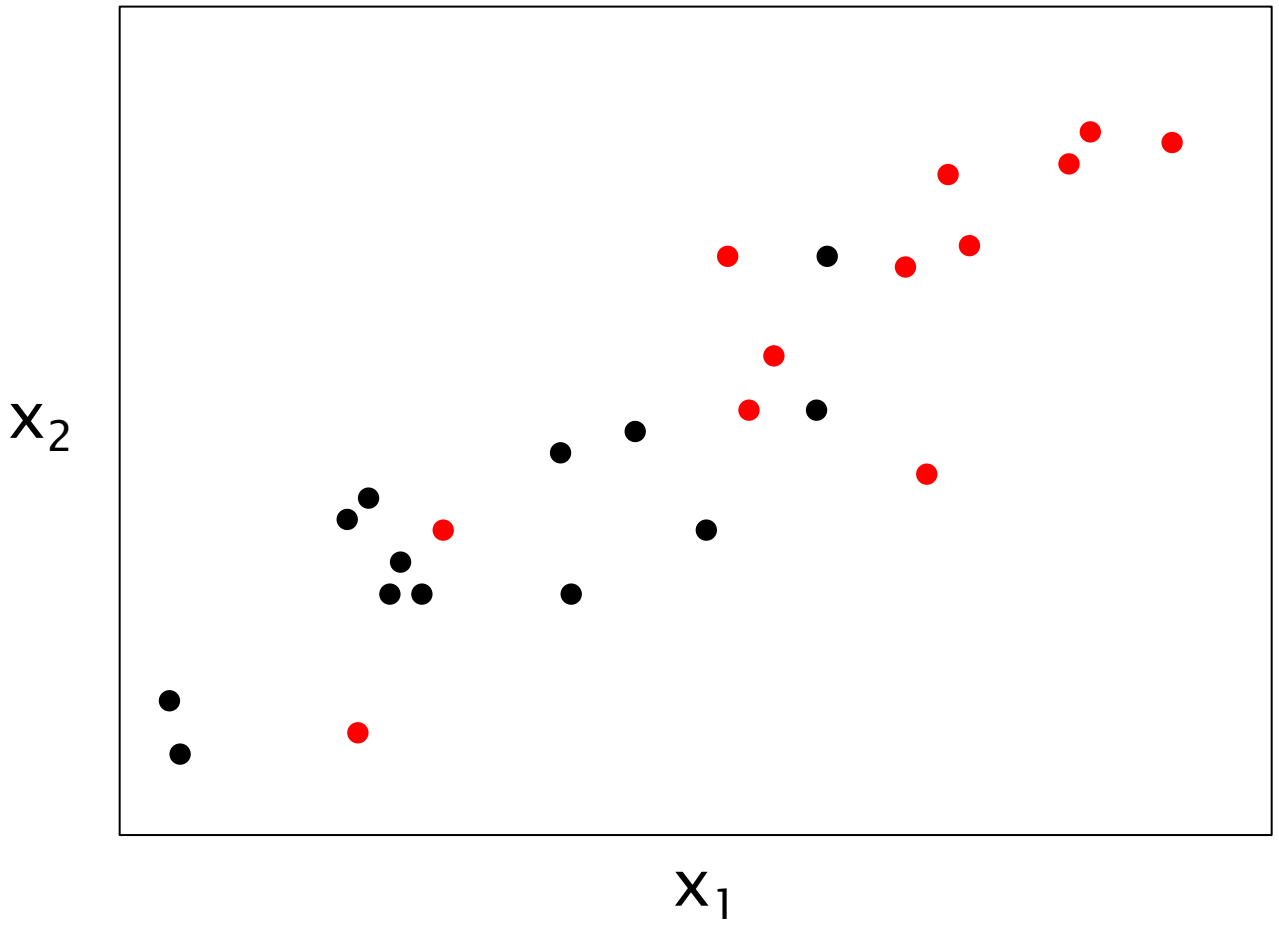
Dimensionality Reduction by Feature Extraction

Outline:

Principal Component Analysis
Independent Component Analysis
ISOMAP

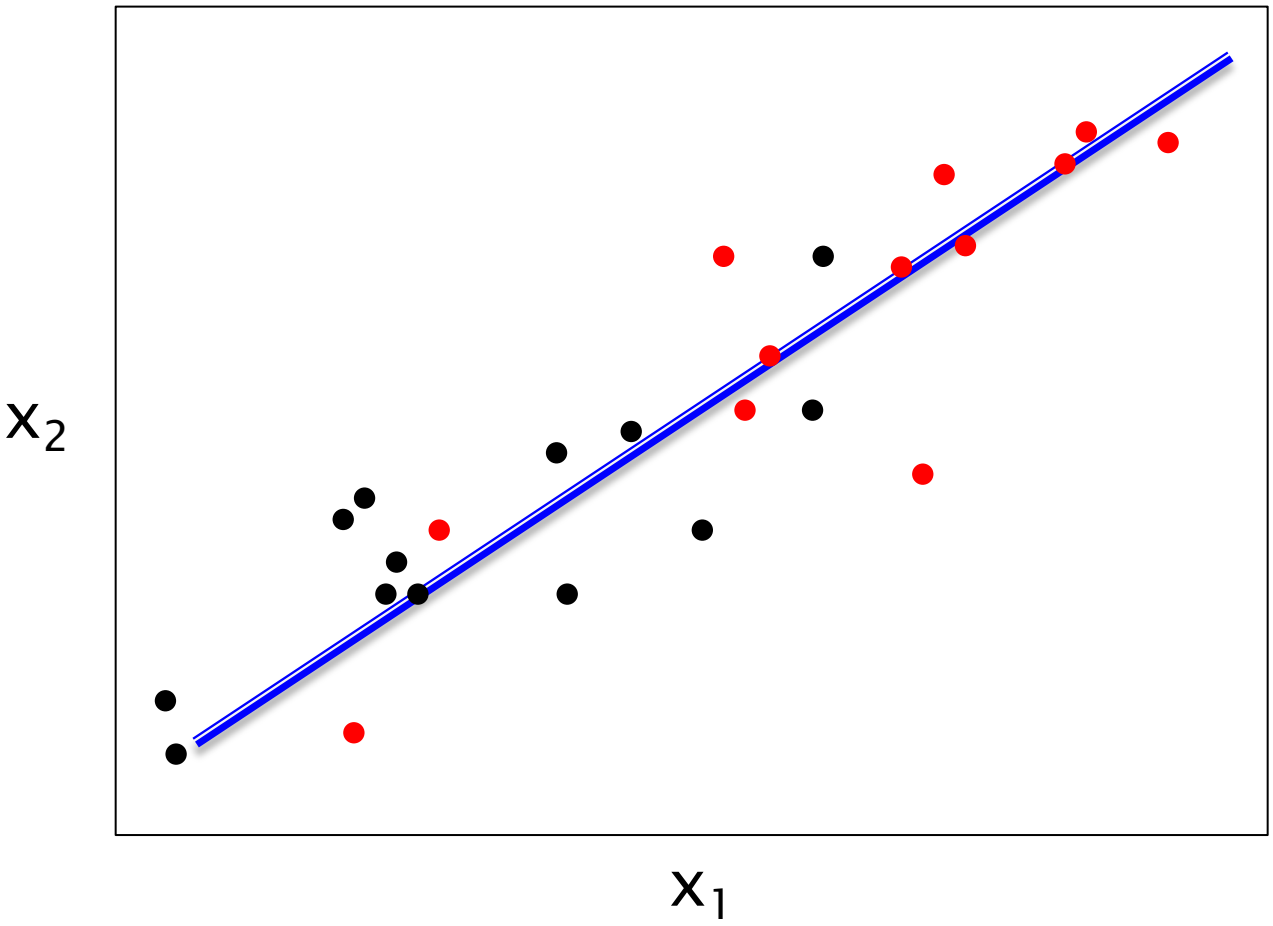
Data



Data

First principal component

Gives direction of
largest variation of the data



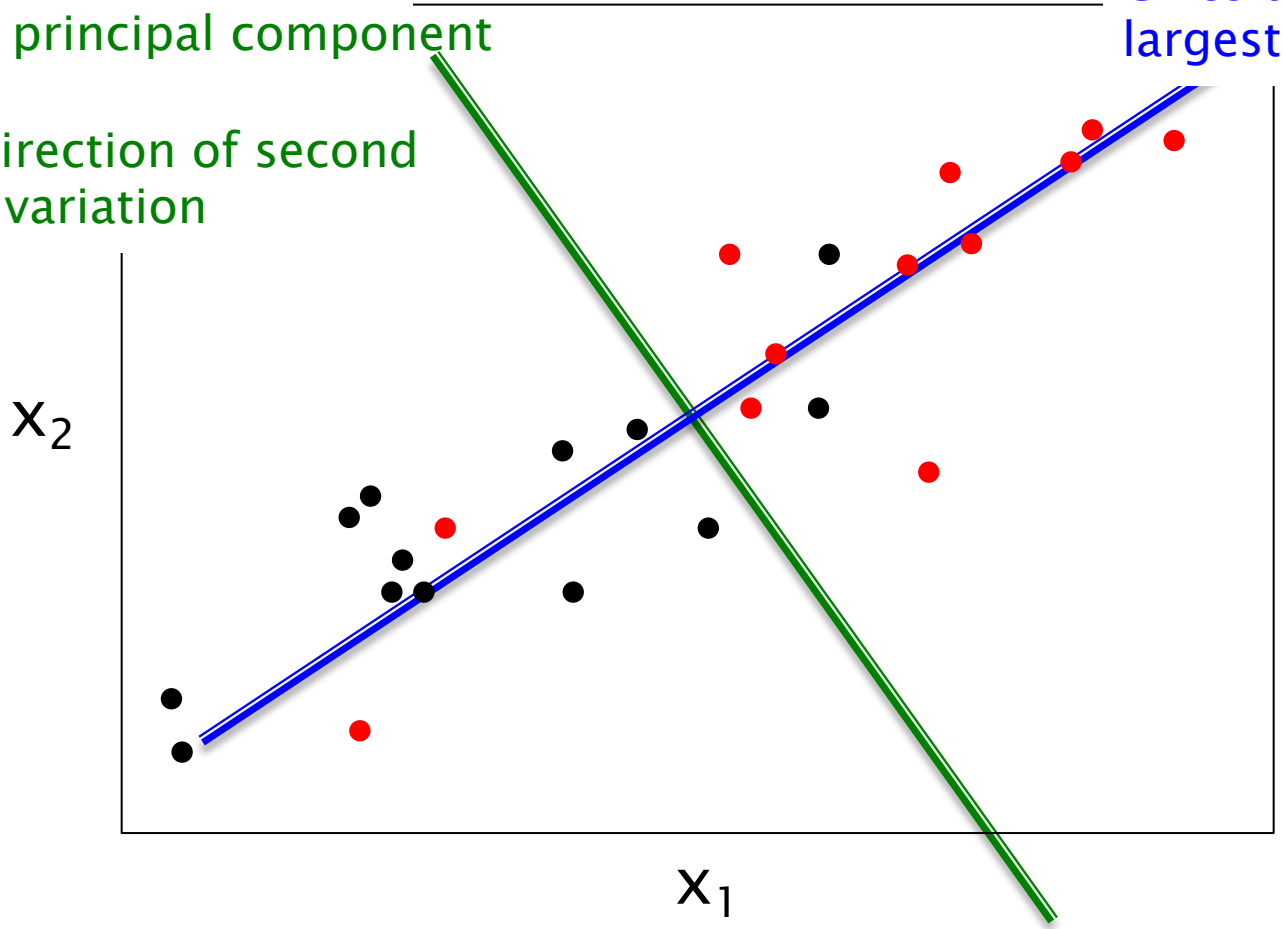
Data

First principal component

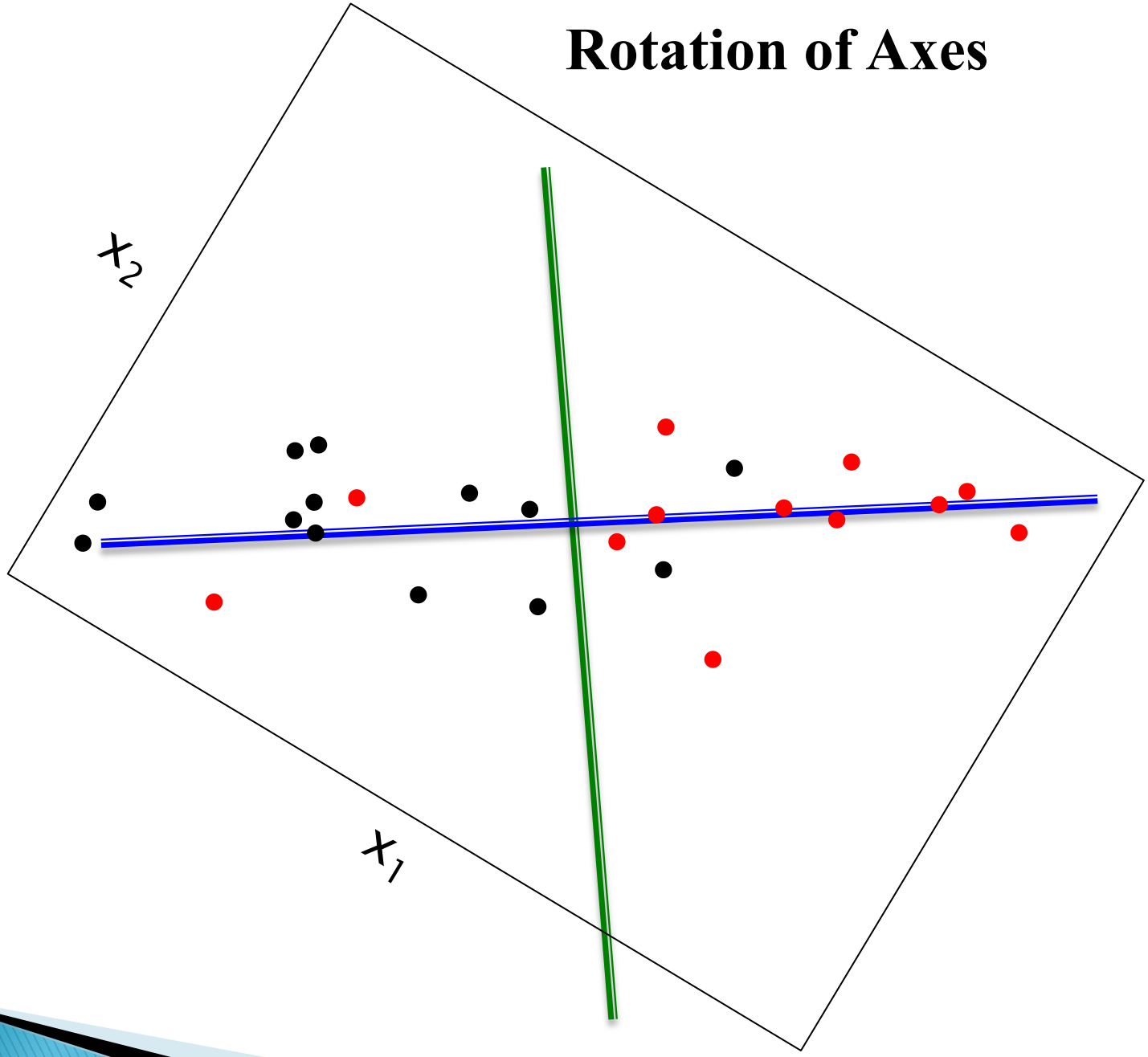
Gives direction of
largest variation of the da

Second principal component

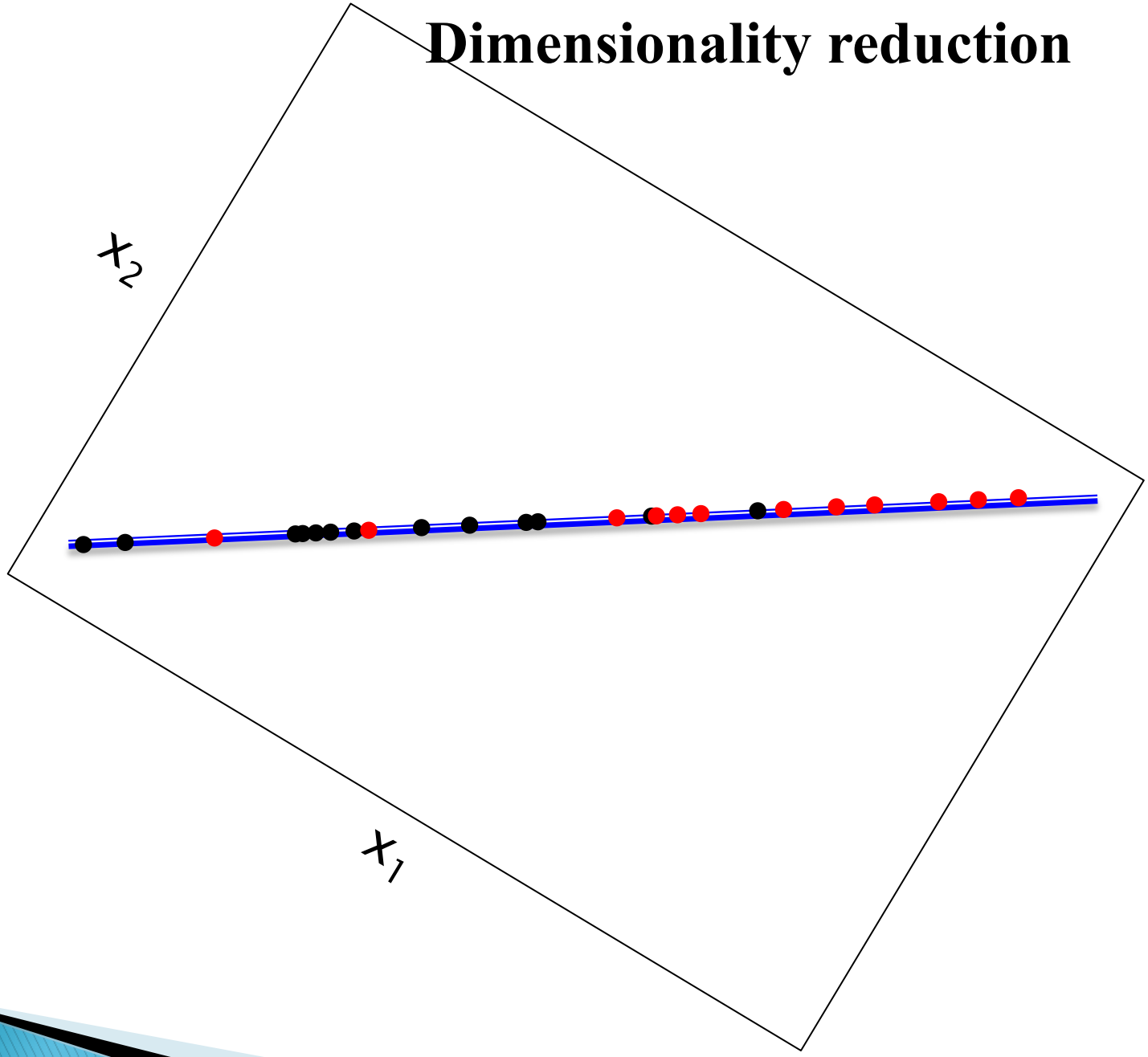
Gives direction of second
largest variation



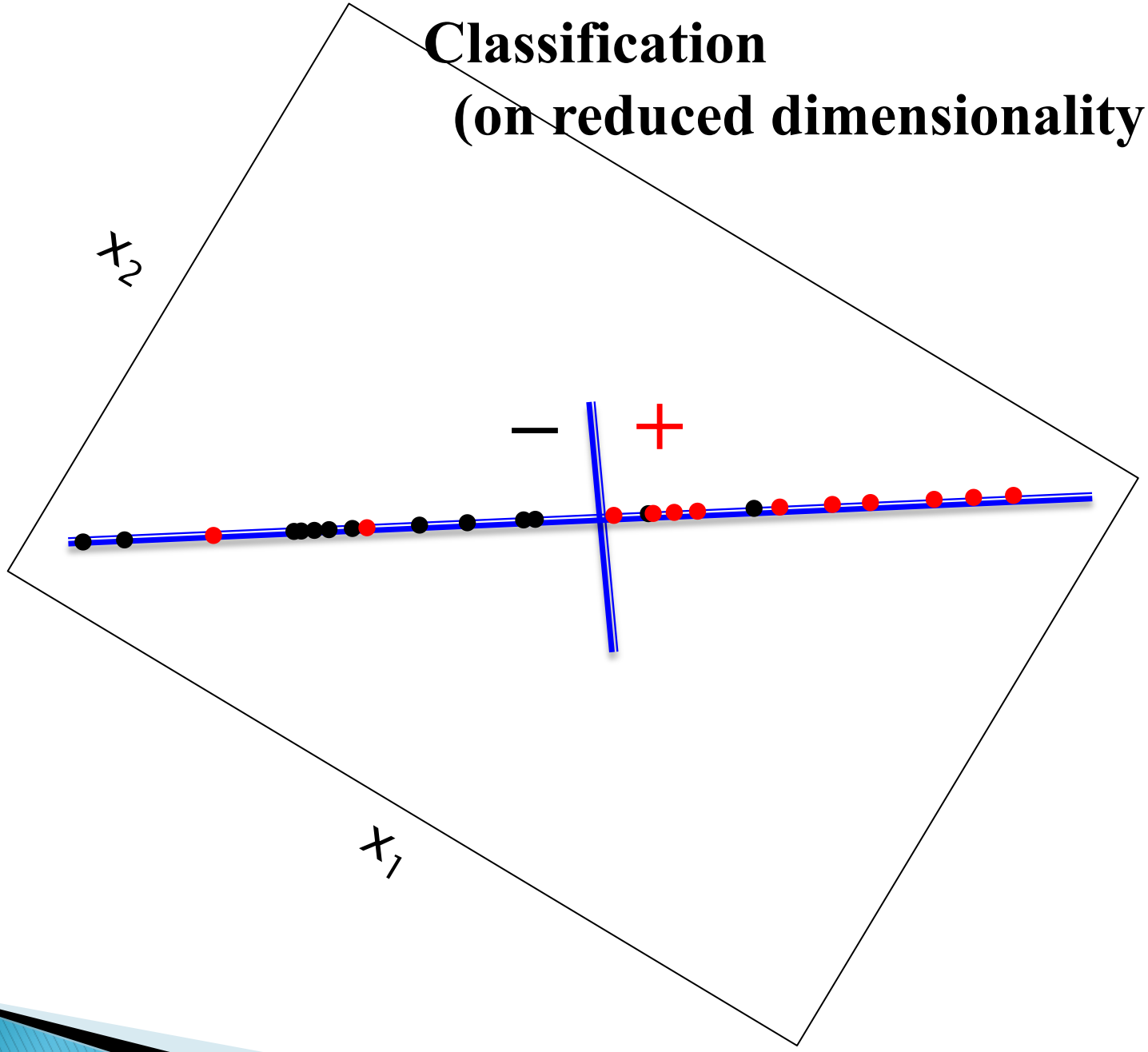
Rotation of Axes



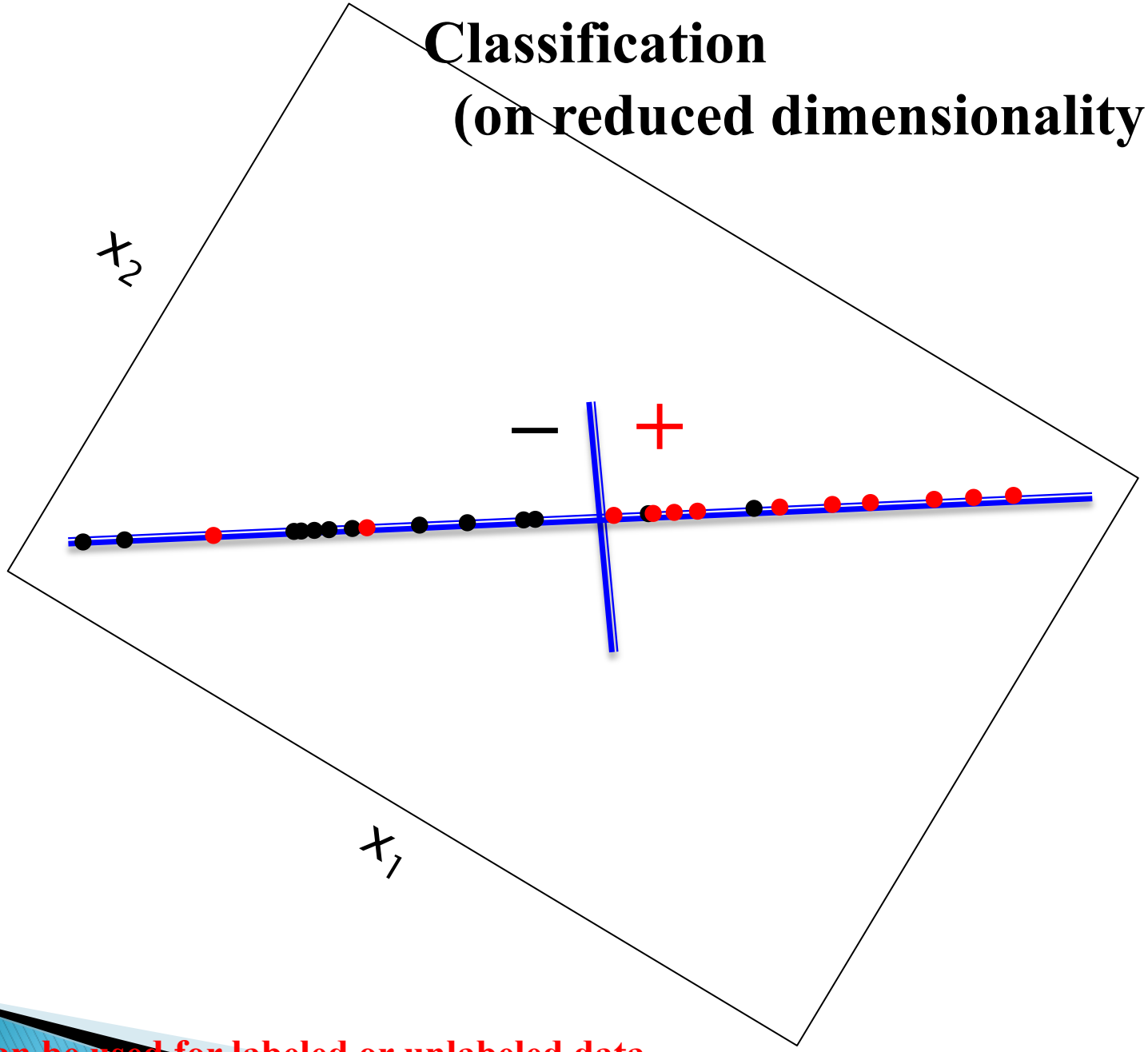
Dimensionality reduction



Classification
(on reduced dimensionality space)



Classification
(on reduced dimensionality space)



Note: Can be used for labeled or unlabeled data.

Principal Components Analysis (PCA)

- **Summary:** PCA finds new orthogonal axes in directions of largest variation in data.
- PCA used to create high-level features in order to improve classification and reduce dimensions of data without much loss of information.
- Used in machine learning and in signal processing and image compression (among other things).

Background for PCA

- Suppose attributes are A_1 and A_2, and we have n training examples. x's denote values of A_1 and y's denote values of A_2 over the training examples.

- Variance of an attribute:

$$\text{var}(A_1) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{(n-1)}$$

- Covariance of two attributes:

$$\mathrm{cov}(A_1, A_2) = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{(n-1)}$$

- If covariance is positive, both dimensions increase together. If negative, as one increases, the other decreases. Zero: independent of each other.

- Covariance matrix
  - Suppose we have n attributes, A_1, ..., A_n.
  - Covariance matrix:

$$C^{n \times n} = (c_{i,j}), \text{ where } c_{i,j} = \mathrm{cov}(A_i, A_j)$$

	Hours(H)	*Mark(M)*
Data	9	39
	15	56
	25	93
	14	61
	10	50
	18	75
	0	32
	16	85
	5	42
	19	70
	16	66
	20	80
Totals	167	749
Averages	13.92	62.42

Covariance:

H	M	$(H_i - \bar{H})$	$(M_i - \bar{M})$	$(H_i - \bar{H})(M_i - \bar{M})$
9	39	-4.92	-23.42	115.23
15	56	1.08	-6.42	-6.93
25	93	11.08	30.58	338.83
14	61	0.08	-1.42	-0.11
10	50	-3.92	-12.42	48.69
18	75	4.08	12.58	51.33
0	32	-13.92	-30.42	423.45
16	85	2.08	22.58	46.97
5	42	-8.92	-20.42	182.15
19	70	5.08	7.58	38.51
16	66	2.08	3.58	7.45
20	80	6.08	17.58	106.89
Total				1149.89
Average				104.54

Table 2.2: 2-dimensional data set and covariance calculation

$$\begin{pmatrix} \mathrm{cov}(H,H) & \mathrm{cov}(H,M) \\ \mathrm{cov}(M,H) & \mathrm{cov}(M,M) \end{pmatrix}$$

$$= \begin{pmatrix} \mathrm{var}(H) & 104.5 \\ 104.5 & \mathrm{var}(M) \end{pmatrix}$$

$$= \begin{pmatrix} 47.7 & 104.5 \\ 104.5 & 370 \end{pmatrix}$$

Covariance matrix

Review of Matrix Algebra

- **Eigenvectors:**
  - Let **M** be an $n \times n$ matrix.
    - **v** is an *eigenvector* of **M** if $\mathbf{M} \times \mathbf{v} = \lambda\mathbf{v}$
    - λ is called the *eigenvalue* associated with **v**
  - You can always choose eigenvectors of length 1:

$$\mathbf{M} \times a\mathbf{v} = \lambda a\mathbf{v}$$

  - If **M** is symmetric with real entries, it has n eigenvectors, and they are orthogonal to one another.
  - Thus, eigenvectors can be used as a new basis for a n-dimensional vector space.

Textbook's notation

- We have original data **X** and mean-subtracted data **B**, and covariance matrix **C** = *cov* (**B**), where ***C*** is an $N \times N$ matrix:

$$\mathbf{C} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T$$

- We find matrix **V** such that the columns of **V** are the N eigenvectors of **C** and

$$\mathbf{D} = \mathbf{V}^{-1}\mathbf{C}\,\mathbf{V} = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \lambda_N \end{pmatrix}$$

where λ_i is the i-th eigenvalue of **C.**

- Each eigenvalue in **D** corresponds to an eigenvector in **V.** The eigenvectors, sorted in order of decreasing eigenvalue, become the "feature vector" for PCA.

Principal Components Analysis

The Principal Components Analysis Algorithm

- Write N datapoints $\mathbf{x}_i = (\mathrm{x}_{1i}, \mathrm{x}_{2i}, \ldots, \mathrm{x}_{Mi})$ as row vectors
- Put these vectors into a matrix $\mathbf{X}$ (which will have size $N \times M$)
- Centre the data by subtracting off the mean of each column, putting it into matrix $\mathbf{B}$
- Compute the covariance matrix $\mathbf{C} = \frac{1}{N}\mathbf{B}^T\mathbf{B}$
- Compute the eigenvalues and eigenvectors of $\mathbf{C}$, so $\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D}$, where $\mathbf{V}$ holds the eigenvectors of $\mathbf{C}$ and $\mathbf{D}$ is the $M \times M$ diagonal eigenvalue matrix
- Sort the columns of $\mathbf{D}$ into order of decreasing eigenvalues, and apply the same order to the columns of V
- Reject those with eigenvalue less than some η, leaving L dimensions in the data

Principal Components Analysis

1. Given original data set $S = \{\mathbf{x}^1, ..., \mathbf{x}^k\}$, produce new set by subtracting the mean of attribute A_i from each x_i.

Data =

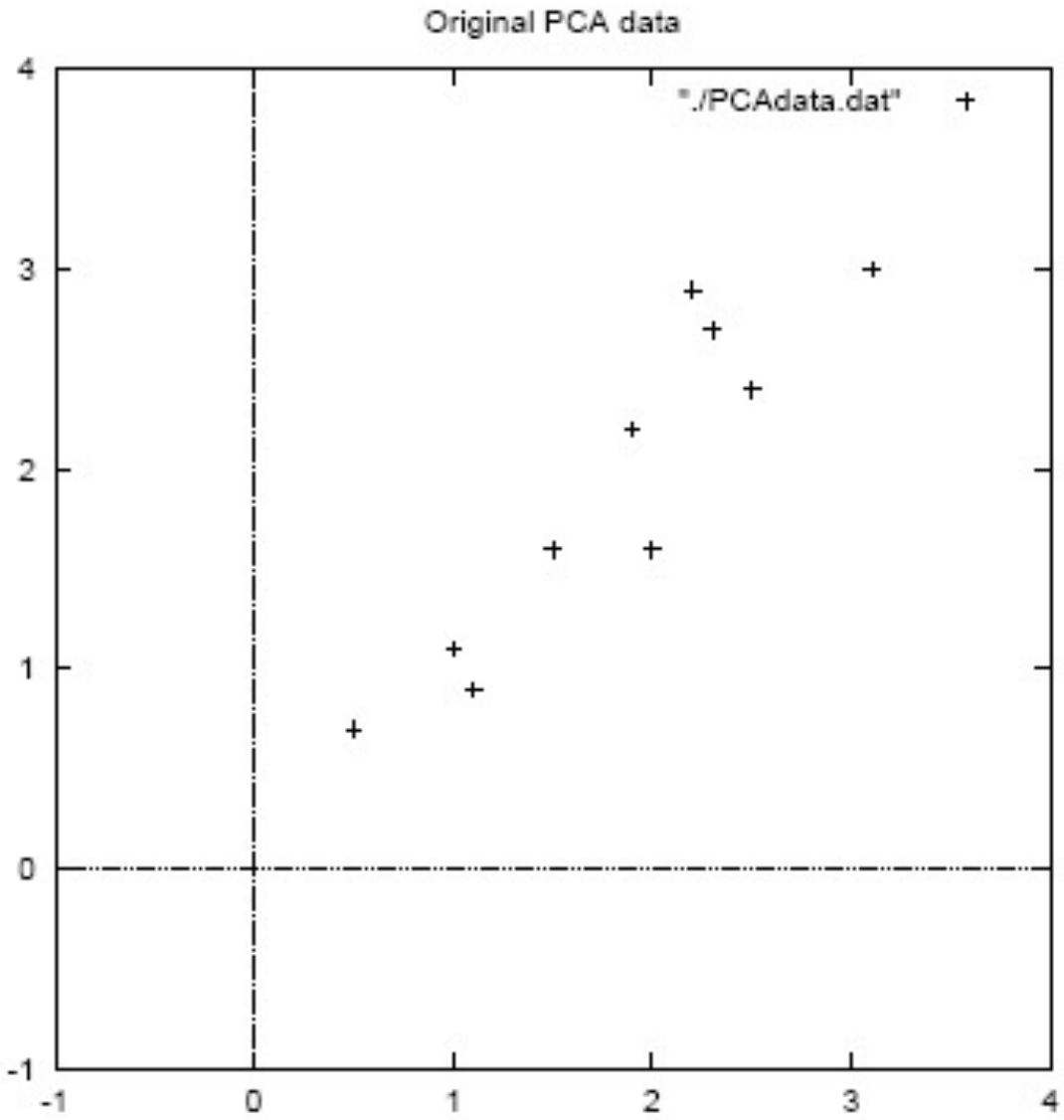
x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

Mean: 1.81 1.91

DataAdjust =

x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Mean: 0 0


Original PCA data
"./PCAdata.dat"
4
3
2
1
0
-1
-1
0
1
2
3
4

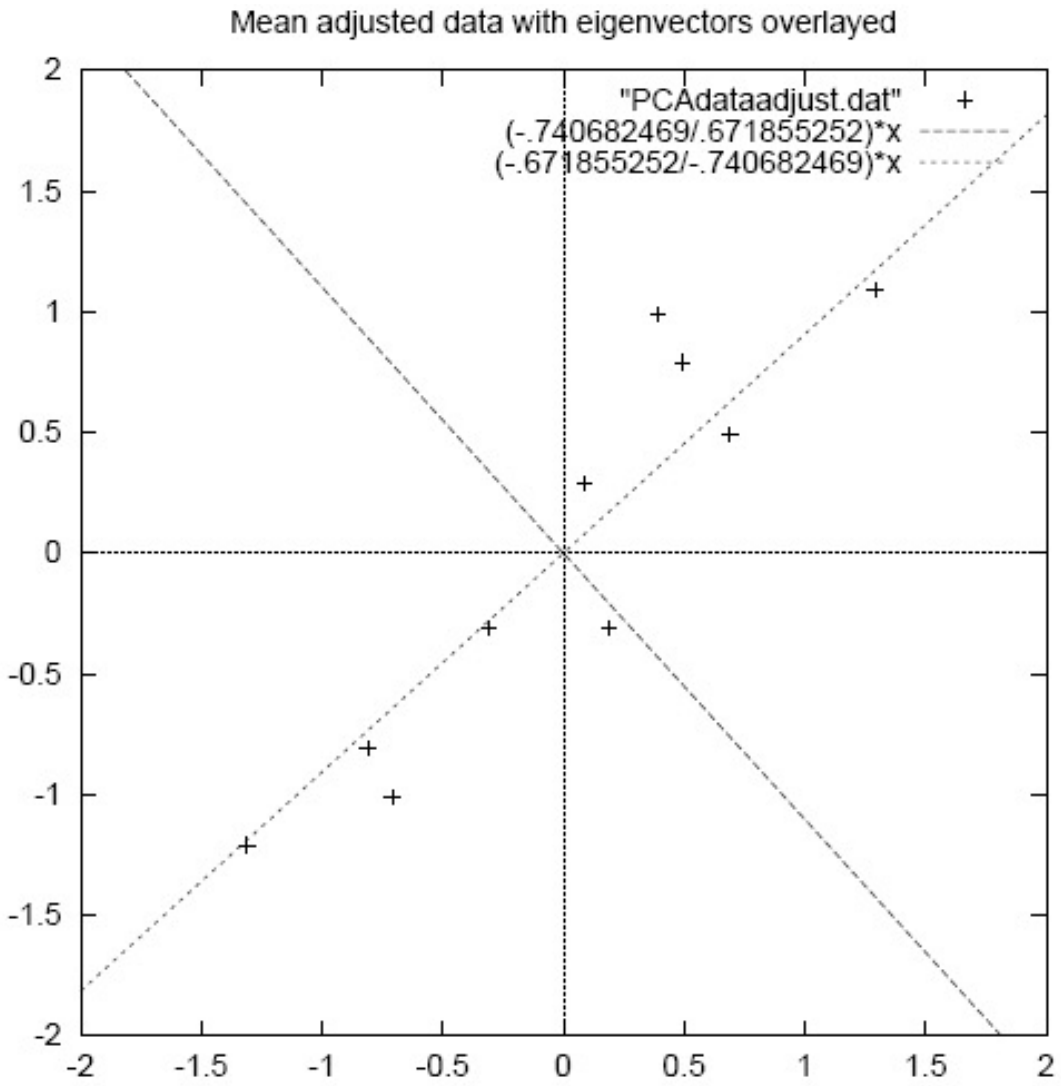
2. Calculate the covariance matrix:

$$cov = \begin{array}{c} \\ x \\ y \end{array} \begin{array}{c} \begin{array}{cc} x & y \end{array} \\ \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix} \end{array}$$

3. Calculate the (unit) eigenvectors and eigenvalues of the covariance matrix:

$$eigenvalues = \begin{pmatrix} .0490833989 \\ 1.28402771 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} -.735178656 & -.677873399 \\ .677873399 & -.735178656 \end{pmatrix}$$



Eigenvector with largest eigenvalue traces linear pattern in data

Figure 3.2: A plot of the normalised data (mean subtracted) with the eigenvectors of the covariance matrix overlayed on top.

4. Order eigenvectors by eigenvalue, highest to lowest.

$$\mathbf{v}_1 = \begin{pmatrix} -.677873399 \\ -.735178956 \end{pmatrix} \quad \lambda = 1.28402771$$

$$\mathbf{v}_2 = \begin{pmatrix} -.735178956 \\ .677873399 \end{pmatrix} \quad \lambda = .0490833989$$

In general, you get n components. To reduce dimensionality to p, ignore $n-p$ components at the bottom of the list.

Construct new "feature vector" (assuming $\mathbf{v}_i$ is a column vector).

Feature vector = ($\mathbf{v}_1$, $\mathbf{v}_2$, ...$\mathbf{v}_p$)

$$FeatureVector1 = \begin{matrix} \mathbf{v}_1 & \mathbf{v}_2 \\ \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix} \end{matrix}$$

or reduced dimension feature vector :

$$FeatureVector2 = \begin{pmatrix} -.677873399 \\ -.735178956 \end{pmatrix}$$

5. Derive the new data set.

$$TransformedData = RowFeatureVector \times RowDataAdjust$$

where *RowDataAdjust = transpose of mean-adjusted data*

$$RowFeatureVector1 = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

This gives original data in terms of chosen components (eigenvectors)—that is, along these axes.

$$RowFeatureVector2 = \begin{pmatrix} -.677873399 & -.735178956 \end{pmatrix}$$

$$RowDataAdjust = \begin{pmatrix} .69 & -1.31 & .39 & .09 & 1.29 & .49 & .19 & -.81 & -.31 & -.71 \\ .49 & -1.21 & .99 & .29 & 1.09 & .79 & -.31 & -.81 & -.31 & -1.01 \end{pmatrix}$$

	x	y
	-.827970186	-.175115307
	1.77758033	.142857227
	-.992197494	.384374989
	-.274210416	.130417207
Transformed Data=	-1.67580142	-.209498461
	-.912949103	.175282444
	.0991094375	-.349824698
	1.14457216	.0464172582
	.438046137	.0177646297
	1.22382056	-.162675287

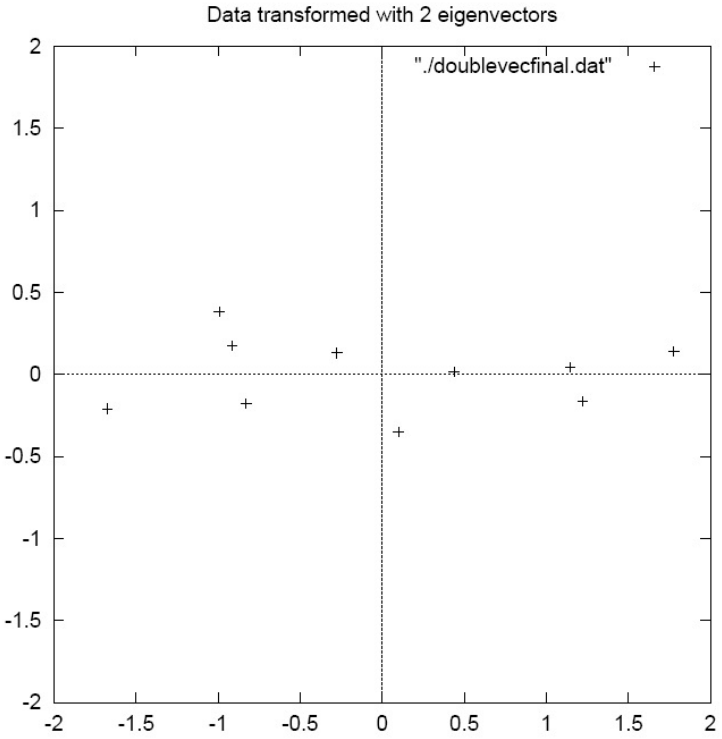
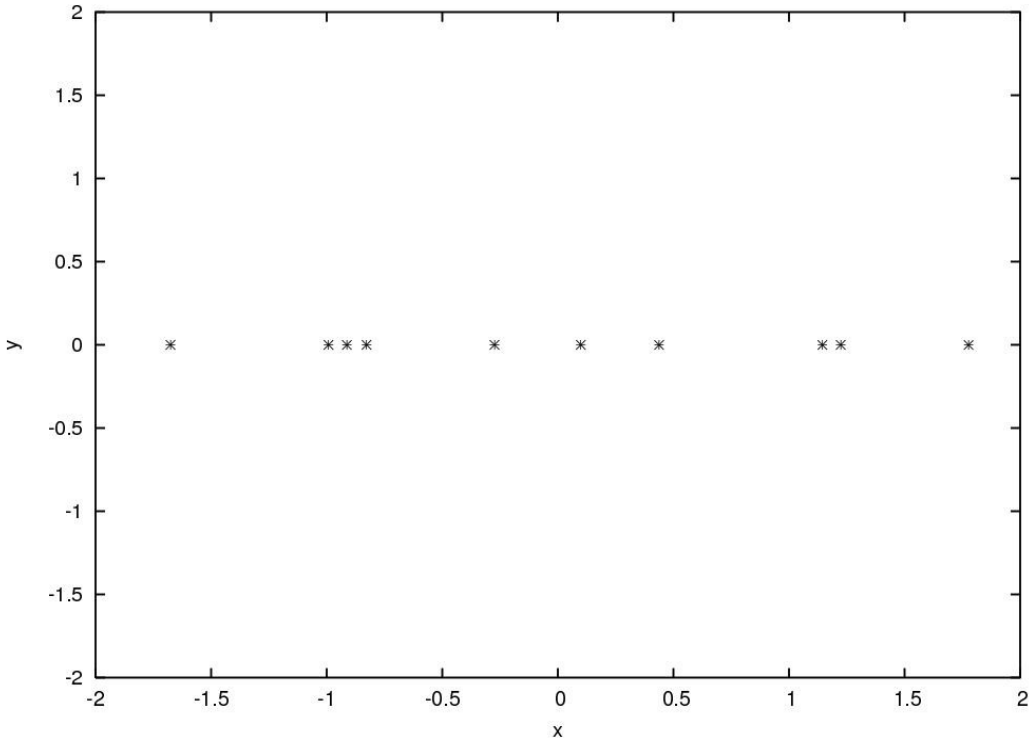


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

Intuition: We projected the data onto new axes that captures the strongest linear trends in the data set. Each transformed data point tells us how far it is above or below those trend lines.

Transformed Data (Single eigenvector)

x
-.827970186
1.77758033
-.992197494
-.274210416
-1.67580142
-.912949103
.0991094375
1.14457216
.438046137
1.22382056



For Image Compression, How Do we Decompress Data? How to Reconstruct the original data?

We did:

$$TransformedData = RowFeatureVector \times RowDataAdjust$$

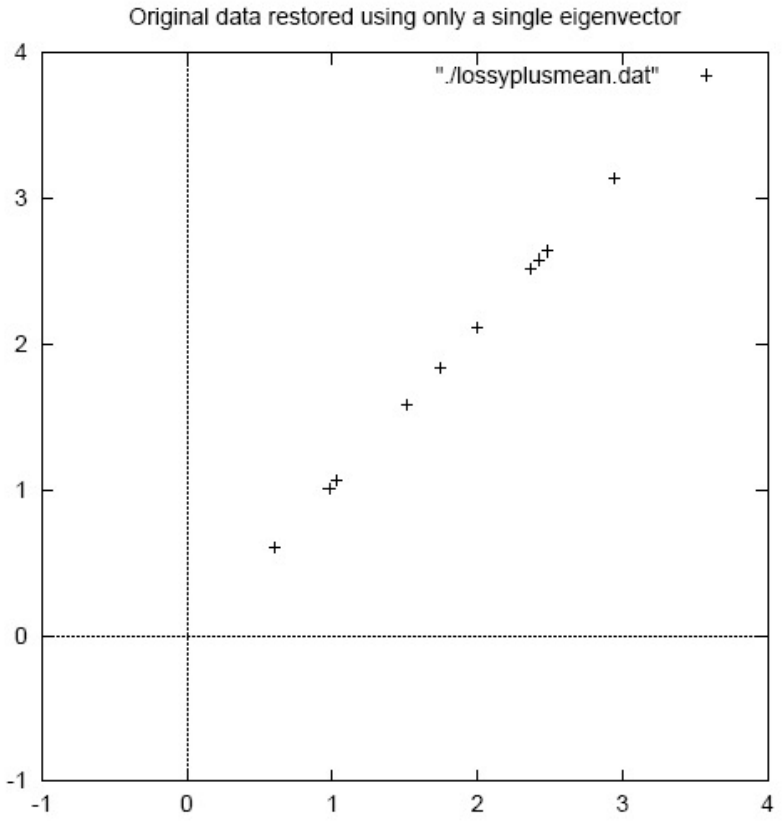
so we can do

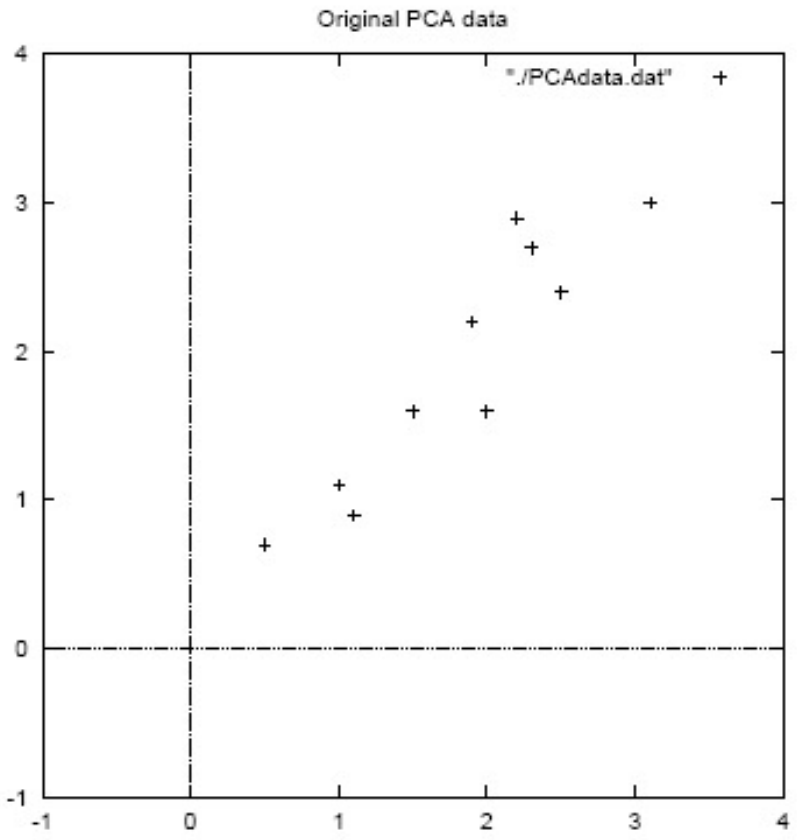
$$RowDataAdjust = RowFeatureVector^{-1} \times TransformedData$$

$$= RowFeatureVector^{T} \times TransformedData$$

and

$$RowDataOriginal = RowDataAdjust + OriginalMean$$





Information is lost.
In Image compression, this is idea of the KL transform.

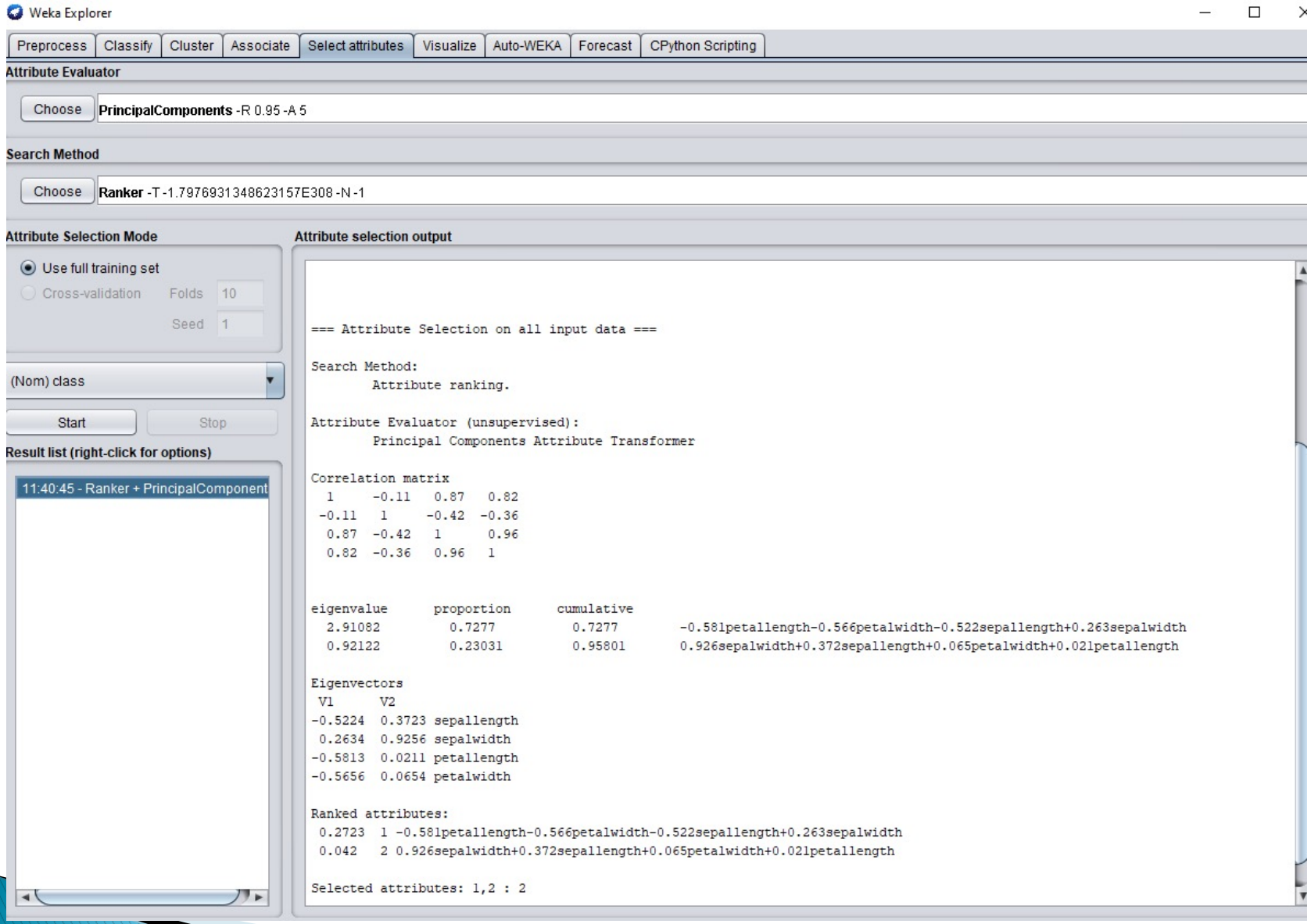
- With new data, compute $TransformedData = RowFeatureVector \times RowDataAdjust$

where $RowDataAdjust = transpose\ of\ mean\text{-}adjusted\ data$

What you need to remember

- General idea of what PCA does
  - Finds new, rotated set of orthogonal axes that capture directions of largest variation
  - Allows some axes to be dropped, so data can be represented in lower-dimensional space.
  - This can improve classification performance and avoid overfitting due to large number of dimensions.
- You don't need to remember details of PCA algorithm.

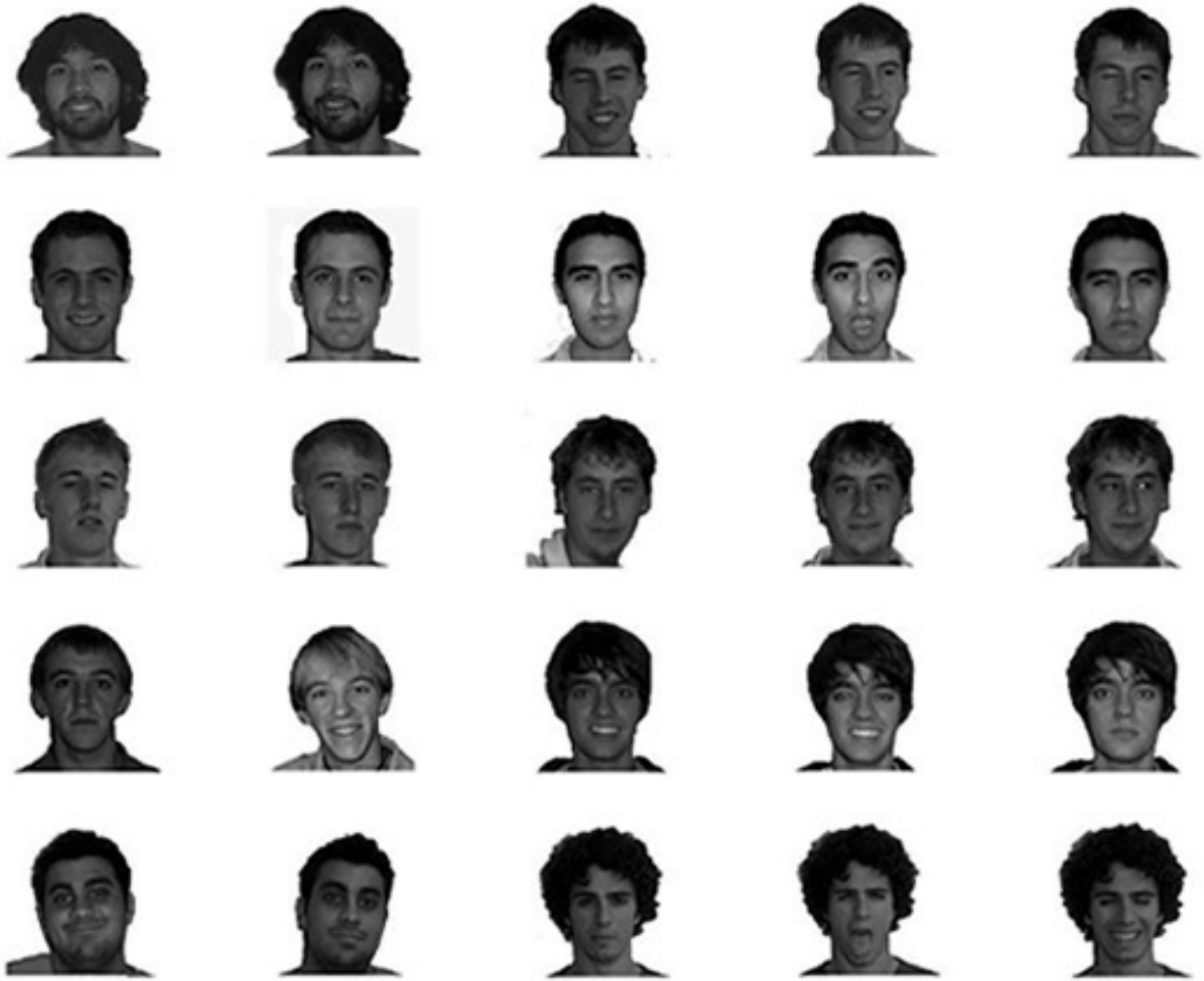
PCA WEKA on Iris data



Watch: https://www.youtube.com/watch?v=zuoMjUAPihA

Example: Linear discrimination using PCA for face recognition ("Eigenfaces")

1. Preprocessing: "Normalize" faces
   - Make images the same size
   - Line up with respect to eyes
   - Normalize intensities



2. Raw features are pixel intensity values (2061 features)

3. Each image is encoded as a vector Γ_i of these features

4. Compute "mean" face in training set:

$$\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

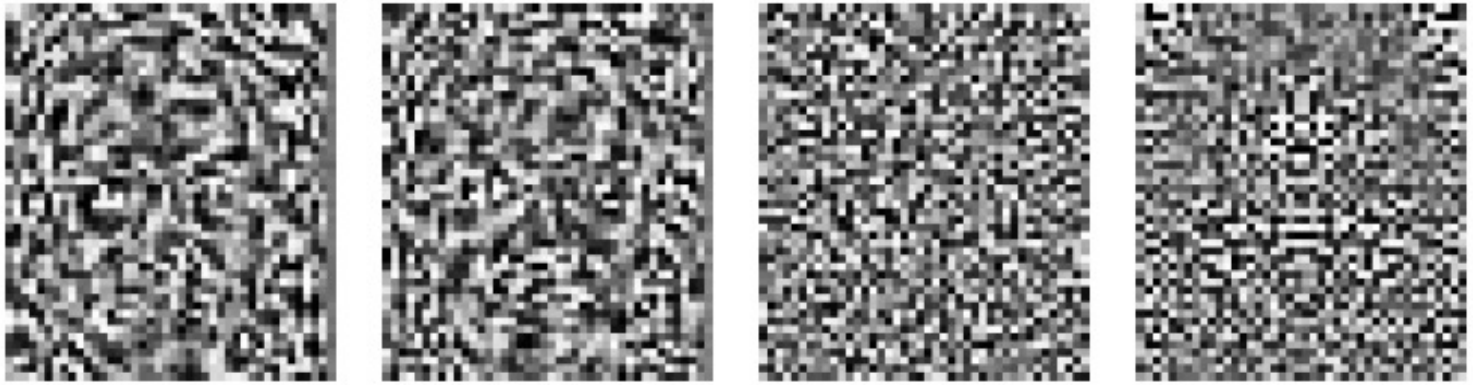
The average face and first four eigenfaces

- Subtract the mean face from each face vector

$$\Phi_i = \Gamma_i - \Psi$$

- Compute the covariance matrix **C**
- Compute the (unit) eigenvectors $\mathbf{v}_i$ of **C**
- Keep only the first *K* principal components (eigenvectors)

Eigenfaces 15, 100, 200, 250, 300



Eigenfaces 400, 450, 1000, 2000

Interpreting and Using Eigenfaces

The eigenfaces encode the principal sources of variation in the dataset (e.g., absence/presence of facial hair, skin tone, glasses, etc.).

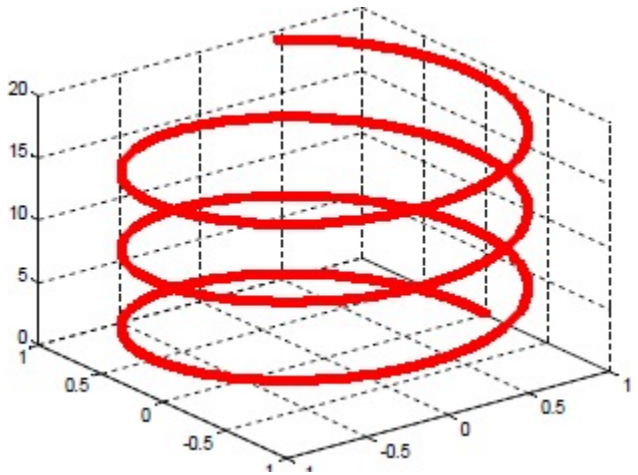
We can represent any face as a linear combination of these "basis" faces.

Use this representation for:

- Face recognition
  (e.g., Euclidean distance from known faces)

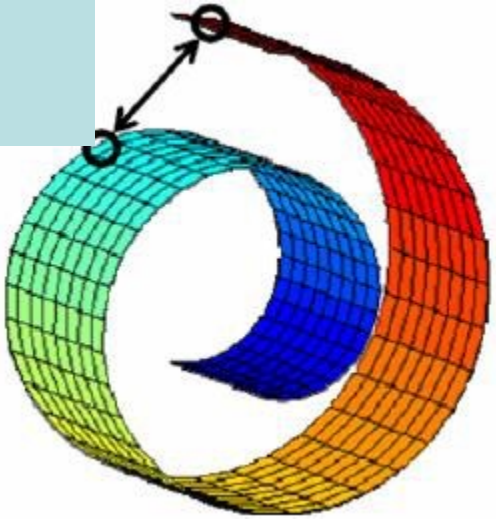
- Linear discrimination
  (e.g., "glasses" versus "no glasses", or "male" versus "female")

- The PCA can not discover the structure of a data set in a spiral form

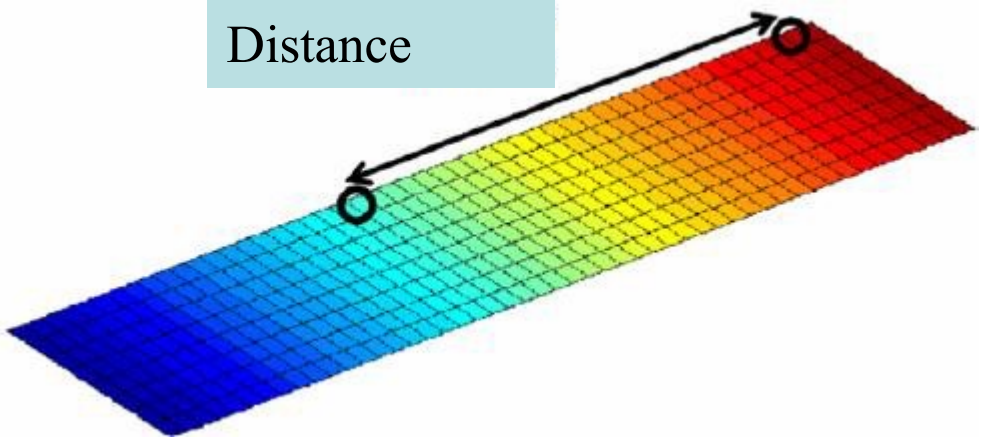


Euclidian Distance vs. Geodesic Distance

Euclidian Distance



Geodesic Distance



Possible solutions

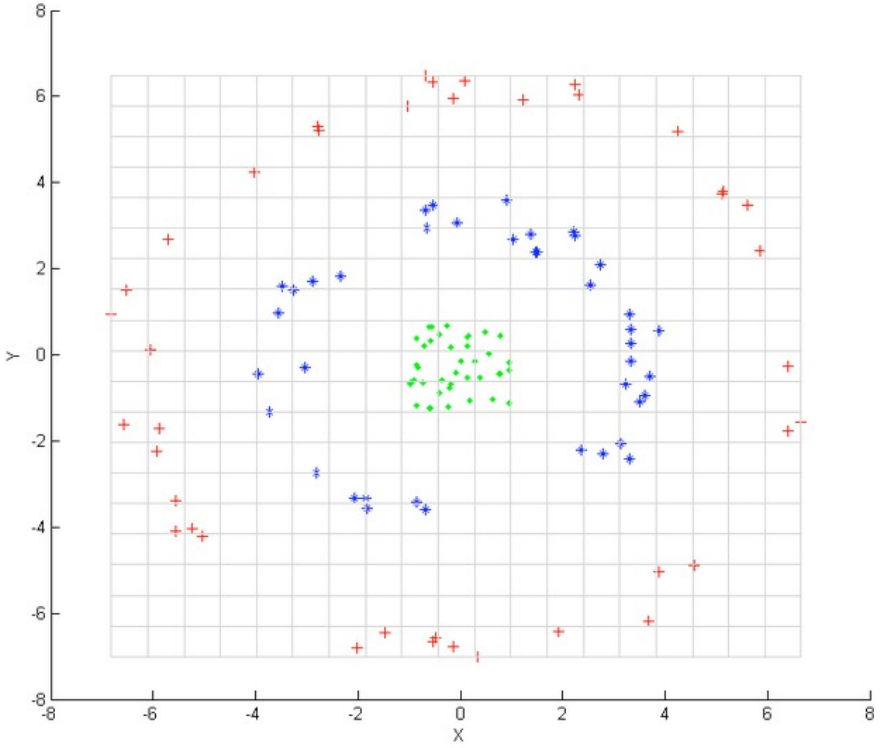
- Kernel PCA
- ISOMAP

Kernel PCA

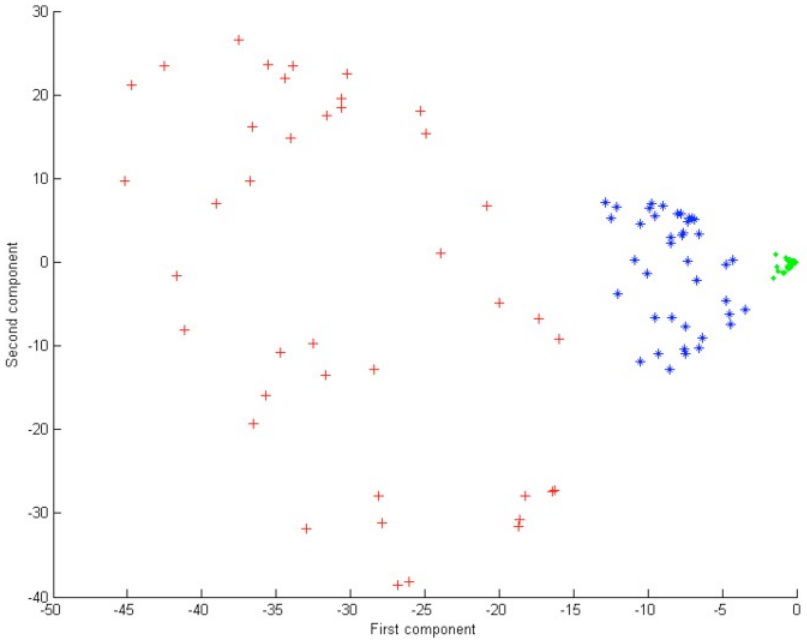
The Kernel PCA Algorithm

- Choose a kernel and apply it to all pairs of points to get matrix $\mathbf{K}$ of distances between the pairs of points in the transformed space
- Compute the eigenvalues and eigenvectors of $\mathbf{K}$
- Normalise the eigenvectors by the square root of the eigenvalues
- Retain the eigenvectors corresponding to the largest eigenvalues

Original data



Data after kernel PCA



Kernel Eigenfaces

(Yang et al., Face Recognition Using Kernel Eigenfaces, 2000)

Training data: ~ 400 images, 40 subjects

Original features: 644 pixel gray-scale values.

Transform data using kernel PCA, reduce dimensionality to number of components giving lowest error.

Test: new photo of one of the subjects

Recognition done using nearest neighbor classification

Table 3: Experimental results on AT&T database

Method	Reduced space	Error Rate (%)
Eigenface	30	2.75
Kernel PCA, d=2	50	2.50
Kernel PCA, d=3	50	2.00
Kernel PCA, d=4	60	2.25
Kernel PCA, d=10	80	2.25

Independent Component Analysis (ICA)

- Independent component analysis attempts to decompose a multivariate signal into independent non-Gaussian signals.
- As an example, sound is usually a signal that is composed of the numerical addition, at each time t, of signals from several sources.
- ICA finds the independent components by maximizing the statistical independence of the estimated components.

Isometric Feature Mapping (ISOMAP)

- The goal of ISOMAP is to find a non-linear manifold containing the data
- We use the fact that for close points, the Euclidean distance is a good approximation to the geodesic distance on the manifold
- We build a graph connecting each point to its k nearest neighbors

ISOMAP

- The lengths of the geodesics are then estimated by searching the length of the shortest path between two points in the graph
- Thereafter, we compute distances to determine a position of points in a space of reduced dimensions

ISOMAP

- ISOMAP [Tenebaum *et al.,* 2000]
  - For neighboring samples, Euclidian distance provides a good approximation to geodesic distance
  - For distant points, geodesic distance can be approximated with a sequence of steps between clusters of neighboring points

ISOMAP

ISOMAP operates in three steps:

1. Build the neighborhood graph *G*
2. For each pair of points of *G*, compute the shortest path (the geodesic distance)
3. Using the classical MDS (Multidimensional Scaling) on geodesic distances

Euclidian Distance is replaced by Geodesic Distance

ISOMAP Algorithm

- Step 1
  - Build the neighborhood graph based on the distances $d_X(i,j)$ in the input space X.
  - This can be performed in two different ways:
    - Connect each point to all points within a fixed radius ε
    - Connect each point to all of its k nearest neighbors
  - A weighted neighborhood graph G is obtained, where $d_X(i,j)$ is the weight of each edge between neighboring points

ISOMAP Algorithm

- Step 2
  - Estimate the geodesic distances $d_M(i,j)$ between all pair of points on the manifold M by computing their shortest path distances $d_G(i,j)$ in the graph G.
  - It can be done using Dijkstra's or Floyd's algorithm.

ISOMAP Algorithm

- Step 3
  - Apply classical MDS to the matrix of graph distances D.

Multidimensional scaling (MDS) is a means of visualizing the level of similarity of individual vectors of a dataset. MDS is used to translate "information about the pairwise distances into a configuration of points mapped into a Cartesian space.

Complexity of ISOMAP

- For large datasets ISOMAP can be quite slow :

- Step 1: Complexity of k-nearest neighbors $O(n^2 D)$

- Step 2 : Complexity of the Djikstra algorithm $O(n^2 \log n + n^2 k)$

- Step 3 : MDS complexity $O(n^2 d)$