中山大學
SUN YAT-SEN UNIVERSITY

Lecture 8. Dimensionality Reduction

Pattern Recognition and Computer Vision

Guanbin Li,
School of Data and Computer Science, Sun Yat-Sen University

扫码签到

What we will learn today?

- Singular value decomposition
- Principal Component Analysis (PCA)
- Image compression

Singular Value Decomposition (SVD)

- There are several computer algorithms that can "factorize" a matrix, representing it as the product of some other matrices

- The most useful of these is the Singular Value Decomposition.

- Represents any matrix A as a product of three matrices:

$$A = U\Sigma V^T$$

- Python command:
  - [U, S, V] = numpy.linalg.svd(A)

Singular Value Decomposition (SVD)

- Any $Y \in \mathbb{R}^{m\times n}$ can be decomposed into
$$Y = U\Sigma V^T$$
where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal.
- i.e., $U^T U = V^T V = I$, $U^T = U^{-1}$, $V^T = V^{-1}$
- $\Sigma \in \mathbb{R}^{m\times n}$ is not a square matrix, but looks like the form of diagonal matrix
$$\Sigma = \left[\begin{array}{ccc|c} \sigma_1 & 0 & 0 & \\ 0 & \ddots & 0 & O \\ 0 & 0 & \sigma_k & \\ \hline & O & & O \end{array}\right]$$
- The algorithm always sorts the entries $\sigma_1, \sigma_2, \ldots, \sigma_k$ from high to low

How to compute SVD?

1. Eigenvector definition

- Suppose we have a square matrix A. We can solve for vector x and scalar λ such that $Ax = \lambda x$
- In other words, find vectors where, if we transform them with A, the only effect is to scale them with no change in direction.
- These vectors are called eigenvectors, and the scaling factors λ are called eigenvalues

Singular Value Decomposition (SVD)

2. Finding SVD

- Eigenvectors are for square matrices, but SVD is for all matrices
- To do svd(A), computers can do this:
  - Take eigenvectors of AA^T (matrix is always square).
    - These eigenvectors are the columns of U.
  - Square root of eigenvalues are the singular values (the entries of Σ).
  - Take eigenvectors of A^TA(matrix is always square).
    - These eigenvectors are columns of V (or rows of V^T)
- SVD is fast, even for large matrices
- There are also other algorithms to compute SVD or part of the SVD
  - Python's np.linalg.svd() command has options to efficiently compute only what you need, if performance becomes an issue

SVD Applications

- We've discussed SVD in a very general way
- But SVD of an image matrix can be very useful for image compression.
- To understand this, we'll look at a less geometric interpretation of what SVD is doing

SVD Applications

- For example,

$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

- Look at how the multiplication works out, left to right:
- Column 1 of U gets scaled by the first value from Σ.

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

- The resulting vector gets scaled by row 1 of V^T to produce a contribution to A

SVD Applications

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \qquad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$+ \overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \qquad \overset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

$$= \qquad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Each product of (column i of U)·(value i from Σ)·(row i of V^T) produces a component of the final A.

SVD Applications

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}}$$

$$\overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

$$\overset{U\Sigma}{\begin{bmatrix} -3.67 & -.71 & 0 \\ -8.8 & .30 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad \overset{A_{partial}}{\begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}}$$

- We're building the columns of A as a linear combination of the columns of U
- Using all columns of U, we'll rebuild the original matrix perfectly
- But, in real-world data, often we can just use the first few columns of U and we'll get something close (e.g. the first $A_{partial}$, above)

SVD Applications

$$\underset{U\Sigma}{\begin{bmatrix} \boxed{-3.67} & -.71 & 0 \\ \boxed{-8.8} & .30 & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} \boxed{-.42} & \boxed{-.57} & \boxed{-.70} \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} \quad A_{partial} = \begin{bmatrix} 1.6 & 2.1 & 2.6 \\ 3.8 & 5.0 & 6.2 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

$$\underset{U\Sigma}{\begin{bmatrix} -3.67 & \boxed{-.71} & 0 \\ -8.8 & \boxed{.30} & 0 \end{bmatrix}} \times \underset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ \boxed{.81} & \boxed{.11} & \boxed{-.58} \\ .41 & -.82 & .41 \end{bmatrix}} \quad A_{partial} = \begin{bmatrix} -.6 & -.1 & .4 \\ .2 & 0 & -.2 \end{bmatrix}$$

- We can call those first few columns of U the ***Principal Components*** of the data
- They show the major patterns that can be added to produce the columns of the original matrix
- The rows of V^T show how the principal components are mixed to produce the columns of the matrix

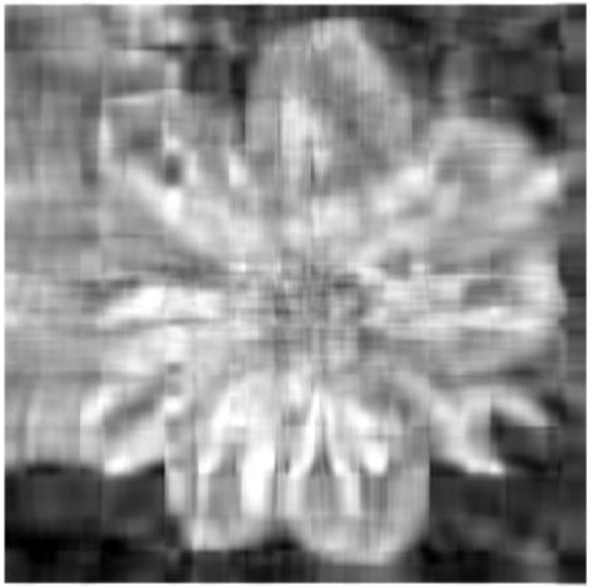
SVD Applications

$$\overset{U}{\begin{bmatrix} -.39 & -.92 \\ -.92 & .39 \end{bmatrix}} \times \overset{\Sigma}{\begin{bmatrix} 9.51 & 0 & 0 \\ 0 & .77 & 0 \end{bmatrix}} \times \overset{V^T}{\begin{bmatrix} -.42 & -.57 & -.70 \\ .81 & .11 & -.58 \\ .41 & -.82 & .41 \end{bmatrix}} = \overset{A}{\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}}$$

We can look at **Σ** to see that the first column has a large effect

while the second column has a much smaller effect in this example

SVD Applications



- For this image, using only the first 10 of 300 principal components produces a recognizable reconstruction
- So, SVD can be used for image compression

What we will learn today?

- Singular value decomposition
- Principal Component Analysis (PCA)
- Image compression

Covariance

- Variance and Covariance are a measure of the “spread” of a set of points around their center of mass (mean)
- Variance
  - $Var(x) = E[(x - E[x])^2] = E[x^2] - E[x]^2$
  - A measure of the deviation from the mean for points in one dimension
- Covariance
  - $Cov(x, y) = E[\big(x - E(x)\big)\big(y - E(y)\big)^T]$
  - A measure of how much each of the dimensions vary from the mean with respect to each other.
  - Covariance is measured between 2 dimensions to see if there is a relationship between the 2 dimensions e.g. number of hours studied & marks obtained.
  - The covariance between one dimension and itself is the variance

Covariance

$$Cov(x, y) = E[\left(x - E(x)\right)\left(y - E(y)\right)^T]$$

- So, if you had a 3-dimensional data set (x, y, z), then you could measure the covariance between the x and y dimensions, the y and z dimensions, and the x and z dimensions. Measuring the covariance between x and x , or y and y , or z and z would give you the variance of the x , y and z dimensions respectively

Covariance matrix

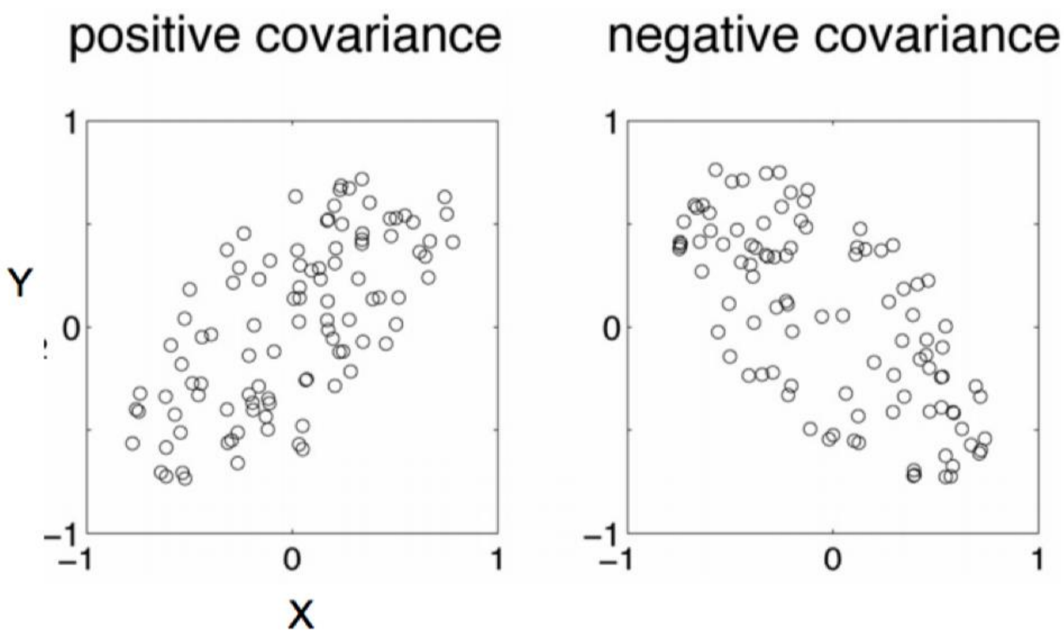
- Representing Covariance between dimensions as a matrix e.g. for 3 dimensions

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

Variances

- $Cov(x, y) = Cov(y, x)$, hence matrix is symmetrical about the diagonal
- N-dimensional data will result in NxN covariance matrix

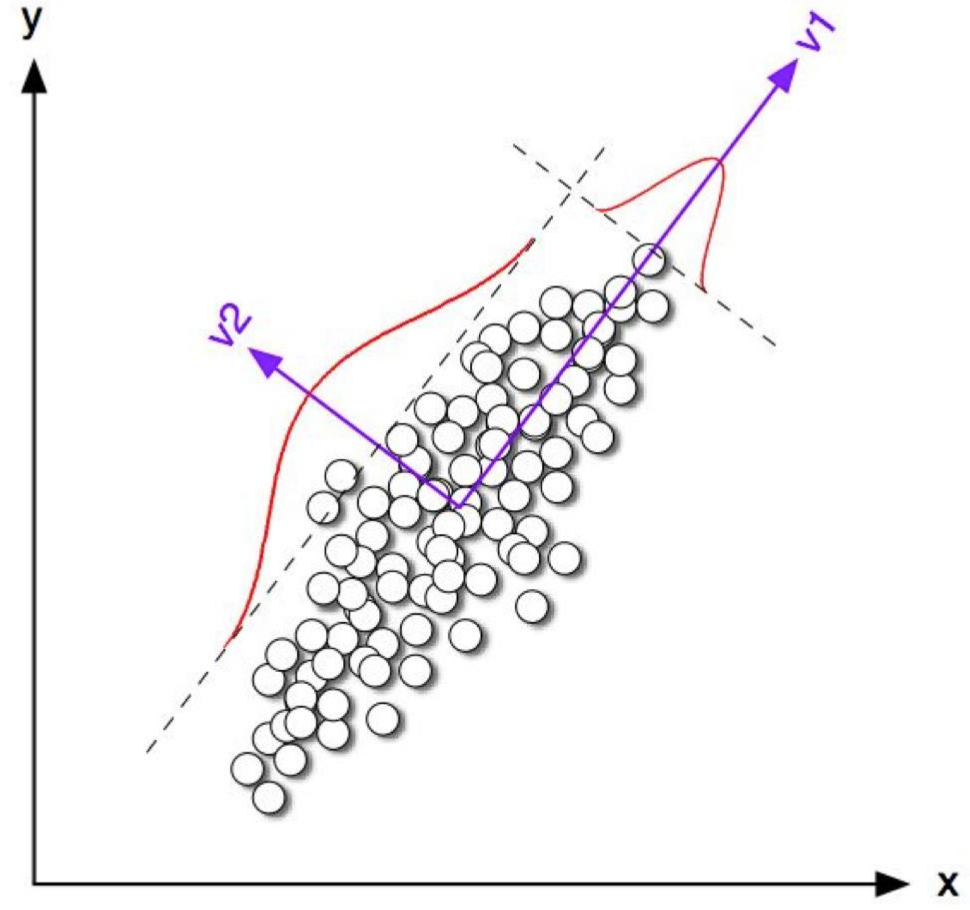
Covariance Interpretation



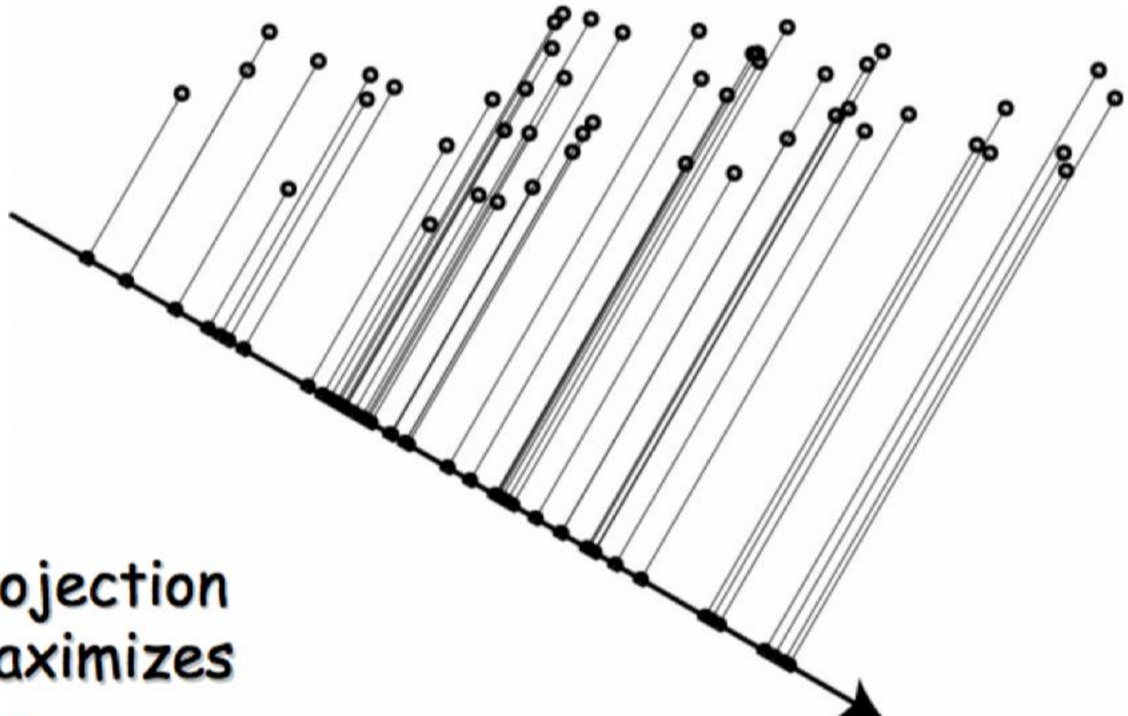
- The sign of covariance between two variables reveals the relation of them
- A positive value of covariance indicates both dimensions increase or decrease together e.g. as the number of hours studied increases, the marks in that subject increase.
- A negative value indicates while one increases the other decreases, or vice-versa e.g. active social life at university vs performance in CS dept.
- If covariance is zero: the two dimensions are independent of each other e.g.
- heights of students vs the marks obtained in a subject

Example

Covariance between the two axis is high. Can we reduce the number of dimensions to just 1?

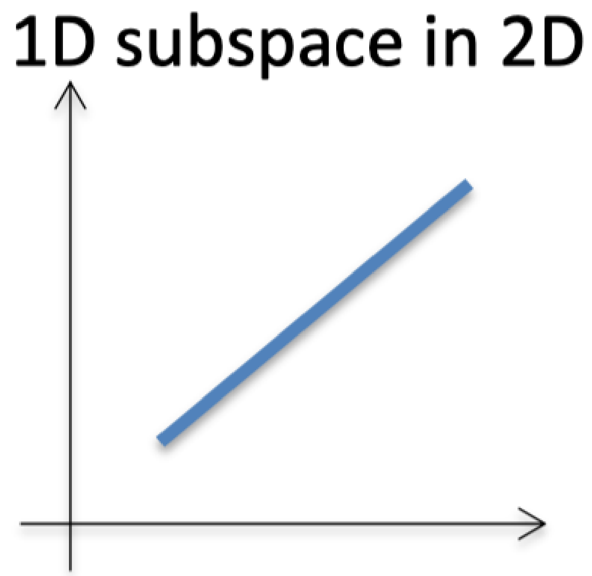


Geometric interpretation of PCA



find projection
that maximizes
variance

Geometric interpretation of PCA



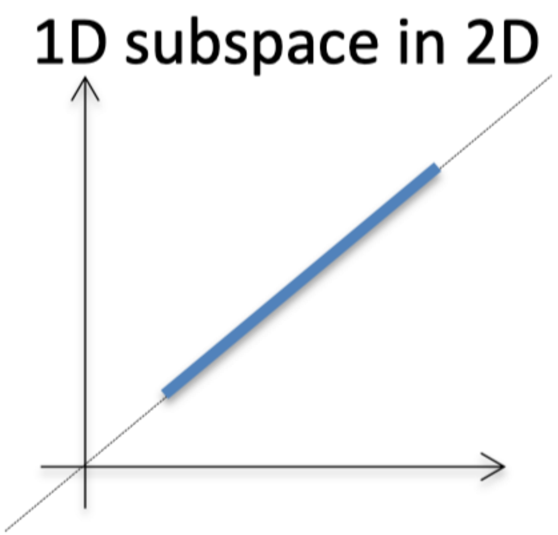
- Let's say we have a set of 2D data points x. But we see that all the points lie on a line in 2D.
- So, 2 dimensions are redundant to express the data. We can express all the points with just one dimension.

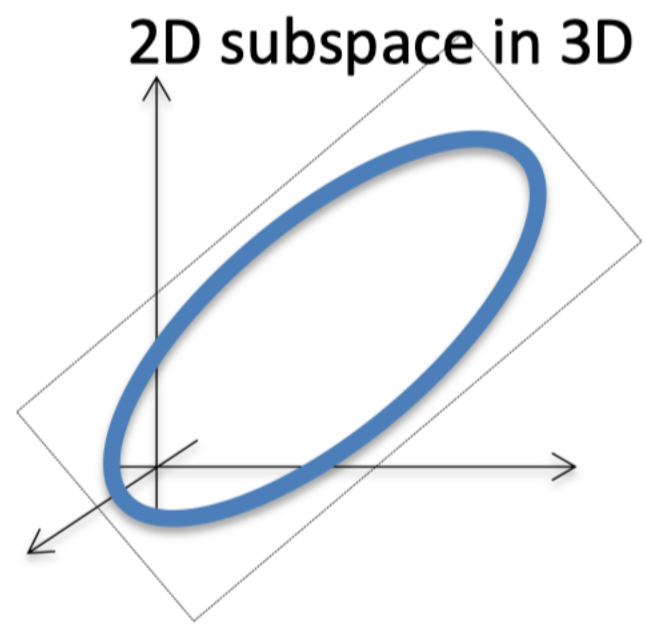
PCA: Principle Component Analysis

- Given a set of points, how do we know if they can be compressed like in the previous example?
- The answer is to look into the correlation between the points
- The tool for doing this is called PCA

PCA Formulation

- Basic idea:
  - If the data lives in a subspace, it is going to look very flat when viewed from the full space, e.g.





PCA Formulation

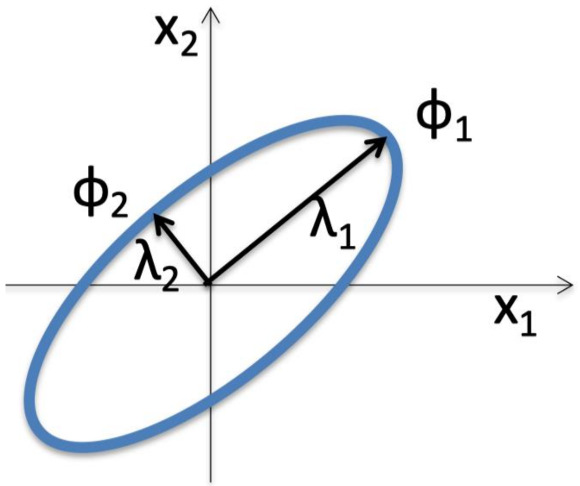
- Assume x is Gaussian with covariance Σ.

- Recall that a gaussian is defined with it's mean and variance:

$$x \sim N(\mu, \Sigma)$$

- Recall that μ and Σ of a gaussian are defined as:

$$\mu = E[x]$$
$$\Sigma = \mathrm{E}\left[(x-\mu)(x-\mu)^{\mathrm{T}}\right]$$

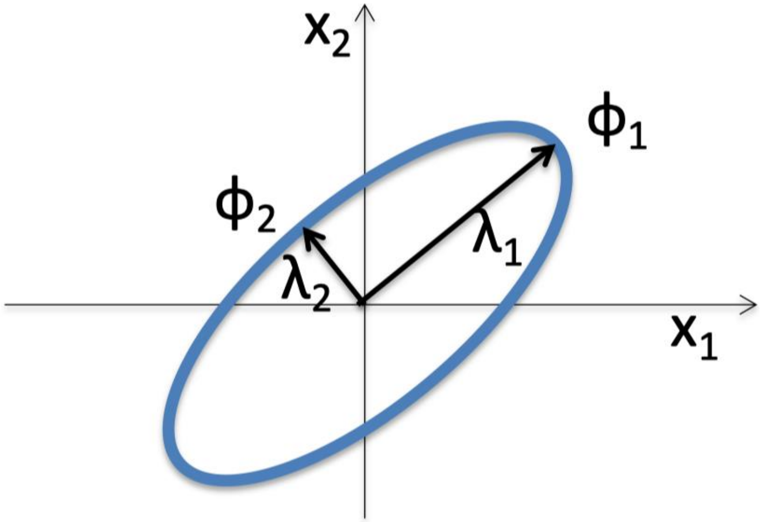


PCA Formulation

- If x is Gaussian with covariance Σ,
  - Principal components φ_i are the eigenvectors of Σ
  - Principal lengths λ_i are the eigenvalues of Σ

- by computing the eigenvalues we know the data is
  - Not flat if $\lambda_1 \approx \lambda_2$
  - Flat if $\lambda_1 \gg \lambda_2$

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{\frac{d}{2}} |\mathbf{\Sigma}|^{\frac{1}{2}}} \exp[-\frac{1}{2}(\mathbf{x}-\mu)^{T}\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)]$$



1. 现在我们有 m 个样本数据，每个样本有 n 个特征，那么设这些原始数据为 X， X 为 n 行 m 列的矩阵。
2. 想要找到一个基 P，使 $Y_{r\times m}=P_{r\times n}X_{n\times m}$，其中 $r<n$，达到降维的目的。

设 X 的协方差矩阵为 C， Y 的协方差矩阵为 D，且 $Y=PX$。

> 我们的目的变为：对原始数据 X 做PCA后，得到的 Y 的协方差矩阵 D 的各个方向方差最大，协方差为0。
> 那么 C 与 D 是什么关系呢?

$$
\begin{aligned}
D &= \frac{1}{m}YY^T \\
&= \frac{1}{m}(PX)(PX)^T \\
&= \frac{1}{m}PXX^TP^T
\end{aligned}
$$

$$
\begin{aligned}
&= PCP^T \\
&= P\begin{bmatrix} \frac{1}{m}\sum_{i=1}^m a_i^2 & \frac{1}{m}\sum_{i=1}^m a_ib_i \\ \frac{1}{m}\sum_{i=1}^m a_ib_i & \frac{1}{m}\sum_{i=1}^m b_i^2 \end{bmatrix}P^T
\end{aligned}
$$

找到能让原始协方差矩阵对角化的P！

PCA Algorithm (training)

- Given sample $D = \{x_1, x_2, \dots, x_n\}$, $x_i \in \mathbb{R}^d$
  - Compute sample mean
$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$
  - Compute sample variance
$$\Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$$
  - Compute eigenvalues and eigenvectors of Σ
$$\Sigma = \Phi\Lambda\Phi^{\mathrm{T}}, \Lambda = \mathrm{diag}\left(\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2\right), \Phi^{\mathrm{T}}\Phi = \mathrm{I}$$
  - Order eigenvalues $\sigma_1^2 > \sigma_2^2 > \cdots > \sigma_n^2$
  - If, for a certain k, $\sigma_k \ll \sigma_1$. We can eliminate the eigenvalues and eigenvectors above k.

PCA Algorithm (testing)

- Given the first k eigenvectors as principle components $\{\phi_i \mid i \in 1,2,\dots,k\}$ and a test sample $T = \{t_1, t_2, \dots, t_n\}, t_i \in \mathbb{R}^d$
  - Subtract mean to each point
  $$t_i' = t_i - \mu$$
  - Project onto eigenvector space $y_i = At_i'$, where
  $$A = \begin{bmatrix} \phi_1^T \\ \vdots \\ \phi_k^T \end{bmatrix}$$
  - $T' = \{y_1, y_2, \dots, y_n\}$ is the result of PCA.

PCA by SVD

- An alternative manner to compute the principal components, based on singular value decomposition
- Quick reminder: SVD
  - Any real $n \times m$ matrix A can be decomposed as

$$A = U\Sigma V^T$$

where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal,

$\Sigma \in \mathbb{R}^{m\times n}$ is not a square matrix, but looks like the form of diagonal matrix

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & \\ 0 & \ddots & 0 & O \\ 0 & 0 & \sigma_k & \\ & O & & O \end{bmatrix}$$

PCA by SVD

- To relate this to PCA, we consider the data matrix

$$X = [x_1, x_2, \dots, x_n] \in \mathbb{R}^{d \times n}$$

- The sample mean is

$$\mu = \frac{1}{n} \sum_i x_i = \frac{1}{n} [x_1, x_2, \dots, x_n] \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$$

- Center the data by subtracting the mean to each column of X. The centered data matrix is

$$X_c = [x_1, x_2, \dots, x_n] - [\mu, \mu, \dots, \mu] = [x_1^c, x_2^c, \dots, x_n^c]$$

PCA by SVD

- The sample covariance matrix is

$$\Sigma = \frac{1}{n}\sum_i x_i^c (x_i^c)^T$$

where x_i^c is the i column of X_c

- This can be written as

$$\Sigma = \frac{1}{n}[x_1^c, x_2^c, \ldots, x_n^c]\begin{bmatrix}(x_1^c)^T \\ \vdots \\ (x_n^c)^T\end{bmatrix} = \frac{1}{n} X_c X_c^T$$

PCA by SVD

- $X_c^T = \begin{bmatrix} (x_1^c)^T \\ \vdots \\ (x_n^c)^T \end{bmatrix} \in \mathbb{R}^{n\times d}$
- We can decompose X_c^T by SVD

$$X_c^T = U\Lambda V^T,$$
$$U^T U = V^T V = I,$$
$$\Lambda = \left[\begin{array}{cccc|} \sigma_1 & 0 & 0 & \\ 0 & \ddots & 0 & O \\ 0 & 0 & \sigma_k & \\ \hline & O & & O \end{array}\right]$$

- Hence

$$\Sigma = \frac{1}{n} X_c X_c^T = \frac{1}{n} U\Lambda V^T V \Lambda U^T = \frac{1}{n} U\Lambda^2 U^T$$

PCA by SVD

$$\Sigma = \frac{1}{n} U \Lambda^2 U^T$$

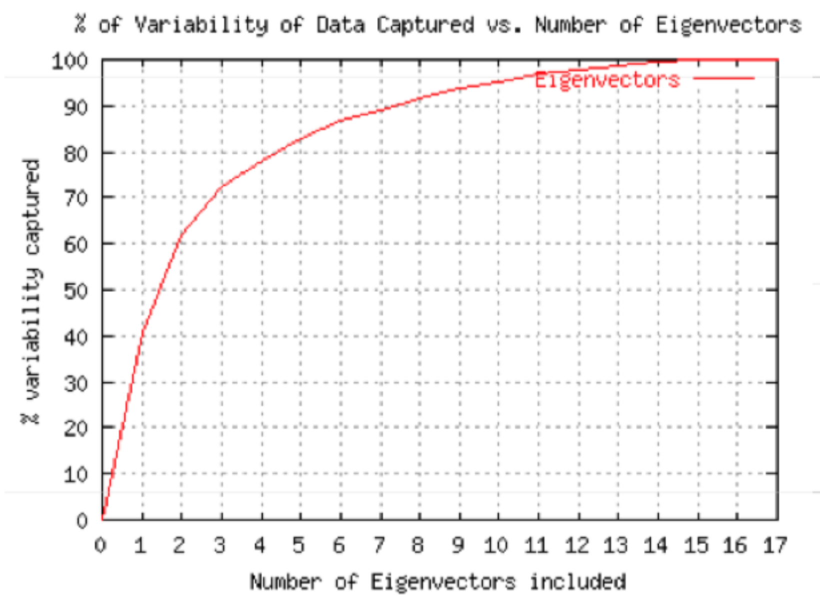
- Note that U is $(d \times d)$ and orthonormal, and Λ^2 is diagonal. This is just the eigenvalue decomposition of Σ
- It follows that
  - The eigenvectors of Σ are the columns of U
  - The eigenvalues of Σ are $\lambda_i = \frac{1}{n}\sigma_i^2$
- This gives an alternative algorithm for PCA

PCA by SVD

- In summary, computation of PCA by SVD
- Given X with one example per column
  - Create the centered data matrix
    $X_c = [x_1, x_2, \dots, x_n] - [\mu, \mu, \dots, \mu] = [x_1^c, x_2^c, \dots, x_n^c]$
  - Compute its SVD
    $$X_c^T = U\Lambda V^T$$
  - Principal components are columns of U, eigenvalues are
    $$\lambda_i = \frac{1}{n}\sigma_i^2$$

Rule of thumb for finding the number of PCA components

- A natural measure is to pick the eigenvectors that explain p% of the data variability
  - Can be done by plotting the ratio r_k as a function of k



$$r_k = \sum_{i=1}^{k} \lambda_i^2 \Big/ \sum_{i=1}^{n} \lambda_i^2$$

  - E.g. we need 3 eigenvectors to cover 70% of the variability of this dataset

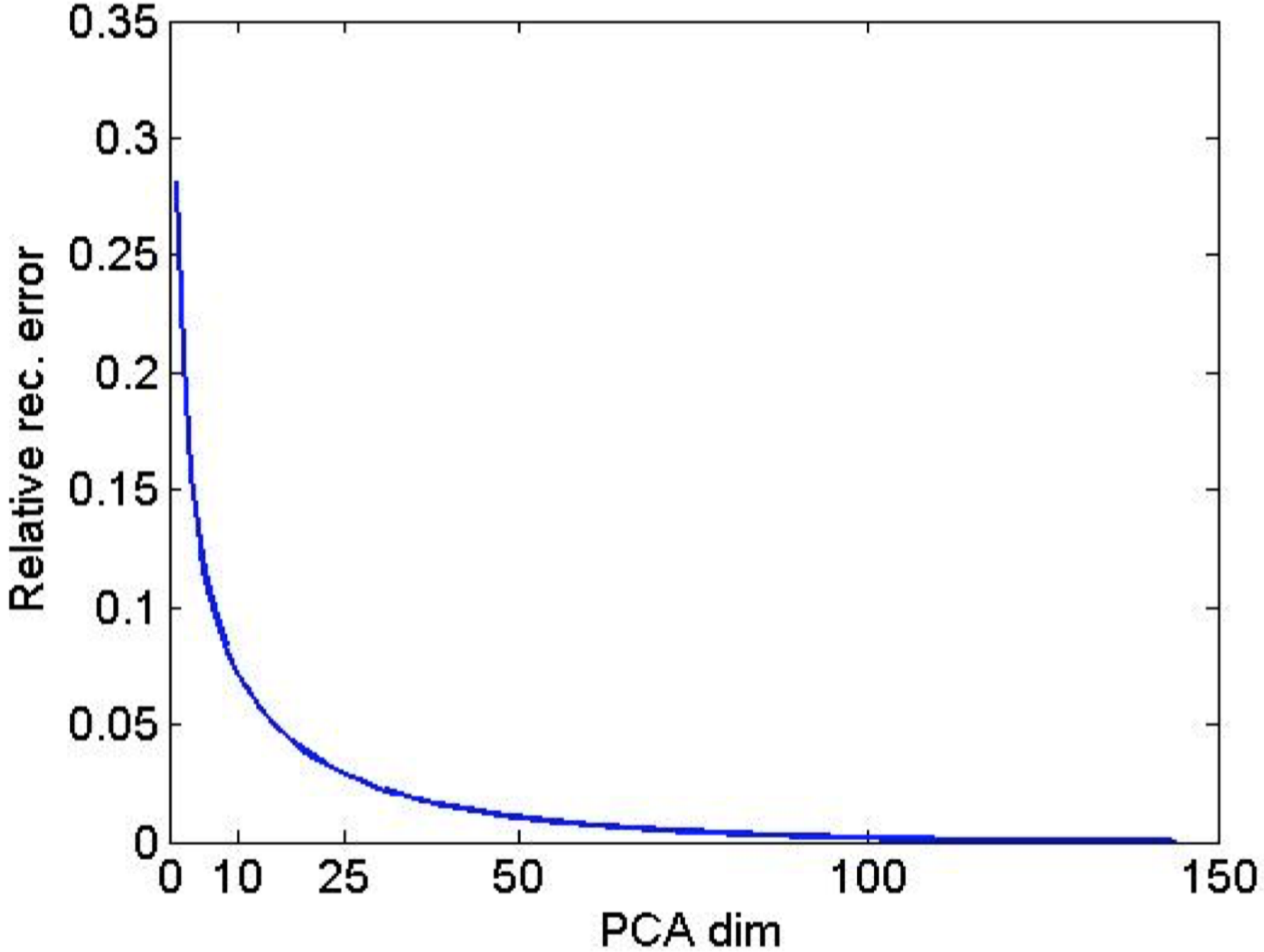
What we will learn today?

- Singular value decomposition
- Principal Component Analysis (PCA)
- Image compression

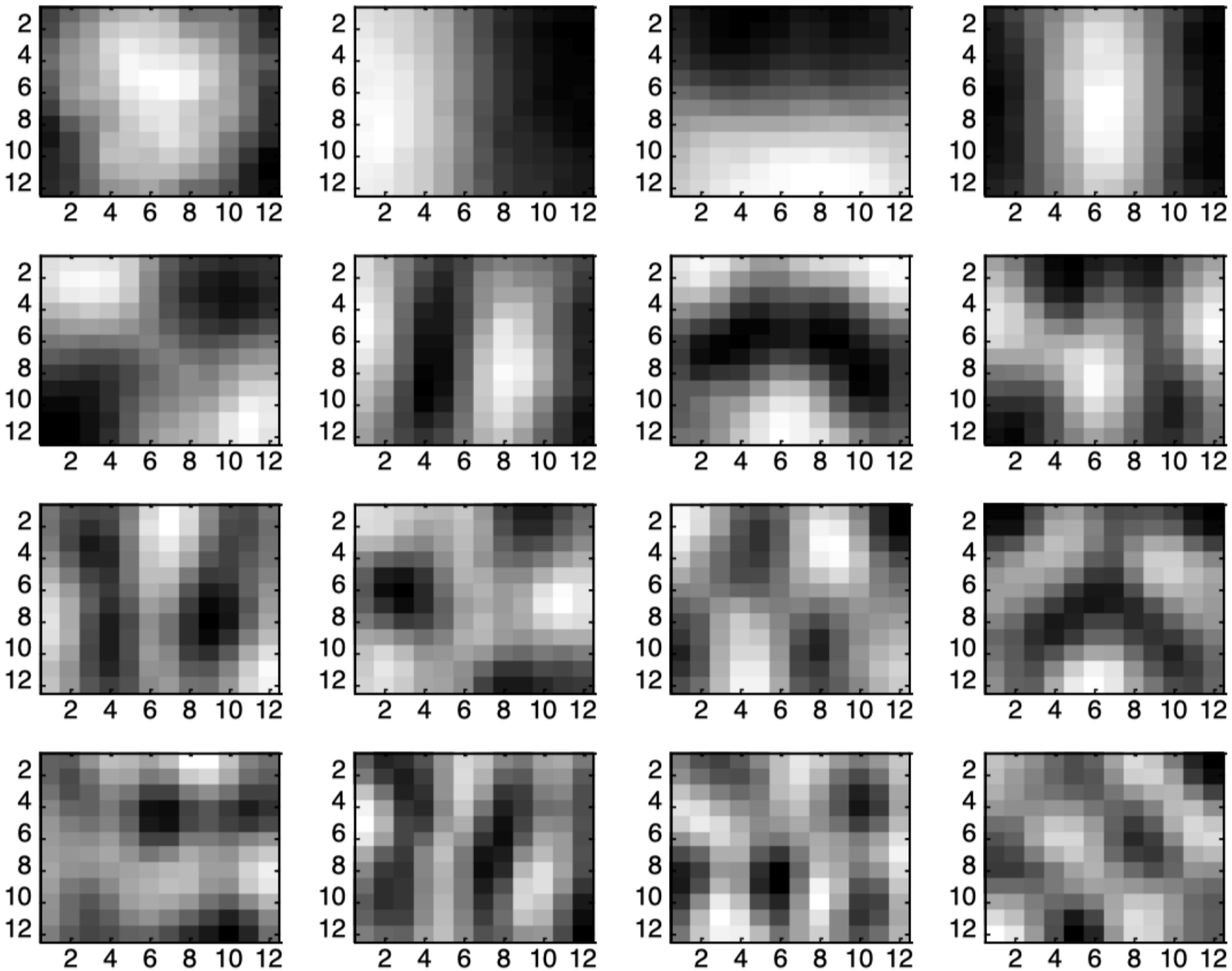
Original Image

- Divide the original 372x492 image into patches:
  - Each patch is an instance that contains 12x12 pixels on a grid
- View each as a 144-D vector

L_2 error and PCA dim



16 most important eigenvectors



PCA compression


144D


60D


16D


6D

What we have learned today?

- We have learned SVD for dimension reduction.

- We introduce PCA algorithm to obtain principle components of data. Combining with SVD, we can employ SVD to eigen-decompose covariance matrix efficiently.

- Besides, we introduce the application of image compression using PCA.

PCA Algorithm

总结一下PCA的算法步骤：

设有 m 条 n 维数据。

1）将原始数据按列组成 n 行 m 列矩阵X

2）将 X 的每一行（代表一个特征）进行零均值化，即减去这一行的均值

3）求出协方差矩阵 $C = \frac{1}{m} X X^{\mathsf{T}}$

4）求出协方差矩阵 C 的特征值及对应的特征向量

5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前 k 行组成矩阵 P

6）$Y = PX$ 即为降维到 k 维后的数据