XCS229 Supplemental Lecture Notes

Andrew Ng

1 General loss functions

Building off of our interpretations of supervised learning as (1) choosing a representation for our problem, (2) choosing a loss function, and (3) minimizing the loss, let us consider a slightly more general formulation for supervised learning. In the supervised learning settings we have considered thus far, we have input data $x \in \mathbb{R}^n$ and targets y from a space $\mathcal{Y}$. In linear regression, this corresponded to $y \in \mathbb{R}$, that is, $\mathcal{Y} = \mathbb{R}$, for logistic regression and other binary classification problems, we had $y \in \mathcal{Y} = \{-1, 1\}$, and for multiclass classification we had $y \in \mathcal{Y} = \{1, 2, \ldots, k\}$ for some number k of classes.

For each of these problems, we made predictions based on $\theta^T x$ for some vector θ, and we constructed a loss function $\mathsf{L} : \mathbb{R} \times \mathcal{Y} \to \mathbb{R}$, where $\mathsf{L}(\theta^T x, y)$ measures the loss we suffer for predicting $\theta^T x$. For logistic regression, we use the logistic loss

$$\mathsf{L}(z, y) = \log(1 + e^{-yz}) \quad \text{or} \quad \mathsf{L}(\theta^T x, y) = \log(1 + e^{-y\theta^T x}).$$

For linear regression we use the squared error

$$\mathsf{L}(z, y) = \frac{1}{2}(z - y)^2 \quad \text{or} \quad \mathsf{L}(\theta^T x, y) = \frac{1}{2}(\theta^T x - y)^2.$$

For multiclass classification, we had a slight variant, where we let $\Theta = [\theta_1 \ \cdots \ \theta_k]$ for $\theta_i \in \mathbb{R}^n$, and used the loss $\mathsf{L} : \mathbb{R}^k \times \{1, \ldots, k\} \to \mathbb{R}$

$$\mathsf{L}(z, y) = \log\left(\sum_{i=1}^{k} \exp(z_i - z_y)\right) \quad \text{or} \quad \mathsf{L}(\Theta^T x, y) = \log\left(\sum_{i=1}^{k} \exp(x^T(\theta_i - \theta_y))\right),$$

the idea being that we wish to have $\theta_y^T x > \theta_i^T x$ for all $i \neq k$. Given a training set of pairs $\{x^{(i)}, y^{(i)}\}$, choose θ by minimizing the empirical risk

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathsf{L}(\theta^T x^{(i)}, y^{(i)}). \tag{1}$$

2 The representer theorem

Let us consider a slight variant of choosing θ to minimize the risk (1). In many situations—for reasons that we will study more later in the class—it is useful to add *regularization* to the risk J. We add regularization for many reasons: often, it makes problem (1) easier to solve numerically, and also it can allow the θ we get out of minimizing the risk (1) able to generalize better to unseen data. Generally, regularization is taken to be of the form $r(\theta) = \|\theta\|$ or $r(\theta) = \|\theta\|^2$ for some norm $\|\cdot\|$ on $\mathbb{R}^n$. The most common choice is so-called ℓ_2-regularization, in which we choose

$$r(\theta) = \frac{\lambda}{2} \|\theta\|_2^2,$$

where we recall that $\|\theta\|_2 = \sqrt{\theta^T\theta}$ is the Euclidean norm, or length, of the vector θ. This gives rise to the *regularized risk*, which is

$$J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathsf{L}(\theta^T x^{(i)}, y^{(i)}) + \frac{\lambda}{2} \|\theta\|_2^2. \tag{2}$$

Let us consider the structure of any θ that minimizes the risk (2). We assume—as we often do—that for each fixed target value $y \in \mathcal{Y}$, the function $\mathsf{L}(z, y)$ is convex in z. (This is the case for linear regression and binary and multiclass logistic regression, as well as a number of other losses we will consider.) It turns out that under these assumptions, we may always write the solutions to the problem (2) as a *linear combination* of the input variables $x^{(i)}$. More precisely, we have the following theorem, known as the *representer theorem.*

Theorem 1. *Suppose in the definition of the regularized risk* (2) *that* $\lambda \geq 0$. *Then there is a minimizer of the regularized risk* (2) *that can be written*

$$\theta = \sum_{i=1}^{m} \alpha_i x^{(i)}$$

for some real-valued weights α_i.

Proof For intuition, we give a proof of the result in the case that $\mathsf{L}(z, y)$, when viewed as a function of z, is differentiable and $\lambda > 0$. In Appendix A, we present a more general statement of the theorem as well as a rigorous proof.

Let $\mathsf{L}'(z, y) = \frac{\partial}{\partial z}\mathsf{L}(z, y)$ denote the derivative of the loss with respect to z. Then by the chain rule, we have the gradient identity

$$\nabla_\theta \mathsf{L}(\theta^T x, y) = \mathsf{L}'(\theta^T x, y)x \quad \text{and} \quad \nabla_\theta \frac{1}{2} \left\|\theta\right\|_2^2 = \theta,$$

where ∇_θ denotes taking a gradient with respect to θ. As the risk must have 0 gradient at all stationary points (including the minimizer), we can write

$$\nabla J_\lambda(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathsf{L}'(\theta^T x^{(i)}, y^{(i)})x^{(i)} + \lambda\theta = \vec{0}.$$

In particular, letting $w_i = \mathsf{L}'(\theta^T x^{(i)}, y^{(i)})$, as $\mathsf{L}'(\theta^T x^{(i)}, y^{(i)})$ is a scalar (which depends on θ, but no matter what θ is, w_i is still a real number), we have

$$\theta = -\frac{1}{\lambda} \sum_{i=1}^{n} w_i x^{(i)}.$$

Set $\alpha_i = -\frac{w_i}{\lambda}$ to get the result. □

3 Nonlinear features and kernels

Based on the representer theorem, Theorem 1, we see that we can always write the vector θ as a linear combination of the data $\{x^{(i)}\}_{i=1}^m$. Importantly, this means we can *always* make predictions

$$\theta^T x = x^T \theta = \sum_{i=1}^{m} \alpha_i x^T x^{(i)}.$$

That is, in *any* learning algorithm, we may can replace all appearances of $\theta^T x$ with $\sum_{i=1}^m \alpha_i {x^{(i)}}^T x$, and then minimize directly over $\alpha \in \mathbb{R}^m$.

Let us consider this idea in somewhat more generality. In our discussion of linear regression, we had a problem in which the input x was the living area of a house, and we considered performing regression using the features x, x^2 and x^3 (say) to obtain a cubic function. To distinguish between these two sets of variables, we'll call the "original" input value the input **attributes** of a problem (in this case, x, the living area). When that is mapped to some new set of quantities that are then passed to the learning algorithm, we'll call those new quantities the input **features**. (Unfortunately, different authors use different terms to describe these two things, but we'll try to use this terminology consistently in these notes.) We will also let ϕ denote the **feature mapping**, which maps from the attributes to the features. For instance, in our example, we had

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}.$$

Rather than applying a learning algorithm using the original input attributes x, we may instead want to learn using some features $\phi(x)$. To do so, we simply need to go over our previous algorithm, and replace x everywhere in it with $\phi(x)$.

Since the algorithm can be written entirely in terms of the inner products $\langle x, z \rangle$, this means that we would replace all those inner products with $\langle \phi(x), \phi(z) \rangle$. Specificically, given a feature mapping ϕ, we define the corresponding **kernel** to be

$$K(x, z) = \phi(x)^T \phi(z).$$

Then, everywhere we previously had $\langle x, z \rangle$ in our algorithm, we could simply replace it with $K(x, z)$, and our algorithm would now be learning using the features ϕ. Let us write this out more carefully. We saw by the representer theorem (Theorem 1) that we can write $\theta = \sum_{i=1}^{m} \alpha_i \phi(x^{(i)})$ for some weights α_i. Then we can write the (regularized) risk

$$\begin{aligned}
J_\lambda(\theta) &= J_\lambda(\alpha) \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathsf{L}\left(\phi(x^{(i)})^T \sum_{j=1}^{m} \alpha_j \phi(x^{(j)}), y^{(i)} \right) + \frac{\lambda}{2} \left\| \sum_{i=1}^{m} \alpha_i \phi(x^{(i)}) \right\|_2^2 \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathsf{L}\left(\sum_{j=1}^{m} \alpha_j \phi(x^{(i)})^T \phi(x^{(j)}), y^{(i)} \right) + \frac{\lambda}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \phi(x^{(i)})^T \phi(x^{(j)}) \\
&= \frac{1}{m} \sum_{i=1}^{m} \mathsf{L}\left(\sum_{j=1}^{m} \alpha_j K(x^{(i)}, x^{(j)}), y^{(i)} \right) + \frac{\lambda}{2} \sum_{i,j} \alpha_i \alpha_i K(x^{(i)}, x^{(j)}).
\end{aligned}$$

That is, we can write the entire loss function to be minimized in terms of the kernel matrix

$$K = [K(x^{(i)}, x^{(j)})]_{i,j=1}^{m} \in \mathbb{R}^{m\times m}.$$

Now, given ϕ, we could easily compute $K(x, z)$ by finding $\phi(x)$ and $\phi(z)$ and taking their inner product. But what's more interesting is that often, $K(x, z)$ may be very inexpensive to calculate, even though $\phi(x)$ itself may be very expensive to calculate (perhaps because it is an extremely high dimensional vector). In such settings, by using in our algorithm an efficient way to calculate $K(x, z)$, we can learn in the high dimensional feature space space given by ϕ, but without ever having to explicitly find or represent vectors $\phi(x)$. As a few examples, some kernels (corresponding to infinite-dimensional vectors ϕ) include

$$K(x, z) = \exp\left(-\frac{1}{2\tau^2}\left\|x - z\right\|_2^2\right),$$

known as the Gaussian or Radial Basis Function (RBF) kernel and applicable to data in any dimension, or the min-kernel (applicable when $x \in \mathbb{R}$, defined by

$$K(x, z) = \min\{x, z\}.$$

See also the lecture notes on Support Vector Machines (SVMs) for more on these kernel machines.

4 Stochastic gradient descent for kernelized machine learning

If we let $K \in \mathbb{R}^{m\times m}$ denote the kernel matrix, and for shorthand define the vectors

$$K^{(i)} = \begin{bmatrix} K(x^{(i)}, x^{(1)}) \\ K(x^{(i)}, x^{(2)}) \\ \vdots \\ K(x^{(i)}, x^{(m)}) \end{bmatrix},$$

so that $K = [K^{(1)}\ K^{(2)}\ \cdots\ K^{(m)}]$, then we may write the regularized risk in a consise form as

$$J_\lambda(\alpha) = \frac{1}{m}\sum_{i=1}^{m} \mathsf{L}(K^{(i)^T}\alpha, y^{(i)}) + \frac{\lambda}{2}\alpha^T K\alpha.$$

Now, let us consider taking a stochastic gradient of the above risk J_λ. That is, we wish to construct a (simple to compute) random vector whose expectation is $\nabla J_\lambda(\alpha)$, which does not have too much variance. To do so, we first compute the gradient of $J_\lambda(\alpha)$ on its own. We calculate the gradient of individual loss terms by noting that

$$\nabla_\alpha \mathsf{L}({K^{(i)}}^T\alpha, y^{(i)}) = \mathsf{L}'({K^{(i)}}^T\alpha, y^{(i)})K^{(i)},$$

while

$$\nabla_\alpha \left[\frac{\lambda}{2}\alpha^T K\alpha\right] = \lambda K\alpha = \lambda\sum_{i=1}^m K^{(i)}\alpha_i.$$

Thus, we have

$$\nabla_\alpha J_\lambda(\alpha) = \frac{1}{m}\sum_{i=1}^m \mathsf{L}'({K^{(i)}}^T\alpha, y^{(i)})K^{(i)} + \lambda\sum_{i=1}^m K^{(i)}\alpha_i.$$

Thus, if we choose a random index $i \in \{1, \ldots, m\}$, we have that

$$\mathsf{L}'({K^{(i)}}^T\alpha, y^{(i)})K^{(i)} + m\lambda K^{(i)}\alpha_i$$

is a stochastic gradient for $J_\lambda(\alpha)$. This gives us a stochastic gradient procedure for Kernel supervised learning problems, shown in Figure 1. One minor remark is in order regarding Algorithm 1: because we multiply the $\lambda K^{(i)}\alpha_i$ term by m to keep the gradient unbiased, it is important that $\lambda > 0$ not be too large, as the algorithm can be a bit unstable otherwise. In addition, a common choice of stepsize is to use $\eta_t = 1/\sqrt{t}$, or a constant multiple thereof.

```
Input: A loss function $\mathsf{L}$, kernel matrix $K = [K^{(1)} \cdots K^{(m)}]$, and labels $\{y^{(i)}\}_{i=1}^m$, and sequence of positive stepsizes $\eta_1, \eta_2, \eta_3, \ldots$

Iterate for $t = 1, 2, \ldots$
    (i) Choose index $i \in \{1, \ldots, m\}$ uniformly at random
    (ii) Update
        $\alpha := \alpha - \eta_t \left[\mathsf{L}'({K^{(i)}}^T\alpha, y^{(i)})K^{(i)} + m\lambda K^{(i)}\alpha_i\right].$
```

Figure 1: Stochastic gradient descent for kernel supervised learning problems.

5 Support vector machines

Now we discuss (one approach) to Support Vector Machines (SVM)s, which apply to binary classification problems with labels $y \in \{-1, 1\}$, and a particular choice of loss function L. In particular, for the support vector machine, we use the margin-based loss function

$$\mathsf{L}(z, y) = [1 - yz]_+ = \max\{0, 1 - yz\}. \tag{3}$$

So, in a sense, SVMs are nothing but a special case of the general theoretical results we have described above. In particular, we have the empirical regularized risk

$$J_\lambda(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \left[1 - y^{(i)} K^{(i)^T} \alpha\right]_+ + \frac{\lambda}{2} \alpha^T K \alpha,$$

where the matrix $K = [K^{(1)} \ \cdots \ K^{(m)}]$ is defined by $K_{ij} = K(x^{(i)}, x^{(j)})$.

In the lecture notes, you can see another way of deriving the support vector machine and a description of why we actually call them support vector machines.

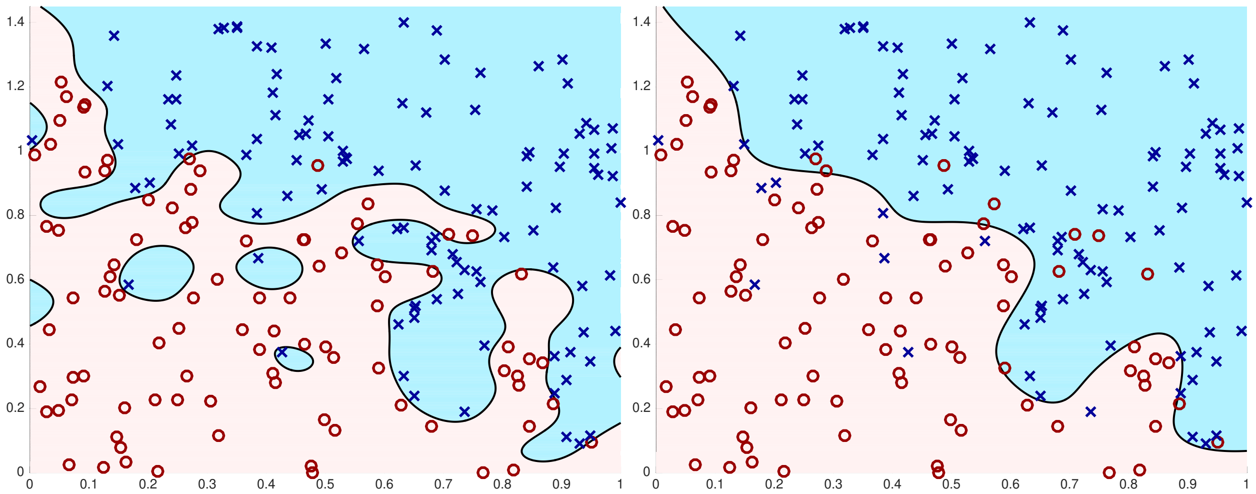
6 An example

In this section, we consider a particular example kernel, known as the Gaussian or Radial Basis Function (RBF) kernel. This kernel is defined by

$$K(x, z) = \exp\left(-\frac{1}{2\tau^2} \|x - z\|_2^2\right), \tag{4}$$

where $\tau > 0$ is a parameter controlling the *bandwidth* of the kernel. Intuitively, for τ very small, we will have $K(x, z) \approx 0$ unless $x \approx z$, that is, x and z are very close, in which case we have $K(x, z) \approx 1$. However, for τ large, then we have a much smoother kernel function K. The feature function ϕ for this kernel is infinite dimensional.[1] That said, it is possible to gain some

[1] If you have seen characteristic functions or Fourier transforms, then you might recognize the RBF kernel as the Fourier transform of the Gaussian distribution with mean zero and variance τ^2. That is, in $\mathbb{R}^n$, let $W \sim \mathcal{N}(0, \tau^2 I_{n \times n})$, so that W has density



$\tau = .1$ $\tau = .2$

Figure 2: Small bandwidths τ for Gaussian kernel $K(x,z) = \exp(-\frac{1}{2\tau^2}\left\|x-z\right\|_2^2)$.

intuition for the kernel by considering the classifications it makes on a new example x: we have prediction

$$\sum_{i=1}^{m} K(x^{(i)}, x)\alpha_i = \sum_{i=1}^{m} \exp\left(-\frac{1}{2\tau^2}\left\|x^{(i)} - x\right\|_2^2\right)\alpha_i,$$

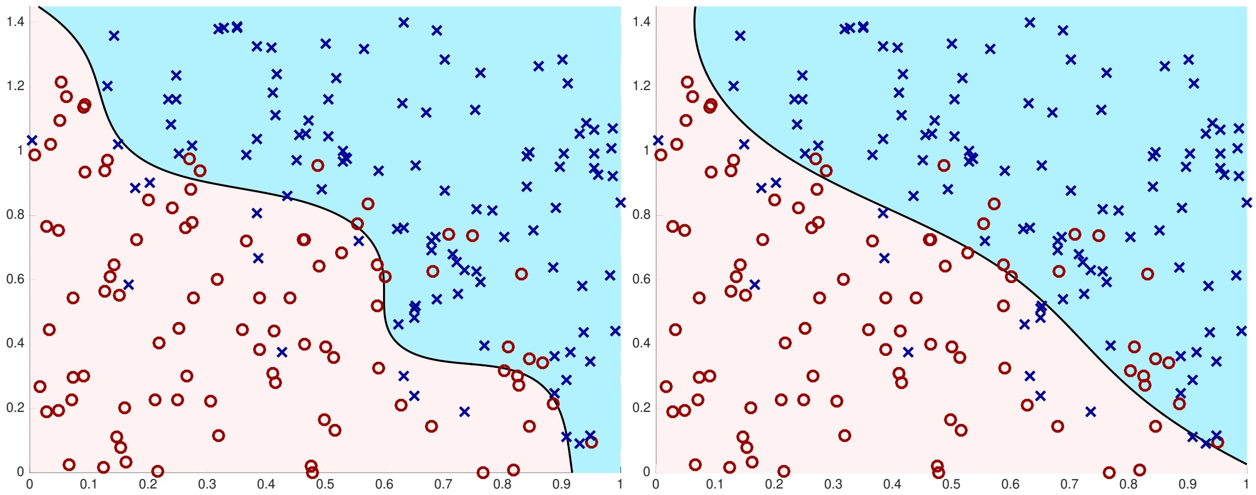
$p(w) = \frac{1}{(2\pi\tau^2)^{n/2}}\exp(-\frac{\|w\|_2^2}{2\tau^2})$. Let $i = \sqrt{-1}$ be the imaginary unit, then for any vector v we have

$$\mathbb{E}[\exp(iv^T W)] = \int \exp(iv^T w)p(w)dw = \int \frac{1}{(2\pi\tau^2)^{n/2}}\exp\left(iv^T w - \frac{1}{2\tau^2}\left\|w\right\|_2^2\right)dw$$
$$= \exp\left(-\frac{1}{2\tau^2}\left\|v\right\|_2^2\right).$$

Thus, if we define the "vector" (really, function) $\phi(x,w) = e^{ix^T w}$ and let a^* be the complex conjugate of $a \in \mathbb{C}$, then we have

$$\mathbb{E}[\phi(x,W)\phi(z,W)^*] = \mathbb{E}[e^{ix^T W}e^{-ix^T W}] = \mathbb{E}[\exp(iW^T(x-z))] = \exp\left(-\frac{1}{2\tau^2}\left\|x-z\right\|_2^2\right).$$

In particular, we see that $K(x,z)$ is the inner product in a space of functions that are integrable against $p(w)$.



$\tau = .4$ $\tau = .8$

Figure 3: Medium bandwidths τ for Gaussian kernel $K(x, z) = \exp(-\frac{1}{2\tau^2} \|x - z\|_2^2)$.

so that this becomes something like a weighting depending on *how close* x is to each $x^{(i)}$—that is, the contribution of weight α_i is multiplied by the similarity of x to $x^{(i)}$ as determined by the kernel function.

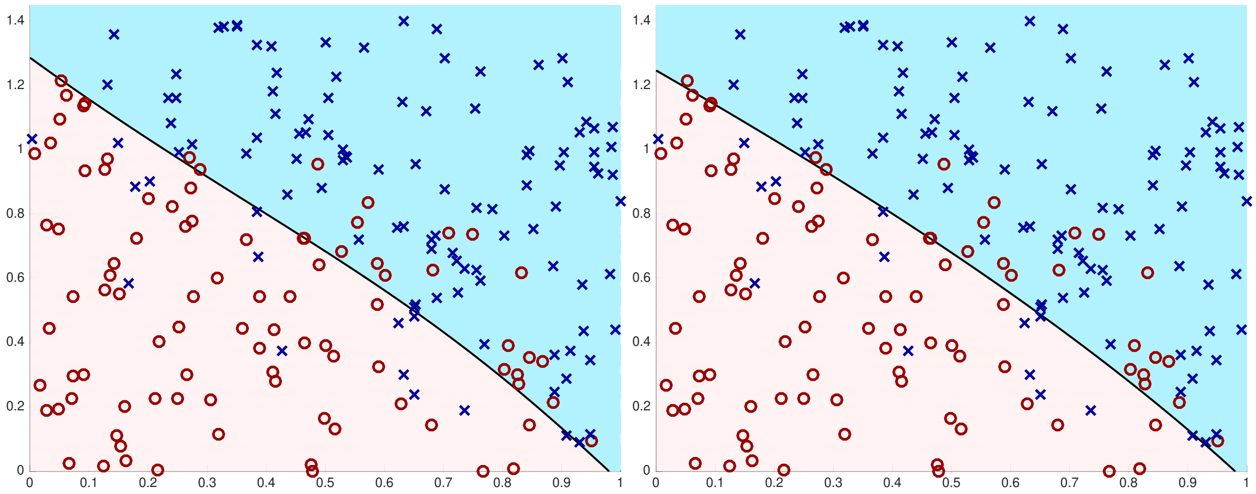
In Figures 2, 3, and 4, we show the results of training 6 different kernel classifiers by minimizing

$$J_\lambda(\alpha) = \sum_{i=1}^{m} \left[1 - y^{(i)} K^{(i)T} \alpha\right]_+ + \frac{\lambda}{2} \alpha^T K \alpha,$$

with $m = 200$ and $\lambda = 1/m$, for different values of τ in the kernel (4). We plot the training data (positive examples as blue `x`'s and negative examples as red `o`'s) as well as the decision surface of the resulting classifier. That is, we plot the lines defined by

$$\left\{ x \in \mathbb{R}^2 \;:\; \sum_{i=1}^{m} K(x, x^{(i)}) \alpha_i = 0 \right\},$$

giving the regions where the learned classifier makes a prediction $\sum_{i=1}^m K(x, x^{(i)})\alpha_i > 0$ and $\sum_{i=1}^m K(x, x^{(i)})\alpha_i < 0$. From the figure, we see that for large τ, we have



$\tau = 1.6$

$\tau = 3.2$

Figure 4: Large bandwidths τ for Gaussian kernel $K(x, z) = \exp(-\frac{1}{2\tau^2}\|x - z\|_2^2)$.

a very simple classifier: it is nearly linear, while for $\tau = .1$, the classifier has substantial variability and is highly non-linear. For reference, in Figure 5, we plot the optimal classifier along with the training data; the optimal classifier minimizes the misclassification error given infinite training data.

A A more general representer theorem

In this section, we present a more general version of the representer theorem along with a rigorous proof. Let $r : \mathbb{R} \to \mathbb{R}$ be any non-decreasing function of its argument, and consider the regularized risk

$$J_r(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathsf{L}({x^{(i)}}^T\theta, y^{(i)}) + r(\|\theta\|_2). \tag{5}$$

Often, we take $r(t) = \frac{\lambda}{2}t^2$, which corresponds to the common choice of ℓ_2-regularization, but the next theorem makes clear that this is not necessary for the representer theorem. Indeed, we could simply take $r(t) = 0$ for all t, and the theorem still holds.

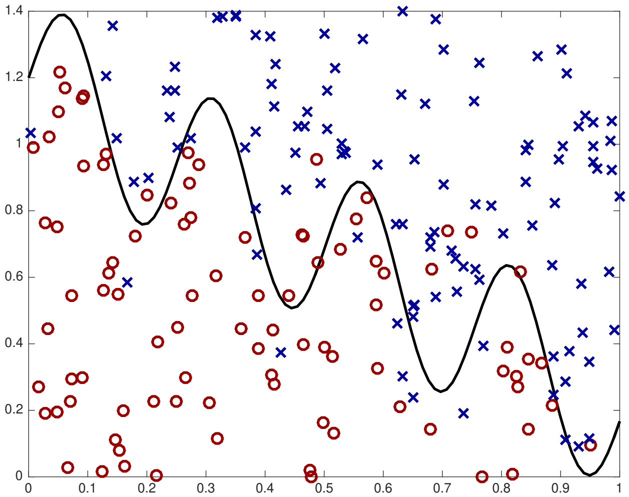


Figure 5: Optimal classifier along with training data.

Theorem 2 (Representer theorem in $\mathbb{R}^n$). *Let $\theta \in \mathbb{R}^n$ be any vector. Then there exists some $\alpha \in \mathbb{R}^m$ and $\theta^{(\alpha)} = \sum_{i=1}^m \alpha_i x^{(i)}$ such that*

$$J_r(\theta^{(\alpha)}) \le J_r(\theta).$$

In particular, there is no loss of generality in always assuming we can write the optimization problem to minimize $J(\theta)$ by only considering θ in the span of the data.

Proof Our proof relies on some machinery from linear algebra, which allows us to keep it concise, but feel free to ask questions if it is too concise.

The vectors $\{x^{(i)}\}_{i=1}^m$ are in $\mathbb{R}^n$, and as a consequence there is some subspace V of $\mathbb{R}^n$ such that

$$V = \left\{ \sum_{i=1}^m \beta_i x^{(i)} : \beta_i \in \mathbb{R} \right\}.$$

Then V has an orthonormal basis $\{v_1, \ldots, v_{n_0}\}$ for vectors $v_i \in \mathbb{R}^n$, where the size (dimension) of the basis is $n_0 \le n$. Thus we can write $V = \{\sum_{i=1}^{n_0} b_i v_i : b_i \in \mathbb{R}\}$, where we recall that orthonormality means that the vectors v_i satisfy $\|v_i\|_2 = 1$ and $v_i^T v_j = 0$ for $i \neq j$. There is also an orthogonal subspace $V^\perp = \{u \in \mathbb{R}^n : u^T v = 0 \text{ for all } v \in V\}$, which has an orthonormal basis of size $n_\perp = n - n_0 \geq 0$, which we write as $\{u_1, \ldots, u_{n_\perp}\} \subset \mathbb{R}^n$. By construction they satisfy $u_i^T v_j = 0$ for all i, j.

Because $\theta \in \mathbb{R}^n$, we know that we can write it uniquely as

$$\theta = \sum_{i=1}^{n_0} \nu_i v_i + \sum_{i=1}^{n_\perp} \mu_i u_i, \quad \text{where } \; \nu_i \in \mathbb{R} \text{ and } \mu_i \in \mathbb{R},$$

and the values μ, ν are unique. Now, we know that by definition of the space V as the span of $\{x^{(i)}\}_{i=1}^m$, there exists $\alpha \in \mathbb{R}^m$ such that

$$\sum_{i=1}^{n_0} \nu_i v_i = \sum_{i=1}^{m} \alpha_i x^{(i)},$$

so that we have

$$\theta = \sum_{i=1}^{m} \alpha_i x^{(i)} + \sum_{i=1}^{n_\perp} \mu_i u_i.$$

Define $\theta^{(\alpha)} = \sum_{i=1}^m \alpha_i x^{(i)}$. Now, for any data point $x^{(j)}$, we have

$$u_i^T x^{(j)} = 0 \;\; \text{for all } i = 1, \ldots, n_\perp,$$

so that $u_i^T \theta^{(\alpha)} = 0$. As a consquence, we have

$$\|\theta\|_2^2 = \left\| \theta^{(\alpha)} + \sum_{i=1}^{n_\perp} \mu_i u_i \right\|_2^2 = \left\|\theta^{(\alpha)}\right\|_2^2 + \underbrace{2 \sum_{i=1}^{n_\perp} \mu_i u_i^T \theta^{(\alpha)}}_{=0} + \left\| \sum_{i=1}^{n_\perp} \mu_i u_i \right\|_2^2 \geq \left\|\theta^{(\alpha)}\right\|_2^2, \tag{6a}$$

and we also have

$$\theta^{(\alpha)^T} x^{(i)} = \theta^T x^{(i)} \tag{6b}$$

for all points $x^{(i)}$.

That is, by using $\|\theta\|_2 \geq \left\|\theta^{(\alpha)}\right\|_2$ and equality (6b), we have

$$\begin{aligned} J_r(\theta) = \frac{1}{m} \sum_{i=1}^m \mathsf{L}(\theta^T x^{(i)}, y^{(i)}) + r(\|\theta\|_2) &\overset{(6b)}{=} \frac{1}{m} \sum_{i=1}^m \mathsf{L}(\theta^{(\alpha)^T} x^{(i)}, y^{(i)}) + r(\|\theta\|_2) \\ &\overset{(6a)}{\geq} \frac{1}{m} \sum_{i=1}^m \mathsf{L}(\theta^{(\alpha)^T} x^{(i)}, y^{(i)}) + r(\left\|\theta^{(\alpha)}\right\|_2) \\ &= J_r(\theta^{(\alpha)}). \end{aligned}$$

This is the desired result. □