EE445 Mod3-Lec4: Principal Component Regression & Kernel PCA

Outline

1. Principal Component Regression (PCR)
2. Kernel PCA
3. (time permitting) Spectral Clustering

Part 1: PCR

What is Principal Component Regression (PCR)?

- Combining PCA with linear regression leads to principal components regression (PCR).
- PCR = PCA + linear regression:
  - Choose how target number of principal components k
  - Use PCA to define a feature vector

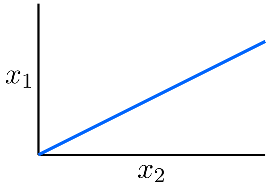
$$\varphi(x^{(i)}) = (\langle\, x^{(i)}, u_1 \,\rangle, \ldots, \langle\, x^{(i)}, u_k \,\rangle)$$

    as in **Mod3-L3** containing the principal component scores $x^{(i)}$—i.e., the projections onto the $u_1, \ldots, u_k$ principal components
  - Use least-squares linear regression (as in **Module 2**) with this model:

$$y^{(i)} = \varphi(x^{(i)})^\top \theta + \varepsilon_i$$

Facts about PCR

- **Why use it?**: In regression analysis, when the independent variables appear multicollinearly (independent variables are correlated), the general effect of the classical regression method for least square estimates of regression coefficients will be poor, but principal component analysis can overcome this deficiency effectively.



- PCR works well when the directions in which the original predictors vary most are the directions that are predictive of the outcome
- PCR is very similar to ridge regression in a certain sense.
- Nice comparison between PCR and partial least squares

  `https://scikit-learn.org/stable/auto_examples/cross_decomposition/plot_pcr_vs_pls.html`

Ridge Regression

- We saw Kernel regression with regularization in **Mod2 Lec4** where ℓ_2–regularization was introduced to support numerical stability of the kernel matrix
- We can in general introduce a $\|\cdot\|_2$ regularization term to vanilla least squares and this is called *ridge regression*:

$$F(\theta) = \frac{1}{2}\|X\theta - y\|^2 + \frac{\lambda}{2}\|\theta\|_2^2 = \frac{1}{2}\theta^\top X^\top X\theta - y^\top X\theta + \frac{1}{2}y^\top y + \frac{\lambda}{2}\theta^\top\theta$$

$$\texttt{first order}: \quad \nabla F(\theta) = X^\top X\theta - X^\top y + \lambda\theta = 0 \implies \hat{\theta} = (X^\top X + \lambda I)^{-1}X^\top y$$

- $\lambda \geq 0$: a tuning parameter that controls the strength of the penalty term
  - $\lambda = 0$: we get the linear regression estimate
  - $\lambda \to \infty$: we get $\hat{\theta}_{\texttt{ridge}} \to 0$
  - $\lambda \in (0, \infty)$: balancing fitting a linear model of y given data X, and **shrinking the coefficients** in θ

PCR and Ridge

- Write $X = USV^\top$ so that

$$\begin{aligned}
(X^\top X + \lambda I)^{-1} &= (VS^\top SV^\top + \lambda I)^{-1} \\
&= (VS^\top SV^\top + \lambda VV^\top)^{-1} \\
&= (V(S^\top S + \lambda I)V^\top)^{-1} \\
&= V(S^\top S + \lambda I)^{-1}V^\top \quad \text{since } V^{-1} = V^\top \\
&= VS^+V^\top, \quad \text{where } S^+ = \mathrm{diag}(1/(\sigma_1^2 + \lambda), \dots, 1/(\sigma_n^2 + \lambda), 0, \dots, 0)
\end{aligned}$$

$$\implies \hat{\theta} = VS^\dagger V^\top VS^\top U^\top y = VS^\dagger S^\top U^\top y = \sum_{i=1}^{n} \frac{\sigma_i}{\sigma_i^2 + \lambda} v_i u_i^\top y$$

- Hence, Ridge regression can be viewed as projecting the y vector onto the principal component directions and then shrinking the projection using λ

PCR vs Ridge

- Ridge regression shrinks everything, but it never shrinks anything to zero.
- By contrast, `PCR` either does not shrink a component at all or shrinks it to zero.
- Yet another alternative is `LASSO`: assuming the features are orthonormal

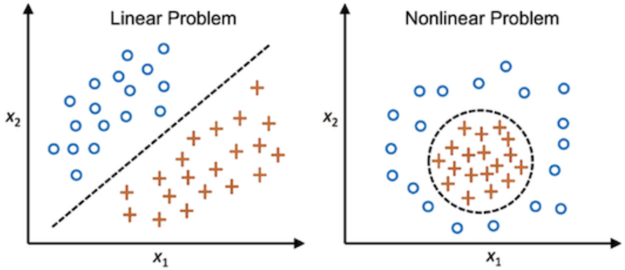
$$\min_{\theta} \frac{1}{2}\|X\theta - y\|^2 + \frac{\lambda}{2}\|\theta\|_1 \;\rightarrow\; \hat{\theta}_j = \hat{\theta}_j^{\texttt{ols}} \max\left\{0, 1 - \frac{m\lambda}{|\hat{\theta}_j^{\texttt{ols}}|}\right\}$$

which can set some coefficients to zero, and scales others

Part 2: Kernel PCA

Motivation

- when the data is non-linear, we may need a more complex polynomial function to separate the data



Recall the Kernel "trick"

- Essence of Kernel trick:
  - If we can write down an algorithm only in terms of $\phi(x^{(i)})^\top \phi(x^{(j)})$, then we don't need to explicitly enumerate $\phi(x^{(i)})$ for all the $i \in \{1, \ldots, m\}$
  - Instead we can just compute $K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^\top \phi(x^{(j)})$
  - And we can even handle infinite dimensional feature maps $\phi(\cdot)$
- Examples:
  - Linear kernel $K(x, z) = z^\top x$
  - radial basis function $K(x, z) = \exp(-\gamma \|x - z\|^2)$
  - polynomial kernels $K(x, z) = (x^\top z + c)^p$
- Predictions only depend on training data through kernel function which is just a dot product.

Rewrite PCA using Kernel "trick"

- Let $u_1, \ldots, u_k$ be the k largest principal components, and define matrix $U = \begin{bmatrix} u_1 & \cdots & u_k \end{bmatrix}$
- That is, the j-th column of U is the j-th eigenvector of the covariance matrix $\Sigma = \frac{1}{m}\sum_{i=1}^m \phi(x^{(i)})\phi(x^{(i)})^\top$.
- **Claim**: Eigenvectors can be expressed as linear combination of features
  Proof: Let (λ, v) be an eigenpair of Σ. Then we claim $v = \sum_{i=1}^m \alpha_i \phi(x^{(i)})$. Indeed,

$$\Sigma v = \frac{1}{m}\sum_{i=1}^m \phi(x^{(i)})\phi(x^{(i)})^\top v = \lambda v \quad \Longrightarrow$$

$$v = \frac{1}{\lambda m}\sum_{i=1}^m \phi(x^{(i)})\phi(x^{(i)})^\top v = \frac{1}{\lambda m}\sum_{i=1}^m \underbrace{(\phi(x^{(i)})^\top v)}_{\text{scalar}} \phi(x^{(i)})^\top$$

  This completes the proof.
- Hence, finding the eigenvectors is equivalent to finding the coefficients α_i

Rewrite PCA using Kernel "trick"

- That is, $u_j = \sum_{l=1}^m \alpha_{jl}\phi(x^{(l)})$ for each $j = 1, \ldots, k$
- Let's find an expression for the α_j's
- By substituting this back into the equation we get:

$$\frac{1}{m}\sum_{i=1}^m \phi(x^{(i)})\phi(x^{(i)})^\top \left(\sum_{l=1}^m \alpha_{jl}\phi(x^{(l)})\right) = \lambda_j \sum_{l=1}^m \alpha_{jl}\phi(x^{(l)})$$

- Rewrite this as

$$\frac{1}{m}\sum_{i=1}^m \phi(x^{(i)}) \left(\sum_{l=1}^m \alpha_{jl}K(x^{(i)}, x^{(l)})\right) = \lambda_j \sum_{l=1}^m \alpha_{jl}\phi(x^{(l)})$$

- Multiply by $\phi(x^{(s)})$ from the left to get

$$\frac{1}{m}\sum_{i=1}^m \phi(x^{(s)})^\top\phi(x^{(i)}) \left(\sum_{l=1}^m \alpha_{jl}K(x^{(i)}, x^{(l)})\right) = \lambda_j \sum_{l=1}^m \alpha_{jl}\phi(x^{(s)})^\top\phi(x^{(l)})$$

Rewrite PCA using Kernel "trick"

- By plugging in the kernel and rearranging we get:

$$K^2\alpha_j = m\lambda_j K\alpha_j \implies K\alpha_j = m\lambda_j\alpha_j$$

  We can remove a factor of K from both sides of the matrix (this only affect the eigenvectors with zero eigenvalue, which will not be a principal component anyway):

- Since $\|u_j\|^2 = 1$, we have that

$$\sum_{i=1}^{m}\sum_{l=1}^{m}\alpha_{jl}\alpha_{ji}\phi(x^{(l)})^\top\phi(x^{(i)}) = 1 \implies \alpha_j^\top K\alpha_j = 1 \implies \lambda_j m\alpha_j^\top\alpha_j = 1$$

- Hence, for some feature vector x its projection on to the principal components is

$$\phi(x)^\top u_j = \sum_{i=1}^{m}\alpha_{ji}\phi(x)^\top\phi(x^{(i)}) = \sum_{i=1}^{m}\alpha_{ji}K(x, x^{(i)})$$

Summary of the Kernel Trick

- We showed that the eigenvectors (principal components) can be expressed as linear combination of the features with some coefficients α_i
- We showed that α_i's are eigenvectors of the kernel matrix, and have some normalization property (norm equal to $1/(m\lambda_i)$)
- Showed the projection of a feature vector onto a pricipal component can be obtained via the kernel matrix

$$\phi(x)^\top u_j = \sum_{i=1}^{m} \alpha_{ji} K(x, x^{(i)})$$

Normalizing the Feature Space

- In general $\phi(x^{(i)})$ may not be zero mean
- We can recenter the features just like before

$$\varphi(x^{(i)}) = \phi(x^{(i)}) - \frac{1}{m}\sum_{j=1}^{m}\phi(x^{(j)})$$

- The new kernel is $\tilde{K}(x^{(i)}, x^{(j)}) = \varphi(x^{i)})^\top \varphi(x^{(j)})$ and without going through the details, we can derive an expression for the new kernel in matrix form

$$\tilde{K} = K - \frac{1}{m}K\mathbf{1} - \frac{1}{m}\mathbf{1}K + \frac{1}{m^2}\mathbf{1}K\mathbf{1}$$

where $\mathbf{1}$ is a matrix with all elements equal to one.

Summary of Kernel PCA

- Pick a kernel
- Construct the normalized kernel matrix of the data:

$$\tilde{K} = K - \frac{1}{m}K\mathbf{1} - \frac{1}{m}\mathbf{1}K + \frac{1}{m^2}\mathbf{1}K\mathbf{1}$$

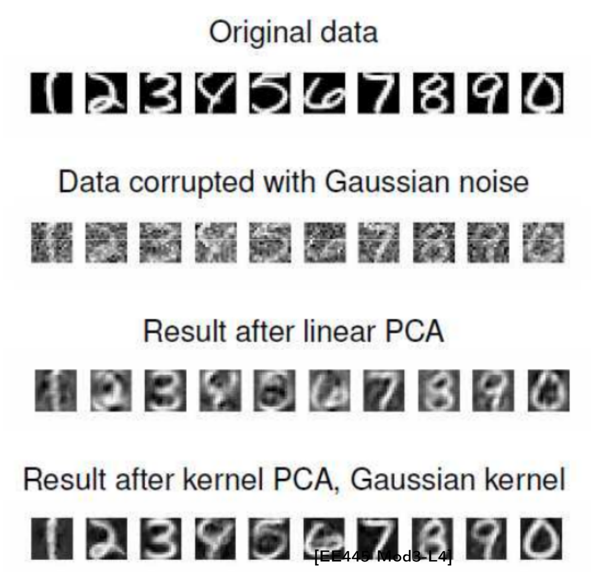
- Solve an eigenvalue problem:

$$\tilde{K}\alpha_i = \lambda_i \alpha_i$$

- For any data point (new or old), we can represent it as

$$y^{(j)} = \sum_{i=1}^{m} \alpha_{ji} K(x, x^{(i)}),\ j = 1, \ldots, k$$

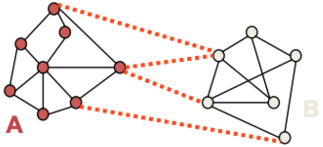
Example: De-noising images

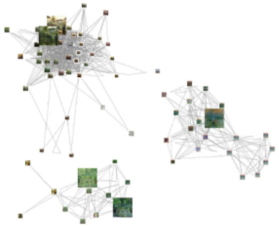

Original data
Data corrupted with Gaussian noise
Result after linear PCA
Result after kernel PCA, Gaussian kernel

Part 3: Spectral Clustering

What is spectral clustering?

- Goal is to group points based on links in a graph





How to create the graph?

- Given a distance metric, we compute the distance between each of our features
- It is common to use a Guassian Kernel (RBF) to compute the similarity between features:

$$s_{ij} = S(i,j) = \exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{\sigma^2}\right)$$

- There are several popular constructions to transform a given set $x^{(1)}, \ldots, x^{(m)}$ of data points with pairwise similarities s_{ij} or pairwise distances d_{ij} into a graph.
  - fully connected graph: Here we simply connect all points with positive similarity with each other, and we weight all edges $w_{ij} = s_{ij}$
  - k-nearest neighbor graphs: Here the goal is to connect vertex v_i with vertex v_j if v_j is among the k-nearest neighbors of v_i.
  - The ε-neighborhood graph: Here we connect all points whose pairwise distances are smaller than ε

Graph Review

- Consider a graph $G = (E, V)$ where $V = \{v_1, \ldots, v_n\}$ is the vertex set and E is the set of
- the graph G is weighted—that is each edge between two vertices v_i and v_j carries a non-negative weight $w_{ij} \geq 0$.
- The weighted adjacency matrix (**Module 1**) of the graph is the matrix W
- The degree of vertex v_i is $d_i = \sum_{j=1}^{n} w_{ij}$ and the matrix D is the degree matrix
- Graph Laplacian:

$$L = D - W$$

Spectral Clustering Algorithm

Input: Similarity matrix $S \in \mathbb{R}^{m\times m}$ (pairwise similarities between edges) and number of clusters k

- **Step 1**: Build W, the (weighted) adjacency matrix of the corresponding graph: e.g.,
- **Step 2**: Compute the graph Laplacian $L = D - W$
- **Step 3**: Compute the first k eigenvectors $u_1, \dots, u_k$ of L
  `CAUTION!:` Here the k-first eigenvectors are those corresponding to the smallest eigenvalues of L.
- **Step 4**: Let $U \in \mathbb{R}^{m\times k}$ be the matrix containing all the u_i's as columns
- **Step 5**: For $i = 1, \dots, m$, let $y^{(i)} \in \mathbb{R}^k$ be the vector corresponding to the i-th row of U
- **Step 6**: Cluster the points $(y^{(i)}$ for $i = 1, \dots m$ with the k-means algorithm into clusters $C_1, \dots, C_k$

Output: Clusters $A_1, \dots, A_k$ with $A_i = \{j \mid y^{(j)} \in C_i\}$

Example

go to `Mod3-Lec4.ipynb`