B: Chapter 12
HTF: Chapter 14.5

Principal Component Analysis

Barnabás Póczos
University of Alberta

Nov 24, 2009

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Contents

- Motivation
- PCA algorithms
- Applications
  - Face recognition
  - Facial expression recognition
- PCA theory
- Kernel-PCA

Some of these slides are taken from

- Karl Booksh Research group
- Tom Mitchell
- Ron Parr

PCA Applications

- Data Visualization
- Data Compression
- Noise Reduction
- Data Classification
- Trend Analysis
- Factor Analysis

Data Visualization

Example:

- Given 53 blood and urine samples (features) from 65 people.

- How can we visualize the measurements?

Data Visualization

- Matrix format (65x53)

Instances

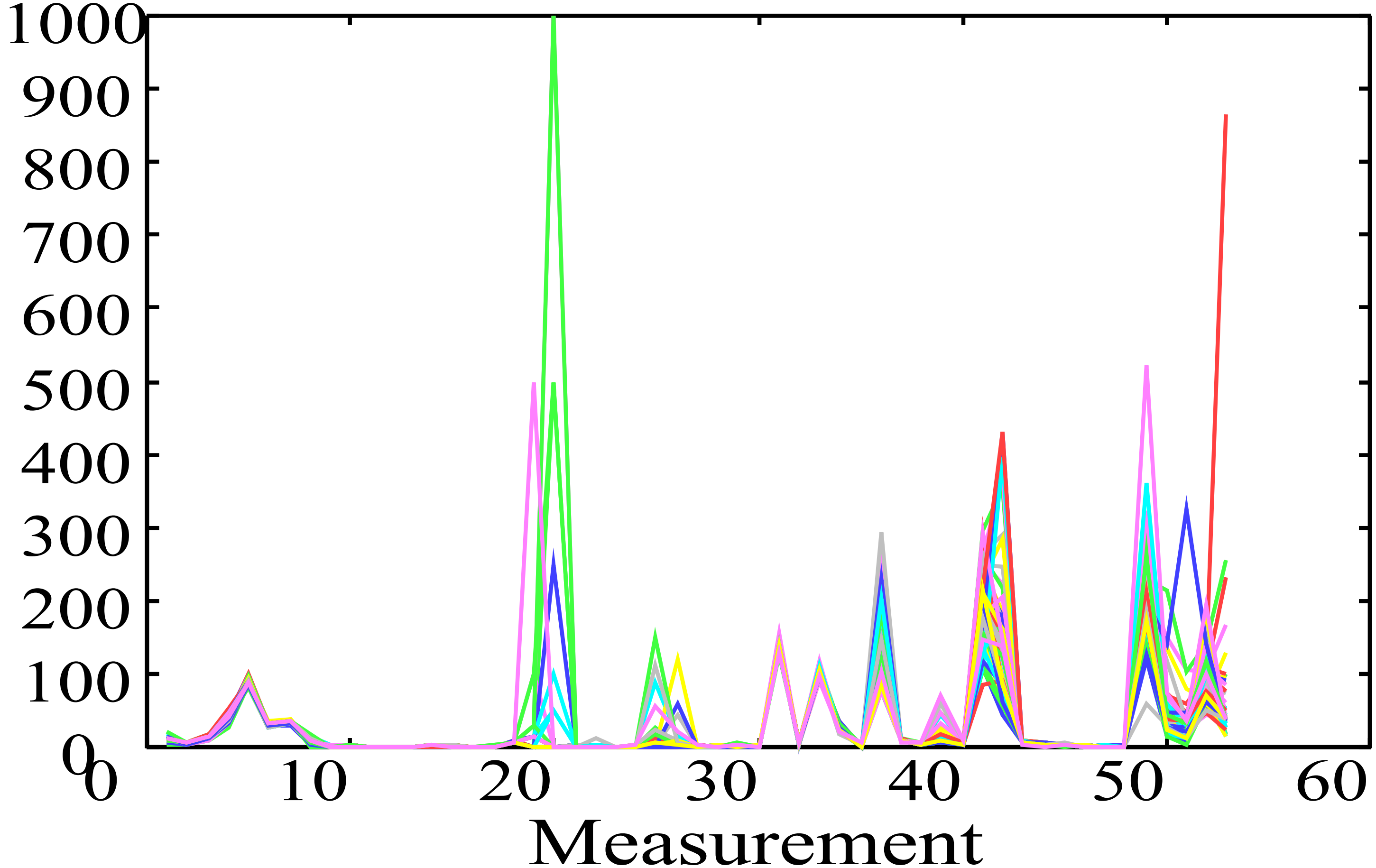
	H-WBC	H-RBC	H-Hgb	H-Hct	H-MCV	H-MCH	H-MCHC
A1	8.0000	4.8200	14.1000	41.0000	85.0000	29.0000	34.0000
A2	7.3000	5.0200	14.7000	43.0000	86.0000	29.0000	34.0000
A3	4.3000	4.4800	14.1000	41.0000	91.0000	32.0000	35.0000
A4	7.5000	4.4700	14.9000	45.0000	101.0000	33.0000	33.0000
A5	7.3000	5.5200	15.4000	46.0000	84.0000	28.0000	33.0000
A6	6.9000	4.8600	16.0000	47.0000	97.0000	33.0000	34.0000
A7	7.8000	4.6800	14.7000	43.0000	92.0000	31.0000	34.0000
A8	8.6000	4.8200	15.8000	42.0000	88.0000	33.0000	37.0000
A9	5.1000	4.7100	14.0000	43.0000	92.0000	30.0000	32.0000

Features

Difficult to see the correlations between the features...

Data Visualization

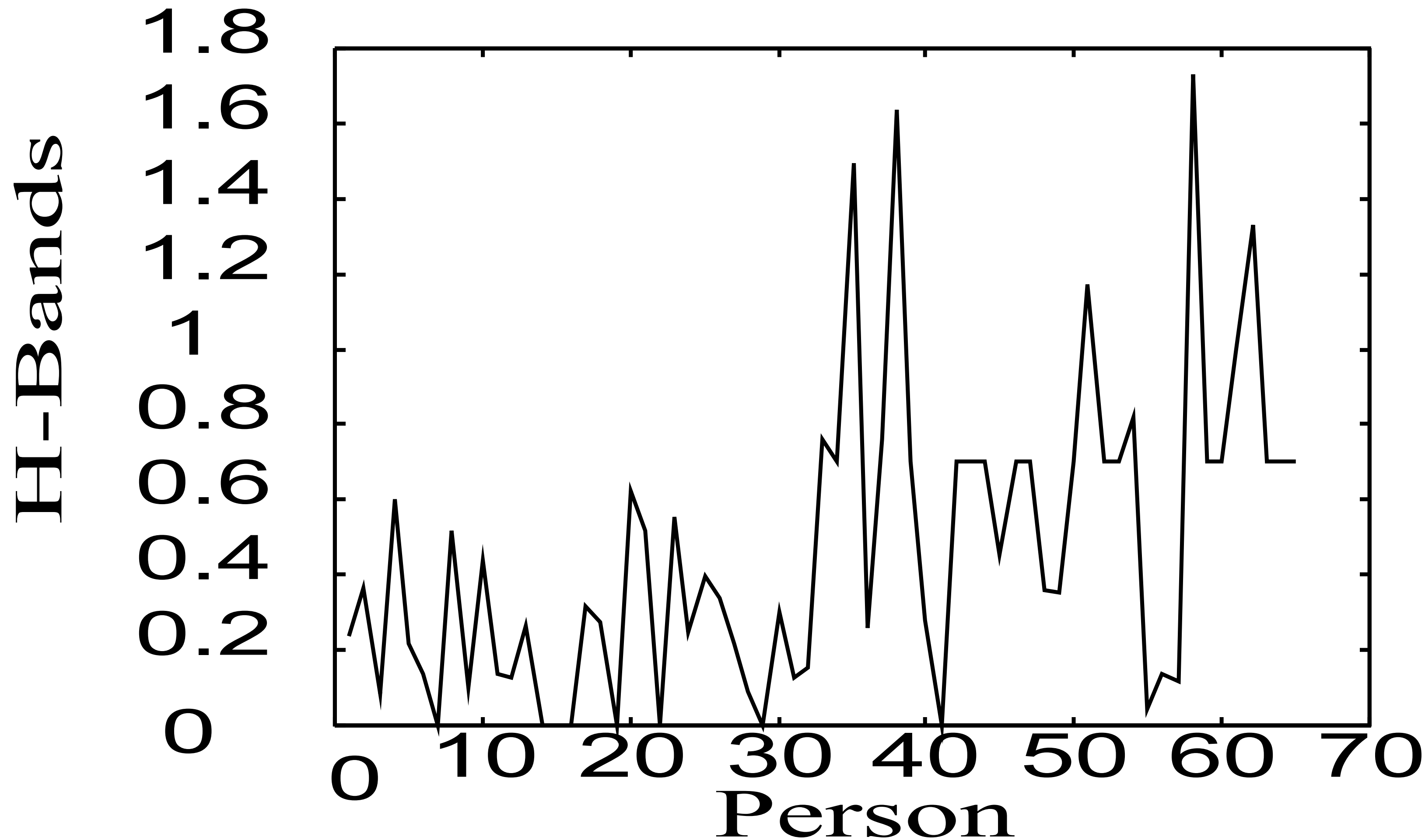
- Spectral format (65 curves, one for each person)



Difficult to compare the different patients...

Data Visualization

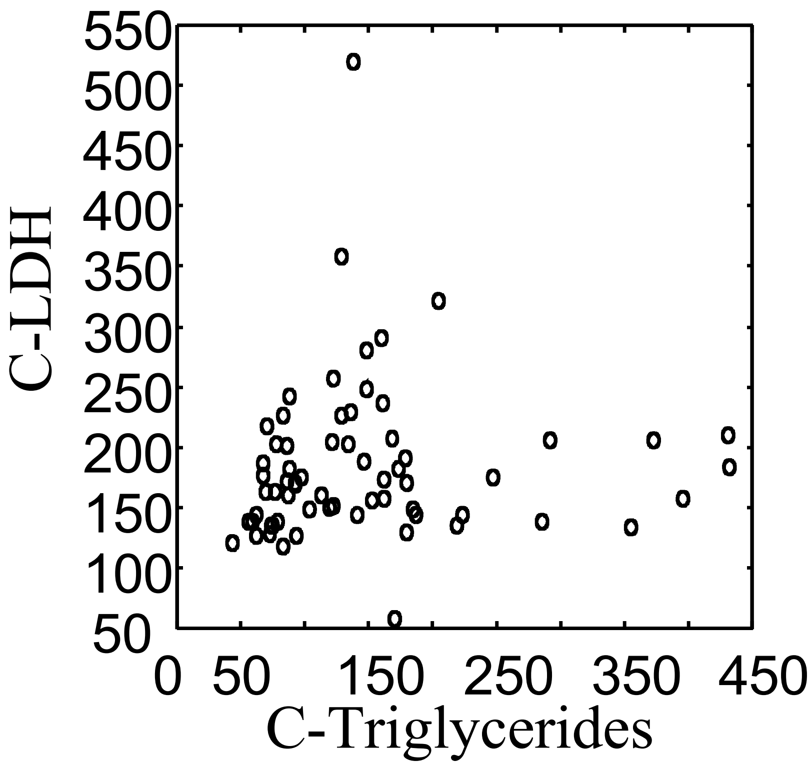
- Spectral format (53 pictures, one for each feature)



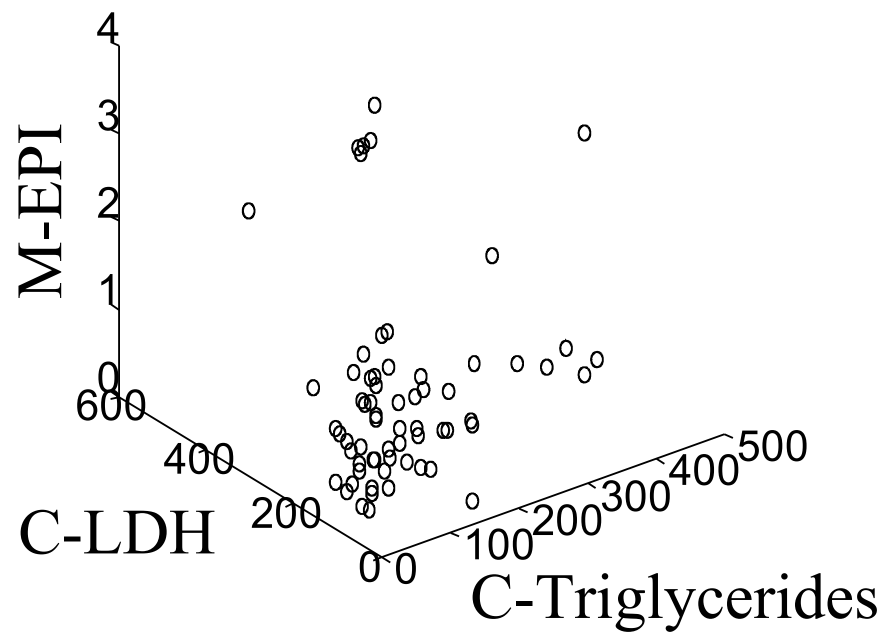
Difficult to see the correlations between the features...

Data Visualization

Bi-variate



Tri-variate



How can we visualize the other variables???

… difficult to see in 4 or higher dimensional spaces...

Data Visualization

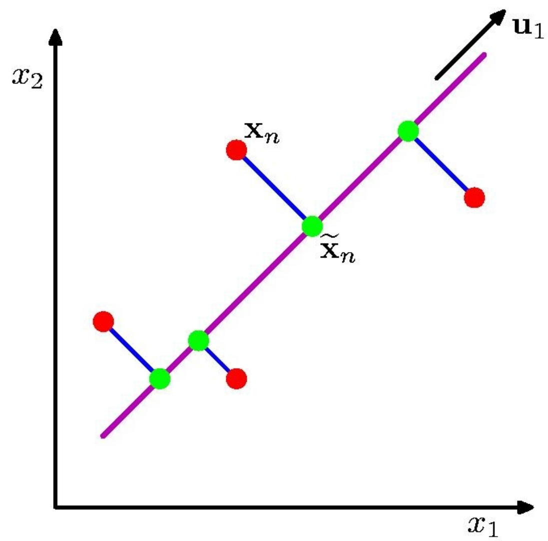
- Is there a representation better than the coordinate axes?

- Is it really necessary to show all the 53 dimensions?
  - … what if there are strong correlations between the features?

- How could we find
  the *smallest* subspace of the 53-D space that
  keeps the *most information* about the original data?

- A solution: **Principal Component Analysis**

Principle Component Analysis



PCA:

Orthogonal projection of data onto lower-dimension linear space that...

- maximizes variance of projected data (purple line)

- minimizes mean squared distance between
  - data point and
  - projections (sum of blue lines)

Principle Components Analysis

Idea:

- Given data points in a 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

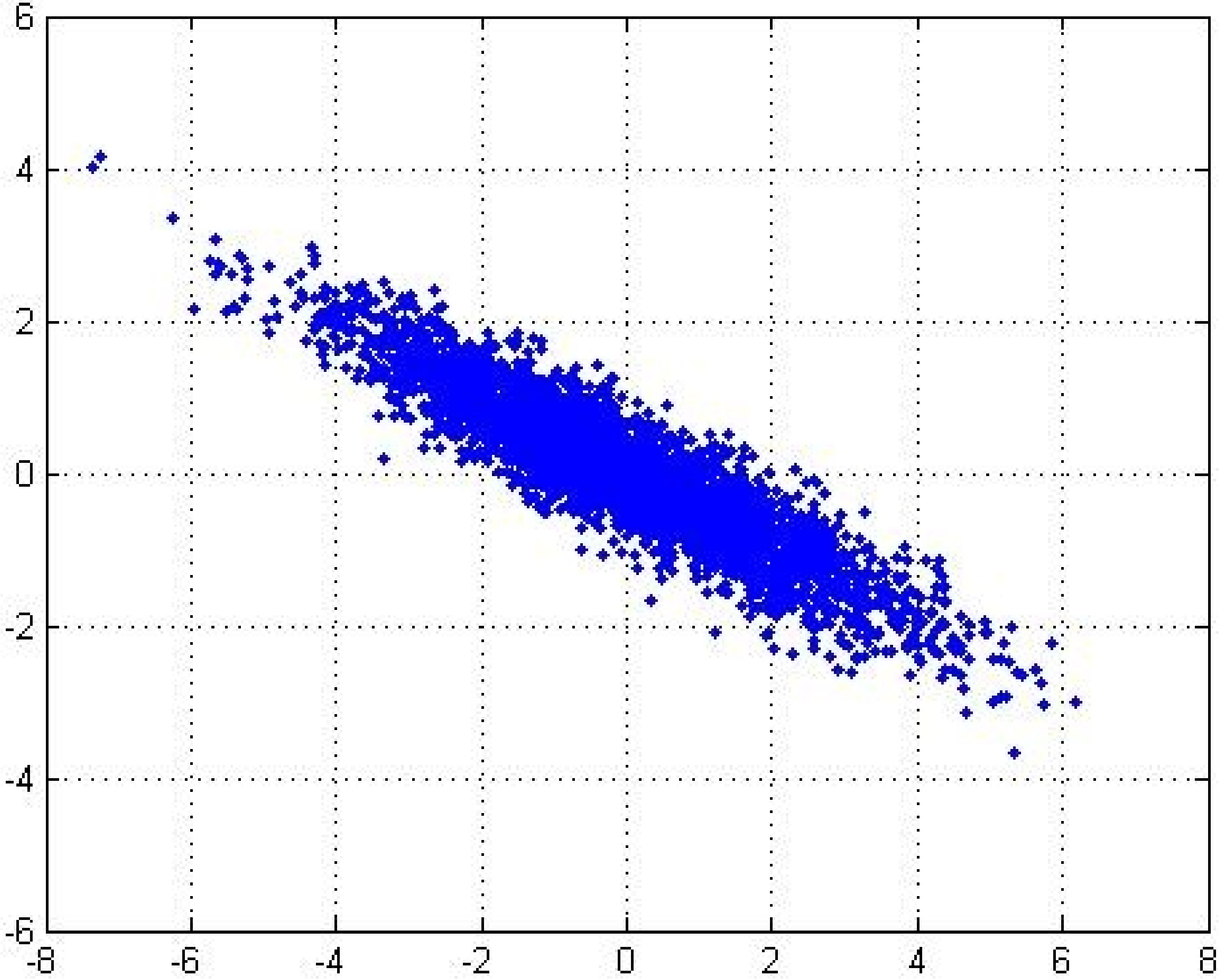
The Principal Components

- **Vectors** originating from the center of mass

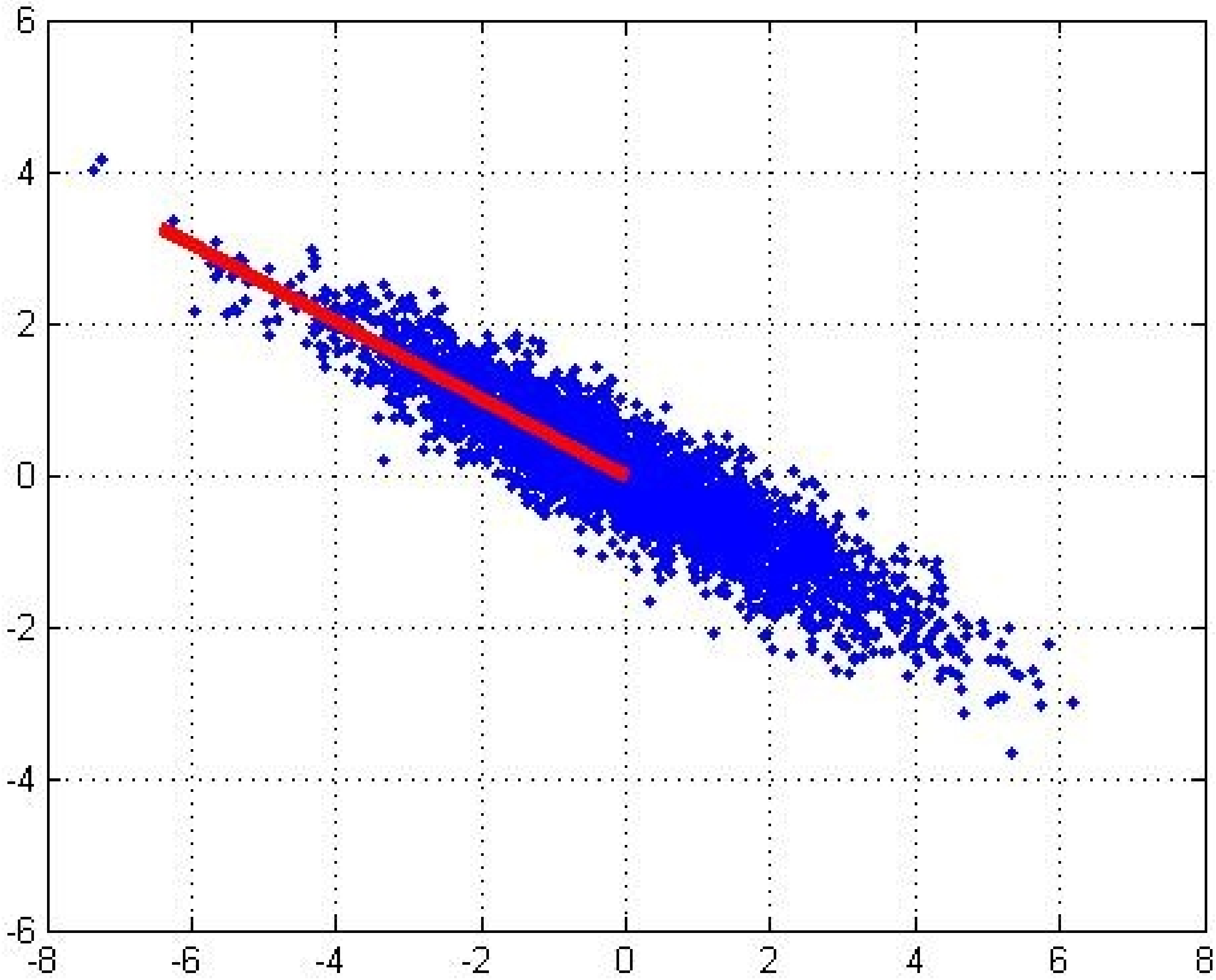
- Principal component #1 points
  in the direction of the **largest variance**.

- Each subsequent principal component…
  - is **orthogonal** to the previous ones, and
  - points in the directions of the **largest variance of the residual subspace**

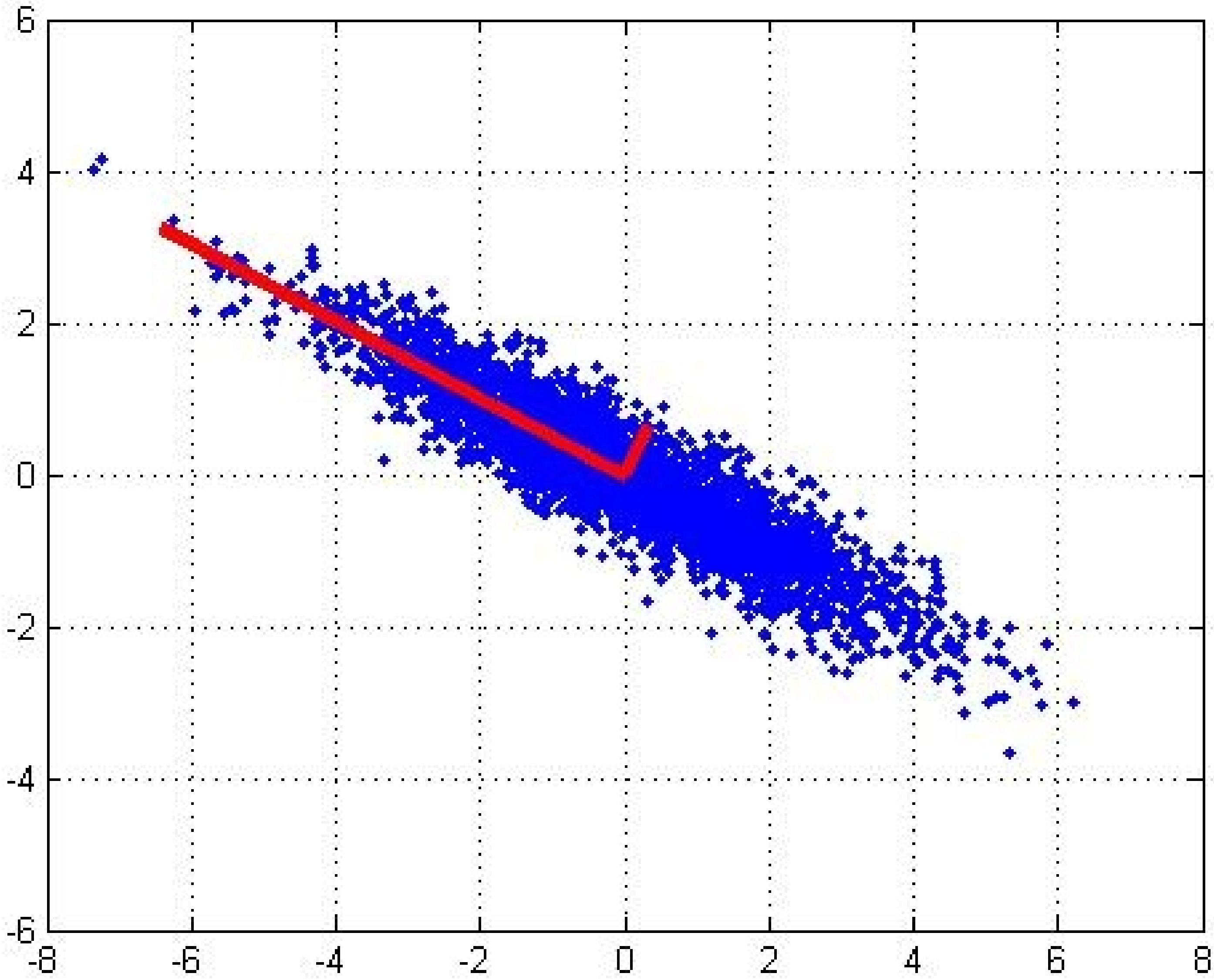
2D Gaussian dataset



1st PCA axis



2nd PCA axis



PCA algorithm I (sequential)

Given the **centered** data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, compute the principal vectors:

$$\mathbf{w}_1 = \arg\max_{\|\mathbf{w}\|=1} \frac{1}{m} \sum_{i=1}^{m} \{(\mathbf{w}^T \mathbf{x}_i)^2\}$$

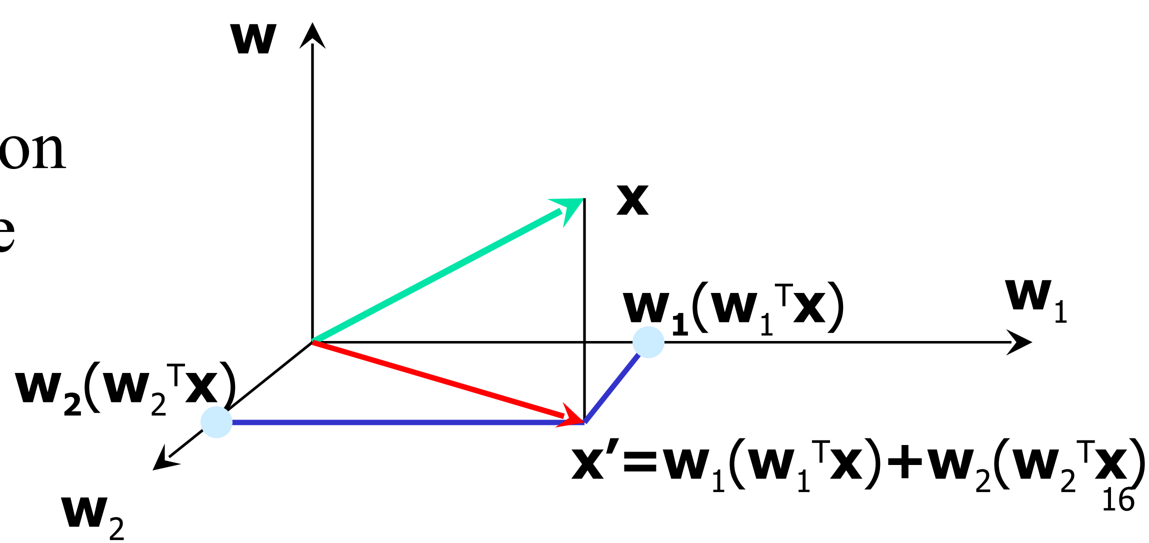
1st PCA vector

We maximize the variance of projection of $\mathbf{x}$

$$\mathbf{w}_k = \arg\max_{\|\mathbf{w}\|=1} \frac{1}{m} \sum_{i=1}^{m} \{[\mathbf{w}^T (\mathbf{x}_i - \underbrace{\sum_{j=1}^{k-1} \mathbf{w}_j \mathbf{w}_j^T \mathbf{x}_i}_{\mathbf{x}' \text{ PCA reconstruction}})]^2\}$$

kth PCA vector

We maximize the variance of the projection in the residual subspace



PCA algorithm II (sample covariance matrix)

- Given data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$, compute covariance matrix $\boldsymbol{\Sigma}$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^T \quad \text{where} \quad \overline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i$$

- **PCA** basis vectors = the eigenvectors of $\boldsymbol{\Sigma}$

- Larger eigenvalue $\Rightarrow$ more important eigenvectors

PCA algorithm II

PCA algorithm($\mathbf{X}$, k): top k eigenvalues/eigenvectors

% $\underline{\mathbf{X}}$ = N × m data matrix,
% … each data point $\mathbf{x}_i$ = column vector, i=1..m

- $$\underline{\mathbf{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{x}_i$$
- $\mathbf{X} \leftarrow$ subtract mean $\underline{\mathbf{x}}$ from each column vector $\mathbf{x}_i$ in $\underline{\mathbf{X}}$
- $\boldsymbol{\Sigma} \leftarrow \mathbf{X}\mathbf{X}^T$ … covariance matrix of $\mathbf{X}$
- $\{ \lambda_i, \mathbf{u}_i \}_{i=1..N}$ = eigenvectors/eigenvalues of $\boldsymbol{\Sigma}$
  … $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$
- Return $\{ \lambda_i, \mathbf{u}_i \}_{i=1..k}$
  % top k principle components

PCA algorithm III (SVD of the data matrix)

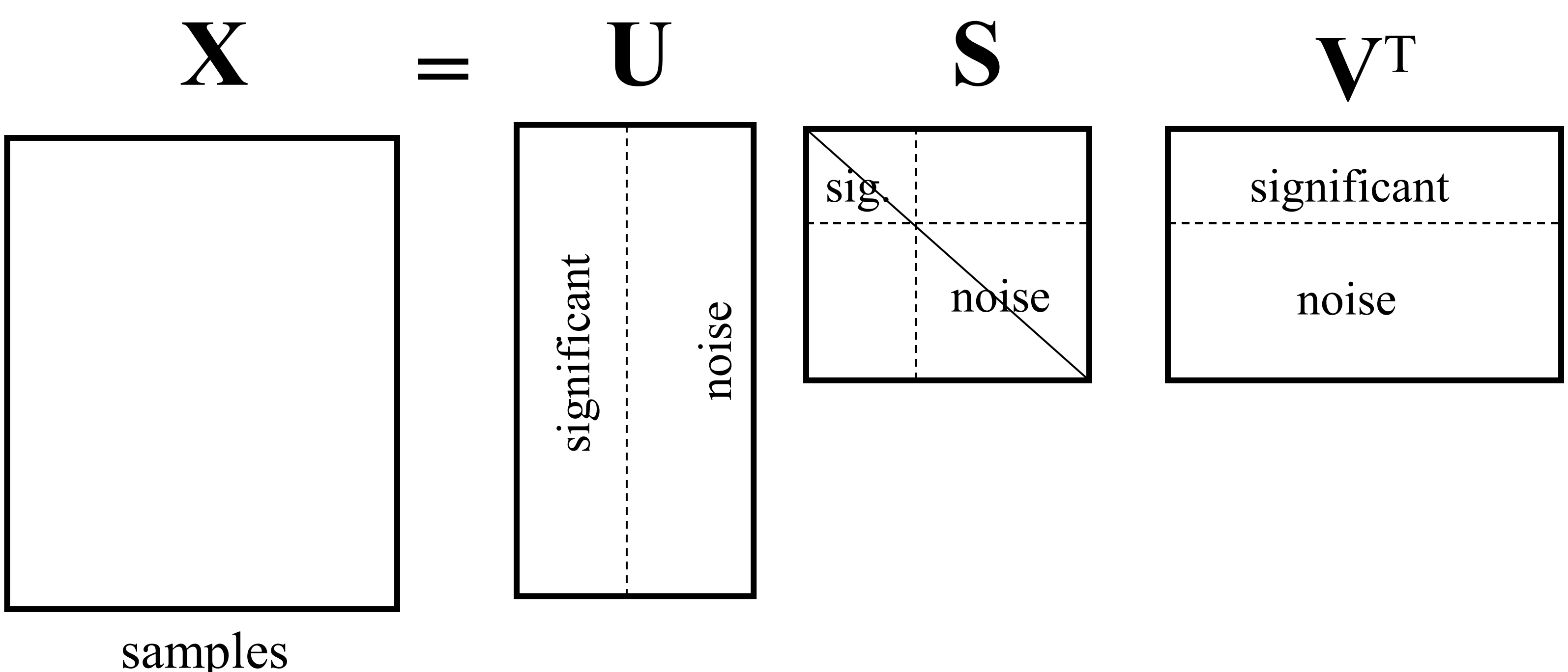
Singular Value Decomposition of the **centered** data matrix **X**.

$$\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{N \times m},$$

m: number of instances,

N: dimension

$$\mathbf{X}_{\text{features} \times \text{samples}} = \mathbf{USV}^T$$



PCA algorithm III

- **Columns of U**
  - the principal vectors, $\{ \mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)} \}$
  - orthogonal and has unit norm – so $U^T U = I$
  - Can reconstruct the data using linear combinations of $\{ \mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(k)} \}$

- **Matrix S**
  - Diagonal
  - Shows importance of each eigenvector

- **Columns of V^T**
  - The coefficients for reconstructing the samples

Face recognition

Challenge: Facial Recognition

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

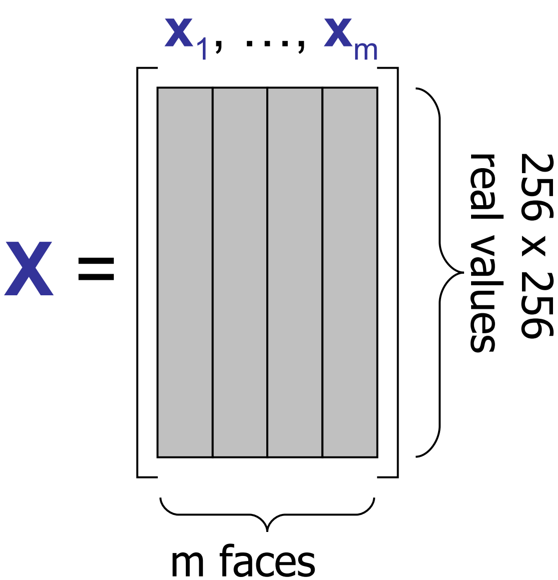
$\Rightarrow$ Can't just use the given 256 x 256 pixels

Applying PCA: Eigenfaces

Method A: Build a PCA subspace for each person and check which subspace can reconstruct the test image the best

Method B: Build one PCA database for the whole dataset and then classify based on the weights.

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



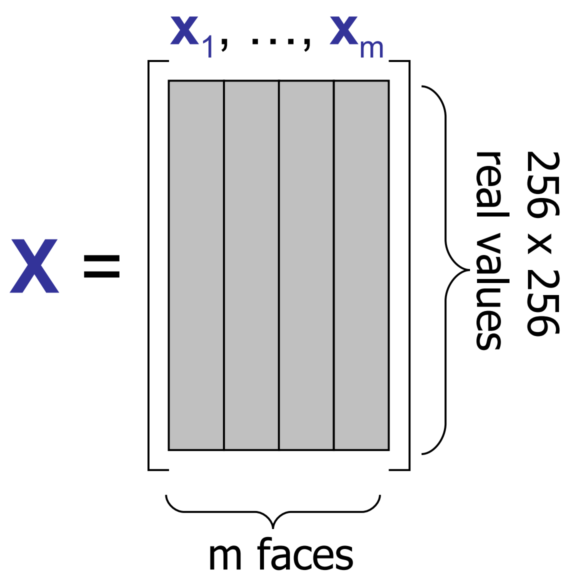
Computational Complexity

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

- But if $N=64K$, EXPENSIVE!

A Clever Workaround

- Note that $m \ll 64K$
- Use $\mathbf{L} = \mathbf{X}^T\mathbf{X}$ instead of $\mathbf{\Sigma} = \mathbf{X}\mathbf{X}^T$
- If $\mathbf{v}$ is eigenvector of $\mathbf{L}$
  then $\mathbf{Xv}$ is eigenvector of $\mathbf{\Sigma}$



Proof:

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

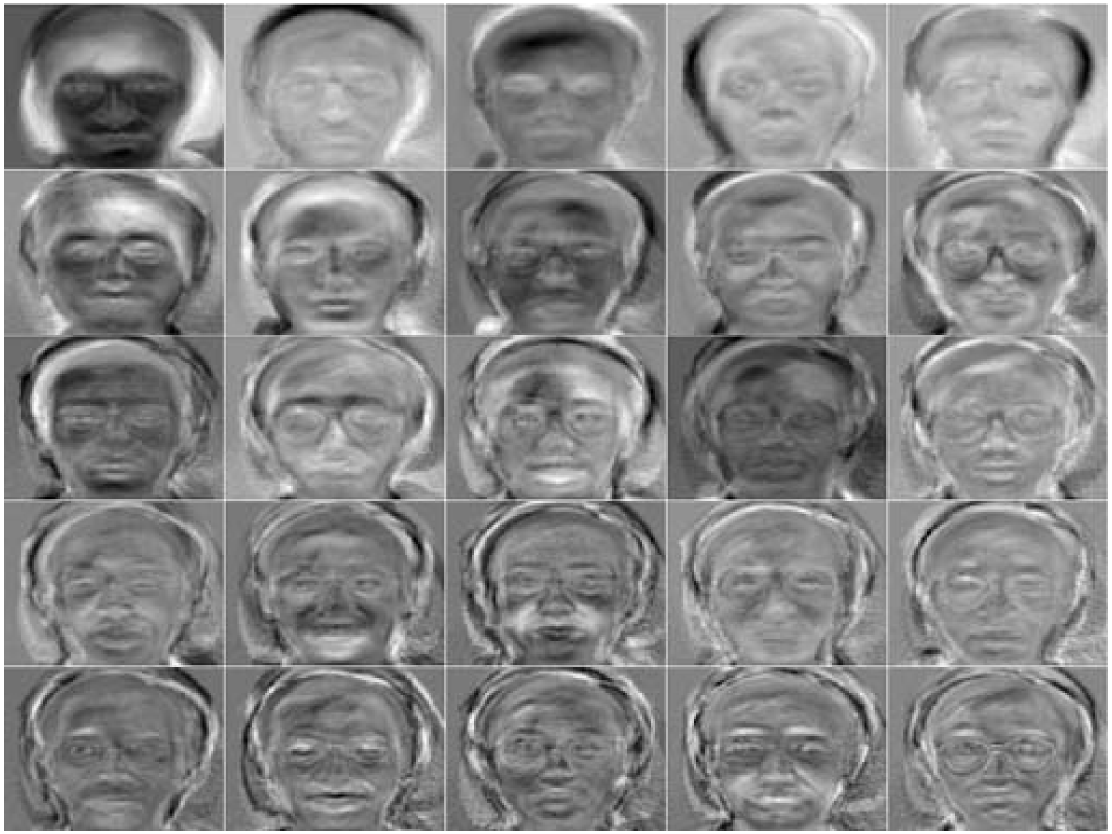
$$\mathbf{X}^T\mathbf{X}\ \mathbf{v} = \gamma\,\mathbf{v}$$

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

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

$$\mathbf{\Sigma}\,(\mathbf{Xv}) = \gamma\,(\mathbf{Xv})$$

Principle Components (Method B)



Reconstructing… (Method B)

- … faster if train with…
  - only people w/out glasses
  - same lighting conditions

Shortcomings

- Requires carefully controlled data:
  - All faces centered in frame
  - Same size
  - Some sensitivity to angle
- Alternative:
  - "Learn" one set of PCA vectors for each angle
  - Use the one with lowest error

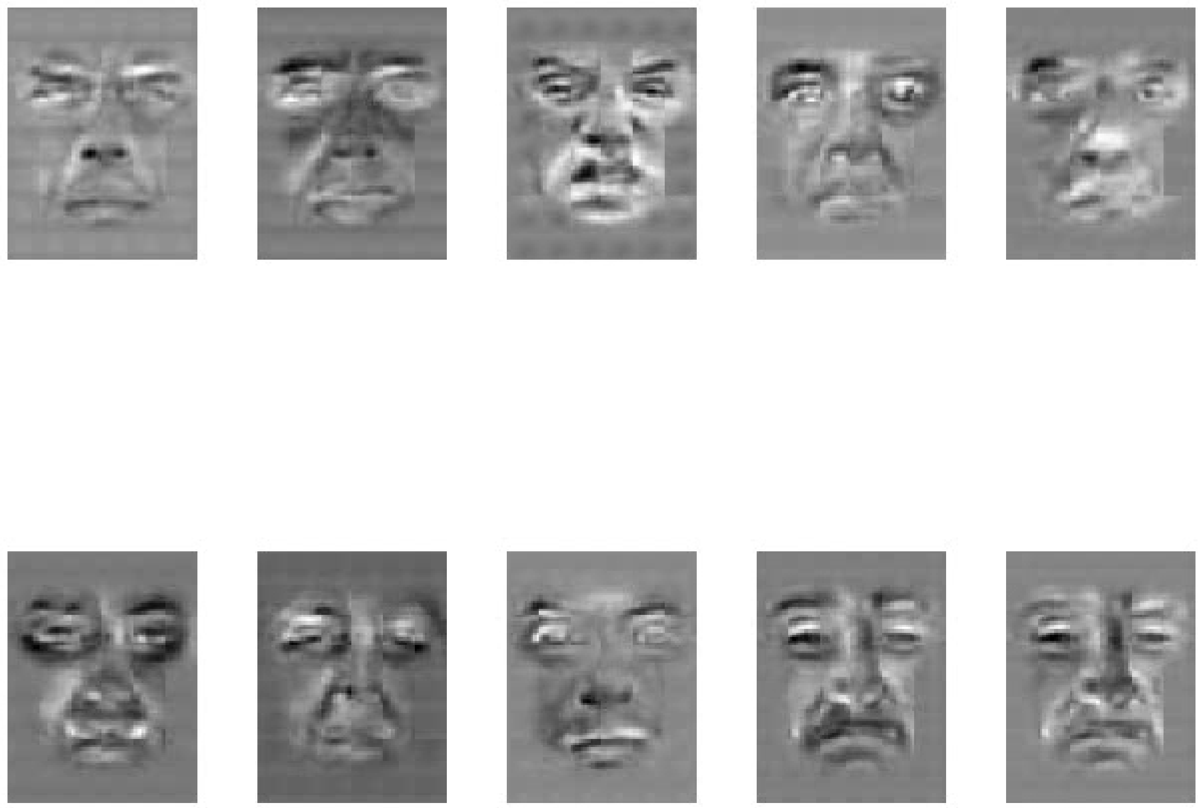
- Method is completely knowledge free
  - (sometimes this is good!)
  - Doesn't know that faces are wrapped around 3D objects (heads)
  - Makes no effort to preserve class distinctions

Facial expression recognition

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

Happiness subspace (method A)

Disgust subspace (method A)



Facial Expression Recognition Movies (method A)

Facial Expression Recognition Movies (method A)

Facial Expression Recognition Movies (method A)

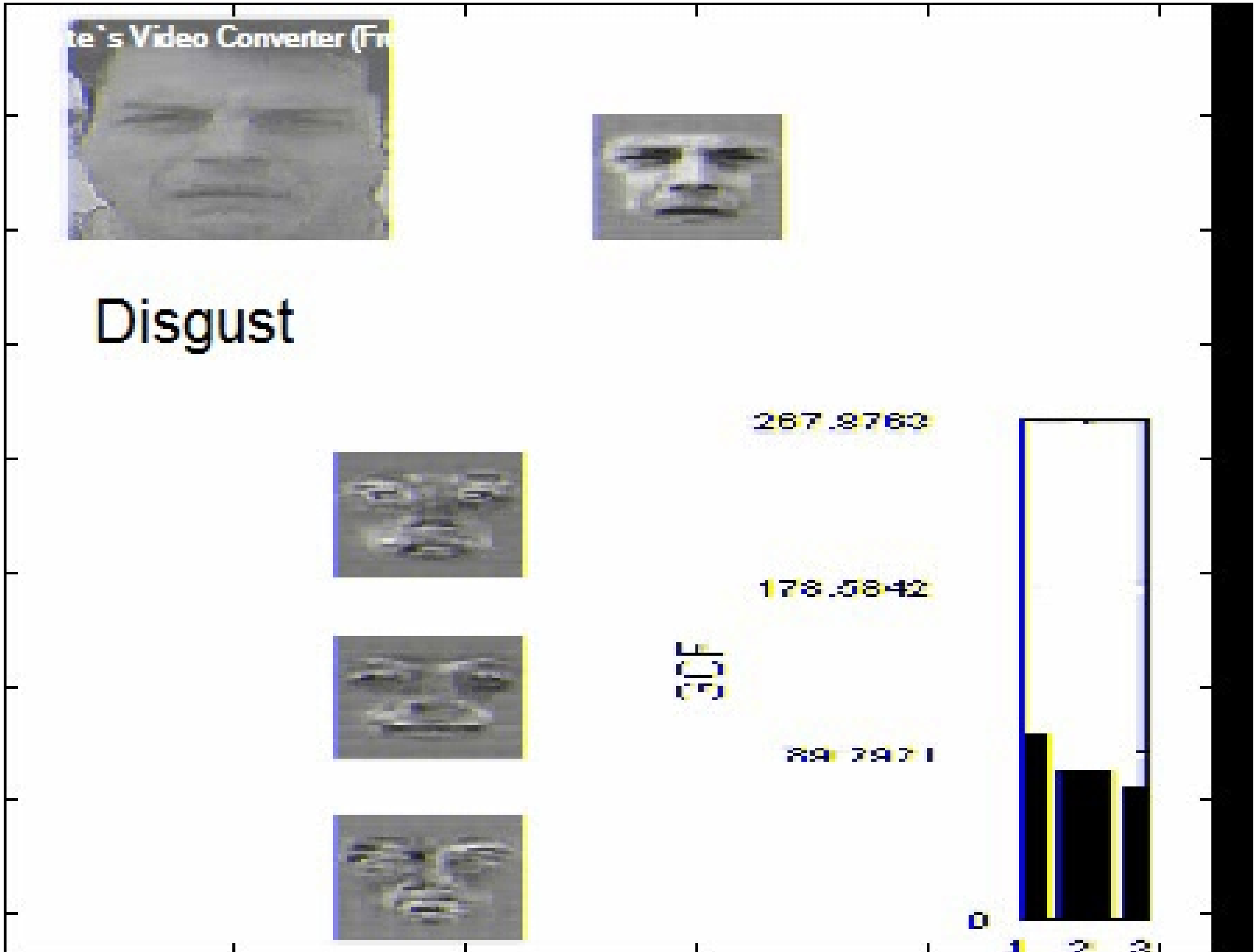
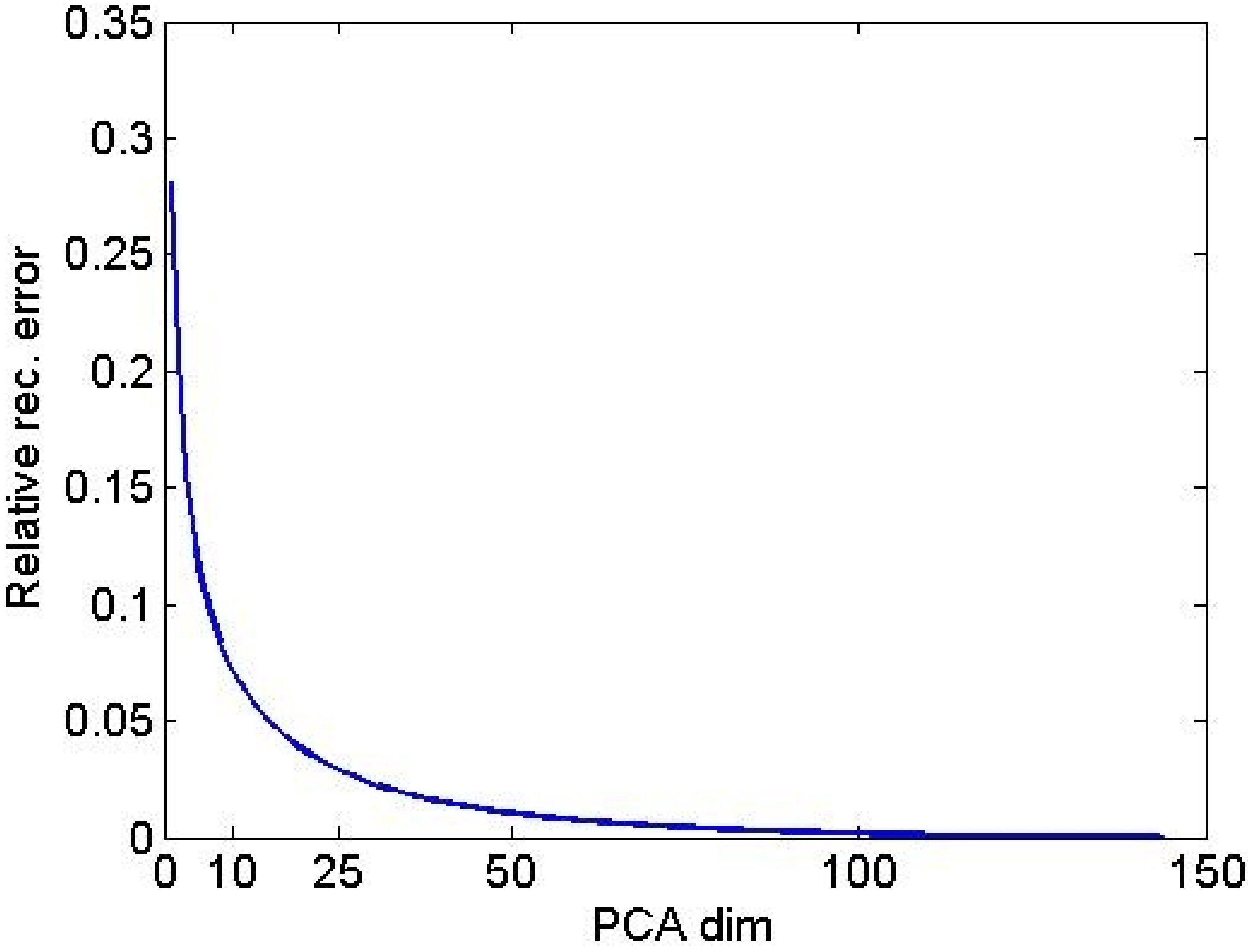


Image Compression

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

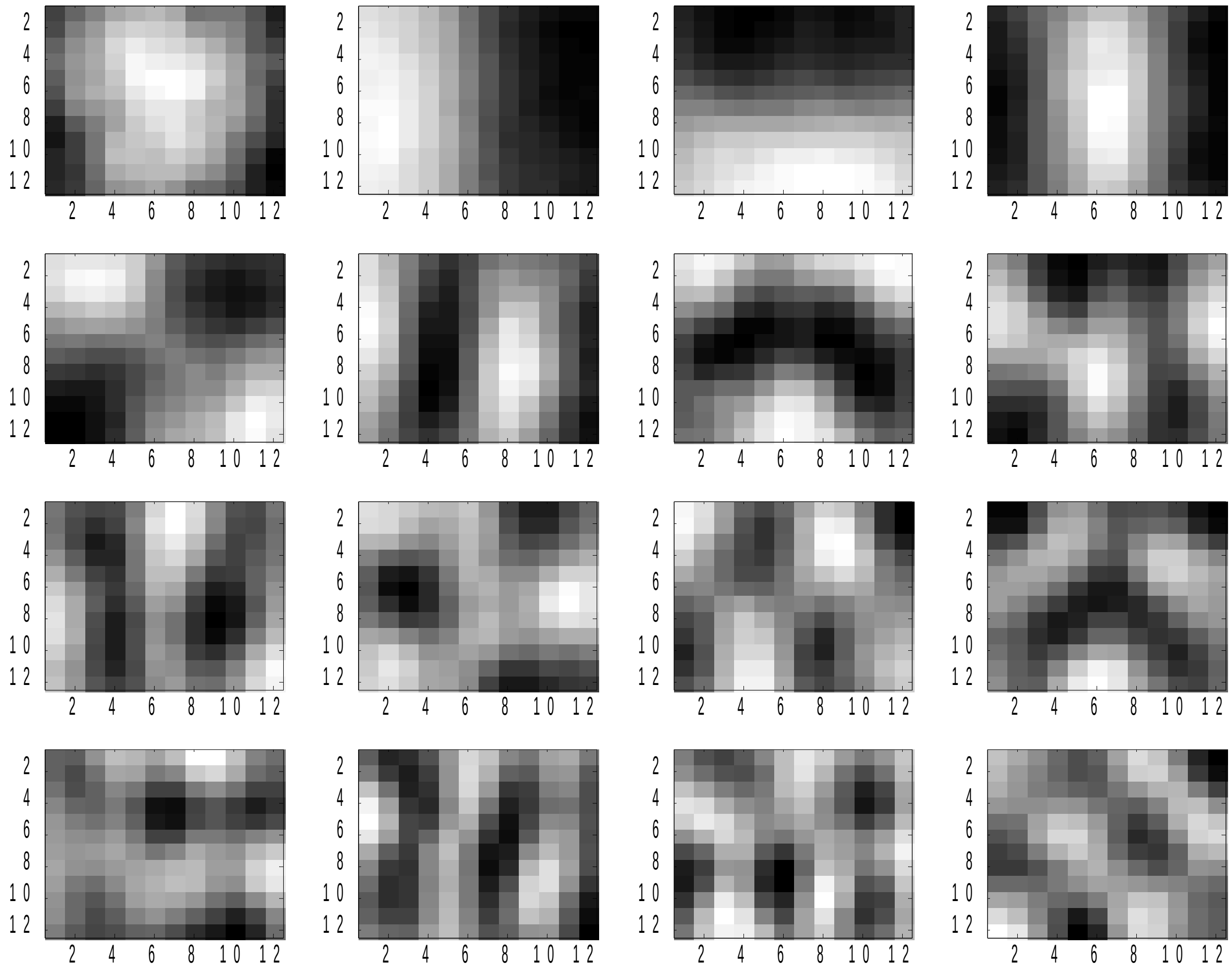
L_2 error and PCA dim



PCA compression: 144D $\Rightarrow$ 60D

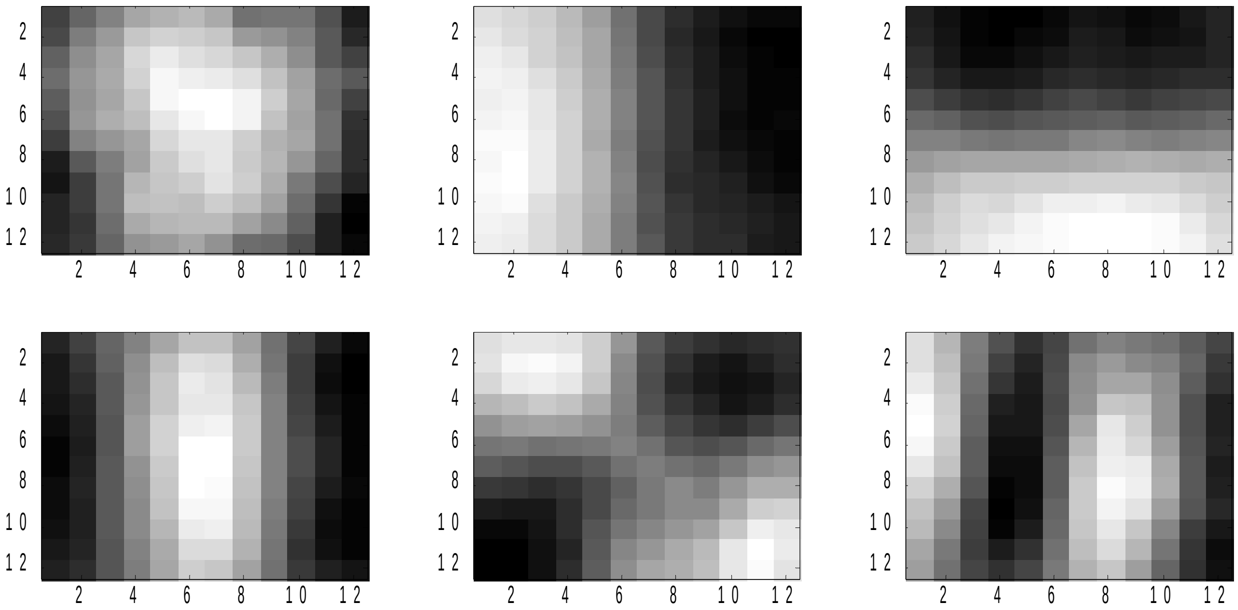
PCA compression: 144D $\Rightarrow$ 16D

16 most important eigenvectors



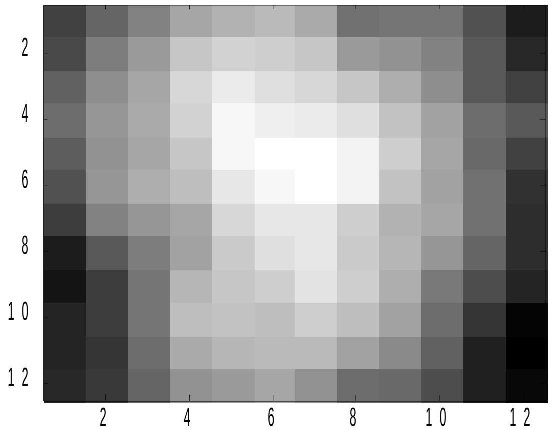
PCA compression: 144D $\Rightarrow$ 6D

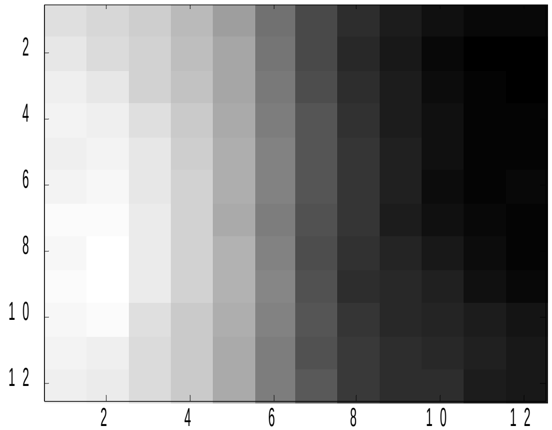
6 most important eigenvectors

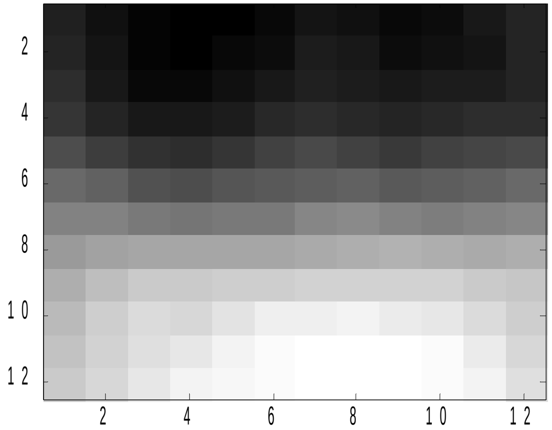


PCA compression: 144D $\Rightarrow$ 3D

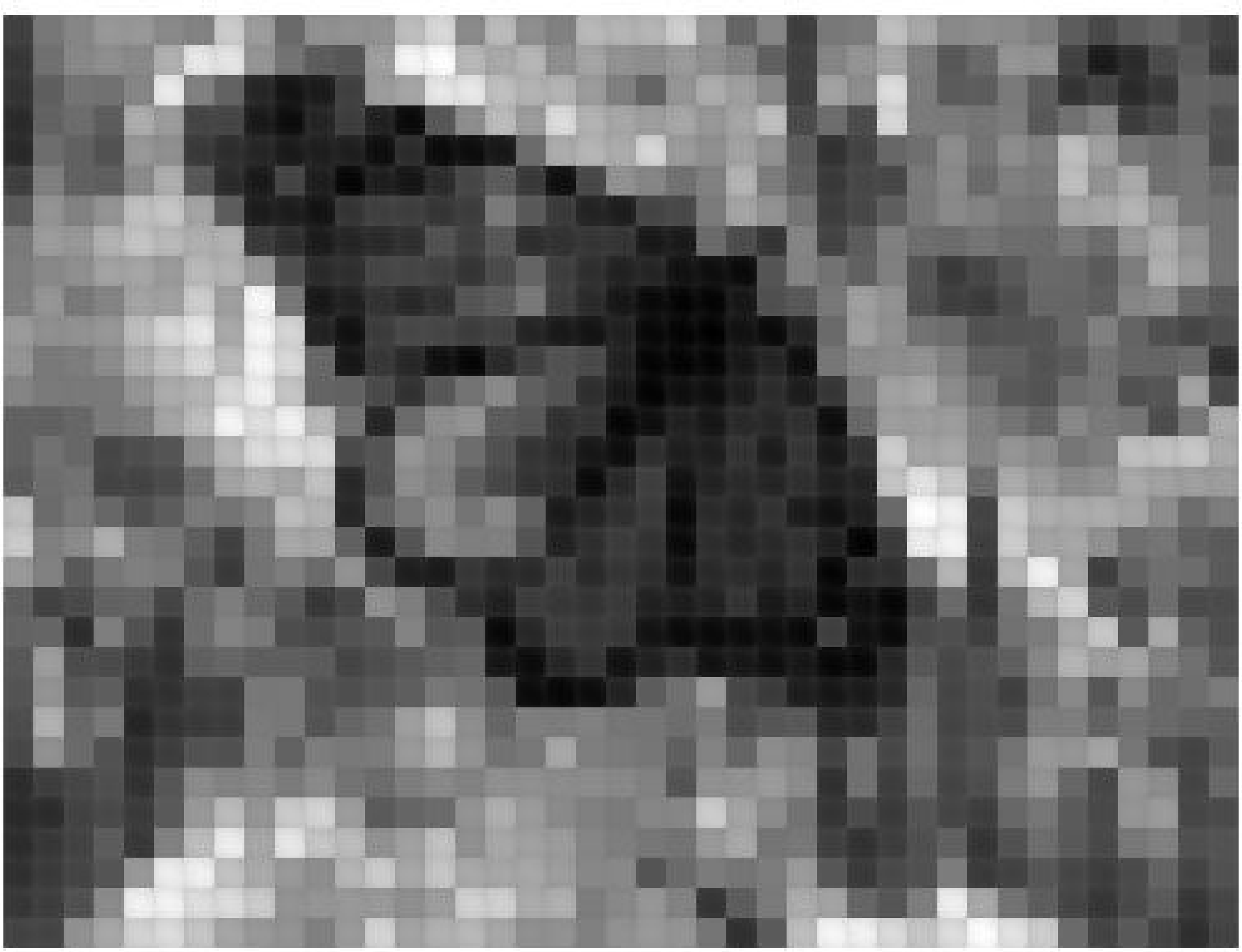
3 most important eigenvectors



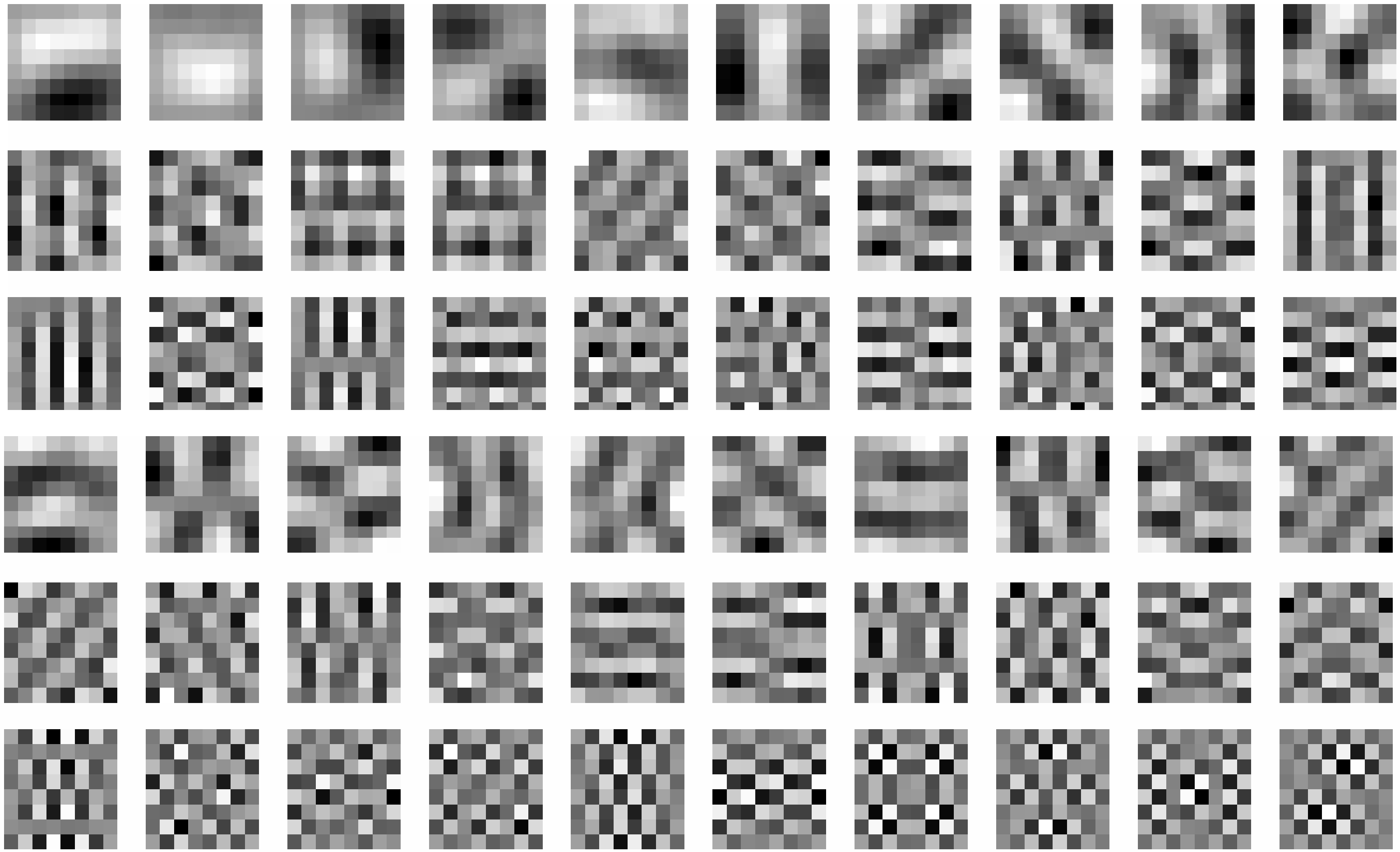




PCA compression: 144D ⇒ 1D

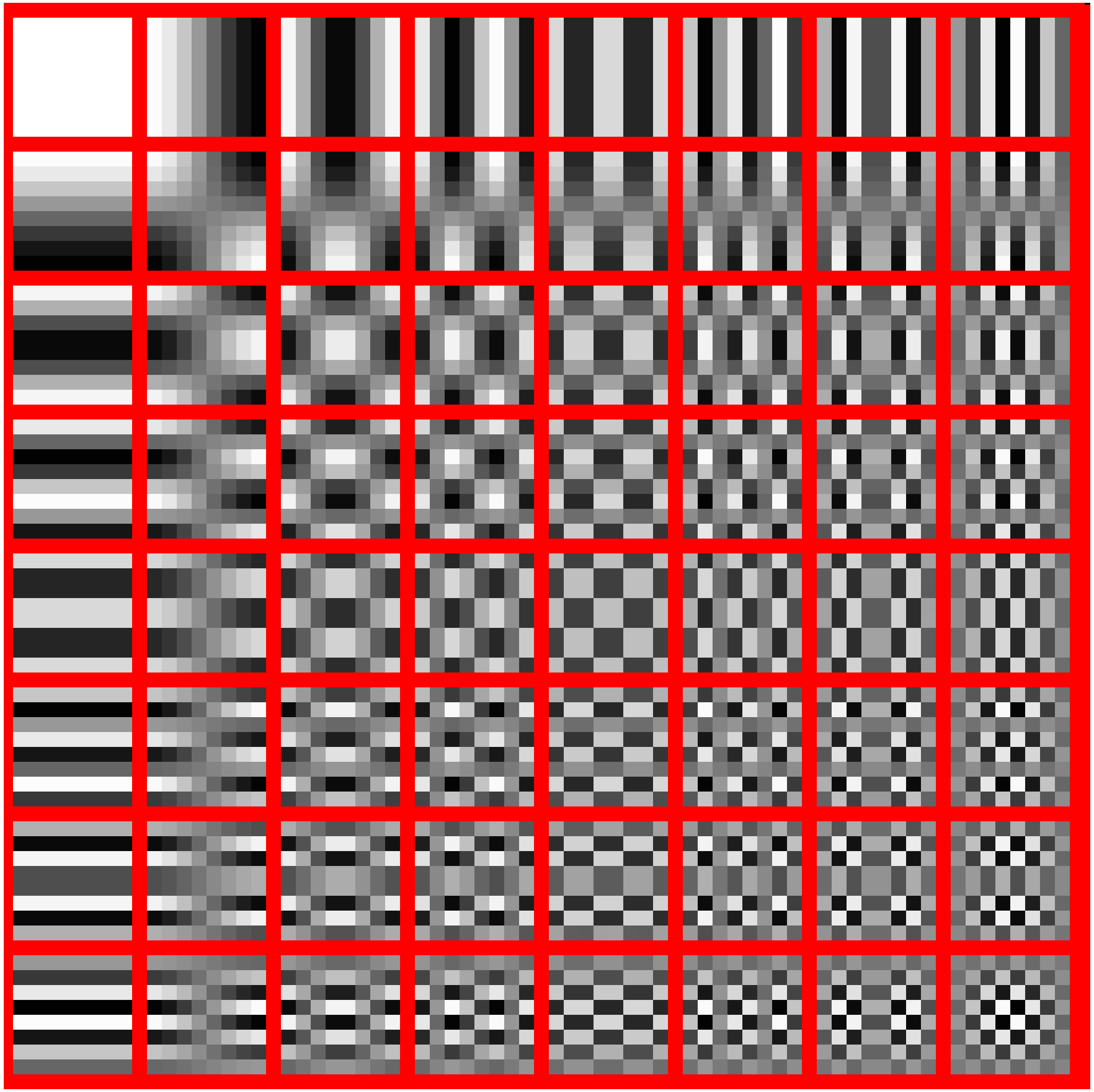


60 most important eigenvectors



Looks like the discrete cosine bases of JPG!...

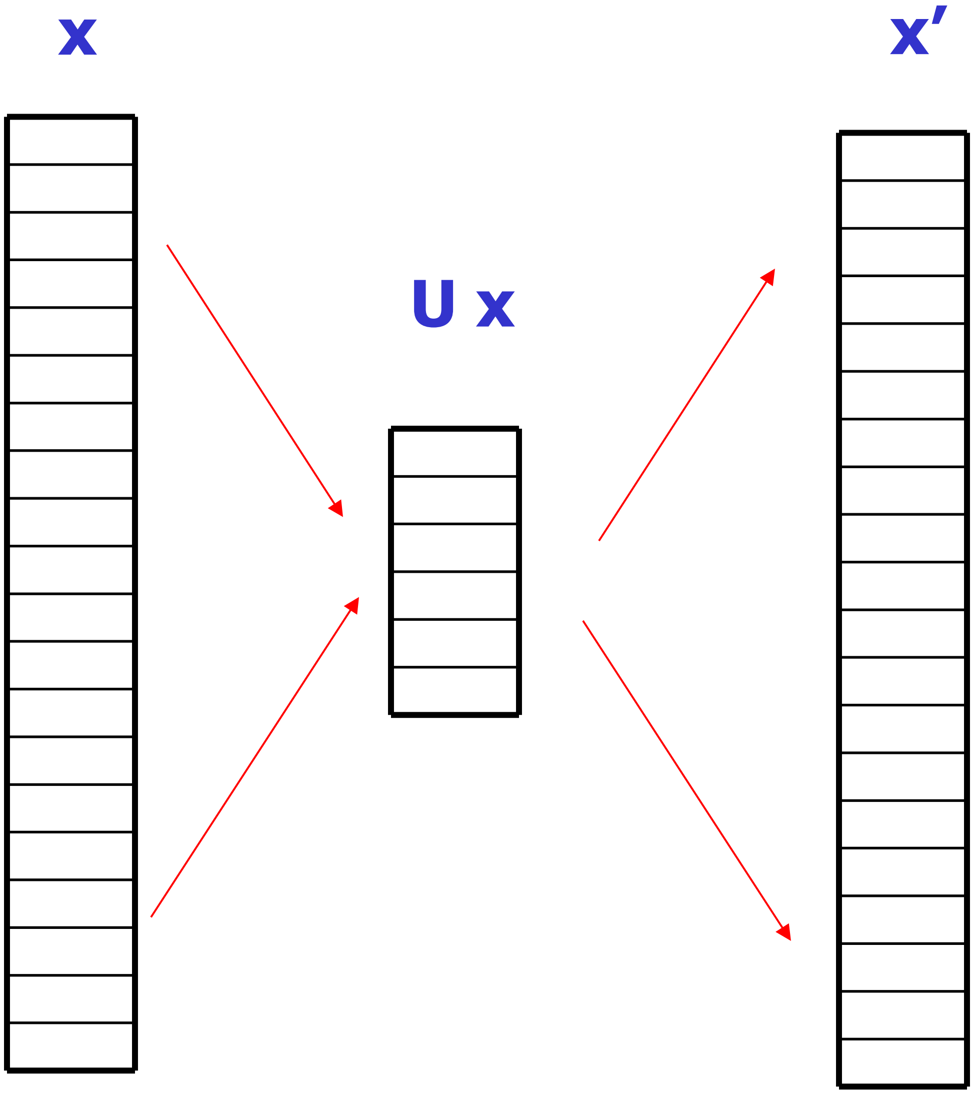
2D Discrete Cosine Basis



http://en.wikipedia.org/wiki/Discrete_cosine_transform

Noise Filtering

Noise Filtering, Auto-Encoder…

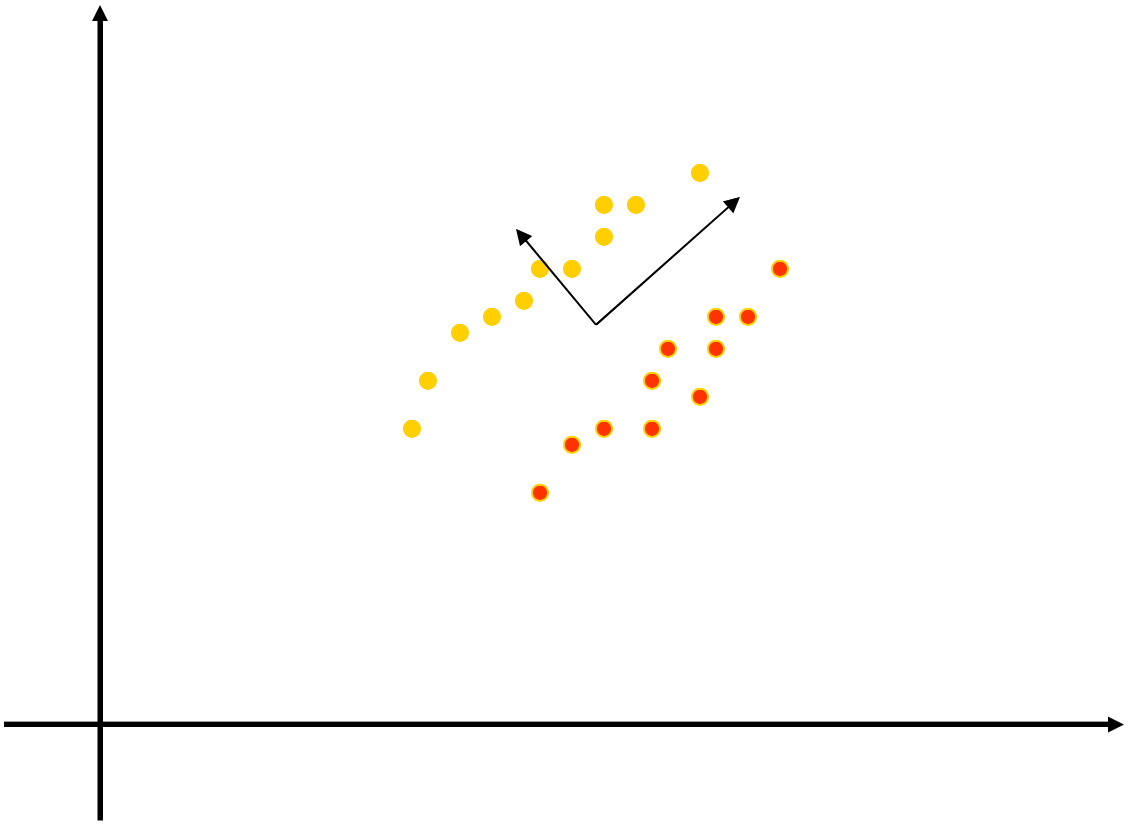


Noisy image

Denoised image using 15 PCA components

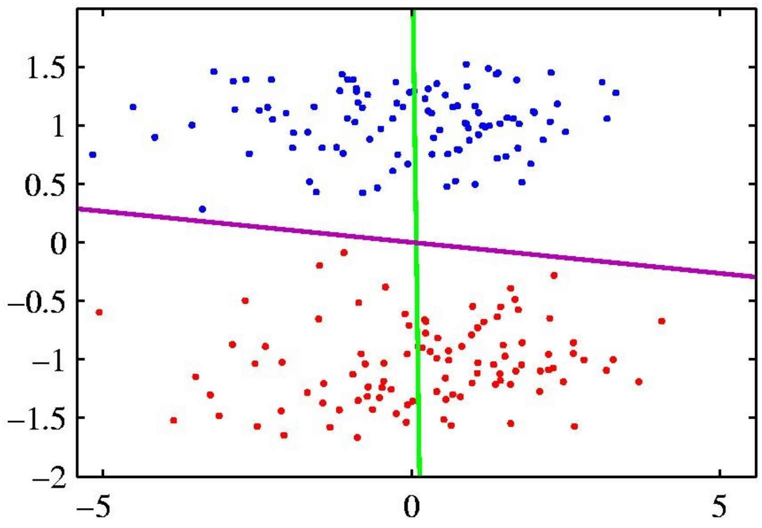
PCA Shortcomings

PCA, a Problematic Data Set



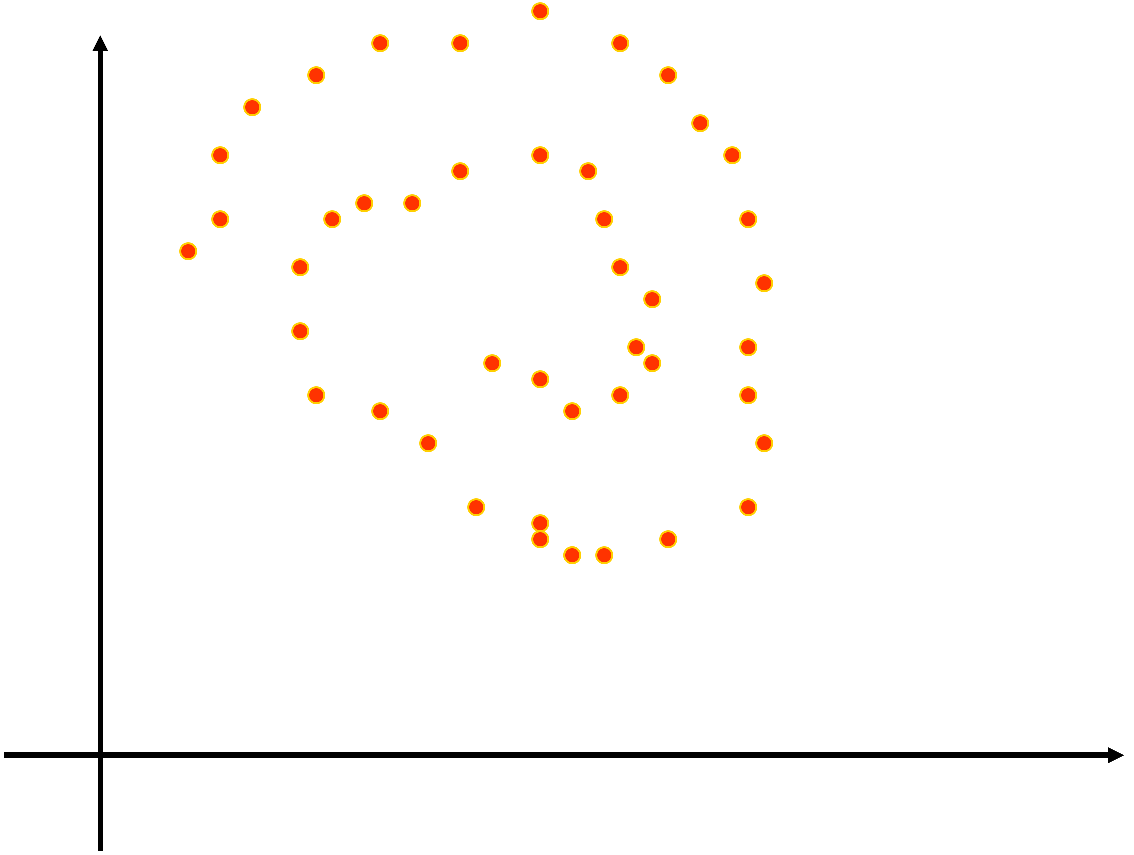
PCA doesn't know labels!

PCA vs Fisher Linear Discriminant



- PCA maximizes variance, *independent of class*
  $\Rightarrow$ magenta
- FLD attempts to separate classes
  $\Rightarrow$ green line

PCA, a Problematic Data Set



PCA cannot capture NON-LINEAR structure!

PCA Conclusions

- PCA
  - finds orthonormal basis for data
  - Sorts dimensions in order of “importance”
  - Discard low significance dimensions
- Uses:
  - Get compact description
  - Ignore noise
  - Improve classification (hopefully)
- Not magic:
  - Doesn’t know class labels
  - Can only capture linear variations
- One of many tricks to reduce dimensionality!

PCA Theory

Justification of Algorithm II

Let $\mathbf{x} \in \mathbb{R}^N$

$\boxed{\begin{array}{l}\text{Let } \mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{N \times m}, \\ m\text{: number of instances, } N\text{: dimension}\end{array}}$

Let $\mathbf{U} = \begin{pmatrix} \mathbf{u}_1^T \\ \vdots \\ \mathbf{u}_N^T \end{pmatrix} \in \mathbb{R}^{N \times N}$ orthogonal matrix, $\mathbf{U}\mathbf{U}^T = \mathbf{I}_N$

$$\mathbf{y} \doteq \mathbf{U}\mathbf{x}, \ \mathbf{x} = \mathbf{U}^T\mathbf{y} = \sum_{i=1}^{N} \mathbf{u}_i y_i$$

$\hat{\mathbf{x}} \doteq \sum_{i=1}^{M} \mathbf{u}_i y_i, \ (M \leq N)$ approximation of $\mathbf{x}$ using M basis vectors only.

$\varepsilon^2 \doteq \mathbb{E}\{\|\mathbf{x} - \hat{\mathbf{x}}\|^2\} = \frac{1}{m} \sum_{j=1}^{m} \|\mathbf{x}_j - \hat{\mathbf{x}}_j\|$, average error

GOAL:

$$\boxed{\arg\min_{\mathbf{U}} \varepsilon^2, \text{ s.t } \mathbf{U}^T\mathbf{U} = \mathbf{I}_N}$$

Justification of Algorithm II

$$
\begin{aligned}
\varepsilon^2 &= \mathbb{E}\{\|\mathbf{x}-\hat{\mathbf{x}}\|^2\} = \mathbb{E}\{\|\sum_{i=1}^{N}\mathbf{u}_i y_i - \sum_{i=1}^{M}\mathbf{u}_i y_i\|^2\} \\
&= \mathbb{E}\{\sum_{i=M+1}^{N} y_i \mathbf{u}_i^T \mathbf{u}_i y_i\} = \sum_{i=M+1}^{N} \mathbb{E}\{y_i^2\} \\
&= \sum_{i=M+1}^{N} \mathbb{E}\{(\mathbf{u}_i^T \mathbf{x})(\mathbf{x}^T \mathbf{u}_i)\} \\
&= \sum_{i=M+1}^{N} \mathbf{u}_i^T \mathbb{E}\{\mathbf{x}\mathbf{x}^T\}\mathbf{u}_i \\
&= \sum_{i=M+1}^{N} \mathbf{u}_i^T \boldsymbol{\Sigma} \mathbf{u}_i
\end{aligned}
$$

x is centered!

Justification of Algorithm II

GOAL: $\arg \min_{\mathbf{u}_{M+1},\ldots,\mathbf{u}_N} \varepsilon^2$

Use Lagrange-multipliers for the constraints.

$$
\begin{aligned}
L &= \varepsilon^2 - \sum_{i=M+1}^{N} \lambda_i(\mathbf{u}_i^T \mathbf{u}_i - 1) \\
&= \sum_{i=M+1}^{N} \mathbf{u}_i^T \mathbf{\Sigma} \mathbf{u}_i - \sum_{i=M+1}^{N} \lambda_i(\mathbf{u}_i^T \mathbf{u}_i - 1)
\end{aligned}
$$

$$
\frac{\partial L}{\partial \mathbf{u}_i} = [2\mathbf{\Sigma}\mathbf{u}_i - 2\lambda_i \mathbf{u}_i] = 0
$$

Justification of Algorithm II

$$\frac{\partial L}{\partial \mathbf{u}_i} = [2\boldsymbol{\Sigma}\mathbf{u}_i - 2\lambda_i\mathbf{u}_i] = 0 \Rightarrow \boldsymbol{\Sigma}\mathbf{u}_i = \lambda_i\mathbf{u}_i$$

$$\Rightarrow [\mathbf{u}_i, \lambda_i] = \text{eigenvector/eigenvalue of } \boldsymbol{\Sigma}.$$

$$\boxed{\varepsilon^2 = \sum_{i=M+1}^{N} \mathbf{u}_i^T \boldsymbol{\Sigma} \mathbf{u}_i = \sum_{i=M+1}^{N} \mathbf{u}_i^T \lambda_i \mathbf{u}_i = \sum_{i=M+1}^{N} \lambda_i}$$

The error ε^2 is minimal
if $\lambda_{M+1}, \ldots \lambda_N$ are the smallest eigenvalues of $\boldsymbol{\Sigma}$,
and $\mathbf{u}_{M+1}, \ldots, \mathbf{u}_N$ are the corresponding eigenvectors.

Kernel PCA

Kernel PCA

Performing PCA in the feature space

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{N \times m}$,
m: number of instances, N: dimension

Lemma

$\mathbf{u}$ is eigenvector of $\boldsymbol{\Sigma} \Rightarrow \mathbf{u}$ is a linear combinaton of the samples

Proof:

$$\lambda \mathbf{u} = \boldsymbol{\Sigma} \mathbf{u} = \left(\frac{1}{m} \sum_{i=1}^{m} \mathbf{x}_i \mathbf{x}_i^T \right) \mathbf{u} = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i^T \mathbf{u}) \mathbf{x}_i$$

$$\Rightarrow \mathbf{u} = \sum_{i=1}^{m} \underbrace{\frac{(\mathbf{x}_i^T \mathbf{u})}{\lambda m}}_{\alpha_i} \mathbf{x}_i = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i$$

Kernel PCA

$$\mathbf{u} = \sum_{i=1}^{m} \underbrace{\frac{(\mathbf{x}_i^T \mathbf{u})}{\lambda m}}_{\alpha_i} \mathbf{x}_i = \sum_{i=1}^{m} \alpha_i \mathbf{x}_i \qquad \mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_m] \in \mathbb{R}^{N \times m},$$

Lemma

To calculate $\boldsymbol{\alpha} \in \mathbb{R}^m$

- just use inner products (Gram matrix): $K_{ij} = \mathbf{x}_i^T \mathbf{x}_j$
- don't need the actual values of $\mathbf{x}_i$

Kernel PCA

Proof

$$\mathbf{\Sigma u} = \lambda \mathbf{u}, \ \ \mathbf{u} = \sum_{j=1}^{m} \alpha_j \mathbf{x}_j$$

$$\Rightarrow \mathbf{x}_i^T \mathbf{\Sigma u} = \lambda \mathbf{x}_i^T \mathbf{u}$$

$$\Rightarrow \mathbf{x}_i^T \left(\frac{1}{m} \sum_{k=1}^{m} \mathbf{x}_k \mathbf{x}_k^T \right) \left(\sum_{j=1}^{m} \alpha_j \mathbf{x}_j \right) = \lambda \mathbf{x}_i^T \left(\sum_{j=1}^{m} \alpha_j \mathbf{x}_j \right)$$

$$\Rightarrow \frac{1}{m} \sum_{k=1}^{m} \sum_{j=1}^{m} (\mathbf{x}_i^T \mathbf{x}_k)(\mathbf{x}_k^T \mathbf{x}_j) \alpha_j = \lambda \sum_{j=1}^{m} (\mathbf{x}_i^T \mathbf{x}_j) \alpha_j$$

$$\Rightarrow \frac{1}{m} \mathbf{K}^2 \boldsymbol{\alpha} = \lambda \mathbf{K} \boldsymbol{\alpha} \quad \text{where } \mathbf{K} \in \mathbb{R}^{m \times m}$$

$$\Rightarrow \mathbf{K} \boldsymbol{\alpha} = m \lambda \boldsymbol{\alpha} \qquad \text{If } \mathbf{K} \text{ is invertible (strictly pos def)}$$

Kernel PCA

- How to use α to calculate the projection of a new sample *t?*

$$\mathbf{u}^T\mathbf{t} = (\sum_{j=1}^{m} \alpha_j \mathbf{x}_j)^T \mathbf{t} = \sum_{j=1}^{m} \alpha_j K(\mathbf{x}_j, \mathbf{t})$$

Again, we don't need values of $\mathbf{x}_j$!

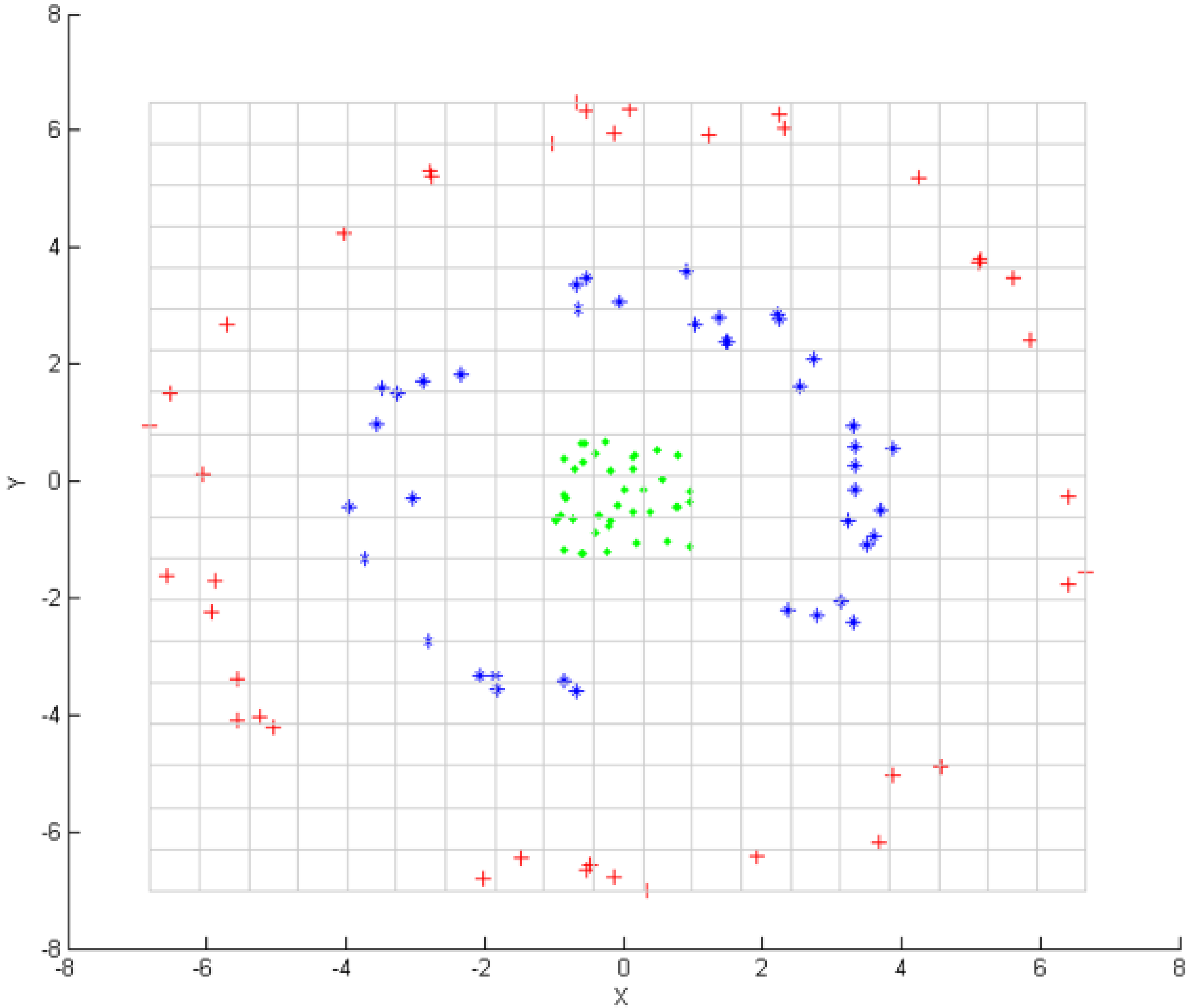
Let $K_{i,j} = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$

Where was I cheating? ☺

The data should be centered in the feature space, too!
But this is manageable...

$$\tilde{K}_{i,j} \doteq \left\langle \phi(\mathbf{x}_i) - \frac{1}{m}\sum_{k=1}^{m} \phi(\mathbf{x}_k), \quad \phi(\mathbf{x}_j) - \frac{1}{m}\sum_{k=1}^{m} \phi(\mathbf{x}_k) \right\rangle$$

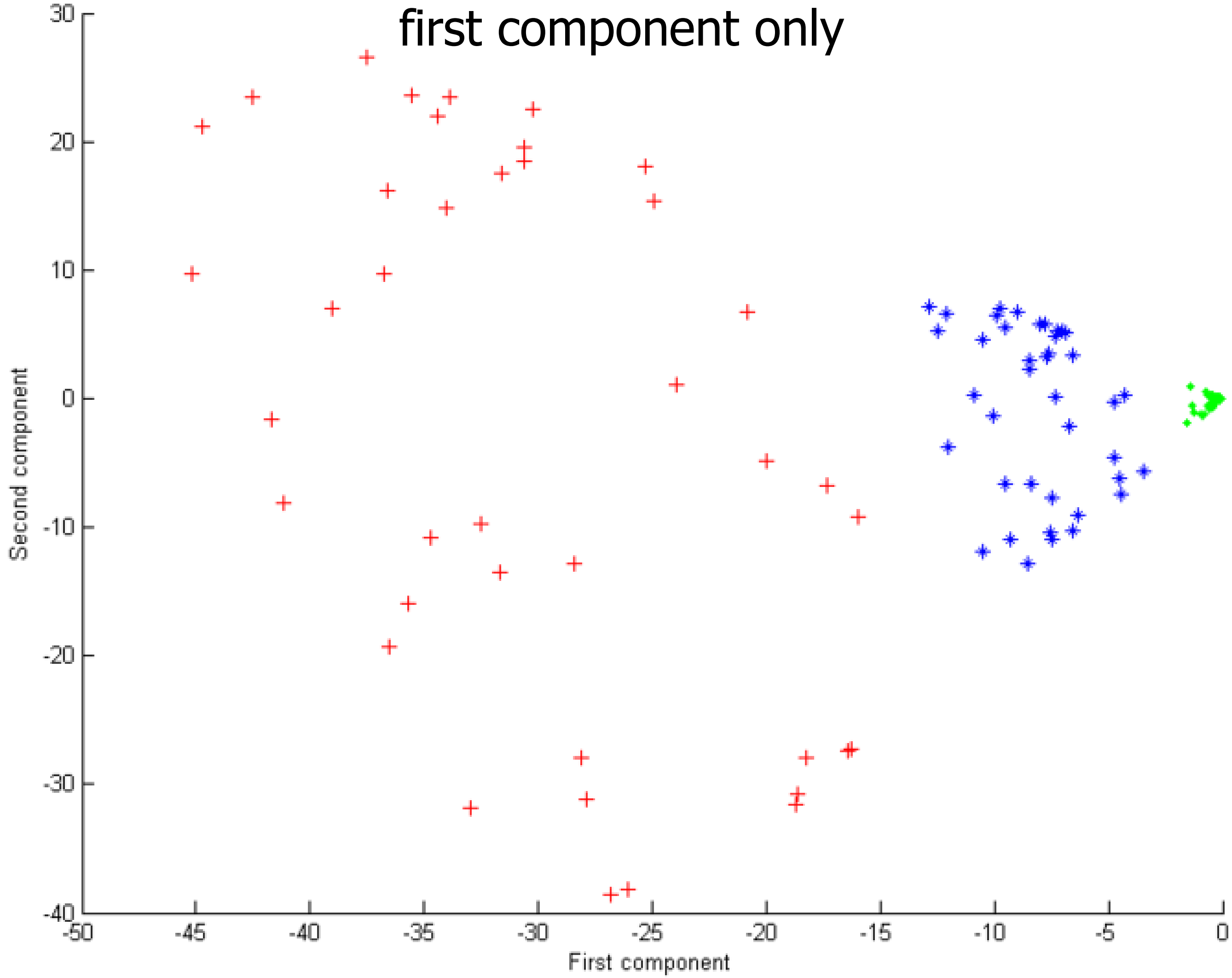
Input points before kernel PCA



http://en.wikipedia.org/wiki/Kernel_principal_component_analysis

Output after kernel PCA

The three groups are distinguishable using the first component only



We haven't covered…

- Artificial Neural Network Implementations
- Mixture of Probabilistic PCA
- Online PCA, Regret Bounds

Thanks for the Attention! ☺

RL & AI

ALBERTA INGENUITY CENTRE FOR MACHINE LEARNING

UNIVERSITY OF ALBERTA QUAECUMQUE VERA