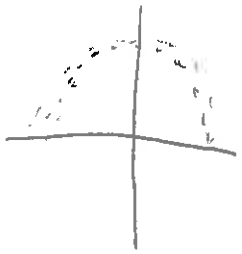
Lecture # 5: 9-18-18

- So far, the methods we have developed are suitable for linear data. What about non-linear data?

Ex:



There is clear structure here, but we don't know how to capture it, since it is nonlinear.

- Recall that our main object in PCA is the (centered) covariance matrix of the data $X_1, \dots, X_n \in \mathbb{R}^{1\times d}$:

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} X_i^T X_i.$$

Instead of consider $X_1, \dots, X_n$ directly, we can embed the data into another space via a map (which we choose), call it ϕ.

"Kernel" $\longrightarrow \phi : \mathbb{R}^d \to \mathbb{R}^M$) where M could be much greater than d.

- Then we can consider (assuming $\{\phi(x_i)\}_{i=1}^n \subset \mathbb{R}^M$ has mean 0), a "kernel covariance matrix"

$$C = \frac{1}{n}\sum_{i=1}^{n} \phi(x_i)^T \phi(x_i). \quad (\star)$$

- If ϕ is well-chosen, then we can learn very rich structure in the data. One potential issue is if M is very large, this becomes a computational nightmare, since computing C as in $(\star)$ is $O(M^2 n)$.

There is an approach for dealing with this that is famous in the machine learning world. It is called the Kernel trick.

Idea: Computing $\phi(x_i)^T \phi(x_i)$ is hard if M (the image space of ϕ) is large. Instead, consider only kernel evaluations $K(x_i, x_j) = \phi(x_j)^T \phi(x_i)$, for some easy to compute kernel function $K: \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$.

Indeed, if we wish to compute the eigenfunction of C:

$$C v_j^T = \lambda_j v_j^T$$

then substituting the definition of $C = \frac{1}{n} \sum_{i=1}^{n} \phi(x_i)^T \phi(x_i)$ yields:

$$\left(\frac{1}{n} \sum_{i=1}^{n} \phi(x_i)^T \phi(x_i) \right) v_j^T = \lambda_j v_j^T$$

$$\Rightarrow \frac{1}{n} \sum_{i=1}^{n} \alpha_j^i \phi(x_i)^T = \lambda_i v_i^T, \quad \text{where } \alpha_i = \phi(x_i) v_j^T \in \mathbb{R}$$

Hence, v_i is a linear combination of the $\{\phi(x_i)^T\}_{i=1}^n$ vectors, so writing the eigenvector equation in terms of this representation:

$$C v_j^T = \lambda_j v_j^T$$

$$\Rightarrow \left[\frac{1}{n} \sum_{i=1}^{n} \phi(x_i)^T \phi(x_i) \right] \left(\sum_{k=1}^{n} \beta_j^k \phi(x_k)^T \right) = \lambda_j \sum_{k=1}^{n} \beta_j^k \phi(x_k)^T$$

Letting $K(x_i, x_j) = \phi(x_j) \phi(x_i)^T$, i.e. writing this inner product with the kernel

yields, after multiplying both sides by $\phi(x_\ell)$ on the left:

$$\frac{1}{n}\sum_{i=1}^{n}\phi(x_\ell)\phi^T(x_i)\phi(x_i)\sum_{k=1}^{n}\beta_j^k\phi(x_k)^T = \lambda_j\sum_{i=1}^{n}\phi(x_\ell)\phi(x_k)^T$$

$$\Rightarrow \frac{1}{n}\sum_{i=1}^{n}k(x_\ell,x_i)\sum_{k=1}^{n}\beta_j^k k(x_i,x_k) = \lambda_j\sum_{i=1}^{n}\beta_j^k k(x_\ell,x_k). \quad (\star)$$

Let $K\in\mathbb{R}^{n\times n}$ be the matrix $K_{ij}=k(x_i,x_j)$. Then if we think of $b_j=\{\beta_j^k\}_{k=1}^n$ as a column vector, this equation reduces to

$$K^2 b_j = n\lambda_j K b_j$$

$$\Rightarrow \quad K b_j = n\lambda_j b_j \quad (\star)$$

The key point here is K is $n\times n$, not $M\times M$.

Supposing we can solve $(\star)$ to get b_j, we can use this to do the projection:

$$\phi(x)^T v_j = \sum_{i=1}^{n}\phi(x)\sum_{k=1}^{n}\beta_j^k\phi(x_k)^T$$

$$= \sum_{i=1}^{n}\beta_j^k k(x,x_k),$$

So all we need is the kernel evaluation $k(x,x_k)$.

An example of a non-linear kernel that is extremely common is the so-called

heat kernel $\quad k(x,y) = e^{-\|x-y\|^2/\sigma^2}$

for some parameter σ.

- We will return to this kernel when we do Spectral graph theory.

Bottom line:

1.) Kernel lets us handle non-linearity

2.) Complexity is a bit more of a problem.

$$\begin{array}{ccc} d \times d & \longrightarrow & n \times n \\ \Sigma & & K \end{array}$$

3.) At least the kernel trick allows us to avoid $M \times M$.