Lecture 7 — April 21

Lecturer: Lester Mackey *Scribe: Zhou Fan, Subhabrata Sen*

7.1 Spectral Clustering

Last time, we introduced the notion of spectral clustering, a family of methods well-suited to finding non-convex/non-compact clusters. Recall that the input to a spectral clustering algorithm is a similarity matrix $S \in \mathbb{R}^{n \times n}$ and that the main steps of a spectral clustering algorithm are

1. Construct the graph weights $W = g(S) \in \mathbb{R}^{n \times n}$.
2. Form the graph Laplacian $L = f(W) \in \mathbb{R}^{n \times n}$.
3. Find the bottom k eigenvectors $U = [u_1, \cdots, u_k]$ of L.
4. Cluster rows $\tilde{x}_i$ of U using k-means.

7.1.1 Graph Laplacian properties

In the last lecture, we discussed three of the most common choices of graph Laplacian L. Today, we will focus on the random walk normalized Laplacian $L_{rw} = I - D^{-1}W$, where

$$D = \text{diag}(d_1, \cdots, d_n) \quad \text{for} \quad d_i = \sum_{j=1}^{n} w_{ij}$$

is the degree matrix and elucidate some of its key properties that make it suitable for clustering.

Suppose we form a weighted graph G with our data-points being the vertices and the edge weights being specified by the weight matrix W. Let us first consider the special case when the graph G has exactly k components with vertex sets $A_1, \cdots, A_k$. Then the Laplacian L_{rw} has the following properties.

- For each connected component j, the indicator vector $V_j = (v_{j1}, \cdots, v_{jn})$, with $v_{jl} = \mathbb{I}\,(\text{vertex } l \in A_j)$ is an eigenvector of L_{rw} with eigenvalue 0.
- The other $n - k$ eigenvalues are > 0. (Note that this implies in particular that L_{rw} has all real eigenvalues.)

This motivates the spectral clustering algorithm. In the ideal case, where our selection of weights segments our data points into k connected components corresponding to true underlying data clusters, the algorithm eigenvector step returns the exact indicators for each component and hence for each cluster. Thus the clusters may be easily identified by k-means

in the next step. However, in practice, all clusters are typically somewhat connected. Our hope is that when the weights between clusters components are relatively small, then the bottom eigenvectors will still reflect the cluster structure.

Next we look at some specific examples of spectral clustering in simulated and real datasets. We will refer to the plots in the lecture slides whenever applicable.

7.1.2 Examples

Example 1. The first example uses data simulated from a mixture of 4 Gaussian distributions. The weight matrix is derived from 10 nearest neighbours. See the next section for a discussion of weight matrix choices. Spectral clustering was performed with two different graph Laplacians. Please refer to the slides for the plots. In this case, the Gaussians are well-separated and the points from each distribution are separated into four connected components in the resulting graph. We see our theory borne out, as the bottom four eigenvalues are all zero (since there are four connected components), and the remaining eigenvalues are positive. Moreover, the eigenvectors exactly indicate to which cluster each datapoint belongs.

Example 2. The second example again uses spectral clustering with the same mixture of Gaussians but using a complete graph weight matrix (see the next section for a discussion of weight matrix choices) that results in all vertices belonging to a single component. Now there is only one zero eigenvalue, and the first eigenvector is constant as our analysis of L_{rw} demands. The next three smallest eigenvalues are non-zero but still small. Moreover, the next three eigenvectors deviate from cluster indicators but still retain enough information about cluster belonging to exactly identify to which cluster each datapoint belongs.

Example 3. The third example discusses a data with roughly two annular clusters, which are roughly concentric. This is a scenario where we expect spectral clustering to outperform k means clustering. We see that this is exactly the outcome in this case. k-means clustering, being adapted to the detection of circular or elliptic clusters, fails to recover the structure in this case. Spectral clustering, on the other hand, correctly recovers the underlying structure. In the first concentric circle example, we see the clean indicator functions that we have some to expect from two separated components. In the second noisy example, we see that the eigenvectors are also noisier with points near the boundary of the two clusters having ambiguous values. Nevertheless, spectral clustering still captures the cluster structure far better than k-means in this setting.

Spectral clustering is used widely in practice for image segmentation. We look at an example of the outcome of spectral clustering on an image. It can be seen that this algorithm is successful in identifying the main features in the image correctly.

7.1.3 Forming the weight matrix

We will next discuss the important issue of constructing the graph edge weight matrix given a similarity matrix. The aim of this construction is to capture local similarity (i.e., to identify the most similar pairs of datapoints and ignore or downweight any other pairs). Some of the

most common constructions are listed below. Keep in mind that each construction treats the datapoints as vertices of a graph, retains some of the edges in the graph, and assigns weights to these edges.

- The ϵ-neighbourhood graph: Here we join any two vertices for which the similarity is $> \epsilon$. This algorithm assigns the same weight to each edge.
- The k-nearest neighbour graph: In this algorithm, we join two vertices i and j if i is among the k most similar vertices to j or vice versa. The edge-weights are assigned according to similarity.
- Mutual k-nearest neigbour graph: This scheme is similar to the previous algorithm. We join two vertices i and j if i is among the k most similar vertices to j and vice versa. The edge-weights are assigned according to similarity.
- Fully connected graph: This algorithm considers the complete graph. This method is useful when weights are assigned using "local" similarity measures (such as those assigned by the Gaussian kernel), which decay rapidly with decreasing similarity.

Example 4. We next looked at the different graphs formed by the different algorithms on a dataset. (Refer to the slides for the plots)

7.1.4 Optimality?

Finally, it is interesting to know whether spectral clustering is optimal in any sense. This would provide further motivation for spectral clustering. It may be shown that a variant of spectral clustering minimizes a relaxation of the normalized cut graph partitioning criterion (Shi and Malik, 2000). A variant of the same algorithm based on L_{rw} approximately minimizes the transition probability of the random walk on the weighted graph from one cluster to another. Moreover, the asymptotic consistency of spectral clustering has been studied under certain statistical models (Rohe/Chatterjee/Yu '10- Spectral clustering and the high-dimensional stochastic block model).

7.2 Dimensionality Reduction

Having focused so far on the clustering or latent class paradigm of unsupervised learning, we now turn our attention to a second key paradigm, that of **dimensionality reduction** or **latent feature modeling**. The general (and somewhat vague) goal in dimensionality reduction is to find low-dimensional representation that captures the "essence" of higher-dimensional data points. There are various motivations:

- **Compression** for improved storage and computational complexity.
- **Visualization** for improved human understanding of data—difficult to plot and interpret data in more than 3 dimensions.
- **Noise reduction** to ameliorate noisy and infrequent or missing measurements.

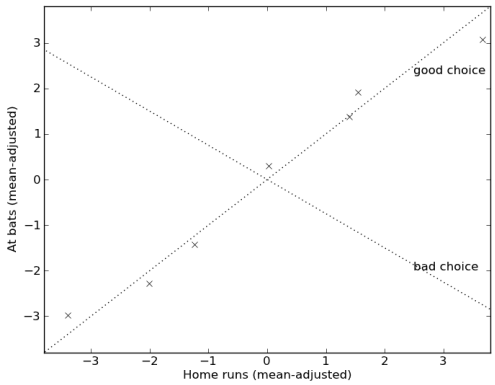
- **Preprocessing** for supervised learning tasks—reduced/denoised representations may lead to better performance or act as regularization to reduce overfitting.
- **Anomaly detection** to characterize normal data values and distinguish them from outliers.

We will first focus on **linear dimensionality reduction**, in which, given high-dimensional data points $x_i \in \mathbb{R}^p$ (e.g., images of faces in $\mathbb{R}^{361}$), we aim to derive "useful" representations $z_i = U^T x_i \in \mathbb{R}^k$ where $U^T \in \mathbb{R}^{k \times p}$ is a **linear mapping** into a low-dimensional space. The different methods we consider will arise from different notions of the word "useful."

7.2.1 Principal Component Analysis (PCA)

Principal component analysis (PCA) is the most common approach to linear dimensionality reduction. The input in this case is a dataset with mean-centered datapoints, $x_1, \ldots, x_n \in \mathbb{R}^p$ obeying $\bar{x} = \frac{1}{n}\sum_{i=1}^n x_i = 0$,[1] and the still somewhat vague goal is to produce useful derived variables $z_i = U^T x_i \in \mathbb{R}^k$, $k < p$. We will make this precise in the context of an example:

Example 5 (2D baseball player statistics)**.**



Suppose that for various baseball players we measure two statistics, at bats and home runs scored, and plot their values as above. Since these variables are very correlated, we suspect that a 1D summary of this data would suffice, and we aim to find a 1D projection that captures most of the data variation. The line labeled "good choice" appears to be a particularly good choice for this data in that even after projecting each datapoint onto the line, most of the variation in the data remains. Meanwhile, were we to project these points onto bad choice, most of the projected values would cluster near zero and lose most of their variability.

[1] If your datapoints are not mean-centered to begin with, one begins by subtracting away the empirical mean.

Extracting one PC

Let us extend this 2D intuition to the more general case of extracting a single principal component (i.e., $k = 1$) from data in $\mathbb{R}^p$. Each 1D subspace S_u is identified by a unit vector u (with $\|u\|_2 = 1$). Projection onto S_u is given by $\tilde{x}_i = uu^T x_i$, and the magnitude of the projection is $\|\tilde{x}_i\|_2 = \|u\|_2 u^T x_i = u^T x_i$. Let $z_{i1} = u^T x_i$ denote this derived variable.

The goal of PCA is to maximize the empirical variance of the derived variable z_1 over choices of projection direction u, i.e.

$$u_1^* = \underset{u:\|u\|_2=1}{\operatorname{argmax}} \frac{1}{n}\sum_{i=1}^n z_{i1}^2 = \underset{u:\|u\|_2=1}{\operatorname{argmax}} \frac{1}{n}\sum_{i=1}^n (u^T x_i)^2.$$

The entries of u_1^* are called the **first principal component (PC) loadings**, the entries $z_{i1}^* = u_1^{*T} x_i$ are the **first PC scores**, and $(z_{11}^*, \ldots, z_{n1}^*)$ is the **first PC**.

Letting $X = \begin{bmatrix} -x_1- \\ \vdots \\ -x_n- \end{bmatrix}$, we may write the above optimization problem as

$$\max_{u:\|u\|_2=1} u^T \left(\sum_{i=1}^n \frac{1}{n} x_i x_i^T\right) u = \max_{u:\|u\|_2=1} \frac{u^T X^T X u}{n} = \lambda_1\left(\frac{X^T X}{n}\right),$$

where $\lambda_1\left(\frac{X^T X}{n}\right)$ is the leading eigenvalue of the covariance matrix $\frac{X^T X}{n}$. (This may be derived using Lagrange multipliers.) As a result, u_1^*, the argmax, is the corresponding leading eigenvector of $\frac{X^T X}{n}$. Thus, one obtains the first principal component of X by multiplying X by the first eigenvector of its covariance matrix.

Extracting k PCs

For $k > 1$, PCA produces a sequence of PC loadings $[u_1^*, \ldots, u_k^*] := U^*$ and PC scores $z_{ij}^* = u_j^{*T} x_i$ according to

$$u_j^* = \underset{u:\|u\|_2=1, \frac{1}{n}\sum_{i=1}^n z_{ij} z_{il}=0 \ \forall l<j}{\operatorname{argmax}} \frac{1}{n}\sum_{i=1}^n z_{ij}^2.$$

That is, PCA maximizes the variance of the j^{th} derived variable Z_j under the constraint that it is uncorrelated with the previous derived variables $\{Z_l\}_{l<j}$. We may write this as

$$u_j^* = \underset{u:\|u\|_2=1, u_j^T \frac{X^T X}{n} u_l=0 \ \forall l<j}{\operatorname{argmax}} \frac{u_j^T X^T X u_j}{n}.$$

This is another standard eigenvalue problem, so that u_j^* is the j^{th} leading eigenvector of $\frac{X^T X}{n}$, and the objective value for u_j^* is $\lambda_j\left(\frac{X^T X}{n}\right)$, the j^{th} leading eigenvalue. Thus, PCA finds the k leading eigenvectors of the data covariance matrix.

Variance explained

In PCA contexts, one often speaks of the **variance explained** by $[u_1^*, \ldots, u_k^*]$. This is precisely the sum of the PC variances, which we have shown to be $\sum_{j=1}^k \lambda_j \left(\frac{X^T X}{n}\right)$. The **fraction of the total variance explained** is

$$\frac{\sum_{j=1}^k \lambda_j \left(\frac{X^T X}{n}\right)}{\text{sum of input } x\text{-component variances}} = \frac{\sum_{j=1}^k \lambda_j \left(\frac{X^T X}{n}\right)}{\text{tr}\left(\frac{X^T X}{n}\right)} = \frac{\sum_{j=1}^k \lambda_j \left(\frac{X^T X}{n}\right)}{\sum_{j=1}^n \lambda_j \left(\frac{X^T X}{n}\right)}.$$

Second PCA perspective

While we have derived PCA from a variance maximization perspective, a second fruitful perspective arises from considering the reconstruction error of projections. A projection of x_i onto a k-dimensional subspace can always be written as $UU^T x_i = U z_i$ for some $U \in \mathbb{R}^{p \times k}$ with orthonormal columns $U^T U = I$. PCA chooses the projection which minimizes the data reconstruction error:

$$U^* = \underset{U \in \mathbb{R}^{p \times k} : U^T U = I}{\text{argmin}} \sum_{i=1}^n \|x_i - UU^T x_i\|_2^2.$$

This problem is also solved by taking the columns of U to be the k leading eigenvectors of $\frac{X^T X}{n}$.

Note however that computing the covariance $\frac{X^T X}{n}$ is often an expensive operation, requiring $\Omega(np^2)$ time. An alternative computational procedure is the following: if $U_{X^T} \Sigma_{X^T} V_{X^T}^T$ is the singular value decomposition of X^T, where U_{X^T} and V_{X^T} contain the orthogonal singular vectors and Σ_{X^T} is the diagonal matrix of singular values, then the PC loadings are given by the k leading left singular vectors. This requires $O(npk)$ computational time.