COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire Lecture #14
Scribe: Li-Fang (Fanny) Cheng March 27, 2014

1 SVM

1.1 A brief review

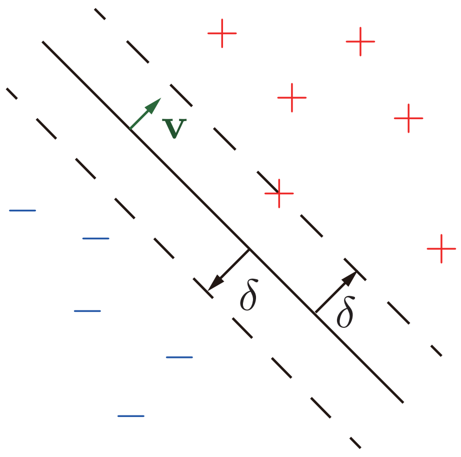

Figure 1: An illustration of the key idea of SVM for linearly separable data.

As discussed in the previous lecture, the key idea of support vector machine (SVM) is to find a hyperplane that can separate the given data. Figure 1 illustrates this idea when the data is linearly separable. The hyperplane is defined by a unit normal vector $\mathbf{v}$, and if we suppose the hyperplane passes through the origin, we can formulate the prediction of a data point $\mathbf{x}$ as

$$y = \text{sign}\left(\mathbf{v} \cdot \mathbf{x}\right)$$

The distance from a data point to the hyperplane in the "right" direction is called the margin:

$$\text{margin}(\mathbf{x}, y) = y\left(\mathbf{v} \cdot \mathbf{x}\right)$$

For a given labeled data set $(\mathbf{x}_i, y_i)$, $i = 1, \cdots m$, we define the smallest margin δ as

$$\delta = \min_i y_i\left(\mathbf{v} \cdot \mathbf{x}_i\right)$$

Our objective is to find a hyperplane with maximized δ by solving convex optimization problems. Figure 2 gives a summary of both the primal convex optimization problem and its dual form.

The next question we would like to ask is what kinds of operations do we need for each sample when solving the optimization problems. According to Equation 1, we can tell that the dot product, $(\mathbf{x}_i \cdot \mathbf{x}_j)$, is the only computation we need for each sample when solving the dual SVM problem.

Goal of SVM problem: find $\mathbf{v}$ to maximize δ

$$\text{define} \quad \mathbf{w} = \frac{\mathbf{v}}{\delta} \Rightarrow \mathbf{v} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Primal Form (Linearly Separable):

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

Dual Form (Linearly Separable):

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$$

$$\max \quad \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{1}$$

$$\text{s.t.} \quad \forall i : \alpha_i \geq 0$$

Primal Form (Soft Margin):

$$\min \quad \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \sum_i \xi_i \tag{2}$$

$$\text{s.t.} \quad \forall i : y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1 - \xi_i \tag{3}$$

$$\xi_i \geq 0 \tag{4}$$

Figure 2: The primal and dual SVM problems.

Sometimes there is a case that the data is linearly inseparable. If the data is “almost” linearly separable, we can use the soft margin SVM. In this case, we allow the hyperplane to make a few mistakes in classification by moving some data points slightly. The optimization problem is then reformulated in Equation 2 to 4, as discussed in the previous lecture.

1.2 More on Linearly Inseparable Data

What can we do if the data is just too far from linearly separable as is the case in Figure 4? In this situation, we have to look for another solution. We map the data to a higher dimensional space where the data can be linearly separable. Here follows an example.

Suppose the original data is in 2-dimensional space; we can use the following method to map it into 6 dimensional space:

$$\mathbf{x} = (x_1, x_2) \mapsto \psi(\mathbf{x}) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2) \tag{5}$$

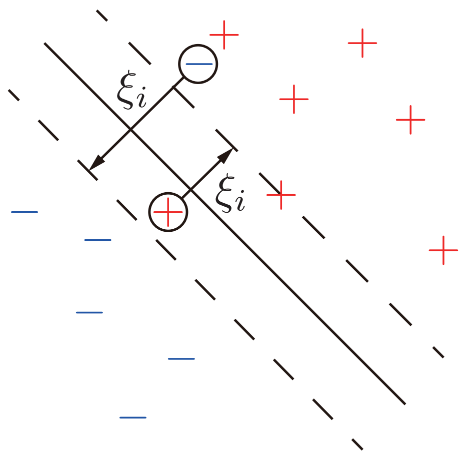

Figure 3: An illustration of the soft margin SVM for nearly linearly separable data.

The new hyperplane consists of the points that $\mathbf{v} \cdot \psi(\mathbf{x}) = 0$, which is written as:

$$a + bx_1 + cx_2 + dx_1x_2 + ex_1^2 + fx_2^2 = 0 \tag{6}$$

where $\mathbf{v} = (a, b, c, d, e, f)$. Equation 6 defines a hyperplane in the 6-dimensional space, while in the original 2-dimensional space, it is the general equation for a conic section, that is, a line, circle, ellipse, parabola or hyperbola. Therefore, by this mapping, we are able to separate the 2-dimensional data in the 6-dimensional space. Figure 5 shows a possible 6-dimensional hyperplane that has the form of an ellipse in the 2-dimensional space for classification.

The method described above can be generalized. If we start with n dimensional space, by adding up all terms of degree at most k, we can have $O(n^k)$ dimensional space.

Next, we have to notice possible problems when adopting this approach. There are a statistical problem and a computational problem:

- Statistical Problem: According to the results above, if we start from 100 dimensions, it is possible that we reach more than a trillion dimensions. That is, we will have more parameters to train and the complexity of the hypothesis will be higher. In this case, we generally need more samples to achieve better fitting results. If we have too few samples, the algorithm overfits easily.

- Computational Problem: The storage space of the data is in proportion to the number of dimensions. It would take too much time to even read the data if the dimension after mapping is too high.

However, SVM can overcome both kinds of problems. First, for the statistical problem, we use the result that

$$\text{VC-dimension} \leq \left(\frac{R}{\delta}\right)^2, \tag{7}$$

where R is the radius of the sphere that contains all the data, and δ is the margin. We should notice that VC-dimension does not depend on the number of dimensions of the data. Therefore, although increasing the dimension of data might increase R, generally δ also gets larger. That is, we can expect that VC-dimension is not growing so fast.

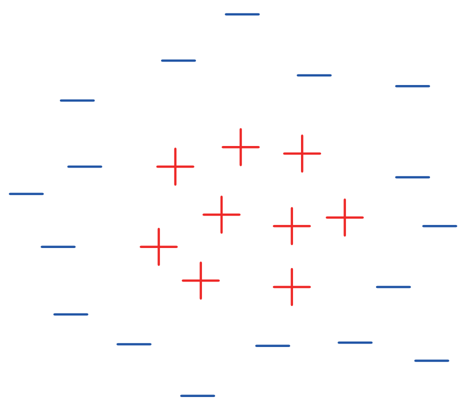

Figure 4: A non-linearly separable data set in $\mathbb{R}^2$.

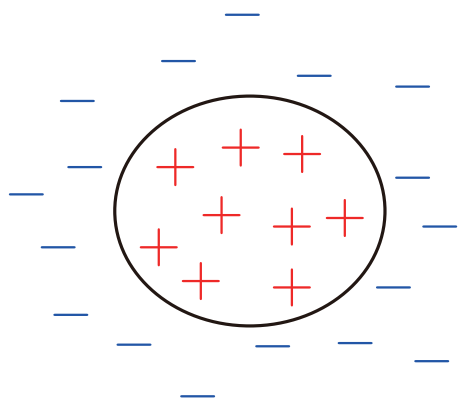

Figure 5: Finding an ellipse to separate the data by mapping from $\mathbb{R}^2$ to $\mathbb{R}^6$.

For the computational problem, recall the previous observation that for each sample, we only need to compute the inner products. Therefore after the mapping, the computation for two samples $\mathbf{x}$ and $\mathbf{z}$ is:

$$\psi(\mathbf{x}) \cdot \psi(\mathbf{z})$$

However, based on the numbers we gave above (mapping from 100 dimensions to a trillion dimensions), the computation could become really slow if the dimension gets too high. We will talk about how to relieve this problem in SVM.

1.3 The kernel trick

We first revisit the mapping function described in Equation 5. Suppose now we modify the mapping function $\psi(\mathbf{x})$ as:

$$\mathbf{x} = (x_1, x_2) \mapsto \psi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2) \tag{8}$$

Since we only change the constant for some terms, this does not affect the hyperplane we can represent after mapping. However, the inner product for $\psi(\mathbf{x})$ and $\psi(\mathbf{z})$ becomes:

$$\begin{aligned}\psi(\mathbf{x})\cdot\psi(\mathbf{z}) &= 1+2x_1z_1+2x_2z_2+2x_1z_1x_2z_2+(x_1z_1)^2+(x_2z_2)^2\\ &= (1+x_1z_1+x_2z_2)^2\\ &= (1+\mathbf{x}\cdot\mathbf{z})^2\end{aligned} \tag{9}$$

That is, if original dimension is n, by adding all terms of degree at most k, we have

$$\psi(\mathbf{x})\cdot\psi(\mathbf{z}) = (1+\mathbf{x}\cdot\mathbf{z})^k = K(\mathbf{x},\mathbf{z}) \tag{10}$$

The computation complexity is now $O(n)$ instead of $O(n^k)$, and the computational problem can be relieved.

The result shown above indicates that it is possible to calculate the inner product in higher dimensional space using only the inner product in lower dimensional space under some specific mapping. Generally, we define a kernel function $K(\mathbf{x},\mathbf{z}) = \psi(\mathbf{x})\cdot\psi(\mathbf{z})$ for the mapping ψ with this property. By replacing the inner product $(\mathbf{x}_i\cdot\mathbf{x}_j)$ in Equation 1 with $K(\mathbf{x}_i,\mathbf{x}_j)$, we can obtain the higher dimensional hyperplane as the solution. This method is called the kernel trick. There are different kinds of kernels in practice. The kernel function shown in Equation 10 is called the polynomial kernel. Another popular kernel is Gaussian radial basis function (RBF) kernel $K(\mathbf{x},\mathbf{z}) = \exp\left(-c\cdot\|\mathbf{x}-\mathbf{z}\|^2\right)$, whose dimension is infinite. Another thing we should notice is that the input to the kernel function $\mathbf{x}$ and $\mathbf{z}$ are not necessarily vectors. For instance, $\mathbf{x}$ and $\mathbf{z}$ can be entirely different kinds of objects, such as strings or trees, as long as the kernel function provides the mapping of inner product for them.

In summary, our objective for solving SVM problem is to maximize the margin δ. When the data is linearly inseparable, we can deal with it by combining the kernel trick and the soft margin approach.

1.4 Comparison of SVM and boosting

We can now compare SVM with boosting. In SVM, we treat the input data as points in Euclidean space: $\mathbf{x}\in\mathbb{R}^n$. As discussed in the previous lecture, it is natural to assume $\|\mathbf{x}\|_2\le 1$. In boosting, we never really touch the data. Instead, what we manipulate are the weak hypotheses. To make things simple, suppose we use a finite weak hypothesis space $\mathcal{H} = \{g_1(x),\cdots,g_N(x)\}$ and the input can be viewed as the vector $\mathbf{h}(x) = \langle g_1(x),\cdots,g_N(x)\rangle$. Recall that the infinity norm for a vector $\mathbf{z}$ is defined as $\|\mathbf{z}\|_\infty = \max_j |z_j|$. Therefore, we have $\|\mathbf{h}(x)\|_\infty = \max_j |g_j(x)| = 1$ since $g_j(x)\in\{-1,+1\}$.

Next, we compare the coefficients to compute and the predictions. In SVM, the algorithm computes the unit normal vector $\mathbf{v}$ for the hyperplane, and the prediction is sign$(\mathbf{v}\cdot\mathbf{x})$. In boosting, the algorithm computes the coefficient $\alpha_t\ge 0$ for the weak hypothesis $h_t\in\mathcal{H}$. Suppose we run the boosting algorithm for T times, the prediction sign $\left(\frac{\sum_{t=1}^T\alpha_t h_t(x)}{\sum_t^T\alpha_t}\right)$ is a convex combination of h_t. Since each $h_t\in\mathcal{H} = \{g_1(x),\cdots,g_N(x)\}$, we can rewrite the prediction into another convex combination of $g_j(x)$ by finding the corresponding weight a_j:

$$\left(\frac{\sum_{t=1}^T\alpha_t h_t(x)}{\sum_t^T\alpha_t}\right) = \sum_{j=1}^N a_j g_j(x) = \mathbf{a}\cdot\mathbf{h}(x),$$

where $\mathbf{a} = \langle a_1,\cdots,a_N\rangle$. It should be noted that $\sum_{j=1}^N|a_j| = \|\mathbf{a}\|_1 = 1$, where $a_j\ge 0$. Therefore, the goal of boosting can be viewed as finding a_j for each g_j, and the prediction for sample x is sign$\left(\mathbf{a}\cdot\mathbf{h}(x)\right)$.

	SVM	AdaBoost
input	$\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_2 \leq 1$	$\mathbf{h}(x)$, $\|\mathbf{h}(x)\|_\infty = 1$
finds	$\|\mathbf{v}\|_2 = 1$	$\|\mathbf{a}\|_1 = 1$, $\mathbf{a} = \langle a_1, \cdots, a_N \rangle$
prediction	$\text{sign}(\mathbf{v} \cdot \mathbf{x})$	$\text{sign}\left(\mathbf{a} \cdot \mathbf{h}(x)\right) = \text{sign}\left(\sum_{j=1}^{N} a_j g_j(x)\right)$
margin	$y(\mathbf{v} \cdot \mathbf{x})$	$y\left(\mathbf{a} \cdot \mathbf{h}(x)\right)$

Figure 6: Comparison of SVM and Boosting.

Finally, we compare the margin of both SVM and boosting algorithms. The margin of SVM is $y\left(\mathbf{v} \cdot \mathbf{x}\right)$, while in boosting it is $y\left(\mathbf{a} \cdot \mathbf{h}(x)\right)$. The goal of both SVM and AdaBoost is to maximize the margin, but the norms that are used here are different: in SVM's, $\|v\|_2 = 1$ and $\|x\|_2 = 1$, while in boosting, $\|a\|_1 = 1$ and $\|h(x)\|_\infty = 1$.

The summary of the comparisons described above are listed in Figure 6.

2 Online Learning

2.1 Introduction

So far we have focused on the PAC learning model. We assume there is a fixed distribution for both the training and testing data, and the training samples are selected randomly. The algorithms we have discussed are batch learning algorithms, which means that after training, the hypothesis is fixed and then used for all future testing samples. Now we move on to online learning. The following are some properties of online learning. First, both training and testing happen at the same time in online learning. The learner gets one training sample at a time, makes the prediction, and then gets the true result as feedback. An example is to predict the stock market. In the morning the online learner makes a prediction about whether the price will go up or down, and then after one day it can receive the true situation and adjust future prediction. Second, the online learning algorithms tend to be simple. Third, online learning model makes no assumption about the distribution of the data, and can even completely drop the assumption that the data is generated randomly. In the following and future lectures, we will show that even without these assumptions, we can still analyze online learning algorithms in a meaningful way.

2.2 Learning with expert advice

We start from looking at an example of the stock market. Figure 7 gives the setting of the example. Suppose there are four experts who will make predictions for the price every morning, and the learner makes the prediction based on the four predictions. The goal of the learner is to provide the performance as good as, or at least not too much worse, than the best expert after a certain amount of time. This general setting is called "learning with expert advice" and is formulated as below:

```
$N = \#$ of experts
for $t = 1, \cdots, T$
    each expert $i$ predicts $\xi_i \in \{0, 1\}$ ,
    learner predicts $\hat{y} \in \{0, 1\}$ ,
    observe outcome $y \in \{0, 1\}$ (mistake if $\hat{y} \neq y$).
```

	Experts				Learner (Master)	Outcome
	1	2	3	4		
day 1	$\uparrow$	$\uparrow$	$\downarrow$	$\uparrow$	$\uparrow$	$\uparrow$
day 2	$\downarrow$	$\uparrow$	$\uparrow$	$\downarrow$	$\downarrow$	$\uparrow$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
# of mistakes	37	12	67	50	18	

Figure 7: The stock market example.

We would like to relate the number of mistakes of the learner to the number of mistakes of the best expert. However, the learner does not know which expert is the best. In the case assuming at least one expert is perfect, we can adopt a simple algorithm described as below:

$$\hat{y} = \text{ majority vote of experts with no mistakes so far}$$

This method is called the "Halving algorithm". To give a more concrete idea about how it works, we revisit the stock market example in Figure 7. In the first day, expert 3 made a mistake, so the learner does not take the prediction of expert 3 into account starting from the second day.

Now we can calculate the mistake bound of the Halving algorithm as follows. Let W be the number of experts that make no mistake so far, or we say the number of *active* experts. Initially we have $W = N$. After the learner made one mistake, we have $W \leq \frac{1}{2}N$ because there are at least half of the active experts that made this mistake. Similarly, after the learner made the second mistake, $W \leq \frac{1}{4}N$, and so on. After the learner made m mistakes, we have $W \leq \frac{1}{2^m}N$. Due to the assumption that at least one expert is perfect, we have $W \geq 1$. That is,

$$1 \leq W \leq \frac{1}{2^m}N \Rightarrow m \leq \lg(N).$$

Finally, we can consider a special case in which we view each expert as one hypothesis. Suppose we have a hypothesis space $\mathcal{H} = \{h_1, \cdots, h_N\}$, and the target concept $c \in \mathcal{H}$. By adopting the Halving algorithm, each round the learner gets one sample x, makes the prediction $\hat{y} \in \{0, 1\}$, and then observes the true result $y = c(x)$. We then have the following mistake bound:

$$\text{\# of mistakes } \leq \lg(N) = \lg(|\mathcal{H}|).$$