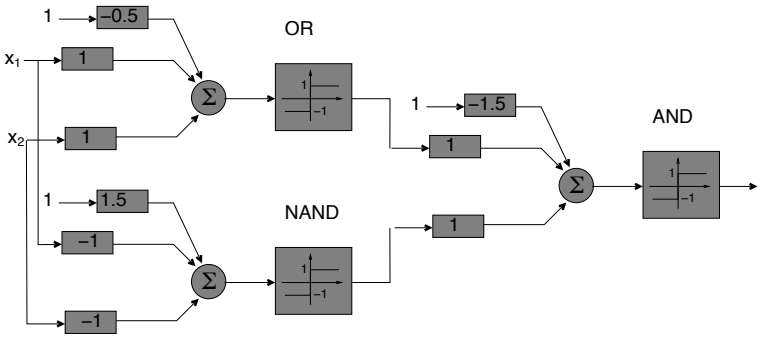
University of Cambridge Engineering Part IIB

Module 4F10: Statistical Pattern Processing

Handout 7: Single Layer Perceptron & Estimating Linear Classifiers



Mark Gales
mjfg@eng.cam.ac.uk
Michaelmas 2013

Introduction

In the previous lectures generative models with Gaussian and Gaussian mixture model class-conditional PDFs were discussed. For the case of tied covariance matrices and Gaussian PDFs, the resulting decision boundary is linear. The linear decision boundary from the generative model was briefly compared with a discriminative model, logistic regression.

An alternative to modelling the class conditional probability density functions is to decide on some functional form for a discriminant function and attempt to construct a mapping from the observation to the class directly.

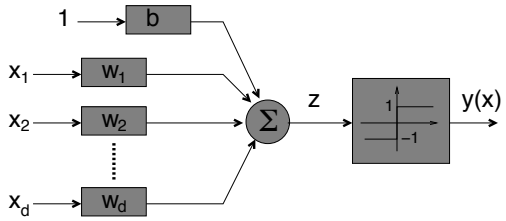
Here we will concentrate on the construction of linear classifiers, however it is also possible to construct quadratic or other non-linear decision boundaries (this is how some types of neural networks operate).

There are several methods of classifier construction for a linear discriminant function. The following schemes will be examined:

- Iterative solution for the weights via the perceptron algorithm which directly minimises the number of misclassifications
- Fisher linear discriminant which aims directly to maximise a measure of class separability

Single Layer Perceptron

A single layer perceptron is shown below. The typical form examined uses a threshold activation function:



The d-dimensional input vector $\boldsymbol{x}$ and scalar value z are related by

$$z = \boldsymbol{w}'\boldsymbol{x} + b$$

z is then fed to the activation function to yield $y(\boldsymbol{x})$. The parameters of this system are

- weights: $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$, selects the direction of the decision boundary
- bias: b, sets the position of the decision boundary.

These parameters are often combined into a single composite vector, $\tilde{\boldsymbol{w}}$, and the input vector extended, $\tilde{\boldsymbol{x}}$.

$$\tilde{\boldsymbol{w}} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \\ b \end{bmatrix}; \qquad \tilde{\boldsymbol{x}} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \\ 1 \end{bmatrix}$$

Single Layer Perceptron (cont)

We can then write

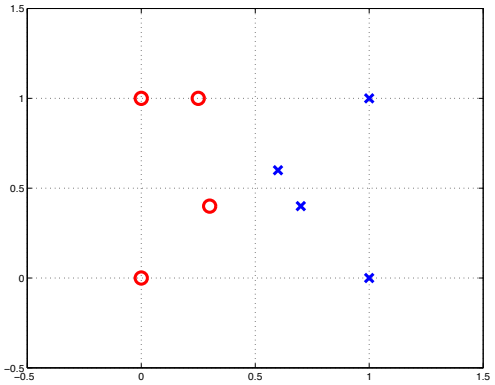
$$z = \tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}$$

The task is to train the set of model parameters $\tilde{\boldsymbol{w}}$. For this example a decision boundary is placed at $z = 0$. The decision rule is

$$y(\boldsymbol{x}) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

If the training data is linearly separable in the d-dimensional space then using an appropriate training algorithm perfect classification (on the training data at least!) can be achieved.

Is the solution unique?

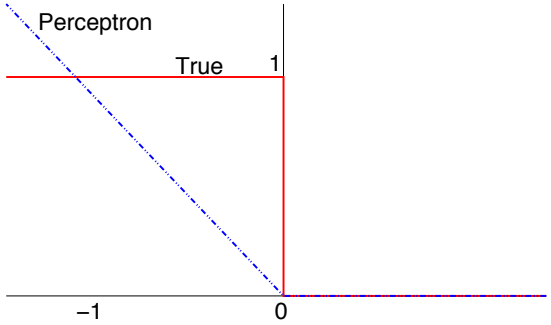


The precise solution selected depends on the training criterion/algorithm used.

Training Criteria

The decision boundary should be constructed to minimise the number of misclassified training examples. For each training example $\tilde{\boldsymbol{x}}_i$

$$\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i > 0 \quad \boldsymbol{x}_i \in \omega_1, \qquad \tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i < 0 \quad \boldsymbol{x}_i \in \omega_2$$



The above figure shows two criteria that could be used (the sample is from class ω_1)

- True: this is the actual cost a simple 1/0 (incorrect/correct). Let $y_i = -1$ for class ω_2, $y_i = 1$ class ω_1, this criterion can be expressed as (using previous decision rule)

  $$E(\tilde{\boldsymbol{w}}) = \frac{1}{2}\sum_{i=1}^{n} |y_i - y(\boldsymbol{x}_i)|$$

  This criterion is not useful if gradient descent based algorithms used (though meet again with SVM).

- Perceptron: an approximation to the true cost function, the loss is the distance that observation is from the decision boundary.

Perceptron Criterion

The distance from the decision boundary is given by $\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}$. However a problem is that we cannot simply sum the values of $\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}$ as a cost function since the sign depends on the class.

The perceptron cost-function can be written using the hinge-loss function defined as

$$[f(\boldsymbol{x})]_+ = \begin{cases} f(\boldsymbol{x}) & \text{if } f(\boldsymbol{x}) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Let the class label training example y_i again be specified as

$$y_i = \begin{cases} 1 & \text{if } \boldsymbol{x}_i \text{ belongs to class 1} \\ -1 & \text{if } \boldsymbol{x}_i \text{ belongs to class 2} \end{cases}$$

The perceptron criterion can be expressed as

$$E(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{n} \left[-y_i\left(\boldsymbol{w}'\boldsymbol{x}_i + b\right)\right]_+ = \sum_{i=1}^{n} \left[-y_i\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i\right]_+$$

To simplify the training it is possible to replace the training observations by a normalised form

$$\overline{\boldsymbol{x}}_i = y_i\tilde{\boldsymbol{x}}_i$$

This means that for a correctly classified symbol

$$\tilde{\boldsymbol{w}}'\overline{\boldsymbol{x}}_i > 0$$

and for mis-classified training examples

$$\tilde{\boldsymbol{w}}'\overline{\boldsymbol{x}}_i < 0$$

This process is sometimes referred to as normalising the data.

Gradient Descent

Finding the "best" parameters is an optimisation problem. First a cost function of the set of the weights must be defined, $E(\tilde{\boldsymbol{w}})$. Some learning process which minimises the cost function is then used. One basic procedure is gradient descent:

1. Start with some initial estimate $\tilde{\boldsymbol{w}}[0]$, $\tau = 0$.
2. Compute the gradient $\nabla E(\tilde{\boldsymbol{w}})|_{\tilde{\boldsymbol{w}}[\tau]}$
3. Update weight by moving a small distance in the steepest downhill direction, to give the estimate of the weights at iteration $\tau + 1$, $\tilde{\boldsymbol{w}}[\tau + 1]$,

$$\tilde{\boldsymbol{w}}[\tau + 1] = \tilde{\boldsymbol{w}}[\tau] - \eta \nabla E(\tilde{\boldsymbol{w}})|_{\tilde{\boldsymbol{w}}[\tau]}$$

   This can be written for just the i^{th} element

$$\tilde{w}_i[\tau + 1] = \tilde{w}_i[\tau] - \eta \left. \frac{\partial E(\tilde{\boldsymbol{w}})}{\partial \tilde{w}_i} \right|_{\tilde{\boldsymbol{w}}[\tau]}$$

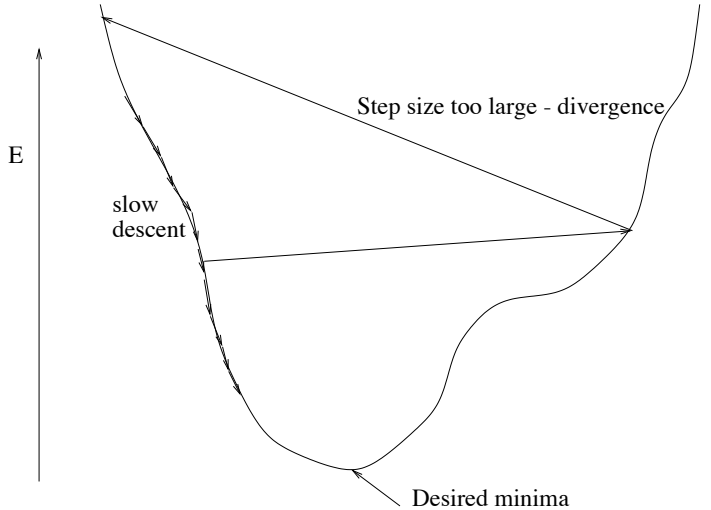
   Set $\tau = \tau + 1$
4. Repeat steps (2) and (3) until convergence, or the optimisation criterion is satisfied

One of the restrictions on using gradient descent is that the cost function $E(\tilde{\boldsymbol{w}})$ must be differentiable (and hence continuous). We will see that this means mis-classification cannot be used as the cost function for gradient descent schemes.

Gradient descent is not usually guaranteed to decrease the cost function (compare to EM).

Choice of η

In the previous slide the *learning rate* term η was used in the optimisation scheme.

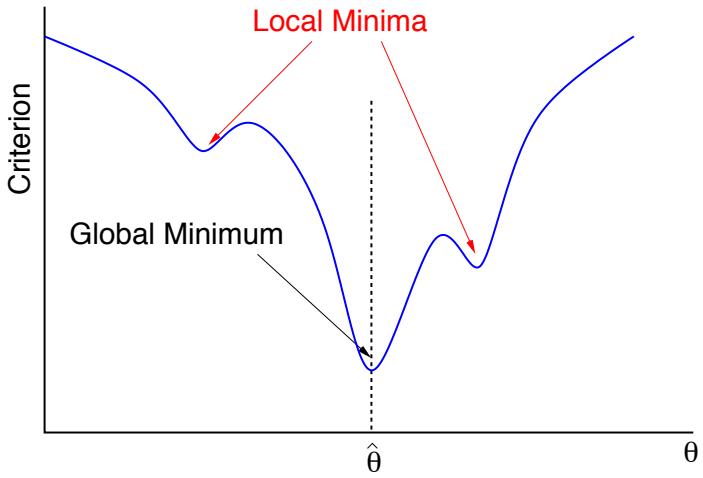


η is positive. When setting η we need to consider:

- if η is too small, convergence is slow;
- if η is too large, we may overshoot the solution and diverge.

Later in the course we will examine improvements for gradient descent schemes for highly complex schemes. Some of these give techniques for automatically setting η.

Local Minima



Another problem with gradient decsent is local minima. At all maxima and minima

$$\nabla E(\tilde{\boldsymbol{w}}) = 0$$

- gradient descent stops at all maxima/minima
- estimated parameters will depend on starting point
- single solution if function convex

Expectation-Maximisation is also only guaranteed to find a local maximum of the likelihood function for example.

Perceptron Algorithm

The perceptron criterion may be expressed as

$$E(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{n} \left[-y_i \tilde{\boldsymbol{w}}' \tilde{\boldsymbol{x}}_i\right]_+ = \sum_{\overline{\boldsymbol{x}}_i \in \mathcal{Y}} (-\tilde{\boldsymbol{w}}' \overline{\boldsymbol{x}}_i)$$

where $\mathcal{Y}$ is the set of mis-classified points. We now want to minimise the perceptron criterion using gradient descent. It is simple to show that

$$\nabla E(\tilde{\boldsymbol{w}}) = \sum_{\overline{\boldsymbol{x}}_i \in \mathcal{Y}} (-\overline{\boldsymbol{x}}_i)$$

Hence the gradient descent update rule is

$$\tilde{\boldsymbol{w}}[\tau + 1] = \tilde{\boldsymbol{w}}[\tau] + \eta \sum_{\overline{\boldsymbol{x}}_i \in \mathcal{Y}[\tau]} \overline{\boldsymbol{x}}_i$$

where $\mathcal{Y}[\tau]$ is the set of mis-classified points using $\tilde{\boldsymbol{w}}[\tau]$. For the case when the samples are linearly separable using a value of $\eta = 1$ is guaranteed to converge. The basic algorithm is:

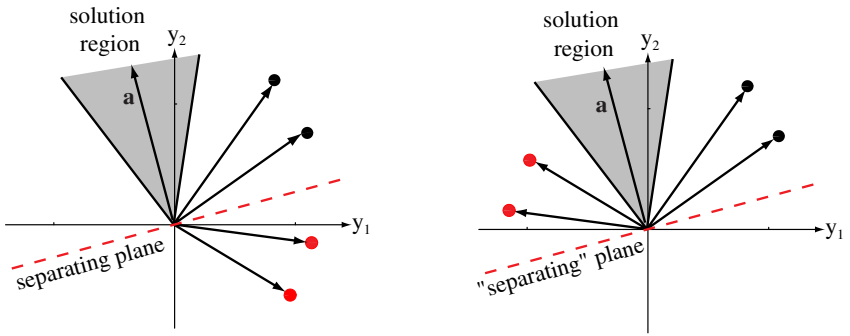
1. Set $\overline{\boldsymbol{x}}_i = y_i \tilde{\boldsymbol{x}}_i$ ($y_i = -1$ for class ω_2, $y_i = 1$ class ω_1).
2. Initialise the weight vector $\tilde{\boldsymbol{w}}[0]$, $\tau = 0$.
3. Using $\tilde{\boldsymbol{w}}[\tau]$ produce the set of mis-classified samples $\mathcal{Y}[\tau]$.
4. Use update rule

   $$\tilde{\boldsymbol{w}}[\tau + 1] = \tilde{\boldsymbol{w}}[\tau] + \sum_{\overline{\boldsymbol{x}}_i \in \mathcal{Y}[\tau]} \overline{\boldsymbol{x}}_i$$

   then set $\tau = \tau + 1$.
5. Repeat steps (3) and (4) until the convergence criterion is satisfied.

Perceptron Solution Region

- Each sample $\tilde{\boldsymbol{x}}_i$ places a constraint on the possible location of a solution vector that classifies all samples correctly.
- $\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i = 0$ defines a hyperplane through the origin on the "weight-space" of $\tilde{\boldsymbol{w}}$ vectors with $\tilde{\boldsymbol{x}}_i$ as a normal vector.
- For normalised data, the solution vector must be on the positive side of every such hyperplane.
- The solution vector, if it exists, is not unique and lies anywhere within the solution region
- Figure below (from DHS) shows 4 training points and the solution region for both un-normalised data and normalised data (note notation swap $\tilde{\boldsymbol{w}} = \boldsymbol{a}, \tilde{\boldsymbol{x}} = \boldsymbol{y}$).



A Simple Example

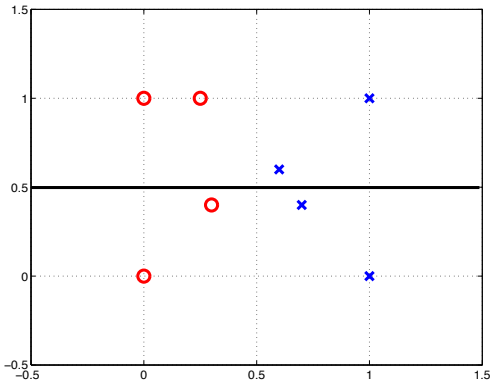
Consider a simple example:

Class 1 has points $\begin{bmatrix}1\\0\end{bmatrix}, \begin{bmatrix}1\\1\end{bmatrix}, \begin{bmatrix}0.6\\0.6\end{bmatrix}, \begin{bmatrix}0.7\\0.4\end{bmatrix}$

Class 2 has points $\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}0.25\\1\end{bmatrix}, \begin{bmatrix}0.3\\0.4\end{bmatrix}$

Initial estimate: $\tilde{\boldsymbol{w}}[0] = \begin{bmatrix}0\\1\\-0.5\end{bmatrix}$

This yields the following initial estimate of the decision boundary.



Given this initial estimate we need to train the decision boundary.

Simple Example (cont)

The first part of the algorithm is to use the current decision boundary to obtain the set of mis-classified points.

For the data from Class ω_1

Class ω_1	1	2	3	4
z	-0.5	0.5	0.1	-0.1
Class	2	1	1	2

and for Class ω_2

Class ω_2	1	2	3	4
z	-0.5	0.5	0.5	-0.1
Class	2	1	1	2

The set of mis-classified points, $\mathcal{Y}[0]$, is

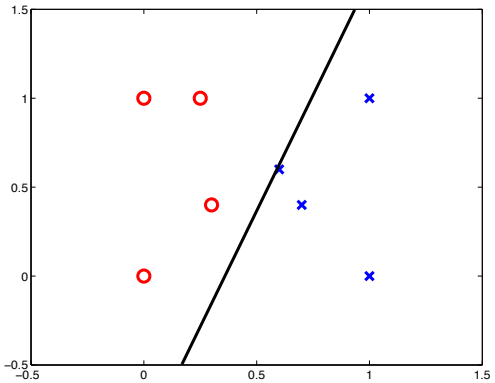
$$\mathcal{Y}[0] = \{1_{\omega_1}, 4_{\omega_1}, 2_{\omega_2}, 3_{\omega_2}\}$$

From the perceptron update rule this yields the updated vector

$$\tilde{\boldsymbol{w}}[1] = \tilde{\boldsymbol{w}}[0] + \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.7 \\ 0.4 \\ 1 \end{bmatrix} + \begin{bmatrix} 0 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} -0.25 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1.45 \\ -0.6 \\ -0.5 \end{bmatrix}$$

Simple Example (cont)

This yields the following decision boundary



Again applying the decision rule to the data we get for class ω_1

Class 1	1	2	3	4
z	0.95	0.35	0.01	0.275
Class	1	1	1	1

and for class ω_2

Class 2	1	2	3	4
z	-0.5	-1.1	-0.7375	-0.305
Class	2	2	2	2

All points correctly classified the algorithm has converged.

Is this a good decision boundary?

Logistic Regression/Classification

Interesting to contrast the perceptron algorithm with logistic regression/classification. This is a linear classifier, but a discriminative model not a discriminant function. The training criterion is

$$\mathcal{L}(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{n} \left[y_i \log \left(\frac{1}{1+\exp(-\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i)} \right) + (1-y_i) \log \left(\frac{1}{1+\exp(\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i)} \right) \right]$$

where (note change of label definition)

$$y_i = \begin{cases} 1, & \boldsymbol{x}_i \text{ generated by class } \omega_1 \\ 0, & \boldsymbol{x}_i \text{ generated by class } \omega_2 \end{cases}$$

It is simple to show that (has a nice intuitive feel)

$$\nabla \mathcal{L}(\tilde{\boldsymbol{w}}) = \sum_{i=1}^{n} \left(y_i - \frac{1}{1+\exp(-\tilde{\boldsymbol{w}}'\tilde{\boldsymbol{x}}_i)} \right) \tilde{\boldsymbol{x}}_i$$

Maximising the log-likelihood is equal to minimising the negative log-likelihood, $E(\tilde{\boldsymbol{w}}) = -\mathcal{L}(\tilde{\boldsymbol{w}})$. Update rule then becomes

$$\tilde{\boldsymbol{w}}[\tau+1] = \tilde{\boldsymbol{w}}[\tau] + \eta \nabla \mathcal{L}(\tilde{\boldsymbol{w}})|_{\tilde{\boldsymbol{w}}[\tau]}$$

Compared to the perceptron algorithm:

- need an appropriate method to select η;
- does not necessarily correctly classify training data even if linearly separable;
- not necessary for data to be linearly separable;
- yields a class posterior (use Bayes' decision rule to get hypothesis).

Fisher's Discriminant Analysis

A different approach to training the parameters of the perceptron is to use Fisher's discriminant analysis. The basic aim here is to choose the projection that maximises the distance between the class means, whilst minimising the within class variance. Note only the projection $\boldsymbol{w}$ is determined. The following cost function is used

$$E(\boldsymbol{w}) = -\frac{(\overline{\mu}_1 - \overline{\mu}_2)^2}{\overline{s}_1 + \overline{s}_2}$$

where $\overline{s}_j$ and $\overline{\mu}_j$ are the projected scatter matrix and mean for class ω_j. The projected scatter matrix is defined as

$$\overline{s}_j = \sum_{\overline{x}_i \in \omega_j} (\overline{x}_i - \overline{\mu}_j)^2$$

The cost function may be expressed as

$$E(\boldsymbol{w}) = -\frac{\boldsymbol{w}'\mathbf{S}_{\text{b}}\boldsymbol{w}}{\boldsymbol{w}'\mathbf{S}_{\text{w}}\boldsymbol{w}}$$

where

$$\mathbf{S}_{\text{b}} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'$$

and

$$\mathbf{S}_{\text{w}} = \mathbf{S}_1 + \mathbf{S}_2$$

where

$$\mathbf{S}_j = \sum_{\boldsymbol{x}_i \in \omega_j} (\boldsymbol{x}_i - \boldsymbol{\mu}_j)(\boldsymbol{x}_i - \boldsymbol{\mu}_j)'$$

the mean of the class $\boldsymbol{\mu}_j$ is defined as usual.

Fisher's Discriminant Analysis (cont)

Differerentiating $E(\boldsymbol{w})$ with respect to the weights, we find that it is minimised when

$$(\hat{\boldsymbol{w}}'\mathbf{S}_{\text{b}}\hat{\boldsymbol{w}})\mathbf{S}_{\text{w}}\hat{\boldsymbol{w}} = (\hat{\boldsymbol{w}}'\mathbf{S}_{\text{w}}\hat{\boldsymbol{w}})\mathbf{S}_{\text{b}}\hat{\boldsymbol{w}}$$

From the definition of $\mathbf{S}_{\text{b}}$

$$\begin{aligned}\mathbf{S}_{\text{b}}\hat{\boldsymbol{w}} &= (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'\hat{\boldsymbol{w}} \\ &= ((\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'\hat{\boldsymbol{w}})\,(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)\end{aligned}$$

We therefore know that

$$\mathbf{S}_{\text{w}}\hat{\boldsymbol{w}} \propto (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

Multiplying both sides by $\mathbf{S}_{\text{w}}^{-1}$ yields

$$\hat{\boldsymbol{w}} \propto \mathbf{S}_{\text{w}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

This has given the direction of the decision boundary. However we still need the bias value b.

If the data is separable using the Fisher's discriminant it makes sense to select the value of b that maximises the margin. Simply put this means that, given no additional information, the boundary should be equidistant from the two points either side of the decision boundary.

Example

Using the previously described data

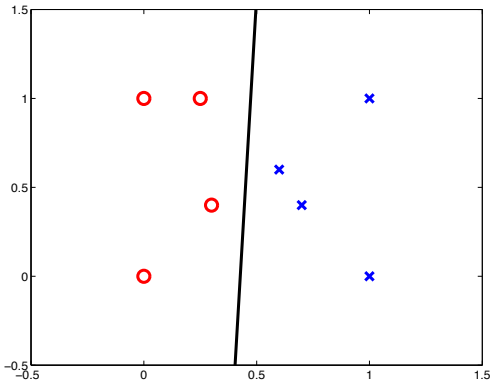
$$\boldsymbol{\mu}_1 = \begin{bmatrix} 0.825 \\ 0.500 \end{bmatrix}; \quad \boldsymbol{\mu}_2 = \begin{bmatrix} 0.1375 \\ 0.600 \end{bmatrix}$$

and

$$\mathbf{S}_{\text{w}} = \begin{bmatrix} 0.2044 & 0.0300 \\ 0.0300 & 1.2400 \end{bmatrix}$$

So solving this yields, for the direction, and taking the midpoint between the observations closest to the decision boundary

$$\hat{\boldsymbol{w}} = \begin{bmatrix} 3.3878 \\ -0.1626 \end{bmatrix}, \quad \tilde{\boldsymbol{w}}[1] = \begin{bmatrix} 3.3878 \\ -0.1626 \\ -1.4432 \end{bmatrix}$$



Kesler's Construct

So far we have only examined binary classifiers. The direct use of multiple binary classifiers can results in no-decision regions (see examples paper).

The multi-class problem can be converted to a 2-class problem. Consider an extended observation $\tilde{\boldsymbol{x}}$ which belongs to class ω_1. Then to be correctly classified

$$\tilde{\boldsymbol{w}}_1'\tilde{\boldsymbol{x}} - \tilde{\boldsymbol{w}}_j'\tilde{\boldsymbol{x}} > 0, \quad j = 2, \ldots, K$$

There are therefore $K-1$ inequalities requiring that the $K(d+1)$-dimensional vector

$$\boldsymbol{\alpha} = \begin{bmatrix} \tilde{\boldsymbol{w}}_1 \\ \vdots \\ \tilde{\boldsymbol{w}}_K \end{bmatrix}$$

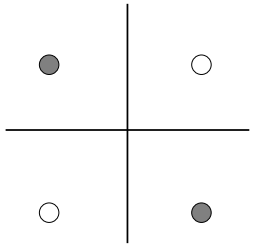
correctly classifies all $K-1$ set of $K(d+1)$-dimensional samples

$$\boldsymbol{\gamma}_{12} = \begin{bmatrix} \tilde{\boldsymbol{x}} \\ -\tilde{\boldsymbol{x}} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \boldsymbol{\gamma}_{13} = \begin{bmatrix} \tilde{\boldsymbol{x}} \\ \mathbf{0} \\ -\tilde{\boldsymbol{x}} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \ldots, \quad \boldsymbol{\gamma}_{1K} = \begin{bmatrix} \tilde{\boldsymbol{x}} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ -\tilde{\boldsymbol{x}} \end{bmatrix}$$

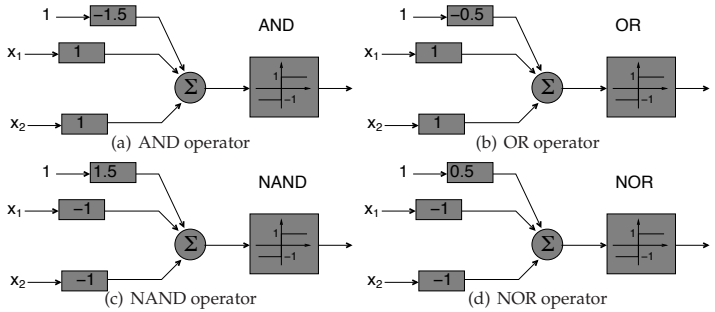
The multi-class problem has now been transformed to a two-class problem at the expense of increasing the effective dimensionality of the data and increasing the number of training samples. We now simply optimise for $\boldsymbol{\alpha}$.

Limitations of Linear Decision Boundaries

Perceptrons were very popular until the 1960's when it was realised that it couldn't solve the XOR problem.

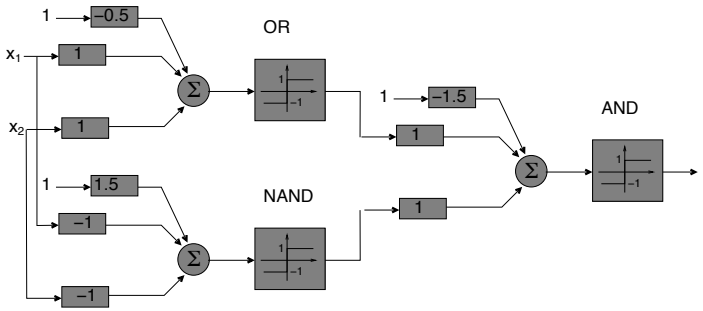


We can use perceptrons to solve the binary logic operators AND, OR, NAND, NOR.

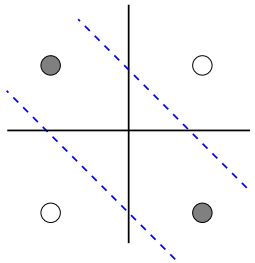


XOR (cont)

But XOR may be written in terms of AND, NAND and OR gates



This yields the decision boundaries



So XOR can be solved using a two-layer network. The problem is how to train multi-layer perceptrons. In the 1980's an algorithm for training such networks was proposed, error back propagation.

Summary

This lecture has examined:

- single layer perceptrons (SLPs)
- perceptron algorithym
- gradient descent
- Fisher's discriminant analysis

The next few lectures will extend these concepts:

- multi-layer perceptons (MLPs)
- training MLPs
- improved gradient descent
- support vector machines (SVMs)
- maximum margin training for linear classifiers
- use of kernels for non-linear decision boundaries.