,
  - Each expert $i$ makes a prediction $\xi_i \in \{0, 1\}$
  - The learner (using the experts' predictions) makes a prediction $\hat{y} \in \{0, 1\}$
  - The learner observes the actual outcome $y \in \{0, 1\}$. There is a mistake if $\hat{y} \neq y$.

We would like a bound on the number mistakes the learner makes based on the number of mistakes of the best expert. First we consider a simple special case when there is one expert who is perfect, making no mistakes at all.

# 4 Learning with a Perfect Expert and the Halving Algorithm

The idea is to eliminate experts when they make a mistake. The algorithm will predict by taking a majority vote of the "surviving" experts each round (that is, of experts who have not been eliminated). To analyze the algorithm, we define

$$W = (\text{the number of surviving experts}).$$

Then $W = N$ initially. Suppose the learner makes a mistake. Then at least half of the experts must have been wrong, so $W$ decreases by at least a factor of 2. Then after $m$ mistakes, $W \leq N/2^m$. But one expert does not make mistakes, so that expert will never be eliminated and $W \geq 1$. Then $1 \leq W \leq N/2^m$, so $m \leq \lg(N)$. We summarize in the following theorem:

**Theorem 1** *Suppose there are $N$ experts and one expert that never errs. Then the halving algorithm will make at most $\lg(N)$ mistakes.*

We now relate the halving algorithm to PAC learning. Suppose we have a finite hypothesis space $\mathcal{H}$ and a target concept $c \in \mathcal{H}$. Suppose we try to discover $c$ using the halving algorithm. The experts are the hypotheses $h \in \mathcal{H}$ and the expert $c \in \mathcal{H}$ is perfect. Then the theorem tells us that to discover $c$ we need at most $\lg(N) = \lg(|H|)$ mistakes. Notice that this is the same as our previous complexity measure of $\mathcal{H}$.

Now we attempt to prove a lower bound on the number of mistakes needed to learn $c$. We define

$$M_A(\mathcal{H}) = \max_{adversaries} (\#mistakes)$$

to be the worst-case number of mistakes made by algorithm $A$ in this setting, and

$$opt(\mathcal{H}) = \min_A (M_A(\mathcal{H}))$$

to be the optimal number of mistakes made by the best *deterministic* algorithm $A$. Then the theorem implies $M_{halving}(\mathcal{H}) \leq \lg |H|$. We will lower bound $opt(\mathcal{H})$ by the VC-dimension $d$ of $\mathcal{H}$ as follows: Let $x_1, x_2, \ldots, x_d$ be a set of points shattered by $\mathcal{H}$. For each round $t = 1, \ldots, d$, an adversary chooses example $x_t$, calculates $A$'s prediction $\hat{y}$, which is possible because $A$ is deterministic, then chooses $y \neq \hat{y}_A$. The resulting $d$ labeled examples will be consistent with some concept in $\mathcal{H}$ because the set $x_1, \ldots, x_d$ is shattered by $\mathcal{H}$. Then $A$ will err on all $d$ examples. Thus $opt(\mathcal{H}) \geq d$. Summarizing, we have shown that

$$VCdim(\mathcal{H}) \leq opt(\mathcal{H}) \leq M_{halving}(\mathcal{H}) \leq \lg |\mathcal{H}|.$$

We now consider the case when there is no perfect expert.

# 5 The Weighted Majority Algorithm

The idea is to keep a weight on each expert and decrease the weight of an expert each time it errs (essentially listening to that expert less in later rounds). Thus we have a parameter $\beta \in [0, 1)$ and, for each expert, a weight $w_i$. Initially, we set $w_i = 1$. Each round, we calculate

$$q_0 = \sum_{i:\xi_i=0} w_i$$

and

$$q_1 = \sum_{i:\xi_i=1} w_i.$$

If $q_1 > q_0$ we predict $\hat{y} = 1$, otherwise we predict $\hat{y} = 0$. Then we observe the correct label $y$ and, for each expert $i$, if $i$ erred we reset $w_i$ to $\beta w_i$ and if $i$ did not err we do nothing. Then we have the following theorem:

**Theorem 1** *Suppose there are $N$ experts. Then*

$$\textit{(\# of mistakes of the learner)} \leq a_\beta \textit{(\# of mistakes of the best expert)} + c_\beta \lg(N)$$

*where*

$$a_\beta = \frac{\lg(1/\beta)}{\lg(\frac{2}{1+\beta})}$$

*and*

$$c_\beta = \frac{1}{\lg(\frac{2}{1+\beta})}$$

Before we give the proof, we give some sample values for the coefficients $a_\beta$ and $c_\beta$. When $\beta = 1/2$, $a_\beta \approx 2.4$ and $c_\beta \approx 2.4$. As $\beta \to 0$, $a_\beta \to \infty$ and $c_\beta \to 1$. As $\beta \to 1$, $a_\beta \to 2$ and $c_\beta \to \infty$. Thus there is a tradeoff between $a_\beta$ and $c_\beta$.

We will prove the theorem by keeping track of $W$, the total weight of all the experts. Initially, $W = N$. Suppose that on some round $y = 0$ (the argument in the case of $y = 1$ is symmetric). Then

$$W_{new} = q_1\beta + q_0 = q_1\beta + (W - q_1) = W - (1 - \beta)q_1$$

Suppose there is a mistake: $\hat{y} \neq y$. Then $q_1 \geq W/2$, so

$$W_{new} \leq W - (1 - \beta)(W/2) = \frac{1 + \beta}{2} W.$$

Then after $m$ mistakes, $W \leq \left(\frac{1+\beta}{2}\right)^m N$.

Now we prove a lower bound on $W$. Let $L_i$ be the total number of mistakes of expert $i$. Then $w_i = \beta^{L_i}$, and so

$$\beta^{L_i} = w_i \leq W \leq \left(\frac{1+\beta}{2}\right)^m N.$$

Solving for $m$, we get

$$m \leq \frac{L_i \lg(1/\beta) + \lg(N)}{\lg \frac{2}{1+\beta}}.$$

This holds for any expert, so we choose $i$ to be the best expert. The bound follows.