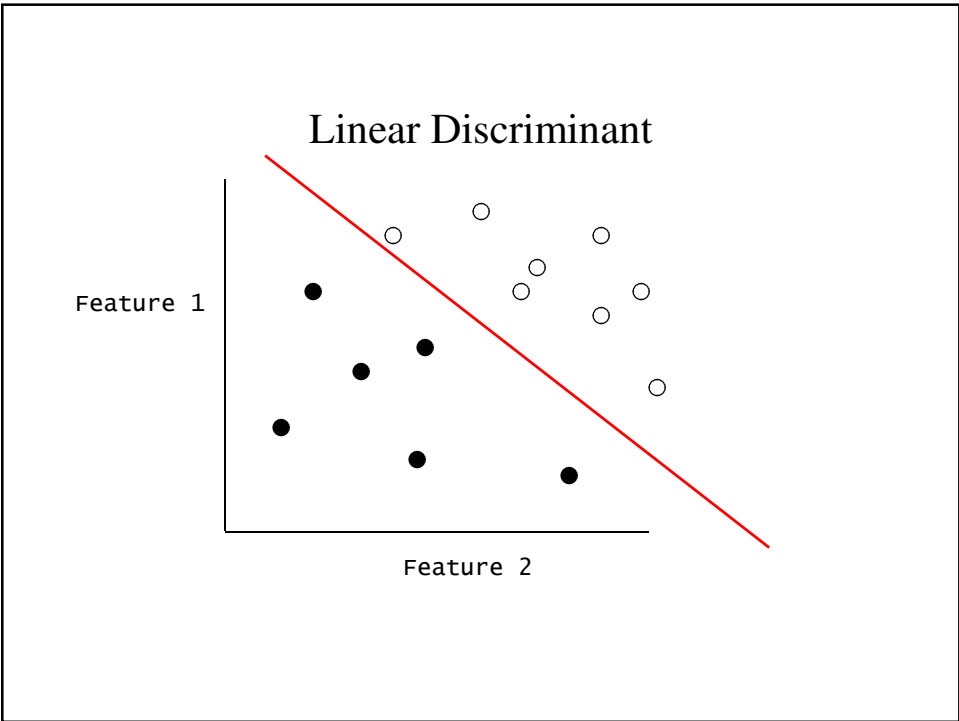
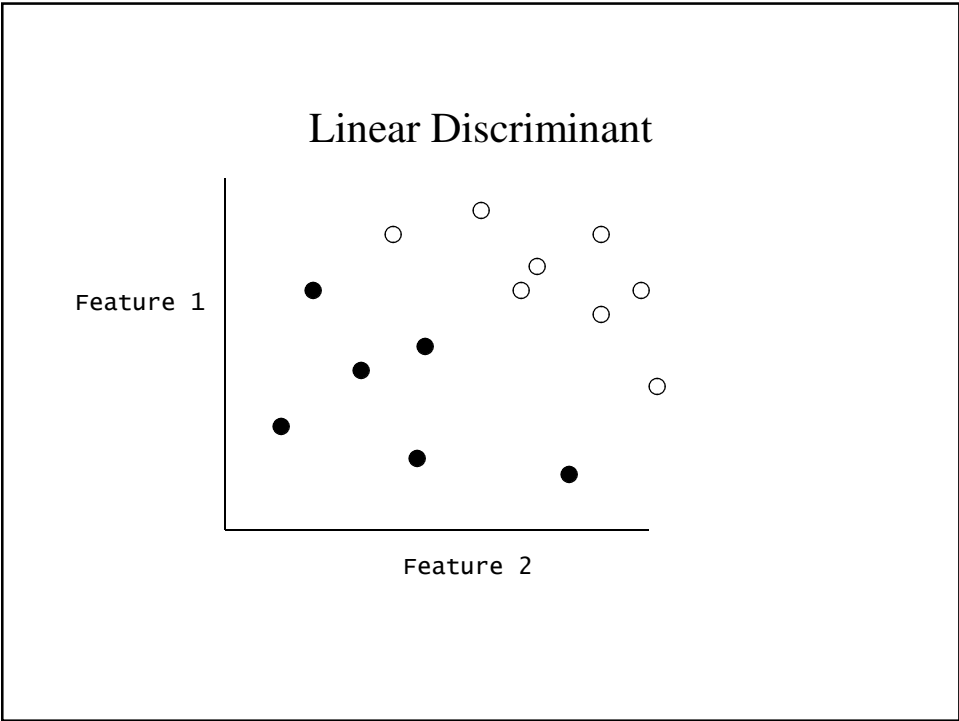


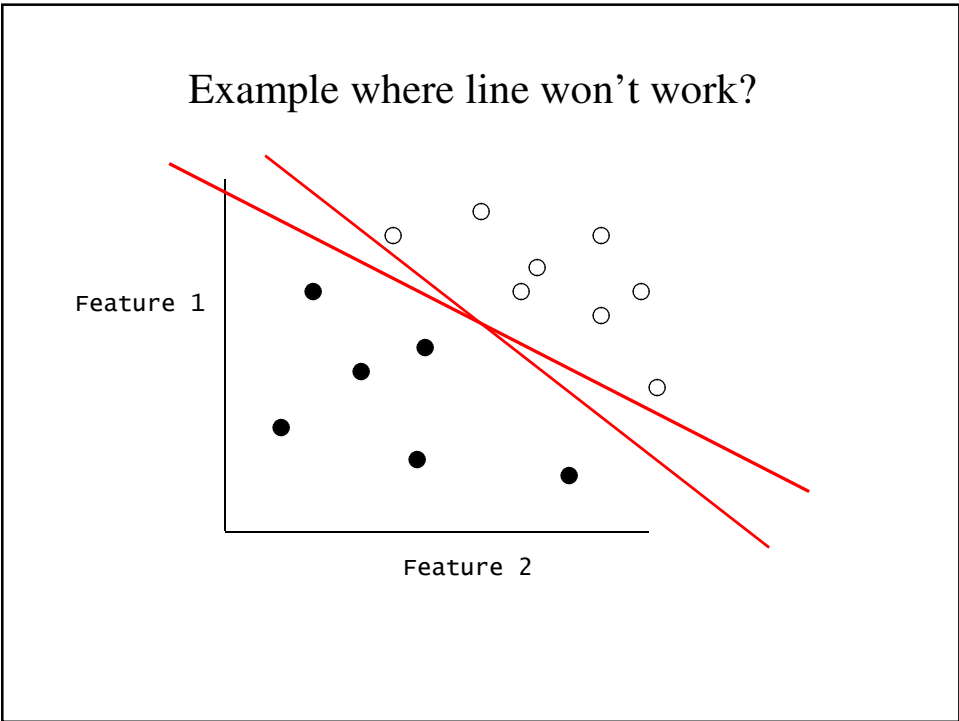
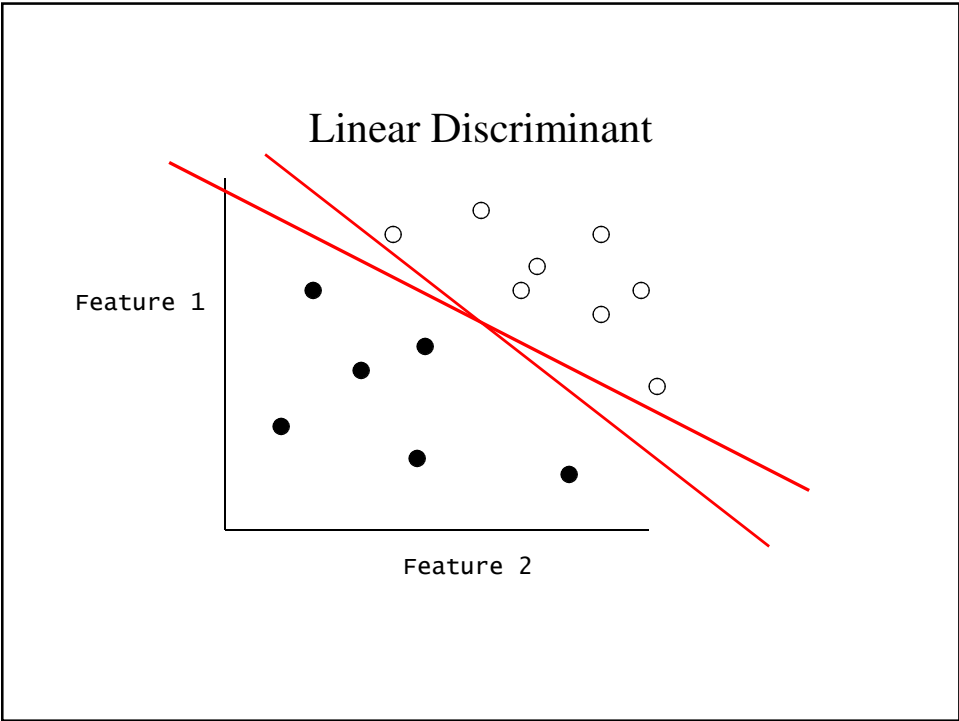
## Linear Classification

Reading: Chapter 4 of C. Bishop book,  
Sections 4.1.1, 4.1.4, 4.1.7,  
on electronic reserve at PSU library.

## Linear Discriminant Functions

- Example: Boolean functions
- Notion of “decision surfaces”
- Linear discriminant function: Assuming data is  $D$ -dimensional, define  $(D-1)$ -dimensional hyperplanes to classify the data into classes.





## Perceptrons

- Precursors to neural networks.
- Discriminant function:

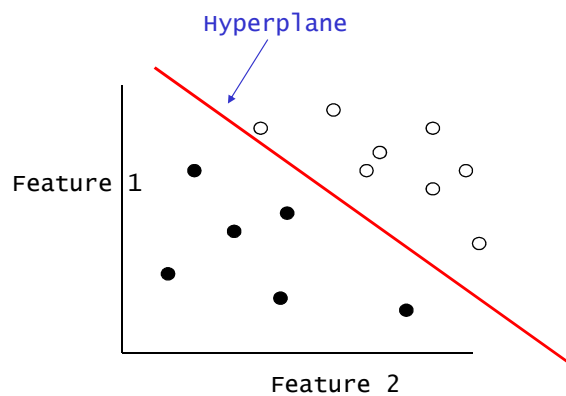
$$y(\mathbf{x}) = f(\mathbf{w}^T \mathbf{x} + w_0)$$
$$= f(w_0 + w_1 x_1 + \dots + w_D x_D)$$

$w_0$  is called the “bias”. (Equivalently,  $-w_0$  is called the “threshold”.)

Classification:  $y(\mathbf{x}) = \text{sgn}(w_1 x_1 + w_2 x_2 + \dots + w_D x_D + w_0)$

$$\text{where } \text{sgn}(y) = \begin{cases} -1 & \text{if } y < 0 \\ +1 & \text{otherwise} \end{cases}$$

## Geometry of the perceptron



In 2d:

$$w_1 x_1 + w_2 x_2 + w_0 = 0$$

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2}$$

## In-class exercise

(a) Find weights for a perceptron that separates “true” and “false” in  $x_1 \wedge x_2$ . Sketch the separation line defined by this discriminant.

(b) Do the same, but for  $x_1 \vee x_2$ .

- To simplify notation, assume a “dummy” coordinate (or attribute)  $x_0 = 1$ . Then we can write:

$$\begin{aligned}y(\mathbf{x}) &= \text{sgn}(\mathbf{w}^T \mathbf{x}) \\ &= \text{sgn}(w_0 x_0 + w_1 x_1 + \dots + w_D x_D)\end{aligned}$$

- We can generalize the perceptron to cases where we project data points  $\mathbf{x}_n$  into “feature space”,  $\phi(\mathbf{x}_n)$ :

$$y(\mathbf{x}) = \text{sgn}(\mathbf{w}^T \phi(\mathbf{x}))$$

## Notation

- Let  $S = \{(\mathbf{x}^n, t^n) : n = 1, 2, \dots, m\}$  be a training set.

Note:  $\mathbf{x}^n$  is a vector of inputs, and  $t^n$  is  $\in \{+1, -1\}$  for binary classification,  $t^n \in \mathfrak{R}$  for regression.

- Output  $o$ :

$$o = \text{sgn}\left(\sum_{j=0}^d w_j x_j = \mathbf{w}^T \cdot \mathbf{x}\right)$$

- Error of a perceptron on a single training example,  $(\mathbf{x}^n, t^n)$

$$E^n = \frac{1}{2}(t^n - o^n)^2$$

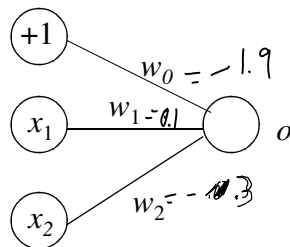
## Example

- $S = \{((0,0), -1), ((0,1), 1)\}$

$$\frac{1}{2}(t-o)^2$$

- Let  $\mathbf{w} = \{w_0, w_1, w_2\} = \{0.1, 0.1, -0.3\}$

$$\Delta w_i =$$



What is  $E^1$ ? = 2

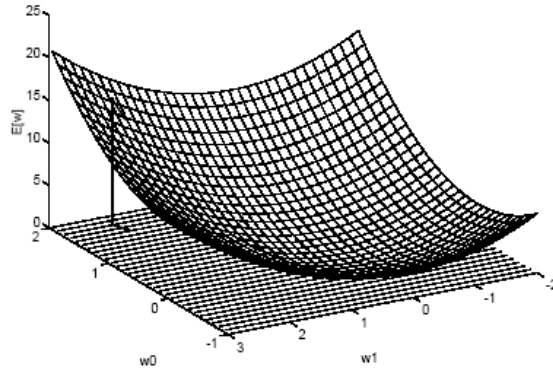
What is  $E^2$ ? = 2

$$\Delta w_j = (t^n - \hat{o}^n) x_j$$

$$\Delta w_0 = (-1 - 1) 1 = -2$$

How do we train a perceptron?

Gradient descent in weight space



From T. M. Mitchell, *Machine Learning*

### Perceptron learning algorithm

- Start with random weights  $\mathbf{w} = (w_1, w_2, \dots, w_d)$ .
- Do gradient descent in weight space, in order to minimize error  $E$ :
  - Given error  $E$ , want to modify weights  $\mathbf{w}$  so as to take a step in direction of steepest descent.

## Gradient descent

- We want to find  $\mathbf{w}$  so as to minimize *sum-squared error*:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=0}^m (t^n - o^n)^2$$

- To minimize, take the derivative of  $E(\mathbf{w})$  with respect to  $\mathbf{w}$ .
- A vector derivative is called a “gradient”:  $\nabla E(\mathbf{w})$

$$\nabla E(\mathbf{w}) = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- Here is how we change each weight:

$$w_j \leftarrow w_j + \Delta w_j$$

where

$$\Delta w_j = -\eta \frac{\partial E}{\partial w_j}$$

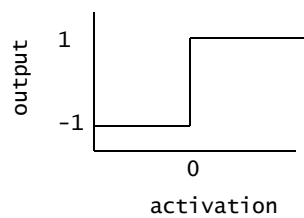
and  $\eta$  is the *learning rate*.

- Error function

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=0}^m (t^n - o^n)^2$$

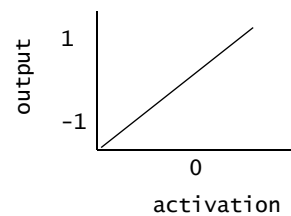
has to be differentiable, so *output function o* also has to be differentiable.

### Activation functions



$$o = \text{sgn}\left(\sum_j w_j x_j + w_0\right)$$

Not differentiable



$$o = \sum_j w_j x_j + w_0$$

Differentiable

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_n (t^n - o^n)^2 \quad (1)$$

$$= \frac{1}{2} \sum_n \frac{\partial}{\partial w_i} (t^n - o^n)^2 \quad (2)$$

$$= \frac{1}{2} \sum_n 2(t^n - o^n) \frac{\partial}{\partial w_i} (t^n - o^n) \quad (3)$$

$$= \sum_n (t^n - o^n) \frac{\partial}{\partial w_i} (t^n - \mathbf{w} \cdot \mathbf{x}^n) \quad (4)$$

$$= \sum_n (t^n - o^n)(-x_i^n) \quad (5)$$

So,

$$\Delta w_i = \eta \sum_n (t^n - o^n) x_i^n \quad (6)$$

This is called the *perceptron learning rule*.

- Problem with *true gradient descent*:

Search process will land in local optimum.

- Common approach to this: use *stochastic gradient descent*:
  - Instead of doing weight update after all training examples have been processed, do weight update after each training example has been processed (i.e., perceptron output has been calculated).
  - Stochastic gradient descent approximates true gradient descent increasingly well as  $\eta \rightarrow 1/\infty$ .

## Training a perceptron

1. Start with random weights,  $\mathbf{w} = (w_1, w_2, \dots, w_d)$ .
2. Select training example  $(\mathbf{x}^n, t^n)$ .
3. Run the perceptron with input  $\mathbf{x}^n$  and weights  $\mathbf{w}$  to obtain  $o$ .
4. Let  $\eta$  be the training rate (a user-set parameter).  
Now,

$$w_i \leftarrow w_i + \Delta w_i$$

where

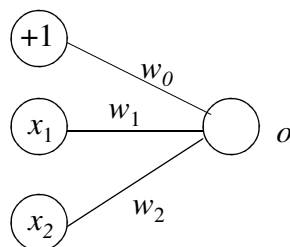
$$\Delta w_i = \eta (t^n - o^n) x_i^n$$

5. Go to 2.

## Example (slightly modified)

- $S = \{((0,0), -1), ((0,1), 1)\}$
- Let  $\mathbf{w} = \{w_0, w_1, w_2\} = \{0.1, 0.1, -0.3\}$

Perceptron weight updates:

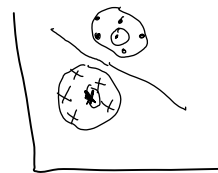


New perceptron weights:

- 1960s: Rosenblatt proved that the perceptron learning rule converges to correct weights in a finite number of steps, provided the training examples are linearly separable.
- 1969: Minsky and Papert proved that perceptrons cannot represent non-linearly separable target functions.

## Fisher's linear discriminant

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$



Note: This projects  $\mathbf{x}$  to one dimension.

We want to find a projection that maximizes the separation of the means of each class, while also giving a small variance within each class to minimize class overlap.

Assume two classes,  $C_1$  and  $C_2$ , with  $N_1$  and  $N_2$  instances, respectively. The means of the two classes are:

$$\mathbf{m}_1 = \frac{1}{N_1} \sum_{n \in C_1} \mathbf{x}_n, \quad \mathbf{m}_2 = \frac{1}{N_2} \sum_{n \in C_2} \mathbf{x}_n$$

Define the *projected class mean* as

$$m_k = \mathbf{w}^T \mathbf{m}_k.$$

Thus we want to choose  $\mathbf{w}$  so as to maximize

$$m_2 - m_1 = \mathbf{w}^T (\mathbf{m}_2 - \mathbf{m}_1)$$

while minimizing within-class variance.

Given our original projection,

$$y(\mathbf{x}) = \mathbf{w}^T \mathbf{x},$$

we can write the variance of the  $y$ 's within class  $C_k$  as:

$$s_k^2 = \sum_{n \in C_k} (y_n - m_k)^2,$$

where  $y_n$  is the value of  $y$  on data-point  $n$  that is a member of class  $C_k$ .

Total within-class variance in the data set is

$$s_1^2 + s_2^2.$$

Define the *Fisher criterion* as

$$J(\mathbf{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2},$$

i.e., the ratio of between-class variance to within-class variance.

We want to maximize this.

To maximize this:

- Compute *between-class covariance matrix* :

$$S_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T$$

- Compute *total within-class covariance matrix* :

$$S_W = \sum_{n \in C_1} (\mathbf{x}_n - \mathbf{m}_1)(\mathbf{x}_n - \mathbf{m}_1)^T + \sum_{n \in C_2} (\mathbf{x}_n - \mathbf{m}_2)(\mathbf{x}_n - \mathbf{m}_2)^T$$

Now we can write  $J(\mathbf{w})$  as follows:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$$

$$J(\mathbf{w}) = \frac{\mathbf{w}^T \mathbf{S}_B \mathbf{w}}{\mathbf{w}^T \mathbf{S}_w \mathbf{w}}$$

Now we can maximize  $J(\mathbf{w})$  by differentiating the above expression.

Result:  $J(\mathbf{w})$  is maximized when

$$\mathbf{w}^T \mathbf{S}_B \mathbf{w} \mathbf{S}_w \mathbf{w} = \mathbf{w}^T \mathbf{S}_w \mathbf{w} \mathbf{S}_B \mathbf{w}$$

Note that we only care about the direction of  $\mathbf{w}$ , not its magnitude, since we can get an equivalent decision surface by multiplying  $\mathbf{w}$  by any scalar. This is because

$$w_0 x_0 + w_1 x_1 + \dots + w_D x_D = 0$$

Thus we can simplify

$$\mathbf{w}^T \mathbf{S}_B \mathbf{w} \mathbf{S}_w \mathbf{w} = \mathbf{w}^T \mathbf{S}_w \mathbf{w} \mathbf{S}_B \mathbf{w}$$

by (1) dropping the scalar factors  $\mathbf{w}^T \mathbf{S}_B \mathbf{w}$  and  $\mathbf{w}^T \mathbf{S}_w \mathbf{w}$  and (2) noting that  $\mathbf{S}_B \mathbf{w}$  is always in the direction of

$(\mathbf{m}_2 - \mathbf{m}_1)$ , to get  $\mathbf{S}_w \mathbf{w} \prec \mathbf{m}_2 - \mathbf{m}_1$ ,

or

$$\mathbf{w} \prec \mathbf{S}_w^{-1} (\mathbf{m}_2 - \mathbf{m}_1)$$

← Fisher's linear discriminant

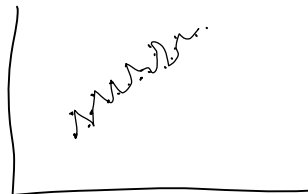
For classification, define a threshold  $y_0$  such that if  $y(\mathbf{x}) < y_0$  then  $\mathbf{x}$  is in class  $C_1$ ; otherwise  $C_2$ .

(More later on finding an optimal threshold.)

- Historical note: Who is Fisher's linear discriminant named for?

## Feature selection/construction

- Adding too many features (i.e., making problem higher-dimensional) decreases performance of classifier.
- Notion of "overfitting".



## Linear classification in feature space

- Principal components analysis (PCA)
  - Reading: L. I. Smith, *A tutorial on principal components analysis* (on class website)
  - PCA used to create high-level features in order to improve classification and reduce dimensions of data without much loss of information.
  - Used in machine learning and in signal processing and image compression (among other things).

## Background for PCA

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

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

- Covariance of two attributes:

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

- 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,  $A_1, \dots, A_n$ .

- 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

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

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

Covariance:

H	M	(H <sub>i</sub> - $\bar{H}$ )	(M <sub>i</sub> - $\bar{M}$ )	(H <sub>i</sub> - $\bar{H}$ )(M <sub>i</sub> - $\bar{M}$ )
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

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

Covariance matrix

Table 2.2: 2-dimensional data set and covariance calculation

## Review of Matrix Algebra

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

- For any eigenvector  $\mathbf{v}$  of  $\mathbf{M}$  and scalar  $a$ ,

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

- Thus you can always choose eigenvectors of length 1:

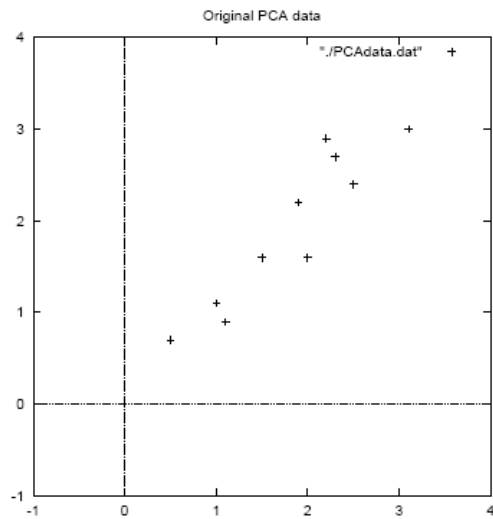
$$\sqrt{v_1^2 + \dots + v_n^2} = 1$$

- If  $\mathbf{M}$  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.

## Principal Components Analysis (PCA)

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

$x$	$y$		$x$	$y$
2.5	2.4		.69	.49
0.5	0.7		-1.31	-1.21
2.2	2.9		.39	.99
1.9	2.2		.09	.29
3.1	3.0	DataAdjust =	1.29	1.09
2.3	2.7		.49	.79
2	1.6		.19	-.31
1	1.1		-.81	-.81
1.5	1.6		-.31	-.31
1.1	0.9		-.71	-1.01
Mean: 1.81	1.91		Mean: 0	0



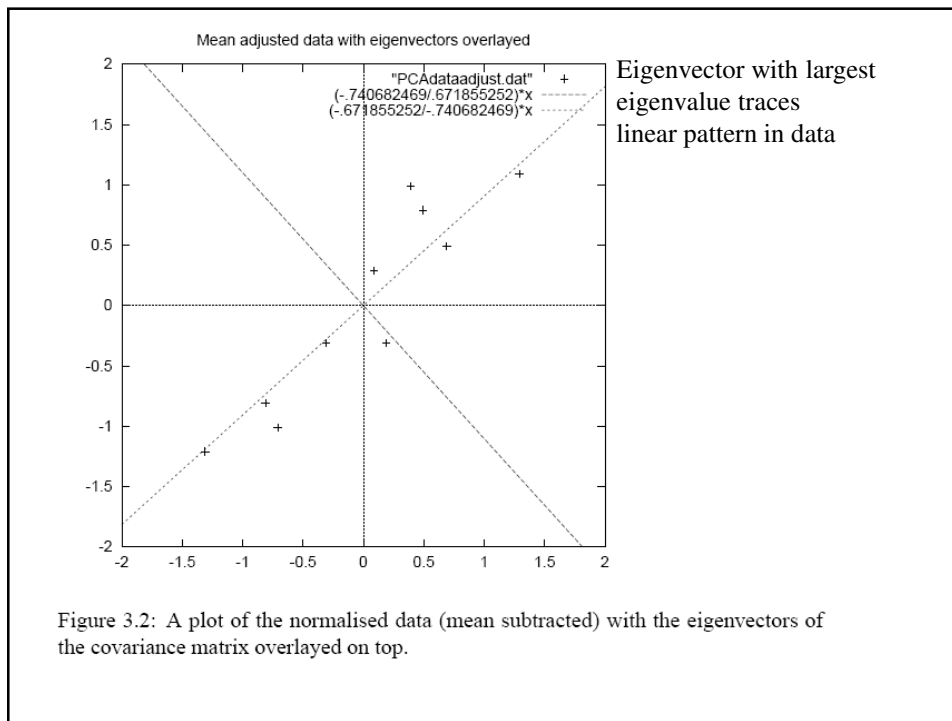
2. Calculate the covariance matrix:

$$cov = \begin{matrix} & \mathbf{x} & \mathbf{y} \\ \mathbf{x} & (.616555556 & .615444444) \\ \mathbf{y} & (.615444444 & .716555556) \end{matrix}$$

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

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

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



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 =  $(\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_p)$

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

or reduced dimension feature vector :

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

5. Derive the new data set.

$$\text{TransformedData} = \text{RowFeatureVector} \times \text{RowDataAdjust}$$

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

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

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

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

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

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

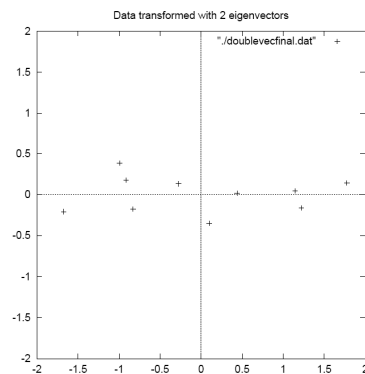


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

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

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

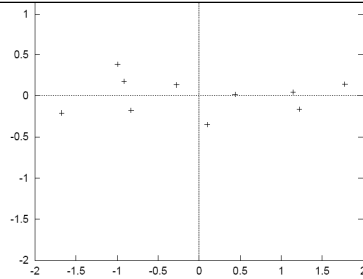
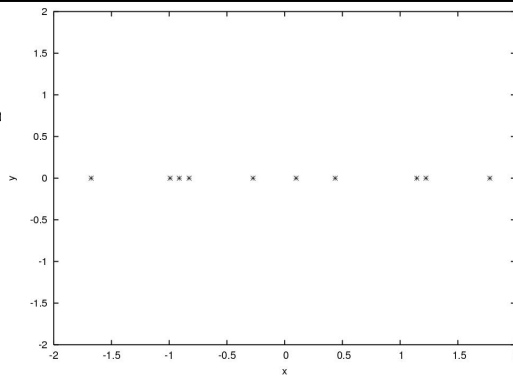


Figure 3.3: The table of data by applying the PCA analysis using both eigenvectors, and a plot of the new data points.

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:

$$\text{TransformedData} = \text{RowFeatureVector} \times \text{RowDataAdjust}$$

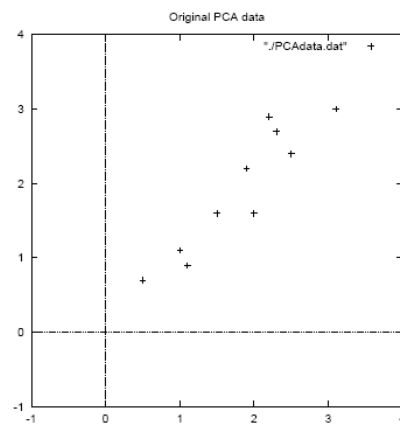
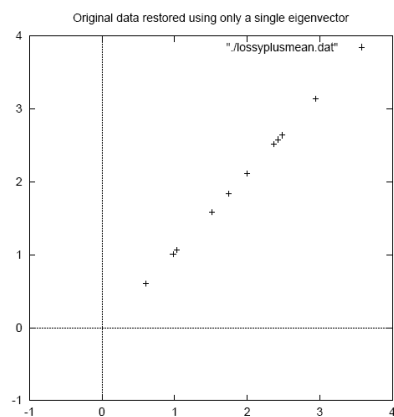
so we can do

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

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

and

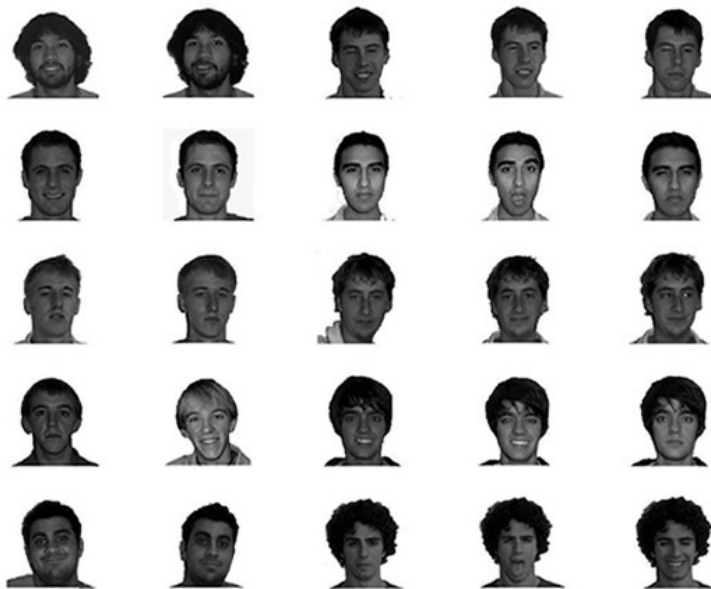
$$\text{RowDataOriginal} = \text{RowDataAdjust} + \text{OriginalMean}$$



## Example: Linear discrimination using PCA for face recognition

### 1. Preprocessing: "Normalize" faces

- Make images the same size
- Line up with respect to eyes
- Normalize intensities



2. Raw features are pixel intensity values (2061 features)
3. Each image is encoded as a vector  $\Gamma_i$  of these features
4. Compute “mean” face in training set:

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

From W. Zhao et al., Discriminant analysis of principal components for face recognition.

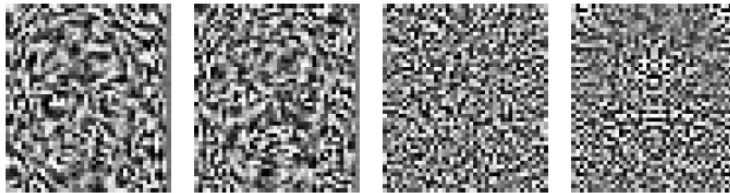


The average face and first four eigenfaces

- Subtract the mean face from each face vector
$$\Phi_i = \Gamma_i - \Psi$$
- Compute the covariance matrix  $\mathbf{C}$
- Compute the (unit) eigenvectors  $\mathbf{v}_i$  of  $\mathbf{C}$
- Keep only the first  $K$  principal components (eigenvectors)



Eigenfaces 15, 100, 200, 250, 300



Eigenfaces 400, 450, 1000, 2000

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

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

Use this representation for:

- Face recognition  
(e.g., Euclidean distance from known faces)
- Linear discrimination  
(e.g., “glasses” versus “no glasses”,  
or “male” versus “female”)

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

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

Use this representation for:

- Face recognition  
(e.g., Euclidean distance from known faces)
- Linear discrimination  
(e.g., “glasses” versus “no glasses”,  
or “male” versus “female”)

## PCA versus Linear Discriminant Analysis

- PCA seeks a new basis that is good for dimensionality reduction.
- LDA seeks a new basis that is good for discrimination.
- Can apply PCA first, then LDA.
- If original data is linearly separable, PCA data is also linearly separable.