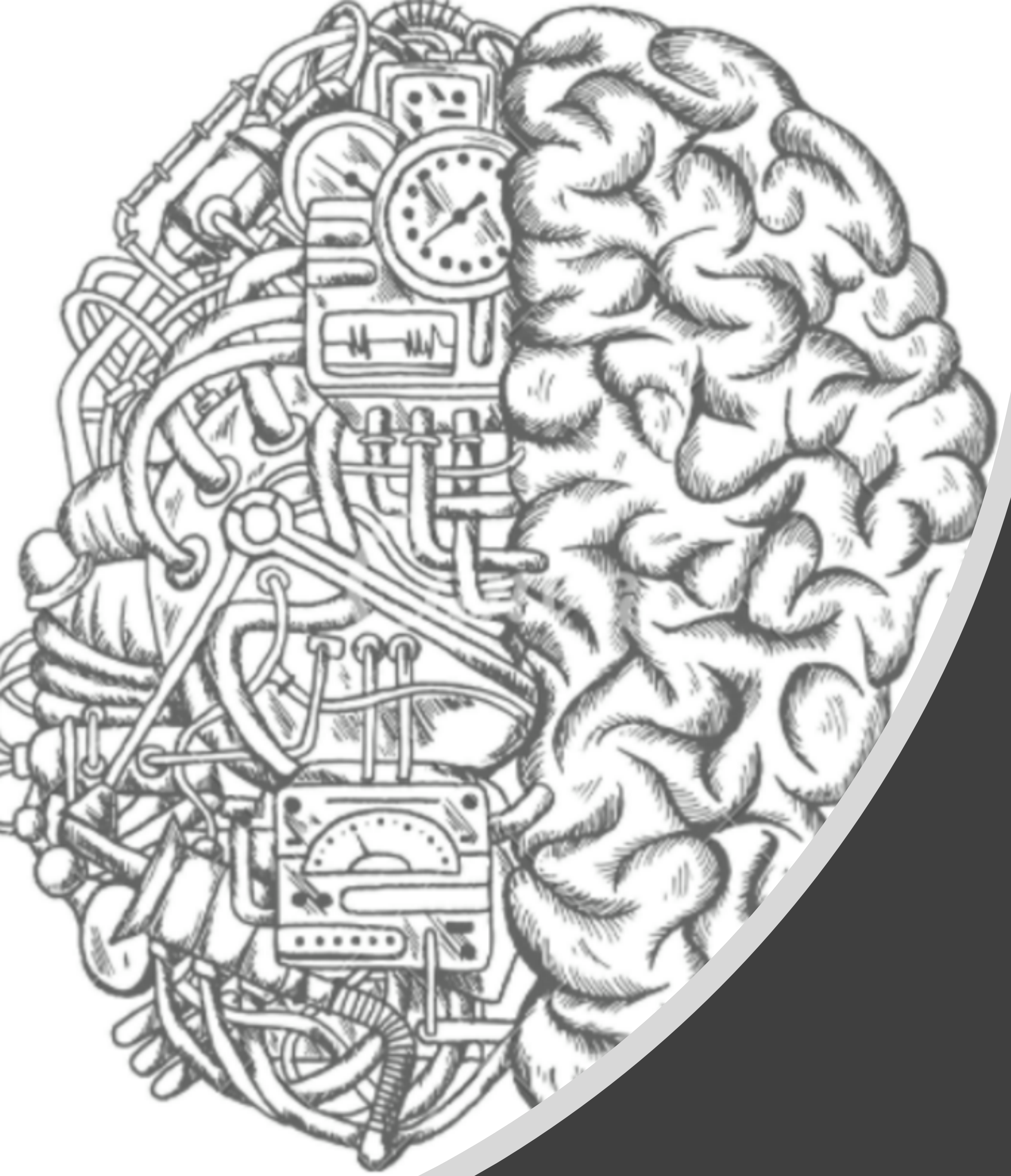


CS 559: Machine Learning Fundamentals and Applications

Lecture 5

Lecturer: Xinchao Wang

xinchao.wang@stevens.edu

Teaching Assistant: Yiding Yang

yyang99@stevens.edu

Overview

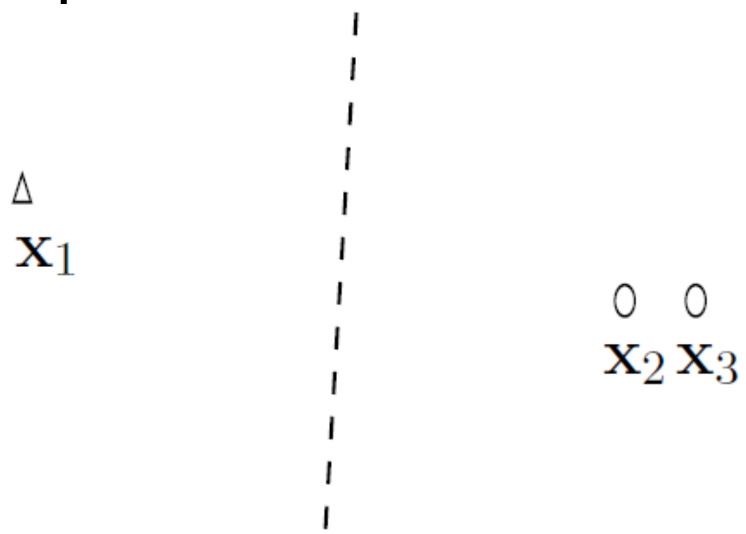
- A note on data normalization/scaling
- Principal Component Analysis (notes)
    - Intro
    - Singular Value Decomposition
- Dimensionality Reduction - PCA in practice (Notes based on Carlos Guestrin's)
- Eigenfaces (notes by Srinivasa Narasimhan, CMU)

Data Scaling

- Without scaling, attributes in greater numeric ranges may dominate
- Example: compare people using annual income (in dollars) and age (in years)

Data Scaling

- The separating hyperplane



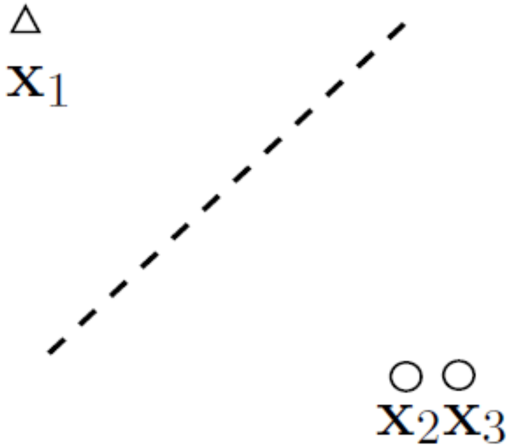
- Decision strongly depends on the first attribute
- What if the second is (more) important?

Data Scaling

- Linearly scale features to [0, 1] interval using min and max values.
- Divide each feature by its standard deviation

Data Scaling

- New points and separating hyperplane



- The second attribute plays a role

Data Scaling

- Distance/similarity measure must be meaningful in feature space
  - This applies to most classifiers (not random forests)
- Normalized Euclidean distance

$$d(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{p} \frac{(x_i - y_i)^2}{\sigma_i^2}},$$

- Mahalanobis distance

$$d(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})}.$$

  - Where S is the covariance matrix of the data

Mahalanobis Distance

- Generalized as distance between two points
- Takes into account correlations in data

Principal Component Analysis (PCA)

PCA Resources

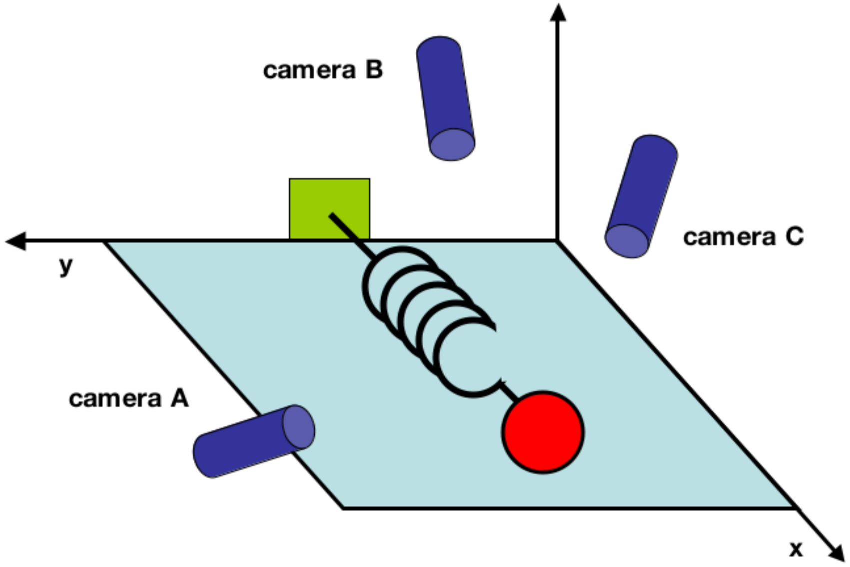
- A Tutorial on Principal Component Analysis
  - by Jonathon Shlens (Google Research), 2014
  - http://arxiv.org/pdf/1404.1100.pdf
- Singular Value Decomposition Tutorial
  - by Michael Elad (Technion, Israel), 2005
  - http://webcourse.cs.technion.ac.il/234299/Spring2005/ho/WCFiles/Tutorial7.ppt
- Dimensionality Reduction (lecture notes)
  - by Carlos Guestrin (CMU, now at UW), 2006
  - http://www.cs.cmu.edu/~guestrin/Class/10701-S06/Slides/tsvms-pca.pdf

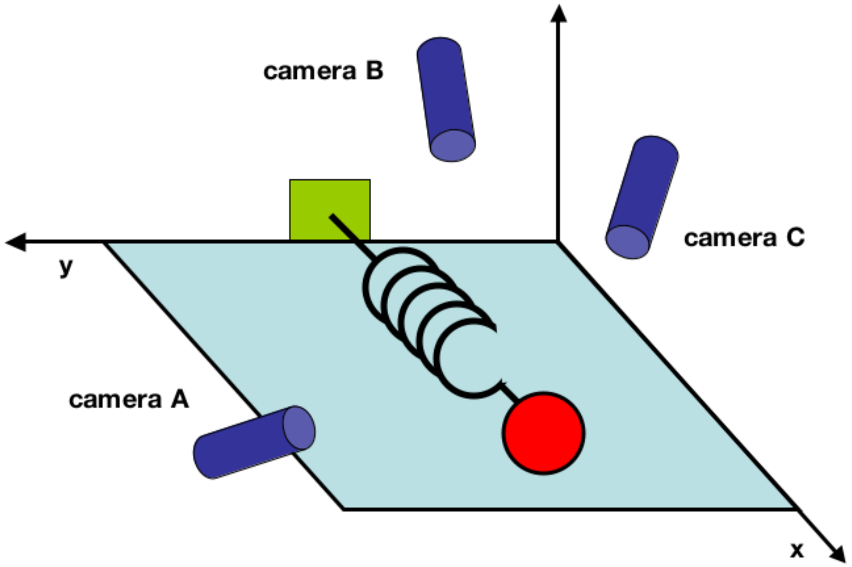
A Tutorial on Principal Component Analysis

Jonathon Shlens

A Toy Problem

- Ball of mass *m* attached to massless, frictionless spring
- Ball moved away from equilibrium results in spring oscillating indefinitely along *x*-axis
- All dynamics are a function of a single variable *x*





- We do not know which or how many axes and dimensions are important to measure
- Place three video cameras that capture 2-D measurements at 120Hz
    - Camera optical axes are not orthogonal to each other
- If we knew what we need to measure, one camera measuring displacement along *x* would be sufficient

Goal of PCA

- Compute the most meaningful basis to re-express a noisy data set
- Hope that this new basis will filter out the noise and reveal hidden structure

- In toy example:
  - Determine that the dynamics are along ***a single axis***
  - Determine the ***important axis***

Naïve Basis

- At each second, record 2 coordinates of ball position in each of the 3 images

$$\vec{X} = \begin{bmatrix} x_A \\ y_A \\ x_B \\ y_B \\ x_C \\ y_C \end{bmatrix}$$

- After 10 minutes at 120Hz, we have 10×60×120=7200 6D vectors
- These vectors can be represented in ***arbitrary coordinate systems***
- Naïve basis is formed by the image axis
  - Reflects the method which gathered the data

Change of Basis

- PCA: Is there another basis, which is a linear combination of the original basis, that best re-expresses our data set?
- Assumption: *linearity*
    - Restricts set of potential bases
    - Implicitly assumes continuity in data

Change of Basis

- **X** is original data (m×n, m=6, n=7200)
- Let **Y** be another m×n matrix such that **Y=PX**
- **P** is a matrix that transforms **X** into **Y**
  - Geometrically it is a rotation and stretch
  - The rows of **P** $\{p_1, \ldots, p_m\}$ are the new basis vectors for the columns of **X**
  - Each element of y_i is a dot product of x_i with the corresponding row of **P** (a projection of x_i onto p_j)

$$\mathbf{PX} = \begin{bmatrix} \mathbf{p_1} \\ \vdots \\ \mathbf{p_m} \end{bmatrix} \begin{bmatrix} \mathbf{x_1} & \cdots & \mathbf{x_n} \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{p_1} \cdot \mathbf{x_1} & \cdots & \mathbf{p_1} \cdot \mathbf{x_n} \\ \vdots & \ddots & \vdots \\ \mathbf{p_m} \cdot \mathbf{x_1} & \cdots & \mathbf{p_m} \cdot \mathbf{x_n} \end{bmatrix}$$

$$\mathbf{y}_i = \begin{bmatrix} \mathbf{p_1} \cdot \mathbf{x_i} \\ \vdots \\ \mathbf{p_m} \cdot \mathbf{x_i} \end{bmatrix}$$

How to find an Appropriate Change of Basis?

- The row vectors $\{p_1, \ldots, p_m\}$ will become the *principal components* of **X**
- What is the best way to re-express **X**?
- What features would we like **Y** to exhibit?

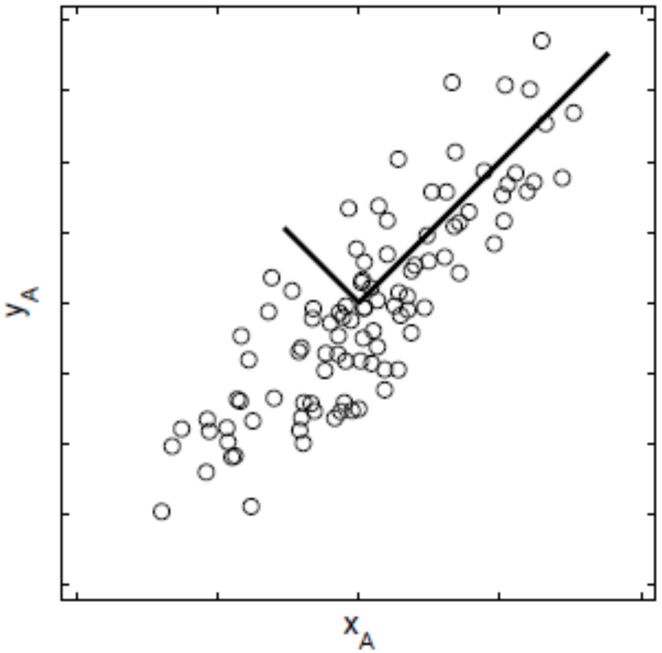
- If we call **X** "garbled data", garbling in a linear system can refer to three things:
  - Noise
  - Rotation
  - Redundancy

Noise

- Measurement noise in any data set must be low or else, no matter the analysis technique, no information about a system can be extracted
- Signal-to-Noise Ratio (SNR)

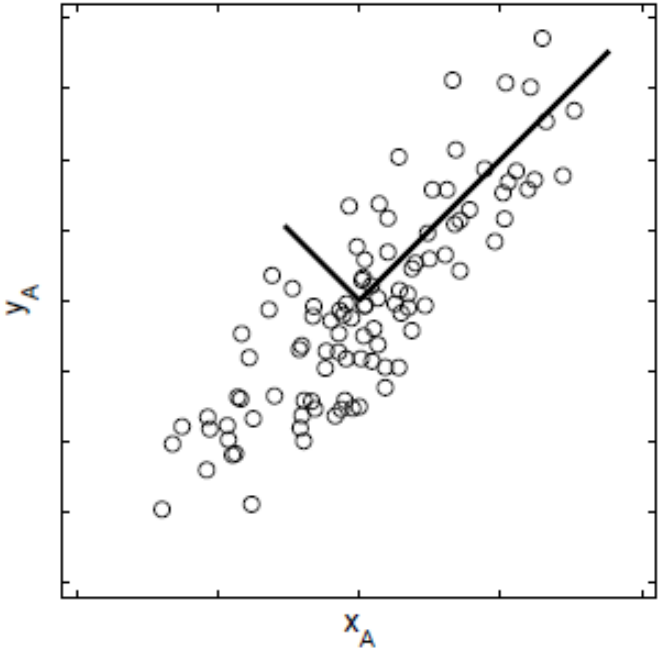
$$SNR = \frac{\sigma^2_{signal}}{\sigma^2_{noise}}$$

Noise



- Ball travels in straight line
  - Any deviation must be noise
- Variance due to signal and noise are indicated in diagram
- SNR: ratio of the two lengths
  - "Fatness" of data corresponds to noise
- Assumption: directions of largest variance in measurement vector space contain dynamics of interest

Rotation

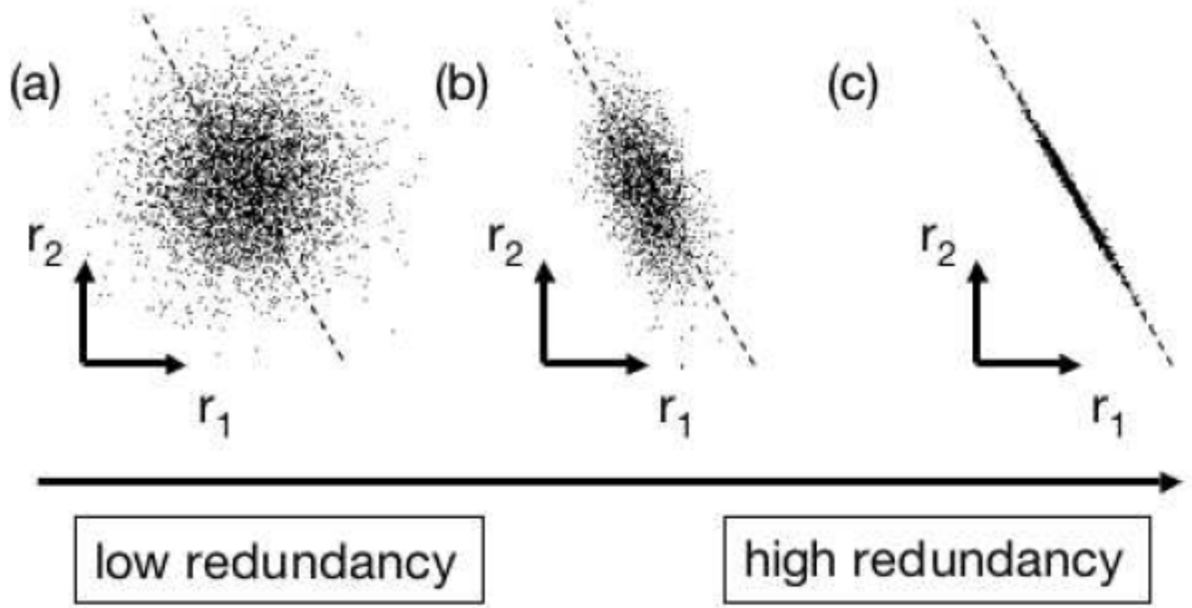


- Neither x_A, nor y_A however are directions with maximum variance
- Maximizing the variance corresponds to finding the appropriate rotation of the naive basis
- In 2D this is equivalent to finding best fitting line

Redundancy

- Is it necessary to record 2 variables for the ball-spring system?
- Is it necessary to use 3 cameras?

Redundancy spectrum for 2 variables



Covariance Matrix

- Assume zero-mean measurements
  - Subtract mean from all vectors in **X**
- Each column of **X** is a set of measurements at a point in time
- Each row of **X** corresponds to all measurements of a particular type (e.g. x-coordinate in image B)
- Covariance matrix $\mathbf{C}_X = \mathbf{X}\mathbf{X}^T$
- ij^{th} element of $\mathbf{C}_X$ is the dot product between the i^{th} measurement type and the j^{th} measurement type
  - Covariance between two measurement types

Covariance Matrix

- Diagonal elements of $\mathbf{C}_X$
  - Large → interesting dynamics
  - Small → noise
- Off-diagonal elements of $\mathbf{C}_X$
  - Large → high redundancy
  - Small → low redundancy
- We wish to maximize signal and minimize redundancy
  - Off-diagonal elements should be zero
- $\mathbf{C}_Y$ must be diagonal

Sketch of Algorithm

- Pick vector in m-D space along which variance is maximal and save as p_1
- Pick another direction along which variance is maximized among directions perpendicular to p_1
- Repeat until m principal components have been selected

- From linear algebra: a square matrix can be diagonalized using its eigenvectors as new basis
- **X** is not square in general (m>n in our case), but $\mathbf{C}_x$ always is
- Solution: Singular Value Decomposition (SVD)

Dimensionality Reduction

Carlos Guestrin

Motivation: Dimensionality Reduction

- Input data may have thousands or millions of dimensions!
  - text data have thousands of words
  - image data have millions of pixels
- Dimensionality reduction: represent data with fewer dimensions
  - Easier learning – fewer parameters
  - Visualization – hard to visualize more than 3D or 4D
  - Discover “intrinsic dimensionality” of data for high dimensional data that is truly lower dimensional (e.g. identity of objects in image << number of pixels)

Feature Selection

- Given set of features $\mathbf{X}=<X_1,\ldots,X_n>$
- Some features are more important than others

- Approach: select subset of features to be used by learning algorithm
  - Score each feature (or sets of features)
  - Select set of features with best score

Greedy Forward Feature Selection

- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
  - Select next best feature X_i
    - e.g., one that results in lowest error when learning with $F_t \cup \{X_i\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - Recurse

Greedy Backward Feature Selection

- Greedy heuristic:
  - Start from set of all features $F_0 = F$
  - Run learning algorithm for current set of features F_t
  - Select next worst feature X_i
    - e.g., one that results in lowest error when learning with $F_t - \{X_i\}$
  - $F_{t+1} \leftarrow F_t - \{X_i\}$
  - Recurse

Lower Dimensional Projections

- How would this work for the ball-spring example?

- Rather than picking a subset of the features, we can derive new features that are combinations of existing features

Projection

- Given m data points: $x^i = (x_1^i,\ldots,x_n^i)$, i=1…m
- Represent each point as a projection:

$$\hat{\mathrm{x}}^i = \bar{\mathrm{x}} + \sum_{j=1}^{k} \mathrm{z}_j^i \mathbf{u}_j \quad \text{where} \quad \bar{\mathrm{x}} = \frac{1}{m}\sum_{i=1}^{m} \mathrm{x}^i \quad \text{and} \quad \mathrm{z}_j^i = (\mathrm{x}^i - \bar{\mathrm{x}})\mathbf{u}_j$$

- If k=n, then projected data are equivalent to original data

PCA

- PCA finds projection that minimizes reconstruction error
  - Reconstruction error: norm of distance between original and projected data
- Given $k \leq n$, find $(\mathbf{u_1,\ldots,u_k})$ minimizing reconstruction error:

$$error_k = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i)^2$$

- Error depends on *k+1..n* unused basis vectors

Basic PCA Algorithm

- Start from m×n data matrix **X**
  - *m* data points (samples over time)
  - *n* measurement types
- Re-center: subtract mean from each row of **X**
- Compute covariance matrix:
  - $\Sigma = \mathbf{X}_c^T \mathbf{X}_c$

Note: Covariance matrix is n×n (measurement types)
(But there may be exceptions)

- Compute eigenvectors and eigenvalues of Σ
- Principal components: k eigenvectors with highest eigenvalues

SVD

- Write $\mathbf{X} = \mathbf{V}\ \mathbf{S}\ \mathbf{U}^T$
  - **X:** data matrix, one row per datapoint
  - **V:** weight matrix, one row per datapoint – coordinates of x^i in eigen-space
  - **S:** singular value matrix, diagonal matrix
    - in our setting each entry is eigenvalue λ_j of Σ
  - $\mathbf{U}^T$**:** singular vector matrix
    - in our setting each row is eigenvector $\mathbf{v}_j$ of Σ

Using PCA for Dimensionality Reduction

- Given set of features $\mathbf{X} = \langle \mathbf{X}_1, \ldots, \mathbf{X}_n \rangle$
- Some features are more important than others
  - Reduce noise and redundancy
- Also consider:
  - Rotation

- Approach: Use PCA on **X** to select a few important features
- Then, apply a classification technique in reduced space

Eigenfaces

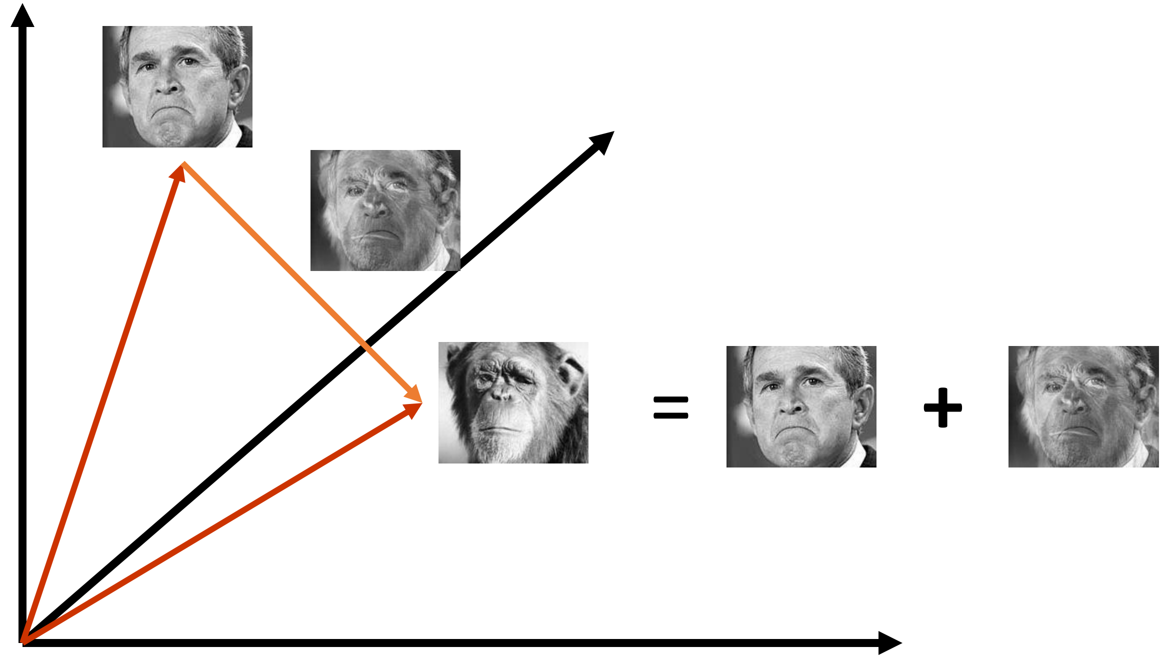
(notes by Srinivasa Narasimhan, CMU)

Eigenfaces

- Face detection and person identification using PCA
- Real time
- Insensitivity to small changes
- Simplicity

- Limitations
  - Only frontal faces – one pose per classifier
  - No invariance to scaling, rotation or translation

Space of All Faces



- An image is a point in a high dimensional space
  - An N x M image is a point in R^{NM}
  - We can define vectors in this space as we did in the 2D case

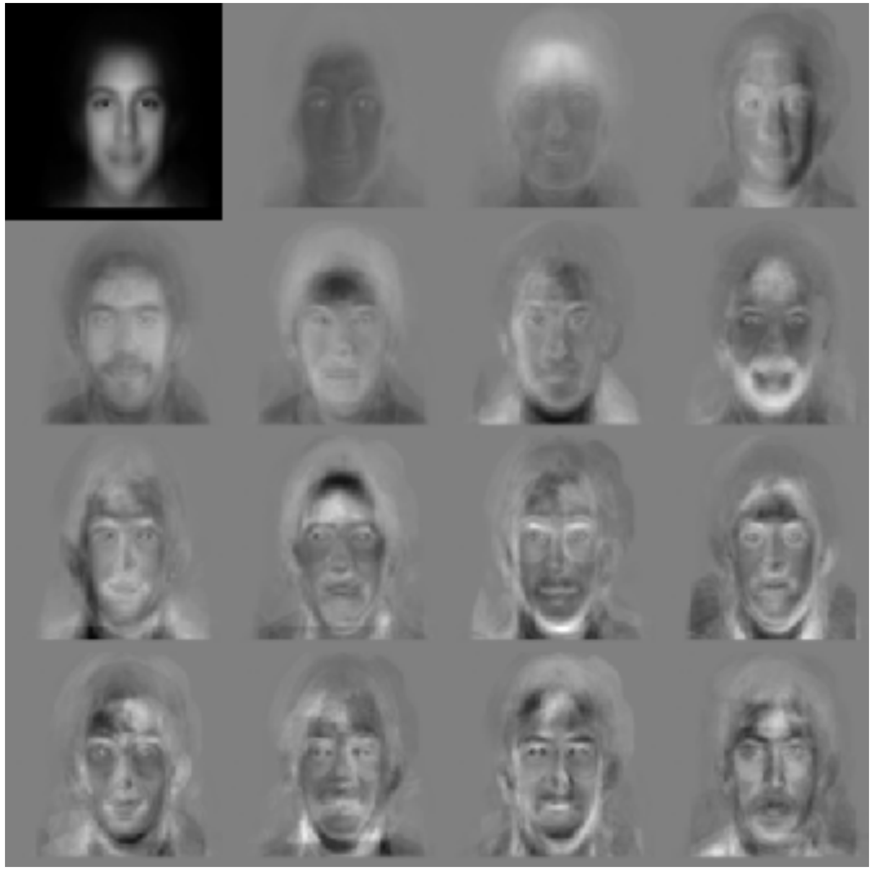
Key Idea

- Images in the possible set $\chi = \{\hat{x}_{RL}^{P}\}$ are highly correlated

- So, compress them to a low-dimensional subspace that captures key appearance characteristics of the visual DOFs

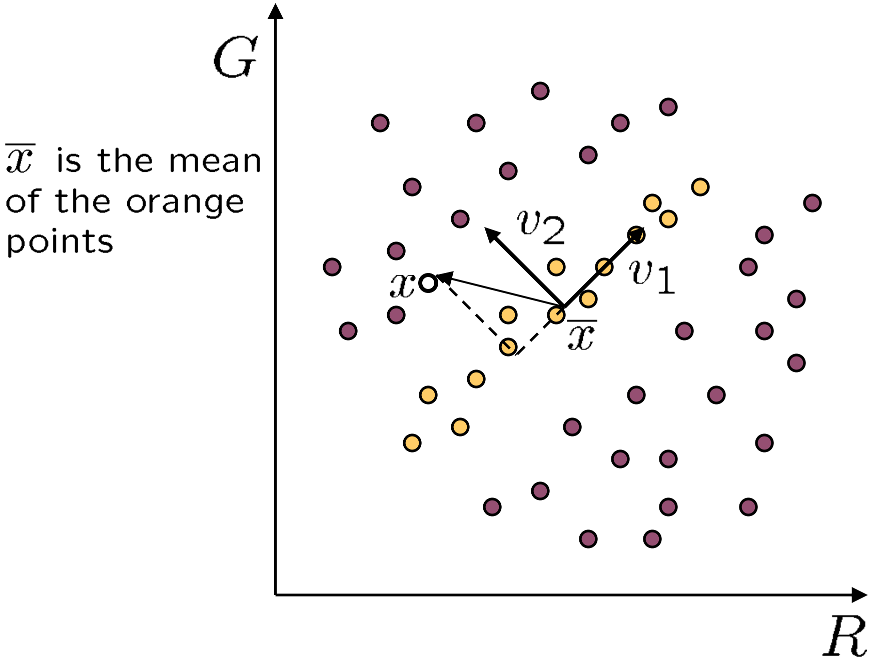
- EIGENFACES [Turk and Pentland]: USE PCA

Eigenfaces



Eigenfaces look somewhat like generic faces

Linear Subspaces



convert **x** into **v_1**, **v_2** coordinates

$$\mathbf{x} \rightarrow ((\mathbf{x} - \overline{x}) \cdot \mathbf{v_1}, (\mathbf{x} - \overline{x}) \cdot \mathbf{v_2})$$

What does the **v_2** coordinate measure?

- distance to line
- use it for classification—near 0 for orange pts

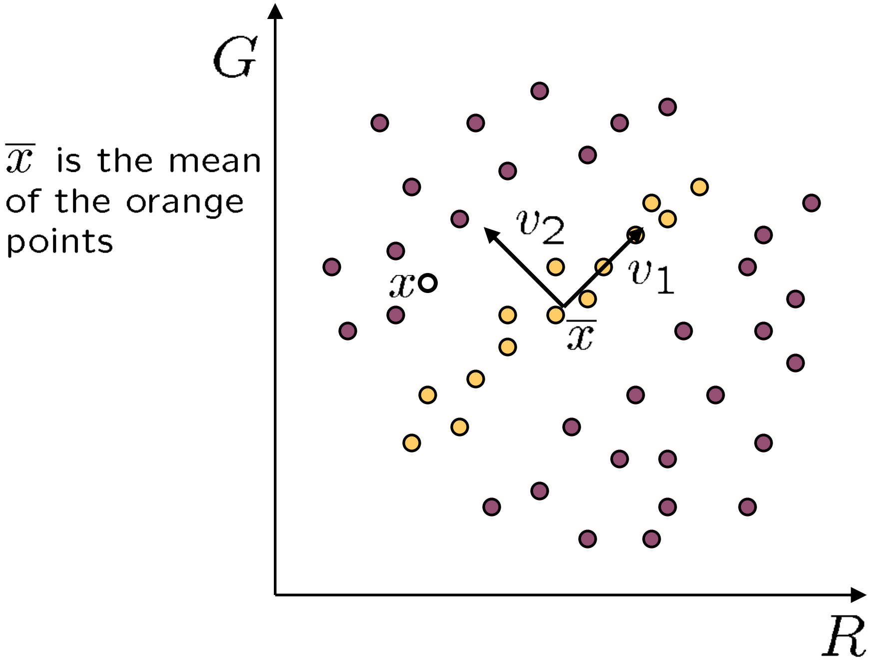
What does the **v_1** coordinate measure?

- position along line
- use it to specify which orange point it is

- Classification can be expensive
  - Must either search (e.g., nearest neighbors) or store large probability density functions.

- Suppose the data points are arranged as above
  - Idea—fit a line, classifier measures distance to line

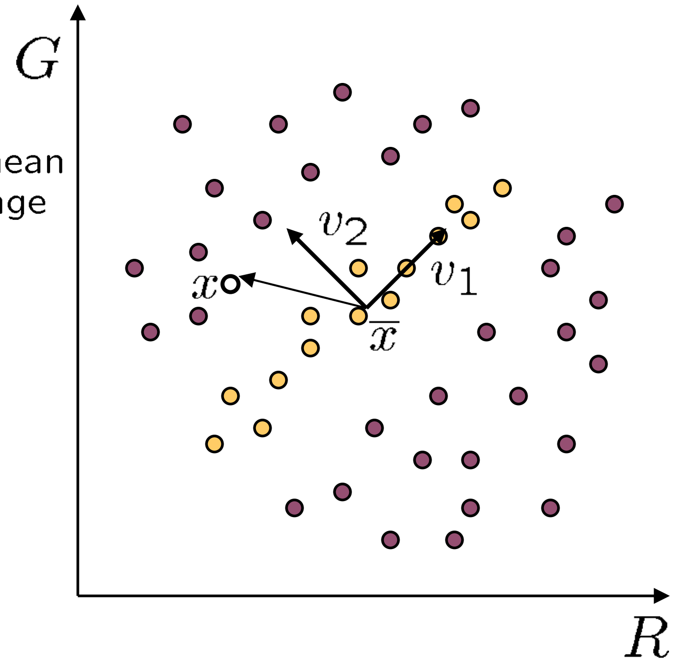
Dimensionality Reduction



- Dimensionality reduction
  - We can represent the orange points with *only* their $\mathbf{v_1}$ coordinates
    - since $\mathbf{v_2}$ coordinates are all essentially 0
  - This makes it much cheaper to store and compare points
  - A bigger deal for higher dimensional problems

Linear Subspaces

$\overline{x}$ is the mean of the orange points



Consider the variation along direction **v** among all of the orange points:

$$var(\mathbf{v}) = \sum_{\text{orange point } \mathbf{x}} \|(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\|^{\mathbf{2}}$$

What unit vector **v** minimizes *var*?

$$\mathbf{v}_2 = min_{\mathbf{v}} \ \{var(\mathbf{v})\}$$

What unit vector **v** maximizes *var*?

$$\mathbf{v}_1 = max_{\mathbf{v}} \ \{var(\mathbf{v})\}$$

$$\begin{aligned} var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\| \\ &= \sum_{\mathbf{x}} \mathbf{v}^{\mathbf{T}}(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}}\mathbf{v} \\ &= \mathbf{v}^{\mathbf{T}}\left[\sum_{\mathbf{x}}(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}}\right]\mathbf{v} \\ &= \mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}}(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}} \end{aligned}$$

Solution: $\mathbf{v_1}$ is eigenvector of **A** with *largest* eigenvalue
$\mathbf{v_2}$ is eigenvector of **A** with *smallest* eigenvalue

Higher Dimensions

- Suppose each data point is N-dimensional
  - Same procedure applies:

$$\begin{aligned} var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\| \\ &= \mathbf{v}^{\mathbf{T}} \mathbf{A} \mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^{\mathbf{T}} \end{aligned}$$

  - The eigenvectors of **A** define a new coordinate system
    - eigenvector with largest eigenvalue captures the most variation among training vectors **x**
    - eigenvector with smallest eigenvalue has least variation
  - We can compress the data by only using the top few eigenvectors
    - corresponds to choosing a “linear subspace”
      - represent points on a line, plane, or “hyper-plane”
    - these eigenvectors are known as the ***principal components***

Problem: Size of Covariance Matrix A

- Suppose each data point is N-dimensional (N pixels)

  - The size of covariance matrix A is N^2
  - The number of eigenfaces is N

  - Example: For N = 256 x 256 pixels,

    Size of A will be 65536 x 65536 !

    Number of eigenvectors will be 65536 !

  Typically, only 20-30 eigenvectors suffice. So, this method is very inefficient!

Efficient Computation of Eigenvectors

If B is $M\text{x}N$ and $M<<N$ then $A=B^TB$ is $N\text{x}N >> M\text{x}M$

- $M \rightarrow$ number of images, $N \rightarrow$ number of pixels
- use BB^T instead, eigenvector of BB^T is easily converted to that of B^TB

$$
\begin{aligned}
&(BB^T)\, y = e\, y \\
&\Rightarrow\ B^T(BB^T)\, y = e\,(B^Ty) \\
&\Rightarrow\ (B^TB)(B^Ty) = e\,(B^Ty) \\
&\Rightarrow\ B^Ty \text{ is the eigenvector of } B^TB
\end{aligned}
$$

Eigenfaces – summary in words

- Eigenfaces are

  the eigenvectors of
  the covariance matrix of
  the probability distribution of
  the vector space of
  human faces

- Eigenfaces are the 'standardized face ingredients' derived from the statistical analysis of many pictures of human faces

- A human face may be considered to be a combination of these standardized faces

Generating Eigenfaces – in words

1. Large set of images of human faces is taken
2. The images are normalized to line up the eyes, mouths and other features
3. The eigenvectors of the covariance matrix of the face image vectors are then extracted
4. These eigenvectors are called eigenfaces

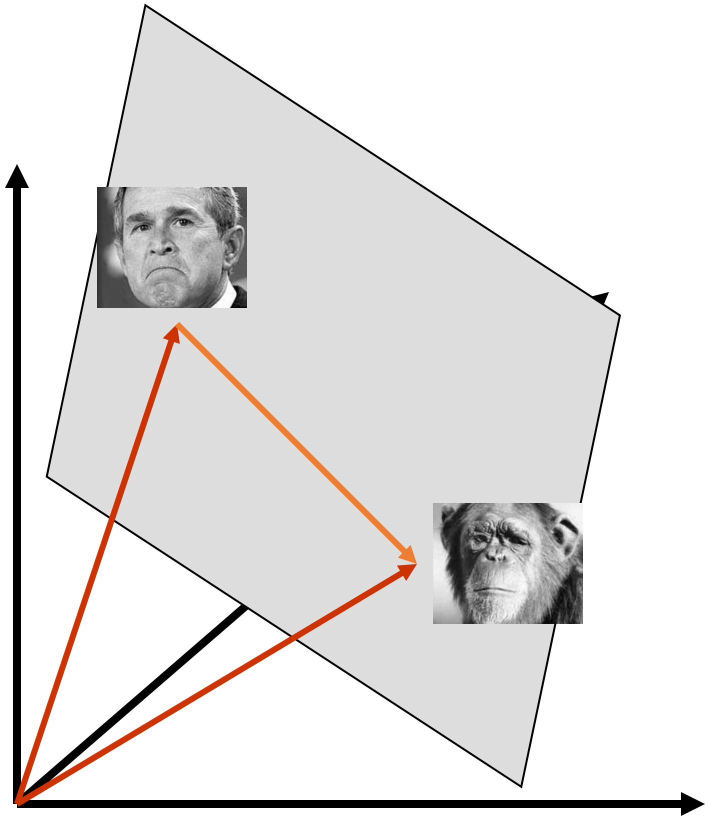
Eigenfaces for Face Recognition

- When properly weighted, eigenfaces can be summed together to create an approximate gray-scale rendering of a human face.

- Remarkably few eigenvector terms are needed to give a fair likeness of most people's faces.

- Hence eigenfaces provide a means of applying data compression to faces for identification purposes.

Dimensionality Reduction



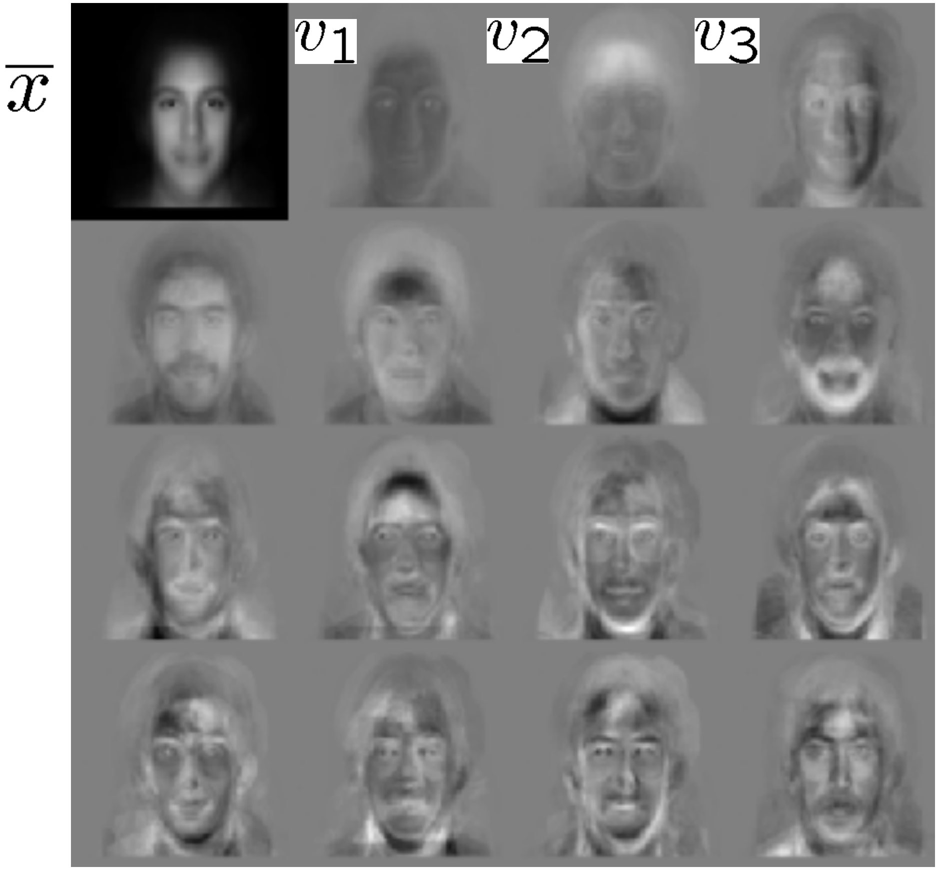
The set of faces is a "subspace" of the set of images

- Suppose it is K dimensional
- We can find the best subspace using PCA
- This is like fitting a "hyper-plane" to the set of faces
  - spanned by vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_K}$

Any face: $\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_k\mathbf{v_k}$

Eigenfaces

- PCA extracts the eigenvectors of **A**
  - Gives a set of vectors $\mathbf{v_1}$, $\mathbf{v_2}$, $\mathbf{v_3}$, ...
  - Each one of these vectors is a direction in face space
    - what do these look like?



Projecting onto the Eigenfaces

- The eigenfaces **$\mathbf{v}_1$, ..., $\mathbf{v}_K$** span the space of faces
  - A face is converted to eigenface coordinates by

$$\mathbf{x} \rightarrow (\underbrace{(\mathbf{x}-\overline{\mathbf{x}}) \cdot \mathbf{v_1}}_{a_1}, \underbrace{(\mathbf{x}-\overline{\mathbf{x}}) \cdot \mathbf{v_2}}_{a_2}, \ldots, \underbrace{(\mathbf{x}-\overline{\mathbf{x}}) \cdot \mathbf{v_K}}_{a_K})$$

$$\mathbf{x} \approx \overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K}$$

$\mathbf{x}$

$a_1\mathbf{v_1}$ $a_2\mathbf{v_2}$ $a_3\mathbf{v_3}$ $a_4\mathbf{v_4}$ $a_5\mathbf{v_5}$ $a_6\mathbf{v_6}$ $a_7\mathbf{v_7}$ $a_8\mathbf{v_8}$

Is this a face or not?

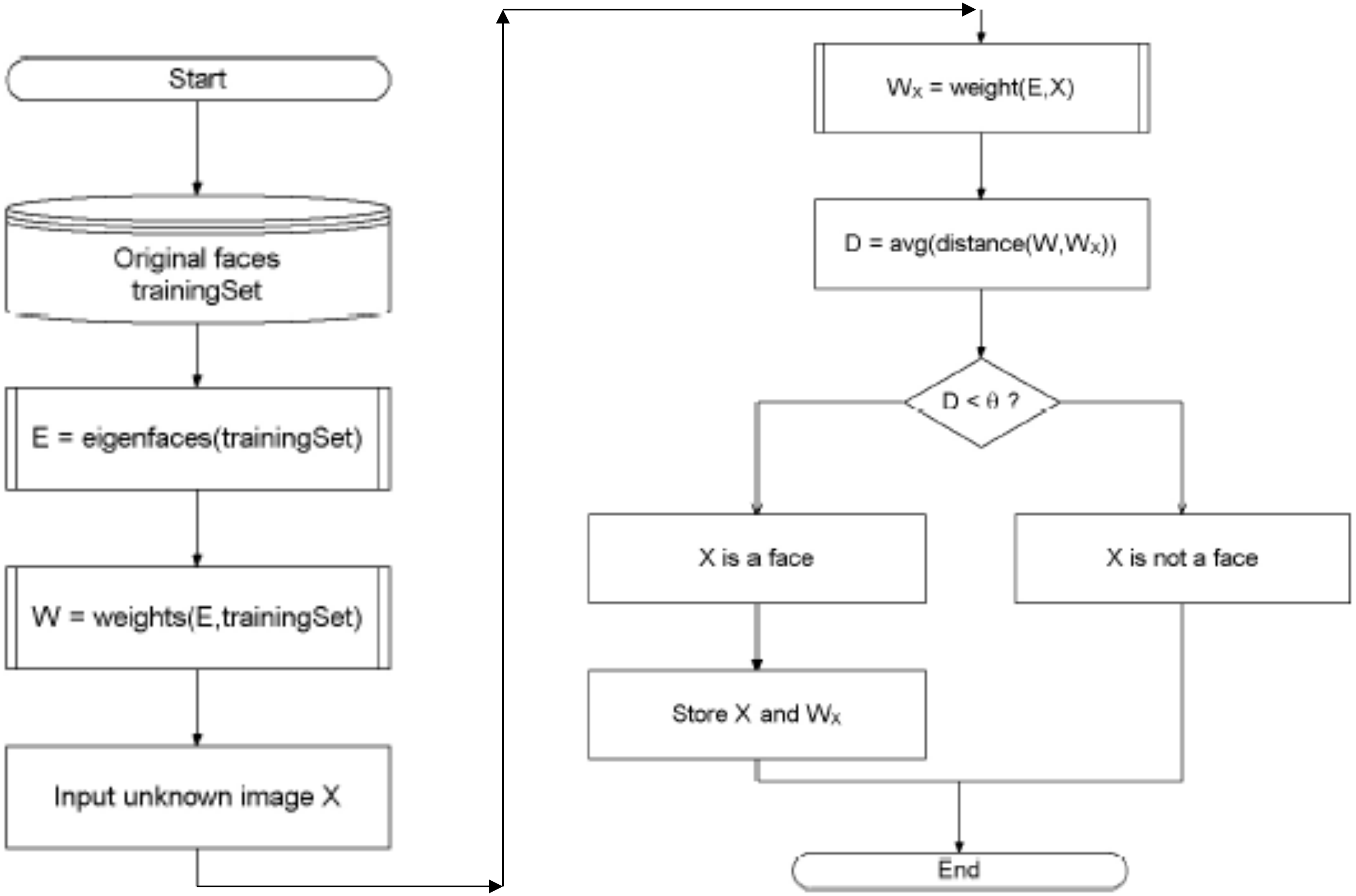


Figure 1: High-level functioning principle of the eigenface-based facial recognition algorithm

Recognition with Eigenfaces

- Algorithm

1. Process the image database (set of images with labels)
    - Run PCA—compute eigenfaces
    - Calculate the K coefficients for each image
2. Given a new image (to be recognized) **x**, calculate K coefficients
3. Detect if x is a face

$$\mathbf{x} \to (a_1, a_2, \ldots, a_K)$$

4. If it is a face, who is it?

$$\|\mathbf{x} - (\overline{\mathbf{x}} + a_1\mathbf{v_1} + a_2\mathbf{v_2} + \ldots + a_K\mathbf{v_K})\| < \text{threshold}$$

- Find closest labeled face in database
    - nearest-neighbor in K-dimensional space

Key Property of Eigenspace Representation

Given

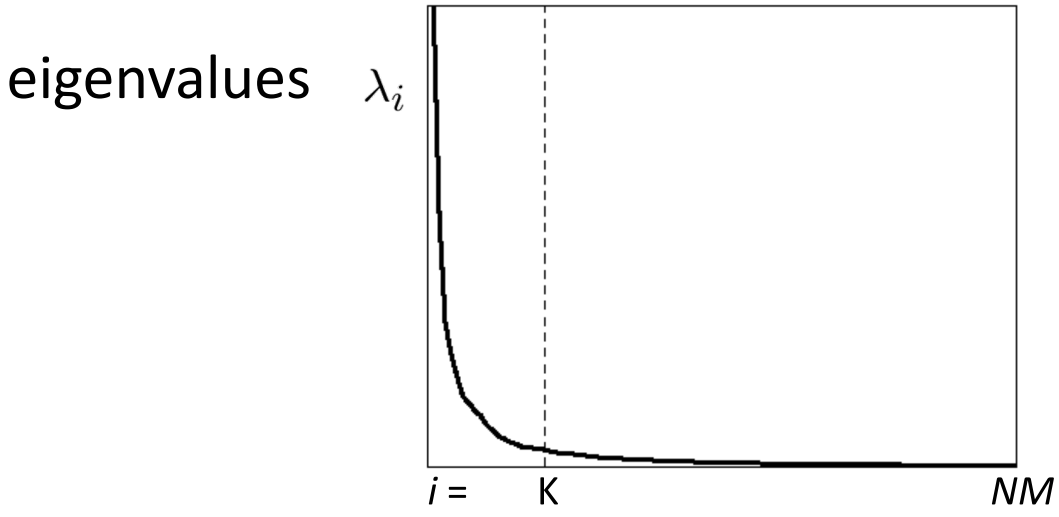
- 2 images x_1, x_2 that are used to construct the Eigenspace
- g_1 is the eigenspace projection of image x_1
- g_2 is the eigenspace projection of image x_2

Then,

$$\boxed{\| g_2 - g_1 \| \approx \| x_2 - x_1 \|}$$

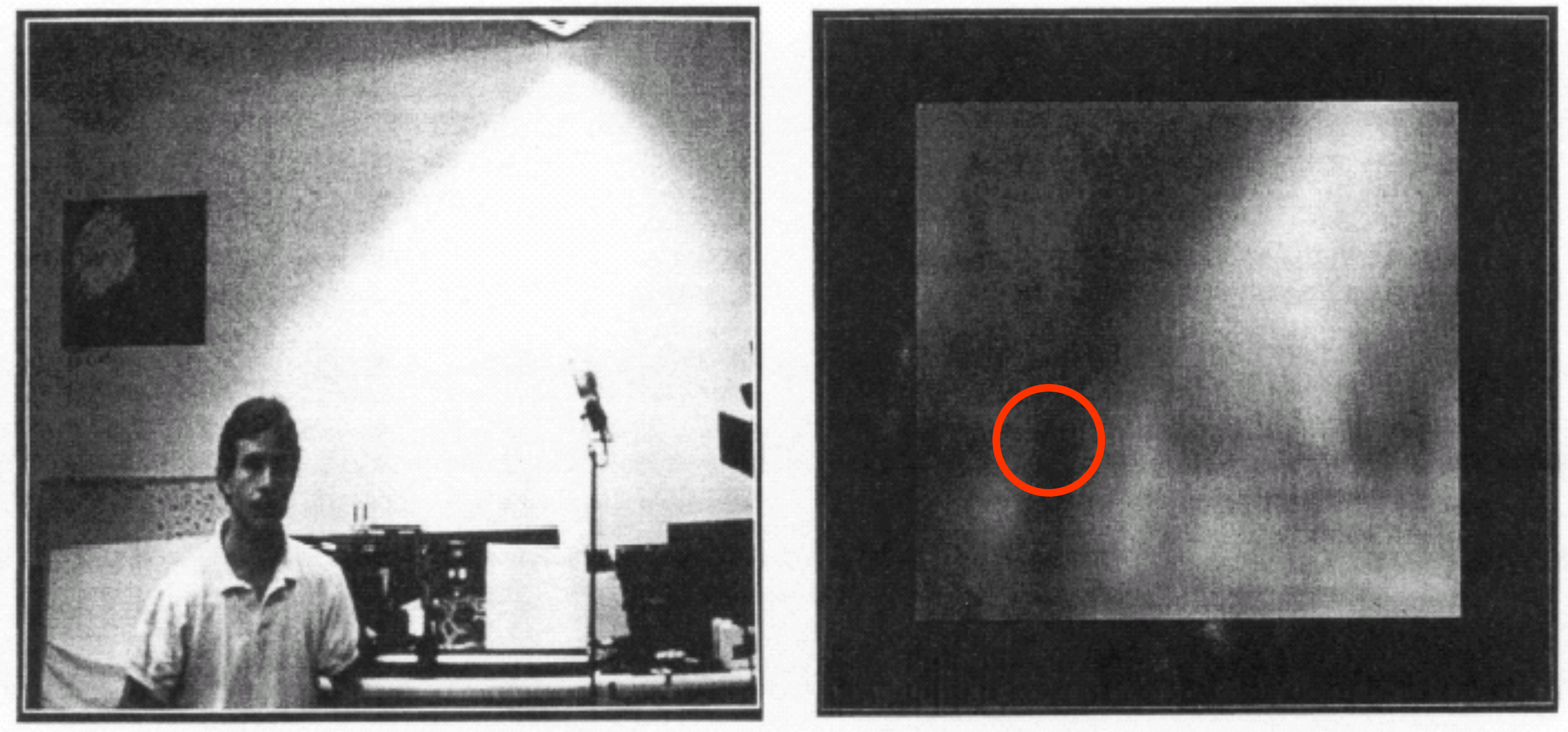
That is, distance in Eigenspace is approximately equal to the distance between original images

Choosing the Dimension K



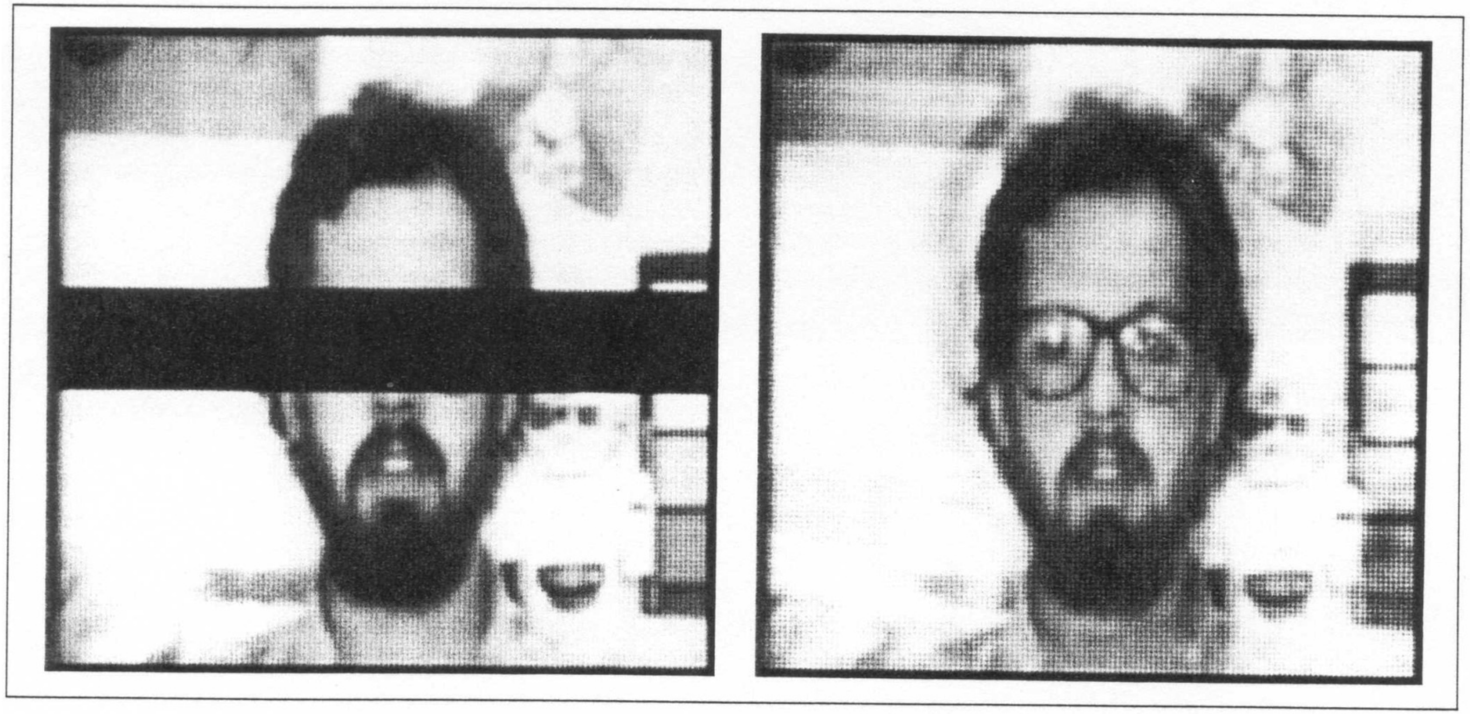
- How many eigenfaces to use?
- Look at the decay of the eigenvalues
  - the eigenvalue tells you the amount of variance "in the direction" of that eigenface
  - ignore eigenfaces with low variance

Results



- Face detection using sliding window
  - Dark: small distance
  - Bright: large distance

Results



- Reconstruction of corrupted image
  - Project on eigenfaces and compute weights
  - Take weighted sum of eigenfaces to synthesize face image

Results

- Left: query
- Right: best match from database

Results

- Each new image is reconstructed with one additional eigenface

Singular Value Decomposition (Will not be in exams)

Michael Elad

Singular Value Decomposition

The eigenvectors of a matrix **A** form a basis for working with **A**

However, for rectangular matrices **A** (m x n), dim(**A**$\underline{x}$) ≠ dim($\underline{x}$) and the concept of eigenvectors does not exist

Note: here each row of A is a measurement in time and each column a measurement type

Yet, $\mathbf{A}^T\mathbf{A}$ (n x n) is a symmetric, real matrix (**A** is real) and therefore, there is an orthonormal basis of eigenvectors $\{\underline{u}_K\}$ for $\mathbf{A}^T\mathbf{A}$.

Consider the vectors $\{\underline{v}_K\}$ $$\underline{v}_k = \frac{\mathbf{A}\underline{u}_k}{\sqrt{\lambda_k}}$$

They are also orthonormal, since: $$\underline{u}_j^T \mathbf{A}^T \mathbf{A} \underline{u}_k = \lambda_k \delta(k-j)$$

Singular Value Decomposition

Since A^TA is positive semidefinite, its eigenvalues are non-negative $\{\lambda_k \geq 0\}$ (Why?)

Define the singular values of **A** as $\sigma_k = \sqrt{\lambda_k}$

and order them in a non-increasing order: $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n \geq 0$

Motivation: One can see, that if **A** itself is square and symmetric, then $\{\underline{u}_k, \sigma_k\}$ are the set of

its own eigenvectors and eigenvalues.

For a general matrix **A**, assume $\{\sigma_1 \geq \sigma_2 \geq \ldots \sigma_R > 0 = \sigma_{r+1} = \sigma_{r+2} = \ldots = \sigma_n\}$.

$$\mathbf{A}\underline{u}_k = 0 \cdot \underline{v}_k, \qquad k = r+1, \ldots, n$$

$$\underline{u}_k^{(n \times 1)}; \;\; \underline{v}_k^{(m \times 1)}$$

Singular Value Decomposition

Now we can write:

$$\begin{bmatrix} | & & | & | & & | \\ \mathbf{A}\underline{u}_1 & \cdots & \mathbf{A}\underline{u}_r & \mathbf{A}\underline{u}_{r+1} & \cdots & \mathbf{A}\underline{u}_n \\ | & & | & | & & | \end{bmatrix} = \mathbf{A}\begin{bmatrix} | & & | & | & & | \\ \underline{u}_1 & \cdots & \underline{u}_r & \underline{u}_{r+1} & \cdots & \underline{u}_n \\ | & & | & | & & | \end{bmatrix} = \mathbf{AU} =$$

$$= \begin{bmatrix} | & & | & | & & | \\ \sigma_1\underline{v}_1 & \cdots & \sigma_r\underline{v}_r & 0\cdot\underline{v}_{r+1} & \cdots & 0\cdot\underline{v}_n \\ | & & | & | & & | \end{bmatrix} = \begin{bmatrix} | & & | & | & & | \\ \underline{v}_1 & \cdots & \underline{v}_r & \underline{v}_{r+1} & \cdots & \underline{v}_n \\ | & & | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & 0 & 0 & & 0 \\ & \ddots & & & & \\ 0 & & \sigma_r & 0 & & 0 \\ 0 & & 0 & 0 & & 0 \\ & & & & \ddots & \\ 0 & & 0 & 0 & & 0 \end{bmatrix} = \mathbf{V\Sigma}$$

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

$$A^{(m\times n)} = V^{(m\times m)}\Sigma^{(m\times n)}U^{(n\times n)^T}$$

SVD: Example

Let us find the SVD for the matrix: $\mathbf{A} = \begin{bmatrix} -1 & 1 \\ 2 & 2 \end{bmatrix}$

In order to find **V**, we need to calculate eigenvectors of $\mathbf{A}^T\mathbf{A}$:

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} -1 & 2 \\ 1 & 2 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}$$

$$(5-\lambda)^2-9=0; \quad \Rightarrow \quad \lambda_{1,2} = \frac{10 \pm \sqrt{100-64}}{2} = 5 \pm 3 = 8, 2$$

SVD: Example

The corresponding eigenvectors are found by:

$$\begin{bmatrix} 5-\lambda_i & 3 \\ 3 & 5-\lambda_i \end{bmatrix} \underline{u}_i = 0$$

$$\begin{bmatrix} -3 & 3 \\ 3 & -3 \end{bmatrix} \underline{u}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \underline{u}_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

$$\begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix} \underline{u}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \underline{u}_2 = \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$$

SVD: Example

Now, we obtain V and Σ :

$$\mathbf{A}\underline{u}_1 = \sigma_1 \underline{v}_1 = \begin{bmatrix} -1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 0 \\ 2\sqrt{2} \end{bmatrix} = 2\sqrt{2} \begin{bmatrix} 0 \\ 1 \end{bmatrix} \qquad \underline{v}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \sigma_1 = 2\sqrt{2};$$

$$\mathbf{A}\underline{u}_2 = \sigma_2 \underline{v}_2 = \begin{bmatrix} -1 & 1 \\ 2 & 2 \end{bmatrix} \begin{bmatrix} -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix} = \sqrt{2} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \qquad \underline{v}_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \sigma_2 = \sqrt{2};$$

$$\mathrm{A=V\Sigma U^T}: \quad \begin{bmatrix} -1 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$