Lecture 5
Dimensionality Reduction and Data Visualization

Bingqing Cheng

January 2020

Outline

Real data often live in high dimensional spaces, which makes them difficult to analyze and interpret using the human brain. As such, dimensionality-reduction methods have long been sought after for all sorts of data analysis applications. In chemistry, physics and materials science, dimensionality reduction has been used to provide a simplified, coarse-grained representation of the structure and dynamics of complex systems, from proteins to nanoparticles.

This lecture briefly reviews some of the dimensionality-reduction methods, which can be seen as preparations for the next lectures that are geared towards classification and visualization of atomistic systems including molecules and materials. We will cover the general concepts behind dimensional reduction, the most fundamental dimensional reduction method–principal component analysis (PCA), and some more advanced extensions.

1 The need for dimensionality reduction

Many machine learning problems involve thousands or even millions of features for each training instance. Not only do all these features make training extremely slow, but they can also make it much harder to find a good solution, as we will see. This problem is often referred to as the curse of dimensionality.

Another problem with the high-dimensional data is that they lives near the edge of sample space. Geometry in high-dimensional space can be counterintuitive. One example that is pertinent to machine learning is the following. Consider data distributed uniformly at random in a D-dimensional hypercube $C = [e/2, e/2]^D$, where e is the edge length. Consider also a D-dimensional hypersphere S of radius $e/2$ centered at the origin and contained within C. The probability that a data point x drawn uniformly at random in C is contained within S is well approximated by the ratio of the volume of S to that of $C : p(||x|| < e/2) \approx (1/2)^D$. Thus, as the dimension of the feature space D increases, p goes to zero exponentially fast.

Fortunately, real-world data is not random or uniformly distributed. In fact, real data usually lives in a much lower dimensional space than the original space in which the features are being measured. This is sometimes referred to as the "blessing of non-uniformity" (in opposition to the curse of dimensionality). Data will typically be locally smooth, meaning that a local variation of the data will not incur a change in the target variable. This idea is central to statistical physics and field theories, where properties of systems with an astronomical number of degrees of freedom can be well characterized by low-dimensional order parameters or intrinsic degrees of freedom. Physically speaking, this means only a small number of features affect a physical phenomenon or system considerably. For example, consider a peptide molecule, which is a short chain of amino acids. For representing this molecule, you can often drop the coordinates of the hydrogen atoms without losing much information. Additionally, instead of specifying all the compositions and coordinates of the functional groups including all the carbon atoms in a benzene ring, perhaps just the name and the position of the functional groups is sufficient.

Apart from speeding up training, dimensionality reduction is also extremely useful for data visualization. Reducing the number of dimensions down to two (or three) makes it possible to plot a condensed view of a high-dimensional training set on a graph and often gain some important insights by visually detecting patterns, such as clusters. Moreover, visualization is essential to communicate your conclusions to people, and for making nice figures for presentations and manuscripts!

2 Principal component analysis (PCA)

Principal component analysis (PCA) is probably the simplest of all dimensionality-reduction schemes, and can be taken as a paradigmatic example to summarize some of the fundamental ideas that underpin other dimensionality reduction approaches.

2.1 Prerequisite in Statistics and Mathematics

Variance Variance σ^2 is a measure of the spread of data in a one-dimensional data set $\{x_{i=1\ldots N}\}$. The formula is

$$\sigma^2 = \frac{\sum_i (x_i - \bar{x})^2}{N-1} \tag{1}$$

Covariance Variance only operate for one-dimensional data set, for measuring the spread of each dimension of a high-dimensional data set independently of the other dimensions. However, it is useful to have a similar measurement find out how much different dimensions vary from the mean with respect to each other. The covariance matrix offers just such a measurement.

For a high-dimensional data set $\{\mathbf{X}_{i=1\ldots N}\}^D$, with each data point $\mathbf{X}_i = \{X_i^{j=1\ldots D}\}$ is a D-dimensional vector, the covariance of a pair of dimensions j

and j':

$$C_{jj'} = \frac{\sum_i (X_i^j - \bar{X^j})(X_i^{j'} - \bar{X^{j'}})}{N-1}. \tag{2}$$

Correlation Pearson's correlation coefficient (PCC) is a measure of the linear correlation between two variables. According to the Cauchy–Schwarz inequality it has a value between +1 and 1, where 1 is total positive linear correlation, 0 is no linear correlation, and 1 is total negative linear correlation. When applied to a D-dimensional data set $\{\mathbf{X}\}^D$ for measuring the correlation between a pair of dimensions j and j', PCC can be expressed as

$$\rho_{jj'} = \frac{C_{jj'}}{\sigma^j \sigma^{j'}}, \tag{3}$$

where σ^j and $\sigma^{j'}$ are the standard deviations (i.e. square root of variances) of the data set in the dimensions j and j', respectively.

Eigenvalues and eigenvectors Consider the high-dimensional data set $\{\mathbf{X}_{i=1\ldots N}\}^D$ that lives in the D-dimensional space, the covariance matrix $\mathbf{C}$ takes a $D \times D$ form. The eigenvalues $\{\lambda^j\}$ and the corresponding eigenvectors $\{\mathbf{v}^j\}$ of the matrix fulfills

$$\mathbf{C}\mathbf{v}^j = \lambda^j \mathbf{v}^j \tag{4}$$

for $j = 1 \ldots D$. One can find the eigenvalues $\{\lambda^j\}$ by solving

$$det(\mathbf{C} - \lambda \mathbf{I}) = 0, \tag{5}$$

where $\mathbf{I}$ identity matrix, and $det(\ldots)$ indicates the determinant of a matrix. The eigenvectors $\{\mathbf{v}^j\}$ can then be determined by

$$(\mathbf{C} - \lambda^j \mathbf{I})\mathbf{v}^j = 0. \tag{6}$$

2.2 A graphical understanding of PCA

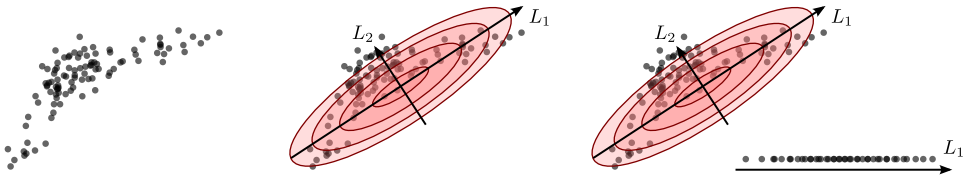

Figure 1: A schematic of PCA.

This is the rather simple idea behind PCA: it seems reasonable to select the axis that preserves the maximum amount of variance, as it will most likely lose less information than the other projections. Another way to justify this choice

is that it is the axis that minimizes the mean squared distance between the original dataset and its projection onto that axis.

PCA identifies the axis that accounts for the largest amount of variance in the training set. In Figure 1, it is the black arrow L_1. It also finds a second axis L_2, orthogonal to the first one, that accounts for the largest amount of remaining variance. In this 2D example there is no choice: its direction is determined by L_1. If it were a higher-dimensional dataset, PCA would also find a third axis, orthogonal to both previous axes, and a fourth, a fifth, and so on—as many axes as the number of dimensions in the dataset.The jth axis is called the jth principal component (PC) of the data. In Figure 1, the first PC is along L_1, and the second PC is the axis on which vector L_2 lies.

As the key last step, we project the coordinates on to the principal components. In the example of Figure 1, the one-dimensional representation of the input data is given the coordinates of the data along the L_1 axis.

2.3 Math behind PCA

Now let's express the above picture mathematically. $\{\mathbf{X}_i\}^D$ denotes the high dimensional data in the D-dimensional space. Assume that each dimension $j = 1, \ldots, D$ of $\{\mathbf{X}_i^j\}^D$ has been mean subtracted so that the data points are centered around the origin. Let $\{\mathbf{x}_i\}^d$, which is the linear projection of each data point onto the low dimensional space with dimension $d \ll D$:

$$\mathbf{x}_i = \mathbf{X}_i\mathbf{c}, \tag{7}$$

and we impose that the projection matrix is normalized, i.e. $\mathbf{c}^T\mathbf{c} = \mathbf{I}$. Because the data have been mean subtracted, the projected data is also centered around the origin, and therefore the variance of the projected data is just

$$\mathbf{x}^T\mathbf{x} = \mathbf{c}^T\mathbf{X}^T\mathbf{X}\mathbf{c}. \tag{8}$$

PCA identifies the d linear combinations of the feature components that capture the largest possible fraction of the variability of the dataset, which amounts to maximize $\mathbf{c}^T\mathbf{X}^T\mathbf{X}\mathbf{c}$. There are a few ways to solve this optimization problem. The classic approach is to compute the eigenvalues of the covariance matrix $\mathbf{C}$ of the D-dimensional data and set to the eigenvector $\mathbf{v}$ associated with the first d largest eigenvalues λ.

The eigenvectors (principal components) of our covariance matrix represent the vector directions of the new feature space and the eigen values represent the magnitudes of those vectors. Since we are looking at our covariance matrix the eigenvalues quantify the contributing variance of each vector.

2.4 A typical workflow of PCA

Step 0: Standard scaling.

For PCA to work properly, you have to subtract the mean from each of the data dimensions. The mean subtracted is the average across each dimension.

Scaling is also important: as one example, the spread of the data in one dimension is determined by the units used when taking the measurements! It is a common practice to scale the data so that they have unit variance, such that each feature will be weighted equally in our calculations.

Step 1: Calculate the covariance matrix.

After the data has been centered and scaled, the convariance matrix is simply $\mathbf{C} = \mathbf{X}^T\mathbf{X}$.

Step 2: Compute the eigenvalues and the eigenvectors.

If an eigenvector has a corresponding eigenvalue of high magnitude it means that our data has high variance along that vector in feature space. Thus, this vector holds a lot information about our data, since any movement along that vector causes large "variance". On the other hand vectors with small eigenvalues have low variance and thus our data does not vary greatly when moving along that vector. Since nothing changes when moving along that particular feature vector i.e changing the value of that feature vector does not greatly effect our data, then we can say that this feature is not very important and we can afford to remove it.

That's the whole essence of eigen values and vectors within PCA. Find the vectors that are the most important in representing our data and discard the rest. Computing the eigenvectors and values of our covariance matrix is an easy one-liner in Numpy. After that, we'll sort the eigenvectors in descending order based on their eigenvalues.

Step 3: Choosing principal components

At this point we have a list of eigen vectors sorted in order of "importance" to our dataset based on their eigen values. Now what we want to do is select the most important feature vectors that we need and discard the rest. We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.

The question then becomes, how do we choose where to truncate? This used to be one of those classic questions with an unsatisfying answer. Without a clean method of determining this, often the best one can do is to eyeball it.

Step 4: Visualize and validate. By forming a projection of the high dimensional data into low dimension (usually two or three), one can visualize the data.

It is often helpful to color the data points using certain properties associated with each sample.

3 Non-linear dimensionality reduction

PCA can be extended to several classes of non-linear dimensionality reduction techniques. Here we give a couple of examples for such extensions.

3.1 KPCA

In Lecture 2 we discussed the kernel trick, a mathematical technique that implicitly maps instances into a very high-dimensional space (called the feature space), enabling nonlinear classification and regression. It turns out that the same trick can be applied to PCA, making it possible to perform complex nonlinear projections for dimensionality reduction. This is called Kernel PCA (KPCA). It is often good at preserving clusters of instances after projection, or sometimes even unrolling datasets that lie close to a twisted manifold. As another advantage, KPCA avoids explicit computation of the features, just the similarities between each pair of data points.

There is a very simple isomorphism between PCA and KPCA: PCA takes the first a few eigenvectors from the convariance matrix , and KPCA considers the first a few eigenvectors from the kernel matrix $\mathbf{K}$. That is it! The nonlinearity of KPCA is incorporated through the definition of the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$.

3.2 Multidimensional scaling (MDS)

The task of metric multidimensional scaling (MDS) can be abstractly formulated as follows: given a $N \times N$ matrix $\mathbf{D}$ of pairwise distances between N points, find a low-dimensional embedding of data points $\{\mathbf{x}_i\}$ such that Euclidean distances between them approximate the given distances:

$$\|\mathbf{x}_i - \mathbf{x}_j\| \approx D_{ij}. \tag{9}$$

A good approximation here seeks to minimize the cost function called "stress":

$$\text{Stress} \sim \left\| \mathbf{D} - \|\mathbf{x}_i - \mathbf{x}_j\| \right\|^2. \tag{10}$$

In general, the solution is not given by any closed formula, and must be computed by a dedicated iterative algorithm.