Kernels

compiled by Alvin Wan from Professor Benjamin Recht's lecture

1 Exposition

Before this segment, our goal was to identify $f : X \to Y$. We take X, build features, and construct a linear map. Neural networks on the other hand construct features *and* the linear map at the same time. In the next few notes, we will cover:

1. Kernels (Note 20)
2. Nearest Neighbors (Note 21, 22)
3. Decision Trees (Note 23, 24)

All three of these are special cases of fitting functions to data, but we are not explicitly learning linear maps. We will end this series of notes with unsupervised learning, where we do not have Y.

2 Motivation

2.1 Ridge Regression Derivation

Take X, which is $n \times d$, where by convention n is the number of samples and d is the number of features. We restate the objective function and the solution for ridge regression.

$$\text{minimize} \|Xw - y\|_2^2 + \lambda \|w\|_2^2$$
$$w^* = (X^T X + \lambda I)^{-1} X^T y$$

We will now apply the matrix inversion lemma, giving us the following.

$$w^* = X^T (X X^T + \lambda I)^{-1} y$$

These two forms tell us that the number of parameters we need to solve for, is either n (second form) or d (first form). This XX^T is known as our **kernel matrix**. We will revisit this and explain why the kernel is significant. First, let us now derive a more general example, showing how for any loss, we can convert to a problem with n parameters.

2.2 General Derivation

Take a general objective function with an L2 penalty.

$$\text{minimize}_w \sum_{i=1}^{n} \text{loss}(w^T x_i, y_i) + \lambda \|w\|_2^2$$

Let us consider the following definition of w.

$$w = \sum_{j=1}^{n} \alpha_j x_j + v$$

where $\forall i, v^T x_i = 0$. We now plug in to obtain our new objective function. From step 2 to 3, the cross-term is 0, because $v^T x_i = 0$.

$$\begin{aligned}
&\text{minimize}_{\alpha,v} \sum_{i=1}^{n} \text{loss}(\sum_{j=1}^{n} \alpha_j x_j^T x_i + v^T x_i, y_i) + \lambda \| \sum_{j=1}^{n} \alpha_j x_j + v\|_2^2 \\
&= \text{minimize}_{\alpha,v} \sum_{i=1}^{n} \text{loss}(\sum_{j=1}^{n} \alpha_j x_j^T x_i, y_i) + \lambda \| \sum_{j=1}^{n} \alpha_j x_j + v\|_2^2 \\
&= \text{minimize}_{\alpha,v} \sum_{i=1}^{n} \text{loss}(\sum_{j=1}^{n} \alpha_j x_j^T x_i, y_i) + \lambda \| \sum_{j=1}^{n} \alpha_j x_j\|_2^2 + \lambda \|v\|_2^2
\end{aligned}$$

Note that this value is minimized when $v = 0$, as all but the last term have no dependence on v. Additionally, this objective is now only dependent on $x_i^T x_j$. In short, we now see that the lifting trick is actually building a similarity between data points. We want similar points to have a large dot product, and we want distinct data points to have a small or negative dot product. We can define $K_{ij} = x_i^T x_j$, and plug in to simplify our expression.

Note that if $\lambda = 0$, v doesn't need to approach 0, but we then have a family of solutions.

$$\text{minimize} \sum_{i=1}^{n} \text{loss}([K\alpha]_i, y_i) + \lambda \alpha^T K \alpha$$

We then form K, called the "kernel matrix" or "Gram matrix", and solve the optimization problem. We plug in K wherever we have a dot product. Why is this significant? This is because we never need to solve a problem with more than n parameters. When predicting for a new x, we simply evaluate

$$f(x) = w^T x = \sum_{i=1}^{n} \alpha_i x_i^T x$$

All of our computation has been reduced to dot products.

3 The Kernel Matrix

Consider the function $k(x, z)$ that evaluates inner products. We use k to lift features, and then evaluate the dot product to reduce dimensionality. Our claim is that k achieves both lifting and the kernel trick, without the intermediate step. We will first consider a one-dimensional example, with $x, z \in \mathbb{R}$.

$$\begin{aligned} k(x, z) &= (1 + xz)^2 \\ &= 1 + 2xz + x^2 z^2 \\ &= \begin{bmatrix} 1 \\ \sqrt{2}x \\ x^2 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{x}z \\ z^2 \end{bmatrix} \end{aligned}$$

We could find these vectors, or we could simply evaluate $k(x, z)$ to compute the dot product. As it turns out, every dot product has a kernel function associated with it. We now consider examples in higher dimensions, using polynomials. Consider $x \in \mathbb{R}^d$, where the number of quadratic monomials is $O(d^2)$.

1. **linear kernel**: $k(x, z) = x^T z$
2. **quadratic kernel**: $k(x, z) = (1 + x^T z)^2$
3. **Gaussian kernel**: $k(x, z) = \exp(-\gamma \|x - z\|^2)$

Each of these obeys several quintessential properties for any inner product.

Definition K is a valid kernel function if the matrix with entries $k_{ij} = k(x_i, x_j)$ is positive semi-definite for *all* sets $\{x_1, x_2 \ldots, x_n\}$.

Why is this sufficient? Take the linear kernel, for example, $K = XX^T$. We definitely get a positive semi-definite matrix. As it turns out, if the matrix K is positive semi-definite, we can always decompose it into a kernel function.

3.1 Verifying Linear, Quadratic Kernels

We will prove that the linear and quadratic kernels are valid. Take A, B to be symmetric, positive semi-definite $(n \times n)$. Does AB need to be positive semi-definite? We don't.

$$\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$$

Take the entry-wise product $C = [A\dot{B}] = A_{ij}B_{ij}$. Is C positive semi-definite? Yes. We can simply diagonalize to show the preservation of positive semi-definiteness. This proves the positive definiteness of the linear and quadratic kernel matrices.

3.2 Verifying Gaussian Kernels

We will now consider the Gaussian kernel, and expand.

$$\begin{aligned} \exp(-\gamma\|x - z\|^2) &= \exp(-\gamma\|x_i\|^2 + 2x_i^T x_j - \gamma\|x_j\|^2) \\ &= \exp(-\gamma\|x_i\|^2)\exp(2x_i^T x_j)\exp(-\gamma\|x_j\|^2) \\ &= (\sum_{k=0}^{\infty} \frac{(2\gamma x_i^T x_j)^k}{k!})\exp(2x_i^T x_j)\exp(-\gamma\|x_j\|^2) \end{aligned}$$

As it turns out, the Gaussian kernel is simply the linear kernel to an infinite power. So, what happens if we expand this into vector form, as we did for the quadratic kernel? Let us first consider the square of the quadratic kernel.

$$\begin{aligned} k(x,z) &= (1+xz)^4 \\ &= 1 + 4xz + 6x^2z^2 + 4x^3z^3 + x^4z^4 \\ &= \begin{bmatrix} 1 \\ 2x \\ \sqrt{6}x^2 \\ 2x^3 \\ x^4 \end{bmatrix} \begin{bmatrix} 1 \\ 2z \\ \sqrt{6}z^2 \\ 2z^3 \\ z^4 \end{bmatrix} \end{aligned}$$

Each of these entries in the vector is a feature. We can continue this inductively to see that the Gaussian kernel would have an infinite number of entries in our vector. In short, our kernel function is actually encompassing an infinite number of features.

4 Radial Basis Function

For the Gaussian kernel, our solution is of the following form.

$$f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) = \sum_{i=1}^{n} \exp(-\gamma \|x_i - x\|^2)$$

When our data x_i is extremely close to x, our exponent approaches 0, and the kernel function approaches 1. If x_i is far from x, our exponent approaches infinity, and our term approaches 0. In other words, we give similar points a positive weight and give points farther away negative weight. Consider the contour space. When x_i is far from the hyperplane, we assign the point a value of 0. This is also called the **Radial Basis Function (RBF)**. Our radius is simply the area for which $f(x)$ produces non-zero values.

One interesting fact is that $k(x,x) = 1$ for all γ, meaning that we've restricted ourselves to the unit sphere. Our parameter, γ is called the kernel bandwidth. Let us consider the behavior of $k(x,z)$ for values of γ.

- If $\gamma \to \infty$, $k(x, z) = 0$ for many x_i, producing narrow contours. Our kernel matrix then is similar to the identity.
- When γ is small, we are back to linear regression. This is because higher order terms tend to 0.

$$\exp(\gamma x^T z) \approx 1 + \gamma x^T z + \frac{1}{2}\gamma^2 (x^T z)^k \dots$$

In other words, with high gamma, the RBF is template matching, as a function of deltas at each of the x_i. As usual, we depend on cross validation to find our two parameters, γ and λ.

Note: We can also combine kernels. With k_1, k_2, we can take another kernel $\alpha_1 k_1 + \alpha_2 k_2$, $\alpha_1, \alpha_2 > 0$. This is a recipe for overfitting, however.

Why not take the Gaussian kernel on MNIST for each pixel? This is because our kernel is then $60,000 \times 60,000 \times 8 \approx 4 \times 10^{10} \approx 30GB$. With that said, we can use the kernel very effectively for smaller datasets.

Let us consider the following featurization, where $v \sim N(0, \gamma I), b \sim U[0, 2\pi]$

$$\begin{aligned} &E[\cos(v^T x + b) cos(x^T z + b) \\ &\approx \exp(-\gamma \|x - z\|^2) \end{aligned}$$

Whereas kernels are often too large to store, we can apply a trick. Specifically, we can write our featurization of x, $\phi(x)$, as the following.

$$\begin{bmatrix} \cos(x_1^T x + b_1) \\ \cos(x_2^T x + b_2) \\ \vdots \\ \cos(x_D^T x + b_D) \end{bmatrix}$$

So, we have $\langle \phi(x), \phi(z) \rangle \approx \exp(-\gamma \|x - z\|^2)$. Two-layer neural nets with Gaussian kernels are equivalent. Gaussian kernels can be thought of as two-layer neural networks, nearest neighbors, or somewhere in between nearest neighbors and linear regression.

Memory required to solve and caching x_i are the downsides of the kernel.