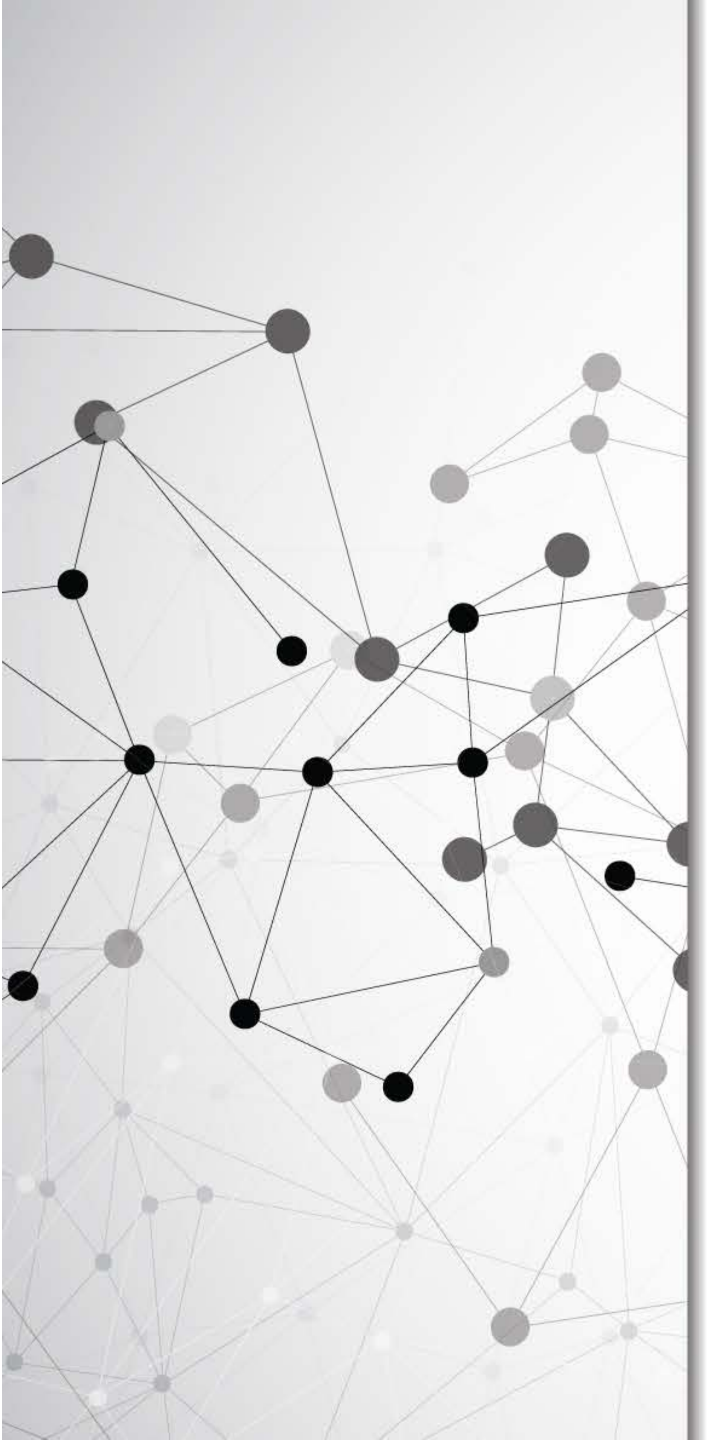
Face Recognition

EE4208

School of Electrical and Electronic Engineering

Dr. Wang Han
Office: S2-B2b-49
Email: hw@ntu.edu.sg
Phone: 6790-4506



Template Matching

Template Matching

A replica of an object of interest is compared to all unknown objects in the image field. If the match between the template and the unknown object is sufficiently close, the unknown object is labelled as the template object. The first one is known as **Sum of Squared Difference (SSD)**, or mean square difference.

$$\gamma(m,n) = -\sum_j \sum_k [f(j,k) - w(j-m, k-n)]^2$$

where f denotes the image field to be searched and w is the template.

If we replace the square with absolute value, this method is also known as **Sum of Absolute Difference (SAD)**. A template match is said to exist at coordinate (m,n) if

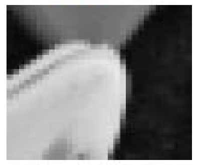
$$\gamma(m,n) = -\sum_j \sum_k |f(j,k) - w(j-m, k-n)| \quad < \text{pre−determined threshold}$$

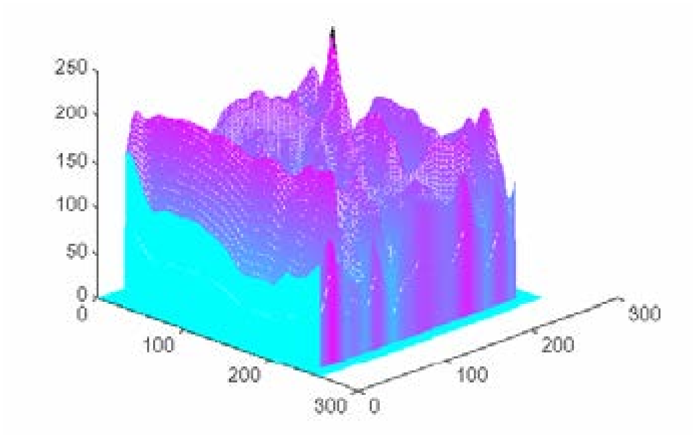
Template Matching (Example)

Scan the template over each pixel of the image and compute γ for each pixel.

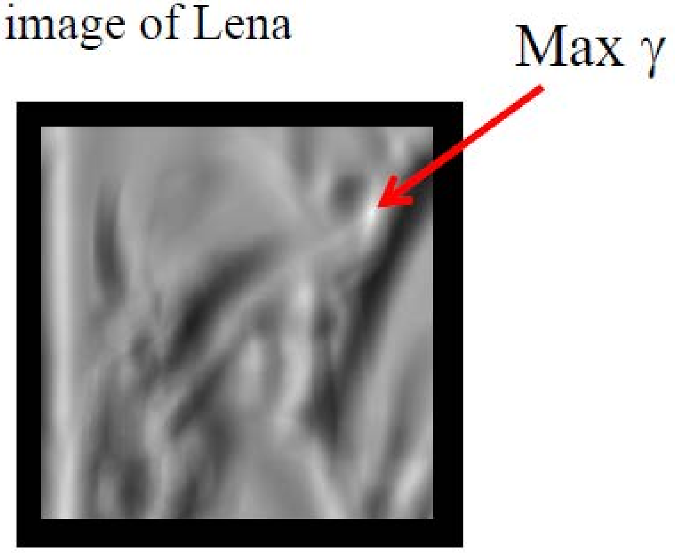
We obtain the 2D γ–map. If we represent higher values of γ as brighter intensity, we get the γ image. The max value of γ is deemed the best match.

Original image of Lena





γ–map



Result of template matching

Template Matching (Example)

1

Vectorise the template, $t =$

1	1	1
1	0	1
0	0	0

$$\vec{t} = (1,1,1,1,0,1,0,0,0)$$

Find the mean, $t_{mean} = \frac{5}{9}$

Adjust $\vec{t}$, so that it is zero-meaned.

$$\vec{t} = (1 - t_{mean}, \dots)$$

$$= \left(\frac{4}{9}, \frac{4}{9}, \frac{4}{9}, \frac{4}{9}, -\frac{5}{9}, \frac{4}{9}, -\frac{5}{9}, -\frac{5}{9}, -\frac{5}{9}\right)$$

Make it into unit vector;

$$\|\vec{t}\| = \sqrt{\left(\frac{4}{9}\right)^2 + \left(\frac{4}{9}\right)^2 + \cdots} = \frac{\sqrt{180}}{9}$$

$$\vec{T} = \frac{\vec{t}}{\|\vec{t}\|} = \frac{9}{\sqrt{180}}\left(\frac{4}{9}, \frac{4}{9}, \dots\right)$$

$$= \frac{1}{\sqrt{180}}(4, 4, 4, 4, -5, \dots)$$

Note: $\|\vec{T}\| = 1$

Copy out each pixel, form a 1×9 vector. If the template is 15×15, the vector is 1×225.

Template Matching (Example)

Repeat the same process to vectorise another 3x3 window. For example, a data below shows the same triangle with different intensity level.

Vectorise the image, $i =$

100	100	100
100	10	100
10	10	10

$$\therefore \vec{\imath} = (100, 100, 100, 100, 10, \dots)$$

Find the mean, $i_{mean} = 60$

Adjust $\vec{\imath}$, so that it is zero-meaned.

$$\vec{t} = (100 - 60, 100 - 60, \dots)$$
$$= (40, 40, 40, 40, -50, \dots)$$

Make it into unit vector;

$$\|\vec{\imath}\| = \sqrt{40^2 + 40^2 + \cdots} = \sqrt{18000}$$

$$\therefore \vec{I} = \frac{\vec{\imath}}{\|\vec{\imath}\|} = \frac{1}{\sqrt{18000}}(40,40, \dots)$$

$$= \frac{1}{\sqrt{180}}(4, 4, 4, 4, -5, \dots)$$

Template Matching (Example)

3

Inner product $\vec{T} \cdot \vec{I}$

Note: $\vec{T} = \vec{I}$

$\therefore \vec{T} \cdot \vec{I} = 1$

4

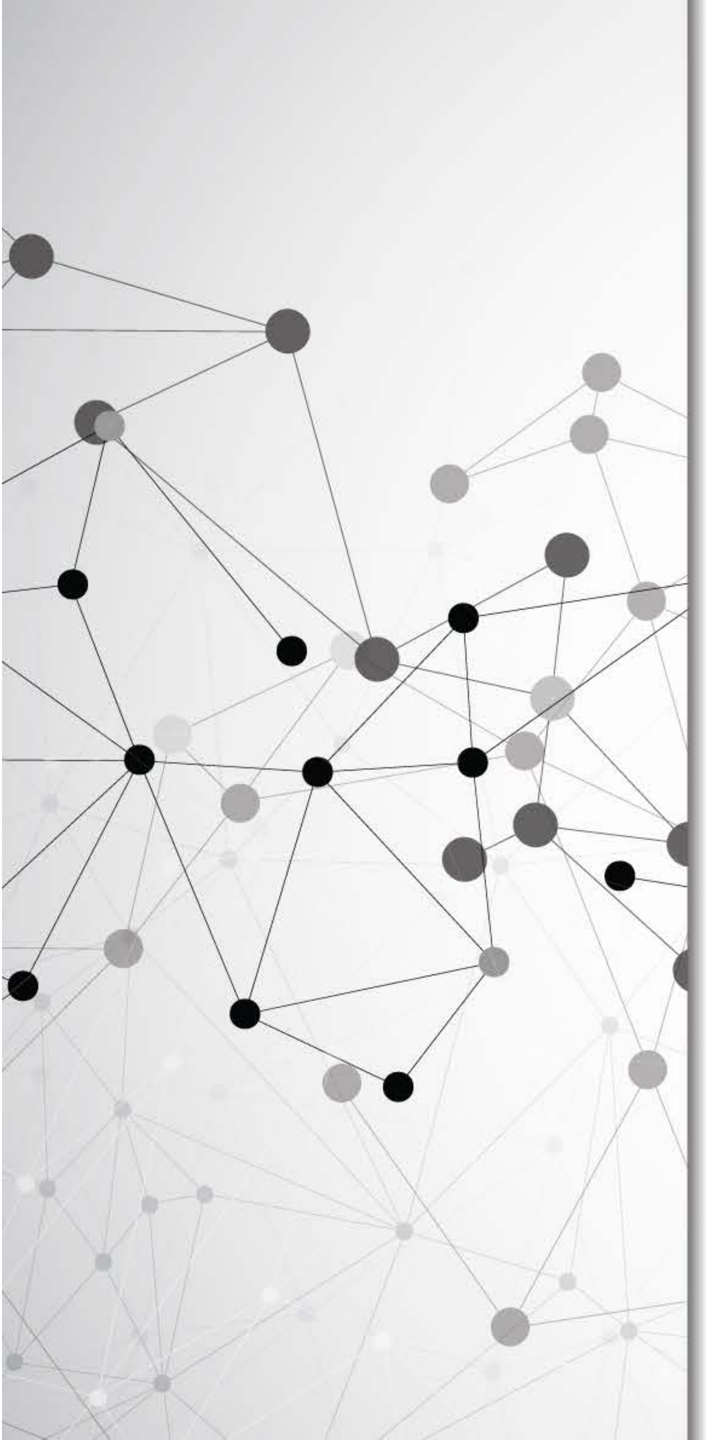
You may try other patterns of $i =$

1	2	3
3	2	1
2	3	4

and observe the new $\vec{T} \cdot \vec{I} < 1$

1 means the two patterns are identical. The worst case is -1 (when two vectors are in opposite direction).

Zero-centred Normalised Correlation Coefficient (ZNCC)

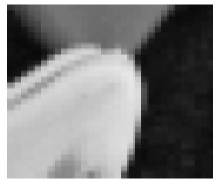


Template Matching

Suppose the template has 10×10 pixels, we slice the template in raster order.

Then form a list of 100 pixels. We treat this list as an 100 dimension vector, namely w.

We search through the image. At each pixel location, we cut out a sample of 10×10, then form another 100-vector, call it f.



This method is better than SAD.

Zero-centred Normalised Correlation Coefficient (ZNCC)

Objective

To measure the similarity between f and w.

Let

$$f = (f_1, f_2, \dots f_{100}) \quad \text{and} \quad w = (w_1, w_2, \dots w_{100})$$

We first find the mean of f and w, and adjust each vector by subtracting the mean f_{av} and w_{av} and then normalise the two vectors into unit vectors.

Mean adjustment is necessary, as this reduces side effect of lighting changes between template and image.

As we understand that if two unit vectors are identical, the inner product of the two will produce 1. Any “dissimilarity” will make the product less than 1. The worst case is -1.

Example

$$f = [6\ 7\ 2\ 5\ 6\ 9\ 7\ 8\ 9]^{\mathrm{T}} \quad \text{and} \quad w = [5\ 6\ 7\ 44\ 5\ 6\ 6\ 7\ 8]^{\mathrm{T}}$$

In general, let $h = f - f_{ave}$ and $g = w - w_{ave}$, then

$$\gamma = \frac{h^T g}{\|h\|\|g\|}.$$

Since $\frac{h}{\|h\|}$ is a unit vector and so is $\frac{g}{\|g\|}$,

therefore $\gamma = \cos\theta$ where θ is the angle between h and g.

Recall: $a^T b = \|a\|\ \|b\| \cos\theta$

This method is called Zero-centred Normalised Correlation Coefficient (ZNCC).

So ZNCC is simply the inner product between two unit vectors!

Now you know why γ is between -1 and 1.

Zero Centred Normalised Correlation Coefficient (ZNCC)

Properties of γ = ZNCC:

$-1 \leq \gamma(\mathrm{s},\mathrm{t}) \leq 1$

Summation is taken over the image region where w and f overlap.

The maximum value of $\gamma(\mathrm{s},\mathrm{t})$ appears at the position where $w(x,y)$ best matches $f(x,y)$.

✓ **Advantage:** Easy to implement.

✗ **Disadvantage:** Sensitive to image rotation and image scale changes, etc.

Summary

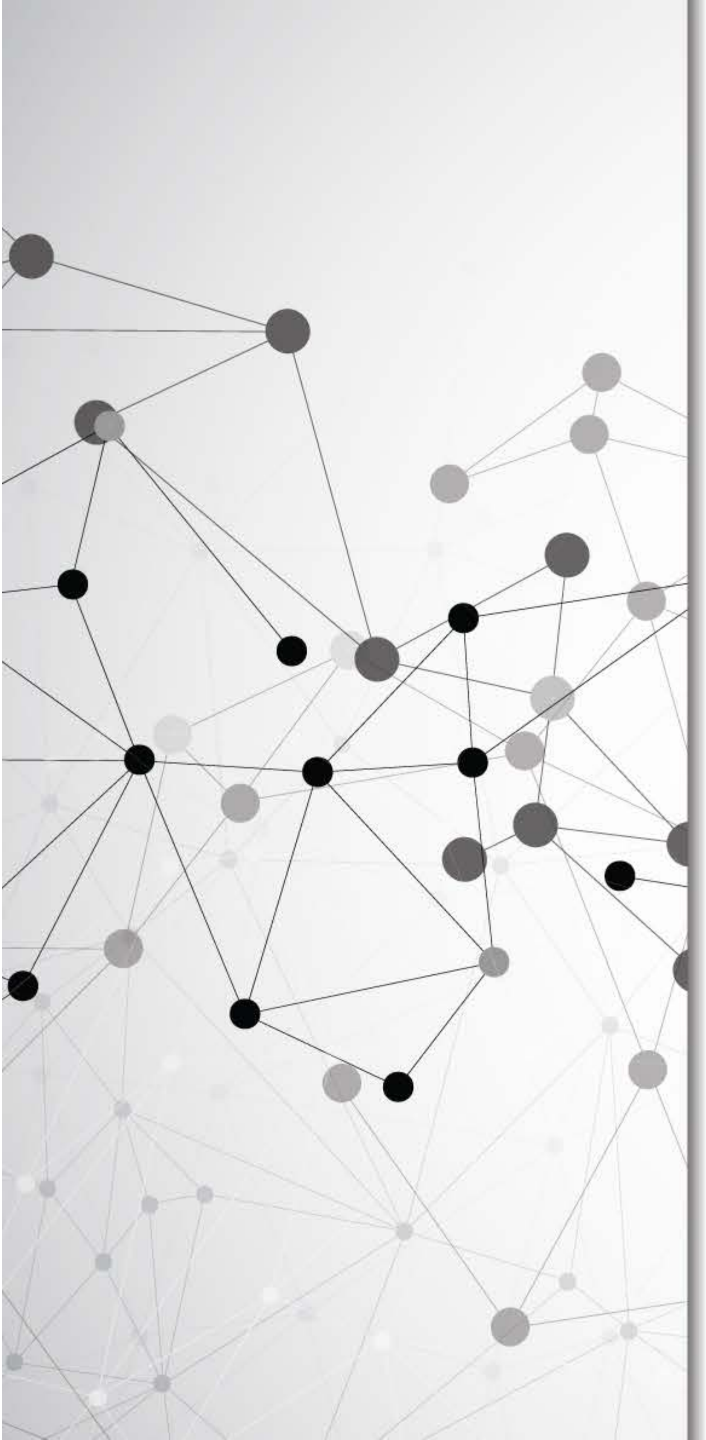
Simple and effective.

Adaptive to light changes.

Cannot cope with scale (size change).

Cannot deal with rotation.

Principal Component Analysis (PCA) Review



Principal Component Analysis (PCA) Review

A survey is conducted to collect two sets of information on the exam of EE4266.

H — represents the number of hours spent in study.
M — represents the exam marks.

The mean (average) is collected for each set.

We compute the mean by:

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

Principal Component Analysis (PCA) Review

	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**

Principal Component Analysis (PCA) Review

We compute the variance for each data set by:

$$var(x) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(X_i - \bar{X})}{n-1}$$

$$var(M) = \frac{\sum_{i=1}^{n}(M_i - \bar{M})(M_i - \bar{M})}{n-1}$$

$$var(H) = \frac{\sum_{i=1}^{n}(H_i - \bar{H})(H_i - \bar{H})}{n-1}$$

Note here we use $(n-1)$, instead of n.

Principal Component Analysis (PCA) Review

- Covariance is used to measure variance cross two data sets.
- If the two data sets are highly dependent to each other, its covariance is high; otherwise low.

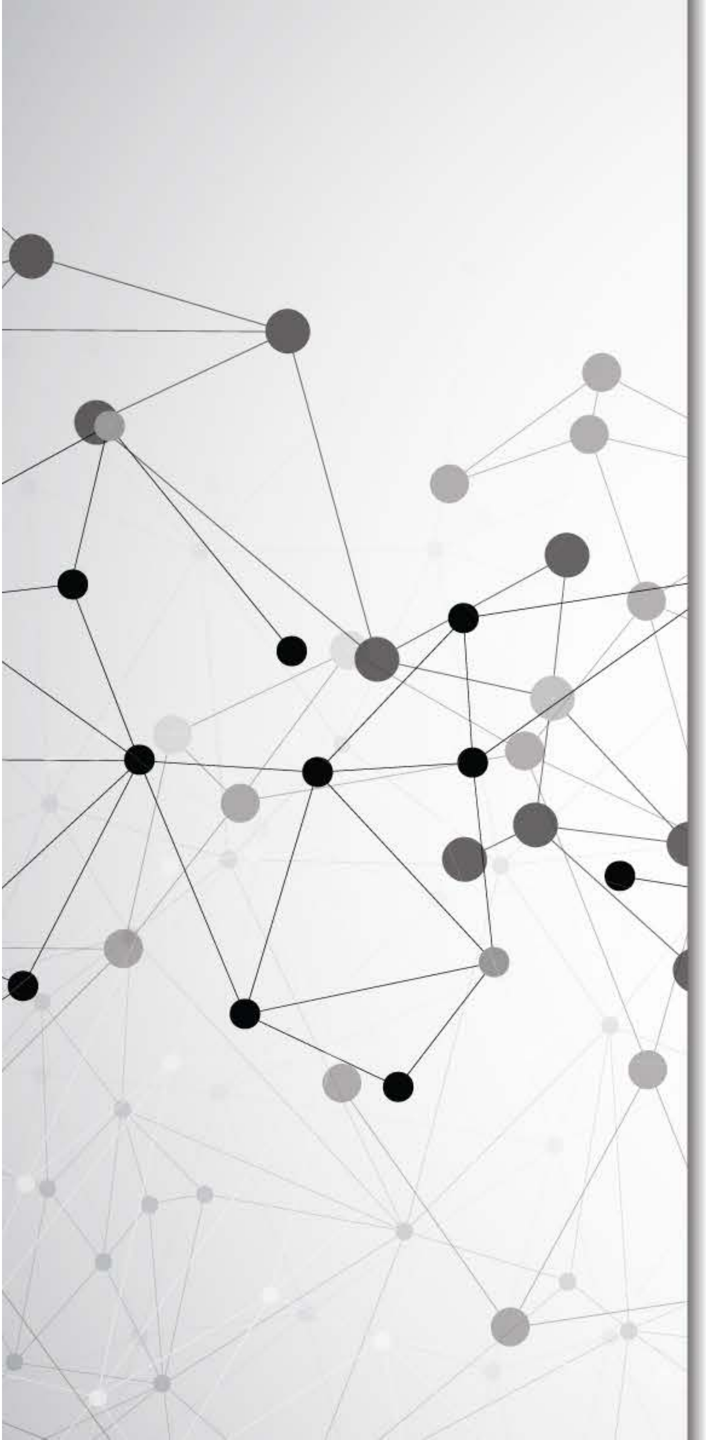
$$cov(X,Y) = \frac{\sum_{i=1}^{n}(X_i - \bar{X})(Y_i - \bar{Y})}{n-1}$$

Principal Component Analysis (PCA) Review

2D data set and covariance calculation:

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
Totals				**1149.89**
Average				**104.54**

Covariance Matrix



Principal Component Analysis (PCA) Review

For 2D data, H and M, we can form a 2×2 matrix:

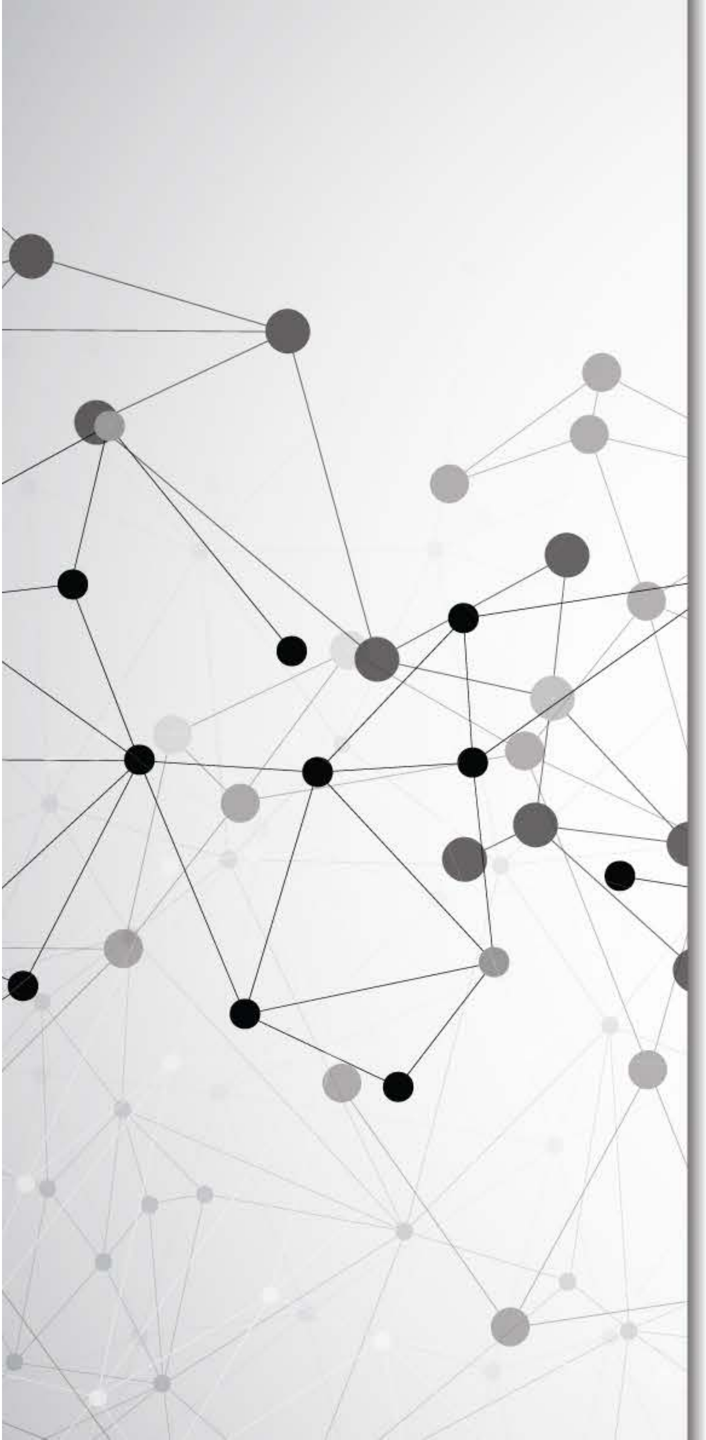
$$C = \begin{pmatrix} cov(H,H) & cov(H,M) \\ cov(M,H) & cov(M,M) \end{pmatrix}$$

For 3D data, X, Y, Z, we can form a 3×3 matrix:

$$C = \begin{pmatrix} cov(X,X) & cov(X,Y) & cov(X,Z) \\ cov(Y,X) & cov(Y,Y) & cov(Y,Z) \\ cov(Z,X) & cov(Z,Y) & cov(Z,Z) \end{pmatrix}$$

Note this is a symmetric square matrix, $cov(a,b) = cov(b,a)$.

Eigenvalues and Eigenvectors



Eigenvalues and Eigenvectors

Given a square matrix C, we can find its eigenvalues and eigenvectors, namely

Let $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, given $C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$

We find τ such that,

$$C \times X = \tau X$$

or

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \tau \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tau \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$= \begin{pmatrix} \tau & 0 \\ 0 & \tau \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

Move r.h. to l.h. and merge:

$$\begin{pmatrix} c_{11} - \tau & c_{12} \\ c_{21} & c_{22} - \tau \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 0$$

Eigenvalues and Eigenvectors

The non-trivial solution for $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ is: $\begin{vmatrix} c_{11} - \tau & c_{12} \\ c_{21} & c_{22} - \tau \end{vmatrix} = 0$

Hence, $(c_{11} - \tau)(c_{22} - \tau) - (c_{12})(c_{21}) = 0$

- Rendering τ_1 and τ_2. (Eigenvalues)
- Sub τ_1 into $C \times X = \tau X$, solve for X_1.
- Normalise $X_1 / ||X_1||$, so that $||X_1|| = 1$.
- Similarly, we obtain X_2 from τ_2.
- Note X_1 is orthogonal to X_2, because C is a symmetric matrix.
- Sub $-X_1$ into the equation $C \times X = \tau X$, it also stands. What does it mean?

Principal Component Analysis (PCA)

? Given 2D dataset below, find the mean for each dimension.

First we perform a mean adjust;

Data =

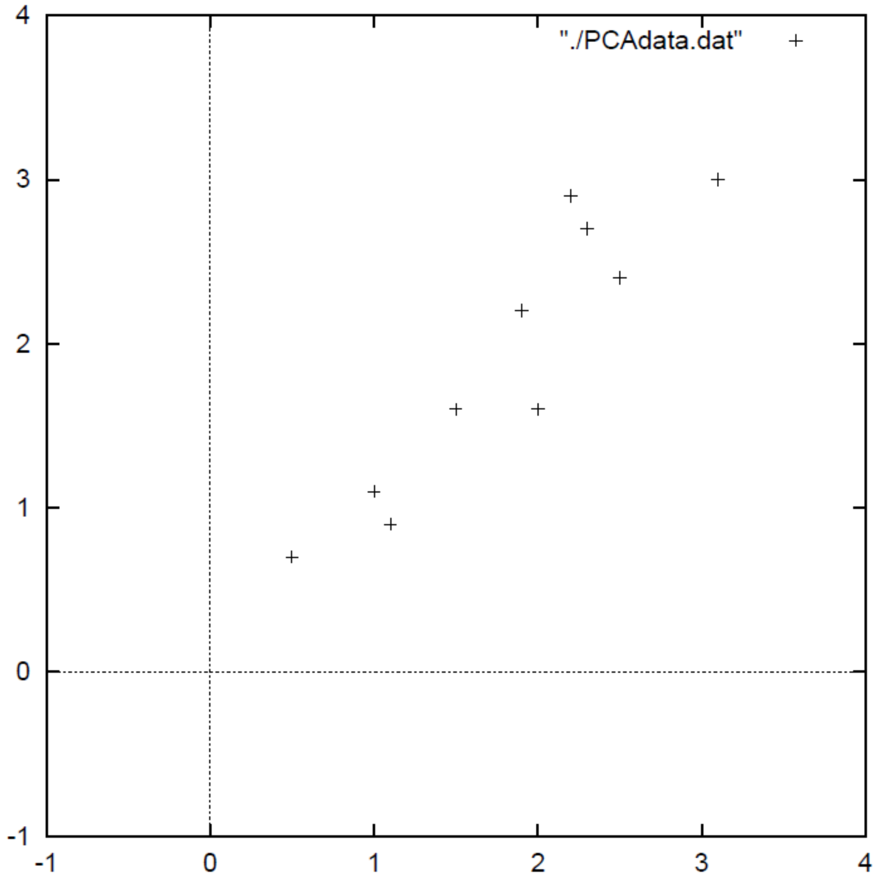
x	y
2.5	2.4
0.5	0.7
2.2	2.9
1.9	2.2
3.1	3.0
2.3	2.7
2	1.6
1	1.1
1.5	1.6
1.1	0.9

DataAdjust =

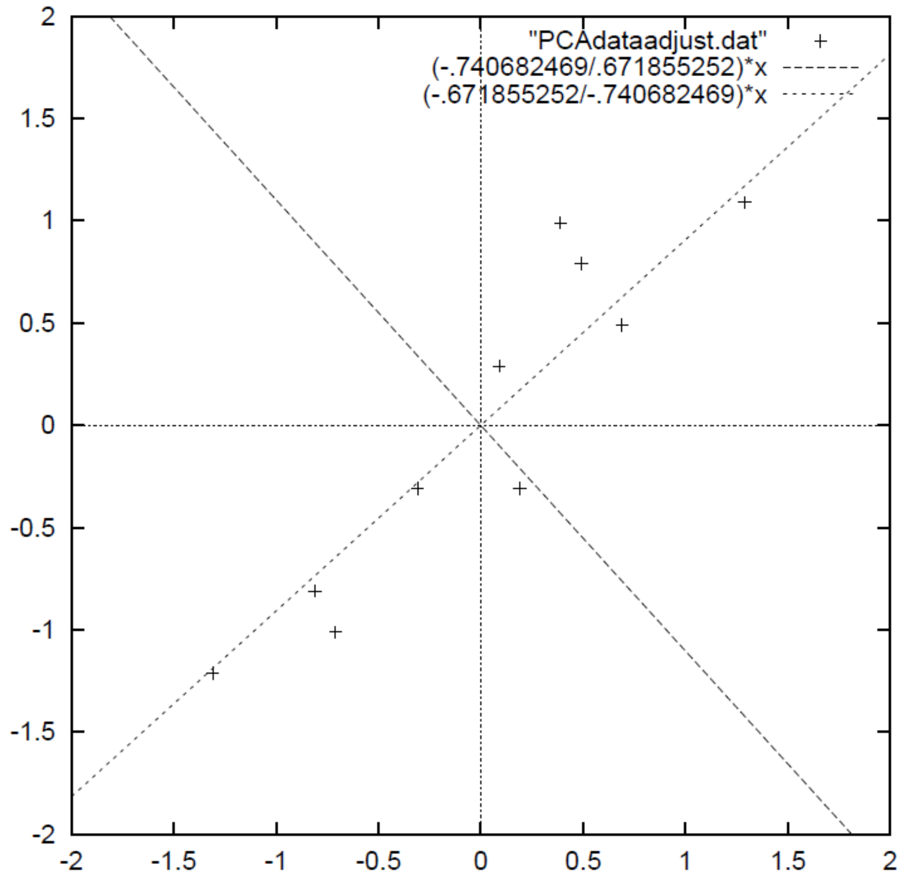
x	y
.69	.49
-1.31	-1.21
.39	.99
.09	.29
1.29	1.09
.49	.79
.19	-.31
-.81	-.81
-.31	-.31
-.71	-1.01

Principal Component Analysis (PCA)

Original PCA data



Mean adjusted data with eigenvectors overlayed



Principal Component Analysis (PCA)

Calculate the covariance matrix: $cov = \begin{pmatrix} .616555556 & .615444444 \\ .615444444 & .716555556 \end{pmatrix}$

Calculate the eigenvectors and eigenvalues of the covariance matrix:

$$\tau_1 = 0.04908, \ \ \tau_2 = 1.28403;$$

$$v_1 = \begin{pmatrix} -0.7352 \\ 0.6779 \end{pmatrix}; \ \ v_2 = \begin{pmatrix} -0.6779 \\ -0.7352 \end{pmatrix}$$

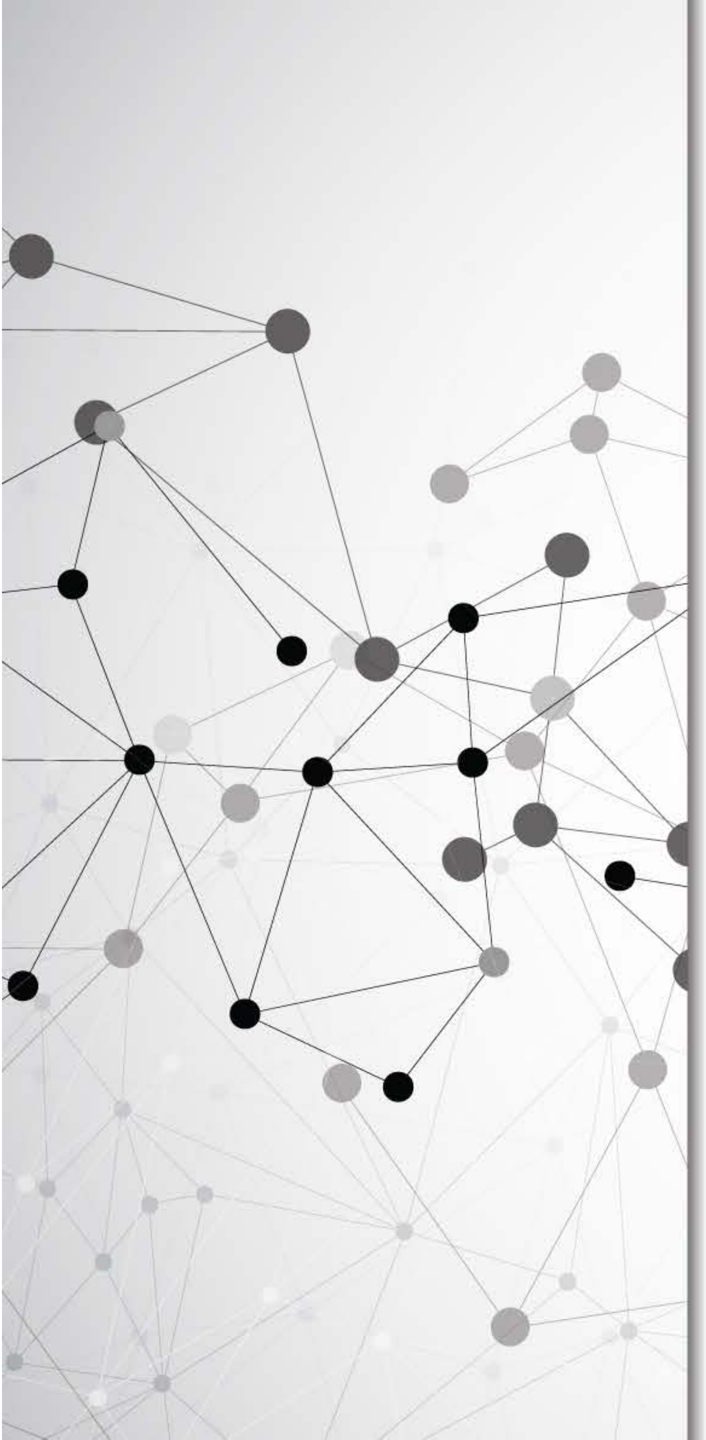
Note: Sub τ_1 into $cov \times X = \tau X$,

the solution of X gives v_1, where $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

And we must make $\|v_1\| = 1$ by normalization: $\left(\frac{v_1}{\|v_1\|}\right)$

Similarly, we obtain v_2.

Feature Vector



Feature Vector

Choosing components and forming a feature vector

$$FeatureVector = (v_1, v_2, \dots, v_n)$$

Where $v_1, v_2, \dots, v_n$ must follow the order of

$$\tau_1 > \tau_2 > \cdots \tau_n$$

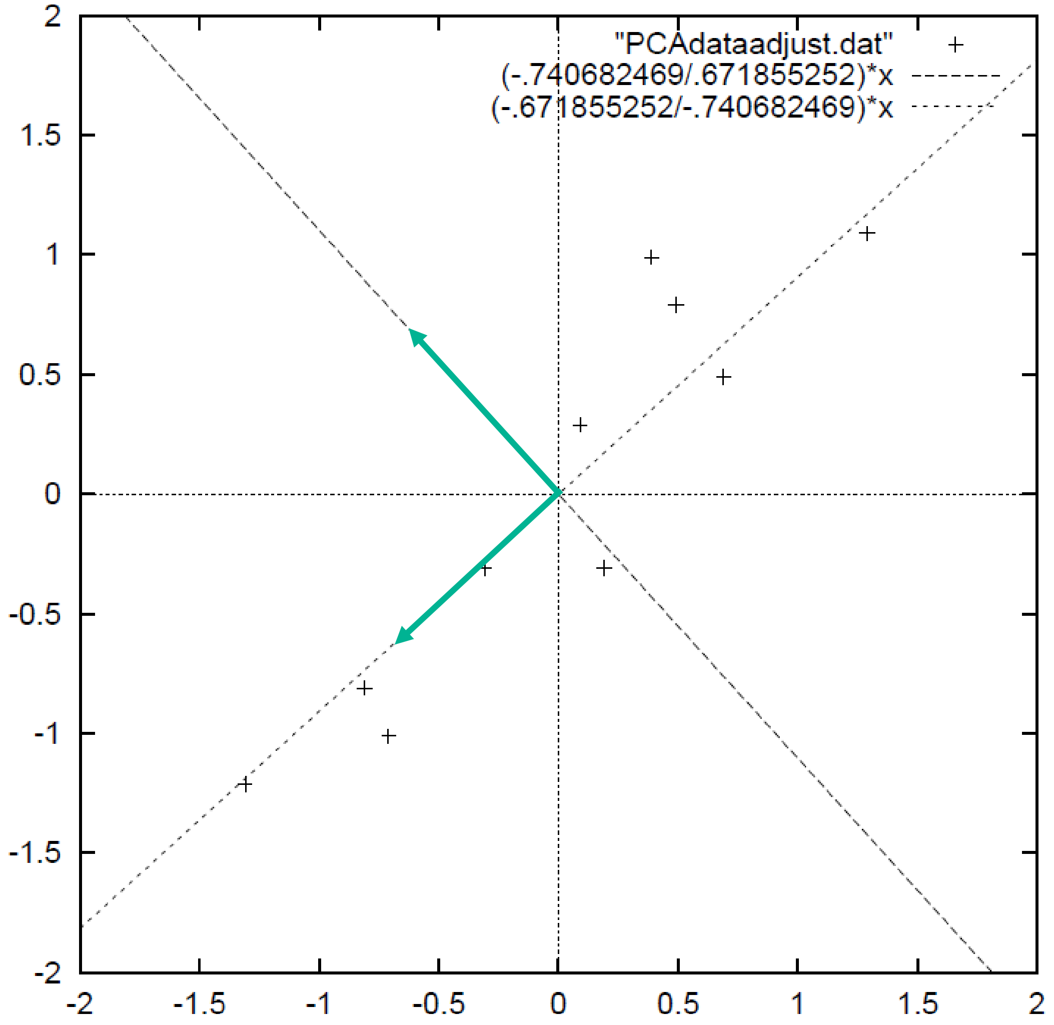
We call v_1 the *principal component* of the data set, since it has the highest eigenvalue.

In our example,

$$FeatureVector = (v_2, v_1)$$

Feature Vector: Example

Mean adjusted data with eigenvectors overlayed



Feature Vector

We want to express the RawAdjusted 2D data in the new coordinates (v_2, v_1).

You may think (v_2, v_1) as our new $x - y$ axes.

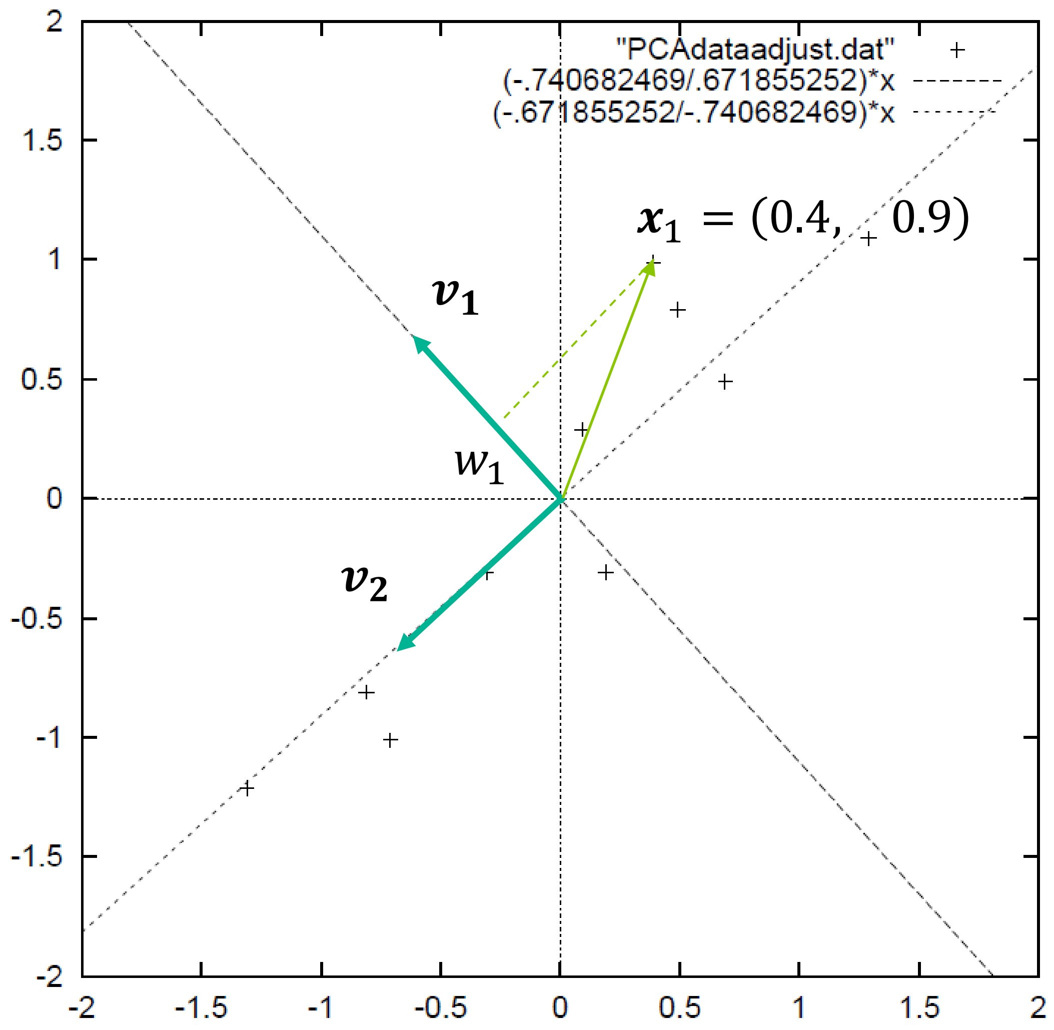
A transformation is required; it is a simple inner product.

For example: The new data $(w_2, w_1) = (-0.9328,, 0.3163)$ is in the new eigenspace. It is also 2D.

Feature Vector

$$w_1 = v_1^T \cdot x_1^T$$
$$= (-0.7352, 0.6779) \cdot \begin{pmatrix} 0.4 \\ 0.9 \end{pmatrix}$$
$$= -0.7352 \times 0.4 + 0.6779 \times 0.9$$
$$= 0.3163$$

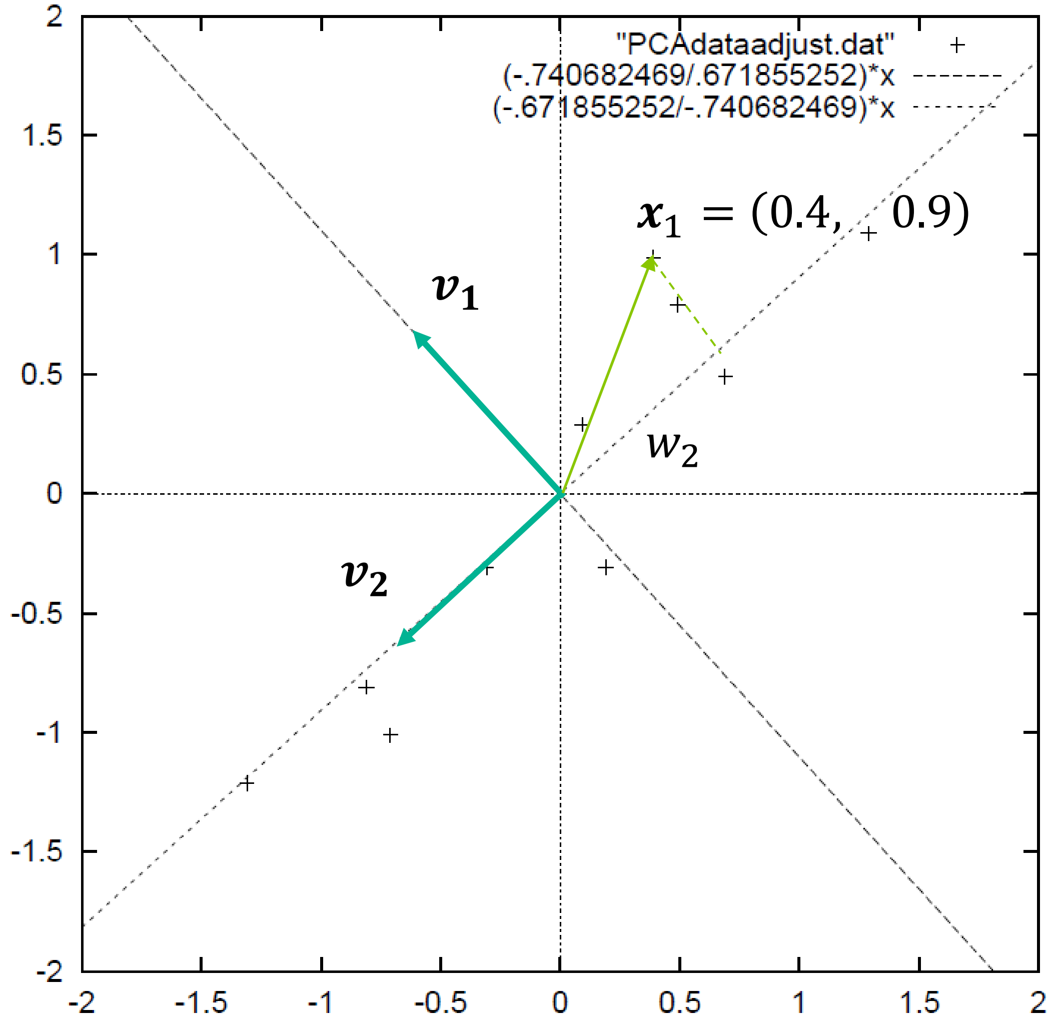
Mean adjusted data with eigenvectors over-layed



Feature Vector

$$
\begin{aligned}
w_2 &= v_2 \cdot x_1^T \\
&= \begin{pmatrix}-0.6779 & -0.7352\end{pmatrix} \cdot \begin{pmatrix}0.4 \\ 0.9\end{pmatrix} \\
&= -0.6779 \times 0.4 + (-0.7352) \times 0.9 \\
&= -0.9328
\end{aligned}
$$

Mean adjusted data with eigenvectors overlayed



Obtain New Data from Feature Vector

The new dataset is defined as:

$$FinalData = RowFeatureVector \times RowDataAdjust$$

Or

$$\begin{pmatrix} (w_2, w_1)_1 \\ (w_2, w_1)_2 \\ \vdots \\ (w_2, w_1)_n \end{pmatrix}^T = \begin{pmatrix} v_2^T \\ v_1^T \end{pmatrix} \times \begin{pmatrix} x_1, y_1 \\ x_2, y_2 \\ \vdots \\ x_n, y_n \end{pmatrix}^T$$

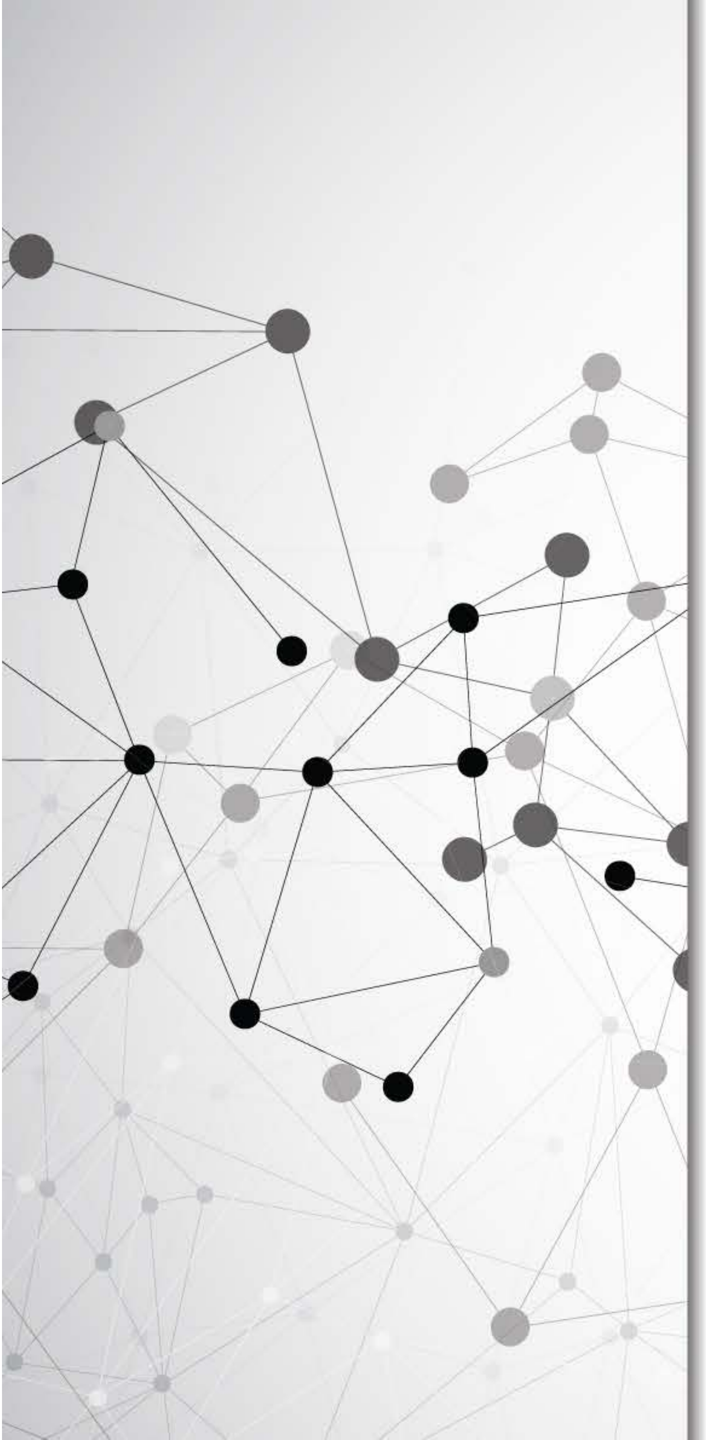
Obtain New Data from Feature Vector

$$= \begin{array}{c} \overbrace{}^{v_2^T} \\ \begin{pmatrix} -0.7351 & 0.6779 \\ 0.6779 & -0.7351 \end{pmatrix} \\ \underbrace{}_{v_1^T} \end{array} \times \begin{array}{cccc} \begin{pmatrix} x \\ y \end{pmatrix}_1 & \begin{pmatrix} x \\ y \end{pmatrix}_2 & \begin{pmatrix} x \\ y \end{pmatrix}_3 & \\ 0.69 & -1.31 & 0.39 & \cdots \\ 0.49 & -1.21 & 0.99 & \cdots \end{array}$$

Obtain New Data from Feature Vector

$Transformed\ Data =$

w_2	w_1
-0.827970186	-0.175115307
1.77758033	0.142857227
-0.992197494	0.384374989
-0.274210416	0.130417207
-1.67580142	-0.209498461
-0.912949103	0.175282444
0.0991094375	-0.349824698
1.14457216	0.0464172582
0.438046137	0.0177646297
1.22382056	-0.162675287



Dimension Reduction

Dimension Reduction

We throw away the data in v_1, reducing the data by 50%.

The loss is minimal, because all data are aligned in the principal direction of v_2.

Dimension Reduction

Hence the data is reduced to:

w_2
-0.827970186
1.77758033
-0.992197494
-0.274210416
-1.67580142
-0.912949103
0.0991094375
1.14457216
0.438046137
1.22382056

Dimension Reduction

Getting the old data back:

$$FinalData = RowFeatureVector \times RowDataAdjust$$

Or

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

$$RowData = RowFeatureVector^{-1} \times FinalData + Mean$$

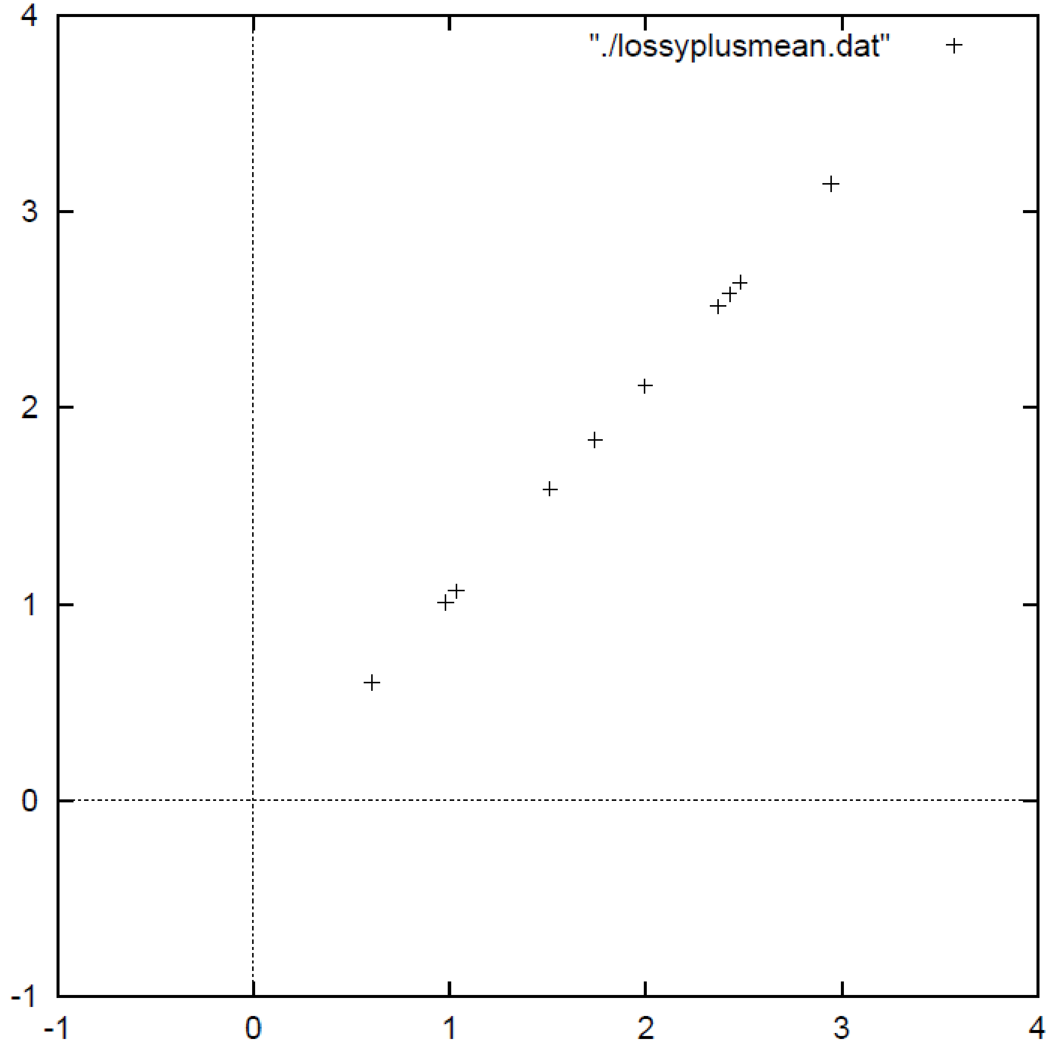
Or

$$RowData = RowFeatureVector^{T} \times FinalData + Mean$$

Because the matrix $RowFeatureVector$ is orthogonal.

Dimension Reduction

Original data restored using only a single eigenvector

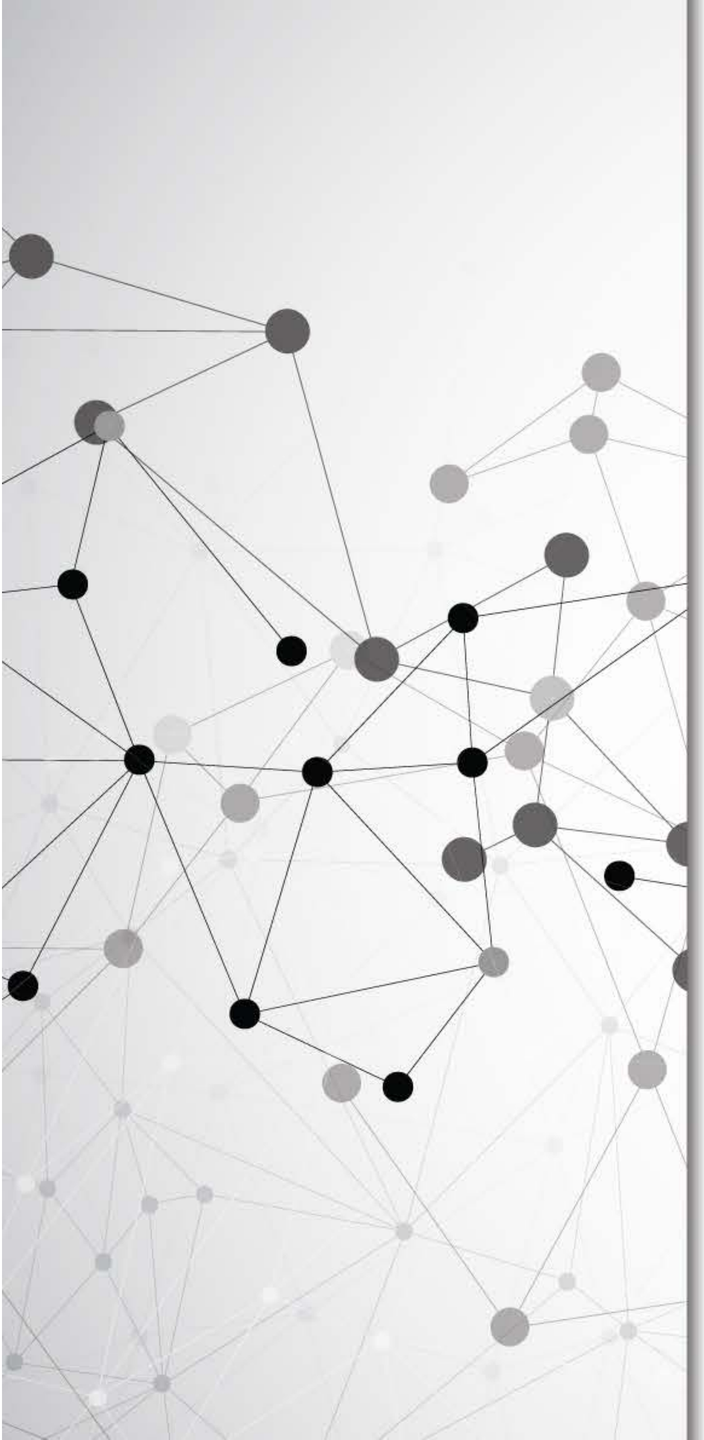


Face Recognition by the Eigenface Approach

Face Recognition by the Eigenface Approach

Based on the Principle Component Analysis (PCA), the main idea of Eigenface Method is to decompose face images into a small set of characteristic feature images called eigenfaces. The eigenfaces, which may also be referred to as the principal components of the original images, function as the orthogonal basis of a linear subspace called “eigenspace” or “face space”. Recognition is performed by projecting a new face image onto this subspace and comparing its position with those of the known ones.

This method is proposed by Turk and Pentland [circa 1991]



Training Stage

Training Stage

We are given a set of training 2D face images $\{F_1, F_2, \dots, F_n\}$. First, the training set images with the size of pixels are straightened out row by row into $M \times N$ dimensional column vectors. So the training set becomes $\{f_1, f_2, \dots, f_n\}$.

Then the average face is found: $f_{ave} = \frac{1}{n}\sum_{i=1}^{n} f_i$

The covariance matrix of the training set is defined as:

$$C = \frac{1}{n}\sum_{i=1}^{n}(f_i - f_{ave})(f_i - f_{ave})^T$$

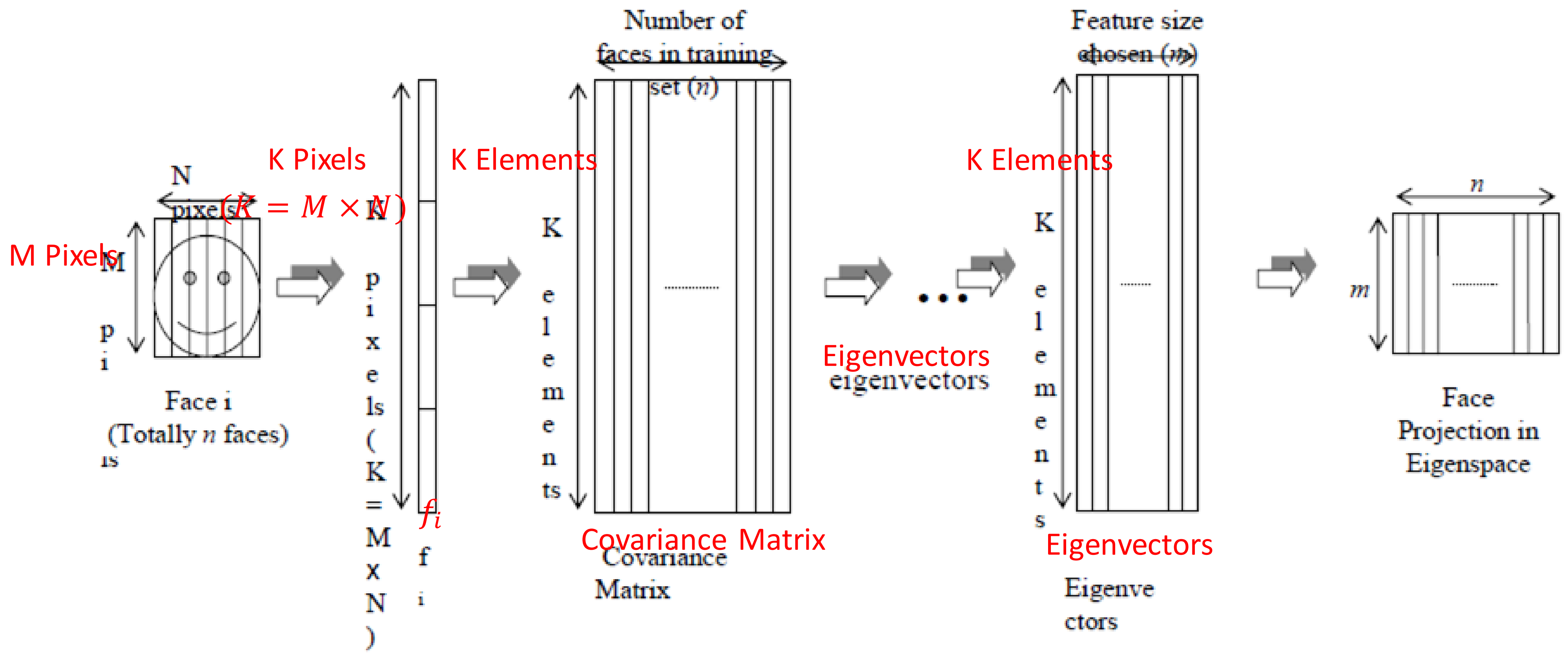
Should be very familiar by now.

Note that the average face is subtracted from every training image so that the resulting face data set has zero mean.

Training Stage

Another way to compute the covariance is by $C = \frac{1}{n} X X^T$

where $X = [f_1 - f_{ave}, f_2 - f_{ave}, \ldots, f_n - f_{ave}]$



Training Stage

$$f_1 - f_{ave} = w_{i1}v_1 + w_{i2}v_2 + \cdots + w_{iK}v_K$$

$v_1, \ldots, v_K$ are the eigenvectors (eigenfaces or principal components) of the corresponding K largest eigenvalues of C.

Recap:
We started off with the training images $\{F_1, F_2, \ldots, F_n\}$, vectorise them into $\{f_1, f_2, \ldots, f_n\}$ and then use them to form the covariance matrix C.

Next is the dimension reduction step.

The m eigenfaces of the m largest eigenvalues of C are kept. Then each face vector (subtracting the average face) is projected onto each of these m eigenfaces to produce m coefficients for each face vector in the training set.

$$f_1 - f_{ave} \simeq w_{i1}v_1 + w_{i2}v_2 + \cdots + w_{im}v_m \qquad m \ll K$$

Training Stage

For each $f_1 - f_{ave}$, take its inner product with each v_j:

$$
\begin{gathered}
(f_i - f_{ave})^T v_1 = w_{i1} \\
(f_i - f_{ave})^T v_2 = w_{i2} \\
\vdots \\
(f_i - f_{ave})^T v_m = w_{im}
\end{gathered}
$$

Then for each i^{th} face, set $w_i = [w_{i1}, w_{i2}, \dots, w_{im}]^T$.

So now the training images are represented by $\{w_1, \dots, w_n\}$ where each w_i is a m dimension vector. This achieves the dimension reduction from $M \times N$ dimension to m dimension feature space. Usually $M \times N \gg m$.

$\{w_1, \dots, w_n\}$ are used for classification instead of $\{f_1, \dots, f_n\}$.

Recognition Stage

How can $\{w_1, \dots, w_n\}$ be used?

We could use them as the input training data for any classifier that we have encountered. E.g. SVM, Bayes Classifier, kNNR, etc.

Example: Use the Nearest Neighbour Classifier

Given an unknown face F_U, vectorise it in the same way as for the training samples to f_U.

Then
$$(f_U - f_{ave})^T v_1 = w_{U1}$$
$$\vdots$$
$$(f_U - f_{ave})^T v_m = w_{Um}$$

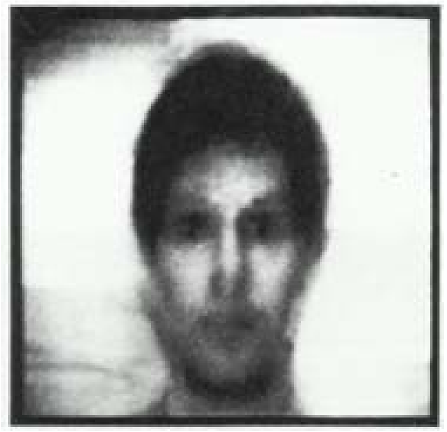
yielding $w_U = [w_1, w_2, \dots, w_n]^T$

Finally, compute its distance from each of $w_1, w_2, \dots, w_n$. If w_A is closest to w_U, then F_U is identified as the person in the face image.

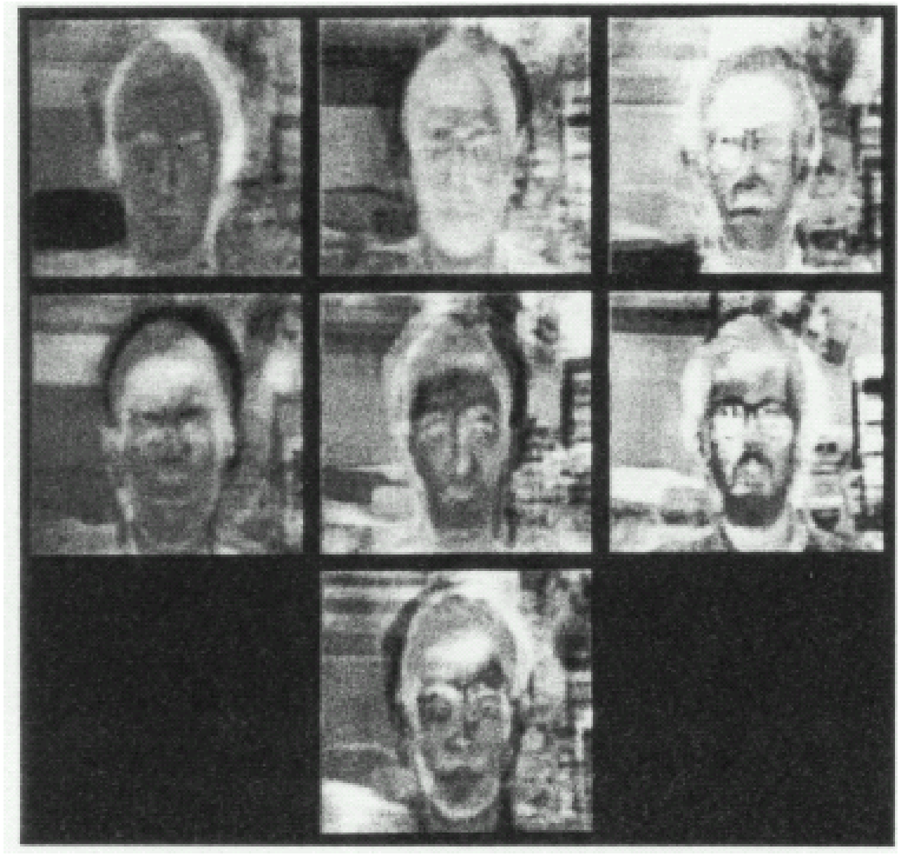
Examples

Eigenfaces Examples

Mean: f_{ave}



Top eigenvectors: $v_1, \ldots, v_k$



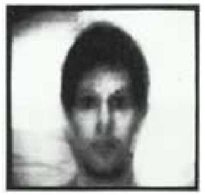
Eigenfaces Examples

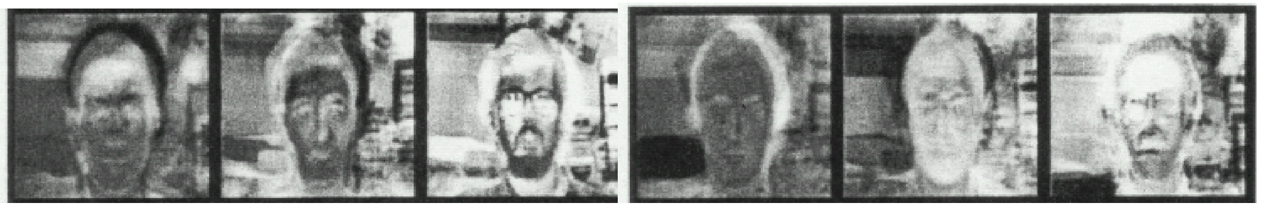
Face x in “face space” coordinates:

$$X \longrightarrow (f_1 - f_{ave})v_1, \dots, (f_m - f_{ave})v_m$$

$$= w_1, w_2, \dots, w_\mathrm{m}$$

Reconstruction:

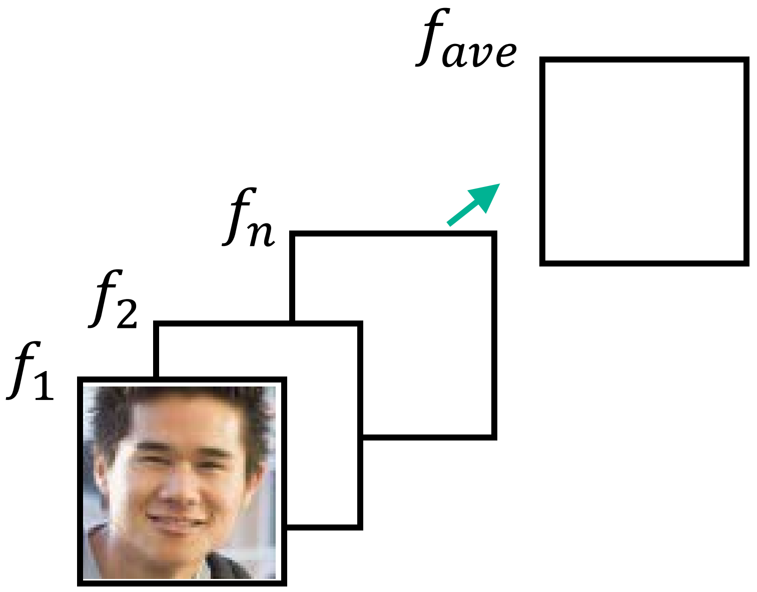




$$\hat{x} \quad = \quad f_{ave} \quad + \quad w_1 v_1 + w_2 v_2 + w_3 v_3 + \cdots$$

Eigenfaces why?

Why?



$$f_1 = (f_1^b, f_2^b, f_3^{hair}, f_4^{hair}, \dots, f_{201}^b, f_{202}^{eye}, f_{203}^{eye}, f_{204}^b, \dots, f_{1201}^b, f_{1202}^{mouth}, f_{1203}^{mouth}, f_{1204}^b, \dots, f_{9909}^{nose}, \dots, f_{10,000}^b)$$

$f_{eye}, f_{nose}, f_{mouth}, f_{chin}, \cdots$

These pixels have large variances/covariance, and contribute to the first few hundred eigenvectors. The rest of the pixels on the face are flat, and have less or negative impact in face comparison/recognition.

PCA Transformation Review

PCA Transformation Review

Let $\vec{x} = \begin{pmatrix}1\\0\end{pmatrix}, \vec{y} = \begin{pmatrix}0\\1\end{pmatrix}$ represent x and y axes.

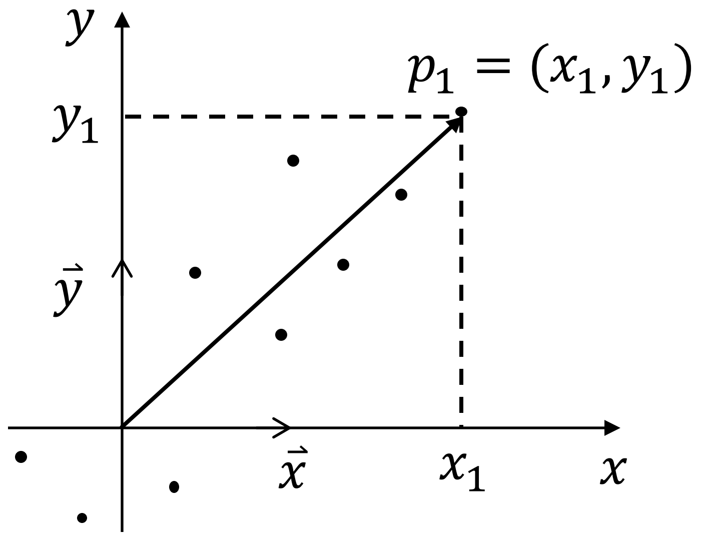
A data point $p_1 = (x_1, y_1)$ can be expressed as a vector:

$$\overrightarrow{p_1} = x_1\vec{y} + y_1\vec{y} = x_1\begin{pmatrix}1\\0\end{pmatrix} + y_1\begin{pmatrix}0\\1\end{pmatrix}$$
$$= \begin{pmatrix}x_1\\y_1\end{pmatrix}$$

or

$$x_1 = \overrightarrow{p_1}\cdot\vec{x} = \begin{pmatrix}x_1\\y_1\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} \longleftarrow$$ (p_1's projection in x axis)

$$y_1 = \begin{pmatrix}x_1\\y_1\end{pmatrix}\cdot\vec{y} = \begin{pmatrix}x_1\\y_1\end{pmatrix}\cdot\begin{pmatrix}0\\1\end{pmatrix} = x_1\cdot 0 + y_1\cdot 1 = y_1$$



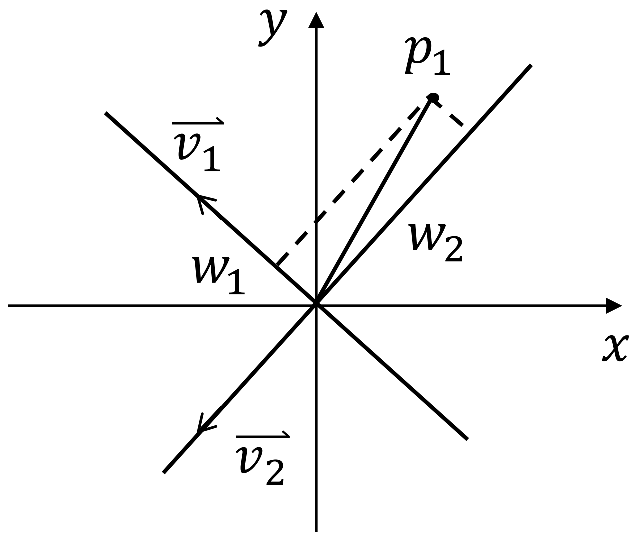
PCA Transformation

Similarly, this point may have projection in $\overrightarrow{v_1}$:

$$w_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \cdot \overrightarrow{v_1}$$

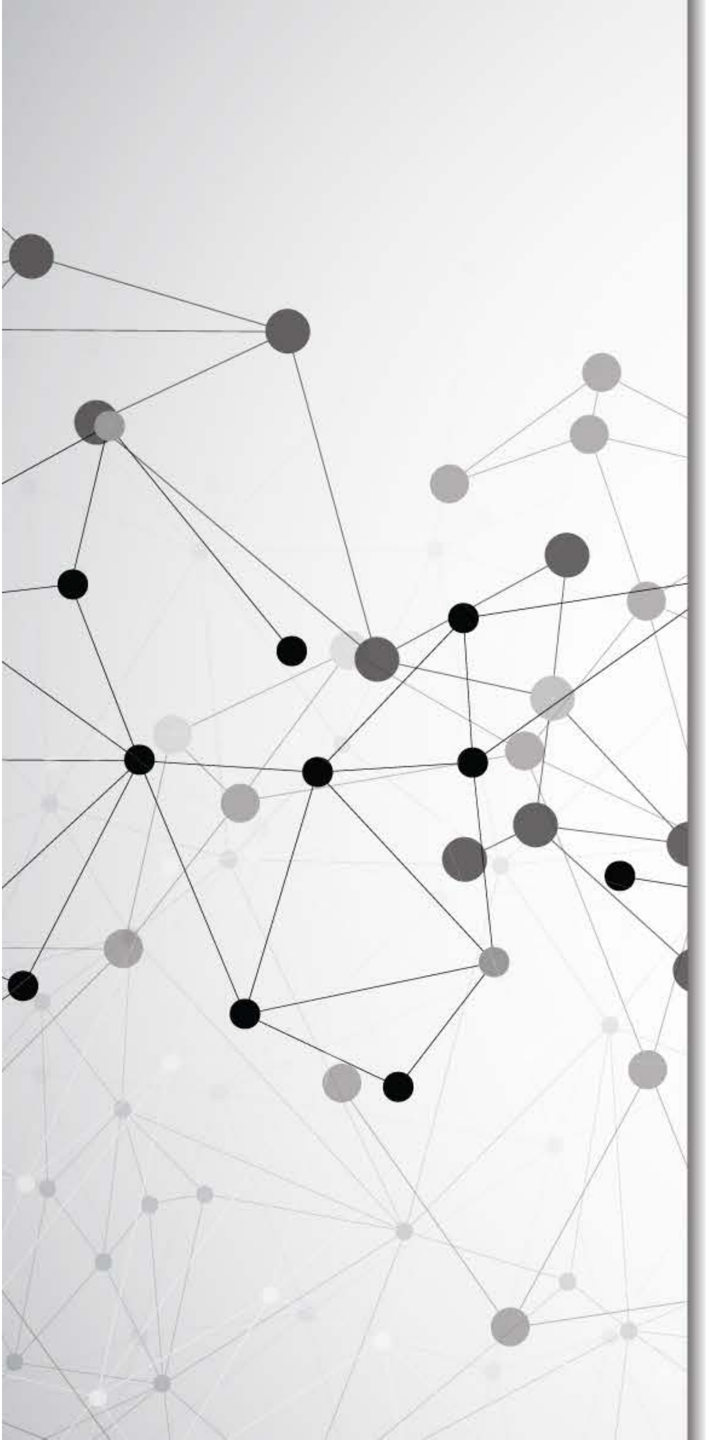
and in $\overrightarrow{v_2}$:

$$w_2 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \cdot \overrightarrow{v_2}$$



We call $\overrightarrow{v_1}$ and $\overrightarrow{v_2}$ form eigenspace (or feature space), and p_1 has coordinates (w_1, w_2), or

$$\overrightarrow{p_1} = w_1\overrightarrow{v_1} + w_2\overrightarrow{v_2} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$



Summary

Summary

Key points discussed in this topic:

- One pixel in a face image represents one dimension.
- In one face image, only 10% (or less) pixels contain vital information about the face details; they have large variances; 90% are either background or flat region, e.g. forehead; they have relatively low variances.
- We select the top 10% coefficients after PCA (e.g. $m = 1000$, out of 10,000) for comparison or recognition. Top means in order of variances or eigen values; this is called dimension reduction.
- Only the first 1000 dimension is stored and participates in the recognition.
  E.g. 60 people in the EE4266 class, $n = 60$; one face is represented as 1 dot in the hyperspace of 10,000 dimension. The training process will result in a database of 60 faces.

Summary

Key points discussed in this topic:

- Recognition is carried out in comparing the test image (one “dot” in hyper space) with pre-stored (trained) face data (dots). The nearest dot is claimed as the result; this process is called classification (NCC).

- Variation in facial expression is allowed, e.g. smiling, talking, angry, etc. One person may keep more than one face in the database.

- When a fresh (untrained) face is input, the system will fail.

Nearest Centre Classifier (NCC)

Nearest Centre Classifier

We begin with the simplest type, **nearest centre classifier (NCC),** also known as *minimum distance classifier*.

For each object class ω_k, we find the centroid m_k of the set of samples $\{x^{(k)}{}_1, x^{(k)}{}_2, \dots, x^{(k)}{}_N\}$ belonging to class ω_k.

For any object feature vector, x, find the distances of x to each of the class centroids (or prototypes).

Then we classify an unknown object, x, as belonging to the class whose centroid (m_k) is closest (i.e. having minimum distance) to the unknown object's feature vector.

Notation: Bold-face small letters denote column vectors.

Nearest Centre Classifier Example

Example:

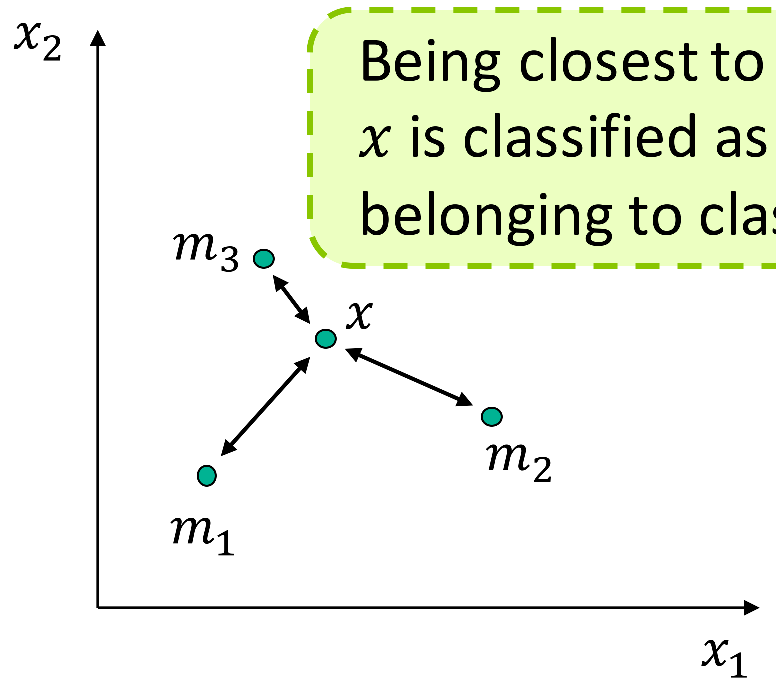
$$\omega_1 : m_1 = \left(x_1{}^{(1)}, x_2{}^{(1)}\right)^T$$
$$\omega_2 : m_2 = \left(x_1{}^{(2)}, x_2{}^{(2)}\right)^T$$
$$\omega_3 : m_3 = \left(x_1{}^{(3)}, x_2{}^{(3)}\right)^T$$

From the figure, x is closest to m_3

since $\|x - m_3\|^2 < \|x - m_1\|^2$

and $\|x - m_3\|^2 < \|x - m_2\|^2$.

Therefore, classify x as belonging to ω_3.



Being closest to m_3, x is classified as belonging to class ω_3.

2D feature space with 3 classes

$\|\cdot\|^2$ is the L_2 norm or the Euclidean norm of a vector.

E.g. If $v = [v_1 \quad v_2 \quad v_3]^T$, then $\|v\|^2 = \sqrt{\left(v_1{}^{(2)} + v_2{}^{(2)} + v_3{}^{(2)}\right)}$

Nearest Centre Classifier Example

Each pattern class is represented by a prototype:

$$m_k = \frac{1}{N_k}\sum_{i=1}^{N_k} X_i^{(k)} \qquad k = 1, 2, \ldots, c \quad (c \text{ classes})$$

where m_k is the mean vector for class w_k,
N_k is the number of pattern vectors from class w_k,
c is the total number of distinct classes.

? Given an unknown pattern vector x, find the closest prototype by finding the minimum Euclidean Distance.

$$D_i(x) = \|x - m_i\|$$

or

$$D_i(x)^2 = \|x - m_i\|^2$$

Note: From here on, we will denote $\|\cdot\|$ is taken to be the L_2 norm or the euclidean norm.

Nearest Centre Classifier Example

where $x \to \omega_i$ if $D_i(x) < D_j(x)$ for $i, j = 1, 2, \dots, c;\ j \neq i$

or $x \to \omega_i$ if $D_i^2(x) < D_j^2(x)$ for $i, j = 1, 2, \dots, c;\ j \neq i$

? How can we simplify the decision function of NCC ?

$$D_j(x)^2 = \left\|x - m_j\right\|^2 = (x - m_j)^T(x - m_j) \quad \text{Since } \|v\|^2 = v^Tv$$

$$= x^Tx - 2x^Tm_j + (m_j)^Tm_j \quad \text{Using } a^Tb = b^Ta$$

Observe that:

- x^Tx is common to all $D_j(x)^2$ and can be discarded without affecting the value of j to maximise $D_j(x)^2$.
- Also note that minimum of a function is equivalent to the maximum of the negative of that function.

Nearest Centre Classifier Example

Thus to save computation time, finding the **minimum** with regards to j of $D_j(x)^2$ is equivalent to finding the **maximum** of $d_j(x)$ where

$$d_j(x) = 2x^T m_j - (m_j)^T m_j$$

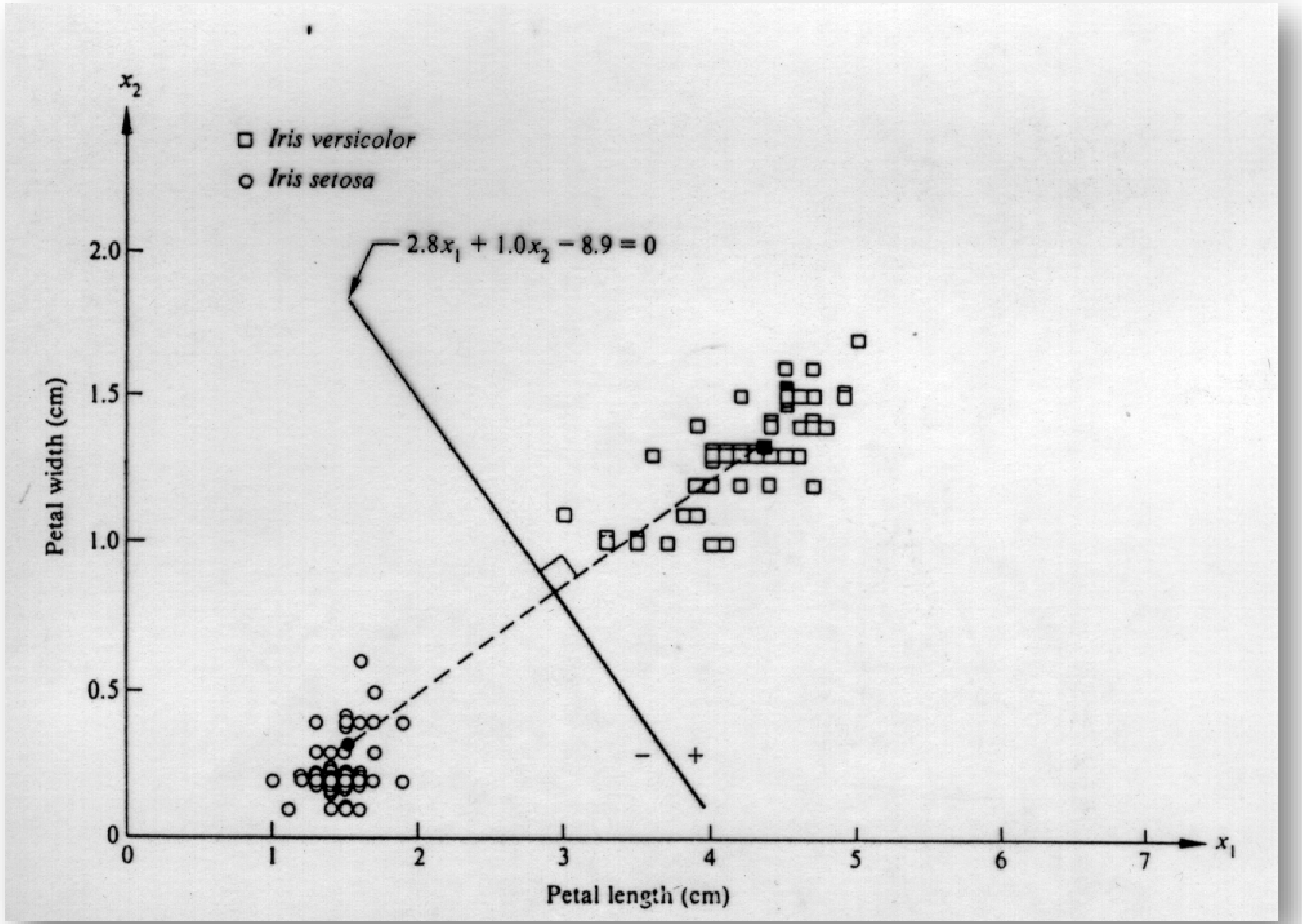
Hence the Nearest Center Classifier computes all $d_j(x)$ for $j = 1, 2, \dots, c$ (c classes). And for i such that

$$d_i(x) > d_j(x) \quad \text{for all } j = 1, 2, \dots, c \text{ and } j \neq i$$

NCC will assign the sample with feature $x \Rightarrow \omega_i$.

Nearest Centre Classifier Example

Example: Classify Iris versicolour and Iris setosa by petal length, x_1, and petal width, x_2.



Nearest Centre Classifier Example

Assume that the centroid for each class is found to be

versicolour $\omega_1 : m_1 = (4.3, 1.3)^T$

setosa $\omega_2 : m_2 = (1.5, 0.3)^T$

Let $x = (x_1, x_2)^T$ be the feature for an unknown sample.

Then

$$d_1(x) = x^T m_1 - \frac{1}{2}(m_1)^T m_1 = 4.3x_1 + 1.3x_2 - 10.1$$

$$d_2(x) = x^T m_2 - \frac{1}{2}(m_2)^T m_2 = 1.5x_1 + 0.3x_2 - 1.17$$

In matrix form

$$\begin{pmatrix} 4.3 & 1.3 & -10.1 \\ 1.5 & 0.3 & -1.17 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$$

Nearest Centre Classifier Example

?

If the petal of an unknown iris has length 2 cm and width 0.5 cm, i.e. $x = (2, 0.5)^T$, which iris is it from?

$$\begin{pmatrix} 4.3 & 1.3 & -10.1 \\ 1.5 & 0.3 & -1.17 \end{pmatrix} \begin{pmatrix} 2 \\ 0.5 \\ 1 \end{pmatrix} = \begin{pmatrix} -0.85 \\ 1.98 \end{pmatrix}$$

Therefore, it is Iris setosa.

?

If the petal of an unknown iris has length 5 cm and width 1.5 cm, i.e. $x = (5, 1.5)^T$, which iris is it from?

$$\begin{pmatrix} 4.3 & 1.3 & -10.1 \\ 1.5 & 0.3 & -1.17 \end{pmatrix} \begin{pmatrix} 5 \\ 1.5 \\ 1 \end{pmatrix} = \begin{pmatrix} 13.35 \\ 6.78 \end{pmatrix}$$

Therefore, it is Iris versicolour.

Linear Decision Boundary

Two classes are said to be ***linearly separable*** if one can find a line (hyperplane) that separates them.

Note that the decision boundary of the nearest center classifier is the ***perpendicular bisector*** of the line joining two class prototypes.

At the decision boundary (surface), $d_1(\mathbf{x}) = d_2(\mathbf{x})$

By defining $d_{12}(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x})$,
then $d_{12}(\mathbf{x}) = 0$ yields the decision boundary for the minimum distance classifier.

Linear Decision Boundary

Hence for the above problem :

At the decision boundary (surface), $d_1(\mathbf{x}) = d_2(\mathbf{x})$

$$\therefore \quad d_{12}(\mathbf{x}) = d_1(\mathbf{x}) - d_2(\mathbf{x}) = 2.8\,x_1 + 1.0\,x_2 - 8.9 = 0$$

(See graph on page 33)

for Iris classification,

$$\mathbf{x} \rightarrow \omega_1 \quad \text{if} \quad d_{12}(\mathbf{x}) > 0 \qquad \text{(why ?)}$$
$$\mathbf{x} \rightarrow \omega_2 \quad \text{if} \quad d_{12}(\mathbf{x}) < 0 \qquad \text{(why ?)}$$

Examples :

$\mathbf{x} = (2, 0.5)^T$; $d_{12}((2, 0.5)^T) = -2.8 \rightarrow \omega_2$: Iris setosa

$\mathbf{x} = (5, 1.5)^T$; $d_{12}((5, 1.5)^T) = 6.6 \rightarrow \omega_1$: Iris versicolor

Nearest Centre Classifier: Advantages

- The main advantage of NCC is that we only need to store the class centres only.
- The computation time for classification is also very fast* since we only need to calculate the distance of an unknown vector from each of the class centres and find the nearest one.

NCC worked well in the previous example. Why?
In fact, MDC (minimum distance classifier)
performs quite poorly in real applications.
Under what condition will NCC work well or not work well?

* For training samples with N data to be used for designing a classifier to distinguish between c classes, the distance of each unknown vector needs to be checked against each of the centres of the c classes. i.e. only c distance computation and not N. Usually, N >> c. In certain applications, N could be in the thousands while c may be 10. We need N distance calculations for Nearest Neighbour Classifier (to be studied later).