Analytic Feature Extraction Methods

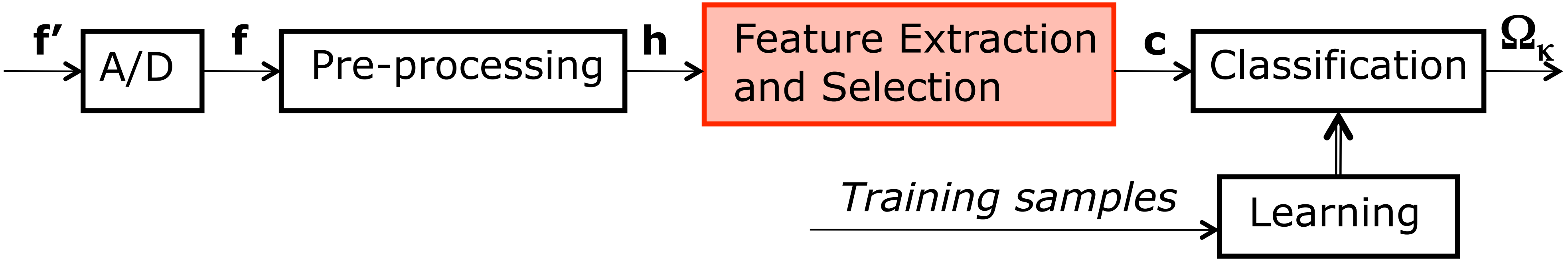
Principal Component Analysis,
Linear Discriminant Analysis

Dr. Elli Angelopoulou

Lehrstuhl für Mustererkennung (Informatik 5)

Friedrich-Alexander-Universität Erlangen-Nürnberg

Pattern Recognition Pipeline



- Heuristic feature extraction methods
  - Projection to new orthogonal basis
  - Linear Predictive Coding (LPC)
  - Geometric Moments
  - Wavelets
- Analytic feature extraction methods
- Feature selection

Analytic Methods for Feature Computation

- Idea: Construct a feature vector so that it supports the postulates of pattern recognition.

- Approach: Find a linear transformation of the pattern so that an optimality criterion is satisfied.

- Let $\vec{f} \in R^N$ be the input signal. The linear transformation $\Phi : \vec{f} \rightarrow \vec{c}$ maps $\vec{f}$ to the feature vector $\vec{c} \in R^M$, so that $M \leq N$ (ideally $M << N$):

$$\vec{c} = \Phi \vec{f}$$

- Problem: Compute a matrix Φ, so that the resulting features $\vec{c}$ optimize a quality criterion.

Goal of PCA

- The goal of Principal Component Analysis (PCA) is to find a transformation matrix Φ such that the resulting features can best describe the variation that is observed in the original data.
- We want to transform the data so that in their new representation the data is not all tightly clustered, but rather spread across the new M dimensional space.
- We want to maximize the distance between the feature vectors.

PCA Optimization Criterion

- We want to maximize the distance between the feature vectors.
- The Euclidean distance between two vectors $\vec{c}_i$ and $\vec{c}_j$ is:

$$\left(\vec{c}_i - \vec{c}_j\right)^T \left(\vec{c}_i - \vec{c}_j\right)$$

- In PCA we want to derive a linear transformation Φ that maximizes this distance over *all* the pairs of points. We want to maximize:

$$\sum_{i=1}^{K} \sum_{j=1}^{K} \left(\vec{c}_i - \vec{c}_j\right)^T \left(\vec{c}_i - \vec{c}_j\right)$$

where *K* is the number of data points.

PCA Optimization Criterion - continued

- In PCA we want to maximize:

$$s_1(\Phi) = \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{c}_i - \vec{c}_j\right)^T\left(\vec{c}_i - \vec{c}_j\right)$$

$$= \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right)^T\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right)$$

- $s_1()$ is the total square distance of all features to each other.
- A trivial solution to this maximization problem is one that has Φ approaching infinity.
- Idea: bind the components of Φ to be within a certain range.

Refined PCA Optimization Criterion

- A simple way for controlling the range of values of the components of Φ is to try to keep its norm as close to unity.
- So we have a 2nd optimization goal: minimize $\left(\|\Phi\|_2 - 1\right)$ where $\|\cdot\|_2$ is an approximation of the Frobenius norm of the matrix . It is the sum of the squares of the elements of Φ.
- We can combine these two optimization goals into a single optimization criterion using a Lagrange multiplier λ:

$$s_1(\Phi) = \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right)^T\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right) - \lambda\left(\|\Phi\|_2 - 1\right)$$

Refined PCA Optimization Criterion – cont.

- Goal of PCA: Find Φ that maximizes

$$s_1(\Phi) = \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right)^T\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right) - \lambda\left(\left\|\Phi\right\|_2 - 1\right)$$

- The 1st term controls the spread of the feature points.
- The 2nd term controls the of Φ.
- In other words, we are looking for a linear transformation Φ, among all possible Φs that maximizes $s_1()$:

$$\hat{\Phi} = \arg\max_{\Phi} s_1(\Phi)$$

Derivation of the PCA Transformation Matrix

- How do we compute the matrix $\hat{\Phi}$ that satisfies

$$\hat{\Phi} = \arg\max_{\Phi} \sum_{i=1}^{K} \sum_{j=1}^{K} \left(\Phi \vec{f}_i - \Phi \vec{f}_j\right)^T \left(\Phi \vec{f}_i - \Phi \vec{f}_j\right) - \lambda\left(\left\|\Phi\right\|_2 - 1\right)$$

- Compute the partial derivative with respect to the terms $\vec{\varphi}_i$ of the transformation matrix Φ. The values of $\vec{\varphi}_i$ that set the partial derivative to zero are the ones that maximize our optimization function.

- Since the equation as-is is quite complex, we will look at each part individually (distance maximization and limiting the norm of the matrix).

Maximizing the Spread

- First, let us simplify the summation by factoring out the transformation matrix:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi\vec{f}_i-\Phi\vec{f}_j\right)^T\left(\Phi\vec{f}_i-\Phi\vec{f}_j\right)$$

$$=\sum_{i=1}^{K}\sum_{j=1}^{K}\left[\Phi\left(\vec{f}_i-\vec{f}_j\right)\right]^T\Phi\left(\vec{f}_i-\vec{f}_j\right)$$

$$=\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{f}_i-\vec{f}_j\right)^T\Phi^T\Phi\left(\vec{f}_i-\vec{f}_j\right)$$

- Let $g_{ij}=\left(\vec{f}_i-\vec{f}_j\right)$ then the previous equation becomes:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}g_{ij}^T\Phi^T\Phi g_{ij}$$

Continued Derivation

- The equation $\sum_{i=1}^{K}\sum_{j=1}^{K} g_{ij}^T \Phi^T \Phi g_{ij}$ is in a very convenient form because it allows us to use a property of the trace of symmetric matrices.
- For a symmetric matrix *M*: $x^T M y = \text{trace}\left(M x y^T\right)$
- By construction $\Phi^T \Phi$ is a symmetric matrix. Thus:

$$\sum_{i=1}^{K}\sum_{j=1}^{K} g_{ij}^T \Phi^T \Phi g_{ij} = \sum_{i=1}^{K}\sum_{j=1}^{K} \text{trace}\left(\Phi^T \Phi g_{ij} g_{ij}^T\right)$$

- But $\text{trace}(M) = \text{trace}\left(M^T\right)$. Hence:

$$\sum_{i=1}^{K}\sum_{j=1}^{K} \text{trace}\left(\Phi^T \Phi g_{ij} g_{ij}^T\right) = \sum_{i=1}^{K}\sum_{j=1}^{K} \text{trace}\left(g_{ij} g_{ij}^T \Phi^T \Phi\right)$$

Continued Derivation 2

- We have shown so far that the square distance of all possible feature pairs is:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{c}_i - \vec{c}_j\right)^T\left(\vec{c}_i - \vec{c}_j\right) = \sum_{i=1}^{K}\sum_{j=1}^{K}\mathrm{trace}\left(g_{ij}g_{ij}^T\Phi^T\Phi\right)$$

where $g_{ij} = \left(\vec{f}_i - \vec{f}_j\right)$

- Let $M_{ij} = g_{ij}g_{ij}^T$
- Since M_{ij} contains only original signal measurements, it is also known as the measurement matrix.
- We can rewrite the distance over all feature pairs as:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{c}_i - \vec{c}_j\right)^T\left(\vec{c}_i - \vec{c}_j\right) = \sum_{i=1}^{K}\sum_{j=1}^{K}\mathrm{trace}\left(M_{ij}\Phi^T\Phi\right)$$

Continued Derivation 3

- Now recall that $\Phi^T = \left(\vec{\varphi}_1, \vec{\varphi}_2, \ldots, \vec{\varphi}_M\right)$ where the $\vec{\varphi}_i$s are column vectors. Then the last equation becomes:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\operatorname{trace}\left(M_{ij}\Phi^T\Phi\right) = \sum_{i=1}^{K}\sum_{j=1}^{K}\operatorname{trace}\left(M_{ij}\left[\vec{\varphi}_1, \vec{\varphi}_2, \ldots, \vec{\varphi}_M\right]\begin{bmatrix}\vec{\varphi}_1\\ \vec{\varphi}_2\\ \vdots\\ \vec{\varphi}_M\end{bmatrix}\right)$$

$$= \sum_{i=1}^{K}\sum_{j=1}^{K}\operatorname{trace}\left(M_{ij}\sum_{k=1}^{M}\vec{\varphi}_k\vec{\varphi}_k^T\right)$$

Continued Derivation 4

- We can reuse the property $x^T My = \operatorname{trace}\left(Mxy^T\right)$ to remove the trace from the previous equation:

$$\sum_{i=1}^{K}\sum_{j=1}^{K}\operatorname{trace}\left(M_{ij}\sum_{k=1}^{M}\vec{\varphi}_k\vec{\varphi}_k^T\right)=\sum_{k=1}^{M}\vec{\varphi}_k^T\sum_{i=1}^{K}\sum_{j=1}^{K}M_{ij}\vec{\varphi}_k$$

- Let $Q=\sum_{i=1}^{K}\sum_{j=1}^{K}M_{ij}$. Reminder: $M_{ij}=\left(\vec{f}_i-\vec{f}_j\right)\left(\vec{f}_i-\vec{f}_j\right)^T$
- Then the optimization function can be rewritten as:

$$s_1(\Phi)=\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi\vec{f}_i-\Phi\vec{f}_j\right)^T\left(\Phi\vec{f}_i-\Phi\vec{f}_j\right)-\lambda\left(\|\Phi\|_2-1\right)$$

$$=\sum_{k=1}^{M}\vec{\varphi}_k^T Q\vec{\varphi}_k-\lambda\left(\sum_{k=1}^{M}\vec{\varphi}_k^T\vec{\varphi}_k-1\right)$$

Continued Derivation 5

- We can now use the simplified form of the optimization function:

$$s_1(\Phi) = \sum_{k=1}^{M} \vec{\varphi}_k^T Q \vec{\varphi}_k - \lambda \left(\sum_{k=1}^{M} \vec{\varphi}_k^T \vec{\varphi}_k - 1 \right)$$

  and examine its partial derivative w.r.t Φ, $\partial s_1(\Phi) / \partial \Phi$

- For each individual basis vector $\vec{\varphi}_k$ we get:

$$\frac{\partial s_1(\Phi)}{\partial \vec{\varphi}_k} = 0 \Rightarrow 2Q\vec{\varphi}_k - 2\lambda\vec{\varphi}_k = 0 \Rightarrow \boxed{Q\vec{\varphi}_k = \lambda\vec{\varphi}_k}$$

- However, this is a typical eigenvalue, eigenvector problem: We have a vector, we apply a transformation to it and we get a scalar multiple (i.e an eigenvalue) of the same vector.

Summary of Derivation

- Thus, the matrix Φ that maximizes the overall spread of the features while having bounded elements, i.e. the matrix that satisfies:

$$s_1(\Phi) = \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right)^T\left(\Phi\vec{f}_i - \Phi\vec{f}_j\right) - \lambda\left(\left\|\Phi\right\|_2 - 1\right)$$

is the one where the component basis vectors satisfy:

$$Q\vec{\varphi}_k = \lambda\vec{\varphi}_k$$

where

$$Q = \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{f}_i - \vec{f}_j\right)\left(\vec{f}_i - \vec{f}_j\right)^T$$

kernel matrix or *covariance matrix*

PCA Algorithm

- The matrix Φ that maximizes the spread of features is constructed as follows:

1. Build *Q*, the *N*x*N* kernel or covariance matrix.
2. Compute the eigenvectors of *Q* via SVD (Q is a positive symmetric matrix so it is easily diagonalizable).
3. The eigenvectors are sorted according to their eigenvalues.
4. Use the most significant *M* eigenvectors.
5. The eigenvectors of *Q* become the rows of Φ.

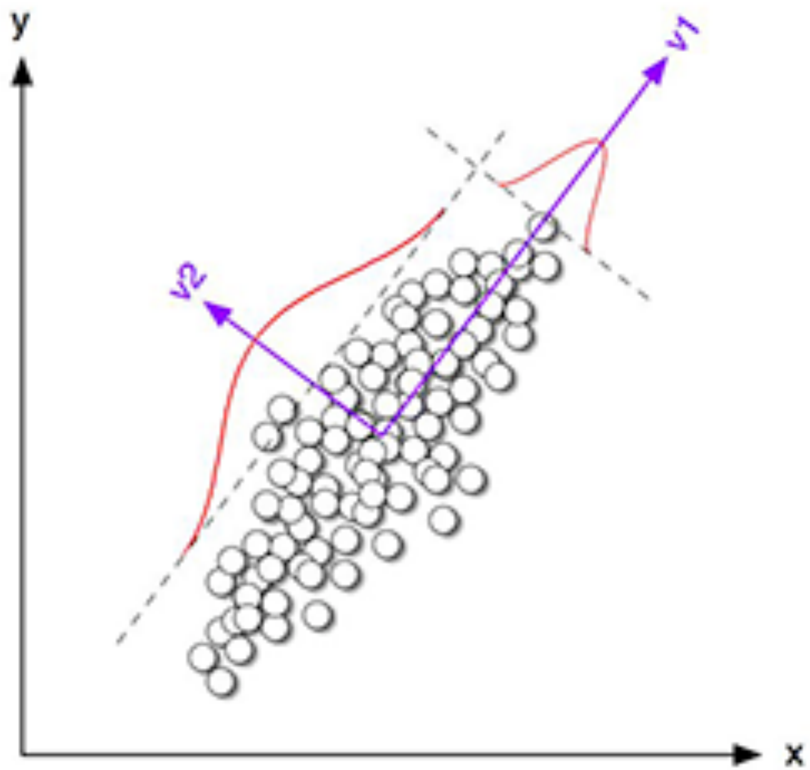
Matrix Diagonalization

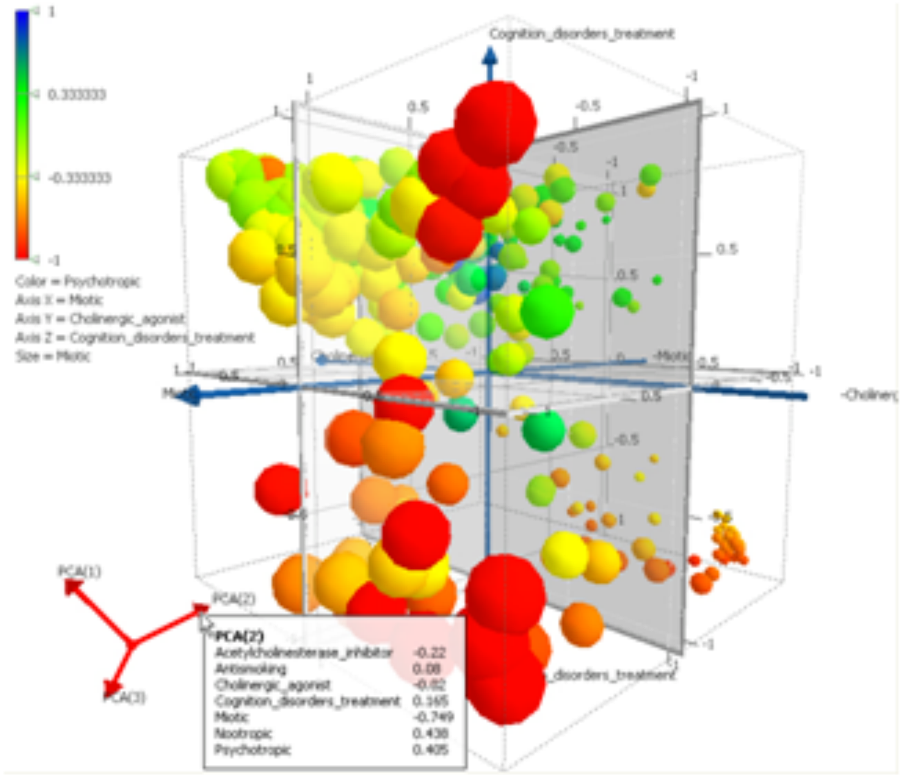
- Given a positive symmetric matrix *Q*, one can compute a matrix *V* that diagonalizes *Q*.

$$V^{-1}QV = D$$

- *D* is a diagonal matrix that contains the eigenvalues of *Q* (often sorted in descending order).
- *V* is a matrix of eigenvectors. Each column of *V* is an eigenvector, whose eigenvalue is in the corresponding column in D.
- There are many methods for diagonalizing a matrix (e.g. Jacobi diagonalization) including SVD which for real symmetric matrices reduced to diagonalization.

Simple PCA Examples



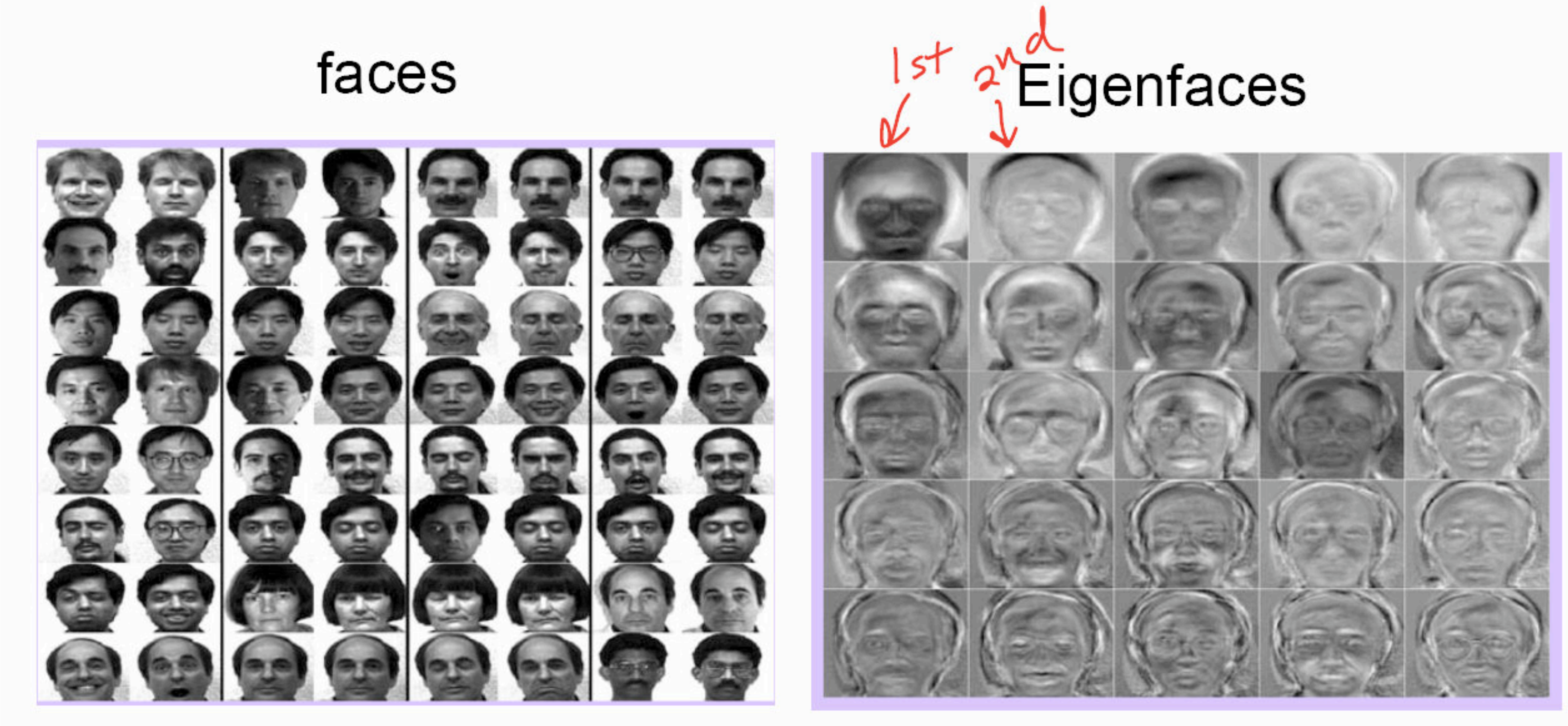


Intuition behind PCA

- The goal of pattern recognition is to reliably identify signals that belong to a specific class (e.g. people, cars, coffee beans of different qualities, etc.).
- It makes sense to use a representation that best captures what "makes a car a car" and how it differs from people.
- Thus, given a signal, we look for the attributes which can explain the observed covariance/co-dependence in a set of variables.
- For better separability of classes we want:
  - attributes that are uncorrelated
  - show high variance, so that they capture the variety of the members within a single class
- These uncorrelated underlying attributes are called *factors* or *principal components*.

PCA Example: Eigenfaces

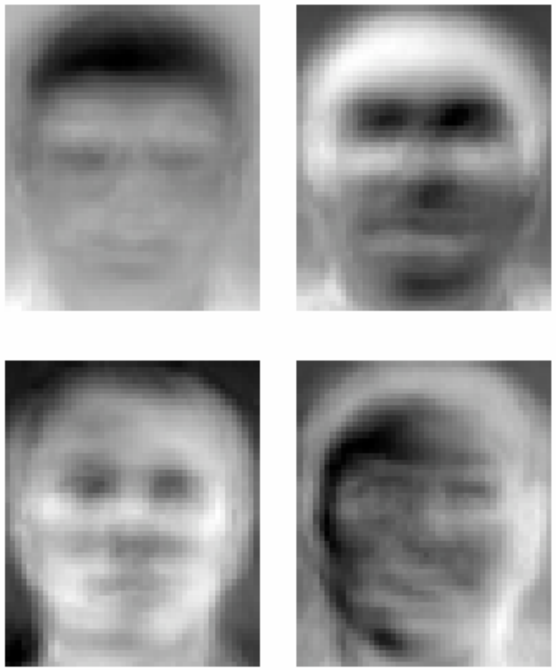
- A very well-known example of the use of PCA in pattern recognition is eigenfaces: a face recognition system, where faces are represented by their eigenvectors.



Building the Eigenfaces

1. Collect a large number of digital images of faces taken under the same lighting conditions.
2. Normalize the images so that the eyes and mouths line up.
3. Treat each normalized face image as a signal vector $\vec{f}$.
4. Construct the covariance matrix Q of the distribution of all the faces in the database.
5. Compute the eigenvectors of Q.
6. These eigenvectors are the *eigenfaces*.

What is an Eigenface?



- Each of the eigenfaces looks like a blurred average human face.
- Each eigenface describes a different property that discriminates one face from another.
- Note the absence of any gender-related attributes.
- Eigenfaces can be thought of as the *standardized face ingredients* which are derived from the statistical analysis of many pictures of human faces.
- A human face can be considered a combination of these standard faces.

Face Recognition

- The eigefaces constitute a basis set of vectors for faces.
- This means that any human face can be represented as a weighted sum of eigenfaces.
- Once the eigenfaces are constructed, one only needs to store the weights (the coefficients) for a particular face.
- A face can be accurately reconstructed from the eigenface coefficients.
- The coefficients themselves can be used for recognition.
- The larger the number of eigenfaces, the more accurate the face reconstruction.

Face Reconstruction

Reconstructing a face from the first N components (eigenfaces)

Adding 1 additional PCA component at each step

In this next image, we show a similar picture, but with each additional face representing an additional 8 principle components. You can see that it takes a rather large number of images before the picture looks totally correct.

Adding 8 additional PCA components at each step

Limitations of Eigenfaces

- Variations in lighting conditions
  - Different lighting conditions for enrolment and query.
  - Bright light causing image saturation.
- Differences in pose
  - When the face appears in different orientations, the 2D feature distances get distorted.
- Expression
  - When the facial expression changes (smile, surprise, etc.) the feature location and shape change.

PCA in Imaging

- PCA has been widely used in general pattern recognition problems for many years.
- However, its application in image processing/ analysis where the entire image is treated as a signal has been avoided.
- Why? It can lead to huge covariance matrices.
- Consider a 1024x1024 image: $\vec{f} \in R^N$, where $N = 2^{20}$
- The covariance matrix *Q* is an *N*x*N* matrix, i.e. it has about 1 trillion entries.
- If each entry is 1 Byte, then one needs 1000GB just to store Q.

Covariance Matrix of Image Data

- Recall that $Q = \sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{f}_i - \vec{f}_j\right)\left(\vec{f}_i - \vec{f}_j\right)^T$ where $\vec{f} \in R^N$.
- Let F be a row vector, where each column is $\left(\vec{f}_i - \vec{f}_j\right)$

$$F = \left[\left(\vec{f}_1 - \vec{f}_1\right) \; \left(\vec{f}_1 - \vec{f}_2\right) \; \cdots \; \left(\vec{f}_i - \vec{f}_j\right) \; \left(\vec{f}_i - \vec{f}_{j+1}\right)\cdots \; \cdots \; \left(\vec{f}_K - \vec{f}_{K-1}\right) \; \left(\vec{f}_K - \vec{f}_K\right)\right]$$

  where F is an NxK^2 matrix.
- Then we can rewrite Q as: $Q = FF^T$
- Recall that computing the PCA transformation matrix involves solving the eigenproblem:

$$Q\vec{\varphi}_k = \lambda\vec{\varphi}_k$$

- This can now be rewritten as:

$$FF^T\vec{\varphi}_k = \lambda\vec{\varphi}_k$$

Covariance Matrix of Image Data – cont.

- Can we "play around" with $FF^T\vec{\varphi}_k = \lambda\vec{\varphi}_k$ to make somehow the PCA computation more space efficient?
- Let's multiply to the left with F^T:

$$F^T FF^T\vec{\varphi}_k = \lambda F^T\vec{\varphi}_k$$

$$\left(F^T F\right)\left(F^T\vec{\varphi}_k\right) = \lambda\left(F^T\vec{\varphi}_k\right)$$

$$\left(F^T F\right)\vec{\psi}_k = \lambda\vec{\psi}_k \quad \text{, where } \vec{\psi}_k = F^T\vec{\varphi}_k$$

- We now have another eigenproblem, but the matrix F^TF is a K^2xK^2 matrix (instead of the original *N*x*N* matrix).
- Now the matrix we need to diagonalize depends on the **number of samples** and not their dimension.

Computing the Correct Eigenvectors

- However, the eigenproblem that is now being solved is

$$\left(F^T F\right)\vec{\psi}_k = \lambda\vec{\psi}_k$$

- Our goal is to compute $\vec{\varphi}_k$ not $\vec{\psi}_k$.
- Let's multiply to the left with *F* this time:

$$F\left(F^T F\right)\vec{\psi}_k = \lambda F\vec{\psi}_k$$

$$\underbrace{FF^T}_{Q}\left(F\vec{\psi}_k\right) = \lambda\left(F\vec{\psi}_k\right)$$

- So, $F\vec{\psi}_k$ is an eigenvector of the original matrix.
- Thus, the eigenvectors of F^TF can be ***lifted*** to the eigenvectors of $Q=FF^T$ by left multiplication by *F*.

PCA Computation on Images

1. Construct a K^2xN matrix F that contains all possible pairs of differences between the samples.

$$F = \left[\left(\vec{f}_1 - \vec{f}_1\right) \quad \left(\vec{f}_1 - \vec{f}_2\right) \quad \cdots \quad \left(\vec{f}_i - \vec{f}_j\right) \quad \left(\vec{f}_i - \vec{f}_{j+1}\right) \cdots \quad \cdots \quad \left(\vec{f}_K - \vec{f}_{K-1}\right) \quad \left(\vec{f}_K - \vec{f}_K\right) \right]$$

2. Compute the eigenvalues and eigenvectors of $F^T F$ which is a K^2xK^2 matrix

3. *Lift* the computed eigenvalues and eigenvectors by left multiplying them by F.

Other Analytic Feature Extraction Methods

- Main idea behind analytic methods for feature computation is to:

  Find a linear transformation of the pattern so that an optimality criterion is satisfied.

- In PCA the optimality criterion is to maximize the spread of the resulting feature vectors over all the samples.
- Keep in mind that the optimality criteria should ultimately lead to good pattern recognition rates.
- Other reasonable criteria?

Good Feature Distribution

- For good classification results we often want:

A. Feature vectors of the same class to be clustered tightly together, to form compact clusters. In other words, within the same class we want small intra-class distance.

B. Feature vectors from different classes to be spread far apart from each other, to be easily separable. In other words, between different classes we want large inter-class distance.

Intra-class Distance

- A measure of intra-class distance is:

$$s_2(\Phi) = \sum_{\kappa=1}^{C}\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\vec{c}_i^{\,\kappa} - \vec{c}_j^{\,\kappa}\right)^T\left(\vec{c}_i^{\,\kappa} - \vec{c}_j^{\,\kappa}\right)$$

minimize

$$= \sum_{\kappa=1}^{C}\sum_{i=1}^{K}\sum_{j=1}^{K}\left(\Phi^\kappa \vec{f}_i - \Phi^\kappa \vec{f}_j\right)^T\left(\Phi^\kappa \vec{f}_i - \Phi^\kappa \vec{f}_j\right)$$

where *C* is the number of classes and *K* is the number of data points.

- We want a transformation matrix Φ that minimizes $s_2()$.

Inter-class Distance

- A measure of inter-class distance is:

$$s_3(\Phi) = \sum_{\kappa=1}^{C} \sum_{\substack{\lambda=1 \\ \lambda\neq\kappa}}^{C} \sum_{i=1}^{K} \sum_{j=1}^{K} \left(\vec{c}_i^{\,\kappa} - \vec{c}_j^{\,\lambda}\right)^T \left(\vec{c}_i^{\,\kappa} - \vec{c}_j^{\,\lambda}\right)$$

maximize

$$= \sum_{k=1}^{C} \sum_{\substack{\lambda=1 \\ \lambda\neq\kappa}}^{C} \sum_{i=1}^{K} \sum_{j=1}^{K} \left(\Phi^{\kappa}\vec{f}_i - \Phi^{\lambda}\vec{f}_j\right)^T \left(\Phi^{\kappa}\vec{f}_i - \Phi^{\lambda}\vec{f}_j\right)$$

where *C* is the number of classes and *K* is the number of data points.

- We want a transformation matrix Φ that maximizes $s_3()$.

Combo of Intra- and Inter-class Distance

- Ideally we would like to have both minimal intra-class and maximal interclass distance.
- We could combine these two criteria in a single minimization function using a Lagrange multiplier.

$$s_4(\Phi) = s_2(\Phi) - \lambda s_3(\Phi)$$

minimize

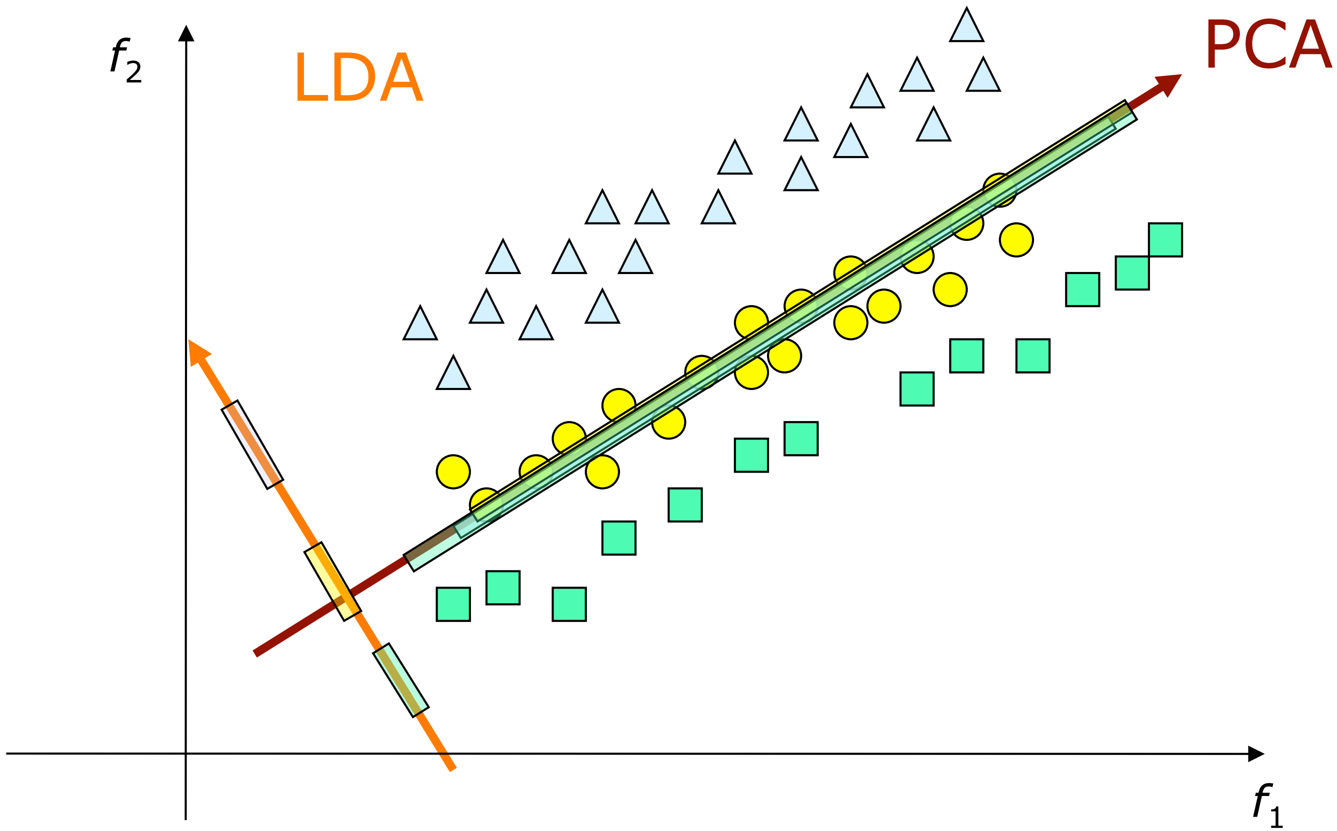
- Alternatively, the intra- and inter-class distance can be combined using ratios:

$$s_5(\Phi) = {s_3(\Phi)}/{s_2(\Phi)}$$

maximize

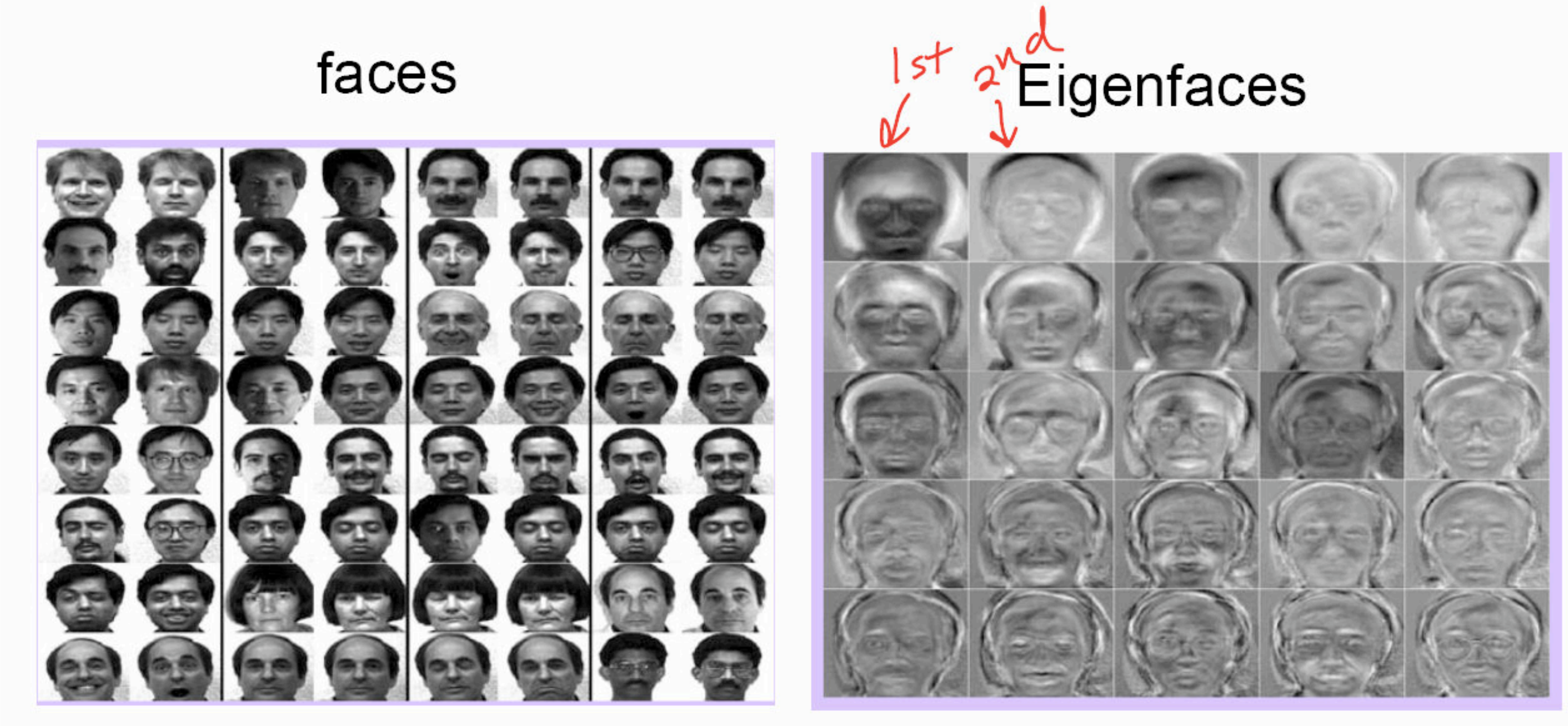
- $s_5(\Phi)$ is also known as Rayleigh Quotient and used in Linear Discriminant Analysis (LDA)
- The resulting Φ is also known as the Fisher Transform.

PCA versus LDA



LDA Example: Fisherfaces

- LDA was also applied on face recognition in order to overcome some of the problems of eigenfaces. The resulting method is known as fisherfaces.



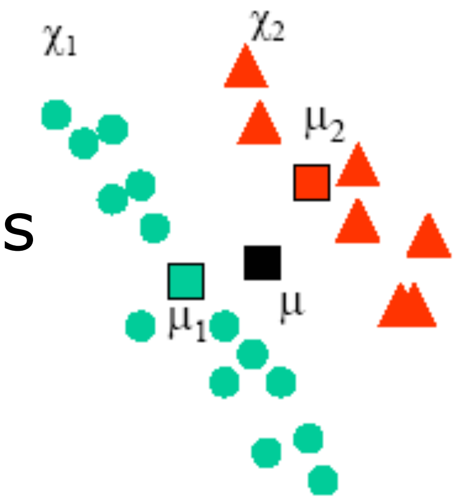
Computing the Fisherfaces

- Let X_1, X_2,…, X_c be the face classes (distinct faces) in the database.
- For each face class X_i, i = 1,2,…,c there are k facial images x_j, j=1,2,…,k.
- Compute the mean image μ_i of each class X_i (i,e, the average face per person):

$$\mu_i = \frac{1}{k}\sum_{j=1}^{k} x_j$$

- The mean image μ of all the classes in the database can be calculated as:

$$\mu = \frac{1}{c}\sum_{i=1}^{c} \mu_i$$



Computing the Fisherfaces – Scatter Matrix

- As a measure of intra-class variation, compute the within-class scatter matrix:

$$S_W = \sum_{i=1}^{c} \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T$$

- As a measure of inter-class variation, compute the between-class scatter matrix:

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu)(\mu_i - \mu)^T$$

- We find the product of S_W^{-1} and S_B and then compute the eigenvectors and eigenvalues of this product ($S_W^{-1} . S_B$)

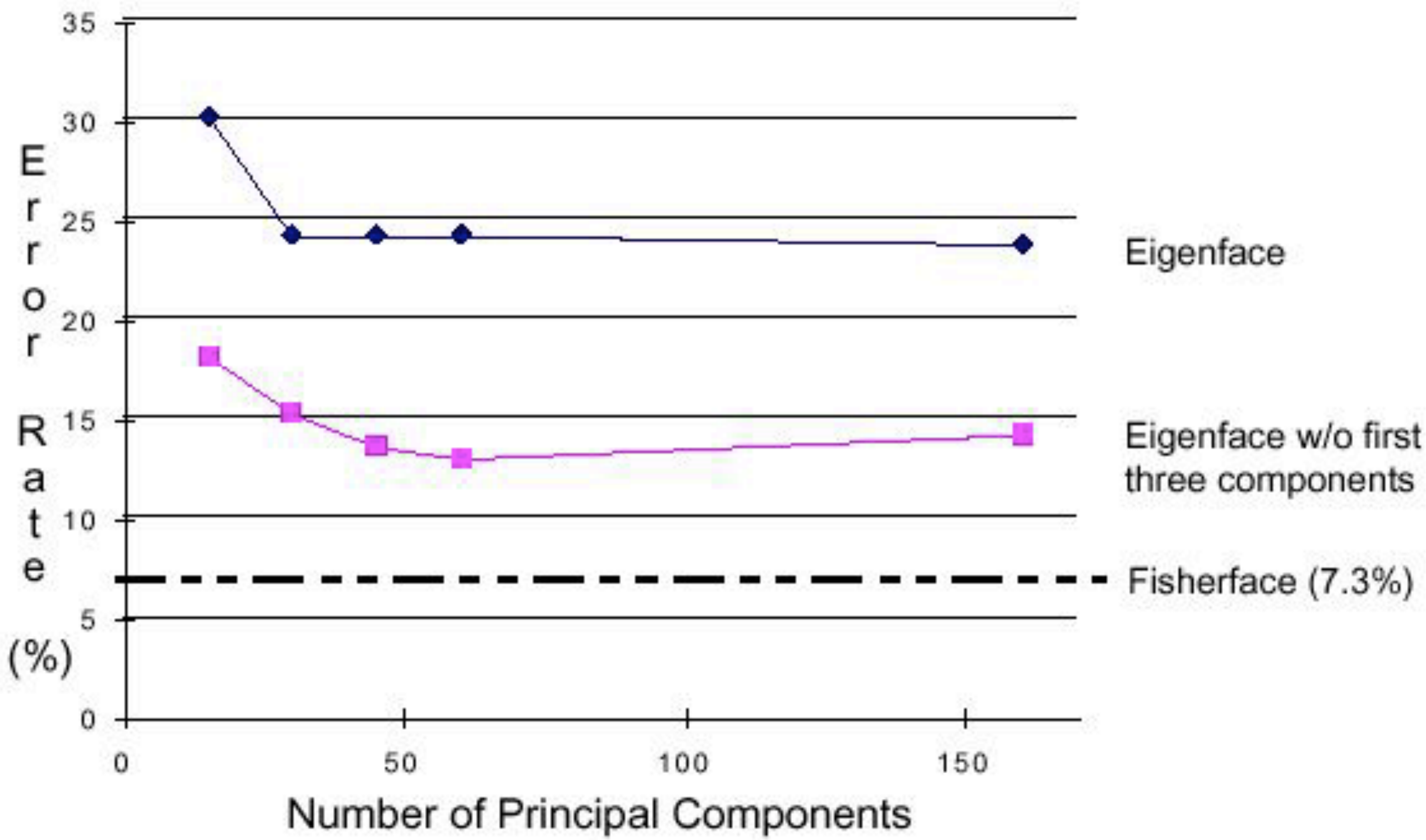
Sample Fisherface

- All possible combinations of 159+1 were tested.

Evaluation of Fisherfaces vs. Eigenfaces

- At the University of Illinois at Urbana Champaign they evaluated fisherfaces against eigenfaces.
- The face database contained 160 images of 16 people.
- For each person, there were 10 images:
  - One with and one without glasses
  - Three different lighting conditions
  - Five different facial expressions
- 159 images were used for training, 1 was used for testing/evaluation. All possible combinations of 159+1 were tested.

Fisherfaces vs. Eigenfaces



Resources

1. The 2D PCA example is courtesy of D. James http://www.cs.cornell.edu/courses/cs322/2008sp/schedule.html
2. The 3D PCA example is from the website of Miner3D http://www.miner3d.com/products/pca.html
3. The eigenface material is based on the slides of Z. B. Joseph http://www.cs.cmu.edu/~zivbj/class/10701/lecture/lec21.pdf
4. The fisherface material is based on the slides of p. Buddharaju http://www2.cs.uh.edu/~rmverma/InformationAssurance_Module3/Biometrics_Lecture3/COSC_6397-Lecture3.ppt
5. The comparison between Fisherfaces and Eigenfaces is courtesy of H. Wang http://courses.engr.illinois.edu/ece598/ffl/paper_presentations/HongchengWang.pdf