LECTURE 16
Dimension reduction and embeddings

A key idea in much of high-dimensional data analysis is that the underlying structure of the data is low dimensional so even if the $X \subseteq \mathbb{R}^p$ the underlying degrees-of-freedom or the support of the data is low dimensional say, $d \ll p$. The key questions are how to find this low dimensional structure, how to use the low dimensional structure, and what assumptions are made in extracting this low dimensional structure from data. The key tool we will used to address these questions are spectral methods. Another aspect of dimensionality reduction is the idea of an embedding. An informal way of thinking about an embedding is finding points in a low dimensional space that satisfy pairwise similarity metrics. Embeddings are often used to take data that are inherently non-numerical and turn them into a set of vectors, we will see examples of this in natural language processing.

16.1. Spectral methods

For the purpose of this lecture by spectral methods we will mean an eigen-decomposition of a positive semi-definite symmetric operator. We will construct different operators from data and these different operators will correspond to different assumptions about the data or different quantities we want to preserve about the data.

16.1.1. Linear dimension reduction

We start with standard linear dimension reduction or Principal Components analysis. Consider the data matrix $\mathbf{X}$ that is $p \times n$

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \cdots & \mathbf{x}_n \end{bmatrix}.$$

Denote $\overline{\mathbf{X}}$ as the $n \times p$ mean matrix

$$\overline{\mathbf{X}} = \begin{bmatrix} \overline{\mathbf{x}} & \cdots & \overline{\mathbf{x}} \end{bmatrix},$$

where $\overline{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i$. We will denote the centered data matrix $\mathbf{Y} = \mathbf{X} - \overline{\mathbf{X}}$. The empirical covariance matrix

$$\widehat{\mathbf{\Sigma}} = \frac{1}{n}\mathbf{Y}^T\mathbf{Y}$$

is an estimate of the covariance matrix $\mathbf{\Sigma}$, and each element $\widehat{\Sigma}_{ij}$ is an estimate of the covariance of variable i and variable j.

The idea behind PCA is to compute the eigenvalues and eigenvectors of the covariance matrix, this is a spectral decomposition of an matrix (operators are infinite dimensional analogs of matrices). So one computes $(\lambda_i, \mathbf{v}_i)_{i=1}^{p}$ where

$$\lambda_i \mathbf{v}_i = \widehat{\mathbf{\Sigma}}\, \mathbf{v}_i,$$

and the λ_i are ordered largest to smallest.

Dimension reduction is carried out by projecting the data $\mathbf{X}$ onto the eigenvectors corresponding to the top k eigenvalues

$$\mathbf{Z} := \mathbf{V}_k\, \mathbf{X}$$

where

$$\mathbf{V}_k = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_k \end{bmatrix}.$$

If the data are normally distributed the variance explained by each component $\mathbf{v}_i$ is λ_i or

$$\text{var}_{\mathbf{X}}[\mathbf{v}_i\, \mathbf{X}] = \lambda_i$$

where $(\lambda_i, \mathbf{v}_i)_{i=1}^{p}$ are the eigenvalues and eigenvectors of the covariance matrix $\mathbf{\Sigma}$.

There is an equivalent approach to dimension reduction using a spectral decomposition of a related operator, that of the empirical Gram matrix which is $n \times n$ and

$$\widehat{\mathbf{G}} = \frac{1}{p}\mathbf{Y}^T\mathbf{Y}$$

is an estimate of the gram matrix, and each element $\widehat{\mathbf{G}}_{ij}$ is an estimate of the correlation of individual i and individual j. Given the matrix $\widehat{\mathbf{G}}$ one computes the eigenvalues and eigenvectors $(\lambda_i, \mathbf{v}_i)_{i=1}^{n}$ where

$$\lambda_i \mathbf{v}_i = \widehat{\mathbf{G}}\, \mathbf{v}_i,$$

then projects onto the eigenvectors corresponding to the top k eigenvalues provides dimension reduction

$$\mathbf{Z} := \mathbf{V}_k^T\, \widehat{\mathbf{G}}$$

where

$$\mathbf{V}_k = \begin{bmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_k \end{bmatrix}.$$

An interesting point about the spectral decomposition of the Gram matrix is that one could have started with simply the pairwise relations and did not need the data to compute the low dimensional projection $\mathbf{Z}$.

16.1.2. Nonlinear dimension reduction and embeddings

We will consider nonlinear dimension reduction from the perspective of embeddings of a metric space. In the context of metric spaces an embedding is a mapping $\phi : X \to Y$ of metric spaces is called an embedding with distortion C if

$$\text{dist}_X(u, v) \leq \text{dist}_Y(\phi(u), \phi(v)) \leq C\text{dist}_X(u, v)$$

where $\text{dist}_X(\cdot, \cdot)$ and $\text{dist}_Y(\cdot, \cdot)$ are distance metrics in spaces X and Y respectively. An important idea here is that we only need distances to do dimension reduction, we will use this when we consider some applications in the next subsection.

16.1.2.1. *Kernel based dimension reduction.* A common approach to nonlinear dimension, called kernel PCA, is to define a similarity kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that is positive (semi) definite.
That is for any $n \in \mathbb{N}$ and $\mathbf{x}_1, ..., \mathbf{x}_n \in \mathcal{X}$ and any $c_1, ..., c_n \in \mathbb{R}$ the following holds for a positive definite kernel

$$\sum_{i,j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) > 0$$

and for a positive semidefinite kernel

$$\sum_{i,j=1}^{n} c_i c_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0.$$

The idea of a kernel function is to encode how (dis)similar two points are or one can think of the kernel function as defining a metric on the space $\mathcal{X}$. Examples of kernel functions $k(\mathbf{u}, \mathbf{v})$ follow

$$\exp\left(-\kappa \|\mathbf{u} - \mathbf{v}\|^2\right), \quad \frac{1}{\kappa + \|\mathbf{u} - \mathbf{v}\|}, \quad \mathbf{u}^T\mathbf{v},$$

note that the last kernel $\mathbf{u}^T\mathbf{v}$ gives us back the Gram matrix and classical linear dimension reduction.

Given observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and a kernel function we can compute the kernel matrix $\mathbf{K}$ where

$$\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j).$$

This is followed by a spectral decomposition of the kernel matrix or computing $(\lambda_i, \mathbf{v}_i)_{i=1}^n$ where

$$\lambda_i \mathbf{v}_i = \mathbf{K}\,\mathbf{v}_i.$$

and projecting onto the eigenvectors corresponding to the top k eigenvalues

$$\mathbf{Z} := \mathbf{V}_k^T\,\mathbf{K}$$

where

$$\mathbf{V}_k = \begin{bmatrix} \\ \mathbf{v}_1 & \cdots & \mathbf{v}_k \\ \\ \end{bmatrix}.$$

The above map $\phi : \mathbf{X} \to \mathbf{Z}$ is an embedding with the following property. For the data $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and kernel function k the embedding ϕ as defined above has the property that for any $i, j = 1, ..., n$

$$L\,k(\mathbf{x}_i, \mathbf{x}_i) \leq \|\mathbf{z}_i - \mathbf{z}_j\| \leq C\,L\,k(\mathbf{x}_i, \mathbf{x}_i),$$

for some constant $L > 0$.

The crux of the idea in nonlinear dimension reduction is the pairwise similarity metric defined by a nonlinear kernel is approximately preserved by the Euclidean distance in the embedded space.

16.1.3. Word embeddings

A common use of embeddings is transforming a collection of words in a document to a vector space. The idea here is given a dictionary of $|V|$ words, where $|V|$ is in the tens of thousands or millions, how do I represent each word as a point in a low-dimensional vector space. If one can represent a word as a low dimensional vector hen one can apply stadard machine learning methods to text, a collection of words. There are several word embedding approaches that fall under the rubric of word2vec methods. We will discuss these approaches in this subsection.

The initial encode of the collection of words is called the one-hot vector where every word is encoded as an $\mathbb{R}^{|V|}$ vector with one element as one and the remaining elements as zero. This representation considers each word as a completely independent entity. An important point about this representation is that all words are equidistant from each other, so this representation does not encode any semantic information. Also it is a very high-dimensional space to consider. Our goal is to reduce the size of the space from $\mathbb{R}^{|V|}$ to something much smaller and encode semantic information.

The key idea in word2vec is that we consider a metric where words that show up together in the same context should embed close to each other in a vector space. This metric embedding can be executed via spectral clustering or autoencoders.

16.1.3.1. *Spectral methods.* A simple idea is to generate a co-occurrence matrix and use a spectral decomposition of the co-occurrence matrix as the word embeddings. Given a text corpus one first defines a window size of k words. One then constructs a symmetric $|V| \times |V|$ co-occurrence matrix $\mathbf{O}$ where $\mathbf{O}_{ij}$ is the number of times that word i and word j co-occur in a sentence within k words of each other. We can now consider the co-occurrence matrix the same way we considered the kernel matrix in the previous subsection and our word embedding is

$$\mathbf{Z} := \mathbf{V}_k^T \, \mathbf{O}$$

where

$$\mathbf{V}_k = \begin{bmatrix} \\ \mathbf{v}_1 & \cdots & \mathbf{v}_k \\ \\ \end{bmatrix},$$

with

$$\lambda_i \mathbf{v}_i = \mathbf{O}\,\mathbf{v}_i.$$

This is not a bad approach but the most commonly used embedding methods are based on autoencoders and deep learning.

16.1.3.2. *Autoencoders.* The idea behind autoencoders is to use a deep neural network with a particular architecture to compute the embedding. If we consider a neural network as a function approximating blackbox $f : x \to y$ then we can consider the n one-hot vectors as inputs $(\mathbf{x}_1, ..., \mathbf{x}_n)$ and the n one-hot vectors as outputs $(\mathbf{y}_1, ..., \mathbf{y}_n)$ with the neural network trained to learn the outputs as accurately as possible, we denote the predicted outputs as $(\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_n)$. The neural network architecture will have multiple layers with a middle layer with far fewer nodes or units than the input or output layers (see figure below), recall the input and output layers have $|V|$ units. Once the network is trained the activation weights for the i-th one-hot vector is the word embedding for the i-th word. Note the word embedding will be in $\mathbb{R}^h$, where h is the number of units at the bottleneck level.

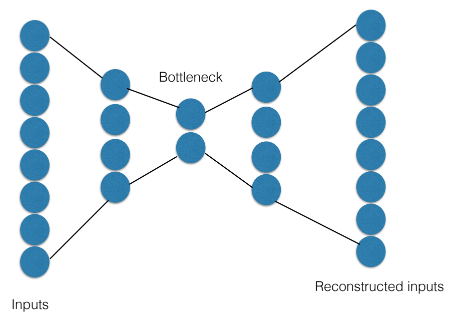

Figure 1. Network architecture for an autoencoder. All layers are fully connected.

The two approaches to word2vec. The word2vec algorithm does not use the exact autoencoding construction but to very closely related constructions called: the Continuous Bag-of-Words Model (CBOW) and Continuous Skip-gram Model.

The CBOW model predicts the current word based on the words around it so the input for the i-th word the input vector is a concatenation of all words within a window of k neighboring words in a sentence so if $\mathbf{x}_{ik} = \{\mathbf{x}_k \in \mathcal{C}_k \text{ and} k \neq i\}$ where $\mathcal{C}$ are all the words in the corpus within k words of the i-th word, then the input $\mathbf{x}_i = (\mathbf{x}_i^1, ..., \mathbf{x}_i^{k-1})$ where the set $(\mathbf{x}_i^1, ..., \mathbf{x}_i^{k-1})$ are the $k-1$ words neighboring $\mathbf{x}_i$. The output for the CBOW model is simply the ith word so $\mathbf{y}_i = \mathbf{x}_i$.

The Continuous Skip-gram Model flips the idea of the CBOW by trying to classify the i-th word based on the k neighboring words. This basically flips the input and output of the CBOW model. so in the Skip-gram Model $\mathbf{x}_i$ is $\mathbf{y}_i$ of the CBOW model and $\mathbf{y}_i$ is $\mathbf{x}_i$ of the CBOW model.