ECS-801: Artificial Intelligence (AI)

Unit-V

Pattern Recognition : Introduction, Design principles of pattern recognition system, Statistical Pattern recognition, Parameter estimation methods - Principle Component Analysis (PCA) and Linear Discriminant Analysis (LDA), Classification Techniques – Nearest Neighbor (NN) Rule, Bayes Classifier, Support Vector Machine (SVM), K – means clustering.

Introduction

We de ne/summarize a pattern recognition system using the block diagram in Figure 1.1.

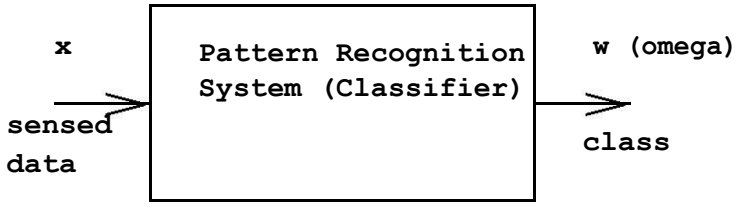
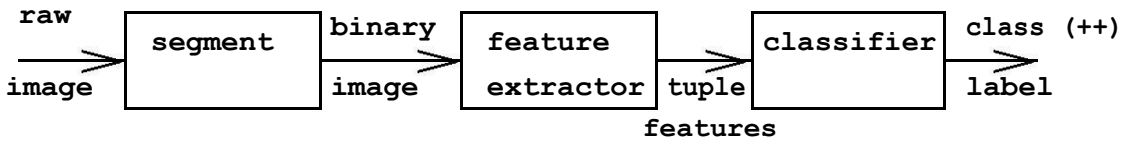


Figure 1.1: Pattern recognition system; **x** a tuple of p measurements, output ω — class label.

In some cases, this might work; i.e. we collect the pixel values of a sub-image that contains the object of interest and place them in a p-tuple **x** and leave everything to the classi er algorithm. However, Figure 1.2 shows a more realistic setup. In this arrangement — still somewhat idealised, for example, we assume that we have somehow isolated the object in a (sub)image — we segment the image into object pixels (value 1, say) and background pixels (value 0).

In such a setup we can do all the problem speci c processing in the rst two stages, and pass the feature vector (in general p-dimensional) to a general purpose classi er.



(*) General pattern recognition

* For example, in the trivial case of identifying '0' and '1' we could have a feature vector (x1 = width, x2 = ink area).

++ class = 0, 1; in a true OCR problem (A-Z;0-9), class = 1..36.

Figure 1.2: Image pattern recognition — problem separated.

Pattern recognition (classi cation) may be posed as an inference problem. The inference involves class labels, that is we have a set of examples (training data), $\mathbf{X}_T = \{\mathbf{x}_i, \omega_i\}_{i=1}^{n}$. **x** is the pattern vector — of course, we freely admit that in certain situations **x** is a simple scalar. ω is the class label, $\omega \in \Omega = \{\omega_1, \ldots, \omega_c\}$; given an unseen pattern **x**, we infer ω. In general, $\mathbf{x} = (x_0\ x_1 \ldots x_{p-1})^T$, a p-dimensional vector; ^{T} denotes transposition.
From now on we concentrate on classi er algorithms, and we refer only occasionally to concrete examples.

2 Simple classifier algorithms:

2.1 Thresholding for one-dimensional data

In our simplistic character recognition system we require to recognise two characters, `0' and `1'. We extract two features: width and inked area. These comprise the feature vector, $\mathbf{x} = (x_1\ x_2)^T$.

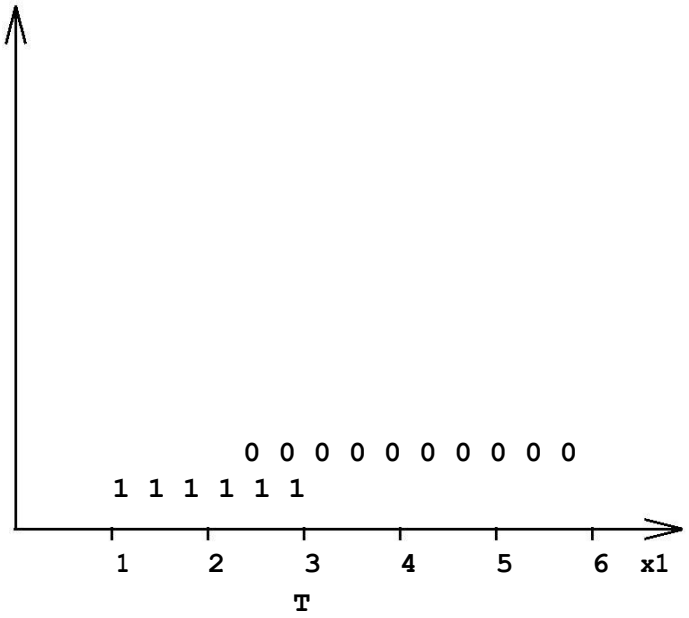


Figure 2.1: Width data, x_1.

Let us see whether we can recognise using width alone (x_1. Figure 2.1 shows some (training) data that has been collected. We see that a threshold (T) set at about x_1 = 2:8 is the best we can do; the classi cation algorithm is:

$$\omega = 1 \text{ when } x_1 \quad T; \tag{2.1}$$

$$= 0 \text{ otherwise:} \tag{2.2}$$

Use of histograms, see Figure 2.2 might be a more methodical way of determining the threshold, T.

If enough training data were available, n ! ∞, the histograms, $h_0(x_1)$; $h_1(x_1)$, properly normalised would approach probability densities: $p_0(x_1)$; $p_1(x_1)$, more properly called ***class conditional*** probability densities: $p(x_1 \mid \omega)$; ω = 0; 1, see Figure 2.3.

When the feature vector is three-dimensional (p = 3) or more, it becomes impossible to estimate the probability densities using histogram binning — there are a great many bins, and most of them contain no data.

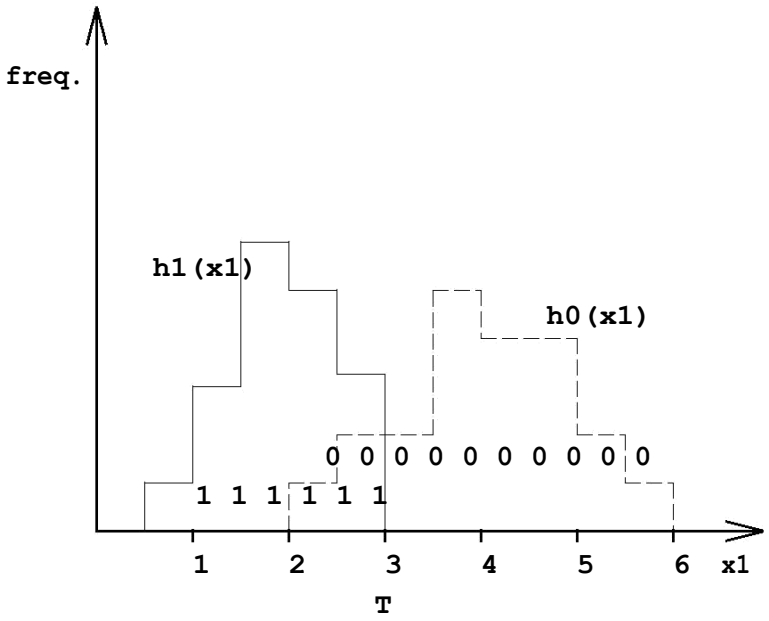


Figure 2.2: Width data, x_1, with histogram.

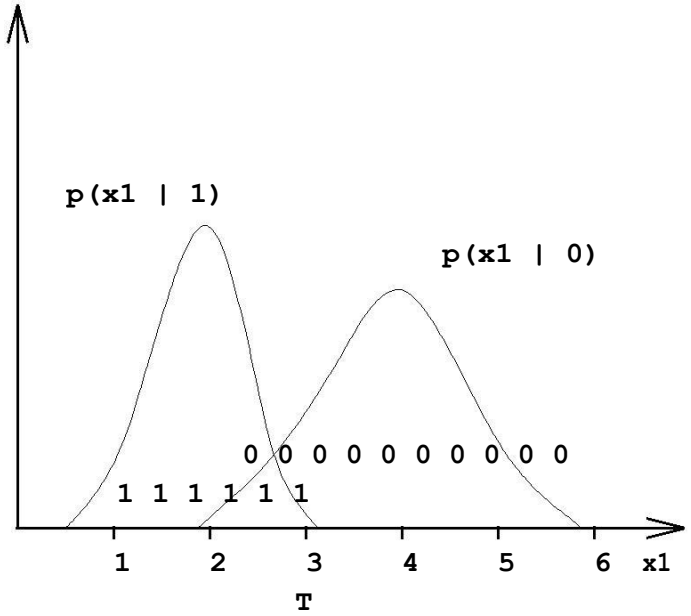


Figure 2.3: Class conditional densities.

2.2 Linear separating lines/planes for two-dimensions

Since there is overlap in the width, x_1, measurement, let us use the two features, $\mathbf{x} = (x_1 x_2)^T$, i.e. (width, area). Figure 2.4 shows a scatter plot of these data.

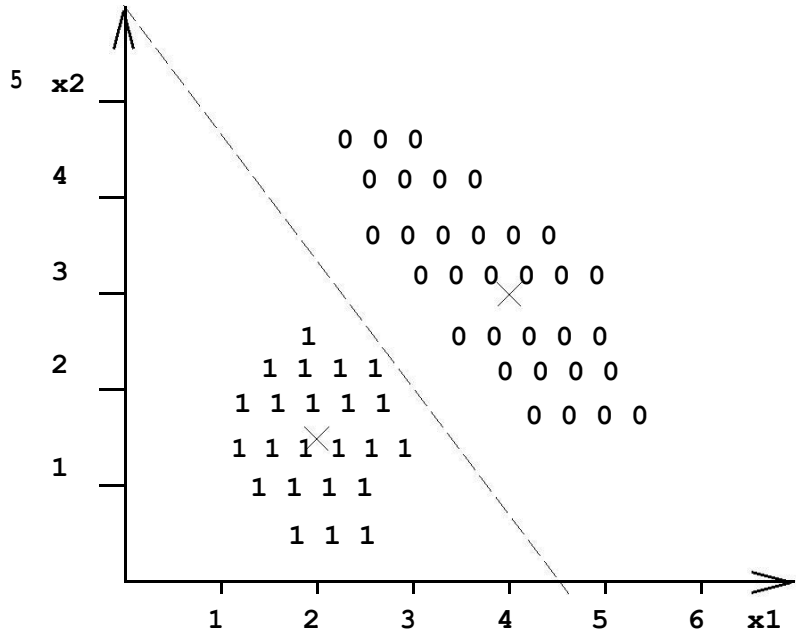


Figure 2.4: Two dimensions, scatter plot.

The dotted line shows that the data are separable by a straight line; it intercepts the axes at $x_1 = 4{:}5$ and $x_2 = 6$.

Apart from plotting the data and drawing the line, how could we derive it from the data? (Thinking of a computer program.)

2.3 Nearest mean classifier:

Figure 2.5 shows the line joining the class means and the perpendicular bisector of this line; the perpendicular bisector turns out to be the separating line. We can derive the equation of the separating line using the fact that points on it are equidistant to both means, μ_0; μ_1, and expand using Pythagoras's theorem,

$$\mathrm{j}\mathbf{x} \quad \mu_0\mathrm{j}^2 \;=\; \mathrm{j}\mathbf{x} \quad \mu_1\mathrm{j}^2; \tag{2.3}$$

$$(x_1 \quad \mu_{01})^2 + (x_2 \quad \mu_{02})^2 \;=\; (x_1 \quad \mu_{11})^2 + (x_2 \quad \mu_{12})^2: \tag{2.4}$$

We eventually obtain

$$(\mu_{01} \quad \mu_{11})x_1 + (\mu_{02} \quad \mu_{12})x_2 \quad (\mu_{01}{}^2 + \mu_{02}{}^2 \quad \mu_{11}{}^2 \quad \mu_{12}{}^2) = 0; \tag{2.5}$$

which is of the form

$$b_1 x_1 + b_2 x_2 \quad b_0 = 0: \tag{2.6}$$

In Figure 2.5, $\mu_{01} = 4$; $\mu_{02} = 3$; $\mu_{11} = 2$; $\mu_{12} = 1{:}5$; with these values, eqn 2.6 becomes

$$4x_1 + 3x_2 \quad 18{:}75 = 0; \tag{2.7}$$

which intercepts the x_1 axis at $18{:}75{=}4 \;\; 4{:}7$ and the x_2 axis at $18{:}75{=}3 = 6{:}25$.

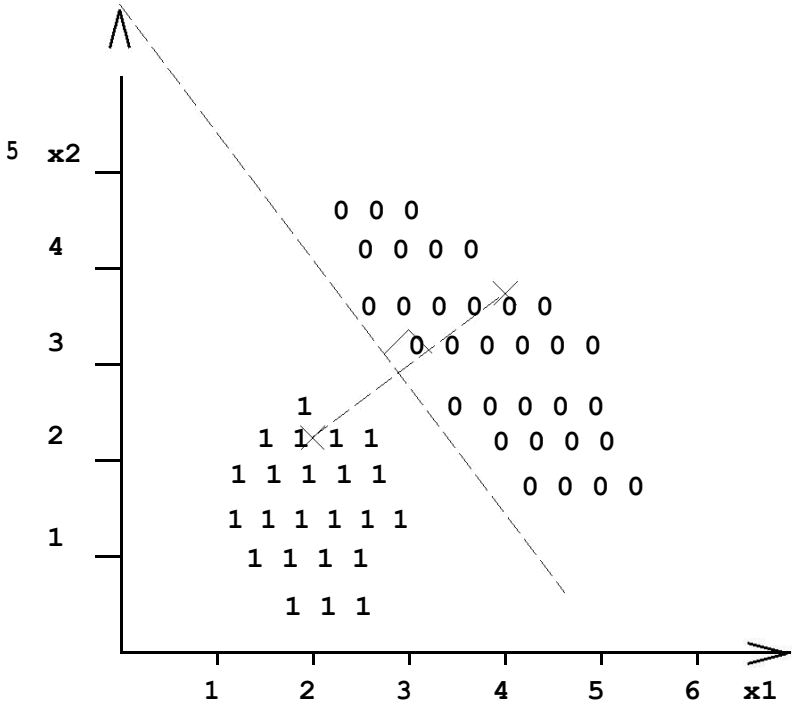


Figure 2.5: Two dimensional scatter plot showing means and separating line.

2.4 Normal form of the separating line, projections, and linear discriminants

Eqn 2.6 becomes more interesting and useful in its ***normal form***,

$$a_1x_1 + a_2x_2 \quad a_0 = 0; \tag{2.8}$$

where $a_1^2 + a_2^2 = 1$; eqn 2.8 can be obtained from eqn 2.6 by dividing across by $\sqrt{b_1^2 + b_2^2}$.
Figure 2.6 shows interpretations of the ***normal form*** straight line equation, eqn 2.8. The coef cients of the unit vector normal to the line are $\mathbf{n} = (a_1a_2)^T$ and a_0 is the perpendicular distance from the line to the origin. Incidentally, the components correspond to the ***direction cosines*** of $\mathbf{n} = (a_1a_2)^T = (\cos\theta\ \sin\theta a_2)^T$. Thus, (Foley, van Dam, Feiner, Hughes & Phillips 1994) **n** corresponds to one row of a (frame) rotating matrix; in other words, see below, section 2.5, dot product of the vector expression of a point with **n**, corresponds to projection onto **n**. (Note that $\cos\pi=2\ \theta = \sin\theta$.)

Also as shown in Figure 2.6, points $\mathbf{x} = (x_1x_2)^T$ on the side of the line to which $= (a_1a_2)^T$ points have $a_1x_1 + a_2x_2\ a_0 > 0$, **n** whilst points on the other side have $a_1x_1 + a_2x_2\ a_0 < 0$; as we know, points ***on*** the line have $a_1x_1 + a_2x_2\ a_0 = 0$.

2.5 Projection and linear discriminant

We know that $a_1x_1 + a_2x_2 = \mathbf{a}^T\mathbf{x}$, the ***dot*** product of $\mathbf{n} = (a_1a_2)^T$ and **x** represents the ***projection*** of points **x** onto **n** — yielding the scalar value along **n**, with a_0 xing the origin. This is plausible: projecting onto **n** yields optimum separability.

Such a projection,

$$g(\mathbf{x}) = a_1x_1 + a_2x_2; \tag{2.9}$$

is called a ***linear discriminant***; now we can adapt equation eqn. 2.2,

$$\omega = 0 \text{ when } g(\mathbf{x}) > a_0; \tag{2.10}$$
$$= 1;\ g(\mathbf{x}) < a_0; \tag{2.11}$$
$$= \text{tie};\ g(\mathbf{x}) = a_0: \tag{2.12}$$

Linear discriminants, eqn. 2.12, are often written as

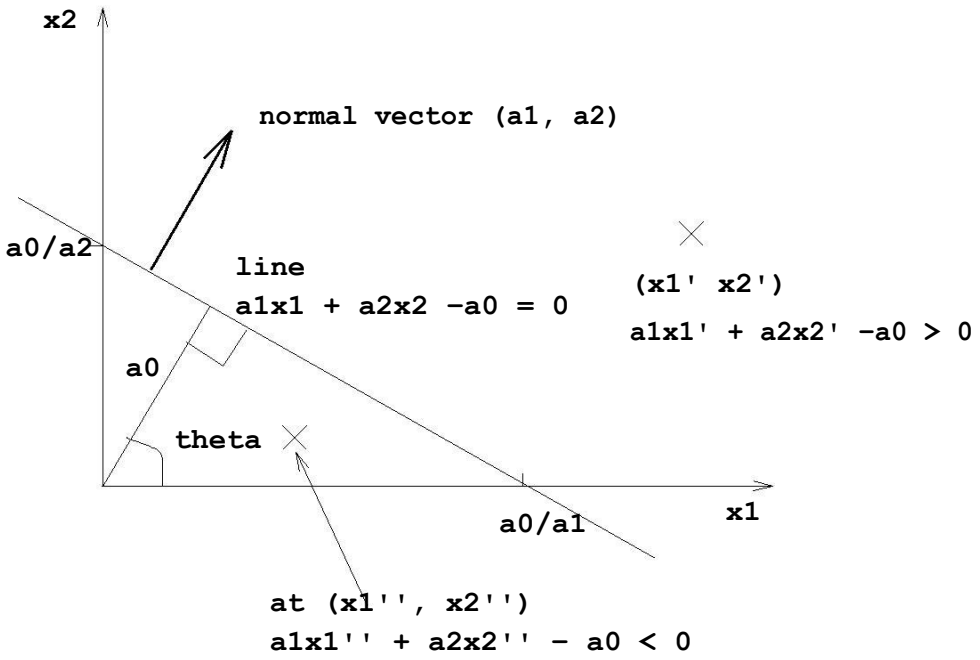


Figure 2.6: Normal form of a straight line, interpretations.

$$g(\mathbf{x}) = a_1 x_1 + a_2 x_2 \quad a_0; \tag{2.13}$$

whence eqn. 2.12 becomes

$$\omega = 0 \text{ when } g(\mathbf{x}) > 0; \tag{2.14}$$
$$= 1;\ g(\mathbf{x}) < 0; \tag{2.15}$$
$$= \text{tie};\ g(\mathbf{x}) = 0: \tag{2.16}$$

2.6 Projections and linear discriminants in p dimensions

Equation 2.13 readily generalises to ***p*** dimensions, **n** is a unit vector in ***p*** dimensional space, ***normal to the the p*** 1 ***separating hyperplane***. For example, when ***p*** = 3, **n** is the unit vector normal to the ***separating plane***.

Other important projections used in pattern recognition are Principal Components Analysis (PCA), see section A.1 and Fisher's Linear Discriminant, see section A.2.

2.7 Template Matching and Discriminants

An intuitive (but well founded) classi cation method is that of ***template matching*** or ***correlation matching***. Here we have perfect or average examples of classes stored in vectors $\{\mathbf{z}_j\}_{j=1}^{c}$, one for each class. Without loss of generality, we assume that all vectors are normalised to unit length.

Classi cation of an newly arrived vector **x** entails computing its ***template/correlation match*** with all ***c*** templates:

$$\mathbf{x}^T \mathbf{z}_j; \tag{2.17}$$

class ω is chosen as ***j*** corresponding to the maximum of eqn. 2.17.

Yet again we see that classi cation involves dot product, projection, and a linear discriminant.

2.8 Nearest neighbour methods

Obviously, we may not always have the linear separability of Figure 2.5. One ***non-parametric*** method is to go beyond nearest mean, see eqn. 2.4, to compute the ***nearest neighbour*** in the entire training data set, and to decide class according to the class of the nearest neighbour.

A variation is ***k nearest neighbour***, where a vote is taken over the classes of k nearest neighbours.

3.Statistical methods:

Recall Figure 2.3, repeated here as Figure 3.1.

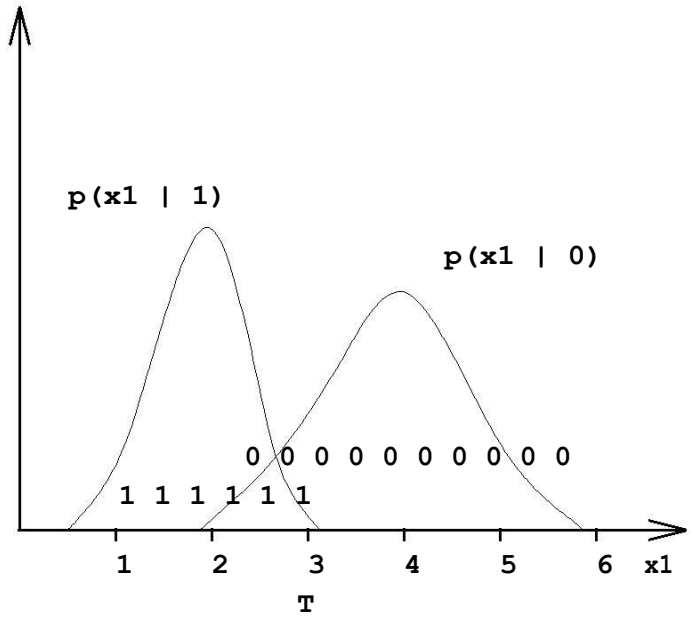


Figure 3.1: Class conditional densities.

We have class conditional probability densities: $p(x_1 \mid \omega)$; $\omega = 0; 1$; given a newly arrived x_1^0 we might decide on its class according to the maximum class conditional probability density at x_1^0, i.e. set a threshold T where $p(x_1 \mid 0)$ and $p(x_1 \mid 1)$ cross, see Figure 3.1.

This is not completely correct. What we want is the ***probability*** of each class (its ***posterior probability***) based on the evidence supplied by the data, combined with any prior evidence.

In what follows, $P(\omega \mid \mathbf{x})$ is the ***posterior probability*** or "a posteriori probability" of class ω given the observation $\mathbf{x}$; $P(w)$ is the ***prior probability*** or "a priori probability". We use upper case $P(:)$ for discrete probabilities, whilst lower case $p(:)$ denotes probability densities.

Let the Bayes decision rule be represented by a function $g(:)$ of the feature vector $\mathbf{x}$:

$$g(\mathbf{x}) = \textit{arg max}_{w_j 2 W}[P(\omega_j \mid \mathbf{x})] \tag{3.1}$$

To show that the Bayes decision rule, eqn. 3.1, achieves the minimum probability of error, we compute the probability of error conditional on the feature vector $\mathbf{x}$ — the ***conditional risk*** — associated with it:

$$R(g(\mathbf{x}) = \omega_j \mid \mathbf{x}) = \sum_{k=1;k6=j}^{c} P(\omega_k \mid \mathbf{x}): \tag{3.2}$$

That is to say, for the point $\mathbf{x}$ we compute the posterior probabilities of all the c 1 classes not chosen.

Since $\Omega = f\omega_1; : : : ; \omega_c g$ form an event space (they are mutually exclusive and exhaustive) and the $P(\omega_k \mid \mathbf{x})_{k=1}^{c}$ are probabili-ties and so sum to unity, eqn. 3.2 reduces to:

$$R(g(\mathbf{x}) = \omega_j) = 1 \quad P(\omega_j \mid \mathbf{x}): \tag{3.3}$$

It immediately follows that, to minimise $R(g(\mathbf{x}) = \omega_j)$, we maximise $P(\omega_j \mid \mathbf{x})$, thus establishing the optimality of eqn. 3.1.

The problem now is to determine $P(\omega \mid \mathbf{x})$ which brings us to ***Bayes' rule***.

3.1 Bayes' Rule for the Inversion of Conditional Probabilities

From the de nition of conditional probability, we have:

$$p(\omega; \mathbf{x}) = P(\omega \mid \mathbf{x})\, p(\mathbf{x}); \tag{3.4}$$

and, owing to the fact that the events in a joint probability are interchangeable, we can equate the joint probabilities :

$$p(\omega; \mathbf{x}) = p(\mathbf{x}; \omega) = p(\mathbf{x} \mid \omega)P(\omega): \tag{3.5}$$

Therefore, equating the right hand sides of these equations, and rearranging, we arrive at ***Bayes' rule*** for the posterior proba-bility $P(\omega \mid \mathbf{x})$:

$$P(\omega \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega)P(\omega)}{p(\mathbf{x})} \quad : \tag{3.6}$$

$P(\omega)$ expresses our belief that ω will occur, prior to any observation. If we have no prior knowledge, we can assume equal priors for each class: $P(\omega^1) = P(\omega^2) = : : : = P(\omega^c); \sum_{j=1}^{c} P(\omega^j) = 1$. Although we avoid further discussion here, we note that the matter of choice of prior probabilities is the subject of considerable discussion especially in the literature on ***Bayesian inference***, see, for example, (Sivia 1996).

$p(\mathbf{x})$ is the ***un***conditional probability density of $\mathbf{x}$, and can be obtained by summing the conditional densities:

$$p(\mathbf{x}) = \sum_{j=1}^{c} p(\mathbf{x} \mid \omega_j)P(\omega_j): \tag{3.7}$$

Thus, to solve eqn. 3.6, it remains to estimate the conditional densities.

3.2 Parametric Methods

Where we can assume that the densities follow a particular form, for example Gaussian, the density estimation problem is reduced to that of estimation of parameters.

The multivariate Gaussian or "Normal" density, p-dimensional, is given by:

$$p(\mathbf{x} \mid \omega_j) = \frac{1}{(2\pi)^{p=2} \mid \mathbf{K}_j \mid^{1=2}} \exp\left[\frac{1}{2}(\mathbf{x} \quad \boldsymbol{\mu}_j)^{T} \mathbf{K}_j^{\ 1}(\mathbf{x} \quad \boldsymbol{\mu}_j)\right] \tag{3.8}$$

$p(\mathbf{x} \mid \omega_j)$ is completely speci ed by $\boldsymbol{\mu}_j$, the p-dimensional mean vector, and $\mathbf{K}_j$ the corresponding $p \quad p$ ***covariance matrix***:

$$\boldsymbol{\mu}_j = \mathbf{E}[\mathbf{x}]_{w=w_j}; \tag{3.9}$$

$$\mathbf{K}_j = \mathbf{E}[(\mathbf{x} \quad \boldsymbol{\mu}_j)(\mathbf{x} \quad \boldsymbol{\mu}_j)^{T}]_{w=w_j}: \tag{3.10}$$

The respective maximum likelihood estimates are:

$$\boldsymbol{\mu}_j = \frac{1}{N_j} \sum_{n=1}^{N_j} \mathbf{x}_n; \tag{3.11}$$

and,

$$\mathbf{K}_j = \frac{1}{N_j \quad 1} \sum_{n=1}^{N_j} (\mathbf{x}_n \quad \boldsymbol{\mu}_j)(\mathbf{x}_n \quad \boldsymbol{\mu}_j)^{t}; \tag{3.12}$$

where we have separated the training data $\mathbf{X}_T = f\mathbf{x}_n; \omega_n g_{n=1}^{N}$ into sets according to class.

3.3 Discriminants based on Gaussian Density

We may write eqn. 3.6 as a discriminant function:

$$g_j(\mathbf{x}) = P(\omega_j \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \omega_j) P(\omega_j)}{p(\mathbf{x})} \quad ; \tag{3.13}$$

so that classi cation, eqn. 3.1, becomes a matter of assigning $\mathbf{x}$ to class w_j if,

$$g_j(\mathbf{x}) > g_k(\mathbf{x}); \; \forall \, k \neq j: \tag{3.14}$$

Since $p(\mathbf{x})$, the denominator of eqn. 3.13 is the same for all $g_j(\mathbf{x})$ and since eqn. 3.14 involves comparison only, we may rewrite eqn. 3.13 as

$$g_j(\mathbf{x}) = p(\mathbf{x} \mid \omega_j) P(\omega_j): \tag{3.15}$$

We may derive a further possible discriminant by taking the logarithm of eqn. 3.15 — since logarithm is a monotonically increasing function, application of it preserves relative order of its arguments:

$$g_j(\mathbf{x}) = \log p(\mathbf{x} \mid \omega_j) + \log P(\omega_j): \tag{3.16}$$

In the multivariate Gaussian case, eqn. 3.16 becomes (Duda & Hart 1973),

$$g_j(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_j)^T \mathbf{K}_j^{-1} (\mathbf{x} - \mu_j) - \frac{p}{2}\log 2\pi - \frac{1}{2}\log |\mathbf{K}_j| + \log P(\omega_j) \tag{3.17}$$

Henceforth, we refer to eqn. 3.17 as the ***Bayes-Gauss classi er***.

The multivariate Gaussian density provides a good characterisation of pattern (vector) distribution where we can model the generation of patterns as `ideal pattern plus measurement noise'; for an instance of a measured vector $\mathbf{x}$ from class ω_j:

$$\mathbf{x}_n = \mu_j + \mathbf{e}_n; \tag{3.18}$$

where $\mathbf{e}_n \sim N(0; \mathbf{K}_j)$, that is, the noise covariance is class dependent.

3.4 Bayes-Gauss Classi er – Special Cases

(Duda & Hart 1973, pp. 26–31)

Revealing comparisons with the other learning paradigms which play an important role in this thesis are made possible if we examine particular forms of noise covariance in which the Bayes-Gauss classi er decays to certain interesting limiting forms:

Equal and Diagonal Covariances ($\mathbf{K}_j = \sigma^2 \mathbf{I}; \; \forall \, j$, where $\mathbf{I}$ is the unit matrix); in this case certain important equivalences with eqn. 3.17 can be demonstrated:

- Nearest mean classi er;
- Linear discriminant;
- Template matching;
- Matched lter;
- Single layer neural network classi er.

Equal but Non-diagonal Covariance Matrices.

- Nearest mean classi er using Mahalanobis distance;
  and, as in the case of diagonal covariance,
- Linear discriminant function;
- Single layer neural network;

3.4.1 Equal and Diagonal Covariances

When each class has the same covariance matrix, and these are diagonal, we have, $\mathbf{K}_j = \sigma^2\mathbf{I}$, so that $\mathbf{K}_j^{-1} = \frac{1}{s}\mathbf{I}$. Since the covariance matrices are equal, we can eliminate the $\frac{1}{2}\log|\mathbf{K}_j|$; the $\frac{D}{2}\log 2\pi$ term is constant in any case; thus, including the simplification of the $(\mathbf{x}-\boldsymbol{\mu}_j)^T\mathbf{K}_j^{-1}(\mathbf{x}-\boldsymbol{\mu}_j)$, eqn. 3.17 may be rewritten:

$$g_j(\mathbf{x}) = \frac{1}{2\sigma^2}(\mathbf{x}-\boldsymbol{\mu}_j)^T(\mathbf{x}-\boldsymbol{\mu}_j) + \log P(\omega_j) \tag{3.19}$$

$$= \frac{1}{2\sigma^2}\|\mathbf{x}-\boldsymbol{\mu}_j\|^2 + \log P(\omega_j): \tag{3.20}$$

Nearest mean classifier If we assume equal prior probabilities $P(\omega_j)$, the second term in eqn. 3.20 may be eliminated for comparison purposes and we are left with a ***nearest mean*** classifier.

Linear discriminant If we further expand the squared distance term, we have,

$$g_j(\mathbf{x}) = \frac{1}{2\sigma^2}(\mathbf{x}^T\mathbf{x} - 2\boldsymbol{\mu}_j^T\mathbf{x} + \boldsymbol{\mu}_j^T\boldsymbol{\mu}_j) + \log P(\omega_j); \tag{3.21}$$

which may be rewritten as a ***linear*** discriminant:

$$g_j(\mathbf{x}) = w_{j0} + \mathbf{w}_j^T\mathbf{x} \tag{3.22}$$

where

$$w_{j0} = \frac{1}{2\sigma^2}(\boldsymbol{\mu}_j^T\boldsymbol{\mu}_j) + \log P(\omega_j); \tag{3.23}$$

and

$$\mathbf{w}_j = \frac{1}{\sigma^2}\boldsymbol{\mu}_j: \tag{3.24}$$

Template matching In this latter form the Bayes-Gauss classifier may be seen to be performing ***template matching*** or ***correlation*** matching, where $\mathbf{w}_j$ = constant $\boldsymbol{\mu}_j$, that is, the prototypical pattern for class j, the mean $\boldsymbol{\mu}_j$, is the ***template***.

Matched filter In radar and communications systems a ***matched filter*** detector is an optimum detector of (subsequence) signals, for example, communication symbols. If the vector $\mathbf{x}$ is written as a time series (a digital signal), $x[n]$; $n = 0; 1; : : :$ then the matched filter for each signal j may be implemented as a convolution:

$$y_j[n] = x[n] \otimes h[n] = \sum_{m=0}^{N-1} x[n-m]\,h_j[m]; \tag{3.25}$$

where the kernel $h[:]$ is a time reversed ***template*** — that is, at each time instant, the correlation between $h[:]$ and the last N samples of $x[:]$ are computed. Provided some threshold is exceeded, the signal achieving the maximum correlation is detected.

Single Layer Neural Network If we restrict the problem to two classes, we can write the classification rule as:

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) \quad 0: \omega_1; \textit{otherwise}\ \omega_2 \tag{3.26}$$

$$= w_0 + \mathbf{w}^T\mathbf{x}; \tag{3.27}$$

where $w_0 = \frac{1}{2s^2}(\boldsymbol{\mu}_1^T\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2^T\boldsymbol{\mu}_2) + \log\frac{P(\omega_1)}{P(\omega_2)}$

and $\mathbf{w} = \frac{1}{s^2}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$.

In other words, eqn. 3.27 implements a linear combination, adds a bias, and thresholds the result — that is, a single layer neural network with a hard-limit activation function.

(Duda & Hart 1973) further demonstrate that eqn. 3.20 implements a hyper-plane partitioning of the feature space.

3.4.2 Equal but General Covariances

When each class has the same covariance matrix, **K**, eqn. 3.17 reduces to:

$$g_j(\mathbf{x}) = (\mathbf{x} - \mu_j)^T \mathbf{K}^{-1}(\mathbf{x} - \mu_j) + logP(\omega_j) \tag{3.28}$$

Nearest Mean Classi er, Mahalanobis Distance If we have equal prior probabilities $P(\omega_j)$, we arrive at a ***nearest mean*** classi er where the distance calculation is weighted. The Mahalanobis distance $(\mathbf{x} - \mu_j)^T \mathbf{K}_j^{-1}(\mathbf{x} - \mu_j)$ effectively weights contributions according to inverse variance. Points of equal Mahalanobis distance correspond to points of equal conditional density $p(\mathbf{x} \mid \omega_j)$.

Linear Discriminant Eqn. 3.28 may be rewritten as a ***linear*** discriminant, see also section 2.5:

$$g_j(\mathbf{x}) = w_{j0} + \mathbf{w}_j^T \mathbf{x} \tag{3.29}$$

where

$$w_{j0} = \frac{1}{2}(\mu_j^T \mathbf{K}^{-1} \mu_j) + logP(\omega_j); \tag{3.30}$$

and

$$\mathbf{w}_j = \mathbf{K}^{-1} \mu_j: \tag{3.31}$$

Weighted template matching, matched lter In this latter form the Bayes-Gauss classi er may be seen to be performing ***weighted*** template matching.

Single Layer Neural Network As for the diagonal covariance matrix, it can be easily demonstrated that, for two classes, eqns. 3.29– 3.31 may be implemented by a single neuron. The only difference from eqn. 3.27 is that the non-bias weights, instead of being simple a difference between means, is now weighted by the inverse of the covariance matrix.

3.5 Least square error trained classi er

We can formulate the problem of classi cation as a least-square-error problem. Let us require the classi er to output a class membership indicator 2 [0; 1] for each class, we can write:

$$\mathbf{d} = f(\mathbf{x}) \tag{3.32}$$

where $\mathbf{d} = (d_1; d_2; : : : d_c)^T$ is the c-dimensional vector of class indicators and **x**, as usual, the p-dimensional feature vector. We can express individual class membership indicators as:

$$d_j = b_0 + \sum_{i=1}^{p} b_i x_i + e: \tag{3.33}$$

In order to continue the analysis, we recall the theory of linear regression (Beck & Arnold 1977).

Linear Regression The simplest linear model, $y = mx + c$, of school mathematics, is given by:

$$y = b_0 + b_1 x + e; \tag{3.34}$$

which shows the dependence of the dependent variable y on the independent variable x. In other words, y is a linear function of x and the observation is subject to noise, e; e is assumed to be a zero-mean random process. Strictly eqn. 3.34 is ***af ne***, since b_0 is included, but common usage dictates the use of ***linear***. Taking the nth observation of $(x; y)$, we have (Beck & Arnold 1977, p. 133):

$$y_n = b_0 + b_1 x_n + e_n \tag{3.35}$$

Least square error estimators for b_0 and b_1, $\hat{b}_0$ and $\hat{b}_1$ may be obtained from a set of paired observations $\{x_n; y_n\}_{n=1}^N$ by minimising the sum of squared residuals:

$$S = \sum_{n=1}^{N} r_n^2 = \sum_{n=1}^{N} (y_n - \hat{y}_n)^2 \tag{3.36}$$

$$S = \sum_{n=1}^{N} (y_n - b_0 - b_1 x_n)^2 \tag{3.37}$$

Minimising with respect to b_0 and b_1, and replacing these with their estimators, $\hat{b}_0$ and $\hat{b}_1$, gives the familiar result:

$$\hat{b}_1 = N[\sum y_n x_n - (\sum y_i)(\sum x_i)] / [N(\sum x_i^2) - (\sum x_i)^2] \tag{3.38}$$

$$\hat{b}_0 = \frac{\sum y_n}{N} - \hat{b}_1 \frac{\sum x_n}{N} \tag{3.39}$$

The validity of these estimates does not depend on the distribution of the errors e_n; that is, assumption of Gaussianity is not essential. On the other hand, all the simplest estimation procedures, including eqns. 3.38 and 3.39, assume the x_n to be error free, and that the error e_n is associated with y_n.

In the case where y, still one-dimensional, is a function of many independent variables — p in our usual formulation of p-dimensional feature vectors — eqn. 3.35 becomes:

$$y_n = b_0 + \sum_{i=1}^{p} b_i x_{in} + e_n \tag{3.40}$$

where x_{in} is the i-th component of the n-th feature vector.

Eqn. 3.40 can be written compactly as:

$$y_n = \mathbf{x}_n^T \mathbf{b} + e_n \tag{3.41}$$

where $\mathbf{b} = (b_0; b_1; \ldots; b_p)^T$ is a $p + 1$ dimensional vector of coef cients, and $\mathbf{x}_n = (1; x_{1n}; x_{2n}; \ldots; x_{pn})$ is the ***augmented*** feature vector. The constant 1 in the augmented vector corresponds to the coef cient b_0, that is it is the so called ***bias term*** of neural networks, see sections 2.5 and 4.

All N observation equations may now be collected together:

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e} \tag{3.42}$$

where $\mathbf{y} = (y_1; y_2; \ldots; y_n; \ldots; y_N)^T$ is the $N \times 1$ vector of observations of the dependent variable, and $\mathbf{e} = (e_1; e_2; \ldots; e_n; \ldots; e_N)^T$. $\mathbf{X}$ is the $Np + 1$ matrix formed by N rows of $p + 1$ independent variables.
Now, the sum of squared residuals, eqn. 3.36, becomes:

$$S = (\mathbf{y} - \mathbf{X}\hat{\mathbf{b}})^T (\mathbf{y} - \mathbf{X}\hat{\mathbf{b}}) \tag{3.43}$$

& Arnold 1977, p. 235):

$$\hat{\mathbf{b}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}. \tag{3.44}$$

The jk-th element of the $(p + 1) \times (p + 1)$ matrix $\mathbf{X}^T\mathbf{X}$ is $\sum_{n=1}^{N} x_{nj} x_{nk}$, in other words, just N the jk-th element of the ***autocorrelation*** matrix, $\mathbf{R}$, of the vector of independent variables $\mathbf{x}$ estimated from the N sample vectors.

If, as in the least-square-error classi er, we have multiple dependent variables (y), in this case, c of them, we can replace $\mathbf{y}$ in eqn. 3.44 with an appropriate matrix $N \times c$ matrix $\mathbf{Y}$ formed by N rows each of c observations. Now, eqn. 3.44 becomes:

$$\hat{\mathbf{B}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y} \tag{3.45}$$

$\mathbf{X}^T\mathbf{Y}$ is a $p + 1 \times c$ matrix, and $\hat{\mathbf{B}}$ is a $(p + 1) \times c$ matrix of coef cients — that is, one column of $p + 1$ coef cients for each class.

Eqn. 3.45 de nes the training `algorithm' of our classi er.

Application of the classi er to a feature vector $\mathbf{x}$ may be expressed as:

$$\hat{\mathbf{y}} = \mathbf{B}\mathbf{x}: \tag{3.46}$$

It remains to nd the maximum of the c components of $\hat{\mathbf{y}}$.

In a two-class case, this least-square-error training algorithm yields an identical discriminant to Fisher's linear discriminant (Duda & Hart 1973). Fisher's linear discriminant is described in section A.2.

Eqn. 3.45 has one signi cant weakness: it depends on the condition of the matrix $\mathbf{X}^T\mathbf{X}$. As with any autocorrelation or auto-covariance matrix, this cannot be guaranteed; for example, linearly dependent features will render the matrix singular. In fact, there is an elegant indirect implementation of eqn. 3.45 involving the singular value decomposition (SVD) (Press, Flannery, Teukolsky & Vetterling 1992), (Golub & Van Loan 1989). The ***Widrow-Hoff*** iterative gradient-descent training procedure (Widrow & Lehr 1990) developed in the early 1960s tackles the problem in a different manner.

3.6 Generalised linear discriminant function

Eqn. 2.13 may be adapted to cope with any function(s) of the features x_i; we can de ne a new feature vector $\mathbf{x}^0$ where:

$$x_k^0 = f_k(\mathbf{x}): \tag{3.47}$$

In the pattern recognition literature, the solution of eqn. 3.47 involving now the vector $\mathbf{x}^0$ is called the ***generalised linear discriminant function*** (Duda & Hart 1973, Nilsson 1965).

It is desirable to escape from the xed model of eqn. 3.47: the form of the $f_k(\mathbf{x})$ must be known in advance. Multilayer perceptron (MLP) neural networks provide such a solution. We have already shown the correspondence between the linear model, eqn. 3.41, and a single layer neural network with a single output node and linear activation function. An MLP with appropriate non-linear activation functions, e.g. sigmoid, provides a ***model-free*** and arbitrary non-linear solution to learning the mapping between $\mathbf{x}$ and $\mathbf{y}$ (Kosko 1992, Hecht-Nielsen 1990, Haykin 1999).

Principal Components Analysis and Fisher's Linear Discriminant Analysis

A.1 Principal Components Analysis

Principal component analysis (PCA), also called Karhunen-Loeve transform (Duda, Hart & Stork 2000) is a linear transfor-mation which maps a p-dimensional feature vector $\mathbf{x} \in R^p$ to another vector $\mathbf{y} \in R^p$ where the transformation is optimised such that the components of **y** contain maximum ***information in a least-square-error sense***. In other words, if we take the rst $r < p$ components ($\mathbf{y}' \in R^q$), then using the inverse transformation, we can reproduce **x** with minimum error. Yet another view is that the rst few components of **y** contain most of the ***variance***, that is, in those components, the transformation stretches the data maximally apart. It is this that makes PCA good for visualisation of the data in two dimensions, i.e. the rst two principal components give an optimum view of the spread of the data.

We note however, unlike ***linear discriminant analysis***, see section A.2, PCA does **not** take account of class labels. Hence it is typically a more useful visualisation of the inherent variability of the data.

In general **x** can be represented, without error, by the following expansion:

$$\mathbf{x} = \mathbf{U}\mathbf{y} = \sum_{i=1}^{p} y_i \mathbf{u}_i \tag{A.1}$$

where

$$y_i \text{ is the ith component of } \mathbf{y} \text{ and} \tag{A.2}$$

where

$$\mathbf{U} = (\mathbf{u}_1; \mathbf{u}_2; : : : ; \mathbf{u}_p) \tag{A.3}$$

is an orthonormal matrix:

$$\mathbf{u}_j^t \mathbf{u}_k = \delta_{jk} = 1; \text{ when } i = k; \text{ otherwise } = 0: \tag{A.4}$$

If we truncate the expansion at $i = q$

$$\mathbf{x}' = \mathbf{U}_q\mathbf{y} = \sum_{i=1}^{q} y_i \mathbf{u}_i; \tag{A.5}$$

we obtain a ***least square error*** approximation of **x**, i.e.

$$\|\mathbf{x} - \mathbf{x}'\| = \textit{minimum}: \tag{A.6}$$

The optimum transformation matrix **U** turns out to be the eigenvector matrix of the sample covariance matrix **C**:

$$\mathbf{C} = \frac{1}{N}\mathbf{A}^{\mathbf{t}}\mathbf{A}; \tag{A.7}$$

where **A** is the $N \ p$ sample matrix.

$$\mathbf{U}\mathbf{C}\mathbf{U}^{\mathbf{t}} = \Lambda; \tag{A.8}$$

the diagonal matrix of eigenvalues.

Linear Discriminant Analysis (LDA):

Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting ("curse of dimensionality") and also reduce computational costs.
Ronald A. Fisher formulated the *Linear Discriminant* in 1936 (The Use of Multiple Measurements in Taxonomic Problems), and it also has some practical uses as classifier. The original Linear discriminant was described for a 2-class problem, and it was then later generalized as "multi-class Linear Discriminant Analysis" or "Multiple Discriminant Analysis" by C. R. Rao in 1948 (The utilization of multiple measurements in problems of biological classification)
The general LDA approach is very similar to a Principal Component Analysis (for more information about the PCA, see the previous article Implementing a Principal Component Analysis (PCA) in Python step by step), but in addition to finding the component axes that maximize the variance of our data (PCA), we are additionally interested in the axes that maximize the separation between multiple classes (LDA).
So, in a nutshell, often the goal of an LDA is to project a feature space (a dataset n-dimensional samples) onto a smaller subspace kk (where $k \leq n-1 k \leq n-1$) while maintaining the class-discriminatory information.
In general, dimensionality reduction does not only help reducing computational costs for a given classification task, but it can also be helpful to avoid overfitting by minimizing the error in parameter estimation ("curse of dimensionality").

A.2 Fisher's Linear Discriminant Analysis

In contrast with PCA (see section A.1), ***linear discriminant analysis*** (LDA) transforms the data to provide optimal class separability (Duda et al. 2000) (Fisher 1936).

Fisher's original LDA, for two-class data, is obtained as follows. We introduce a linear discriminant ***u*** (a p-dimensional vector of ***weights*** — the weights are very similar to the weights used in ***neural networks***) which, via a dot product, maps a feature vector **x** to a scalar,

$$y = \mathbf{u}^{\mathbf{t}}\mathbf{x}: \tag{A.9}$$

u is optimised to maximise simultaneously, (a) the separability of the classes (***between-class separability***), and (b) the clus-tering together of same class data (***within-class clustering***).

Mathematically, this criterion can be expressed as:

$$J(\mathbf{u}) = \frac{\mathbf{u}^t \mathbf{SBu}}{\mathbf{u}^t \mathbf{SW}} : \quad \mathbf{u} \quad \text{(A.10)}$$

where **SB** is the ***between-class*** covariance,

$$\mathbf{SB} = (\mathbf{m1} \quad \mathbf{m2})(\mathbf{m1} \quad \mathbf{m2})^t ; \quad \text{(A.11)}$$

and

$$\mathbf{Sw} = \mathbf{C1} + \mathbf{C2}; \quad \text{(A.12)}$$

the sum of the class-conditional covariance matrices, see section A.1. **m1** and **m2** are the class means.

u is now computed as:

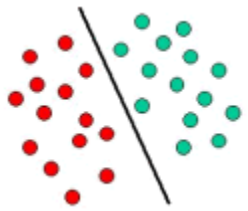
$$\mathbf{u} = \mathbf{S_w}^{1}\mathbf{m1} \quad \mathbf{m2}: \quad \text{(A.13)}$$

There are other formulations of LDA (Duda et al. 2000) (Venables & Ripley 2002), particularly extensions from two-class to multi-class data.

In addition, there are extensions (Duda et al. 2000) (Venables & Ripley 2002) which form a second discriminant, orthogonal to the rst, which optimises the separability and clustering criteria, subject to the orthogonality constraint. The second dimension/discriminant is useful to allow the data to be view as a two-dimensional scatter plot.

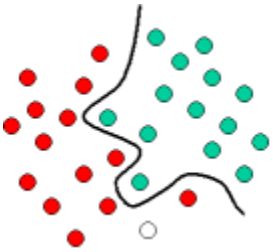
Support Vector Machines (SVM) :

Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).

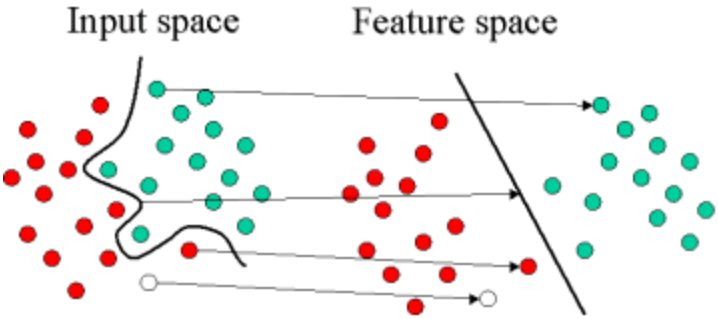


The above is a classic example of a linear classifier, i.e., a classifier that separates a set of objects into their respective groups (GREEN and RED in this case) with a line. Most classification tasks, however, are not that simple, and often more complex structures are needed in order to make an optimal separation, i.e., correctly classify new objects (test cases) on the basis of the examples that are available (train cases). This situation is depicted in the illustration below. Compared to the

previous schematic, it is clear that a full separation of the GREEN and RED objects would require a curve (which is more complex than a line). Classification tasks based on drawing separating lines to distinguish between objects of different class memberships are known as hyperplane classifiers. Support Vector Machines are particularly suited to handle such tasks.



The illustration below shows the basic idea behind Support Vector Machines. Here we see the original objects (left side of the schematic) mapped, i.e., rearranged, using a set of mathematical functions, known as kernels. The process of rearranging the objects is known as mapping (transformation). Note that in this new setting, the mapped objects (right side of the schematic) is linearly separable and, thus, instead of constructing the complex curve (left schematic), all we have to do is to find an optimal line that can separate the GREEN and the RED objects.



Technical Notes

Support Vector Machine (SVM) is primarily a classier method that performs classification tasks by constructing hyperplanes in a multidimensional space that separates cases of different class labels. SVM supports both regression and classification tasks and can handle multiple continuous and categorical variables. For categorical variables a dummy variable is created with case values as either 0 or 1. Thus, a categorical dependent variable consisting of three levels, say (A, B, C), is represented by a set of three dummy variables:

A: {1 0 0}, B: {0 1 0}, C: {0 0 1}

To construct an optimal hyperplane, SVM employs an iterative training algorithm, which is used to minimize an error function. According to the form of the error function, SVM models can be classified into four distinct groups:

- Classification SVM Type 1 (also known as C-SVM classification)
- Classification SVM Type 2 (also known as nu-SVM classification)
- Regression SVM Type 1 (also known as epsilon-SVM regression)
- Regression SVM Type 2 (also known as nu-SVM regression)

Following is a brief summary of each model.

CLASSIFICATION SVM TYPE 1

For this type of SVM, training involves the minimization of the error function:

$$\frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$$

subject to the constraints:

$$y_i \left(w^T \phi(x_i) + b \right) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, i = 1, \ldots, N$$

where C is the capacity constant, w is the vector of coefficients, b is a constant, and ξ_i represents parameters for handling nonseparable data (inputs). The index i labels the N training cases. Note that $y \in \pm 1$ represents the class labels and xi represents the independent variables. The kernel ϕ is used to transform data from the input (independent) to the feature space. It should be noted that the larger the C, the more the error is penalized. Thus, C should be chosen with care to avoid over fitting.

CLASSIFICATION SVM TYPE 2

In contrast to Classification SVM Type 1, the Classification SVM Type 2 model minimizes the error function:

$$\frac{1}{2} w^T w - \nu \rho + \frac{1}{N} \sum_{i=1}^{N} \xi_i$$

subject to the constraints:

$$y_i \left(w^T \phi(x_i) + b \right) \geq \rho - \xi_i, \xi_i \geq 0, i = 1, \ldots, N \text{ and } \rho \geq 0$$

In a regression SVM, you have to estimate the functional dependence of the dependent variable y on a set of independent variables x. It assumes, like other regression problems, that the relationship between the independent and dependent variables is given by a deterministic function f plus the addition of some additive noise:

Regression SVM

y = f(x) + noise

The task is then to find a functional form for f that can correctly predict new cases that the SVM has not been presented with before. This can be achieved by training the SVM model on a sample set, i.e., training set, a process that involves, like classification (see above), the sequential optimization of an error function. Depending on the definition of this error function, two types of SVM models can be recognized:

REGRESSION SVM TYPE 1

For this type of SVM the error function is:

$$\frac{1}{2}w^T w + C\sum_{i=1}^{N}\xi_i + C\sum_{i=1}^{N}\xi^*_i$$

which we minimize subject to:

$$w^T\phi(x_i) + b - y_i \le \varepsilon + \xi_i^*$$
$$y_i - w^T\phi(x_i) - b_i \le \varepsilon + \xi_i$$
$$\xi_i, \xi_i^* \ge 0, i = 1,\ldots,N$$

REGRESSION SVM TYPE 2

For this SVM model, the error function is given by:

$$\frac{1}{2}w^T w - C\left(v\varepsilon + \frac{1}{N}\sum_{i=1}^{N}\left(\xi_i + \xi_i^*\right)\right)$$

which we minimize subject to:

$$\left(w^T\phi(x_i) + b\right) - y_i \le \varepsilon + \xi_i$$
$$y_i - \left(w^T\phi(x_i) + b_i\right) \le \varepsilon + \xi^*_i$$
$$\xi_i, \xi^*_i \ge 0, i = 1,\ldots,N, \varepsilon \ge 0$$

There are number of kernels that can be used in Support Vector Machines models. These include linear, polynomial, radial basis function (RBF) and sigmoid:

Kernel Functions

$$K(\mathbf{X_i}, \mathbf{X_j}) = \begin{cases} \mathbf{X_i} \cdot \mathbf{X_j} & \text{Linear} \\ \left(\gamma \mathbf{X_i} \cdot \mathbf{X_j} + \mathrm{C}\right)^{\mathrm{d}} & \text{Polynomial} \\ \exp\left(-\gamma \mid \mathbf{X_i} - \mathbf{X_j} \mid^2\right) & \text{RBF} \\ \tanh\left(\gamma \mathbf{X_i} \cdot \mathbf{X_j} + \mathrm{C}\right) & \text{Sigmoid} \end{cases}$$

where $K(\mathbf{X_i}, \mathbf{X_j}) = \phi(\mathbf{X_i}) \bullet \phi(\mathbf{X_j})$

that is, the kernel function, represents a dot product of input data points mapped into the higher dimensional feature space by transformation ϕ

***Gamma* is an adjustable parameter of certain kernel functions.**

The RBF is by far the most popular choice of kernel types used in Support Vector Machines. This is mainly because of their localized and finite responses across the entire range of the real x-axis.

k-means clustering algorithm

k-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this

algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} \left(\| x_i - v_j \| \right)^2$$

where,

'$||x_i - v_j||$' is the Euclidean distance between x_i and v_j.

'c_i' is the number of data points in i^{th} cluster.

'c' is the number of cluster centers.

Algorithmic steps for k-means clustering

Let X = $\{x_1, x_2, x_3, \ldots\ldots, x_n\}$ be the set of data points and V = $\{v_1, v_2, \ldots\ldots, v_c\}$ be the set of centers.

1) Randomly select 'c' cluster centers.

2) Calculate the distance between each data point and cluster centers.

3) Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers..

4) Recalculate the new cluster center using:

$$v_i = (1 / c_i) \sum_{j=1}^{c_i} x_i$$

where, 'c_i' represents the number of data points in i^{th} cluster.

5) Recalculate the distance between each data point and new obtained cluster centers.

6) If no data point was reassigned then stop, otherwise repeat from step 3).

Advantages

1) Fast, robust and easier to understand.

2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t is # iterations. Normally, k, t, d << n.

3) Gives best result when data set are distinct or well separated from each other.

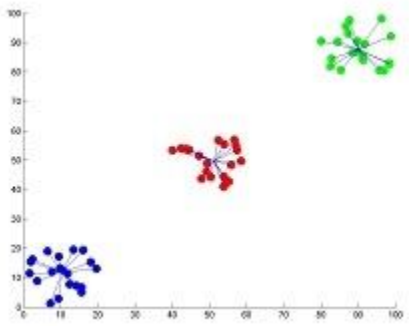


Fig I: Showing the result of k-means for *'N'* = 60 and *'c'* = 3

Note: For more detailed figure for k-means algorithm please refer to k-means figure sub page.

Disadvantages

1) The learning algorithm requires apriori specification of the number of cluster centers.

2) The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.

3) The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).

4) Euclidean distance measures can unequally weight underlying factors.

5) The learning algorithm provides the local optima of the squared error function.

6) Randomly choosing of the cluster center cannot lead us to the fruitful result. Pl. refer Fig.

7) Applicable only when mean is defined i.e. fails for categorical data.

8) Unable to handle noisy data and outliers.

9) Algorithm fails for non-linear data set.

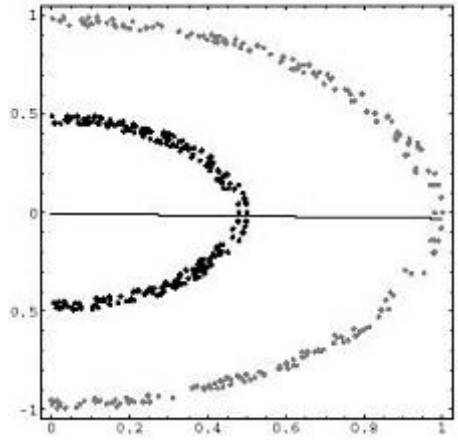


Fig II: Showing the non-linear data set where k-means algorithm fails