COS 511: Theoretical Machine Learning

Lecturer: Rob Schapire
Scribe: Pierre Bayle

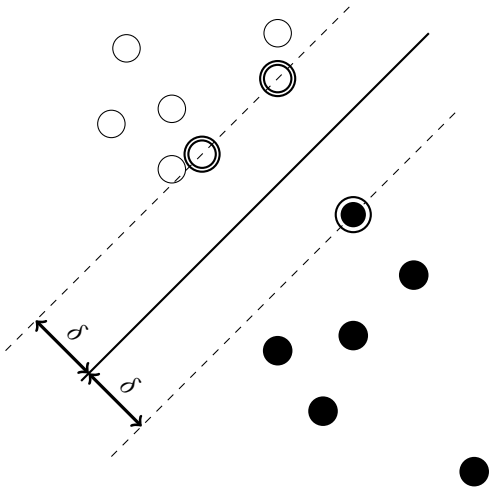
Lecture #14
March 28, 2018

1 Support Vector Machines

Let's recall that the goal of Support Vector Machines is to find a separating hyperplane whose margin is as big as possible.

Given $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ where $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \{-1, +1\}$, the goal is to find $\mathbf{v} \in \mathbb{R}^n$ to solve

$$\begin{array}{ll} \max & \delta \\ \text{subject to} & y_i(\mathbf{v} \cdot \mathbf{x}_i) \geq \delta, \ i = 1, \ldots, m \\ & \|\mathbf{v}\|_2 = 1 \end{array}$$



The above illustration of an SVM is based on an illustration found online [1].

We have the following result in terms of VC dimension.

- VC-dim (linear threshold functions with margin δ) $\leq \frac{1}{\delta^2}$ if $\|\mathbf{x}_i\|_2 \leq 1, \forall i$.
- VC-dim (linear threshold functions with margin δ) $\leq \left(\frac{R}{\delta}\right)^2$ if $\|\mathbf{x}_i\|_2 \leq R, \forall i$.

1.1 Primal problem

We saw in the previous lecture that the previous maximization problem can be written as

$$\begin{array}{ll} \min\limits_{\mathbf{w}} & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ \text{subject to} & y_i(\mathbf{w} \cdot \mathbf{x}_i) \geq 1, \ i = 1, \ldots, m \end{array}$$

[1] https://elbauldelprogramador.com/en/creating-trees-dependency-graphs-svms-in-tikz/

This is called the primal problem.
In terms of the previous problem, $\mathbf{w}$ is defined as $\mathbf{w} = \frac{\mathbf{v}}{\delta}$.

1.2 Dual problem

We saw that this problem can be written in terms of other variables

$$\begin{aligned} \max_{\alpha_i} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) \\ \text{subject to} \quad & \alpha_i \geq 0,\ i = 1, \ldots, m \end{aligned}$$

$\mathbf{w}$ of the primal problem can then be computed as $\sum_i \alpha_i y_i \mathbf{x}_i$.

1.3 Output hypothesis

The output hypothesis is defined as

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

1.4 Observations

There are two observations to be made:

- Thanks to the dual problem we can compute the vector $\mathbf{w}$ as a linear combination of the examples. The analysis carried out in the last class even showed that $\mathbf{w}$ is a linear combination of the support vectors (which are the only examples $\mathbf{x}_i$ for which the corresponding dual variables α_i are non-zero).
- In the dual problem the examples only appear via their inner products. We only need to know how to compute the inner products. We will give more explanation later.

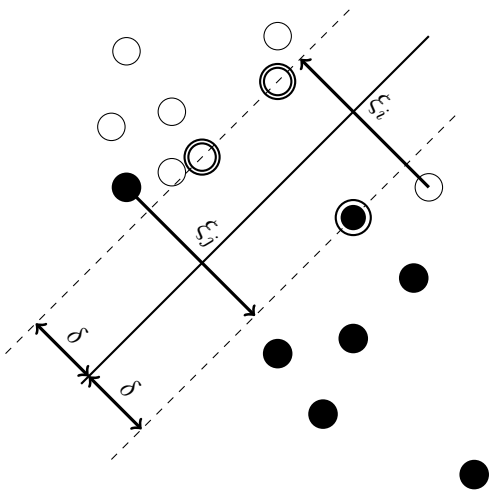
2 Non-linearly separable case

So far we only studied the case where the data was assumed to be linearly separable, that is to say that there exists a separating hyperplane.

However with a real dataset there is no guarantee that our data is linearly separable. What happens if we remove this assumption? How can SVM be used in that case? We will talk about two ways to deal with this.

2.1 Soft-margin approach

To have a first understanding of this approach, let's assume that the data is "almost" linearly separable.



This illustration is based on the same online illustration as before.

The idea is that we allow the algorithm to move a few data points if necessary. In the below optimization problem, for a given i, ξ_i represents the distance that an example x_i would need to be moved to have margin at least 1. The idea is then to minimize both our usual objective $\frac{1}{2}\|\mathbf{w}\|_2^2$ and the sum $\sum_i \xi_i$.

The optimization problem is

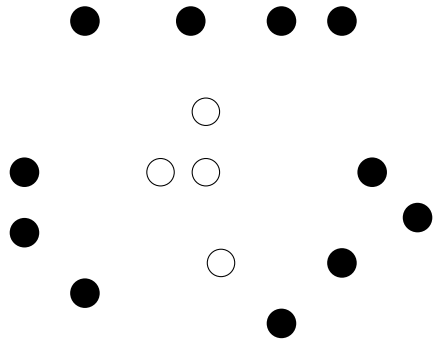
$$\begin{aligned} \min_{\mathbf{w}} \quad & \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_i \xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}\cdot\mathbf{x}_i) \geq 1-\xi_i,\ i=1,\ldots,m \\ & \xi_i \geq 0,\ i=1,\ldots,m \end{aligned}$$

The second term is a penalization term, and C a constant.

We won't focus that much on this approach, but rather on the following one.

2.2 Projection into a higher dimensional space

If we have to deal with the kind of data displayed in the figure below, we see that it is not possible to linearly separate our data.



The idea is that by projecting the data set into a higher dimensional space, the data set can be linearly separable in this new space and we can use the simple SVM in this new space.

For example, with $\mathbb{R}^2$ being the original space, we can define the following mapping F from $\mathbb{R}^2$ to $\mathbb{R}^6$:

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

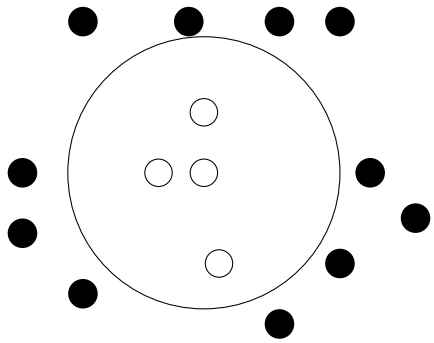
Actually, a "useful" mapping would have other constants in front of the terms x_1, x_2 and x_1x_2, as we will see later, but let's assume for the moment that we use F as defined above.

An SVM in this new space gives a vector $\mathbf{v} = (a, b, c, d, e, f)$ that defines the following hyperplane in $\mathbb{R}^6$:

$$\{\mathbf{x} \in \mathbb{R}^6 : 0 = \mathbf{v} \cdot F(\mathbf{x}) = a + bx_1 + cx_2 + dx_1x_2 + ex_1^2 + fx_2^2\}$$

In the new space $\mathbb{R}^6$, this set is a hyperplane. In terms of the initial two-dimensional space $\mathbb{R}^2$, this set is a conic section. For instance it can be an ellipse, or a circle.

Therefore this approach can help us separate the data we had before:



In general, we can start with an n-dimensional space and use mappings whose output vectors are all the monomials of degree smaller than or equal to k. Therefore we map to a space of dimension $\mathcal{O}(n^k)$.

However, using SVM in the higher dimensional space without giving more thought about it is a terrible idea, for two main reasons:

- Intuitively, the amount of data we need is getting bigger with the dimension. This is the curse of dimensionality, a statistical problem.
- The computational time and the storage are also getting bigger. This is a computational problem. For instance, beginning with a 100-dimensional space and considering all the monomials of degree smaller than or equal to 6, we end up with vector of size $\mathcal{O}(100^6)$ in the higher dimensional space.

But SVM can get rid of these two problems:

- Concerning the statistical problem, we have two different ways to analyze SVM's. Both ways suggest that increasing the dimension with the previous mappings won't necessarily be a problem, even if the increase can be gigantic.
  - In terms of support vectors: the number of support vectors doesn't depend explicitly on the dimension. It could even happen that we need fewer support vectors when increasing the dimension.

– In terms of VC dimension: the bound $\left(\frac{R}{\delta}\right)^2$ doesn't depend explicitly on the dimension.

- Concerning the computational problem, we can notice that the only operation on vectors is the inner product. Let's explain it further below.

We are faced with the computation of $F(\mathbf{x}) \cdot F(\mathbf{z})$ for $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$. The obvious thing to do would be to compute $F(\mathbf{x}), F(\mathbf{z})$ and then to compute their inner product. But as said above this can be very time and memory-consuming. To solve this issue, we introduce the *kernel trick*. First, we introduce it in $\mathbb{R}^2$. As said above, we change the mapping F by multiplying some components by constants.

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

The hyperplane defined by $\{\mathbf{x} \in \mathbb{R}^6 : 0 = \mathbf{v} \cdot F(\mathbf{x})\}$ is exactly the same as before.
Now let's compute $F(\mathbf{x}) \cdot F(\mathbf{z})$.

$$\begin{aligned} F(\mathbf{x}) \cdot F(\mathbf{z}) &= 1 + 2x_1z_1 + 2x_2z_2 + 2x_1x_2z_1z_2 + x_1^2z_1^2 + x_2^2z_2^2 \\ &= (1 + x_1z_1 + x_2z_2)^2 \\ &= (1 + \mathbf{x} \cdot \mathbf{z})^2 \end{aligned}$$

This is what is called the *kernel trick*. This is really important in terms of computational savings: in order to compute $F(\mathbf{x}) \cdot F(\mathbf{z})$, we only need to compute the inner product in the original space $\mathbf{x} \cdot \mathbf{z}$, add 1, and square the result. We never have to explicitly map to the high-dimensional space.

This "trick" generalizes.
For n a positive integer and a suitable choice of F, we can have, for $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$

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

In general, a kernel is a real-valued function $K : (\mathbf{x}, \mathbf{z}) \mapsto K(\mathbf{x}, \mathbf{z})$ defined on pairs and satisfying the Mercer conditions:

- K is symmetric

- K is positive definite

For example, $(1 + \mathbf{x} \cdot \mathbf{z})^k$ defines a polynomial kernel and $\exp(-c\|\mathbf{x} - \mathbf{z}\|_2^2)$ defines a Gaussian radial basis function kernel. The class of kernels is very large: other kernels have be defined on entirely different spaces, such as strings, trees, and lots more.

When mapping in a higher dimensional space, δ and R will both tend to get larger. Therefore, to bound the VC dimension with the formula given previously, we see that there is a trade-off between δ and R.

In practice, we can start with a given k (recall that this k is the highest degree of the monomials considered), for instance $k = 1$, and increase it sequentially. We would see that performance first improves and then degrades.

3 Comparison of SVM and Boosting

We will compare SVM's and Boosting in order to find their similarities and differences. First we need to rewrite a few results in the context of Boosting so that they look like what we get in the context of SVM's.

A Boosting algorithm never "touches" the examples directly, but instead, only interacts with the data by way of the weak learning algorithm and the weak hypotheses. So in a sense, the data, from the Boosting algorithm's prospective, is given by the predictions of the weak hypotheses on the data. This motivates us to focus on the predictions of the weak hypotheses.

Assume that the hypothesis space $\mathcal{H}$ of all possible weak hypotheses is finite. We can list all the elements of this space:

$$\mathcal{H} = \{g_1, \ldots, g_N\}$$

Following the motivation written above, we can define the following vector $\mathbf{h}(x) \in \{-1, +1\}^N$ for x an example

$$\mathbf{h}(x) = \langle g_1(x), \ldots, g_N(x) \rangle$$

We have $|g_j(x)| = 1, \forall j$, so $\max_j |g_j(x)| = 1$, that is to say $\|\mathbf{h}(x)\|_\infty = 1$.

During Boosting, we find $\alpha_1, \ldots, \alpha_T, h_1, \ldots, h_T$. The prediction associated to a vector x is then sign $\left(\frac{\sum_t \alpha_t h_t(x)}{\sum_t \alpha_t}\right)$. This prediction can be rewritten, for some $a_1, \ldots, a_N$, as sign $\left(\sum_{j=1}^N a_j g_j(x)\right)$, where the g_j are the elements of $\mathcal{H}$. Notice that the majority of the coefficients a_j are likely to be zero.

Let's define $\mathbf{a} = \langle a_1, \ldots, a_N \rangle$. The prediction above can then be written as $\text{sign}(\mathbf{a} \cdot \mathbf{h}(x))$. We have $a_j \geq 0, \forall j$, and $\sum_j a_j = 1$. Therefore $\|\mathbf{a}\|_1 = 1$.

Recall from previous lectures that in the case of Boosting, the margin is equal to $yf(x)$ where $f(x)$, in the notation above, can be written as $\mathbf{a} \cdot \mathbf{h}(x)$. Therefore the margin is $yf(x) = y(\mathbf{a} \cdot \mathbf{h}(x))$.

Thanks to these remarks, we can now compare SVM and Boosting:

	SVMs	Boosting
inputs	$\|\mathbf{x}\|_2 \leq 1$	$\|\mathbf{h}(\mathrm{x})\|_\infty = 1$
we find	$\|\mathbf{v}\|_2 = 1$	$\|\mathbf{a}\|_\mathbf{1} = \mathbf{1}$
prediction	$\text{sign}(\mathbf{v} \cdot \mathbf{x})$	$\text{sign}(\mathbf{a} \cdot \mathbf{h}(\mathbf{x}))$
margin	$y(\mathbf{v} \cdot \mathbf{x})$	$y(\mathbf{a} \cdot \mathbf{h}(\mathbf{x}))$

SVM and Boosting are very similar, the main differences are the norms:

- SVM is related to the Euclidean norm $\|.\|_2$.
- Boosting is related to the norms $\|.\|_\infty$ and $\|.\|_1$.

4 The online learning model

4.1 Introduction

For much of the rest of the course, we will focus in the lectures on a completely different topic: the online learning model. In this lecture we will introduce it.

Previously in the class, we were focusing on algorithms that compute a hypothesis after receiving a batch of random examples. This hypothesis is "frozen", in the sense that it doesn't change after receiving this batch of examples.

In the online learning model, we get one example at a time, we then make a prediction and get feedback, then we get a second example, and so on. Training and testing happen at the same time. In comparison with other learning models, the algorithms tend to be more simple in the online learning model, even if it is becoming less true nowadays. One very important point about the online learning model is that we can analyze it without any statistical assumption about the data or even with adversarial data.

An example of such a configuration is when everyday we want to predict whether the stock market is going up or down on this particular day, and do this every day. At the beginning of every day, we predict whether it is going to be up or down. At the end of the day, we see the real outcome: this is the feedback. We can imagine a similar configuration for weather forecasting.

4.2 Learning with expert advice

An example of online learning is learning with expert advice. The "experts" can be people, simple prediction rules, algorithms, etc.

N is the number of experts.

For $t = 1, \ldots, T$:

- each expert makes a prediction $\xi_i \in \{0, 1\}$
- the learner predicts $\hat{y} \in \{0, 1\}$ based on the predictions of the experts
- the learner observes the actual outcome $y \in \{0, 1\}$
- there is a mistake if $\hat{y} \neq y$

The goal for the learner (also called master) is to make as few mistakes as possible, even in comparison with the best expert, and without any assumption about how the data is generated.

For example, if the learner wants to predict whether a particular stock is going to be up or down at the end of every day, with expert advice, the configuration looks like the following.

	experts				learner(master)	actual outcome
	1	2	3	4		
day 1	↑	↑	↓	↑	↑	↑
day 2	↓	↑	↑	↓	↓	↑
⋮						
number of mistakes	37	12	67	50	18	

Let's see what can we say about the number of mistakes when we assume that there is at least one expert that makes no mistake at all.

We assume that there exists an expert that is never wrong, so as soon as one expert makes a mistake, the learner should never listen to that expert again. Then assume that at each round, the learner makes its prediction by taking the majority vote of the remaining experts. This is the *halving algorithm*.

Define W as the number of experts that didn't make any mistake so far. W will change throughout time. Initially, $W = N$ the total number of experts. And $W \geq 1$ is always true because we assume that there is at least one never mistaken expert.

If the learner makes a mistake, then at least one half (because the learner is using a majority vote) of the remaining experts made a mistake at this round: we don't listen to them anymore.

Therefore, after m mistakes, we have $W \leq \left(\frac{1}{2}\right)^m N$. So we can bound W.

$$1 \leq W \leq \left(\frac{1}{2}\right)^m N$$

From $1 \leq \left(\frac{1}{2}\right)^m N$, we deduce

$$m \leq \log_2 N$$

Without any stochastic assumption, we were able to bound the total number of mistakes made by the learner.