6.883: Online Methods in Machine Learning

Alexander Rakhlin

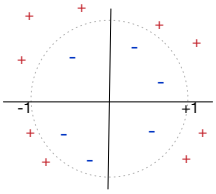
LECTURE 8

This lecture is partly based on chapter 16 in [SSBD14].

1. KERNELS

Last week we started discussing non-linear classifiers: we introduced nonlinearities by composing (possibly several times) matrix multiplication and a "transfer function" σ. The resulting structure can be viewed as a network. The effective classification boundary is highly nonlinear.

In this lecture we discuss a second popular approach to building a nonlinear classification boundary, a *kernel method*. As already mentioned last week, we will attempt to replace $\langle w, x\rangle$ with $\langle w, \phi(x)\rangle$ for some mapping ϕ into a possibly high-dimensional (or even infinite-dimensional) space. Why is this a good idea? Consider the 2d data



Clearly, these data are not separable by a hyperplane. However, the classes are exactly described by

$$\text{sign}(x[1]^2 + x[2]^2 - 1)$$

where $x[1]$ and $x[2]$ are coordinates of a data point $x \in \mathbb{R}^2$. Suppose we map each data point $x \in \mathbb{R}^2$ to $\phi(x) = (1, x[1]^2, x[2]^2) \in \mathbb{R}^3$. Then the hyperplane $w = (-1, 1, 1)$ has zero error on the data in this new space. Of course, by taking degree high enough, any fixed amount of data can be made linearly separable in that high-dimensional space. We see that building a perfect predictor of the given (training) data is not by itself necessarily a good idea. This issue of model complexity and overfitting will be discussed later.

1.1 A convex problem

While the classification boundary induced by the kernel mapping is non linear, the optimization problem can still be convex. Let's look at the SVM objective for binary classification with x replaced by $\phi(x)$:

$$\min_w \ \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i \langle w, \phi(x_i)\rangle\} + \lambda \|w\|^2 \tag{1}$$

where w ranges over the space in which $\phi(x)$ lives. We could even take this space to be an infinite dimensional space, as long as inner product makes sense there (modulo technicalities, this is known as a Hilbert space).

The claim is that an optimal w in (1) belongs to the span of n vectors $\{\phi(x_1), \ldots, \phi(x_n)\}$. Indeed, if it does not, there is a component orthogonal to the span. This component does not affect the loss on the data (since it is orthogonal to any $\phi(x_i)$), yet increases the norm. Hence, for any solution, one obtains a better solution by removing this orthogonal component. In fact, the argument holds for any optimization problem of the form

$$\min_w \ \Phi(\langle w, \phi(x_1)\rangle, \ldots, \langle w, \phi(x_n)\rangle) + \Psi(\|w\|) \tag{2}$$

for a nondecreasing Ψ.

This simple observation has profound consequences. In particular, it says that the minimization problem over the (possibly) infinite dimensional space reduces to a finite-dimensional problem, since we only need to search for vectors of the form

$$w = \sum_{j=1}^{n} \alpha_j \phi(x_j).$$

The vector of coefficients α is n-dimensional, and the optimization problem (1) takes on the form

$$\min_\alpha \ \frac{1}{n}\sum_{i=1}^{n} \max\left\{0, 1 - y_i \left\langle \sum_{j=1}^{n} \alpha_j \phi(x_j), \phi(x_i)\right\rangle\right\} + \lambda \Big\| \sum_{j=1}^{n} \alpha_j \phi(x_j)\Big\|^2 \tag{3}$$

By defining $K(x_i, x_j) = K_{i,j} = \langle \phi(x_i), \phi(x_j)\rangle$, we rewrite the problem as

$$\min_\alpha \ \frac{1}{n}\sum_{i=1}^{n} \max\left\{0, 1 - y_i \sum_{j=1}^{n} \alpha_j K_{i,j}\right\} + \lambda \sum_{i,j=1}^{n} \alpha_i \alpha_j K_{i,j} \tag{4}$$

or, in matrix notation,

$$\min_\alpha \ \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i (K\alpha)_i\} + \lambda \alpha^\mathsf{T} K \alpha. \tag{5}$$

1.2 No need to compute $\phi(x)$

Curiously enough, the optimization problem (5) no longer depends on the mapping ϕ explicitly. Instead, it depends on K, the matrix of inner products $\langle \phi(x_i), \phi(x_j)\rangle$. This matrix K (known as the Gram matrix) is $n \times n$, which may be much smaller than the matrix $[\phi(x_1), \ldots, \phi(x_n)]$. Computing/storing the K matrix is, in many cases, much easier than computing the mappings $\phi(x_i)$. The SVM objective only depends on the inner products in the high-dimensional space, and not on the actual vectors. This observation applies to many machine learning methods where data enters through inner products with the parameter vector; many of these methods can be "kernelized" (this was a hot topic about 15 years ago, but this door is now pretty much closed).

One then asks what K one may choose. A mapping ϕ gives rise to K, but will any K gives rise to some ϕ? Mercer's theorem, under certain conditions, tells us that we only need to ensure that K is positive semidefinite for any set $x_1, \ldots, x_n$. On the other hand, we might not even care that (5) can be converted back to (1) with some ϕ.

1.3 Examples of kernels

Gaussian kernel:

$$K(x_i, x_j) = \exp\left\{-\frac{\|x_i - x_j\|^2}{\sigma}\right\}.$$

This kernel was use in the initial work of Aizerman et al. One may think of placing positive potentials at examples of one class and negative potentials at examples from another class. (For this reason, Aizerman et al called K a potential function).

Polynomial kernel, another popular choice, is defined by

$$K(x_i, x_j) = (1 + \langle x_i, x_j \rangle)^k$$

1.4 Kernelized SGD

Recall that rather than solving (1), we equivalently solve (5) over $\alpha \in \mathbb{R}^n$. Any such vector gives a linear (in the high-dimensional space) function

$$x \mapsto \langle w, \phi(x) \rangle = \left\langle \sum_{j=1}^n \alpha_j \phi(x_j), \phi(x) \right\rangle = \sum_{j=1}^n \alpha_j K(x_j, x),$$

as long as K is well defined outside of the sample. So, even on a new example, we do not ever need to calculate $\phi(x)$: the prediction is given via the kernel function.

Prediction of the class label for the given x is

$$\text{sign}\left(\sum_{j=1}^n \alpha_j K(x_j, x)\right).$$

Recall that SGD for hinge loss chooses a random example (x_i, y_i) and checks whether the prediction has margin less than 1:

$$y_i \sum_{j=1}^n \alpha_j K(x_j, x_i) < 1. \tag{6}$$

If this is the case, a correction needs to be made. The correction is a subgradient of hinge $\max\{0, 1 - y_i \langle w, \phi(x_i) \rangle\}$, which is $y_i \phi(x_i)$, with an appropriate step size scaling. Recall the closed-form representation of the vector w for Pegasos from Lecture 4:

$$w_{t+1} = \frac{1}{\lambda t} \sum_{s=1}^t y_{i_s} \phi(x_{i_s}) \mathbf{1}\{y_{i_s} \langle w_s, \phi(x_{i_s}) \rangle < 1\} \tag{7}$$

where i_s denotes the index of the random example chosen at step s, and we have replaced x_{i_s} by $\phi(x_{i_s})$. Since the vectors $\phi(x_j)$ repeat in the above sum, we may write (following [SSBD14])

$$w_{t+1} = \frac{1}{\lambda t} \sum_{j=1}^n \beta_j^t \phi(x_j) \tag{8}$$

with β^t being the current vector of coefficients, being updated at every step by $y_{i_s} \mathbf{1}\{y_{i_s} \langle w_s, \phi(x_{i_s}) \rangle < 1\}$.

The final output of the algorithm is computed by averaging

$$\widehat{\alpha} = \frac{1}{T} \sum_{t=1}^T \frac{1}{\lambda t} \beta^t = \frac{1}{T} \sum_{t=1}^T \alpha^t$$

Algorithm 1 SGD for kernel SVM

Input: $\lambda > 0$ (regularization parameter)
Init: $\beta^1 = 0$
for t=1,...,T **do**
  Set $\alpha^t = \frac{1}{\lambda t}\beta^t$
  Sample $i \sim \text{Unif}[n]$
  Set $\beta_j^{t+1} = \beta_j^t$ for all $j \neq i$.
  if $y_i \sum_{j=1}^n \alpha_j^t K(x_j, x_i) < 1$ **then**
    $\beta_i^{t+1} = \beta_i^t + y_i$
  else
    $\beta_i^{t+1} = \beta_i^t$
  end if
end for

The coefficients $\widehat{\alpha}$ then give the decision function

$$x \mapsto \text{sign}\left(\sum_{j=1}^n \widehat{\alpha}_j K(x_j, x)\right)$$

We remark that kernelized SGD is not a truly online algorithm since memory requirement grows with n, and prediction at a new x requires computing the kernel function between x and all the training data points (unless we are lucky and most α_j coefficients end up being zero).

1.5 Comparison to neural nets

By taking an expressive enough kernel, we can achieve zero classification error with a highly nonlinear classifier. Similarly, by taking large enough neural network one may fit the data perfectly. What makes the non-convex neural network formulation preferred in this situation to convex kernel methods is a subject of current research in the community. (more details discussed in class)

In the next lecture we will talk about the bias-variance tradeoff with respect to the expressiveness of the model.

References

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.