Machine Learning

CSE 6363 (Fall 2019)

Lecture 4 SVD and PCA

Dajiang zhu, Ph.D.

Department of Computer Science and Engineering

Orthogonal Matrix

- Suppose **A** is a square matrix. **A** is called orthogonal matrix if

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{I}$$

where **I** is an identity matrix, $\mathbf{A}^{\mathrm{T}}$ is the transpose of **A**.

- For an orthogonal matrix, we have

$$\mathbf{A}^{-1} = \mathbf{A}^{\mathrm{T}}$$

Ref: Mingyue Ding

Eigenvalue & Eigenvector

- Suppose **A** is a square matrix. If having a number, λ, and a non-zero vector, **X** , satisfy

$$\boxed{\mathbf{AX} = \lambda \mathbf{X}}$$

- We called λ the eigenvalue of **A**, and **X** is the eigenvector of **A**
- If we know the eigenvalues of **A**, the eigenvectors can be determined by substituting the eigenvalues into above equation.

Ref: Mingyue Ding

Calculation of Eigenvalues

- We can determine the eigenvalues of **A** by solving the following equation:

$$\boxed{\left|\mathbf{A} - \lambda \mathbf{I}\right| = 0}$$

where **I** is an identity matrix

What is SVD?

- Find something important!

- Decompose into “concepts” and tell us the order of their importance!

What is SVD?

Any $m \times n$ matrix $\mathbf{A}$ with rank of r, can be decomposed into

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T$$

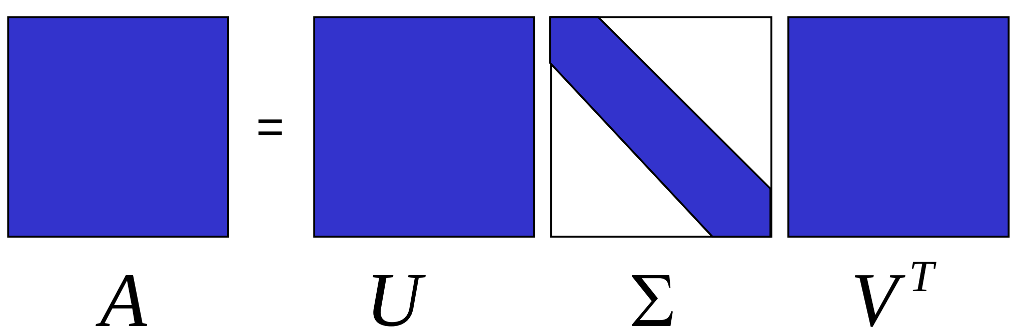
where **U and V** are orthogonal matrices and $\mathbf{D}$ is a diagonal matrix containing singular values, $\{\mu_i, i = 1, 2, \cdots, r\}$. This factored matrix representation is known as the SVD.

Ref: Mingyue Ding

SVD More Formally

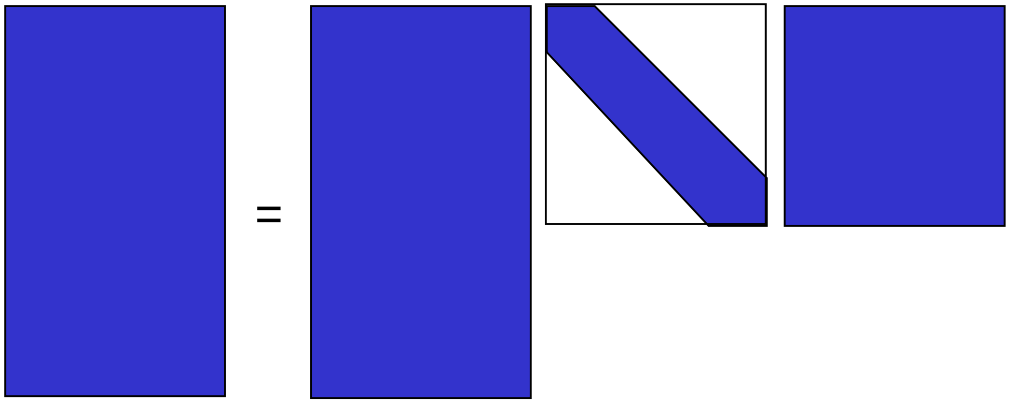
- The diagonal values of Σ ($\mu_1, \ldots, \mu_n$) are called the singular values. It is accustomed to sort them: $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$
- The columns of U ($\mathbf{u}_1, \ldots, \mathbf{u}_n$) are called the left singular vectors.
- The columns of V ($\mathbf{v}_1, \ldots, \mathbf{v}_n$) are called the right singular vectors.

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



Reduced SVD

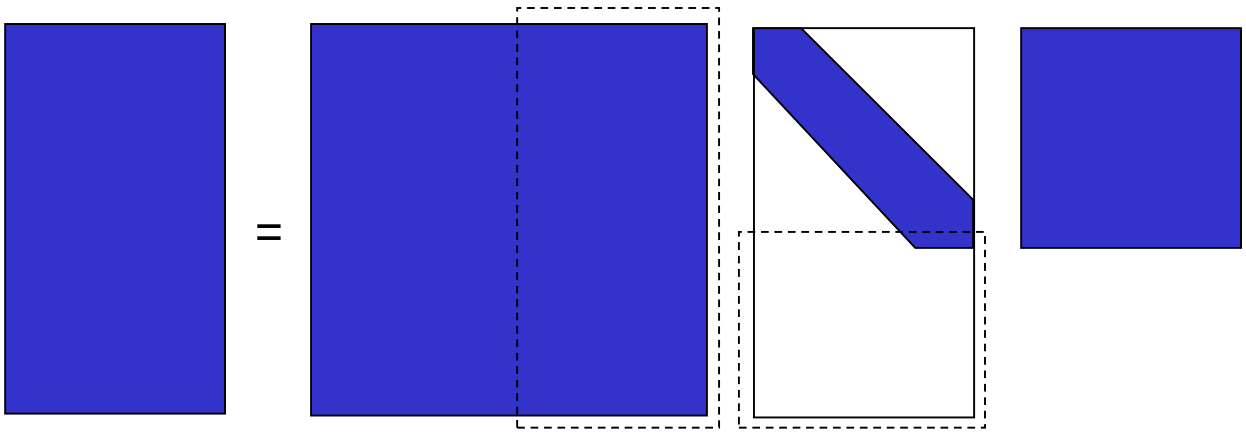
- For rectangular matrices, we have two forms of SVD. The reduced SVD looks like this:
  - The columns of U are orthonormal
  - Cheaper form for computation and storage



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

Full SVD

- We can complete U to a full orthogonal matrix and pad Σ by zeros accordingly



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

How to do SVD

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

How about and AA^T and $A^T A$?

How to do SVD

A simple demonstraton of SVD:

$$\begin{pmatrix} 5 & 5 \\ -1 & 7 \end{pmatrix}$$

Singular Values

- Suppose **A** is a *mxn* matrix and its rank is r ($r \le n$). We can calculate the non-zero eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$, e.g.,

$$\lambda_1 \ge \lambda_2 \cdots \ge \lambda_r$$

- We call $\mu_i = \sqrt{\lambda_i}\ (i = 1, 2, \cdots, r)$ as the singular values of **A**

Ref: Mingyue Ding

Matrix Inverse and Solving Linear Systems

- Matrix inverse: $A = U\Sigma V^T$

$$A^{-1} = \left(U\Sigma V^T\right)^{-1} = \left(V^T\right)^{-1}\Sigma^{-1}U^{-1} =$$

$$= V\begin{bmatrix} \frac{1}{\sigma_1} & & \\ & \ddots & \\ & & \frac{1}{\sigma_n} \end{bmatrix}U^T$$

- So, to solve $A\mathbf{x} = \mathbf{b}$

$$\mathbf{x} = V\Sigma^{-1}U^T\mathbf{b}$$

Application: Image Compression

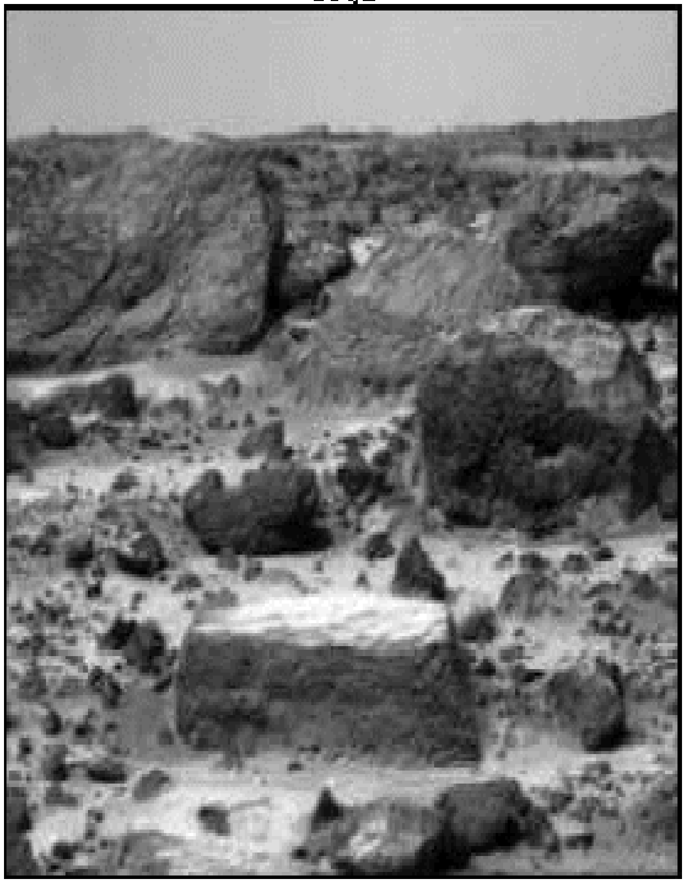
- Uncompressed m by n pixel image: $m \times n$ numbers
- Rank q approximation of image:
  - q singular values
  - The first q columns of $\boldsymbol{U}$ (m-vectors)
  - The first q columns of $\boldsymbol{V}$ (n-vectors)
  - Total: $q \times (m + n + 1)$ numbers

Ref: Bernick

Example: Yogi (Uncompressed)

- Source: [Will]
- Yogi: Rock photographed by Sojourner Mars mission.
- 256 $\times$ 264 grayscale bitmap $\rightarrow$ 256 $\times$ 264 matrix $\boldsymbol{M}$
- Pixel values $\in [0,1]$
- ~ 67584 numbers

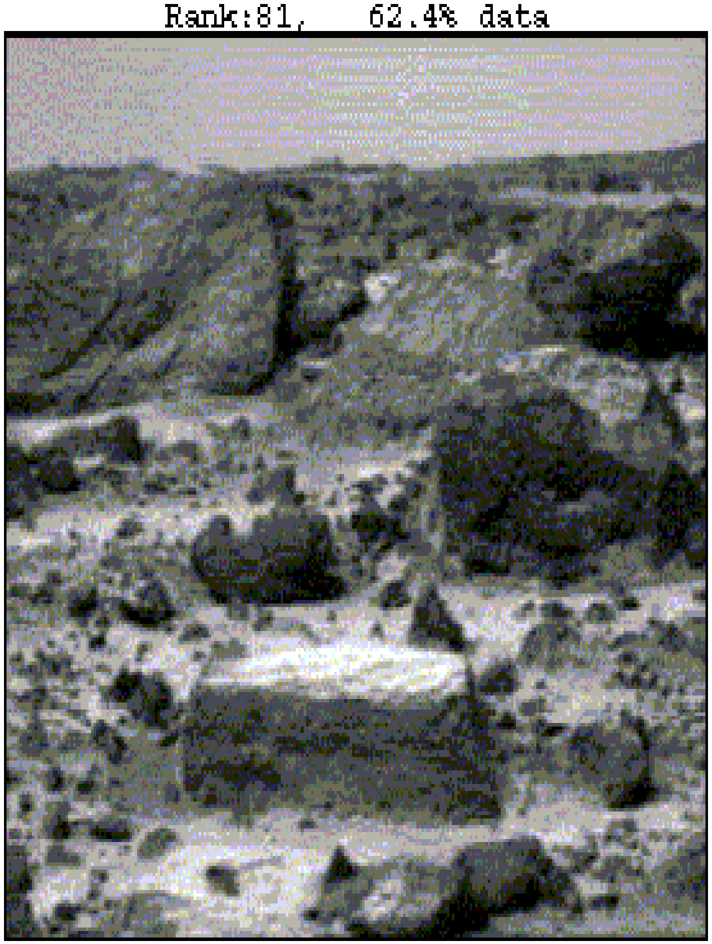
Yogi



Ref: Bernick

Example: Yogi (Compressed)

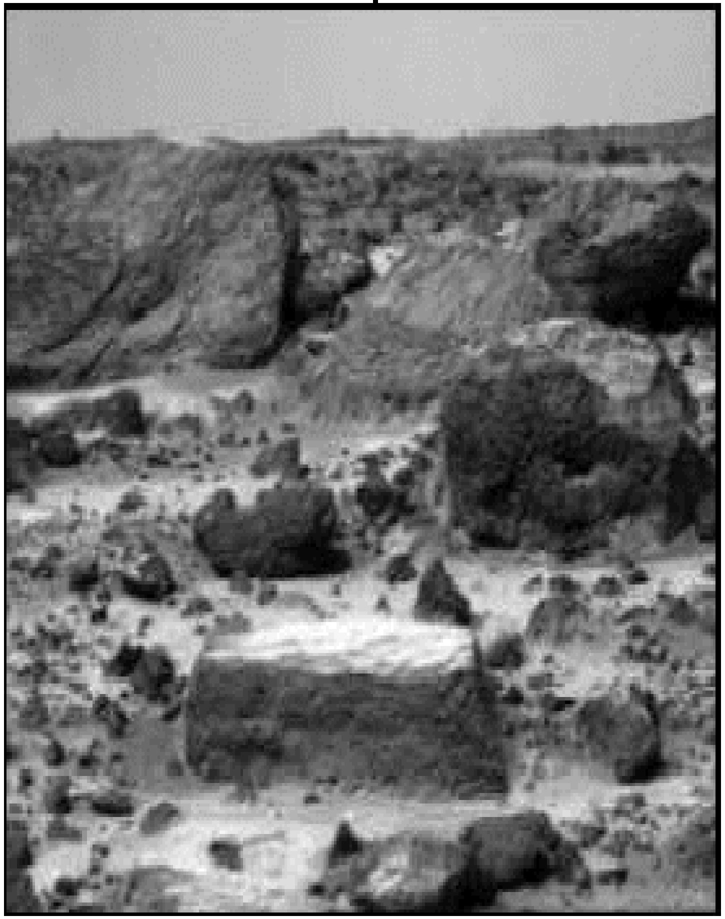
- $\boldsymbol{M}$ has 256 singular values
- Rank 81 approximation of $\boldsymbol{M}$:
- $81 \times (256 + 264 + 1) =$ ~ 42201 numbers



Ref: Bernick

Example: Yogi (Both)

Yogi



Rank:81, 62.4% data

Ref: Bernick

Application: Noise Filtering

- Data compression: Image degraded to reduce size
- Noise Filtering: Lower-rank approximation used to improve data.
  - Noise effects primarily manifest in terms corresponding to smaller singular values.
  - Setting these singular values to zero removes noise effects.

Ref: Bernick

Principal Components Analysis (PCA)

Question:

- Is there another basis, which is a linear combination of the original basis, that best re-expresses our data set?

Following questions:

1. What is the "best way" to re-express our data?
2. What is a good choice of basis P for P X = Y.

An Vision Application: Facial Recognition

- Want to identify specific person, based on facial image
- Robust to …
  - Facial hair, glasses, …
  - Different lighting

  ⇒ Can't just use given 256 x 256 pixels
- Need another option!

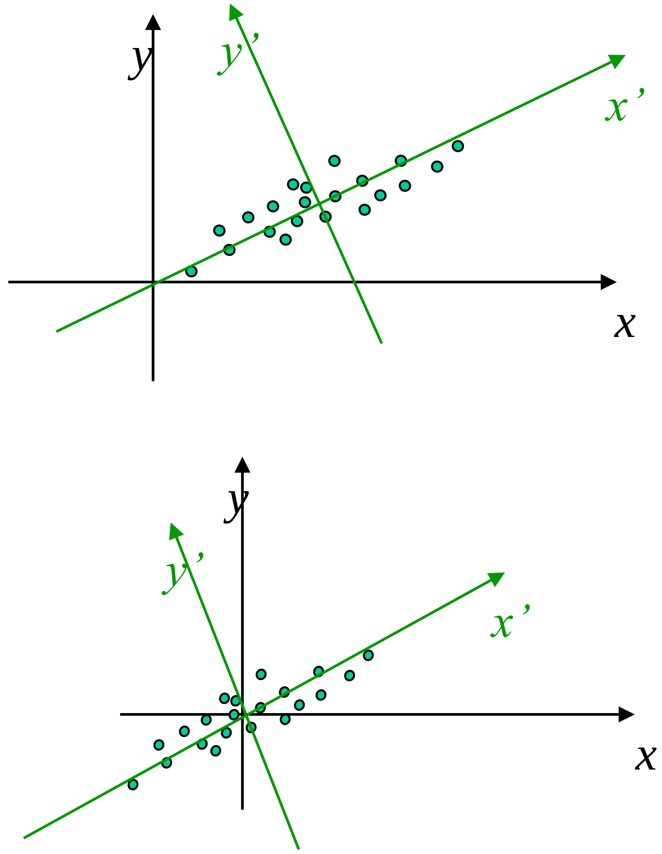
Why Do We Care

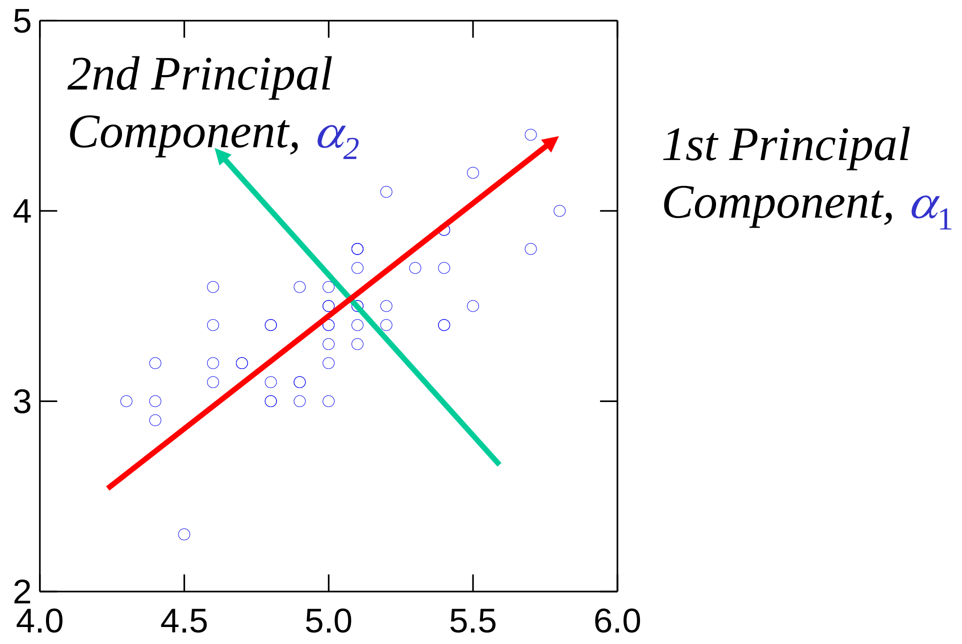
- Lower dimensional representations permit
  - Compression
  - Noise filtering
- As preprocessing for classification:
  - Reduces feature space dimension
    - Simpler Classifiers
    - Possibly better generalization
  - May facilitate simple (nearest neighbor) methods

Principal Components Analysis (PCA)

- Idea:
  - Given data points in d-dimensional space, project into *lower dimensional* space while *preserving as much information* as possible
    - Eg, find best planar approximation to 3D data
    - Eg, find best 12-D approximation to 10^4-D data
  - In particular, choose projection that minimizes *squared error* in reconstructing original data

Principal Components Analysis





Projection

- Orthonormal basis $\rightarrow$ trivial projection
- Given basis $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$
  can project any d-dim $\mathbf{x}$ to k values
  - $\alpha_1 = \mathbf{u}_1^T \mathbf{x} \quad \alpha_2 = \mathbf{u}_2^T \mathbf{x} \; \ldots \; \alpha_k = \mathbf{u}_k^T \mathbf{x}$
  - $\alpha = \mathbf{U}^T \mathbf{x}$
  - $\mathbf{x} \approx \Sigma_i \, \alpha_i \mathbf{u}_i = \Sigma_i \, (\mathbf{u}_i^T \mathbf{x}) \, \mathbf{u}_i$ ["=" if all d values]

- We will use "centered" vectors:
  $\mathbf{x}' = \mathbf{x} - \underline{\mathbf{x}}$ where $\underline{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^n$

$$\boxed{\alpha_i = \mathbf{u}_i^T (\mathbf{x} - \underline{\mathbf{x}})}$$

Minimize Reconstruction Error

- Assume data is set of N d-dimensional vectors, $\mathbf{x}^n = \langle x_1^n \ldots x_d^n \rangle$
- Represent each in terms of any d orthogonal basis vectors

$$\mathbf{x}^n = \sum_{i=1}^{d} z_i^n \mathbf{u}_i; \quad \mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$$

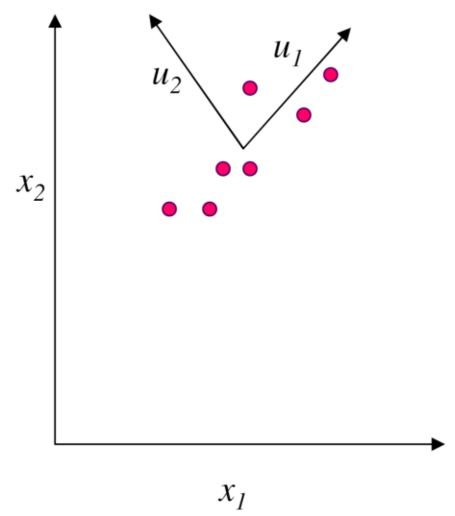
PCA: given k<d. Find { $\mathbf{u}_1$, …, $\mathbf{u}_k$ }

that minimizes $$E_k = \sum_{n=1}^{N} \left\| x^n - \hat{x}_k^n \right\|_2^2$$

where $$\hat{\mathbf{x}}_k^n = \underline{\mathbf{x}} + \sum_{i=1}^{k} \alpha_i^n \mathbf{u}_i$$

Mean

$$\underline{\mathbf{x}} = \frac{1}{N} \sum_{n=1}^{N} \mathbf{x}^n$$



PCA

- Note $$\hat{\mathbf{x}}^n_d = \underline{\mathbf{x}} + \sum_{i=1}^{d} \alpha_i^n \mathbf{u}_i \equiv \mathbf{x}^n$$
- So… $$\mathbf{x}^n - \hat{\mathbf{x}}^n_k = \sum_{i=k+1}^{d} \alpha_i^n \mathbf{u}_i = \sum_{i=k+1}^{d} \left((\mathbf{x}^n - \underline{\mathbf{x}})^T \mathbf{u}_i \right) \mathbf{u}_i$$
- So… $$E_k = \sum_{n=1}^{N} \left\| \sum_{i=k+1}^{d} \left((\mathbf{x}^n - \underline{\mathbf{x}})^T \mathbf{u}_i \right) \mathbf{u}_i \right\|^2 = \sum_{n=1}^{N} \sum_{i=k+1}^{d} \left[(\mathbf{x}^n - \underline{\mathbf{x}})^T \mathbf{u}_i \right]^2$$

$$= \sum_{i=k+1}^{d} \sum_{n=1}^{N} \left[\mathbf{u}_i^{\mathrm{T}} (\mathbf{x}^n - \underline{\mathbf{x}}) \right] \left[(\mathbf{x}^n - \underline{\mathbf{x}})^T \mathbf{u}_i \right]$$

$$= \sum_{i=k+1}^{d} \mathbf{u}_i^{\mathrm{T}} \, \Sigma \, \mathbf{u}_i$$

Covariance matrix:

$$\boxed{\Sigma = \sum_n (\mathrm{x}^n - \bar{\mathrm{x}})(\mathrm{x}^n - \bar{\mathrm{x}})^T}$$

PCA: given k<d. Find { $\mathbf{u}_1$, …. $\mathbf{u}_k$ }

that minimizes $$E_k = \sum_{n=1}^{N} \left\| x^n - \hat{x}_k^n \right\|_2^2$$

where $$\hat{\mathbf{x}}^n_k = \underline{\mathbf{x}} + \sum_{i=1}^{k} \alpha_i^n \mathbf{u}_i$$

Justifying Use of Eigenvectors

- Goal
  - minimize: $\mathbf{u}^T \Sigma \mathbf{u}$
  - subject to: $\mathbf{u}^T \mathbf{u} = 1$
- Use Lagrange Multipliers… minimize:

$$f(\mathbf{u}) = \mathbf{u}^T \Sigma \mathbf{u} - \lambda[\mathbf{u}^T \mathbf{u} - 1]$$

- Set derivative to 0: $\boxed{\Sigma \mathbf{u} - \lambda \mathbf{u} = 0}$
- Def'n of eigenvalue λ, eigenvector $\mathbf{u}$!
- If multiple vectors $\mathbf{u}_i$:
  - Minimize sum of independent terms…
  - Each is eigen value/vector

PCA

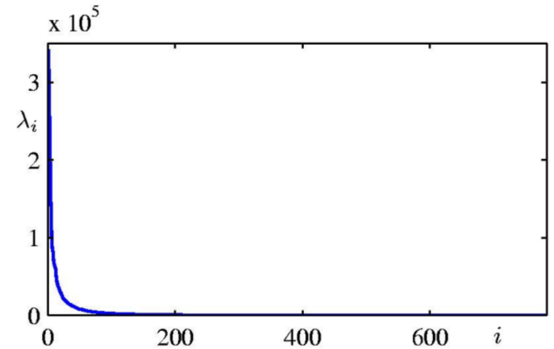
Minimize $E_k = \sum_{i=k+1}^{d} \mathbf{u}_i^T \Sigma \mathbf{u}_i$

$$\rightarrow \quad \Sigma \mathbf{u}_i = \lambda_i \mathbf{u}_i$$

Eigenvalue Eigenvector

$$\Rightarrow E_k = \sum_{i=k+1}^{d} \mathbf{u}_i^T \Sigma \mathbf{u}_i = \sum_{i=k+1}^{d} \mathbf{u}_i^T \lambda_i \mathbf{u}_i$$

$$= \sum_{i=k+1}^{d} \lambda_i \mathbf{u}_i^T \mathbf{u}_i = \sum_{i=k+1}^{d} \lambda_i$$



(a)

So... to minimize E_k, take SMALLEST eigenvalues { λ_i }

PCA Algorithm

PCA algorithm(X, *k*): top *k* eigenvalues/eigenvectors

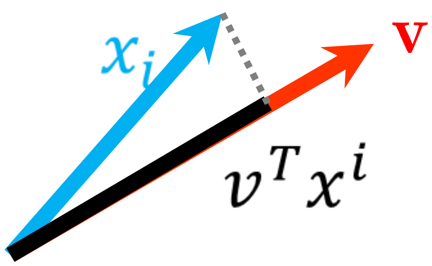
% X = d × N data matrix,
% … each data point x^n = column vector

- $\underline{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n$
- A ← subtract mean $\underline{x}$ from each column vector x^n in X
- Σ ← A A^T … covariance matrix of A
- $\{ \lambda_i, \mathbf{u}_i \}_{i=1..d}$ = eigenvectors/eigenvalues of Σ
  … $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$
- Return $\{ \lambda_i, \mathbf{u}_i \}_{i=1..k}$
  % top *k* principle components

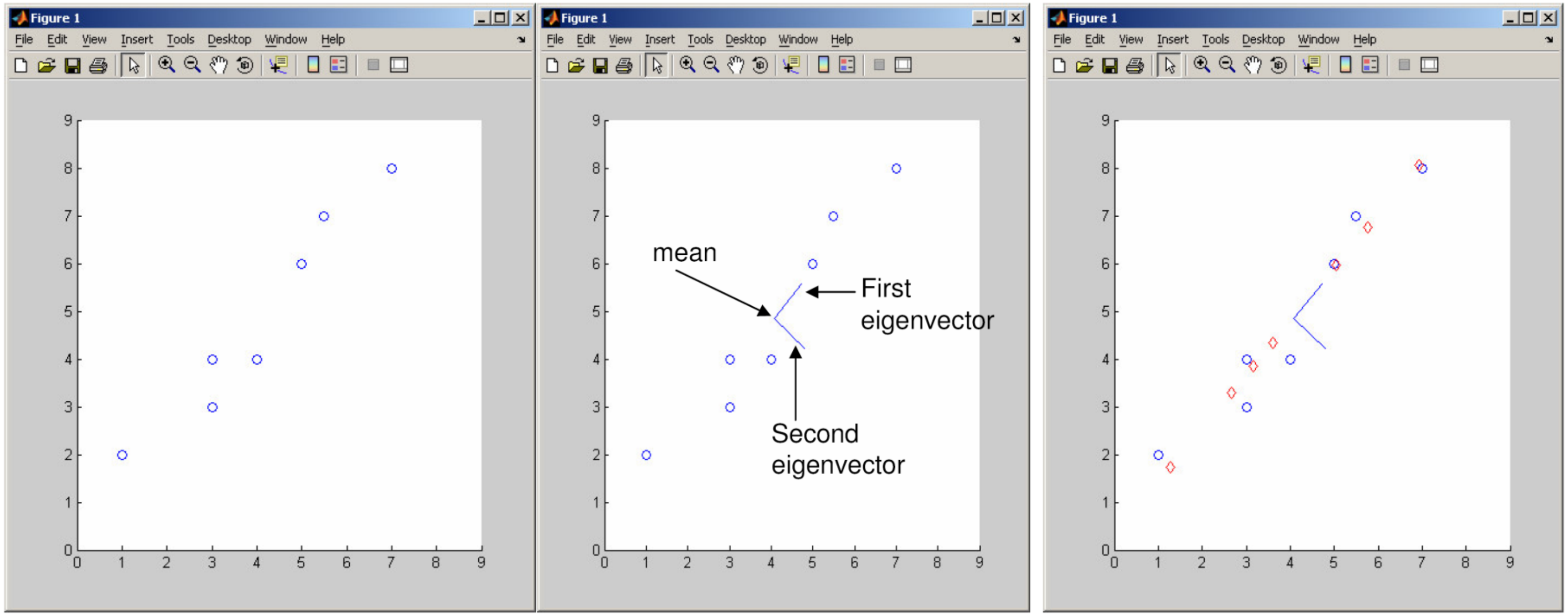
Different interpretation of PCA

- **Minimum Reconstruction Error:** PCA finds vectors v such that projection on to the vectors yields minimum MSE reconstruction (As we just used)

- **Maximum Variance Subspace:** PCA finds vectors v such that projections on to the vectors capture maximum variance in the data



PCA Example



Reconstructed data using only first eigenvector (k=1)

PCA and SVD

- We can compute the principal components by SVD of X:

$$\underline{X = U\Sigma V^T}$$

$$XX^T = U\Sigma V^T (U\Sigma V^T)^T =$$

$$= U\Sigma V^T V\Sigma^T U^T = \underline{U\tilde{\Sigma}^2 U^T}$$

- Thus, the left singular vectors of X are the principal components! We sort them by the size of the singular values of X.

PCA for Image Compression




d=1 **d=2** **d=4** **d=8**

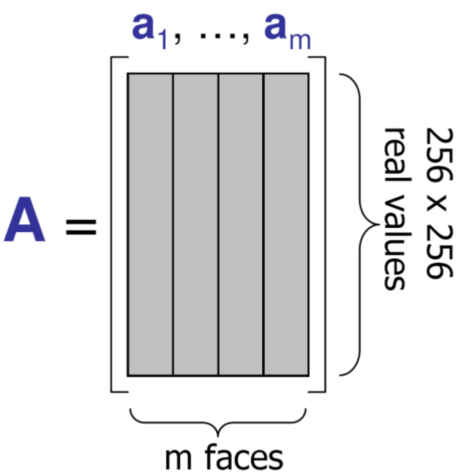
d=16 **d=32** **d=64** **d=100** **Original Image**





Eigenfaces

- Example data set: Images of faces
  - Famous Eigenface approach
    [Turk & Pentland], [Sirovich & Kirby]
- Each face **a** is …
  - 256 x 256 values (luminance at location)
  - **a** in $\Re^{256\times256}$ (view as 1D vector)
- Form $A = [\,\mathbf{a}_1, \ldots, \mathbf{a}_m\,]$
- Compute $\Sigma = AA^T$
- Problem: Σ is 64K $\times$ 64K … HUGE!!!



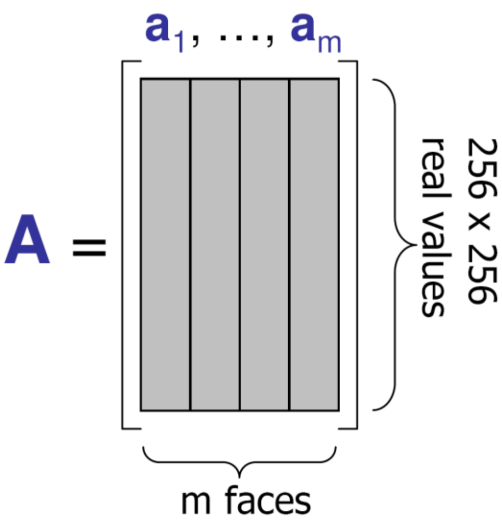
18

Computational Complexity

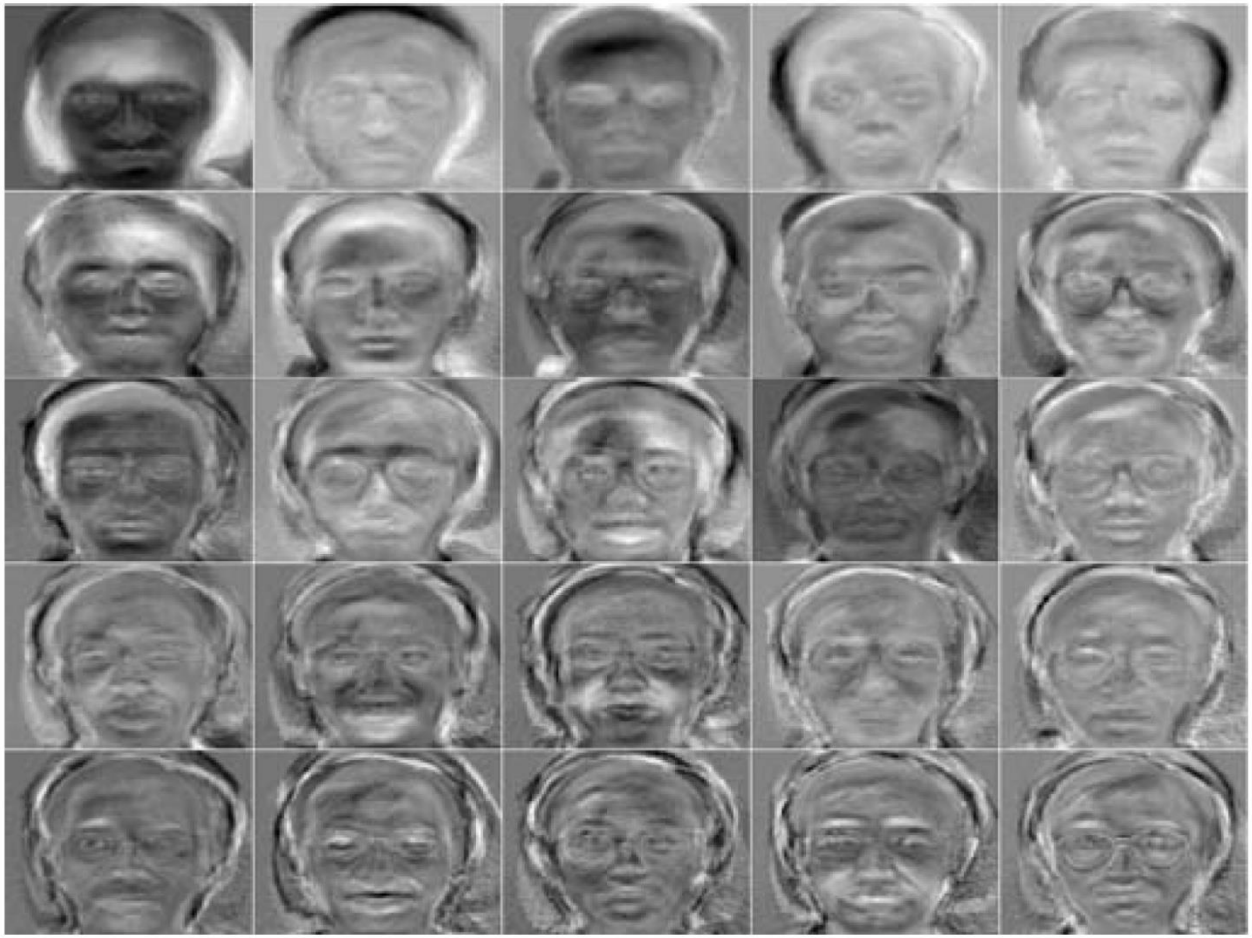
- Suppose m instances, each of size d
  - Eigenfaces: $m=500$ faces, each of size $d=64K$
- Given $d \times d$ covariance matrix Σ, can compute
  - all d eigenvectors/eigenvalues in $O(d^3)$
  - first k eigenvectors/eigenvalues in $O(k\, d^2)$
- But if $d=64K$, EXPENSIVE!

A Clever Workaround

- Note that $m<<64K$
- Use $L=A^TA$ instead of $\Sigma=AA^T$
- If $\mathbf{v}$ is eigenvector of L
  then $A\mathbf{v}$ is eigenvector of Σ



Principle Components



Principle Components

Pros and Cons

- 
  - Eigenvector method
  - No tuning parameters
  - Non-iterative
  - No local optima

- 
  - Limited to second order statistics
  - Limited to linear projections