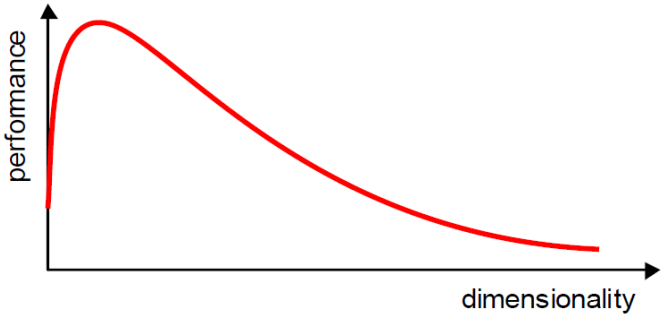
Dimensionality Reduction

Based on slides of Dr. George Bebis CS479

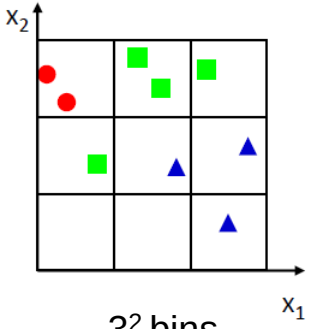
Curse of Dimensionality

- Increasing the number of features will not always improve classification accuracy.
- In practice, the inclusion of more features might actually lead to worse performance.
- The number of training examples required increases exponentially with dimensionality **d** (i.e., $k^{\mathbf{d}}$).

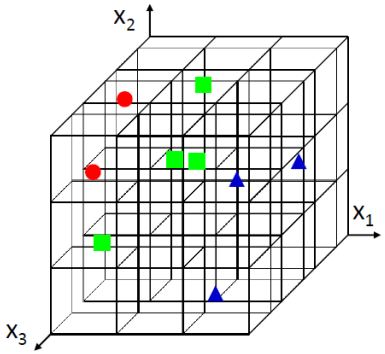


k=3

3^1 bins



3^2 bins



3^3 bins

k: number of bins per feature

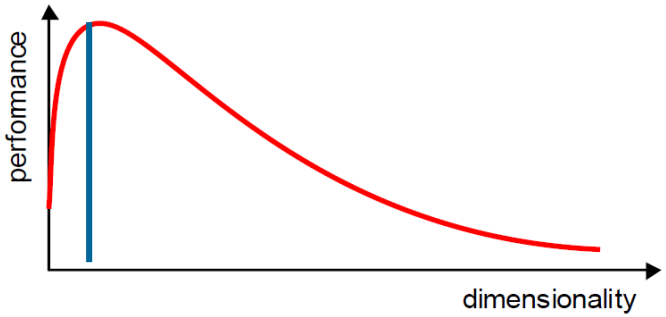
Why DR

Most machine learning and data mining techniques may not be effective for high-dimensional data

- Curse of Dimensionality
- Query accuracy and efficiency degrade rapidly as the dimension increases.
- The intrinsic dimension may be small.
- For example, the number of genes responsible for a certain type of disease may be small.

Dimensionality Reduction

- What is the objective?
  - Choose an optimum set of features of lower dimensionality to improve classification accuracy.



- Different methods can be used to reduce dimensionality:
  - Feature extraction
  - Feature selection

Application of Dimensionality Reduction

- Customer relationship management
- Text mining
- Image retrieval
- Microarray data analysis
- Protein classification
- Face recognition
- Handwritten digit recognition
- Intrusion detection

Feature Selection

- Definition
  - A process that chooses an optimal subset of features according to an objective function
- Objectives
  - To reduce dimensionality and remove noise
  - To improve mining performance
    - Speed of learning
    - Predictive accuracy
    - Simplicity and comprehensibility of mined results

Different Aspects of Search

- **Search starting points**
  - **Empty set**
  - **Full set**
  - **Random point**
- **Search directions**
  - **Sequential forward selection**
  - **Sequential backward elimination**
  - **Bidirectional generation**
  - **Random generation**

Dimensionality Reduction (cont'd)

Feature extraction: finds a set of new features (i.e., through some mapping f()) from the existing features.

Feature selection: chooses a subset of the original features.

The mapping f() could be linear or non-linear

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

K<<N

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \rightarrow \mathbf{y} = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \cdot \\ \cdot \\ x_{i_K} \end{bmatrix}$$

K<<N

Feature Reduction vs. Feature Selection

- Feature reduction
  - All original features are used
  - The transformed features are linear combinations of the original features
- Feature selection
  - Only a subset of the original features are selected
- Continuous versus discrete

Feature Extraction

- Linear combinations are particularly attractive because they are simpler to compute and analytically tractable.

- Given $x \in R^N$, find an K x N matrix T such that:

$$y = Tx \in R^K \text{ where } K<<N$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{T,\ f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

This is a projection from the N-dimensional space to a K-dimensional space.

Feature Extraction (cont'd)

- From a mathematical point of view, finding an optimum mapping $\mathbf{y}=f(\mathbf{x})$ is equivalent to optimizing an **objective** criterion.

- Different methods use different objective criteria, e.g.,

  - Minimize Information Loss: represent the data as accurately as possible in the lower-dimensional space.

  - Maximize Discriminatory Information: enhance the class-discriminatory information in the lower-dimensional space.

Feature Extraction (cont'd)

- Popular linear feature extraction methods:
  - Principal Components Analysis (PCA): Seeks a projection that **preserves** as much **information** in the data as possible.
  - Linear Discriminant Analysis (LDA): Seeks a projection that **best discriminates** the data.

- Many other methods:
  - Making features as independent as possible (Independent Component Analysis or ICA).
  - Retaining interesting directions (Projection Pursuit).
  - Embedding to lower dimensional manifolds (Isomap, Locally Linear Embedding or LLE).

Vector Representation

- A vector **x** ϵ R^n can be represented by n components:

$$\mathbf{x}: \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix}$$

- Assuming the standard base $<v_1, v_2, \ldots, v_N>$ (i.e., unit vectors in each dimension), x_i can be obtained by projecting **x** along the direction of v_i:

$$x_i = \frac{\mathbf{x}^T v_i}{v_i^T v_i} = \mathbf{x}^T v_i$$

- **x** can be "reconstructed" from its projections as follows:

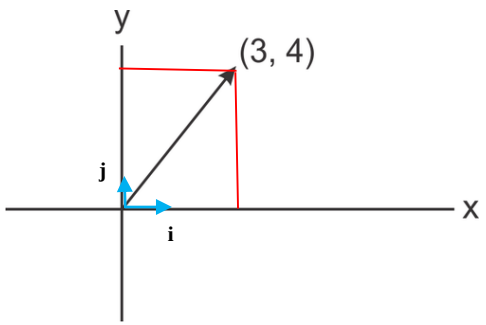
$$\mathbf{x} = \sum_{i=1}^{N} x_i v_i = x_1 v_1 + x_2 v_2 + \ldots + x_N v_N$$

- Since the basis vectors are the same for all x ϵ R^n (standard basis), we typically represent them as a n-component vector.

Vector Representation (cont'd)

- **Example** assuming n=2:

$$\mathbf{x}: \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$



- Assuming the standard base <v_1=i, v_2=j>, x_i can be obtained by projecting x along the direction of v_i:

$$x_1 = \mathbf{x}^T i = \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 3$$

$$x_2 = \mathbf{x}^T j = \begin{bmatrix} 3 & 4 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 4$$

- **x** can be "reconstructed" from its projections as follows:

$$\mathbf{x} = 3i + 4j$$

Principal Component Analysis (PCA)

- If $\mathbf{x} \in R^N$, then it can be written a linear combination of an orthonormal set of N basis vectors $<v_1, v_2, \ldots, v_N>$ in R^N (e.g., using the standard base):

$$v_i^T v_j = \begin{cases} 1 & if \ \ i = j \\ 0 & otherwise \end{cases}$$

$$\mathbf{x} = \sum_{i=1}^{N} x_i v_i = x_1 v_1 + x_2 v_2 + \ldots + x_N v_N$$

$$where \quad x_i = \frac{\mathbf{x}^T v_i}{v_i^T v_i} = \mathbf{x}^T v_i$$

$$\mathbf{x}: \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ . \\ . \\ x_N \end{bmatrix}$$

- PCA seeks to approximate **x** in a subspace of R^N using a new set of K<<N basis vectors $<u_1, u_2, \ldots, u_K>$ in R^N:

$$\hat{\mathbf{x}} = \sum_{i=1}^{K} y_i u_i = y_1 u_1 + y_2 u_2 + \ldots + y_K u_K \quad \text{(reconstruction)} \qquad where \quad y_i = \frac{\mathbf{x}^T u_i}{u_i^T u_i} = \mathbf{x}^T u_i$$

$$\hat{\mathbf{x}}: \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_K \end{bmatrix}$$

such that $\|\mathbf{x} - \hat{\mathbf{x}}\|$ is minimized!

(i.e., minimize information loss)

Principal Component Analysis (PCA)

- The "optimal" set of basis vectors $<u_1, u_2, \ldots, u_K>$ can be found as follows (we will see why):

(1) Find the eigenvectors u_i of the covariance matrix of the (training) data Σ_x

$$\Sigma_x u_i = \lambda_i u_i$$

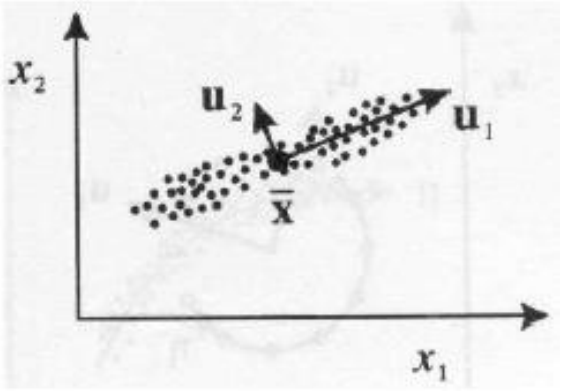
(2) Choose the K "largest" eigenvectors u_i (i.e., corresponding to the K "largest" eigenvalues λ_i)

$<u_1, u_2, \ldots, u_K>$ correspond to the "optimal" basis!

We refer to the "largest" eigenvectors u_i as principal components.

Interpretation of PCA

- PCA chooses the eigenvectors of the covariance matrix corresponding to the largest eigenvalues.
- The eigenvalues correspond to the variance of the data along the eigenvector directions.
- Therefore, PCA projects the data along the directions where the data varies most.
- PCA preserves as much information in the data by preserving as much variance in the data.



u_1: direction of max variance
u_2: orthogonal to u_1

Geometric Rationale of PCA

- objects are represented as a cloud of *n* points in a multidimensional space with an axis for each of the *p* variables
- the centroid of the points is defined by the mean of each variable
- the variance of each variable is the average squared deviation of its *n* values around the mean of that variable.

$$V_i = \frac{1}{n-1} \sum_{m=1}^{n} \left(X_{im} - \overline{X}_i \right)^2$$

Geometric Rationale of PCA

- degree to which the variables are linearly correlated is represented by their covariances.

$$C_{ij} = \frac{1}{n-1}\sum_{m=1}^{n}\left(X_{im} - \overline{X}_i\right)\left(X_{jm} - \overline{X}_j\right)$$

Covariance of variables *i* and *j*

Sum over all *n* objects

Value of variable *i* in object *m*

Mean of variable *i*

Value of variable *j* in object *m*

Mean of variable *j*

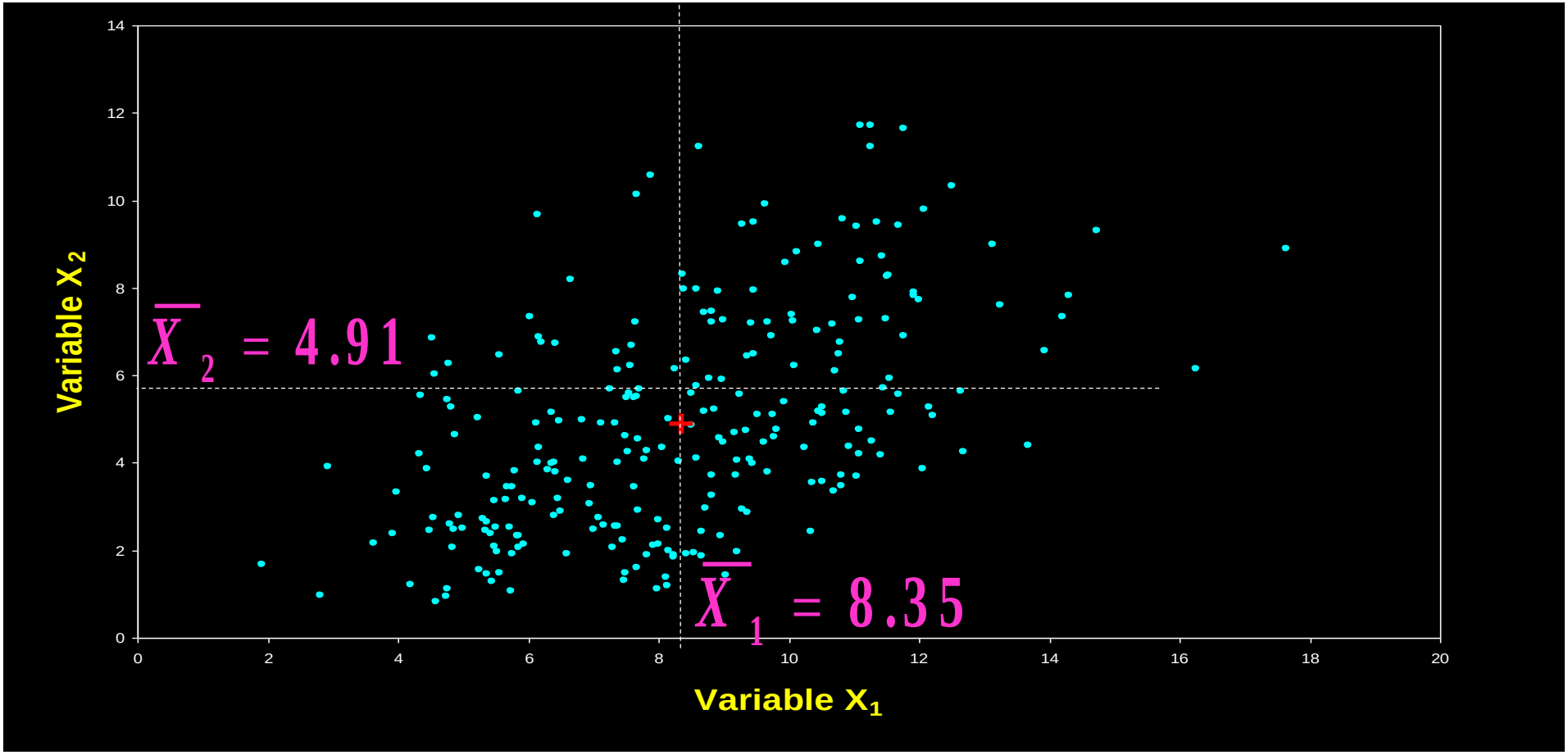
Geometric Rationale of PCA

- objective of PCA is to rigidly rotate the axes of this *p*-dimensional space to new positions (principal axes) that have the following properties:
  - ordered such that principal axis 1 has the highest variance, axis 2 has the next highest variance, ... , and axis *p* has the lowest variance
  - covariance among each pair of the principal axes is zero (the principal axes are uncorrelated).

$$C = \begin{pmatrix} cov(x,x) & cov(x,y) & cov(x,z) \\ cov(y,x) & cov(y,y) & cov(y,z) \\ cov(z,x) & cov(z,y) & cov(z,z) \end{pmatrix}$$

2D Example of PCA

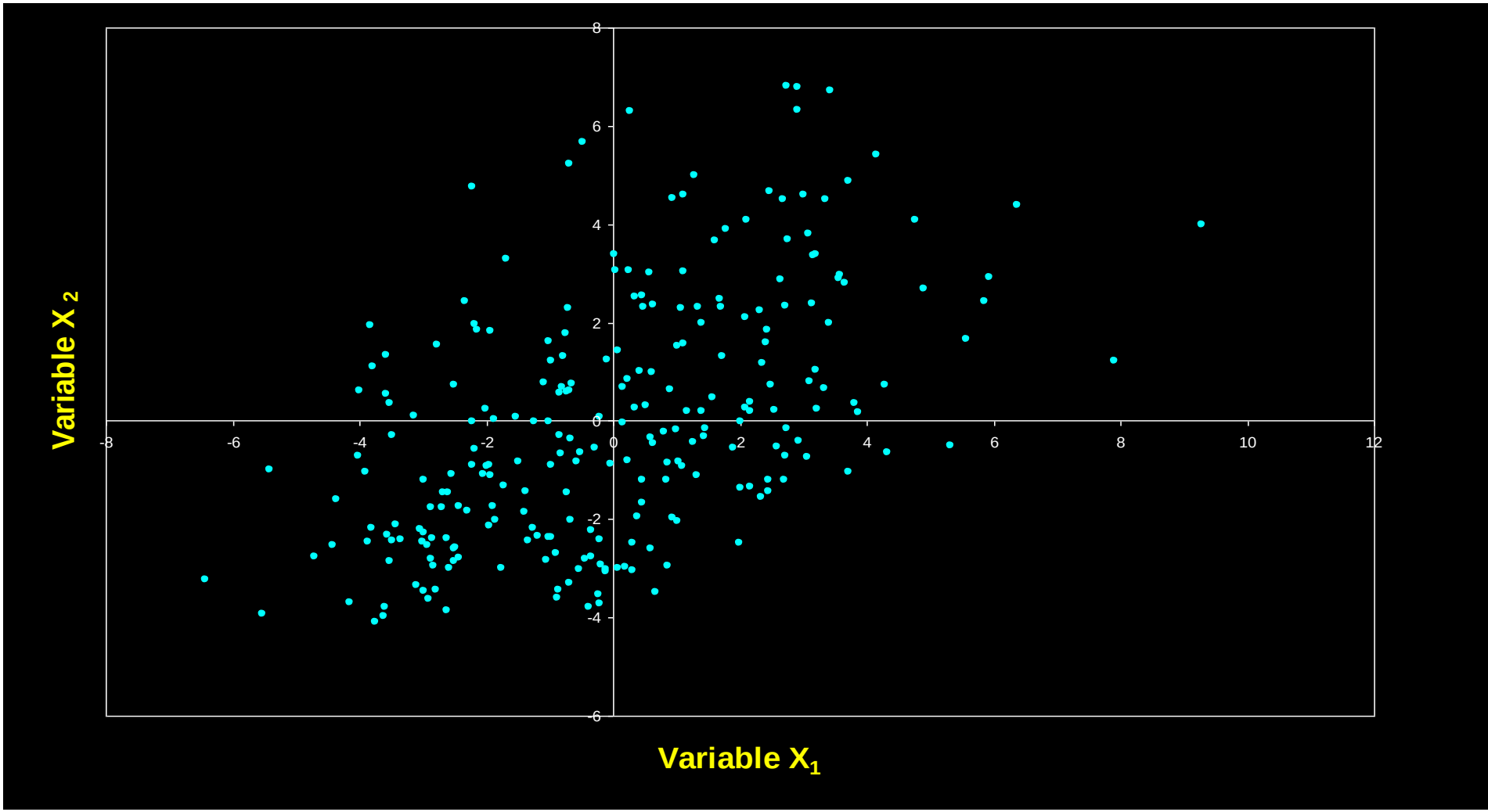
- variables X_1 and X_2 have positive covariance & each has a similar variance.



$V_1 = 6.67$ $V_2 = 6.24$ $C_{1,2} = 3.42$

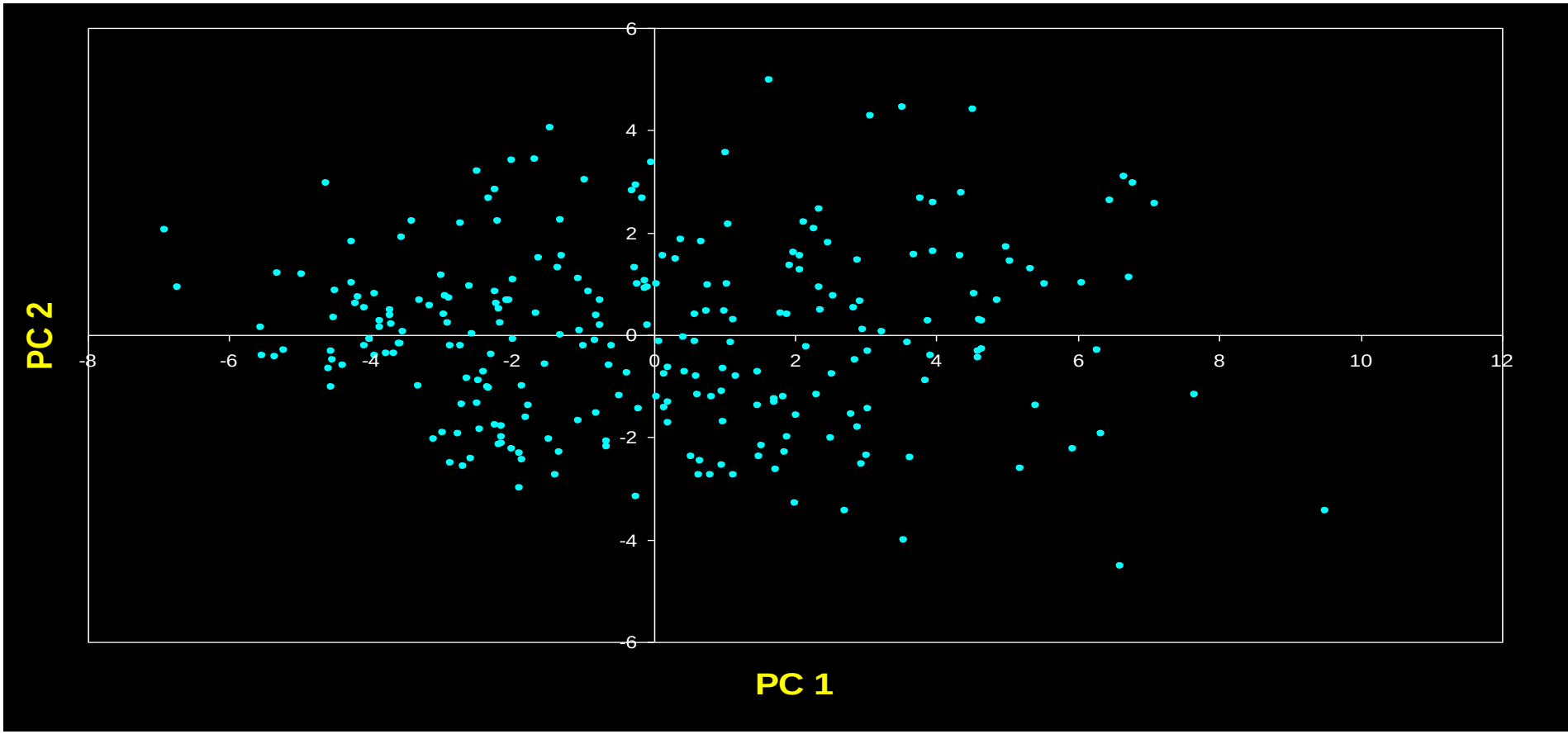
Configuration is Centered

- each variable is adjusted to a mean of zero (by subtracting the mean from each value).



Principal Components are Computed

- PC 1 has the highest possible variance (9.88)
- PC 2 has a variance of 3.03
- PC 1 and PC 2 have zero covariance.



The Dissimilarity Measure Used in PCA is Euclidean Distance

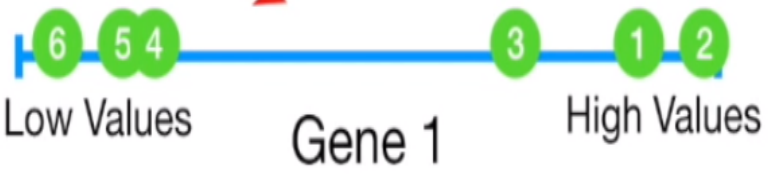
- PCA uses Euclidean Distance calculated from the *p* variables as the measure of dissimilarity among the n objects

- PCA derives the best possible *k* dimensional ($k < p$) representation of the Euclidean distances among objects.

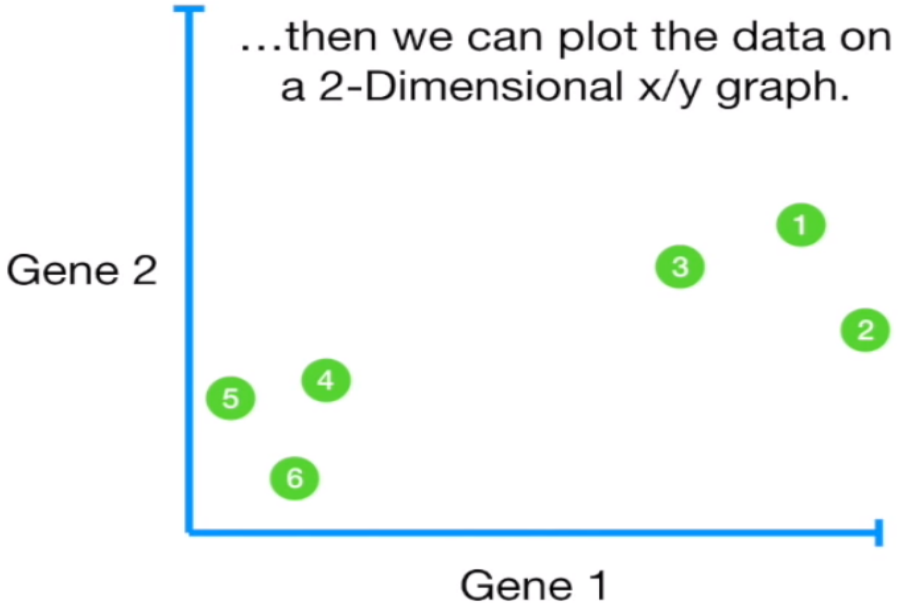
Gene Example

	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1

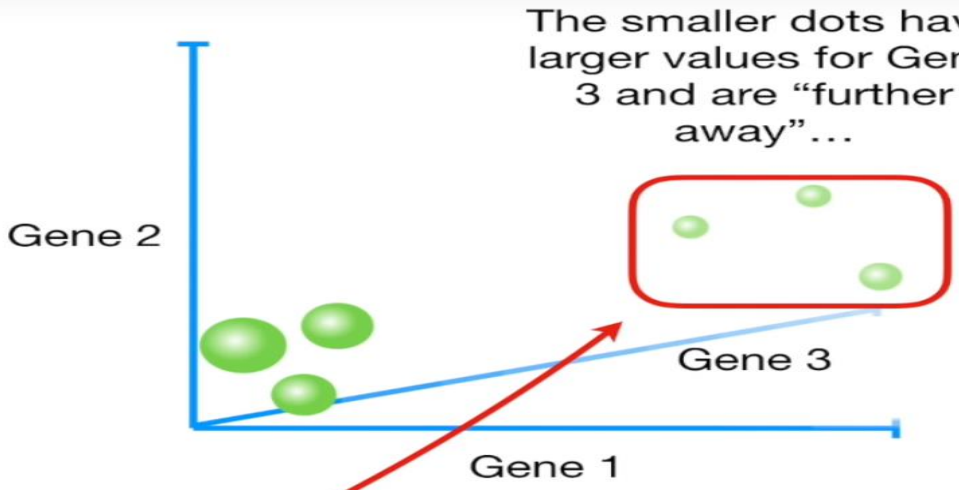
Even though it's a simple graph, it shows us that mice 1, 2 and 3 are more similar to each other than they are to mice 4, 5 6.



	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1



	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1
Gene 3	12	9	10	2.5	1.3	2

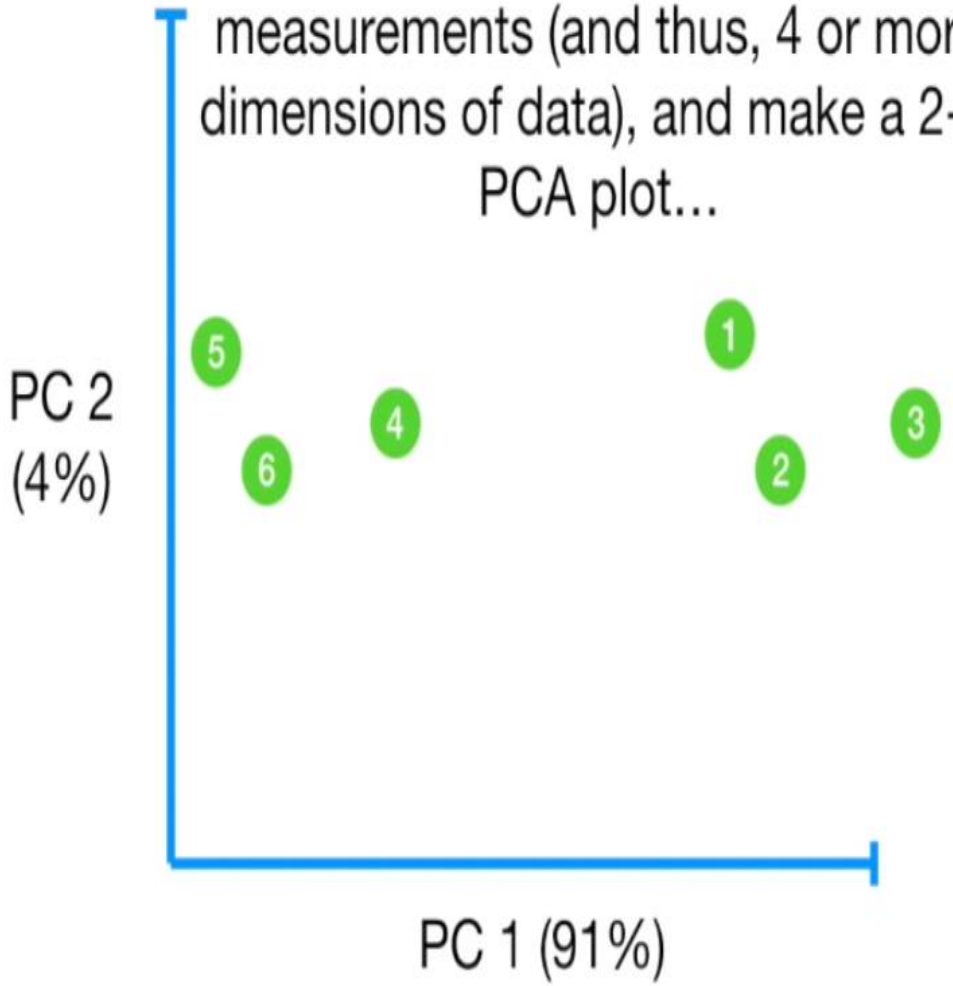


	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1
Gene 3	12	9	10	2.5	1.3	2
Gene 4	5	7	6	2	4	7

If we measured 4 genes, however, we can no longer plot the data - 4 genes require 4 dimensions.

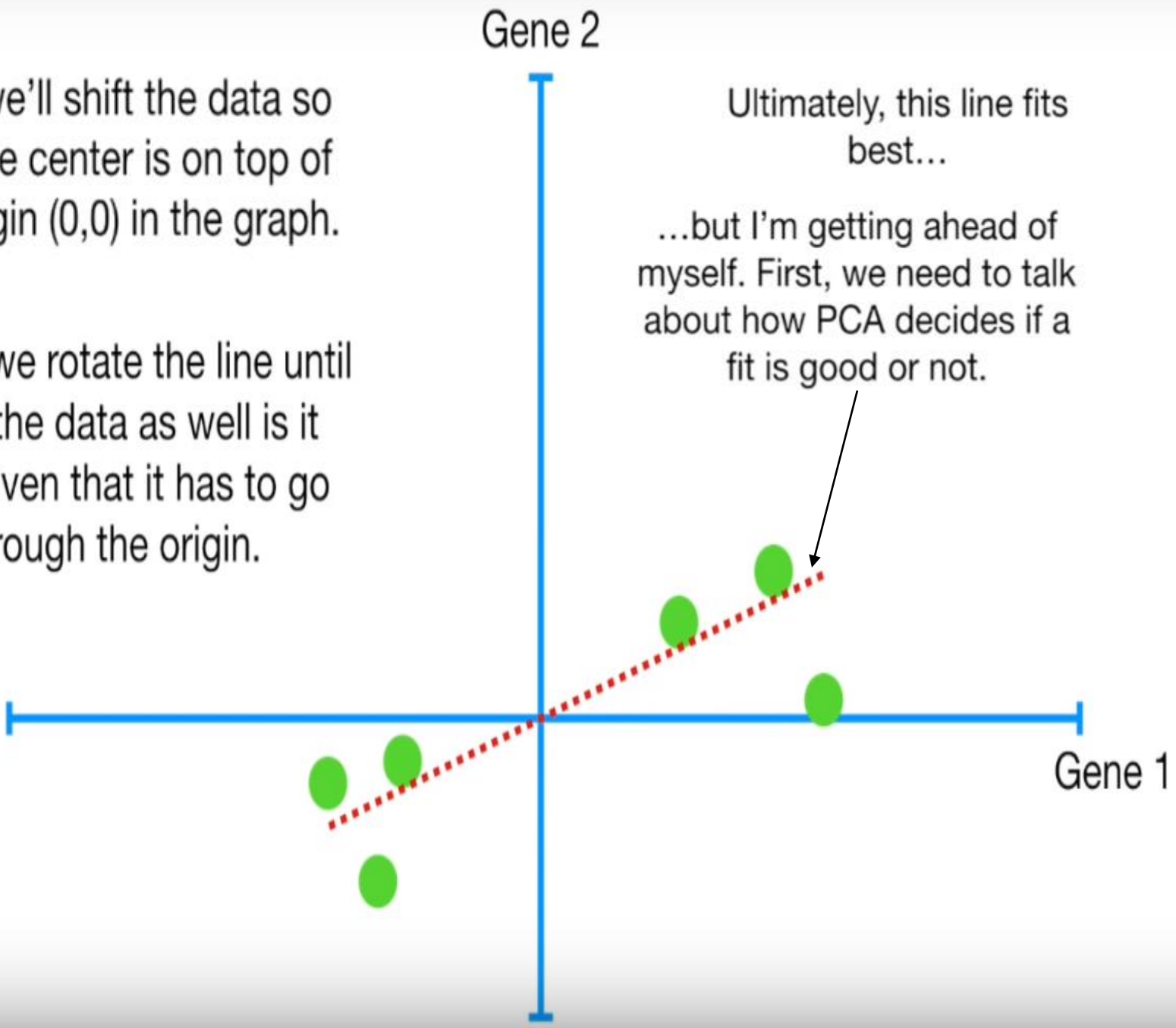
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1
Gene 3	12	9	10	2.5	1.3	2
Gene 4	5	7	6	2	4	7

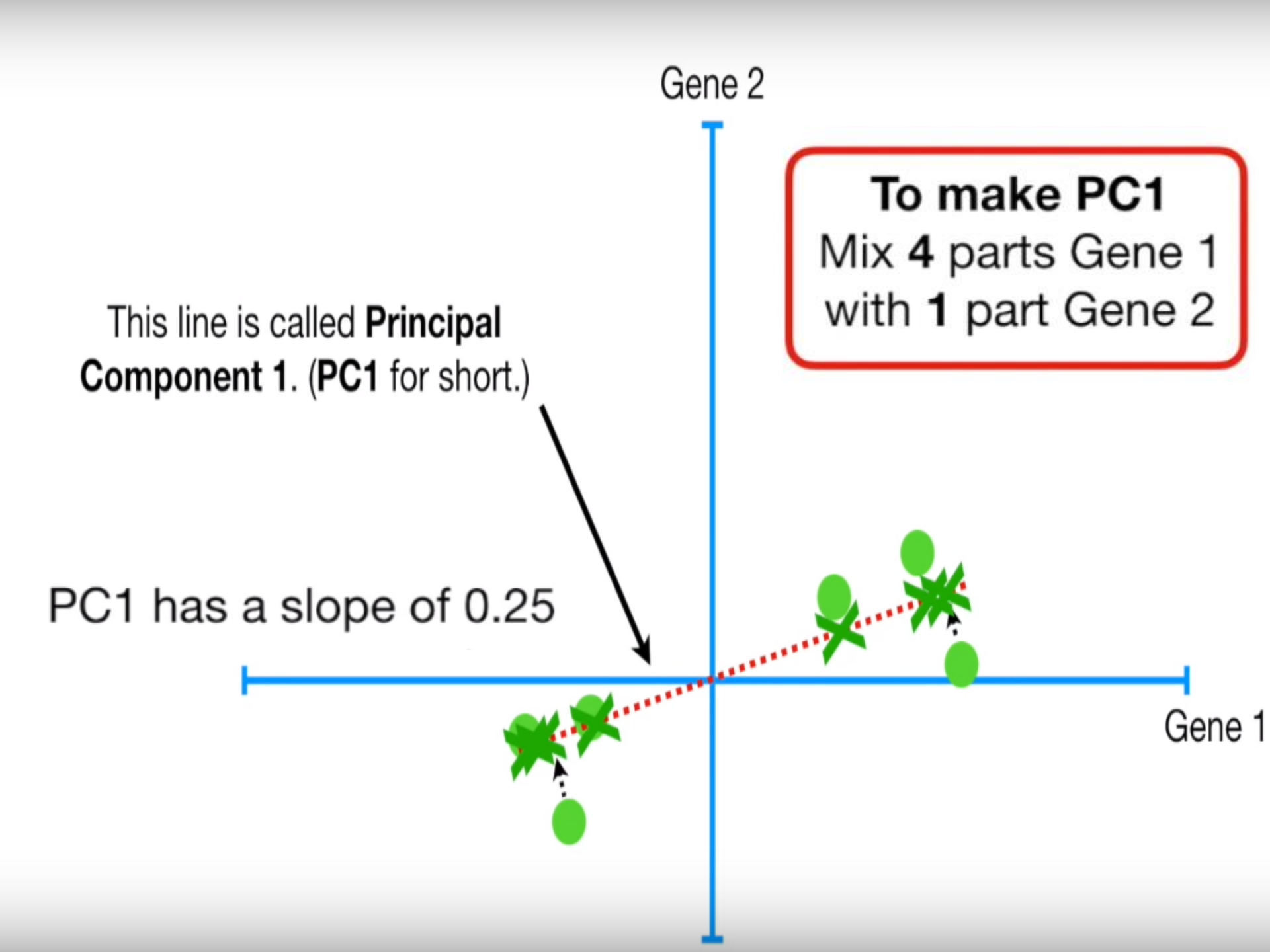
So we're going to talk about how PCA can take 4 or more gene measurements (and thus, 4 or more dimensions of data), and make a 2-D PCA plot...



Now we'll shift the data so that the center is on top of the origin (0,0) in the graph.

…then we rotate the line until it fits the data as well is it can, given that it has to go through the origin.



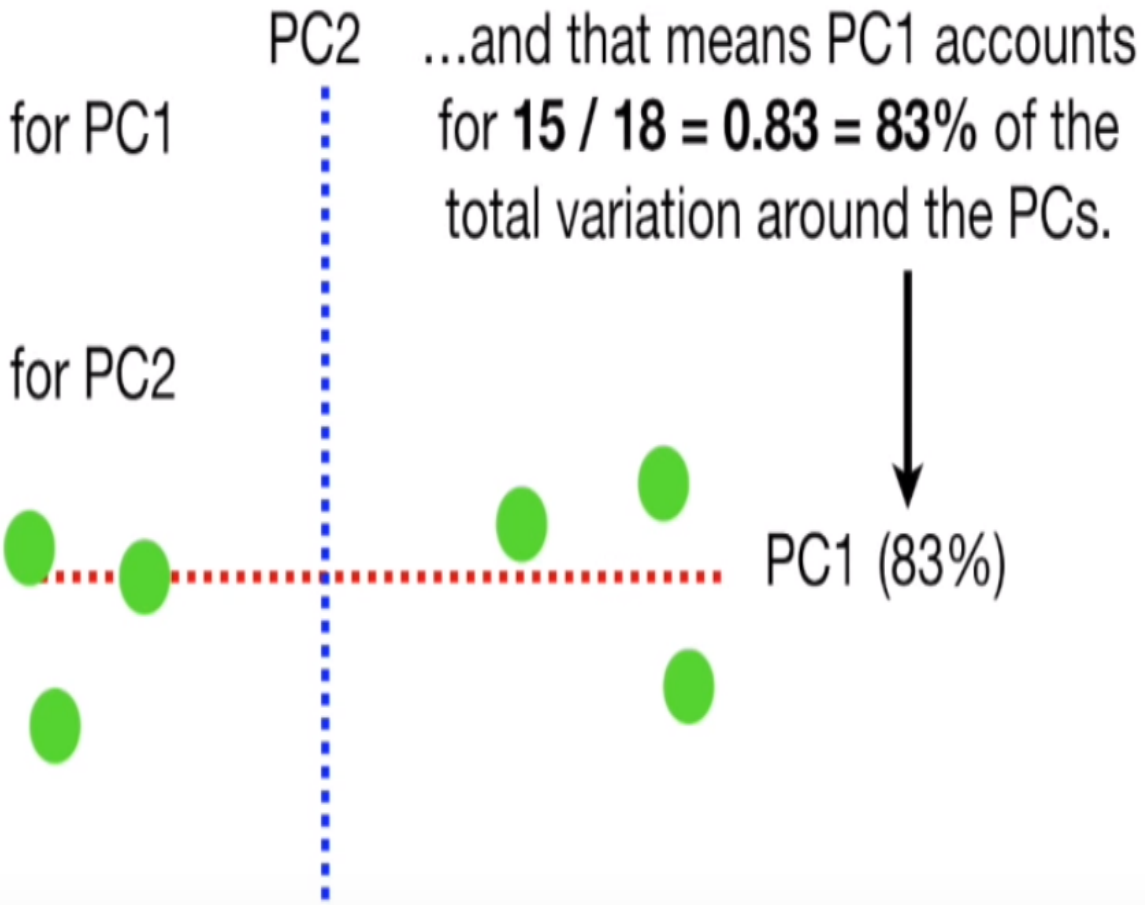

Gene 2
To make PC1
Mix 4 parts Gene 1
with 1 part Gene 2
This line is called Principal Component 1. (PC1 for short.)
PC1 has a slope of 0.25
Gene 1

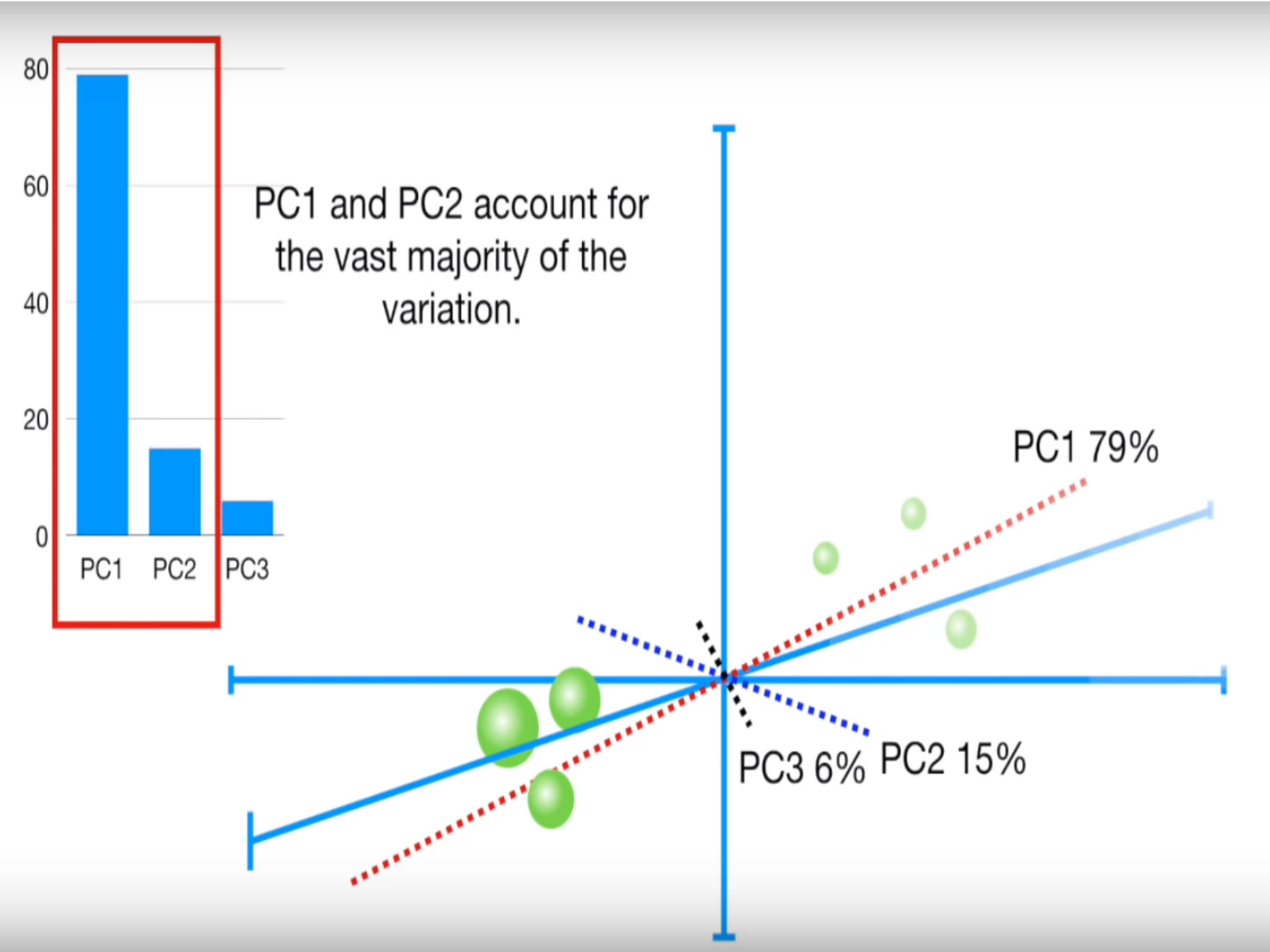
For the sake of the example, imagine that the Variation for **PC1 = 15**, and the variation for **PC2 = 3**.

That means that the total variation around both PCs is **15 + 3 = 18**...

$$\frac{SS(\text{distances for PC1})}{n - 1} = \text{Variation for PC1}$$

$$\frac{SS(\text{distances for PC2})}{n - 1} = \text{Variation for PC2}$$



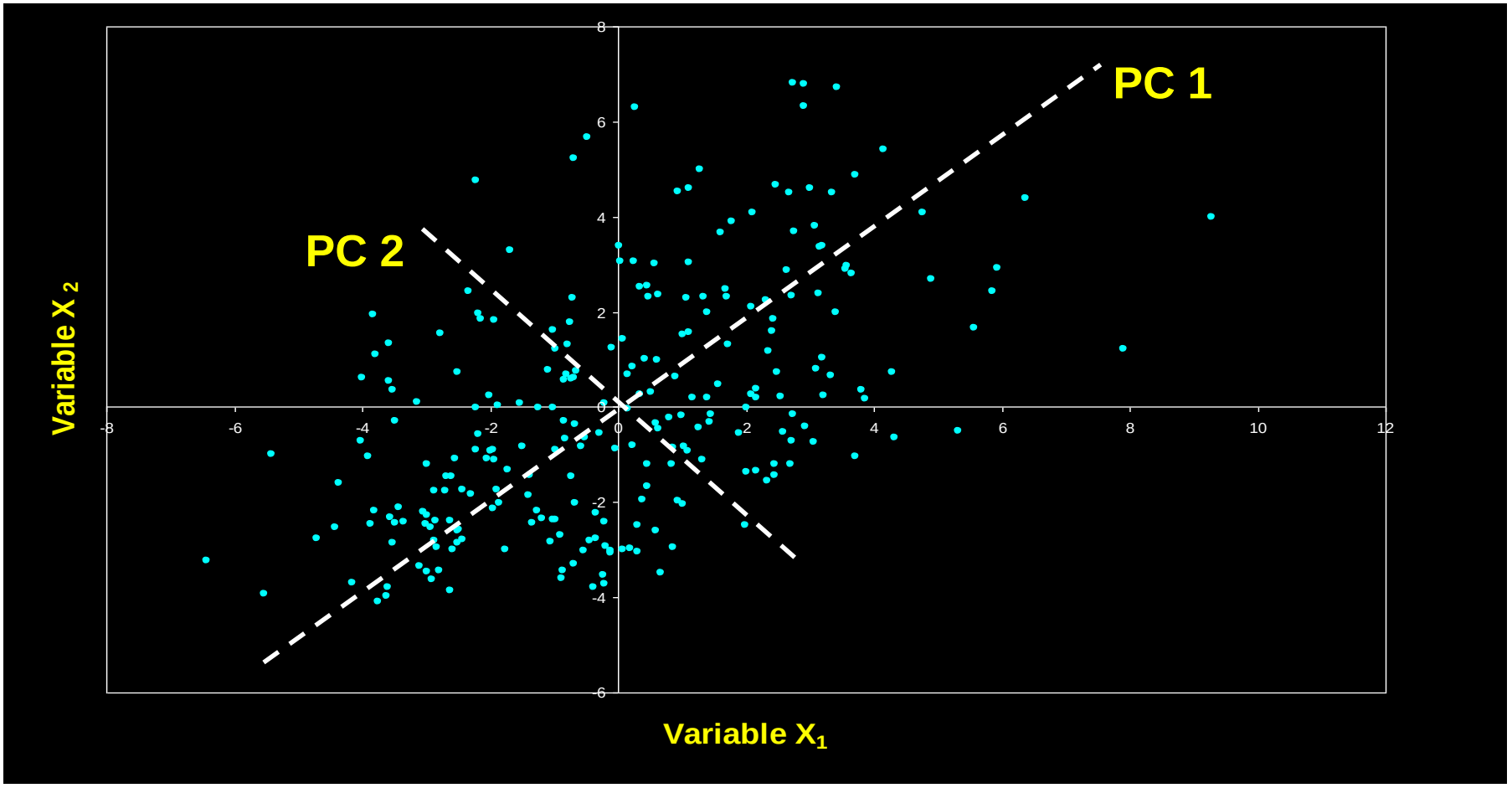

80
60
40
20
0
PC1
PC2
PC3
PC1 and PC2 account for the vast majority of the variation.
PC1 79%
PC3 6%
PC2 15%

Generalization to p-dimensions

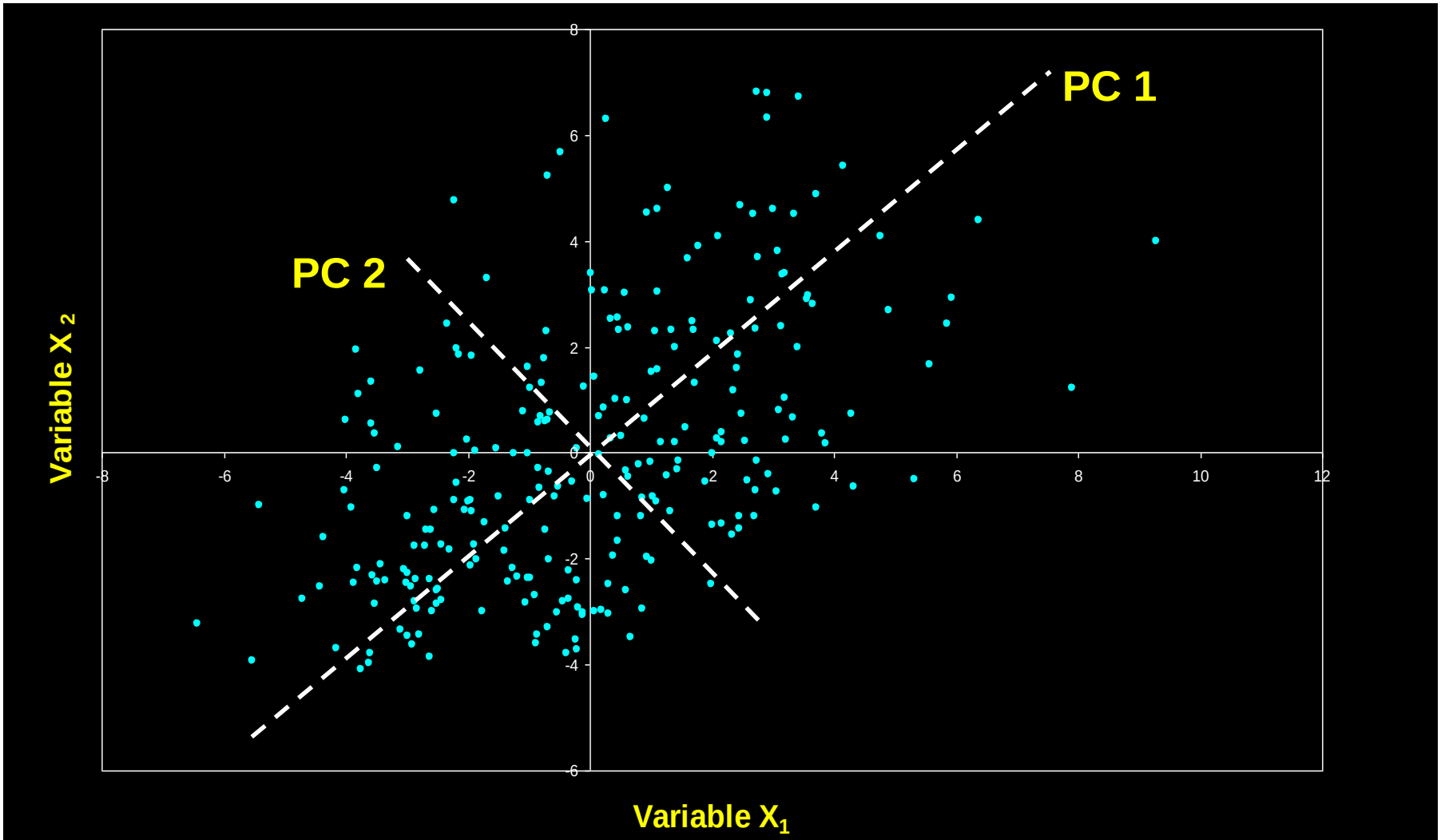
- In practice nobody uses PCA with only 2 variables
- The algebra for finding principal axes readily generalizes to *p* variables
- PC 1 is the direction of maximum variance in the *p*-dimensional cloud of points
- PC 2 is in the direction of the next highest variance, subject to the constraint that it has zero covariance with PC 1.

- PC 3 is in the direction of the next highest variance, subject to the constraint that it has zero covariance with both PC 1 and PC 2
- and so on... up to PC *p*

- each principal axis is a linear combination of the original two variables
- $PC_j = a_{i1}Y_1 + a_{i2}Y_2 + \ldots a_{in}Y_n$
- a_{ij}'s are the coefficients for factor i, multiplied by the measured value for variable j



- PC axes are a rigid rotation of the original variables
- PC 1 is simultaneously the direction of maximum variance and a least-squares "line of best fit" (squared distances of points away from PC 1 are minimized).



Generalization to p-dimensions

- if we take the first *k* principal components, they define the *k*-dimensional "hyperplane of best fit" to the point cloud
- of the total variance of all *p* variables:
  - PCs 1 to *k* represent the maximum possible proportion of that variance that can be displayed in *k* dimensions
  - *i.e.* the squared Euclidean distances among points calculated from their coordinates on PCs 1 to *k* are the best possible representation of their squared Euclidean distances in the full *p* dimensions.

Covariance vs Correlation

- using covariances among variables only makes sense if they are measured in the same units
- even then, variables with high variances will dominate the principal components
- these problems are generally avoided by standardizing each variable to unit variance and zero mean.

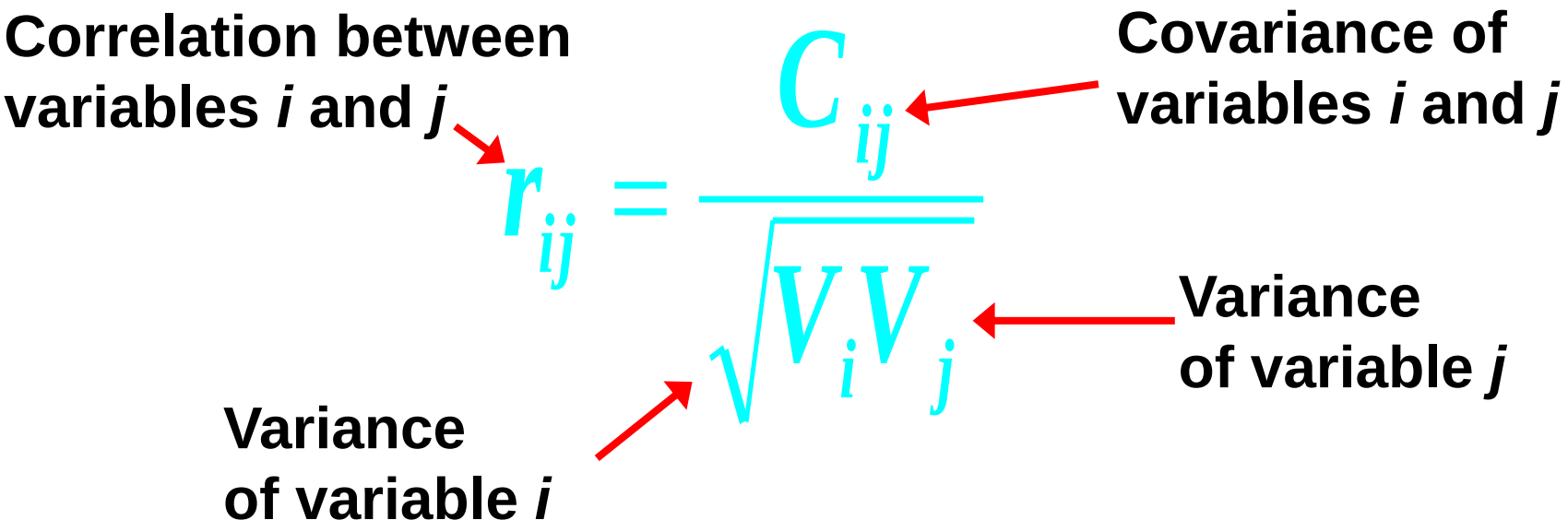
$$X'_{im} = \frac{\left(X_{im} - \overline{X}_i\right)}{\mathrm{SD}_i}$$

Mean variable *i*

Standard deviation of variable *i*

Covariance vs Correlation

- covariances between the standardized variables are correlations
- after standardization, each variable has a variance of 1.000
- correlations can be also calculated from the variances and covariances:



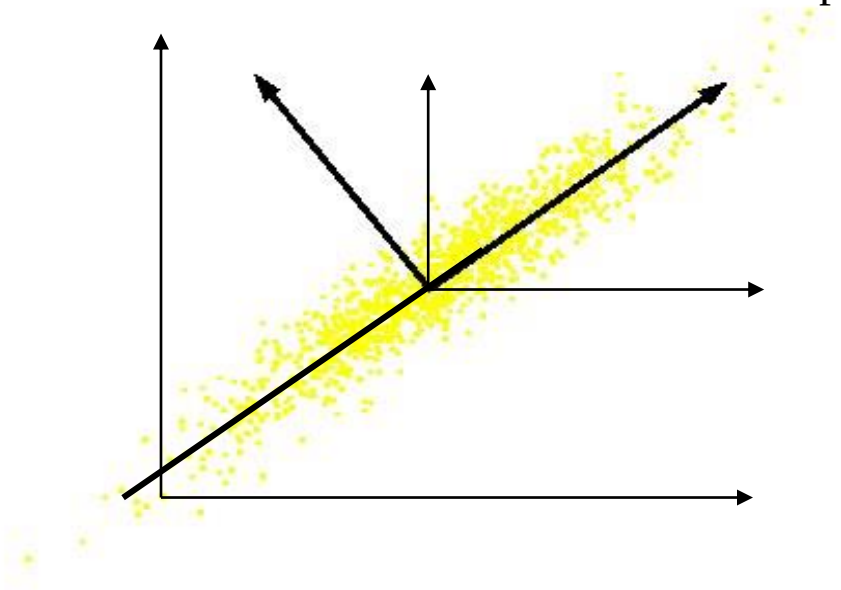
Principal Components Analysis (PCA)

- Principle
  - Linear projection method to reduce the number of parameters
  - Transfer a set of correlated variables into a new set of uncorrelated variables
  - Map the data into a space of lower dimensionality
  - Form of unsupervised learning

- Properties
  - It can be viewed as a rotation of the existing axes to new positions in the space defined by original variables
  - New axes are orthogonal and represent the directions with maximum variability

Computing the Components

- Data points are vectors in a multidimensional space
- Projection of vector **x** onto an axis (dimension) **u** is **u.x**
- Direction of greatest variability is that in which the average square of the projection is greatest
  - I.e. **u** such that $E((\mathbf{u}.\mathbf{x})^2)$ over all **x** is maximized
  - (we subtract the mean along each dimension, and center the original axis system at the centroid of all data points, for simplicity)
  - This direction of **u** is the direction of the first Principal Component



Computing the Components

- $E((\mathbf{u}.\mathbf{x})^2) = E((\mathbf{u}.\mathbf{x})(\mathbf{u}.\mathbf{x})^T) = E(\mathbf{u}.\mathbf{x}.\mathbf{x}^T.\mathbf{u}^T)$
- The matrix $\mathbf{C} = \mathbf{x}.\mathbf{x}^T$ contains the correlations (similarities) of the original axes based on how the data values project onto them
- So we are looking for w that maximizes $\mathbf{u}\mathbf{C}\mathbf{u}^T$, subject to $\mathbf{u}$ being unit-length
- It is maximized when w is the principal eigenvector of the matrix $\mathbf{C}$, in which case
  - $\mathbf{u}\mathbf{C}\mathbf{u}^T = \mathbf{u}\lambda\mathbf{u}^T = \lambda$ if $\mathbf{u}$ is unit-length, where λ is the principal eigenvalue of the correlation matrix C
  - The eigenvalue denotes the amount of variability captured along that dimension

Why the Eigenvectors?

Maximise $\mathbf{u}^T\mathbf{x}\mathbf{x}^T\mathbf{u}$ s.t $\mathbf{u}^T\mathbf{u} = 1$

How to Maximize? Which kind of function?

Square Function (what about derivative equals zero?)

Adding Constraint: Maximise $\mathbf{u}^T\mathbf{x}\mathbf{x}^T\mathbf{u}$ s.t $\mathbf{u}^T\mathbf{u} = 1$

How to Solve? Lagrangian Dual Problem:

Construct Lagrangian

$$\mathbf{u}^T\mathbf{x}\mathbf{x}^T\mathbf{u} - \lambda(\mathbf{u}^T\mathbf{u}-1) = \mathbf{u}^T\mathbf{x}\mathbf{x}^T\mathbf{u} - \lambda\mathbf{u}^T\mathbf{u} - \lambda$$

Vector of partial derivatives (by $\mathbf{u}$) set to zero

$$2\mathbf{x}\mathbf{x}^T\mathbf{u} - 2\lambda\mathbf{u} = (\mathbf{x}\mathbf{x}^T - \lambda\mathbf{I})\,\mathbf{u} = 0 \Rightarrow \mathbf{x}\mathbf{x}^T\,\mathbf{u} = \lambda\mathbf{u}$$

As $\mathbf{u} \neq \mathbf{0}$ then $\mathbf{u}$ must be an eigenvector of $\mathbf{x}\mathbf{x}^T$ with eigenvalue λ

Singular Value Decomposition

The first root is called the prinicipal eigenvalue which has an associated orthonormal ($\mathbf{u}^T\mathbf{u} = 1$) *eigenvector* $\mathbf{u}$

Subsequent roots are ordered such that $\lambda_1 > \lambda_2 > \ldots > \lambda_M$ with rank($\mathbf{D}$) non-zero values.

Eigenvectors form an orthonormal basis i.e. $\mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}$

The eigenvalue decomposition of $\mathbf{x}\mathbf{x}^T = \mathbf{U}\boldsymbol{\Sigma}\mathbf{U}^T$

where $\mathbf{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_M]$ and $\boldsymbol{\Sigma} = \text{diag}[\lambda_1, \lambda_2, \ldots, \lambda_M]$

Similarly the eigenvalue decomposition of $\mathbf{x}^T\mathbf{x} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{V}^T$

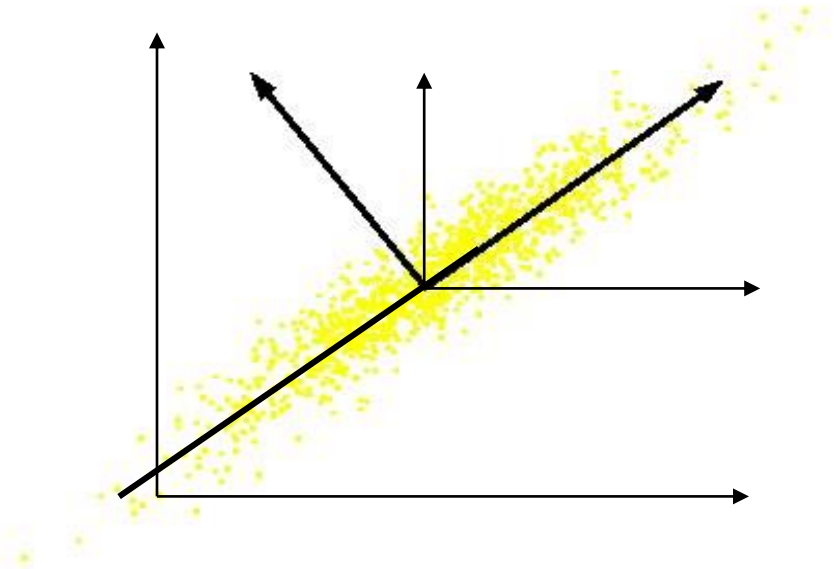
The SVD is closely related to the above $\mathbf{x} = \mathbf{U}\,\boldsymbol{\Sigma}^{1/2}\,\mathbf{V}^T$

The left eigenvectors **U,** right eigenvectors **V,**

singular values = square root of eigenvalues**.**

Computing the Components

- Similarly for the next axis, etc.
- So, the new axes are the eigenvectors of the matrix of correlations of the original variables, which captures the similarities of the original variables based on how data samples project to them



- **Geometrically: centering followed by rotation**
  - **Linear transformation**

PCs, Variance and Least-Squares

- The first PC retains the greatest amount of variation in the sample
- The k^{th} PC retains the k^{th} greatest fraction of the variation in the sample
- The k^{th} largest eigenvalue of the correlation matrix C is the variance in the sample along the k^{th} PC

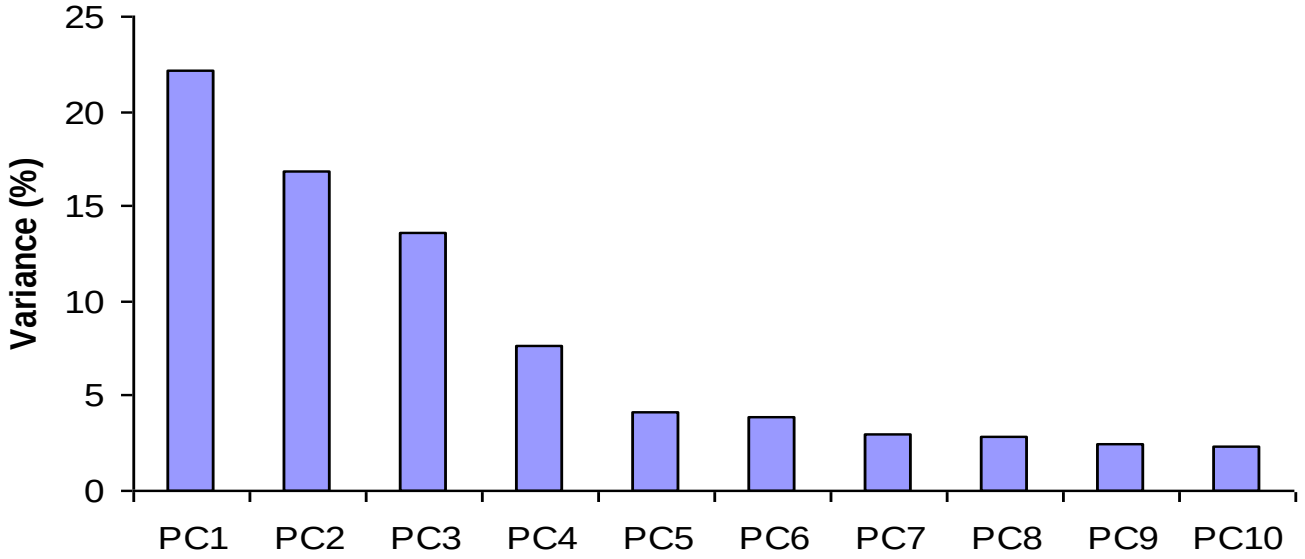
- The least-squares view: PCs are a series of linear least squares fits to a sample, each orthogonal to all previous ones

How Many PCs?

- For n original dimensions, correlation matrix is nxn, and has up to n eigenvectors. So n PCs.
- Where does dimensionality reduction come from?

Dimensionality Reduction

Can *ignore* the components of lesser significance.

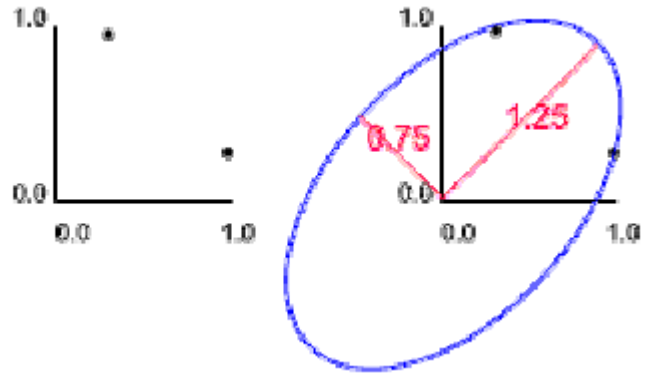


You do lose some information, but if the eigenvalues are small, you don't lose much

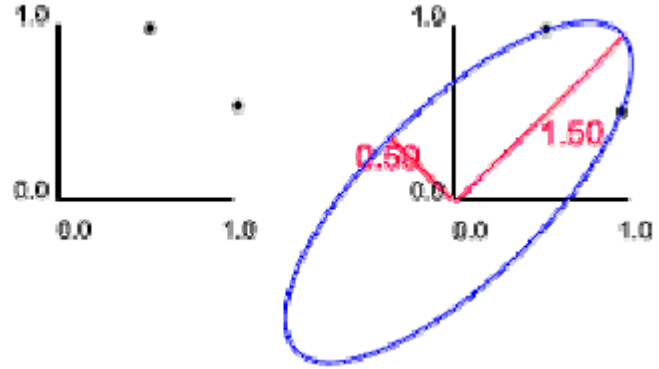
- **n dimensions in original data**
- **calculate n eigenvectors and eigenvalues**
- **choose only the first p eigenvectors, based on their eigenvalues**
- **final data set has only p dimensions**

Eigenvectors of a Correlation Matrix

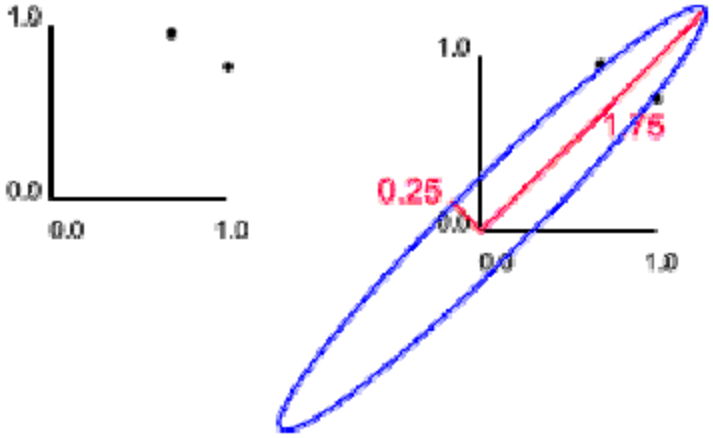
1.00 0.25
0.25 1.00



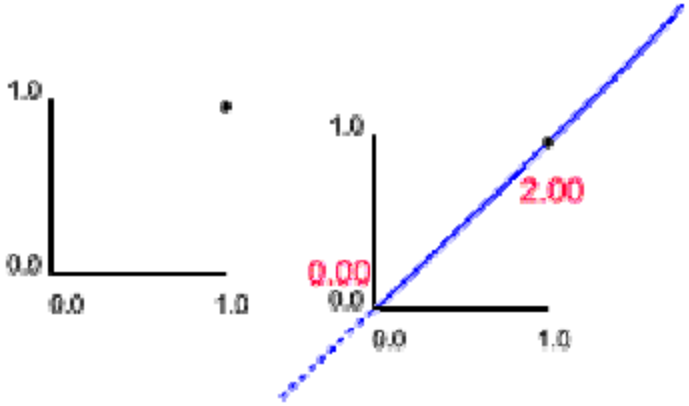
1.00 0.50
0.50 1.00



1.00 0.75
0.75 1.00



1.00 1.00
1.00 1.00



The Algebra of PCA

- first step is to calculate the cross-products matrix of variances and covariances (or correlations) among every pair of the *p* variables
- square, symmetric matrix
- diagonals are the variances, off-diagonals are the covariances.

	X_1	X_2
X_1	6.6707	3.4170
X_2	3.4170	6.2384

Variance-covariance Matrix

	X_1	X_2
X_1	1.0000	0.5297
X_2	0.5297	1.0000

Correlation Matrix

The Algebra of PCA

- in matrix notation, this is computed as

$$S = X'X$$

- where X is the *n* x *p* data matrix, with each variable centered (also standardized by SD if using correlations).

	X_1	X_2
X_1	6.6707	3.4170
X_2	3.4170	6.2384

Variance-covariance Matrix

	X_1	X_2
X_1	1.0000	0.5297
X_2	0.5297	1.0000

Correlation Matrix

Manipulating Matrices

- transposing: could change the columns to rows or the rows to columns
-

$$X = \begin{bmatrix} 10 & 0 & 4 \\ 7 & 1 & 2 \end{bmatrix} \qquad X' = \begin{bmatrix} 10 & 7 \\ 0 & 1 \\ 4 & 2 \end{bmatrix}$$

- multiplying matrices
  - must have the same number of columns in the premultiplicand matrix as the number of rows in the postmultiplicand matrix

The Algebra of PCA

- sum of the diagonals of the variance-covariance matrix is called the trace
- it represents the total variance in the data
- it is the mean squared Euclidean distance between each object and the centroid in *p*-dimensional space.

	X_1	X_2
X_1	6.6707	3.4170
X_2	3.4170	6.2384

Trace = 12.9091

	X_1	X_2
X_1	1.0000	0.5297
X_2	0.5297	1.0000

Trace = 2.0000

The Algebra of PCA

- finding the principal axes involves eigenanalysis of the cross-products matrix (S)
- the eigenvalues (latent roots) of S are solutions (λ) to the characteristic equation

$$\left|\mathbf{S} - \lambda \mathbf{I}\right| = \mathbf{0}$$

The Algebra of PCA

- the eigenvalues, $\lambda_1, \lambda_2, \ldots \lambda_p$ are the variances of the coordinates on each principal component axis
- the sum of all *p* eigenvalues equals the trace of S (the sum of the variances of the original variables).

	X_1	X_2
X_1	6.6707	3.4170
X_2	3.4170	6.2384

$\lambda_1 = 9.8783$

$\lambda_2 = 3.0308$

Trace = 12.9091

Note: $\lambda_1+\lambda_2$ =12.9091

The Algebra of PCA

- each eigenvector consists of *p* values which represent the "contribution" of each variable to the principal component axis
- eigenvectors are uncorrelated (orthogonal)
  - their cross-products are zero.

Eigenvectors

	u_1	u_2
X_1	0.7291	-0.6844
X_2	0.6844	0.7291

0.7291*(-0.6844) + 0.6844*0.7291 = 0

The Algebra of PCA

- coordinates of each object *i* on the k^{th} principal axis, known as the scores on PC *k*, are computed as

$$z_{ki} = u_{1k}x_{1i} + u_{2k}x_{2i} + \cdots + u_{pk}x_{pi}$$

- where Z is the *n* x *k* matrix of PC scores, X is the *n* x *p* centered data matrix and U is the *p* x *k* matrix of eigenvectors.

The Algebra of PCA

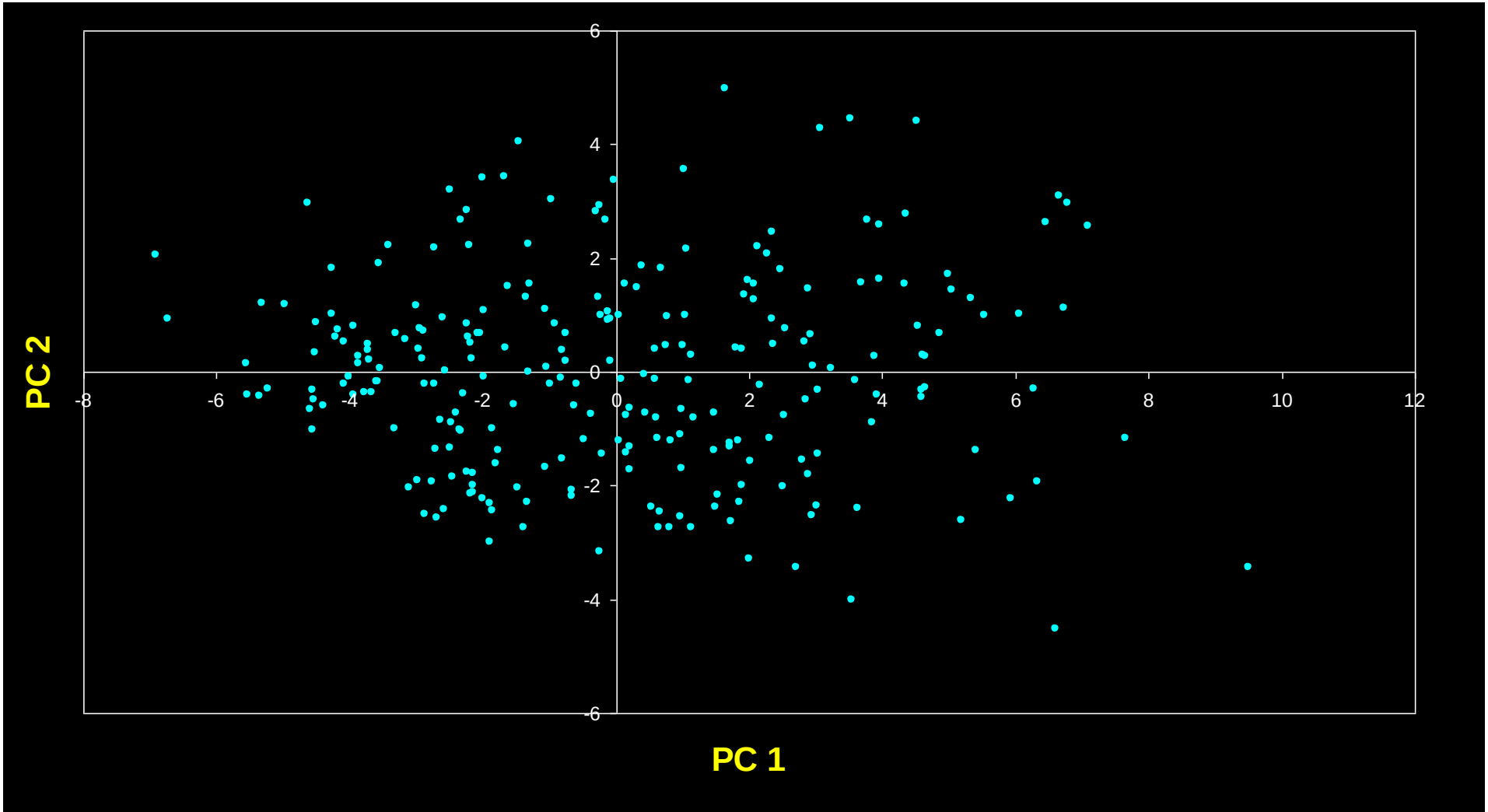
- variance of the scores on each PC axis is equal to the corresponding eigenvalue for that axis

- the eigenvalue represents the variance displayed ("explained" or "extracted") by the k^{th} axis

- the sum of the first k eigenvalues is the variance explained by the k-dimensional ordination.

λ_1 = 9.8783 λ_2 = 3.0308 Trace = 12.9091

PC 1 displays ("explains")
9.8783/12.9091 = 76.5% of the total variance



The Algebra of PCA

- The cross-products matrix computed among the *p* principal axes has a simple form:
  - all off-diagonal values are zero (the principal axes are uncorrelated)
  - the diagonal values are the eigenvalues.

	PC_1	PC_2
PC_1	9.8783	0.0000
PC_2	0.0000	3.0308

Variance-covariance Matrix of the PC axes

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:

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

- 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:

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

	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

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

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

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

Covariance matrix

- Eigenvectors:
  - Let **M be** an $n \times n$ matrix.
    - **v** is an *eigenvector* of M if $M \times \mathbf{v} = \lambda\mathbf{v}$
    - λ is called the *eigenvalue* associated with **v**
  - For any eigenvector **v** of **M** and scalar *a*,

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

  - Thus you can always choose eigenvectors of length 1:

$$\sqrt{v_1^2 + \ldots + v_n^2} = 1$$

  - If **M** has any eigenvectors, it has *n* of them, and they are orthogonal to one another.
  - Thus eigenvectors can be used as a new basis for a n-dimensional vector space.

PCA

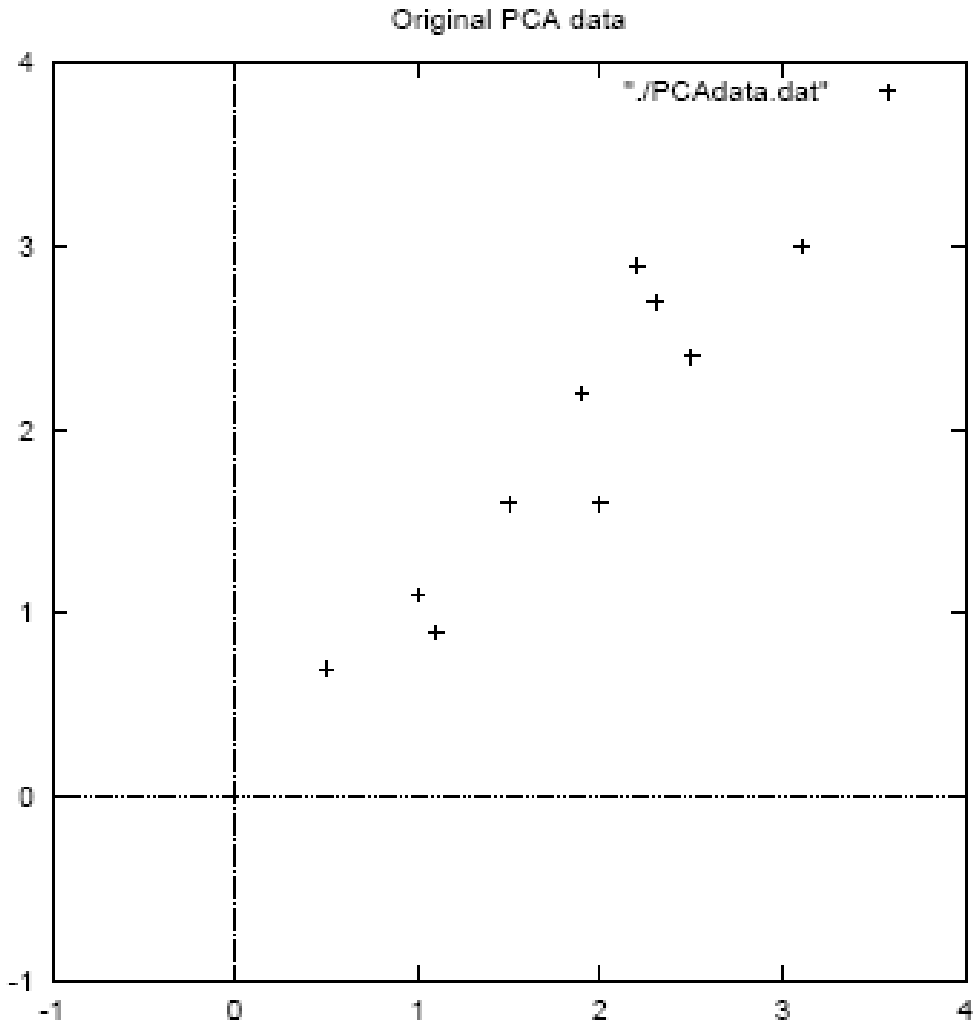
1. Given original data set S = {$\mathbf{x}^1$, ..., $\mathbf{x}^k$}, produce new set by subtracting the mean of attribute A_i from each x_i.

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

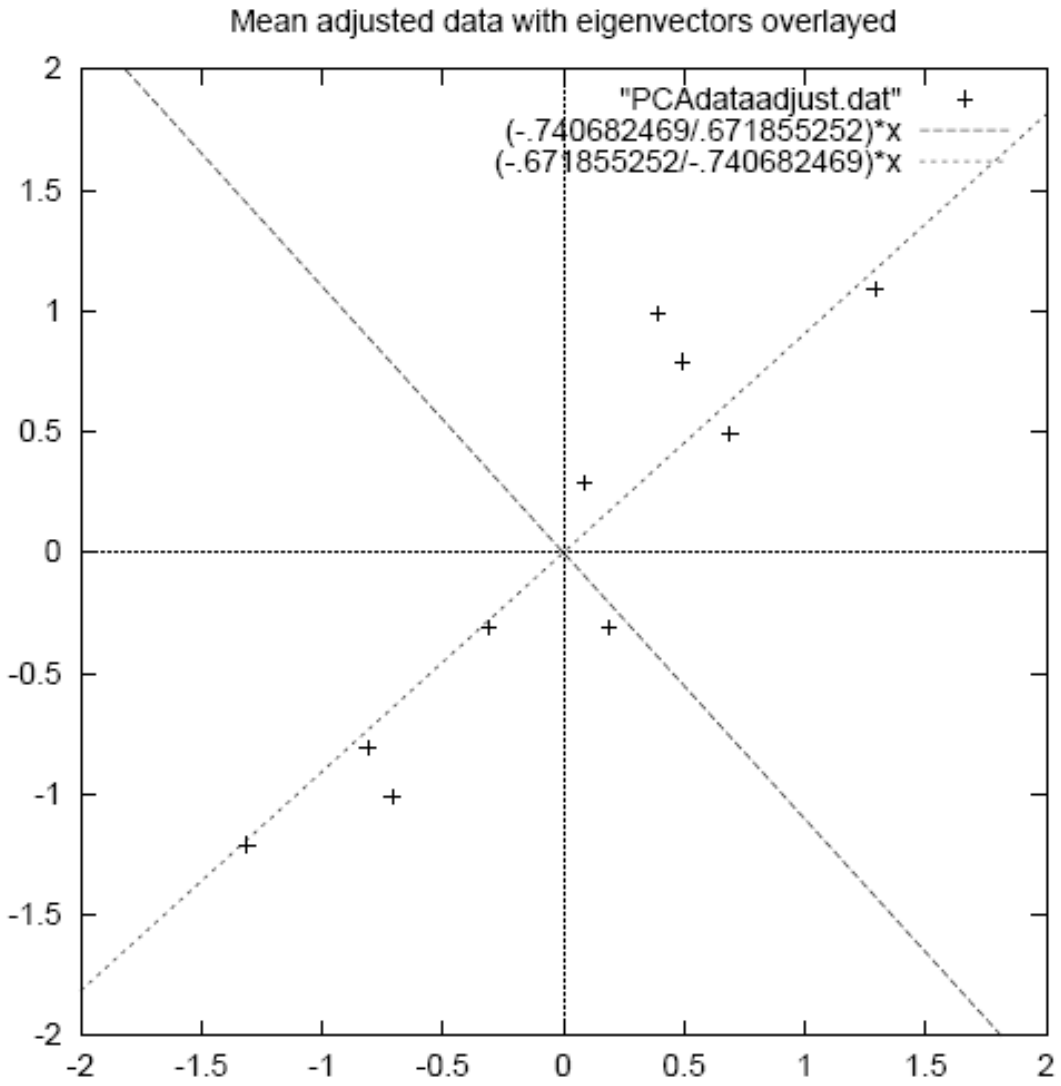
2. Calculate the covariance matrix:

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

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.

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

$$FeatureVector1 = \begin{pmatrix} -.677873399 & -.735178956 \\ -.735178956 & .677873399 \end{pmatrix}$$

or reduced dimension feature vector:

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

5. Derive the new data set.

$$TransformedData = RowFeatureVector \times RowDataAdjust$$

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

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

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

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

Transformed Data=

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

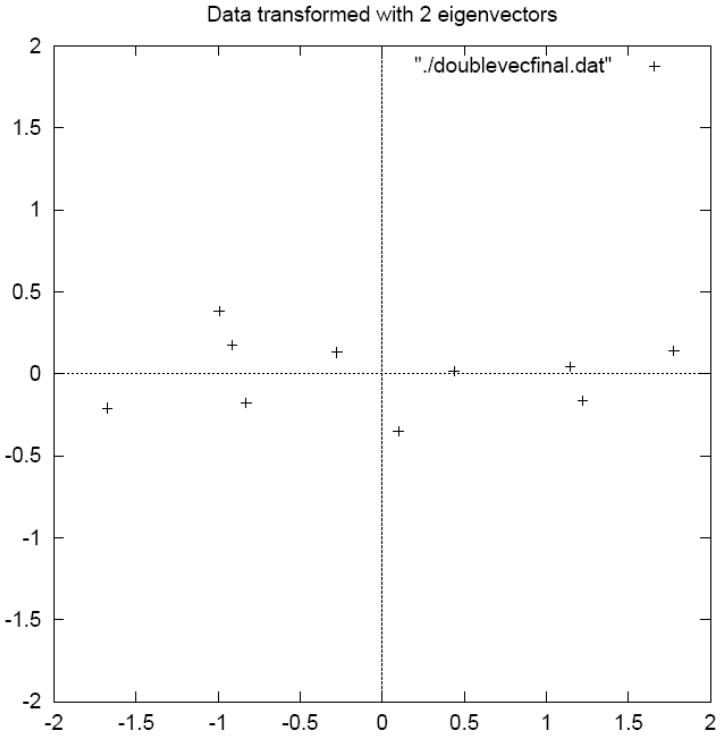
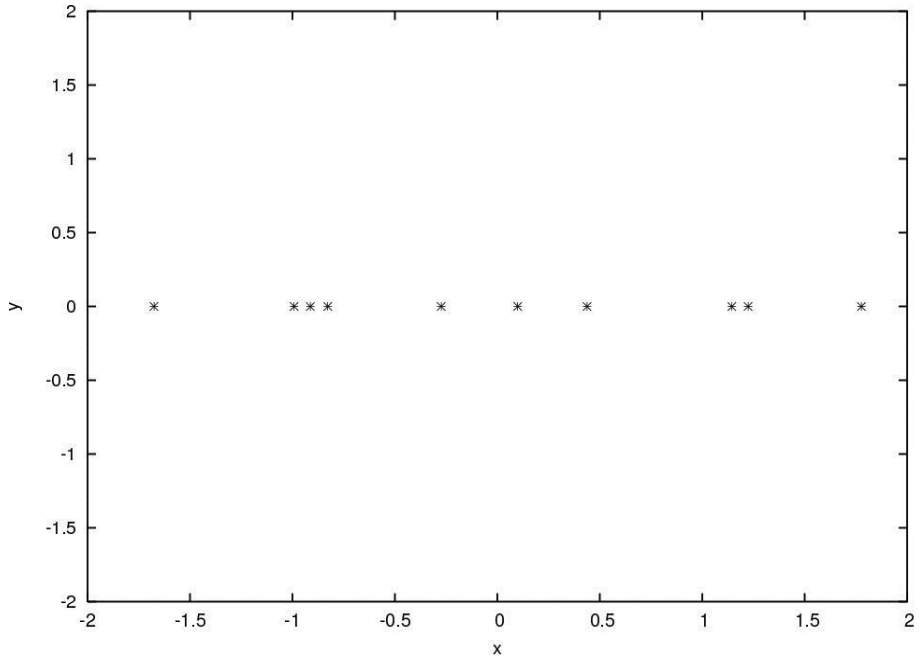


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

Transformed Data (Single eigenvector)

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



Reconstructing the original data

We did:

$$TransformedData = RowFeatureVector \times RowDataAdjust$$

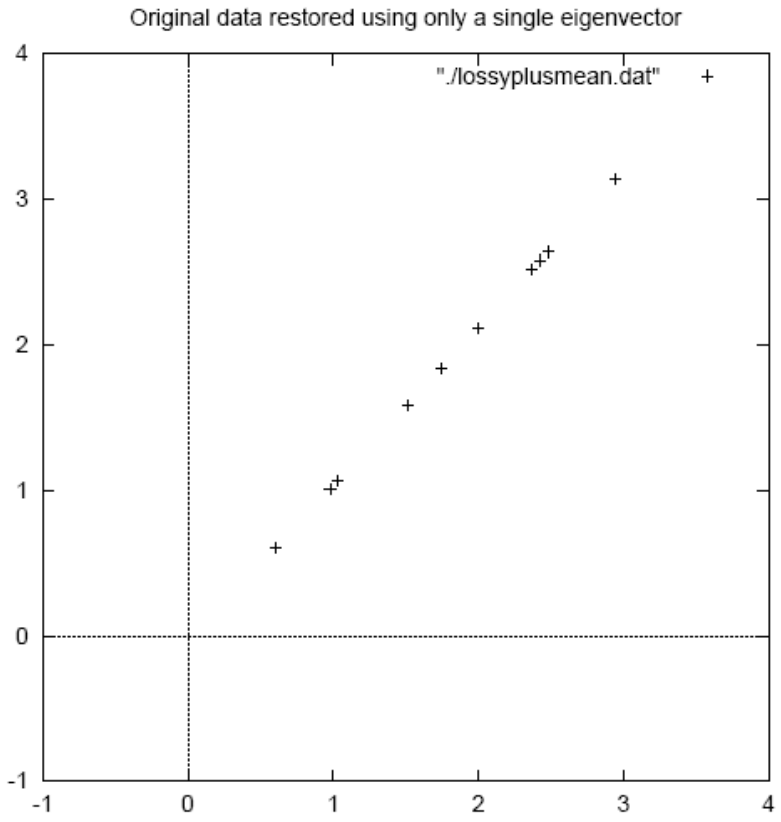
so we can do

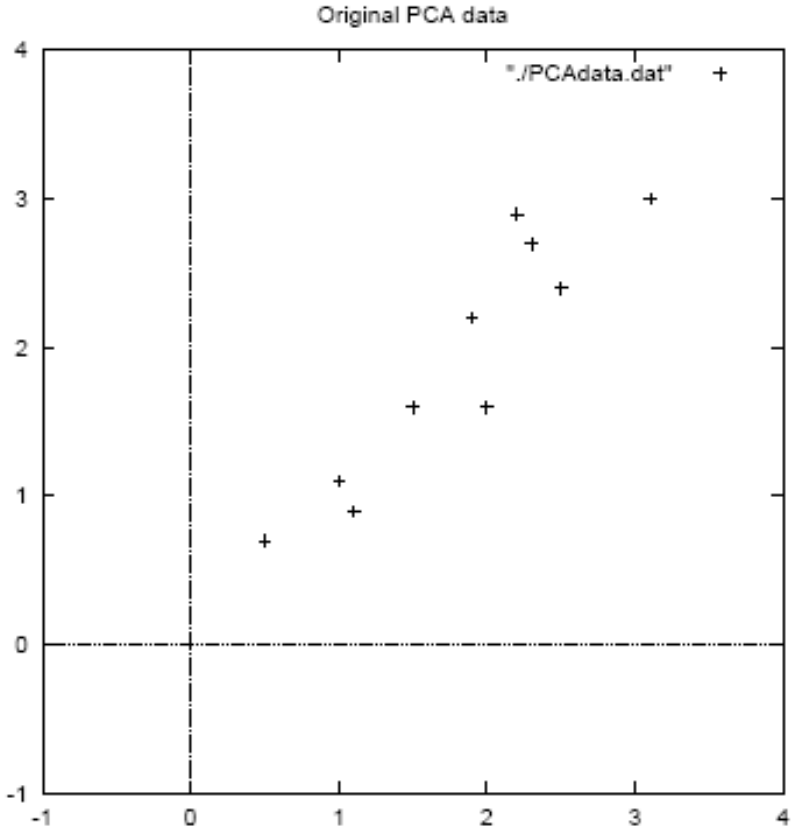
$$RowDataAdjust = RowFeatureVector^{-1} \times TransformedData$$

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

and

$$RowDataOriginal = RowDataAdjust + OriginalMean$$


Original data restored using only a single eigenvector
"./lossyplusmean.dat"
4
3
2
1
0
-1
-1
0
1
2
3
4


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

Example

- Compute the PCA of the following dataset:

(1,2),(3,3),(3,5),(5,4),(5,6),(6,5),(8,7),(9,8)

- Compute the sample covariance matrix is:

$$\hat{\Sigma} = \frac{1}{n}\sum_{k=1}^{n}(\mathbf{x}_k - \hat{\boldsymbol{\mu}})(\mathbf{x}_k - \hat{\boldsymbol{\mu}})^t$$

$$\Sigma_x = \begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix}$$

- The eigenvalues can be computed by finding the roots of the characteristic polynomial:

$$\Sigma_x v = \lambda v \Rightarrow |\Sigma_x - \lambda I| = 0$$
$$\Rightarrow \begin{vmatrix} 6.25 - \lambda & 4.25 \\ 4.25 & 3.5 - \lambda \end{vmatrix} = 0$$
$$\Rightarrow \boldsymbol{\lambda_1 = 9.34; \lambda_2 = 0.41}$$

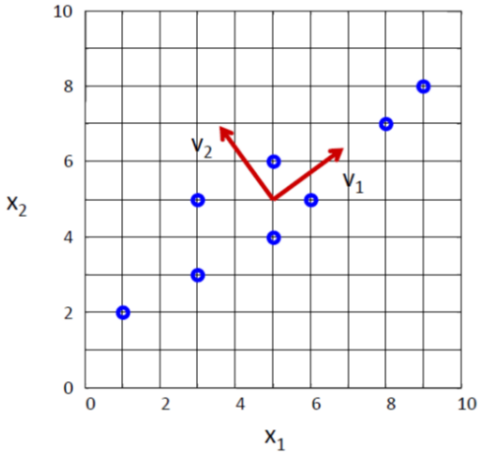
Example (cont'd)

- The eigenvectors are the solutions of the systems:

$$\Sigma_{\mathbf{x}} u_i = \lambda_i u_i$$

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} \lambda_1 v_{11} \\ \lambda_1 v_{12} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} 0.81 \\ 0.59 \end{bmatrix}$$

$$\begin{bmatrix} 6.25 & 4.25 \\ 4.25 & 3.5 \end{bmatrix} \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} \lambda_2 v_{21} \\ \lambda_2 v_{22} \end{bmatrix} \Rightarrow \begin{bmatrix} v_{21} \\ v_{22} \end{bmatrix} = \begin{bmatrix} -0.59 \\ 0.81 \end{bmatrix}$$



Note: if u_i is a solution, then cu_i is also a solution where $c \neq 0$.

Eigenvectors can be normalized to unit-length using:

$$\hat{v}_i = \frac{v_i}{\| v_i \|}$$

How do we choose K ?

- *K* is typically chosen based on how much information (variance) we want to preserve:

Choose the smallest K that satisfies the following inequality:

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > T \quad \textit{where } T \textit{ is a threshold } (e.g., 0.9)$$

- If T=0.9, for example, we "preserve" 90% of the information (variance) in the data.

- If K=N, then we "preserve" 100% of the information in the data (i.e., just a "change" of basis and $\hat{\mathbf{x}} = \mathbf{x}$)

Approximation Error

- The approximation error (or reconstruction error) can be computed by:

$$\| \mathbf{x} - \hat{\mathbf{x}} \|$$

where $$\hat{\mathbf{x}} = \sum_{i=1}^{K} y_i u_i + \bar{\mathbf{x}} = y_1 u_1 + y_2 u_2 + \ldots + y_K u_K + \bar{\mathbf{x}}$$

(reconstruction)

- It can also be shown that the approximation error can be computed as follows:

$$\| \mathbf{x} - \hat{\mathbf{x}} \| = \frac{1}{2} \sum_{i=K+1}^{N} \lambda_i$$

Data Normalization

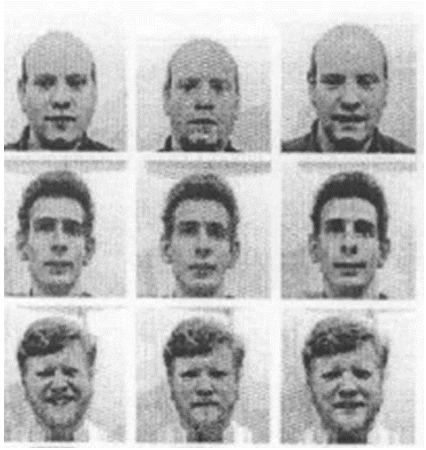
- The principal components are dependent on the ***units*** used to measure the original variables as well as on the ***range*** of values they assume.
- Data should always be normalized prior to using PCA.
- A common normalization method is to transform all the data to have zero mean and unit standard deviation:

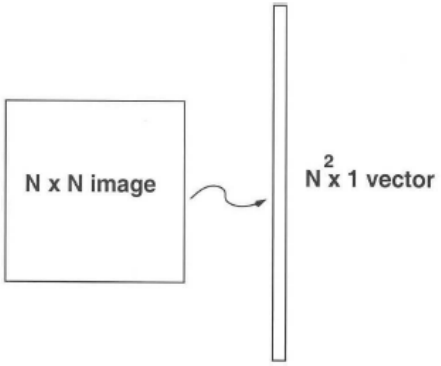
$$\frac{x_i - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of the i-th feature x_i

Application to Images

- The goal is to represent images in a space of lower dimensionality using PCA.
  - Useful for various applications, e.g., face recognition, image compression, etc.
- Given M images of size N x N, first represent each image as a 1D vector (i.e., by stacking the rows together).
  - Note that for face recognition, faces must be centered and of the same size.





The space of all face images

- When viewed as vectors of pixel values, face images are extremely high-dimensional
  - 100x100 image = 10,000 dimensions
  - Slow and lots of storage
- But very few 10,000-dimensional vectors are valid face images
- We want to effectively model the subspace of face images

slide by Derek Hoiem

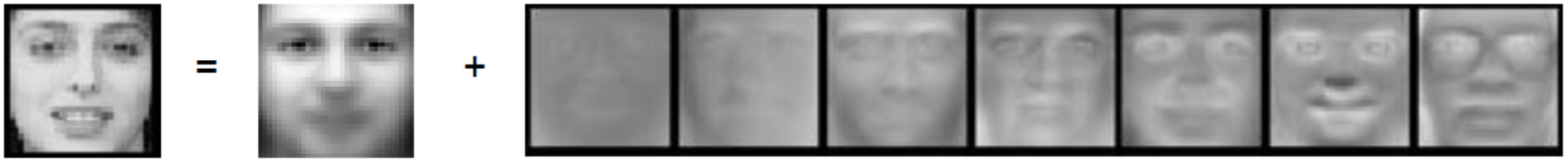
Representation and reconstruction

- Face **x** in "face space" coordinates:

$$\mathbf{x} \rightarrow [\mathbf{u}_1^{\mathrm{T}}(\mathbf{x} - \mu), \ldots, \mathbf{u}_k^{\mathrm{T}}(\mathbf{x} - \mu)]$$

$$= \quad w_1, \ldots, w_k$$

- Reconstruction:

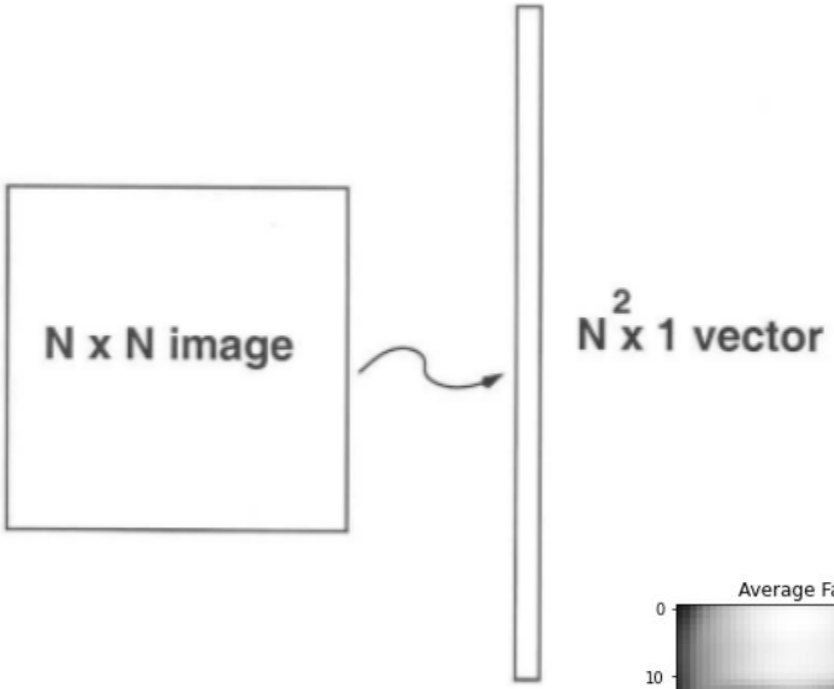


$\hat{x} = \mu + w_1u_1 + w_2u_2 + w_3u_3 + w_4u_4 + \ldots$

slide by Derek Hoiem

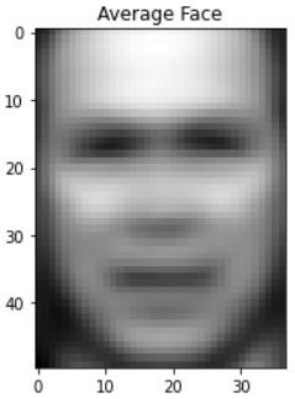
Steps

- We first convert these images into vectors of size N^2 such that:



Faces $x_1, x_2, x_3 \ldots x_m$

Calculate Average Face $\psi = \frac{1}{m} \sum_{i=1}^{m} x_i$



Subtract Average $\mathrm{a}_i = x_i - \psi$

- Now, we find Covariance matrix by multiplying *A* with A^T. A has dimensions $N^2 * M$, thus A^T has dimenions $M * N^2$. When we multiplied this gives us matrix of $N^2 * N^2$, which gives us N^2 eigenvectors of N^2 size which is not computationally efficient to calculate. So we calculate our covariance matrix by multiplying A^T and *A*. This gives us $M * M$ matrix which has *M (assuming* $M << N^2$*)* eigenvectors of size *M*.

Cov(A) = AA^T with the dimension N^2X N^2

Cov(A) = A^TA with the dimension MX M where M << N^2

$$A^T A\nu_i = \lambda_i \nu_i$$

$$AA^T A\nu_i = \lambda_i A\nu_i$$

$$C'u_i = \lambda_i u_i$$

where,

$$C' = AA^T \text{ and } u_i = A\nu_i$$

A Training set consisting of total *M* images

convetred

Face Vector Space

normalize face vectors:

calculate average face vector Ψ

then subtract the mean (average) face vector from EACH face vector to get the Normalized face vectors $\Phi_i = \Gamma_i - \Psi$

To calculate eigenvectors, we need to calculate the covariance matrix **C**

$\mathbf{C} = AA^T$ where $A = [\Phi_1, \Phi_2 \ldots \Phi_M]$

$N^2 \times M$

Face Recognition Using Eigenfaces

Application to Images (cont'd)

- The key challenge is that the covariance matrix Σ_x is now very large (i.e., N^2 x N^2) – see Step 3:

Step 3: compute the covariance matrix Σ_x

$$\Sigma_{\mathbf{x}} = \frac{1}{M}\sum_{i=1}^{M}\Phi_{\mathbf{i}}\Phi_i^T = \frac{1}{M}AA^T$$

where $A=[\Phi_1\ \Phi_2\ \ldots\ \Phi_M]$
(N^2 x M matrix)

- $\Sigma_{\mathbf{x}}$ is now an N^2 x N^2 matrix – computationally expensive to compute its eigenvalues/eigenvectors λ_i, u_i

$$(AA^T)u_i = \lambda_i u_i$$

Application to Images (cont'd)

- We will use a simple "trick" to get around this by relating the eigenvalues/eigenvectors of AA^T to those of A^TA.

- Let us consider the matrix A^TA instead (i.e., M x M matrix)
  - Suppose its eigenvalues/eigenvectors are μ_i, v_i

$$(A^TA)v_i = \mu_i v_i$$

  - Multiply both sides by A:

$$A(A^TA)v_i = A\mu_i v_i \quad \text{or} \quad \boxed{(AA^T)(Av_i) = \mu_i(Av_i)}$$

  - Assuming $(AA^T)u_i = \lambda_i u_i$

$$\boxed{\lambda_i = \mu_i \quad \text{and} \quad u_i = Av_i}$$

$A=[\Phi_1\ \Phi_2\ \ldots\ \Phi_M]$
(N^2 x M matrix)

Application to Images (cont'd)

- But do AA^T and A^TA have the same number of eigenvalues/eigenvectors?
  - AA^T can have up to N^2 eigenvalues/eigenvectors.
  - A^TA can have up to M eigenvalues/eigenvectors.
  - It can be shown that the M eigenvalues/eigenvectors of A^TA are also the M largest eigenvalues/eigenvectors of AA^T
- **Steps 3-5** of PCA need to be updated as follows:

Application to Images (cont'd)

Step 3 compute A^TA (i.e., instead of AA^T)

Step 4: compute μ_i, v_i of A^TA

Step 4b: compute λ_i, u_i of AA^T using $\lambda_i=\mu_i$ and $u_i=Av_i$, then normalize u_i to unit length.

Step 5: dimensionality reduction step – approximate **x** using only the first K eigenvectors (K<M):

$$\hat{\mathbf{x}} - \overline{\mathbf{x}} = \sum_{i=1}^{K} y_i u_i = y_1 u_1 + y_2 u_2 + \ldots + y_K u_K$$

each image can be represented by a K-dimensional vector

$$\hat{\mathbf{x}} - \overline{\mathbf{x}} : \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_K \end{bmatrix}$$

Example

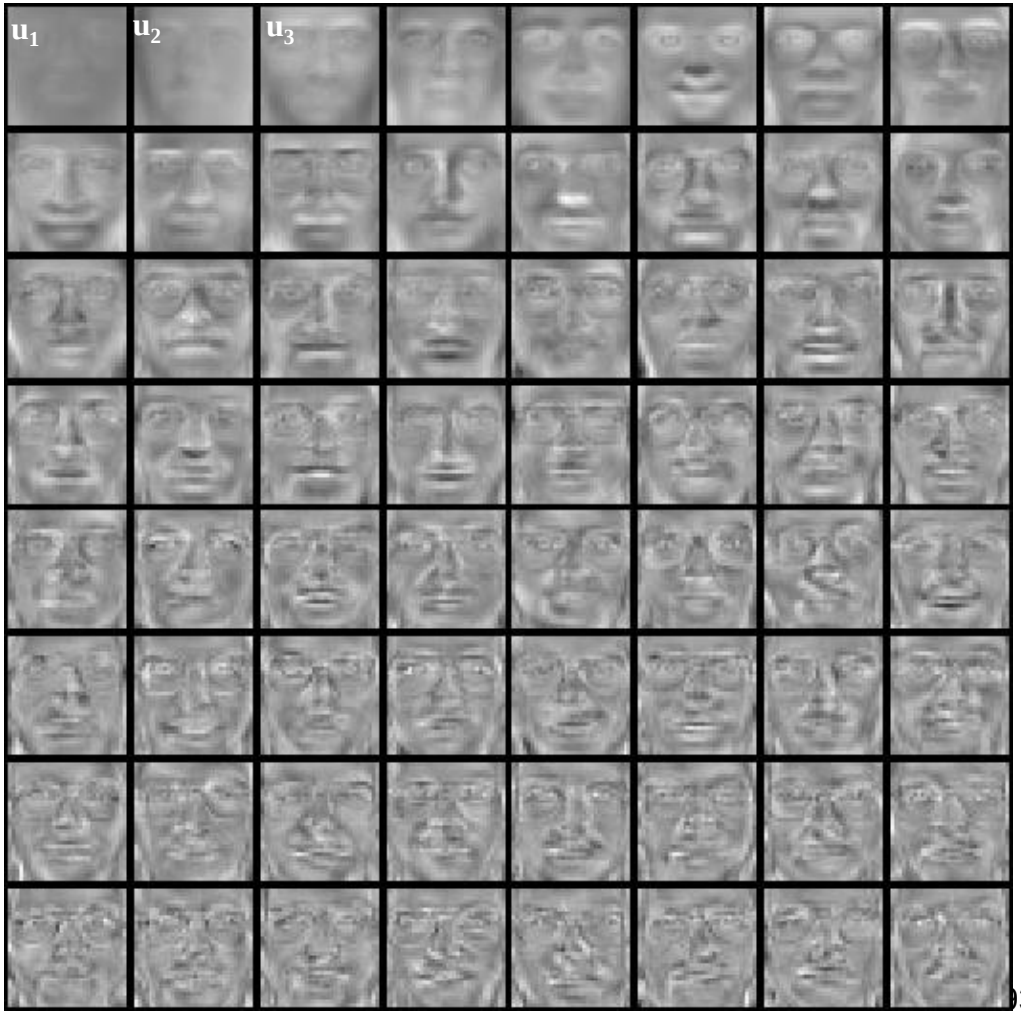
Dataset

Example (cont'd)

Mean face: $\bar{\mathbf{x}}$

Top eigenvectors: $\mathbf{u}_1,\ldots\mathbf{u}_k$

(visualized as an image - eigenfaces)



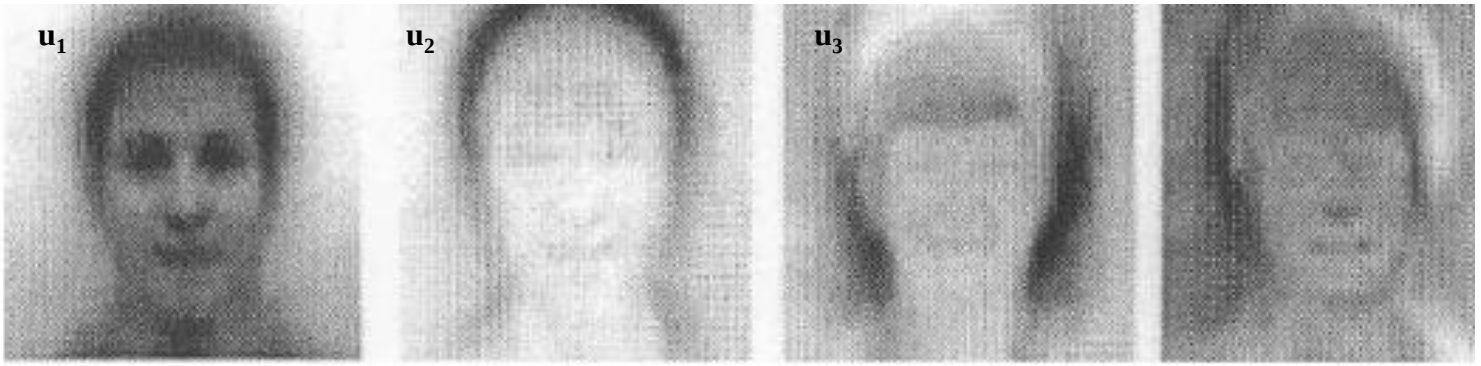
Example (cont'd)

- How can you visualize the eigenvectors (eigenfaces) as an image?
  - Their values must be first mapped to integer values in the interval [0, 255] (required by PGM format).
  - Suppose f_{min} and f_{max} are the min/max values of a given eigenface (could be negative).
  - If $x \epsilon [f_{min}, f_{max}]$ is the original value, then the new value $y \epsilon [0,255]$ can be computed as follows:

$$y=(int)255(x - f_{min})/(f_{max} - f_{min})$$

Application to Images (cont'd)

- Interpretation: represent a face in terms of eigenfaces



$$\hat{\mathbf{x}} = \sum_{i=1}^{K} y_i u_i = y_1 u_1 + y_2 u_2 + \ldots + y_K u_K + \bar{\mathbf{x}}$$

$$\hat{\mathbf{x}} - \bar{\mathbf{x}} : \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_K \end{bmatrix}$$

Case Study: Eigenfaces for Face Detection/Recognition

- M. Turk, A. Pentland, "Eigenfaces for Recognition", *Journal of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71-86, 1991.

- Face Recognition
  - The simplest approach is to think of it as a template matching problem.
  - Problems arise when performing recognition in a high-dimensional space.
  - Use *dimensionality reduction*!

Face Recognition Using Eigenfaces

- Process the image database (i.e., set of images with labels) – typically referred to as "training" phase:
  - Compute PCA space using image database (i.e., training data)
  - Represent each image in the database with K coefficients Ω_i

$$\Omega_i = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_K \end{bmatrix}$$

Face Recognition Using Eigenfaces

Given an unknown face **x**, follow these steps:

Step 1: Subtract mean face $\bar{\mathbf{x}}$ (computed from training data)

$$\Phi = \mathbf{x} - \bar{\mathbf{x}}$$

Step 2: Project unknown face in the eigenspace:

$$\hat{\Phi} = \sum_{i=1}^{K} y_i u_i \quad where \quad y_i = \Phi^T u_i$$

$$\Omega: \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_K \end{bmatrix}$$

Step 3: Find closest match Ω_i from training set using:

$$e_r = \min_i \| \Omega - \Omega_i \| = \min_i \sum_{j=1}^{K} (y_j - y_j^i)^2 \quad \textbf{or} \quad \min_i \sum_{j=1}^{K} \frac{1}{\lambda_j} (y_j - y_j^i)^2$$

Euclidean distance

Mahalanobis distance

The distance $\boldsymbol{e_r}$ is called *distance in face space (difs)*

Step 4: Recognize **x** as person "k" where k is the ID linked to Ω_i

Note: for intruder rejection, we need $e_r < T_r$, for some threshold T_r

Face detection vs recognition

Detection

Recognition

“Sally”

Face Detection Using Eigenfaces

Given an unknown image **x**, follow these steps:

Step 1: Subtract mean face $\bar{\mathbf{x}}$ (computed from training data):

$$\Phi = \mathbf{x} - \bar{\mathbf{x}}$$

Step 2: Project unknown face in the eigenspace:

$$\hat{\Phi} = \sum_{i=1}^{K} y_i u_i \qquad where \;\; y_i = \Phi^T u_i$$

Step 3: Compute $e_d = \| \Phi - \hat{\Phi} \|$

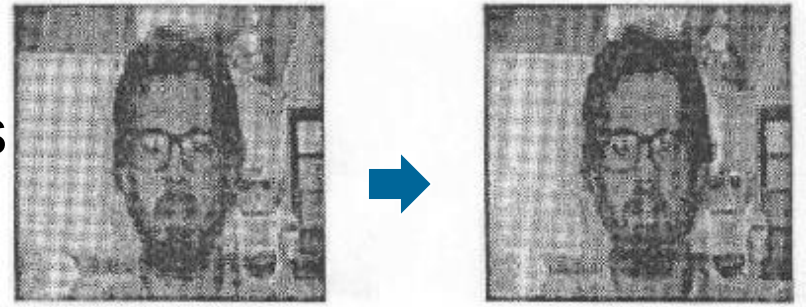
The distance $\boldsymbol{e_d}$ is called *distance from face space (dffs)*

Step 4: if $e_d < T_d$, then **x** is a face.

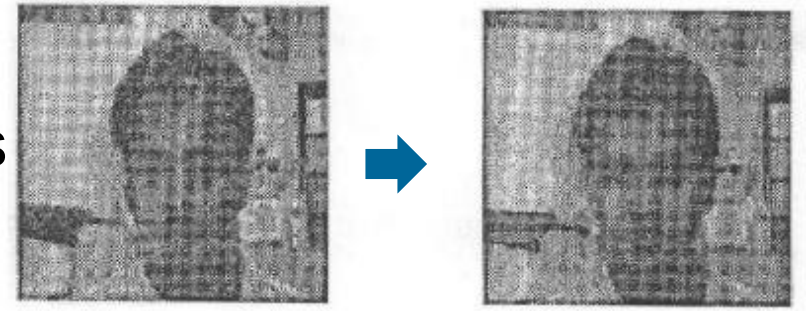
Eigenfaces

Input Reconstructed

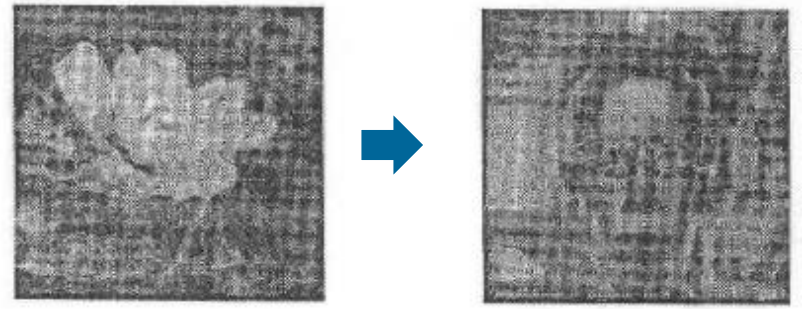
Reconstructed image looks like a face.



Reconstructed image looks like a face.



Reconstructed image looks like a face again!



Reconstruction from partial information

- Robust to partial face occlusion.

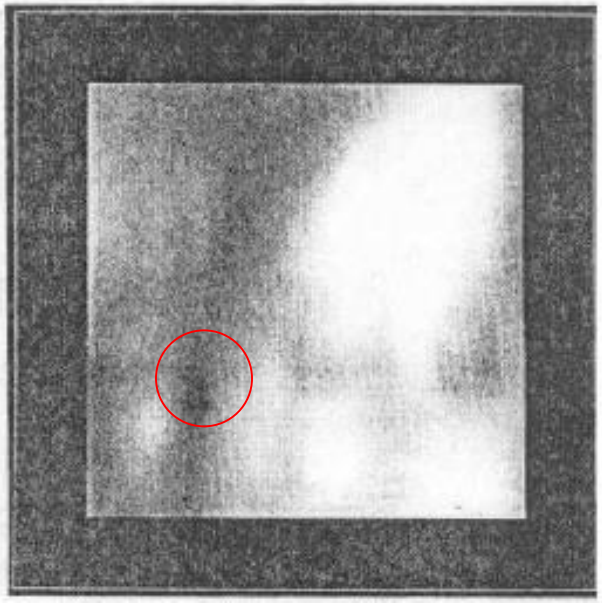
Input Reconstructed

Eigenfaces

- Can be used for face detection, tracking, and recognition!

Visualize **dffs** as an image:

$$e_d = \| \Phi - \hat{\Phi} \|$$



Dark: small distance
Bright: large distance

Limitations

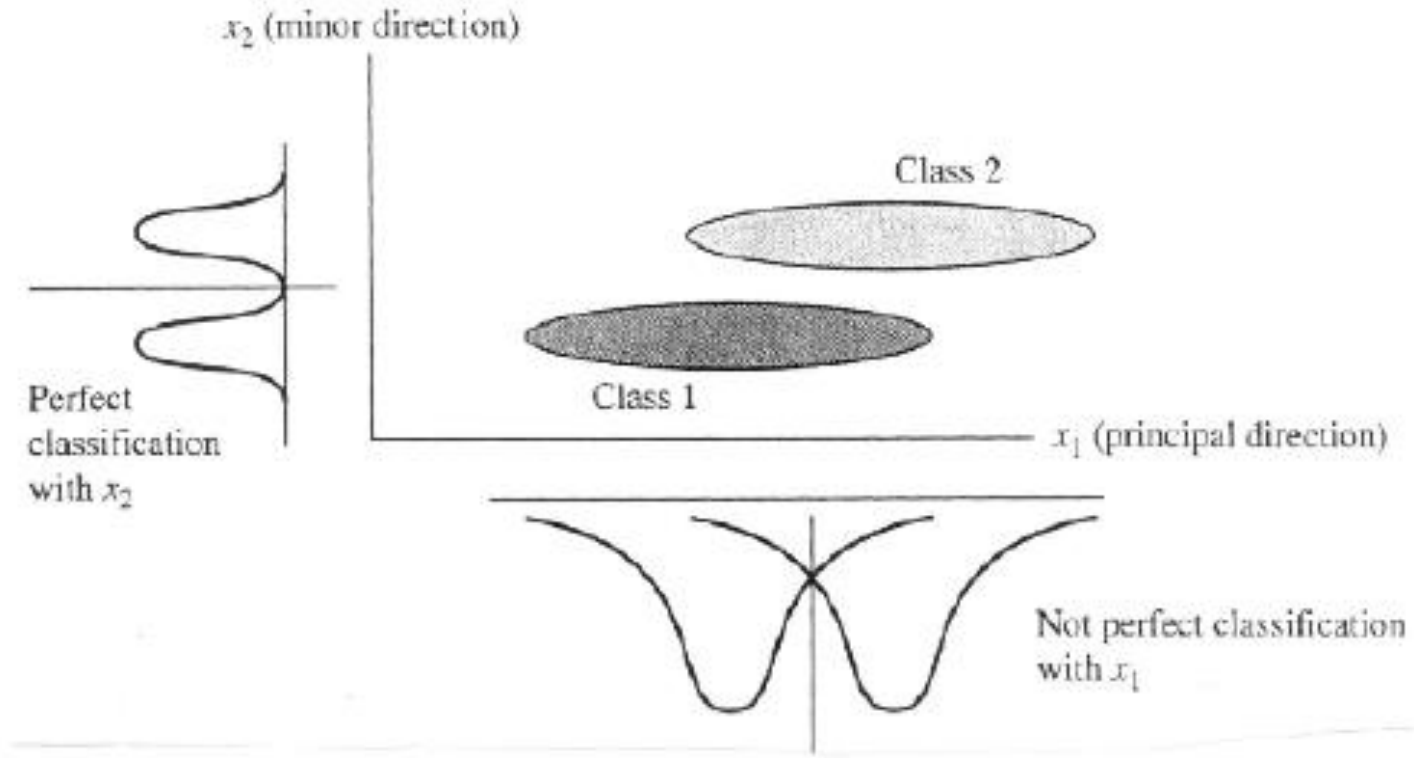
- Background changes cause problems
  - De-emphasize the outside of the face (e.g., by multiplying the input image by a 2D Gaussian window centered on the face).
- Light changes degrade performance
  - Light normalization might help but this is a challenging issue.
- Performance decreases quickly with changes to face size
  - Scale input image to multiple sizes.
  - Multi-scale eigenspaces.
- Performance decreases with changes to face orientation (but not as fast as with scale changes)
  - Out-of-plane rotations are more difficult to handle.
  - Multi-orientation eigenspaces.

Limitations (cont'd)

- Not robust to misalignment.

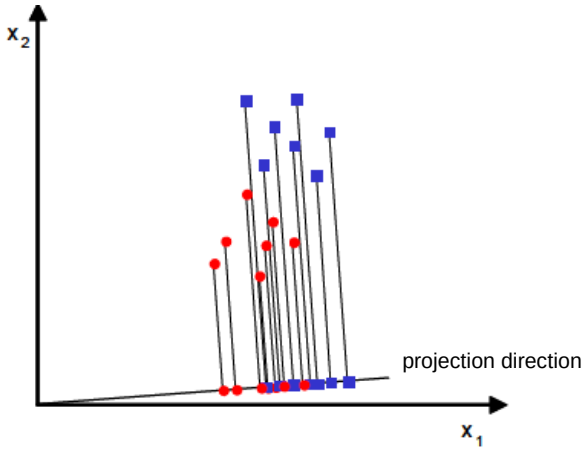
Limitations (cont'd)

- PCA is not always an optimal dimensionality-reduction technique for classification purposes.

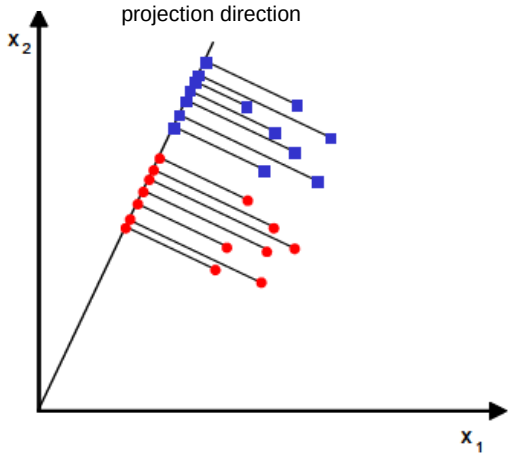


Linear Discriminant Analysis (LDA)

- What is the goal of LDA?
  - Seeks to find directions along which the classes are best separated (i.e., increase discriminatory information).
  - It takes into consideration the scatter (i.e., variance) within-classes and between-classes.



Bad separability



Good separability

Linear Discriminant Analysis (LDA) (cont'd)

- Let us assume C classes with each class containing M_i samples, i=1,2,..,C and M the total number of samples:

$$M = \sum_{i=1}^{C} M_i$$

- Let $\boldsymbol{\mu}_i$ is the mean of the i-th class, i=1,2,…,C and $\boldsymbol{\mu}$ is the mean of the whole dataset:

$$\boldsymbol{\mu} = \frac{1}{C} \sum_{i=1}^{C} \boldsymbol{\mu}_i$$

Within-class scatter matrix

$$S_w = \sum_{i=1}^{C} \sum_{j=1}^{M_i} (\mathbf{x}_{ij} - \boldsymbol{\mu}_i)(\mathbf{x}_{ij} - \boldsymbol{\mu}_i)^T$$

Between-class scatter matrix

$$S_b = \sum_{i=1}^{C} (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$$

Linear Discriminant Analysis (LDA) (cont'd)

- Suppose the desired projection transformation is:

$$\mathbf{y} = U^T \mathbf{x}$$

- Suppose the scatter matrices of the projected data $\mathbf{y}$ are:

$$\tilde{S}_b, \tilde{S}_w$$

- LDA seeks transformations that maximize the between-class scatter and minimize the within-class scatter:

$$\max \frac{|\tilde{S}_b|}{|\tilde{S}_w|} \quad \text{or} \quad \max \frac{|U^T S_b U|}{|U^T S_w U|}$$

Linear Discriminant Analysis (LDA) (cont'd)

- It can be shown that the columns of the matrix *U* are the eigenvectors (i.e., called *Fisherfaces*) corresponding to the largest eigenvalues of the following ***generalized eigen-problem***:

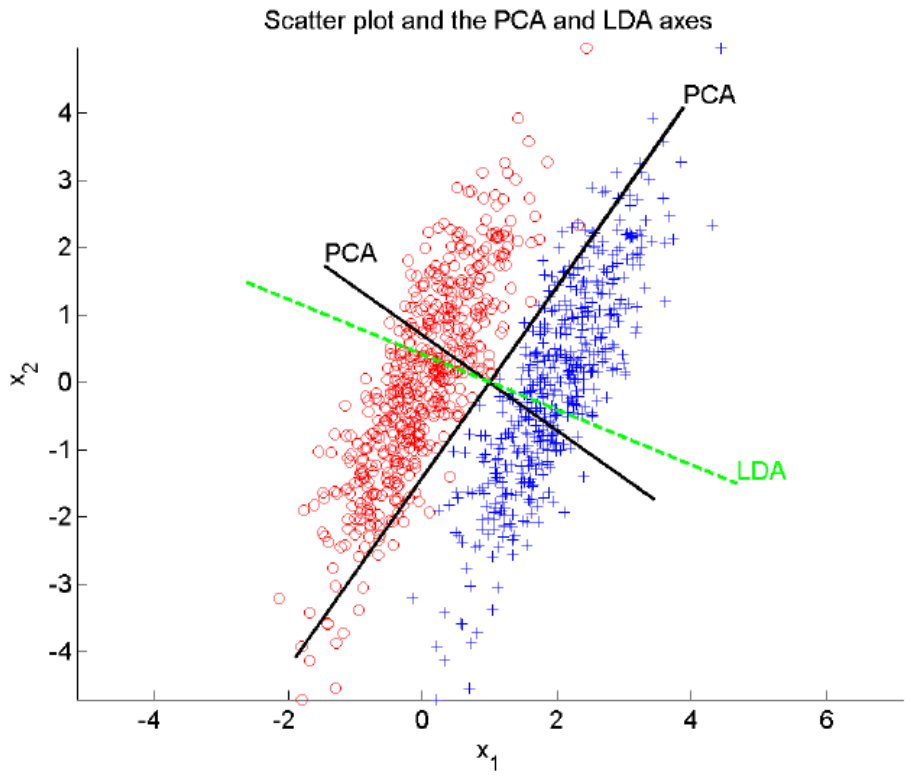
$$S_b u_k = \lambda_k S_w u_k$$

  - It can be shown that S_b has at most rank C-1; therefore, the max number of eigenvectors with non-zero eigenvalues is C-1, that is:

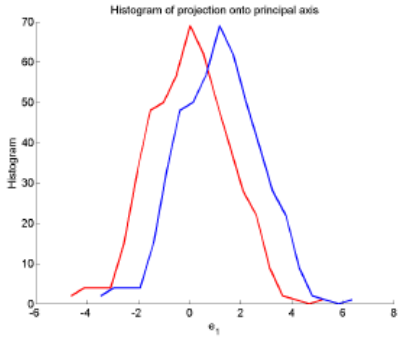
max dimensionality of LDA sub-space is C-1

e.g., when C=2, we always end up with one LDA feature no matter what the original number of features was!

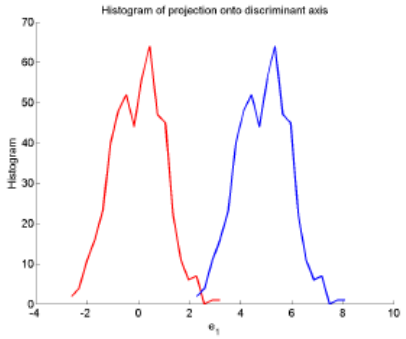
Example



(a) Scatter plot.



(b) Projection onto the first PCA axis.



(c) Projection onto the first LDA axis.

Linear Discriminant Analysis (LDA) (cont'd)

- If S_w is non-singular, we can solve a *conventional* eigenvalue problem as follows:

$$S_b u_k = \lambda_k S_w u_k$$

$$S_w^{-1} S_b u_k = \lambda_k u_k$$

- In practice, S_w is singular due to the high dimensionality of the data (e.g., images) and a much lower number of data ($M << N$)

Linear Discriminant Analysis (LDA) (cont'd)

- To alleviate this problem, PCA could be applied first:

1) First, apply PCA to reduce data dimensionality:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ x_N \end{bmatrix} \xrightarrow{PCA} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_M \end{bmatrix}$$

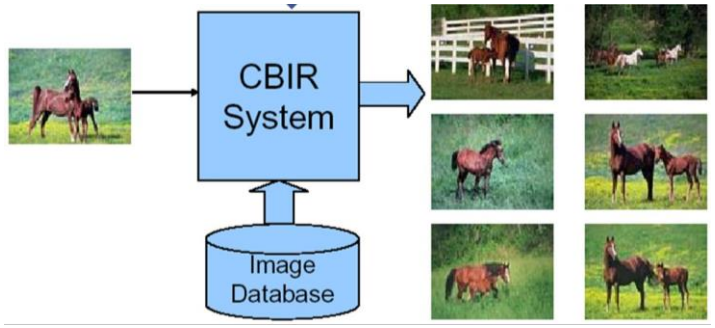
2) Then, apply LDA to find the most discriminative directions:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_M \end{bmatrix} \xrightarrow{LDA} \mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ . \\ . \\ z_K \end{bmatrix}$$

Case Study I

- D. Swets, J. Weng, "Using Discriminant Eigenfeatures for Image Retrieval", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, pp. 831-836, 1996.

- Content-based image retrieval:



  - Application: *query-by-example* content-based image retrieval

  - Question: how to select a *good* set of image features?

Case Study I (cont'd)

- Assumptions
  - Well-framed images are required as input for training and query-by-example test probes.
  - Only a small variation in the size, position, and orientation of the objects in the images is allowed.

Figure 6. Representatives of training images from some of the various classes learned. Objects of interest are at the center of the fovea images. In the learning phase for this experiment, the training images were generated using manual extraction of the areas of interest.

Case Study I (cont'd)

- Terminology
  - Most Expressive Features (MEF): features obtained using PCA.
  - Most Discriminating Features (MDF): features obtained using LDA.

- Numerical instabilities
  - Computing the eigenvalues/eigenvectors of $S_w^{-1}S_B u_k = \lambda_k u_k$ could lead to *unstable* computations since $S_w^{-1}S_B$ is not always symmetric.
  - Check the paper for more details about how to deal with this issue.

Case Study I (cont'd)

- Comparing projection directions between MEF with MDF:
  - PCA eigenvectors show the tendency of PCA to capture major variations in the training set such as lighting direction.
  - LDA eigenvectors discount those factors unrelated to classification.

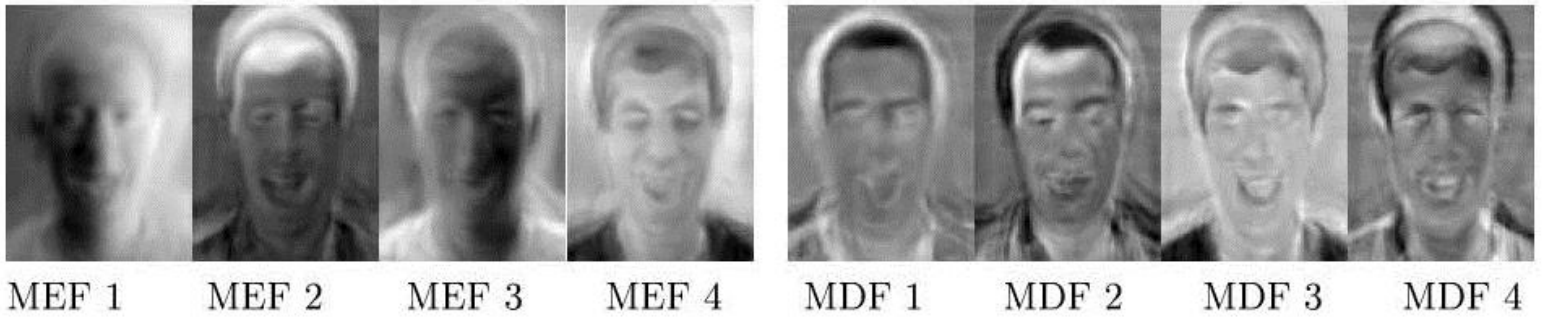
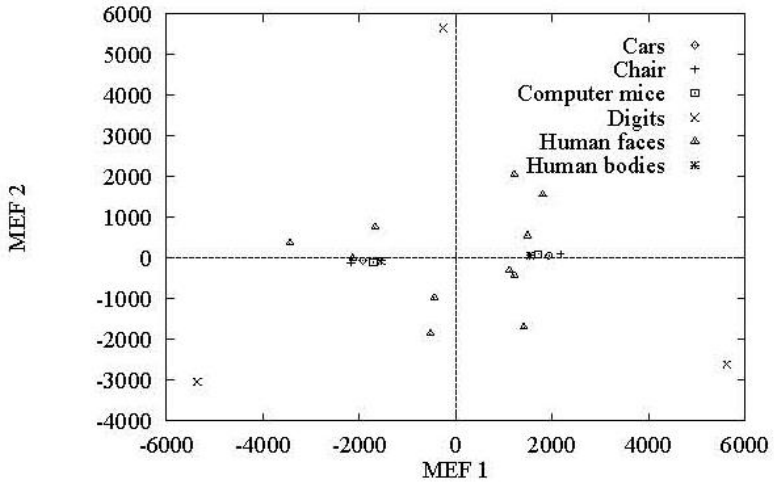


Figure 2. A sample of MEF and MDF vectors treated as images. The MEF vectors show the tendency of the principal components to capture major variations in the training set, such as lighting direction. The MDF vectors show the ability of the MDFs to discount those factors unrelated to classification. The training images used to produce these vectors are courtesy of the Weizmann Institute.

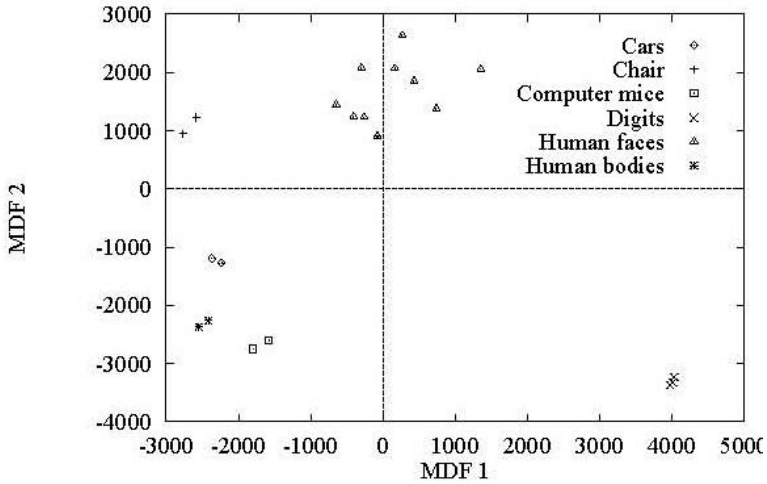
Case Study I (cont'd)

- Clustering effect



(a) MEF space

PCA space



(b) MDF space

LDA space

Case Study I (cont'd)

- Methodology

  1) Represent each training image in terms of MDFs (or MEFs for comparison).

  2) Represent a query image in terms of MDFs (or MEFs for comparson).

  3) Find the ***k* closest neighbors** (e.g., using Euclidean distance).

Case Study I (cont'd)

- Experiments and results

Face images

- A set of face images was used with 2 expressions, 3 lighting conditions.
- Testing was performed using a disjoint set of images.

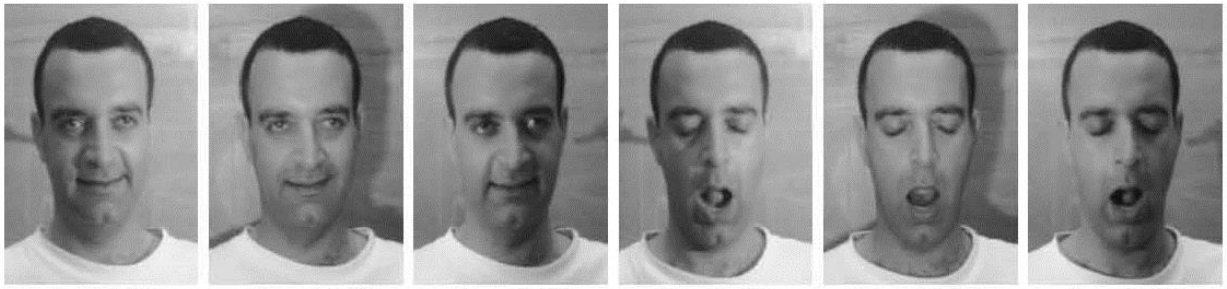


Figure 5. A sample of the Weizmann Institute face data. The frontal images for an individual contain two expressions; for each expression, a set of three images with differing lighting conditions forms the set of images available for an individual.

Case Study I (cont'd)

Top match (k=1)

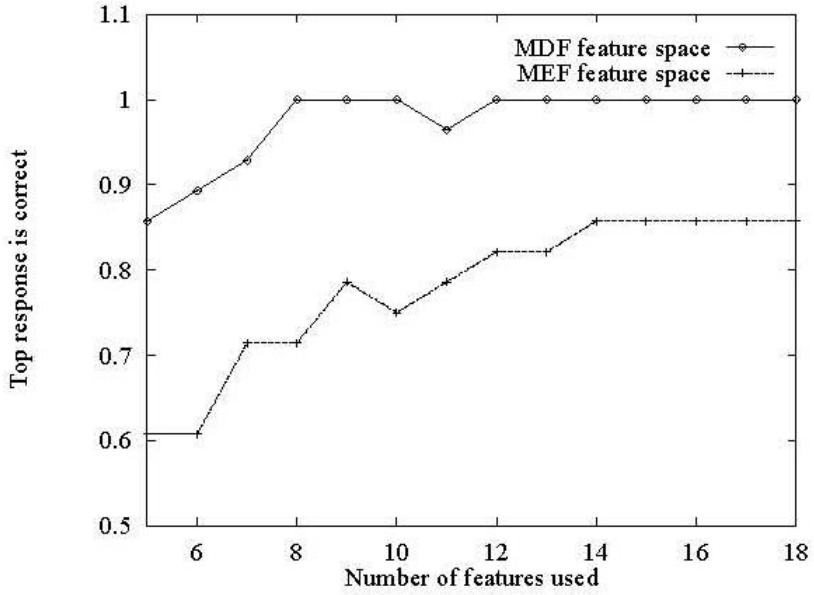
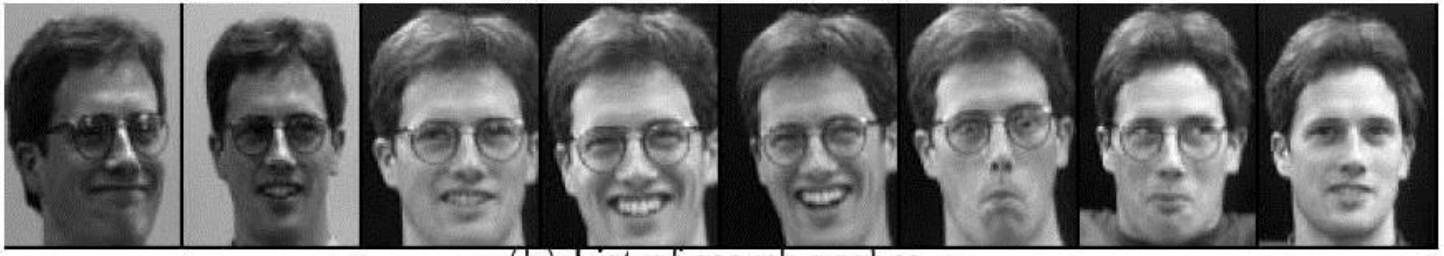


Figure 4. The performance of the system for different numbers of MEF and MDF features, respectively. The number of features from the subspace used was varied to show how the MDF subspace outperforms the MEF subspace. 95% of the variance for the MDF subspace was attained when 15 features were used; 95% of variance for the MEF subspace did not occur until 37 features were used. Using 95% of the MEF variance resulted in an 89% recognition rate, and that rate was not improved using more features.

Case Study I (cont'd)

- Examples of correct search probes

(a) List of training images



(b) List of search probes

Figure 8. Example of how well within-class variation is handled. The system correctly retrieved images from the class defined by the training samples for each of the search probes.

Case Study I (cont'd)

- Example of a failed search probe

(a) Search probe (b) Training images

Figure 7. Example of a failed search probe. The retrieval failed to select the appropriate class due to a lack of 3D rotation in the set of training images.

Case Study II

- A. Martinez, A. Kak, "PCA versus LDA", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 2, pp. 228-233, 2001.

- Is LDA always better than PCA?
  - There has been a tendency in the computer vision community to prefer LDA over PCA.
  - This is mainly because LDA deals directly with discrimination between classes while PCA does not pay attention to the underlying class structure.

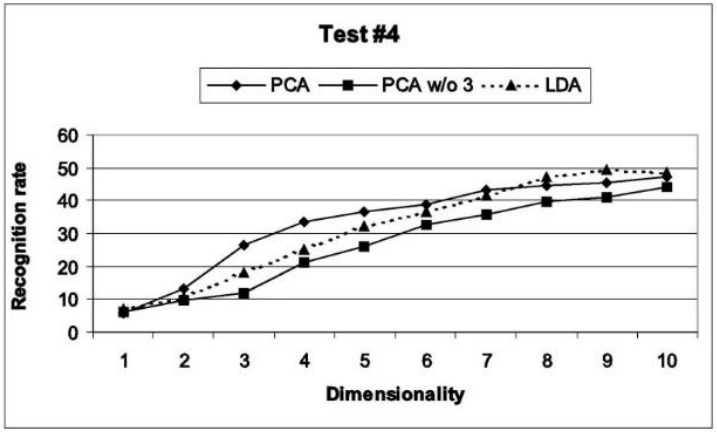
Case Study II (cont'd)

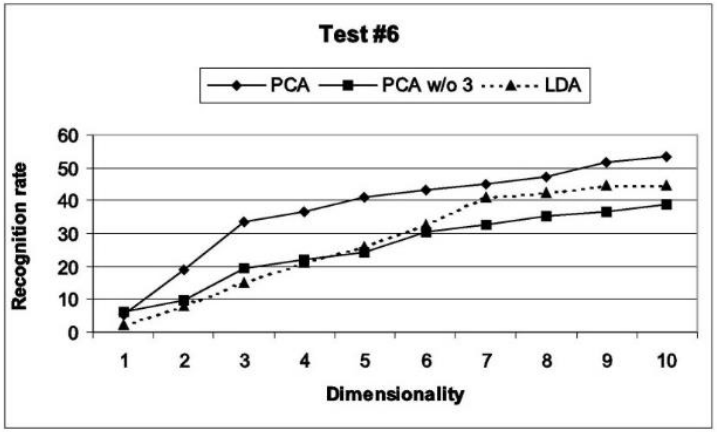
AR database

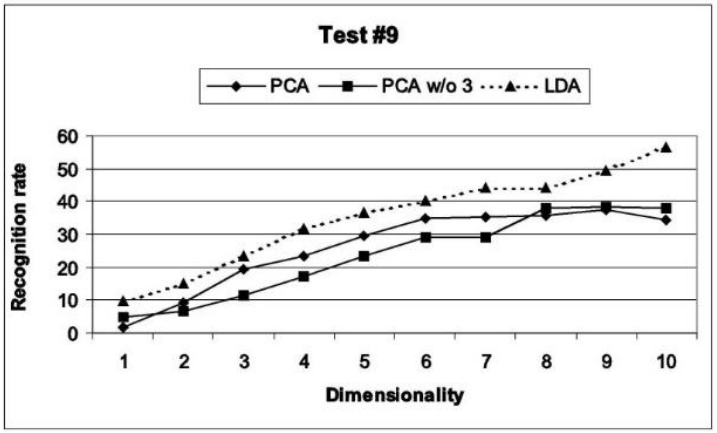
Fig. 3. Images of one subject in the AR face database. The images (a)-(m) were taken during one session and the images (n)-(z) at a different session.

Case Study II (cont'd)

LDA is not always better when the training set is small



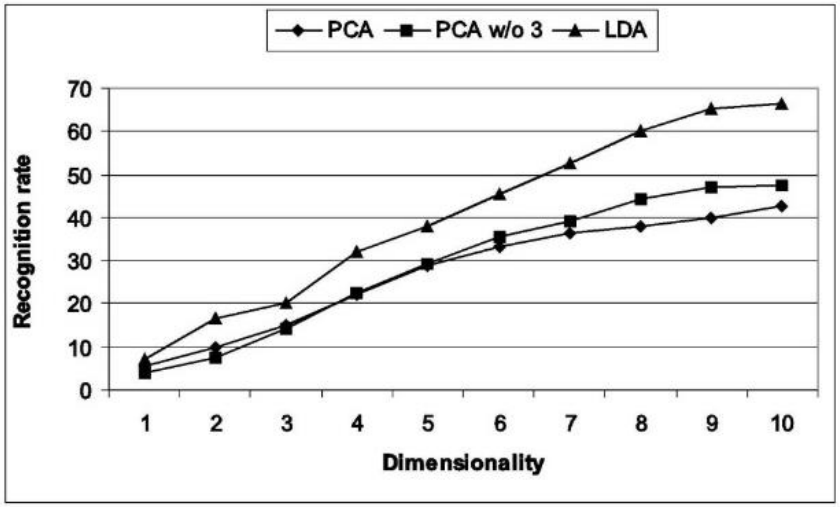


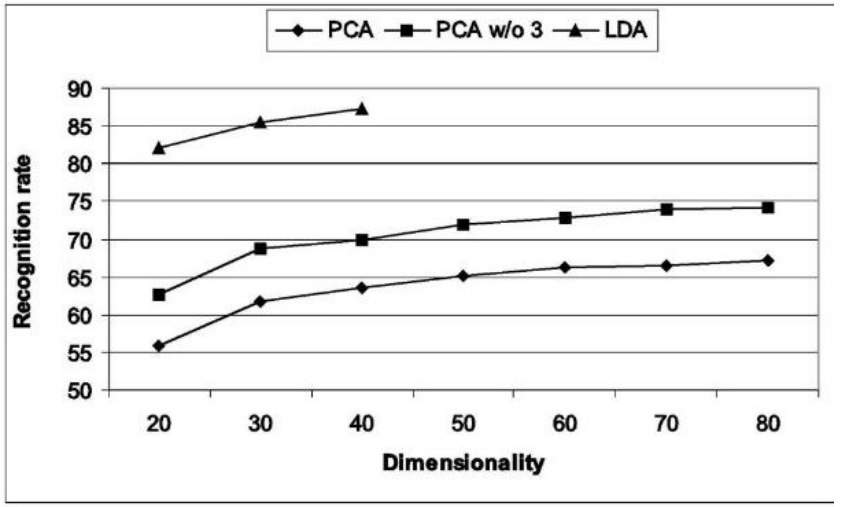


PCA w/o 3: not using the first three principal components that seem to encode mostly variations due to lighting

Case Study II (cont'd)

LDA outperforms PCA when the training set is large





PCA w/o 3: not using the first three principal components that seem to encode mostly variations due to lighting