CSE 475: Statistical Methods in AI **Monsoon 2018**

Lec 11: Linear Dimensionality Reduction

Lecturer: C. V. Jawahar *Date: Sep. 10, 2018*

In general, we start with a feature representation $\mathbf{x}$ corresponding to a physical phenomena or an object of interest. (In the case of supervised learning, we also have a corresponding y. i.e., $((\mathbf{x}_i, y_i))$ In practice, this $\mathbf{x}$ is not the result of very careful selection of measurements/features, These features are either what we could think of as relevant or what is feasible in practice. There could be redundancy and correlation within these features too. They may not be the best for our problem also. A problem of interest to us is to find a new feature representation $\mathbf{z}$, often lower in dimension, as

$$\mathbf{z} = \mathbf{U}\mathbf{x}. \tag{11.20}$$

If $\mathbf{x}$ is a d-dimensional vector and $\mathbf{z}$ is a k-dimensional vector (where $k < d$), $\mathbf{U}$ is a $k \times d$ matrix.

Two variants:

- The above linear transformation (matrix multiplication) leads to a set of "new" features that are "derived" out existing features. New features are linear combination of existing features. We will have a closer look at this today. This can be supervised or unsupervised. i.e., y may be available (supervised) or unavailable (unsupervised).
- Second direction to create a new lower dimensional representation by selecting only the useful/important features from the original set. This is a classical *subset selection* problem, which is *hard* to solve. (Similar to your familiar knapsack problem.) This is not in our scope. Such problems are attempted with greedy or backtracking algorithms.

Some people distinguish these two carefully as *feature extraction* and *feature selection*. We can also look at them as *dimensionality reduction*. Dimensionality reduction makes the downstream computations efficient. Some time storage/memory is also made efficient through the dimensionality reduction. If your original $\mathbf{x}$ is "raw-data" (say an image as such represented as a vector), then such techniques are treated as principled ways to define and extract features.

When we do the dimensionality reduction, we may be removing useful information (or noise). Due to this, there is a minor chance that we may reduce the accuracy of the down stream task (say classification). In general, dimensionality reduction schemes aim at no major reduction in accuracy (happy with some increase in accuracy), but in a lower dimension.

If "noise" (irrelevant information) is removed orsuppressed in the entire process, we may expect some increase in the accuracy. At the same time, if we loose some " relevant information", then we may loose the accuracy. But, alas, we do not know what is noise and what is signal. We would like the machine to figure out this from the examples/data. If both these problems are solved independendly (as is the case in the classical ML schemes), there is noway, we can tell the dimensionality reduction scheme what is the best to do.

The above equation is for linear dimensionality reduction. We also have many equivalent non-linear dimensionality reduction schemes. They are mostly not in our scope.

11.33 PCA

Principal Component Analysis (PCA) is one of the most popular technique for dimensionality reduction. It is unsupervised. It can be seen from two different view points:

- A dimensionality reduction that retains maximum variance along the new dimensions.
- A representation/compression from which one can reconstruct the original data with minimal error.

11.33.1 Minimizing Variance

Let $\mathbf{u}$ be a dimension on which we want to project the data $\mathbf{x}_i$ so that we obtain a new representation z_i that has maximum variance.

It is easy to see that the mean of the original representation gets projected to the mean of the new representation.

$$\bar{z} = \frac{1}{N}\sum_{i=1}^{N} z_i = \frac{1}{N}\sum_{i=1}^{N} \mathbf{u}^T\mathbf{x}_i = \frac{1}{N}\mathbf{u}^T\sum_{i=1}^{N}\mathbf{x}_i = \mathbf{u}^T\mu$$

We are interested in finding a $\mathbf{u}$ that maximizes the variance after the projection

$$\arg\max \frac{1}{N}\sum_{i=1}^{N}(z_i - \bar{z})^2 = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}^T\mathbf{x}_i - \mathbf{u}^T\mu)^2$$

$$\arg\max \frac{1}{N}\left(\mathbf{u}^T\sum_{i=1}^{N}[\mathbf{x}_i - \mu][\mathbf{x_i} - \mu]^T\right)\mathbf{u}$$

$$\arg\max \frac{1}{N}\mathbf{u}^T\Sigma\mathbf{u}$$

with the constraint of $||\mathbf{u}|| = 1$, the solution to this problem can be seen as the eigen vector corresponding to the largest eigen value of the covariance matrix Σ. (see the derivation somewhere else in the notes.) When the data is centered or mean is subtracted, one can see that $\Sigma = \mathbf{X}\mathbf{X}^T$. Where $\mathbf{X}$ is a $d \times N$ data matrix. (Note: may be we did use the notation in an earlier lecture for the transpose of this matrix.)

Best dimension to project so as to maximize the variance is the eigen vector corresponding to the largest eigen value. The second best will be the second largest one and so on.

11.33.2 Minimizing Reconstruction Loss

Let $\mathbf{u_1}, \ldots \mathbf{u_d}$ be d orthonormal vectors. We can represent the vectors $\mathbf{x}$ as $\sum_{i=1}^{d}\alpha_i\mathbf{u}_i$. Where the scalar α_i is $\mathbf{x}^T\mathbf{u_i}$. However, if we use smaller than d basis vectors, there could be some loss or reconstruction error. Let us consider the loss when we use only one $\mathbf{u}$. i.e.,

$$\mathbf{x} - \mathbf{u}\mathbf{u}^T\mathbf{x}$$

Sum of the reconstruction loss for all the N data samples is now:

$$\sum_{i=1}^{N}||\mathbf{x_i} - \mathbf{u}\mathbf{u}^T\mathbf{x}_i||^2$$

$$= \sum_{i=1}^{N}\left(\mathbf{x}_i^T\mathbf{x}_i + (\mathbf{u}\mathbf{u}^T\mathbf{x}_i)^T(\mathbf{u}\mathbf{u}^T\mathbf{x}_i) - 2\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i\right)$$

We would like to minimize this. First term is positive (non negative). It is independnet of $\mathbf{u}$. Therefore, we would like to minimize:

$$\sum_{i=1}^{N}(\mathbf{x_i}^T\mathbf{u}\mathbf{u}^T\mathbf{u}\mathbf{u}^T\mathbf{x_i} - 2\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i)$$

We also know that $\mathbf{u}^T\mathbf{u} = 1$.

$$= \sum_{i=1}^{N} -\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i = \sum_{i=1}^{N} -\mathbf{u}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{u} = -\mathbf{u}^T\Sigma\mathbf{u}$$

Minimizing the reconstruction error now becomes that of Maximizing

$$\mathbf{u}^T\Sigma\mathbf{u}$$

with our familiar constraint of $\mathbf{u}^T\mathbf{u} = 1$. This reduces the solution as the eigen vectors corresponding to the largest eigen values.

11.33.3 PCA: Algorithm

- Center the data by subtracting the mean μ
- Compute the covariance matrix $\Sigma = \frac{1}{N}\mathbf{X}\mathbf{X}^T = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T$
- Compute eigen values and eigen vectors of Σ.
- Take the k eigen vectors corresponding to the largest k eigen values. Keep the eigen vectors as the rows and create a matrix $\mathbf{U}$ of $k \times d$.
- Compute the reduced dimensional vectors as

$$\mathbf{z}_i = \mathbf{U}\mathbf{x}_i$$

11.33.3.1 How many eigen vectors?

In practice what should be the value of k? This is often decided by looking at how much information is lost in doing the dimensionality reduction. An estimate of this is

$$\frac{\sum_{i=k+1}^{d}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$$

Often k is picked such that the above ratio is less than 5% or 10%.

Q: Why this ratio is useful? Can you find an explanation?

11.34 Example: Eigen Face

A classical example/application of PCA is in face recognition and face representation. Let us assume that we are given N face images each of $\sqrt{d} \times \sqrt{d}$. We can visualize this as a d dimensional vector. Assume that our input is all the faces from a passport office. All the faces are approximately of the same size. They are all frontal. And also expression neutral.

- Let us assume that mean μ is subtracted from each of the face. If all of the inputs were faces, then the mean is also looking very similar to a face.
- Let the input be $\mathbf{x}_1 \ldots \mathbf{x_N}$ be the mean subtracted faces and $\mathbf{X}$ be the $d \times N$ matrix of all these faces. (Q: Practically which would be larger here? N or d?)

- We are interested in finding the eigen vectors of the matrix $\mathbf{XX}^T$. Say they are $\mathbf{u_1} \ldots \mathbf{u_k}$. Note that $k \leq \min(N, d)$.
- We then project each of the inputs $\mathbf{x}$ to these new eigen vectors and obtain a new feature.

$$z_i = \mathbf{u_i}^T \mathbf{x} \quad i = 1, \ldots, k$$

- The new representation $\mathbf{z}$ is obtained like this. You can also look at this as forming a $k \times d$ matrix $\mathbf{U}$ which has its rows eigen vectors.
- We obtain now a set of compact (k dimensional) representation $\mathbf{z_i}$ for each of the input face images $\mathbf{x_i}$, where $i = 1, \ldots, N$

11.34.1 Eigen Vectors of $\mathbf{XX^T}$ from $\mathbf{X^TX}$

It is worth to see whether N of d is large more closely. In many cases N is larger than d. The reverse is also possible. In the case of eigen faces, it is quite possible that the image size is say 100×100 (or $d = 10000$) and N is only 1000. Let us see how the eigen vectors of $\mathbf{X^TX}$ and $\mathbf{XX^T}$ are related? The first one is a $N \times N$ matrix while the second is a $d \times d$ matrix. (note that eigen vector computation is a costly numerical computation and a smaller matrix is clearly preferred. Q: What is the computational complexity? $O(?)$)

Let us assume that $\mathbf{u}_1, \ldots \mathbf{u}_m$ are the m eigen vectors of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are the eigen vectors of $\mathbf{XX}^T$. Note that $m \leq \min(d, N)$.

$$\mathbf{X}^T\mathbf{X}\mathbf{u} = \lambda\mathbf{u} \tag{11.21}$$

$$\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v} \tag{11.22}$$

Note that $\mathbf{u}$ is $N \times 1$ and $\mathbf{v}$ is $d \times 1$. Assume $d >> N$. We are interested in computing $\mathbf{v}$ from $\mathbf{u}$.

Let us premultiply both side by $\mathbf{X}^T$.

$$\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{X}\mathbf{v}$$

Let us replace $\mathbf{X}^T\mathbf{v}$ by $\mathbf{u}$.

$$\mathbf{X}\mathbf{X}^T\mathbf{u} = \lambda\mathbf{u}$$

The computational procedure can be now:

- Compute the m principal eigen vectors for $\mathbf{XX}^T$, say $\mathbf{v}_1, \ldots \mathbf{v}_m$.
- Compute the eigen vectors of our interest as

$$\mathbf{u}_i = \mathbf{X}^T\mathbf{v}_i$$

Q: Write the steps of the Eigen face based face representation learning.