# COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Athindran Ramesh Kumar

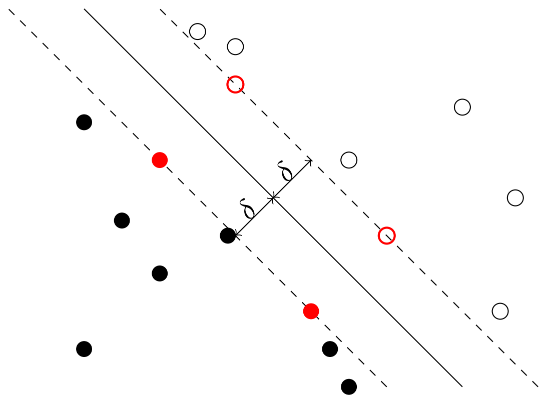
Lecture #14
March 27, 2019

## 1 Recap: SVM for linearly separable data

In the previous lecture, we developed a method known as the support vector machine for obtaining the maximum margin separating hyperplane for data that is linearly separable, i.e., there exists at least one hyperplane that perfectly separates the positive and negative labeled points. The inspiration for seeking maximum margin classifiers arose from our observation in boosting that increasing the number of rounds of boosting increased the margin of our weighted majority classifier and hence decreased the test error. Further, we note that:

- VC-dim (Linear threshold function with margin $\delta$) $\leq \frac{1}{\delta^2}$ if all training and test examples have $\|\mathbf{x}\| \leq 1$
- VC-dim (Linear threshold function with margin $\delta$) $\leq \left(\frac{R}{\delta}\right)^2$ if all training and test examples have $\|\mathbf{x}\| \leq R$

In other words, the VC-dimension of the set of linear threshold functions with margin $\delta$ is bounded above by a quantity independent of the dimension of the space when the training and test examples have length at most $R$. The VC-dimension decreases as the margin increases. This gives us further motivation to explicitly find maximum margin classifiers.



The above figure shows a linearly separable set of points along with a maximum margin separating hyperplane with margin $\delta$.

We are given $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \{-1, +1\}$. The primal problem for finding the maximum margin hyperplane can be formulated as:

$$\begin{aligned} &\max \delta \\ \forall i \; &y_i(\mathbf{v} \cdot \mathbf{x}_i) \geq \delta \\ &\|\mathbf{v}\|_2 = 1 \end{aligned}$$

By introducing a new variable, $\mathbf{w} = \frac{\mathbf{v}}{\delta}$, we transformed the problem as:

$$\min \frac{\|\mathbf{w}\|_2^2}{2}$$
$$\forall i \; y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1$$

$\mathbf{w}$ can be obtained from the training examples (specifically, the support vectors among the training examples) as $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$.

The $\alpha_i$'s are obtained by solving the dual problem formulated below:

$$\max \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$
$$s.t \; \forall i \; \alpha_i \geq 0$$

The dual problem can be solved by any non-linear programming method such as gradient descent.

For predicting the label of a new example $\mathbf{x}$, we use:

$$\begin{aligned} h(\mathbf{x}) &= sign(\mathbf{w} \cdot \mathbf{x}_i) \\ &= sign(\sum_i \alpha_i y_i \mathbf{x} \cdot \mathbf{x}_i) \end{aligned}$$

### Observations

- A crucial observation to note is that only inner products between each pair of training examples and each training example and test example is required for predicting the labels of the test examples. This observation will be used later in the class.
- Another observation is that $\mathbf{w}$ is a linear combination of only the support vectors (the training examples with $\alpha_i > 0$). Hence, the maximum margin separating hyperplane depends only on the support vectors.

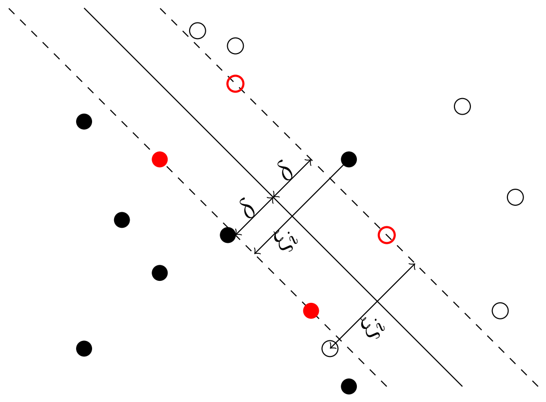
The SVM has had a big impact on machine learning. In the SVM, we are able to formulate exactly what we want to optimize and solve the optimization problem fairly easily.

## 2 SVM for data that is not linearly separable

There are two strategies for dealing with linearly inseparable data. Both the strategies are often combined for practical applications.

### 2.1 Soft-margin SVM

This strategy can be used in the scenario when the data is almost linearly separable, i.e., there are a few data points that violate the requirement as shown in the figure below.



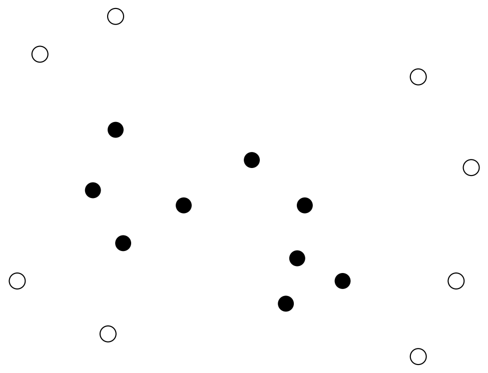
The strategy is to allow the learning algorithms to move the points that are wrongly classified as little as possible in order to make the data linearly separable with maximum possible margin. We introduce variable $\xi_i$ to represent the amount by which each point $i$ has to be moved. Now, we can formulate the problem as:

$$\min \frac{\|\mathbf{w}\|_2^2}{2} + C\sum_i \xi_i$$
$$\forall i \ y_i(\mathbf{w}\cdot\mathbf{x}_i) + \xi_i \geq 1$$
$$\forall i \ \xi_i \geq 0$$

Note that we try to maximize the margin and also minimize the amounts by which each wrongly classified point is moved by using a weighted combination of the two in the objective. The parameter $C$ can be tuned as needed. This problem can be solved using a similar technique by formulating the Lagrangian, writing the KKT conditions and solving the dual problem.

## 2.2 Kernel SVM

Another technique to solve the problem of linearly inseparable data is to project the data into a higher dimensional space in which the data is linearly separable. For example consider the set of points as shown below:



These set of points are neither linearly separable nor almost linearly separable.

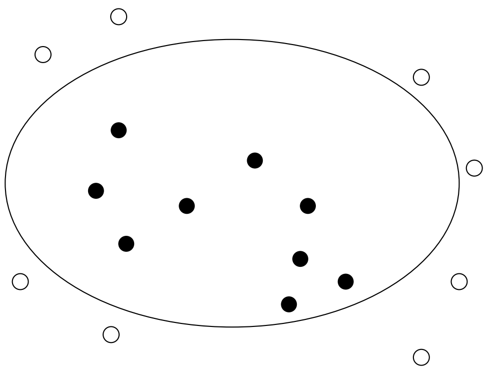
Suppose we define a mapping $\boldsymbol{\psi}$ from $\mathbb{R}^2$ to $\mathbb{R}^6$ as shown below:

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

Now, the equation for the separating hyperplane in 6 dimensions is given by:

$$\mathbf{v} = (a, b, c, d, e, f)$$
$$\mathbf{v} \cdot \boldsymbol{\psi}(\mathbf{x}) = 0$$
$$a + bx_1 + cx_2 + dx_1x_2 + ex_1^2 + fx_2^2 = 0$$

In the mapped space, $\mathbb{R}^6$, this is the equation of a hyperplane. In the original space, $\mathbb{R}^2$, this is the equation of an arbitrary conic section. This conic section can separate the set of points in two dimensions as shown below.



Thus, after mapping the space to a higher dimensional space, we have made the data linearly separable. Now, we can use the basic SVM in this high dimensional space. This idea generalizes. In general, if we have $n$ dimensions, we can form all monomial terms up to degree $k$. This would give rise to a higher dimensional space of dimensionality $O(n^k)$. Is this a good idea? Let us discuss further.

## 2.3 Problems with this idea

### Problem 1

As the dimension of the space we map to increases, it would seem that we will need substantially more data, especially if mapping to a very high dimensional space. However, note that we said that VC-dimension of linear threshold functions with margin $\delta$ is at most $\left(\frac{R}{\delta}\right)^2$ if $\|\mathbf{x}\| \leq R$. This indicates that the VC-dimension does not scale with the number of dimensions and depends only on the margin. However, $R$ could still increase as we increase the number of dimensions. Still, at least the VC-dimension does not explicitly depend on the dimension of the space we are mapping to, so the amount of training data needed will not necessarily increase.

### Problem 2

The main problem with this idea is the computational burden incurred by operating in this high-dimensional space. We discussed earlier that we would be operating in a space of dimensionality $O(n^k)$. If $n = 100$ and $k = 6$, we would be working with a trillion dimensions. This is a huge computational problem and requires gigantic time and memory. This problem is overcome by using the kernel trick.

### 2.4 Kernel trick

Note that the mapping $\boldsymbol{\psi}$ from $\mathbb{R}^2$ to $\mathbb{R}^6$ is given by:

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

We tweak the mapping by multiplying benign constants for convenience.

$$\mathbf{x} = (x_1, x_2) \mapsto (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2) = \boldsymbol{\psi}(\mathbf{x})$$
$$\mathbf{z} = (z_1, z_2) \mapsto (1, \sqrt{2}z_1, \sqrt{2}z_2, \sqrt{2}z_1z_2, z_1^2, z_2^2) = \boldsymbol{\psi}(\mathbf{z})$$

An observation we made regarding SVM's is that only the inner product of the examples are used in the algorithm. Let us compute the inner product $\boldsymbol{\psi}(\mathbf{x}) \cdot \boldsymbol{\psi}(\mathbf{z})$.

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

More generally, when we start in $\mathbb{R}^n$ and add all terms up to degree $k$, we can write the inner product as:

$$\boldsymbol{\psi}(\mathbf{x}) \cdot \boldsymbol{\psi}(\mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^k$$

What the above equation implies is that we do not need to blow up to higher dimensions to compute the inner product in the high-dimensional space but we can compute the inner product in the low-dimensional space itself. This observation is known as the *kernel trick*. In other words, instead of finding the mapping to the higher dimension $\boldsymbol{\psi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{z})$ and computing the inner product in the higher dimensions, we can compute the inner product $\boldsymbol{\psi}(\mathbf{x}) \cdot \boldsymbol{\psi}(\mathbf{z})$ through some operation on $\mathbf{x}$ and $\mathbf{z}$. In fact, we can define, design and use the kernel $(\boldsymbol{\psi}(\boldsymbol{x}) \cdot \boldsymbol{\psi}(\boldsymbol{z}))$ directly without bothering about the higher dimensional space. The map can even be infinite dimensional.

In general, a kernel is defined as a real-valued function $K(x, z)$ satisfying certain properties known as the Mercer conditions (namely, for any set of points $x_i$, the matrix $M : M_{i,j} = K(x_i, x_j)$ is symmetric and positive definite). Some common kernels include:

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^k \text{ (Polynomial kernel)}$$
$$K(\mathbf{x}, \mathbf{z}) = \exp(-c\|(\mathbf{x} - \mathbf{z})\|_2^2) \text{ (Radial basis kernel)}$$

The dual problem of the SVM can now be reformulated as:

$$\max \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$
$$s.t\ \forall i\ \alpha_i \geq 0$$

For predicting the label of a new example $\mathbf{x}$, we use:

$$h(\mathbf{x}) = sign\left(\sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i)\right)$$

The design of kernels is more of an art and domain knowledge also comes into play. We note that if we use the polynomial kernel, we have to decide the parameter $k$. As $k$ increases, both the margin $\delta$ and the term $R$ in the VC-dimension increase. We can see that the performance will improve up to some $k$ and decrease thereafter. This thus reminds us of the phenomenon of overfitting.

## 3 Relationship between boosting and SVM

We looked at boosting and SVM in the past few classes. Both were related to the margin of the classifier obtained. In this section, we will discuss some of the similarities and differences between boosting and SVM. First, we will look at boosting in a new light.

In the boosting algorithm, we were primarily concerned with the weak learning algorithm and the weak learning hypotheses. Assume that the space $\mathcal{H}$ of hypotheses available to the weak learning algorithm is finite, i.e., $\mathcal{H} = \{g_1, \ldots, g_N\}$ where $g_j$ represents a weak hypothesis. For a given $\mathbf{x}$ we denote $\mathbf{h}(\mathbf{x}) = \langle g_1(\mathbf{x}), \ldots g_N(\mathbf{x})\rangle$. Note that:

$$\|\mathbf{h}(\mathbf{x})\|_\infty = \max_j |g_j(\mathbf{x})| = 1$$

The resulting boosting classifier with $T$ rounds of boosting is given by:

$$\begin{aligned} f(\mathbf{x}) &= sign\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right) \\ &= sign\left(\frac{\sum_t \alpha_t h_t(\mathbf{x})}{\sum_t \alpha_t}\right) \\ &= sign\left(\sum_{j=1}^{N} a_j g_j(\mathbf{x})\right) \quad a_j \geq 0, \sum_j a_j = 1 \\ &= sign\left(\mathbf{a} \cdot \mathbf{h}(\mathbf{x})\right) \end{aligned}$$

The output classifier is a weighted majority of the weak learning classifiers obtained from the boosting algorithm. Note that these weak learning classifiers are from $\mathcal{H}$. We can normalize the coefficients of these classifiers and make the coefficients of the other classifiers in $\mathcal{H}$ equal to 0 to obtain the expression above. Said differently, the convex combination of $h_t$'s produced by boosting can be rewritten, as above, as a convex combination of all the $g_j$'s, where each weak hypothesis $g_j$ gets weight $a_j$. Let $\mathbf{a} = \langle a_1, \ldots, a_N\rangle$. Observe that:

$$\|\mathbf{a}\|_1 = \sum_j |a_j| = 1.$$

Now, we are ready to compare the boosting and SVM methods.

| | SVM | Adaboost |
|---|---|---|
| Examples | $\|\mathbf{x}\|_2 \leq 1$ | $\|\mathbf{h}(\mathbf{x})\|_\infty = 1$ |
| Find | $\|\mathbf{v}\|_2 = 1$ | $\|\mathbf{a}\|_1 = 1$ |
| Predict | $sign(\mathbf{v} \cdot \mathbf{x})$ | $sign(\mathbf{a} \cdot \mathbf{h}(\mathbf{x}))$ |
| Margin | $y(\mathbf{v} \cdot \mathbf{x})$ | $y(\mathbf{a} \cdot \mathbf{h}(\mathbf{x}))$ |

Note that the two problems look very similar except for the fact that the norms on some of the terms are different. However, the solution methods for the two problems are completely different. This will not be the last time in the class we note that a difference in the norm can lead to totally different solutions and algorithms with quite different properties.

## 4 Online learning

This is the start of a major shift in the class. Until now, we have been looking at PAC learning and its variants. In other words, the algorithm gets a batch of random examples and spits out a hypothesis that generalizes and performs well on the test data with some guarantees.

Now, we are going to shift to the paradigm of online learning. In *online learning*, the algorithm gets one example at a time, makes a prediction and finds out whether the prediction is correct or not. Thus, training and testing happen at the same time. The algorithm is judged by the number of mistakes it makes online. Further, we do away with the assumption that the data is random. In fact, the data can even be adversarial. One might think that making such assumptions can make the problem incredibly hard. However, we will obtain simple algorithms with simple analysis techniques for online learning. These results are obtained without giving up on the performance or compute requirement of the algorithms.

## 4.1 Prediction from expert advice

One fundamental online learning problem is prediction from expert advice. The learner has to make a prediction at each time step. For making this prediction, the learner gets the suggestions from $N$ experts. After making a prediction, the learner observes whether the prediction was correct or wrong. For example, each morning, we might want to predict whether the stock market will go up or down that day, based on the predictions of experts. Then, in the evening, we find out if our prediction was correct or not. The goal is to make as few mistakes as possible. The configuration for predicting whether the stock market will go up or down based on the advice from 4 experts is shown in the table below.

| | Experts | | | | Learner | Outcome |
|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | (master) | |
| Day 1 | $\uparrow$ | $\uparrow$ | $\downarrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| Day 2 | $\downarrow$ | $\uparrow$ | $\uparrow$ | $\downarrow$ | $\downarrow$ | $\uparrow$ |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| . | | | | | | |
| No. of mistakes | 37 | 12 | 67 | 50 | 18 | |

The problem of prediction from expert advice can be summarized as follows:
$N = \#$ Experts
for $t = 1 \ldots T$

- Each expert $i$ predicts $\xi_i \in \{0,1\}$.
- Learner predicts $\hat{y} \in \{0,1\}$.
- Observe outcome $y \in \{0,1\}$
- Mistake if $\hat{y} \neq y$

At the very least in this setting, we should hope that the learner does not make too many more mistakes than the best expert. The experts could be humans or simple prediction rules or learning algorithms.

## 4.2 Halving algorithm and Bounds

Assume that there exists an expert who makes no mistakes. In this setting, we can use the halving algorithm and bound the number of mistakes made by the algorithm. In the *halving algorithm*, we no longer listen to an expert once it makes a mistake. We make a prediction at each time as the majority vote of the surviving experts (that is, experts who have made no mistakes so far).

$$\hat{y} = \text{Majority vote of predictions of surviving experts}$$

Let $W$ be the number of surviving experts. We know that $W \geq 1$ as there is one expert who will never make any mistake. Initially $W = N$. After 1 mistake $W \leq \frac{1}{2}N$. This is because the learner makes the decision based on majority vote, so if the learner makes a mistake, then at least half of the experts have made a mistake and will drop out. Similarly, after 2 mistakes $W \leq \frac{1}{4}N$. And after $m$ mistakes $W \leq \frac{1}{2^m}N$. Therefore,

$$1 \leq W \leq \frac{1}{2^m}N$$

This implies that:

$$m \leq \lg(N)$$

Thus, without making any statistical assumptions on the data, we were able to get an upper bound on the number of mistakes made by the algorithm. However, we made the strong assumption that there exists at least one expert who makes no mistake.