DATA, MSML, BIOI 602
Principles of Data Science

SVD and PCA

Heng Huang
Department of Computer Science

Orthogonal Matrix

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

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

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

- For an orthogonal matrix, we have

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

Eigenvalue & Eigenvector

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

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

- 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:

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

where **I** is an identity matrix

Ref: Mingyue Ding

Singular Values

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

What is SVD?

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

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

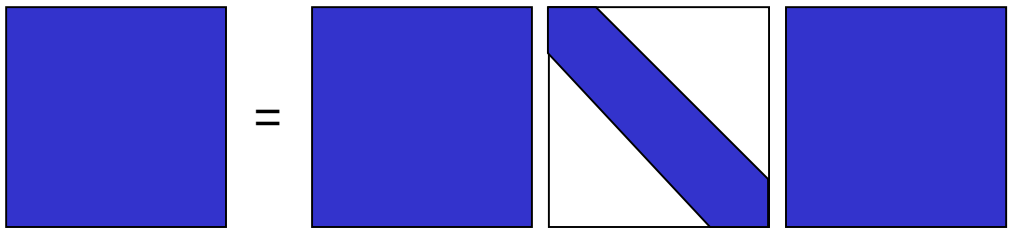
where **U and V** are orthogonal matrices and **D** is a diagonal matrix containing singular values, $\{\mu_i, i = 1, 2, \cdots, r\}$. This factored matrix representation is known as the SVD.

Ref: Mingyue Ding

SVD More Formally

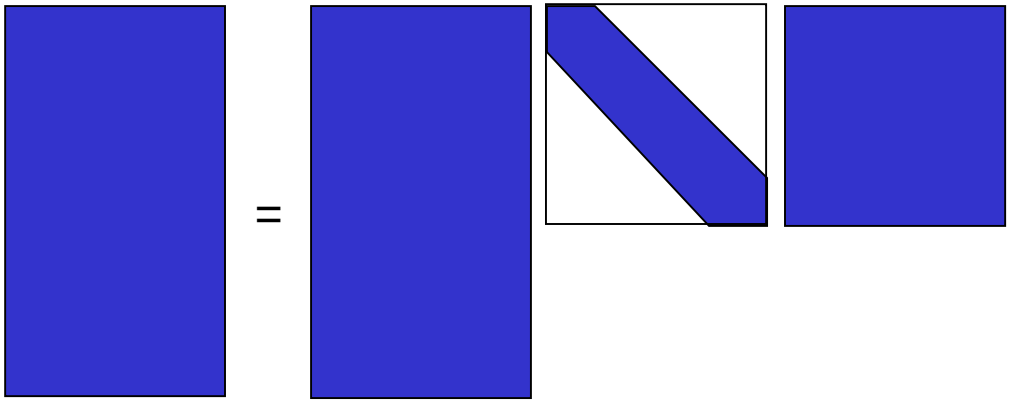
- The diagonal values of Σ (μ_1, …, μ_n) are called the singular values. It is accustomed to sort them: $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_n$
- The columns of U ($\mathbf{u}_1$, ..., $\mathbf{u}_n$) are called the left singular vectors. They are the axes of the ellipsoid.
- The columns of V ($\mathbf{v}_1$, ..., $\mathbf{v}_n$) are called the right singular vectors. They are the preimages of the axes of the ellipsoid.

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



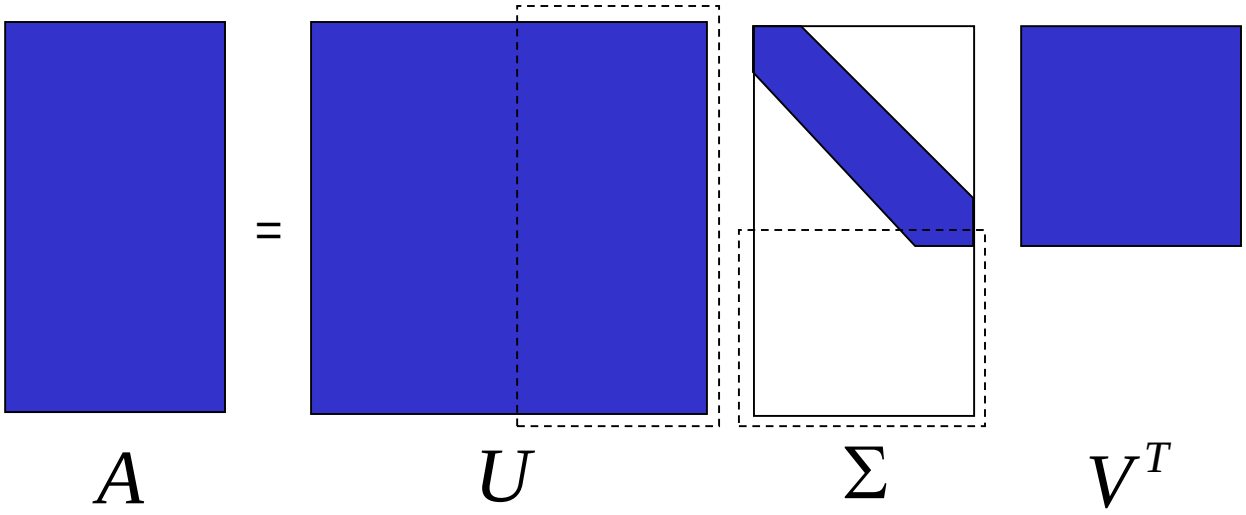
Reduced SVD

- 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



Full SVD

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



Example of SVD

- Suppose

$$\mathbf{A} = \begin{pmatrix} 1.2 & 0.9 & -4 \\ 1.6 & 1.2 & 3 \end{pmatrix}$$

1) Calculate the eignvalues of $\mathbf{A}^T\mathbf{A}$

$$\lambda_1 = 25, \lambda_2 = 6.25$$

2) The non-zero singular values, $\mu_i = \sqrt{\lambda_i}$,e.g.,

$$\mu_1 = 5, \mu_2 = 2.5$$

3) **V** formed from the orthonormal eigenvectors as columns,

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{V}\boldsymbol{\Sigma}^{\mathrm{T}}\mathbf{U}^{\mathrm{T}}\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{V}\boldsymbol{\Sigma}^{\mathrm{T}}\boldsymbol{\Sigma}\mathbf{V}^{\mathrm{T}}.$$

Ref: Mingyue Ding

Example of SVD

$$\mathbf{V} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix}^T = \begin{pmatrix} 0 & 0.8 & -0.6 \\ 0 & 0.6 & 0.8 \\ 1 & 0 & 0 \end{pmatrix}$$

- Calculate $\mathbf{U} = \{\mathbf{A}v_j / \mu_j,\ j = 1,2\}$

  Where v_j is the non-zero eigenvector of $\mathbf{A}^T\mathbf{A}$

  We have

$$\mathbf{U} = \begin{pmatrix} -0.8 & 0.6 \\ 0.6 & 0.8 \end{pmatrix}$$

Example of SVD

- Now the SVD of **A** is

$$\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^T = \begin{pmatrix} -0.8 & 0.6 \\ 0.6 & 0.8 \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 \\ 0 & 2.5 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0.8 & -0.6 \\ 0 & 0.6 & 0.8 \\ 1 & 0 & 0 \end{pmatrix}^T$$

$$\mathbf{A}^+ = \mathbf{V}\mathbf{D}^+\mathbf{U}^T = \begin{pmatrix} 0 & 0.8 & -0.6 \\ 0 & 0.6 & 0.8 \\ 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{2.5} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -0.8 & 0.6 \\ 0.6 & 0.8 \end{pmatrix}^T$$

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 $A\mathbf{x} = \mathbf{b}$

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

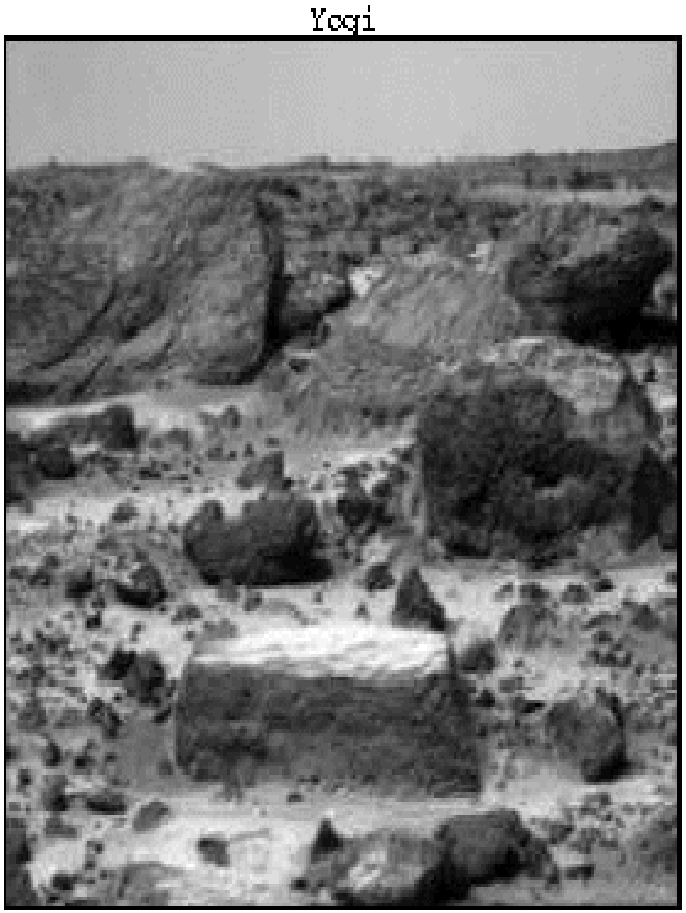
Application: Image Compression

- 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)

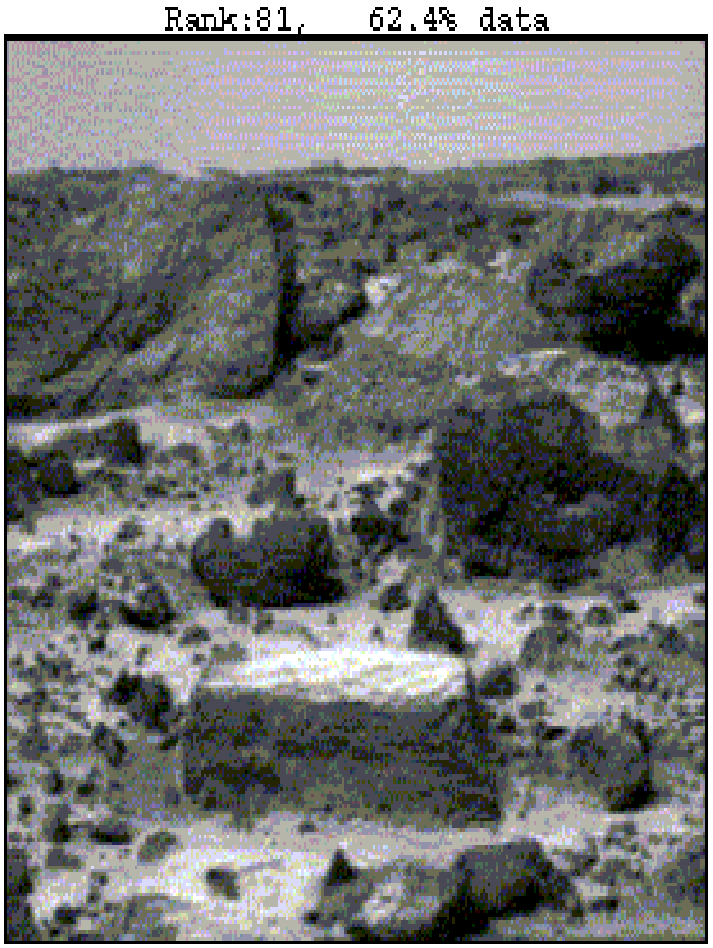
- Source: [Will]
- Yogi: Rock photographed by Sojourner Mars mission.
- 256 $\times$ 264 grayscale bitmap $\rightarrow$ 256 $\times$ 264 matrix $\boldsymbol{M}$
- Pixel values $\in$ [0,1]
- ~ 67584 numbers



Ref: Bernick

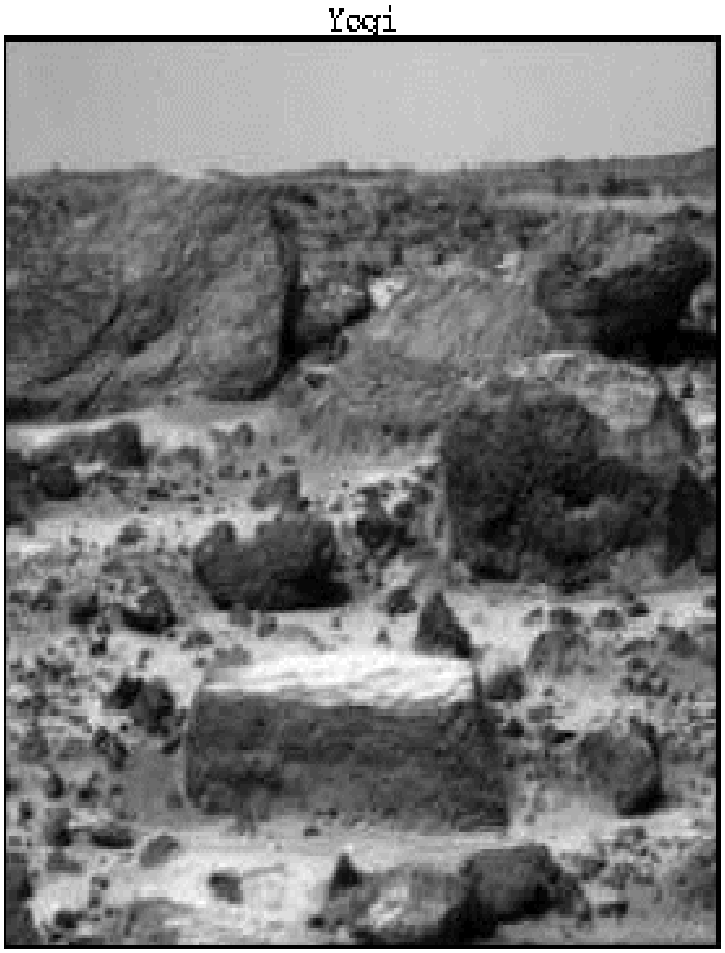
Example: Yogi (Compressed)

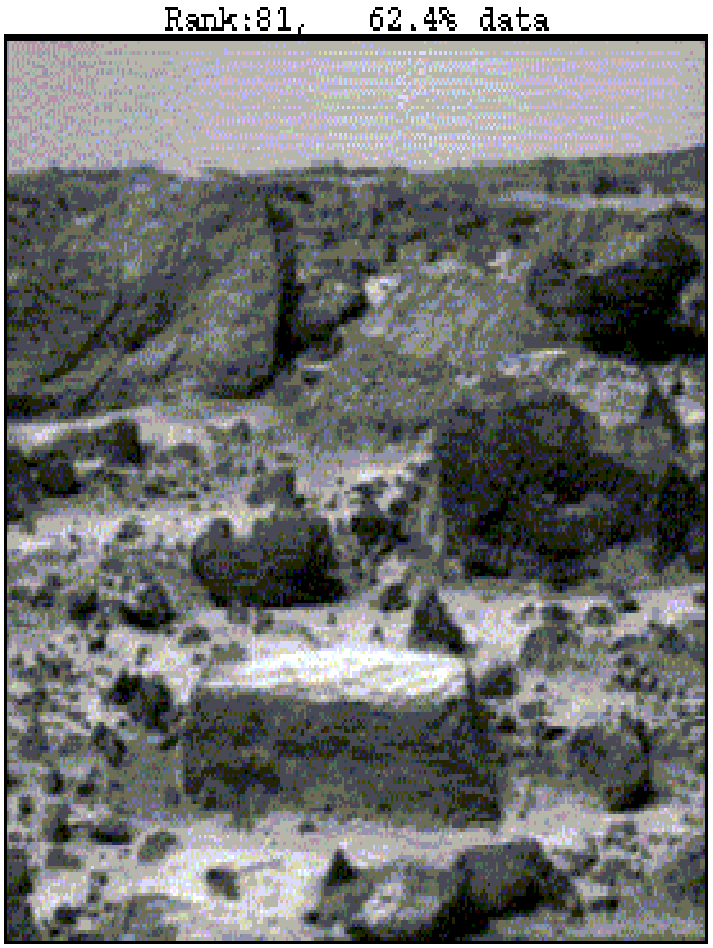
- $\boldsymbol{M}$ has 256 singular values
- Rank 81 approximation of $\boldsymbol{M}$:
- $81 \times (256 + 264 + 1) =$ ~ 42201 numbers



Ref: Bernick

Example: Yogi (Both)





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)

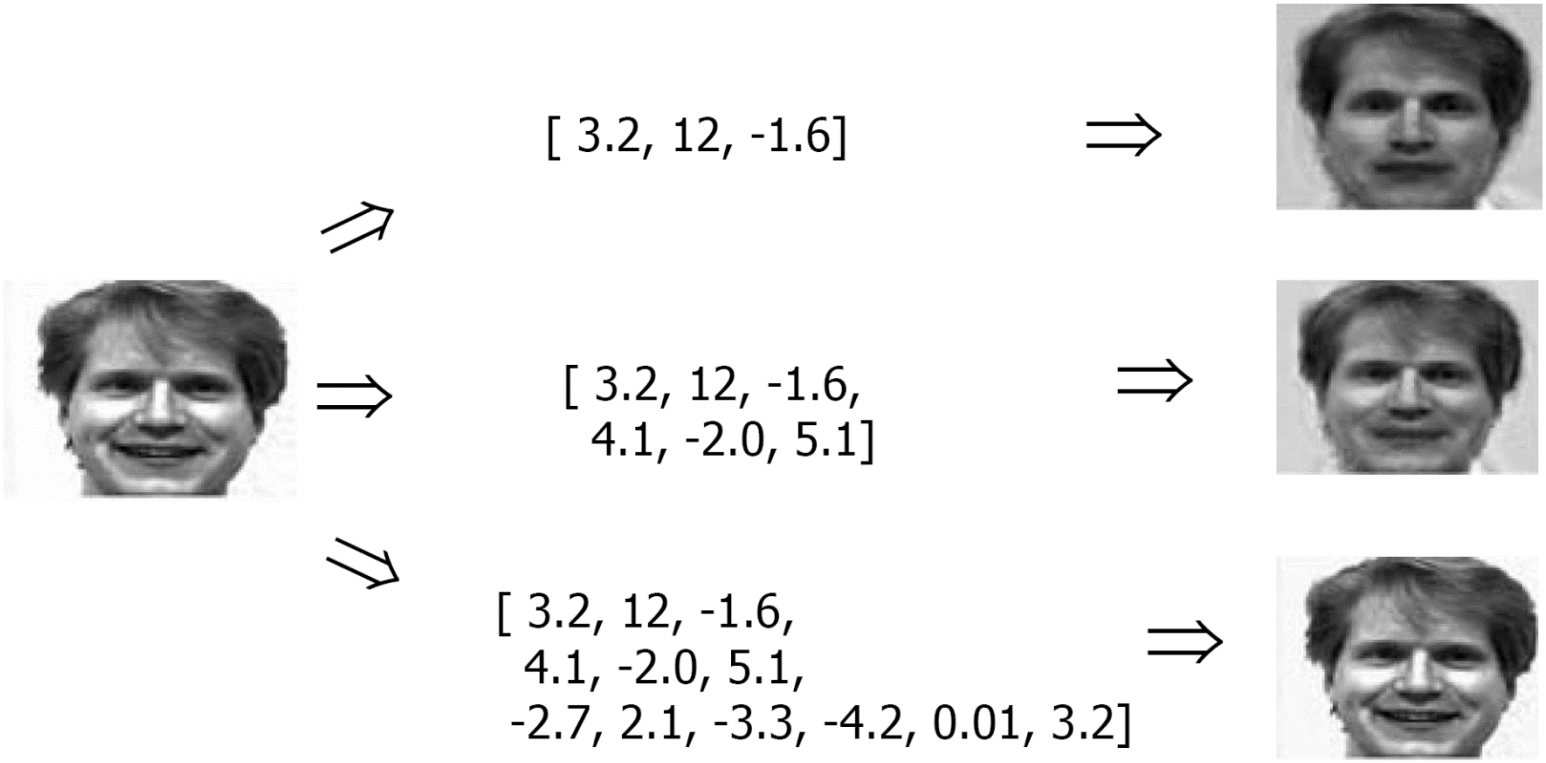
- Idea:
  - Given data points in d-dimensional space, project into *lower dimensional* space while *preserving as much information* as possible
    - Eg, find best planar approximation to 3D data
    - Eg, find best 12-D approximation to 10^4-D data
  - In particular, choose projection that minimizes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

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!

An Vision Application: Facial Recognition



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

Projection

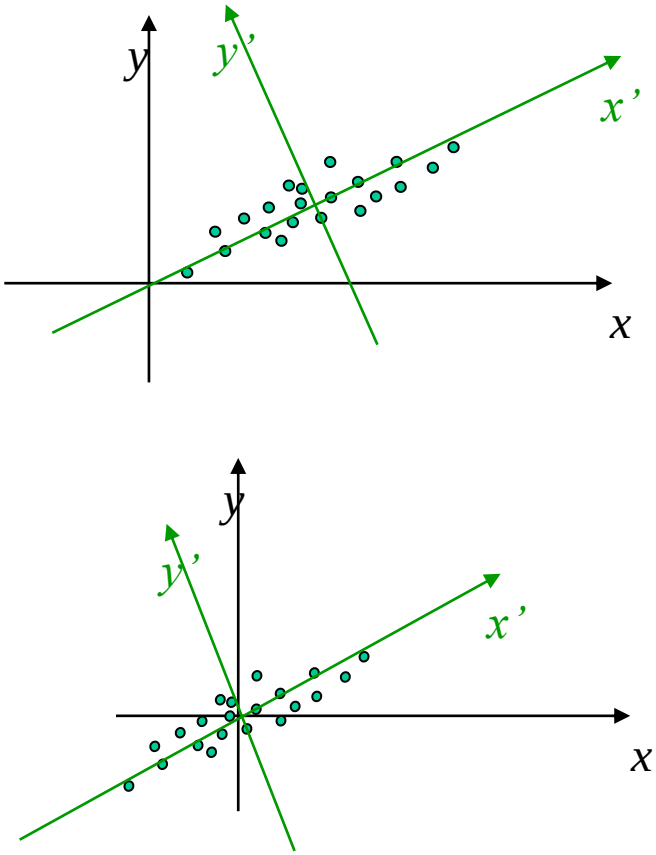
- Orthonormal basis $\rightarrow$ trivial projection
- Given basis $U = \{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$

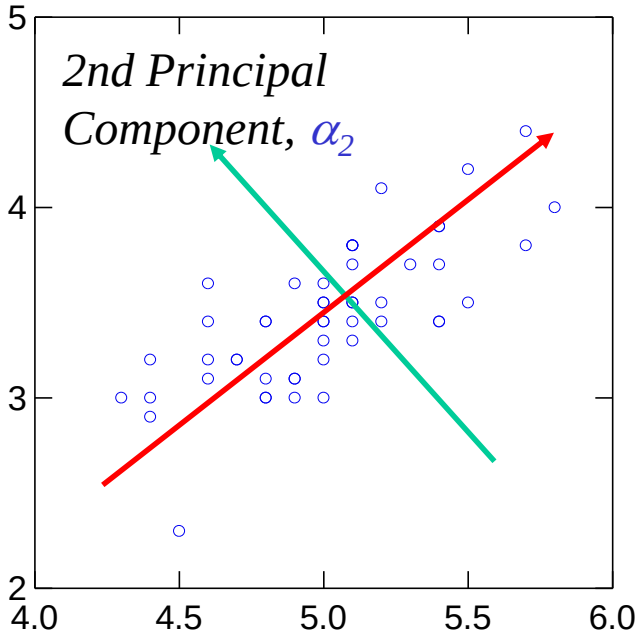
  can project any d-dim $\mathbf{x}$ to k values
  - $\alpha_1 = \mathbf{u}_1^T \mathbf{x}$ $\alpha_2 = \mathbf{u}_2^T \mathbf{x}$ ... $\alpha_k = \mathbf{u}_k^T \mathbf{x}$
  - $\boldsymbol{\alpha} = \mathbf{U}^T \mathbf{x}$
  - $\mathbf{x} \approx \sum_i \alpha_i \mathbf{u}_i = \sum_i (\mathbf{u}_i^T \mathbf{x})\, \mathbf{u}_i$ ["=" if all d values]

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

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

Principal Components Analysis





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

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

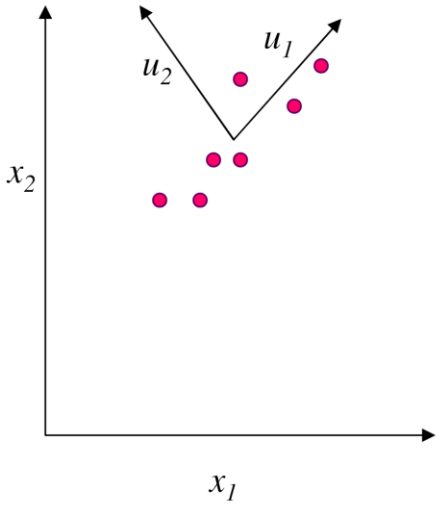
PCA: given k<d. Find { $\mathbf{u}_1$, …, $\mathbf{u}_k$ }

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

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

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

Covariance matrix:

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

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

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

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

Justifying Use of Eigenvectors

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

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

- Set derivative to 0:

$$\sum \mathbf{u} - \lambda \mathbf{u} = 0$$

- 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

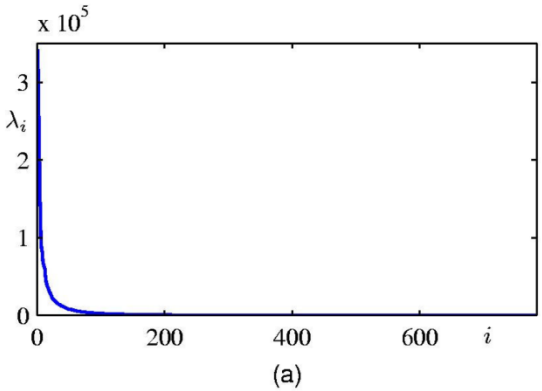
Minimize $E_k = \sum_{i=k+1}^{d} \mathbf{u}_i^T \Sigma \mathbf{u}_i$

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

Eigenvalue Eigenvector

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

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



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

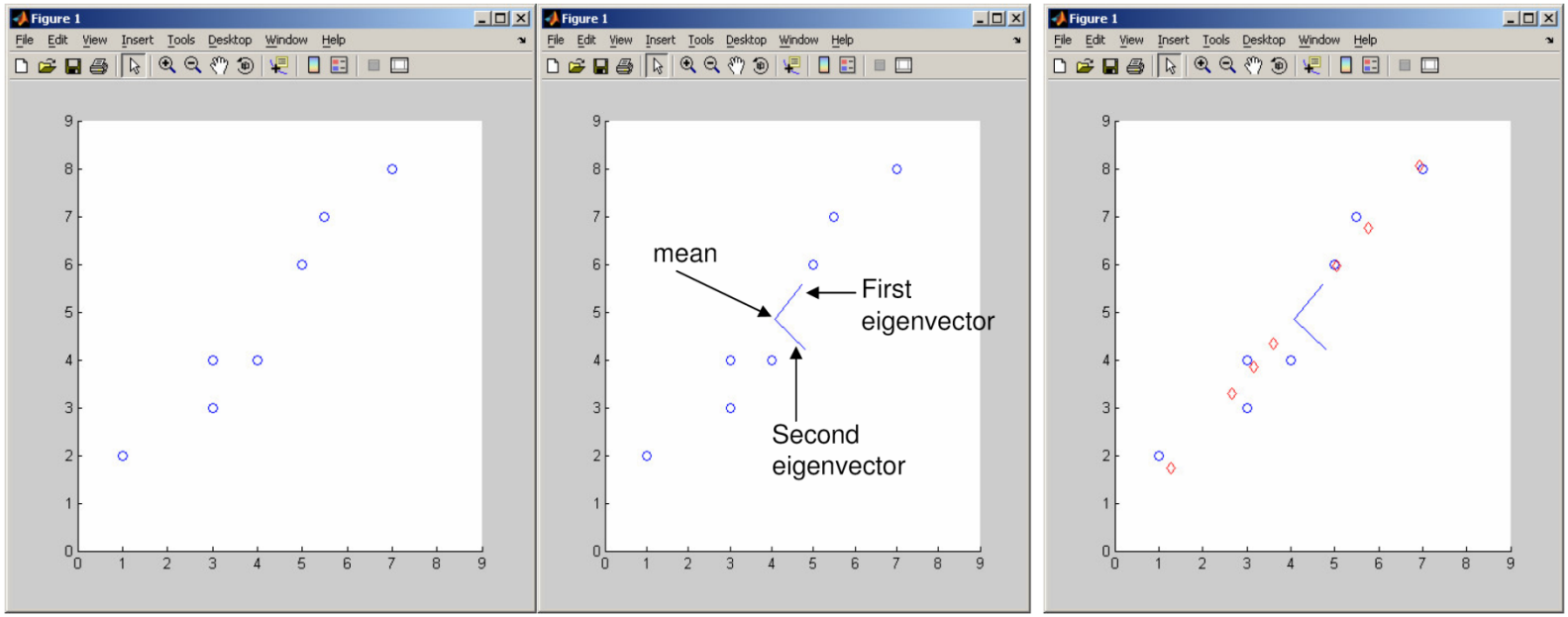
PCA Algorithm

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
- $\Sigma \leftarrow A\,A^T$ … 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

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

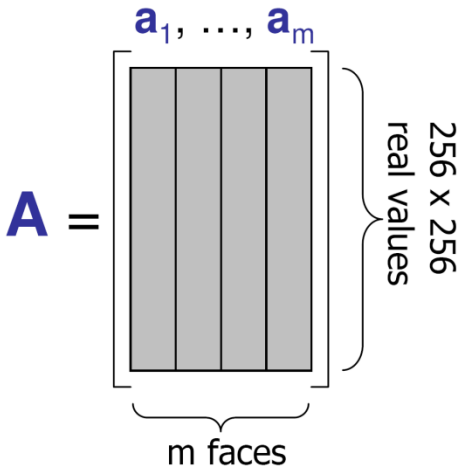
d=32

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

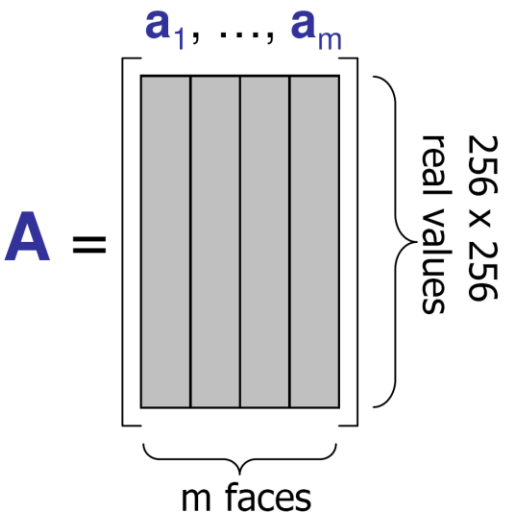
Computational Complexity

- Suppose m instances, each of size d
  - Eigenfaces: m=500 faces, each of size d=64K
- Given d×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 $L = A^T A$ instead of $\Sigma = AA^T$
- If $\mathbf{v}$ is eigenvector of L
  then $A\mathbf{v}$ is eigenvector of Σ



Proof:

$$L\,\mathbf{v} = \gamma\,\mathbf{v}$$

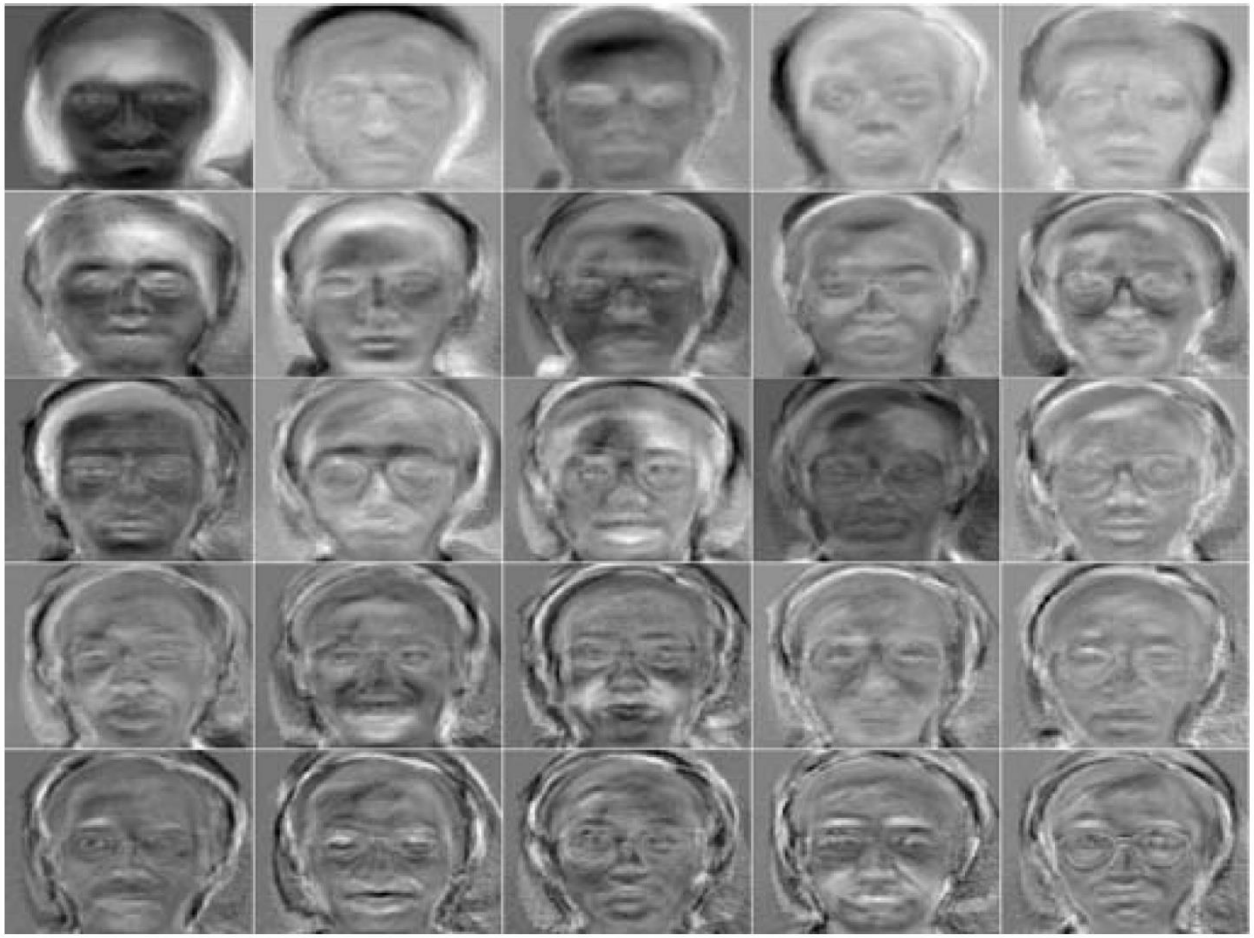
$$A^T A\,\mathbf{v} = \gamma\,\mathbf{v}$$

$$A\,(A^T A\,\mathbf{v}) = A(\gamma\,\mathbf{v}) = \gamma\,A\mathbf{v}$$

$$(A\,A^T)A\,\mathbf{v} = \gamma\,(A\mathbf{v})$$

$$\Sigma\,(A\mathbf{v}) = \gamma\,(A\mathbf{v})$$

Principle Components



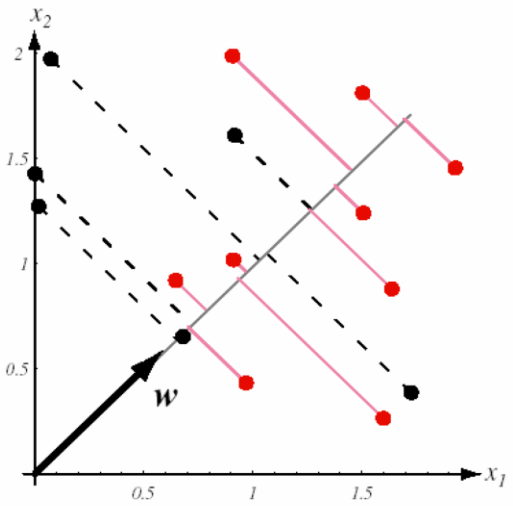
DATA, MSML, BIOI 602
Principles of Data Science

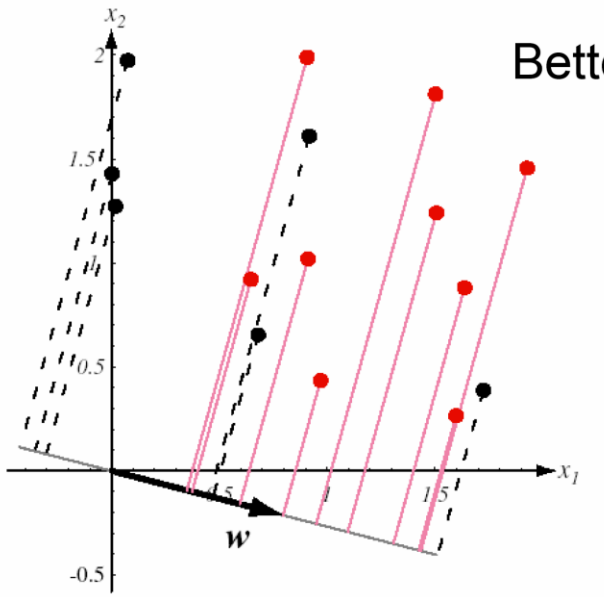
Fisher Linear Discriminant Analysis

Heng Huang
Department of Computer Science

Fisher Linear Discriminant

Classes mixed



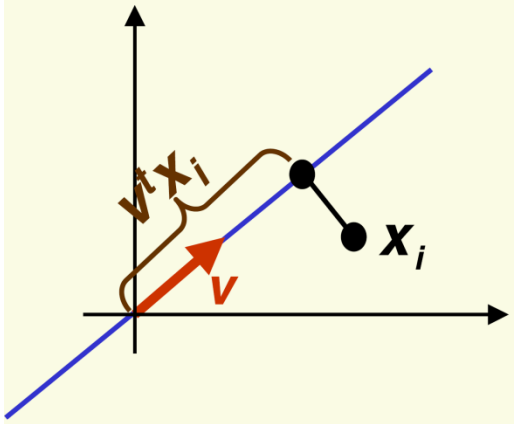


Better Separation

The figure on the right shows greater separation between subsets, one set of the points with dashed line, another with solid line.

Fisher Linear Discriminant

- Suppose we have 2 classes and ***d***-dimensional samples $\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n$ where
  - $\boldsymbol{n}_1$ samples come from the first class
  - $\boldsymbol{n}_2$ samples come from the second class
- consider projection on a line
- Let the line direction be given by unit vector ***v***



- Scalar $\boldsymbol{v}^t\boldsymbol{x}_i$ is the distance of projection of $\boldsymbol{x}_i$ from the origin
- Thus it $\boldsymbol{v}^t\boldsymbol{x}_i$ is the projection of $\boldsymbol{x}_i$ into a one dimensional subspace

Ref: Olga Veksler

Fisher Linear Discriminant

- Thus the projection of sample $\boldsymbol{x}_i$ onto a line in direction $\boldsymbol{v}$ is given by $\boldsymbol{v}^t\boldsymbol{x}_i$
- How to measure separation between projections of different classes?
- Let $\tilde{\mu}_1$ and $\tilde{\mu}_2$ be the means of projections of classes 1 and 2
- Let μ_1 and μ_2 be the means of classes 1 and 2
- $|\tilde{\mu}_1 - \tilde{\mu}_2|$ seems like a good measure

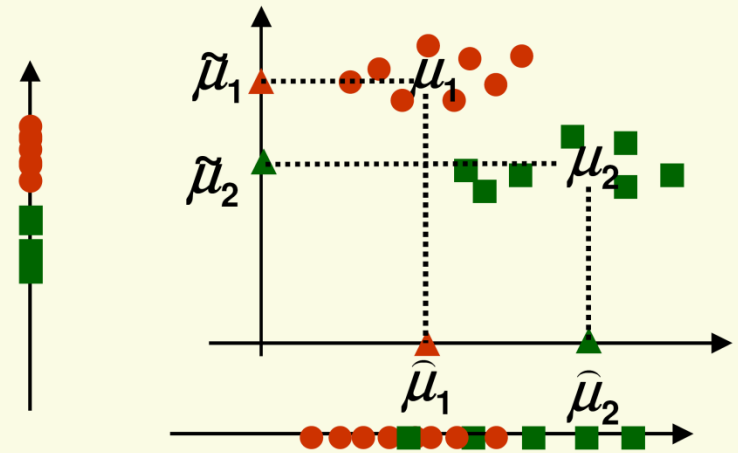
$$\tilde{\mu}_1 = \frac{1}{n_1}\sum_{x_i \in C1}^{n_1} \boldsymbol{v}^t \boldsymbol{x}_i = \boldsymbol{v}^t\left(\frac{1}{n_1}\sum_{x_i \in C1}^{n_1} \boldsymbol{x}_i\right) = \boldsymbol{v}^t\mu_1$$

similarly, $\tilde{\mu}_2 = \boldsymbol{v}^t\mu_2$

Ref: Olga Veksler

Fisher Linear Discriminant

- How good is $|\tilde{\mu}_1 - \tilde{\mu}_2|$ as a measure of separation?
  - The larger $|\tilde{\mu}_1 - \tilde{\mu}_2|$, the better is the expected separation

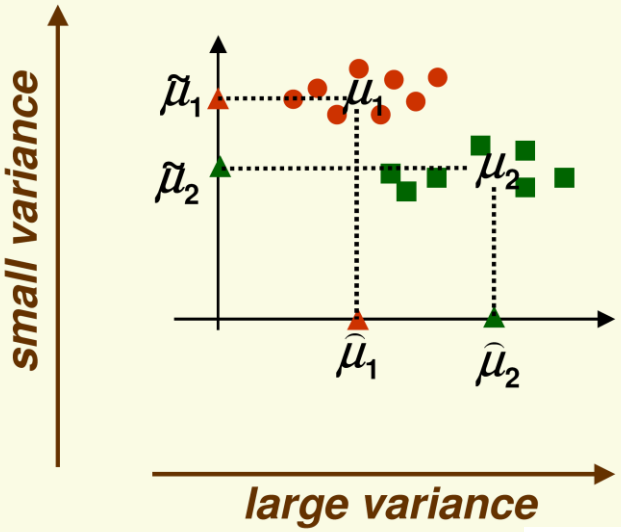


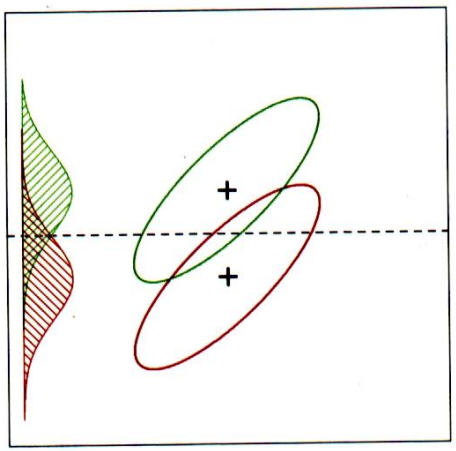
- the vertical axes is a better line than the horizontal axes to project to for class separability
- however $|\hat{\mu}_1 - \hat{\mu}_2| > |\tilde{\mu}_1 - \tilde{\mu}_2|$

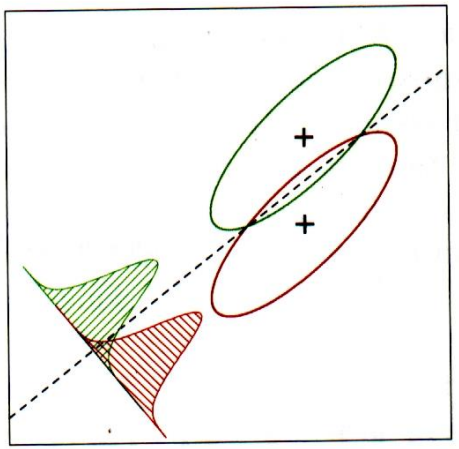
Ref: Olga Veksler

Fisher Linear Discriminant

- The problem with $|\tilde{\mu}_1 - \tilde{\mu}_2|$ is that it does not consider the variance of the classes







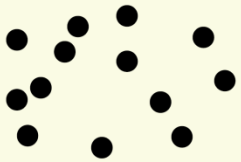
Ref: Olga Veksler

Fisher Linear Discriminant

- We need to normalize $|\tilde{\mu}_1 - \tilde{\mu}_2|$ by a factor which is proportional to variance
- Have samples $z_1,\ldots,z_n$. Sample mean is $\mu_z = \frac{1}{n}\sum_{i=1}^{n} z_i$
- Define their ***scatter*** as

$$s = \sum_{i=1}^{n} (z_i - \mu_z)^2$$

- Thus scatter is just sample variance multiplied by n
  - scatter measures the same thing as variance, the spread of data around the mean
  - scatter is just on different scale than variance

larger scatter: 

smaller scatter: 

Ref: Olga Veksler

Fisher Linear Discriminant

- Fisher Solution: normalize $|\tilde{\mu}_1 - \tilde{\mu}_2|$ by scatter
- Let $y_i = v^t x_i$, i.e. y_i 's are the projected samples
- Scatter for projected samples of class 1 is

$$\tilde{s}_1^2 = \sum_{y_i \in Class\ 1} (y_i - \tilde{\mu}_1)^2$$

- Scatter for projected samples of class 2 is

$$\tilde{s}_2^2 = \sum_{y_i \in Class\ 2} (y_i - \tilde{\mu}_2)^2$$

Ref: Olga Veksler

Fisher Linear Discriminant

- We need to normalize by both scatter of class 1 and scatter of class 2
- Thus Fisher linear discriminant is to project on line in the direction $\boldsymbol{v}$ which maximizes

want projected means are far from each other

$$J(\boldsymbol{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

want scatter in class 1 is as small as possible, i.e. samples of class 1 cluster around the projected mean $\tilde{\mu}_1$

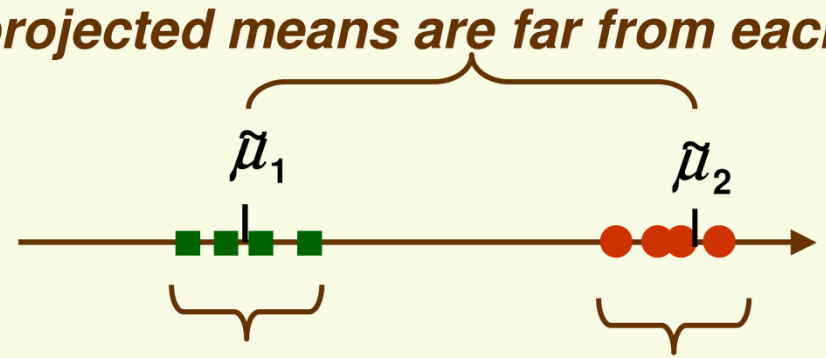
want scatter in class 2 is as small as possible, i.e. samples of class 2 cluster around the projected mean $\tilde{\mu}_2$

Ref: Olga Veksler

Fisher Linear Discriminant

$$J(\boldsymbol{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

- If we find $\boldsymbol{v}$ which makes $J(\boldsymbol{v})$ large, we are guaranteed that the classes are well separated



small $\tilde{s}_1$ implies that projected samples of class 1 are clustered around projected mean

small $\tilde{s}_2$ implies that projected samples of class 2 are clustered around projected mean

Ref: Olga Veksler

Fisher Linear Discriminant

$$J(\boldsymbol{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

- All we need to do now is to express $\boldsymbol{J}$ explicitly as a function of $\boldsymbol{v}$ and maximize it
  - straightforward but need linear algebra and Calculus
- Define the separate class scatter matrices $\boldsymbol{S_1}$ and $\boldsymbol{S_2}$ for classes 1 and 2. These measure the scatter of original samples $\boldsymbol{x_i}$ (before projection)

$$\boldsymbol{S}_1 = \sum_{\boldsymbol{x}_i \in \textit{Class } 1} (\boldsymbol{x}_i - \mu_1)(\boldsymbol{x}_i - \mu_1)^t$$

$$\boldsymbol{S}_2 = \sum_{\boldsymbol{x}_i \in \textit{Class } 2} (\boldsymbol{x}_i - \mu_2)(\boldsymbol{x}_i - \mu_2)^t$$

Ref: Olga Veksler

Fisher Linear Discriminant

- Now define the ***within*** the class scatter matrix

$$S_W = S_1 + S_2$$

- Recall that $\tilde{s}_1^2 = \sum_{y_i \in Class\ 1} (y_i - \tilde{\mu}_1)^2$
- Using $y_i = v^t x_i$ and $\tilde{\mu}_1 = v^t \mu_1$

$$\begin{aligned}
\tilde{s}_1^2 &= \sum_{y_i \in Class\ 1} \left(v^t x_i - v^t \mu_1\right)^2 \\
&= \sum_{y_i \in Class\ 1} \left(v^t (x_i - \mu_1)\right)^t \left(v^t (x_i - \mu_1)\right) \\
&= \sum_{y_i \in Class\ 1} \left((x_i - \mu_1)^t v\right)^t \left((x_i - \mu_1)^t v\right) \\
&= \sum_{y_i \in Class\ 1} v^t (x_i - \mu_1)(x_i - \mu_1)^t v = v^t S_1 v
\end{aligned}$$

Ref: Olga Veksler

Fisher Linear Discriminant

- Similarly $\tilde{s}_2^2 = \boldsymbol{v}^t \boldsymbol{S}_2 \boldsymbol{v}$
- Therefore $\tilde{s}_1^2 + \tilde{s}_2^2 = \boldsymbol{v}^t \boldsymbol{S}_1 \boldsymbol{v} + \boldsymbol{v}^t \boldsymbol{S}_2 \boldsymbol{v} = \boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}$
- Define between the class scatter matrix

$$\boldsymbol{S}_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t$$

- $\boldsymbol{S}_B$ measures separation between the means of two classes (before projection)
- Let's rewrite the separations of the projected means

$$\begin{aligned}(\tilde{\mu}_1 - \tilde{\mu}_2)^2 &= (\boldsymbol{v}^t \mu_1 - \boldsymbol{v}^t \mu_2)^2 \\ &= \boldsymbol{v}^t (\mu_1 - \mu_2)(\mu_1 - \mu_2)^t \boldsymbol{v} \\ &= \boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}\end{aligned}$$

Ref: Olga Veksler

Fisher Linear Discriminant

- Thus our objective function can be written:

$$J(\boldsymbol{v}) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{\boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}}$$

- **Maximize** $J(\boldsymbol{v})$ by taking the derivative w.r.t. $\boldsymbol{v}$ and setting it to 0

$$\frac{d}{d\boldsymbol{v}} J(\boldsymbol{v}) = \frac{\left(\frac{d}{d\boldsymbol{v}} \boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}\right) \boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v} - \left(\frac{d}{d\boldsymbol{v}} \boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}\right) \boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}}{\left(\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}\right)^2}$$

$$= \frac{(2\boldsymbol{S}_B \boldsymbol{v}) \boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v} - (2\boldsymbol{S}_W \boldsymbol{v}) \boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}}{\left(\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}\right)^2} = 0$$

Ref: Olga Veksler

Fisher Linear Discriminant

- Need to solve $\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}(\boldsymbol{S}_B \boldsymbol{v}) - \boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}(\boldsymbol{S}_W \boldsymbol{v}) = \boldsymbol{0}$

$$\Rightarrow \frac{\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}(\boldsymbol{S}_B \boldsymbol{v})}{\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}} - \frac{\boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}(\boldsymbol{S}_W \boldsymbol{v})}{\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}} = \boldsymbol{0}$$

$$\Rightarrow \boldsymbol{S}_B \boldsymbol{v} - \frac{\boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}(\boldsymbol{S}_W \boldsymbol{v})}{\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}} = \boldsymbol{0}, \quad \frac{\boldsymbol{v}^t \boldsymbol{S}_B \boldsymbol{v}}{\boldsymbol{v}^t \boldsymbol{S}_W \boldsymbol{v}} = \lambda$$

$$\Rightarrow \underbrace{\boldsymbol{S}_B \boldsymbol{v} = \lambda \boldsymbol{S}_W \boldsymbol{v}}$$

generalized eigenvalue problem

Ref: Olga Veksler

Multiple Discriminant Analysis

- We want to find a transform matrix W that maximizes the ratio of the determinants of the between-class scatter to the within-class scatter:

$$J(W) = \frac{\left|\tilde{S}_B\right|}{\left|\tilde{S}_W\right|} = \frac{\left|W^t S_B W\right|}{\left|W^t S_W W\right|}$$

- The columns of an optimal W are the generalized eigenvectors corresponding to the largest eigenvalues

$$S_B w_i = \lambda_i S_W w_i$$

- If S_W is nonsingular, $S_W{}^{-1} S_B w_i = \lambda_i w_i$