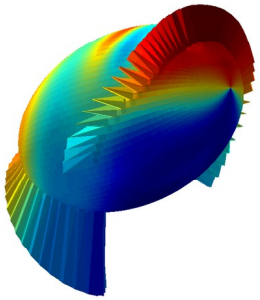
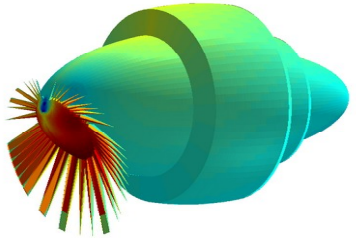
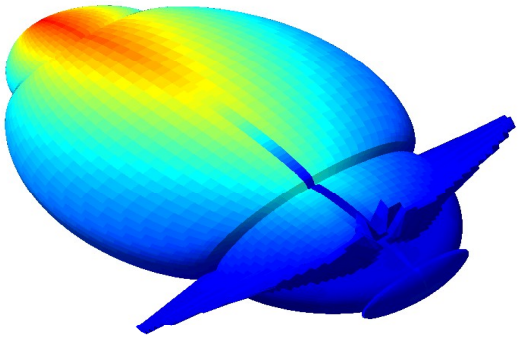
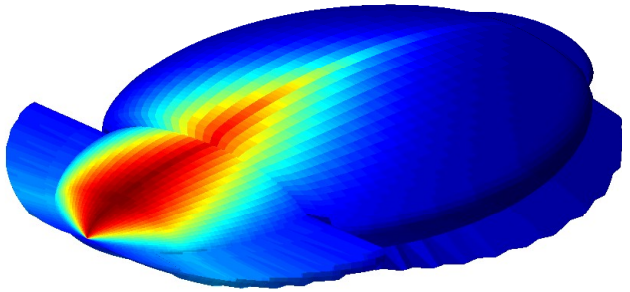




Lecture 4
Classification





Problem Description

Inputs: $x^k \in \mathbf{X}$ (feature space)

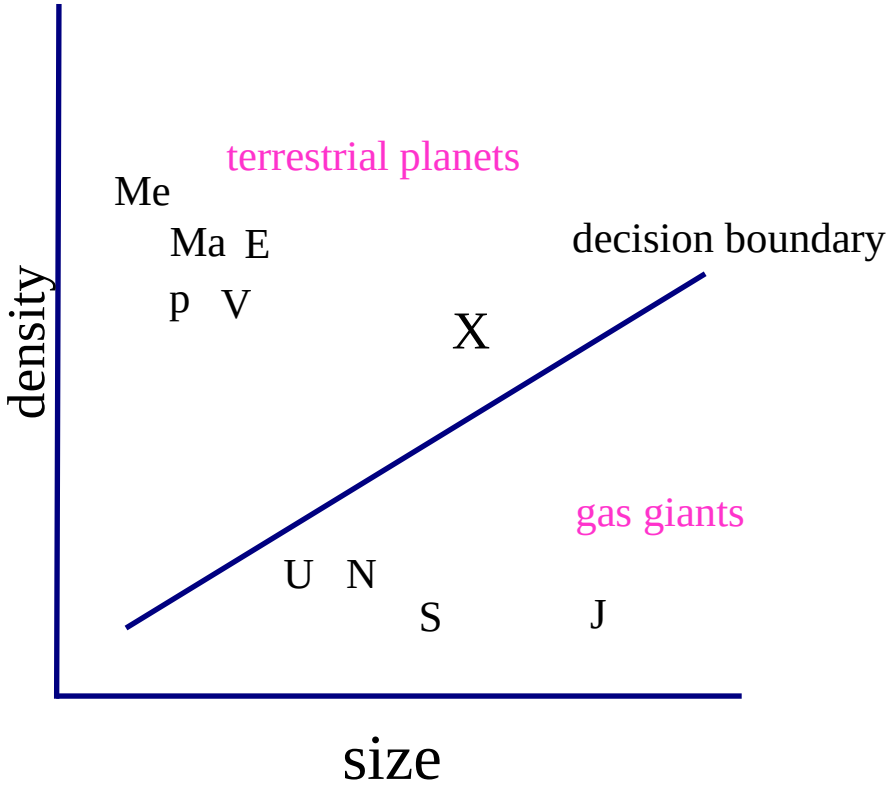
Labels: $C_i \in \mathbf{Y}$ (classes)

Classifier

e.g. x^k are planets $\boldsymbol{H}: \mathbf{X} \rightarrow \mathbf{Y}$

= [size , density]

$\mathbf{X}$



The task for the classifier is to determine the optimal decision boundary

Problem Description

Inputs: $x^k \in \mathbf{X}$ (feature space)

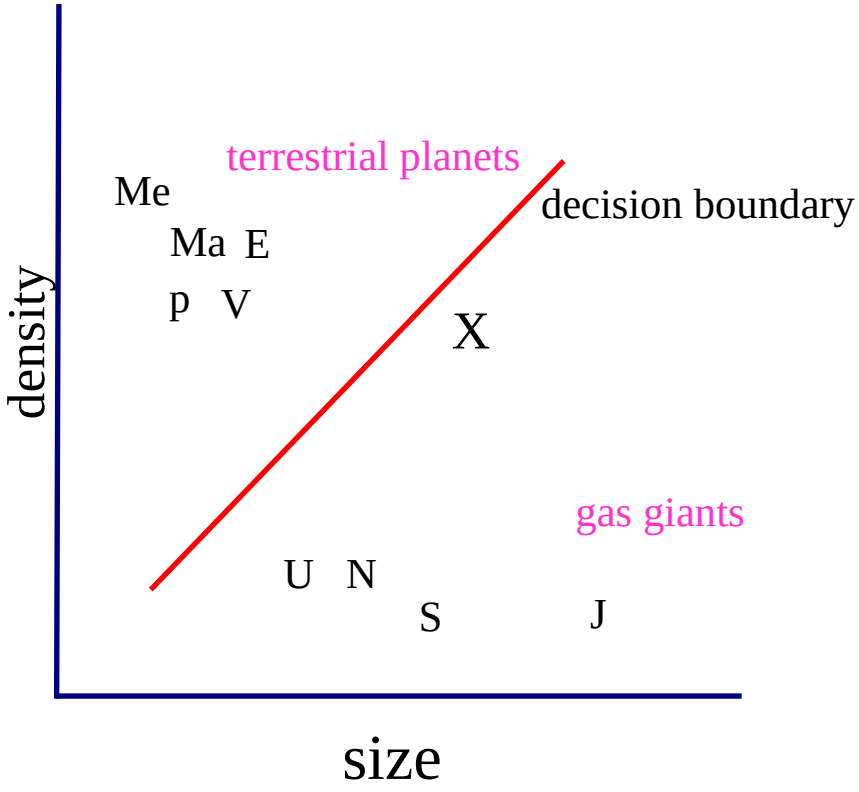
Labels: $C_i \in \mathbf{Y}$ (classes)

Classifier

e.g. x^k are planets $\boldsymbol{H}: \mathbf{X} \rightarrow \mathbf{Y}$

= [size , density]

$\mathbf{X}$

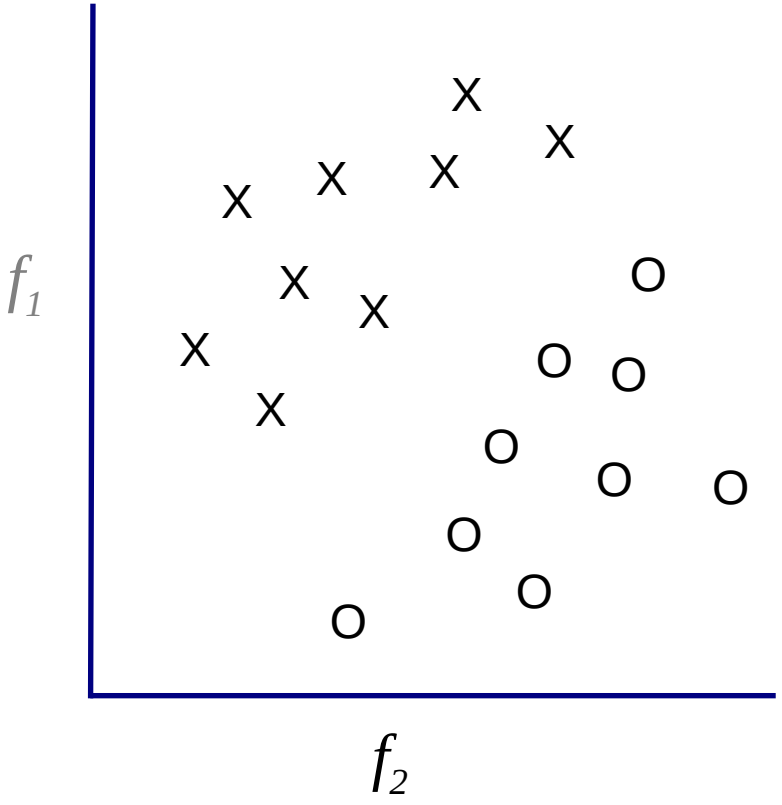


The task for the classifier is to determine the optimal decision boundary

Easy and Difficult Classification Problems

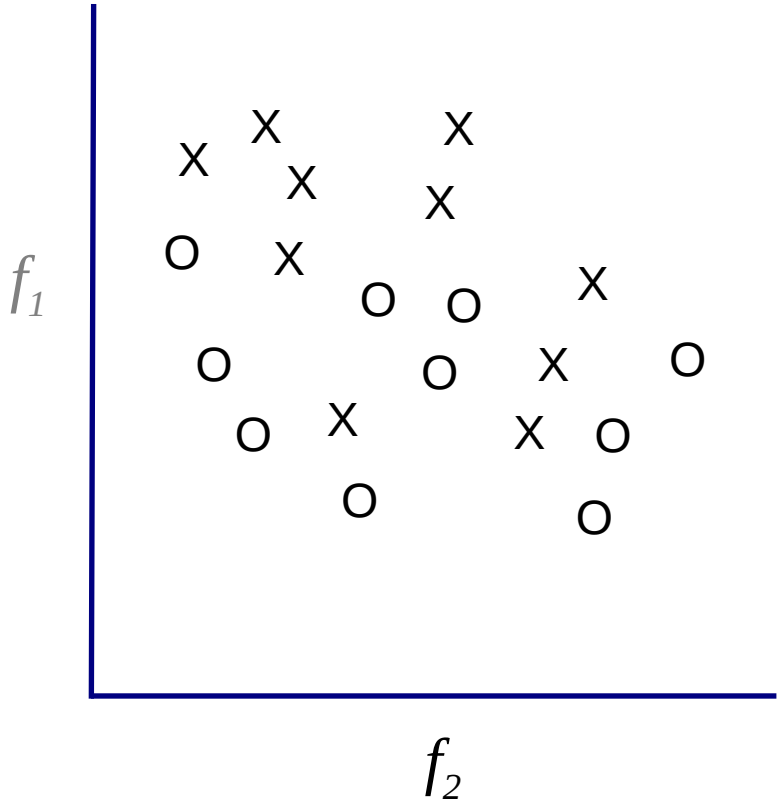
Case I

Classes are easy to separate



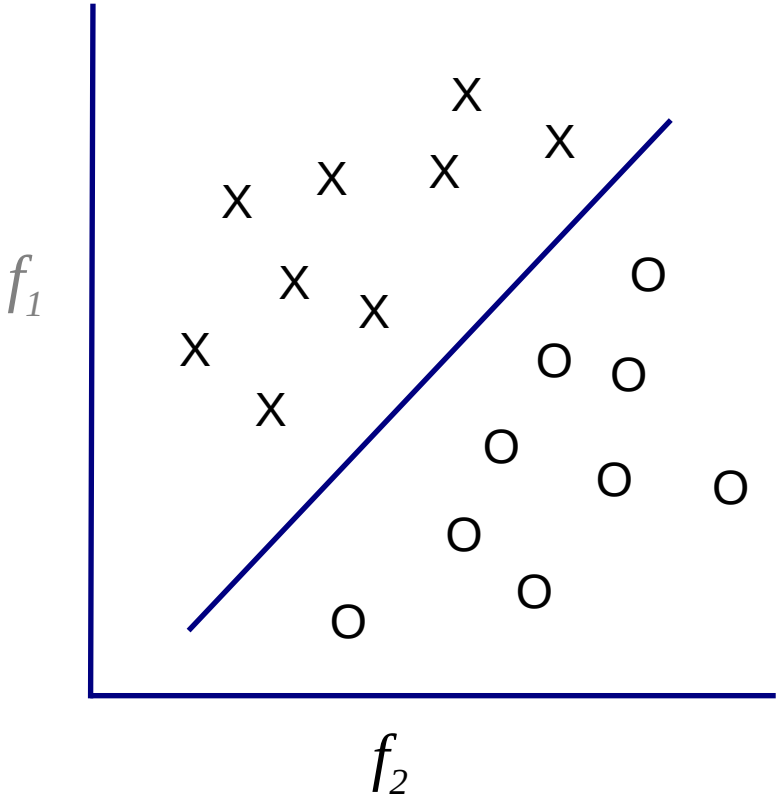
Case II

Classes are hard to separate

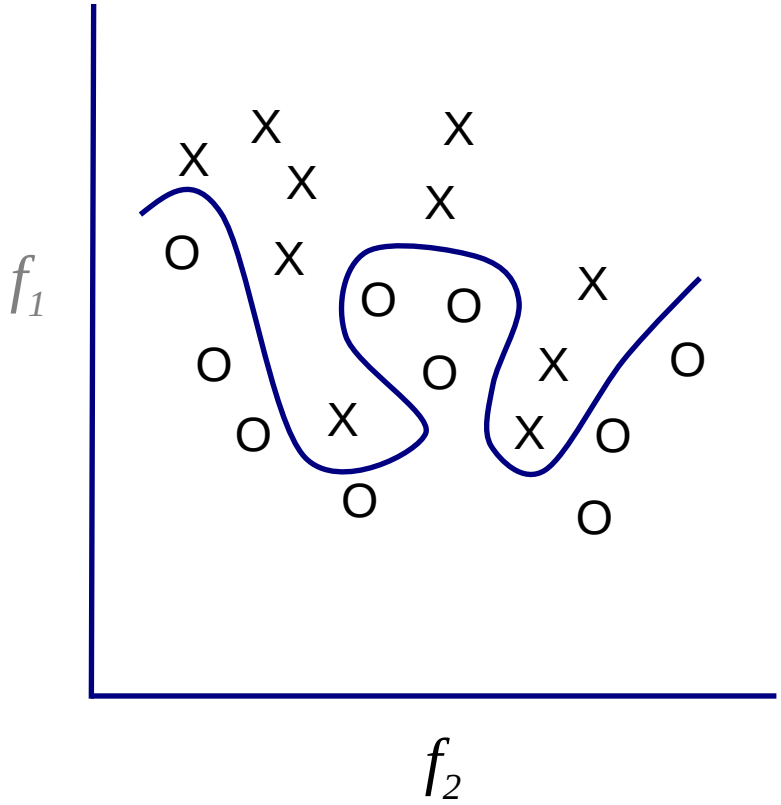


Easy and Difficult Classification Problems

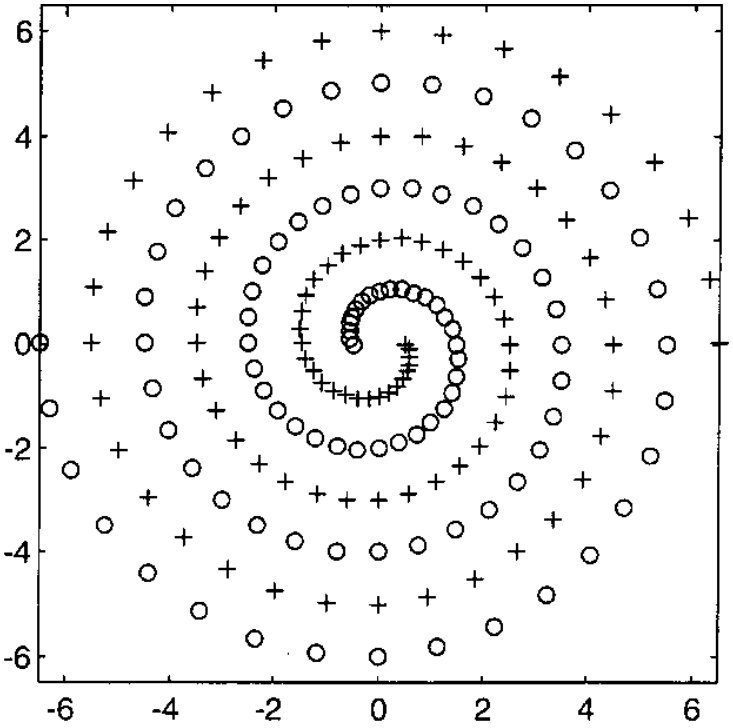
Linearly separable classes: The classes can be separated by a hyperplane in feature space.

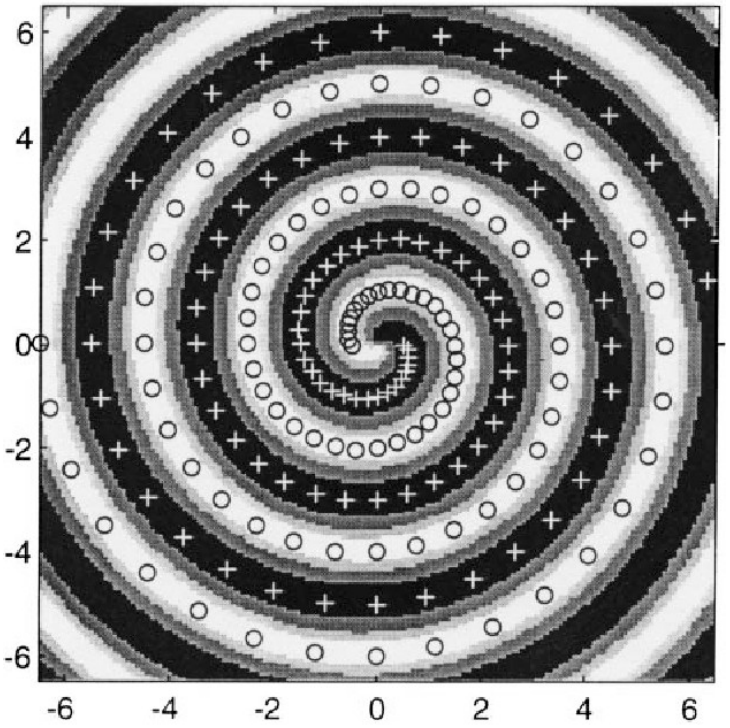


Linearly inseparable classes



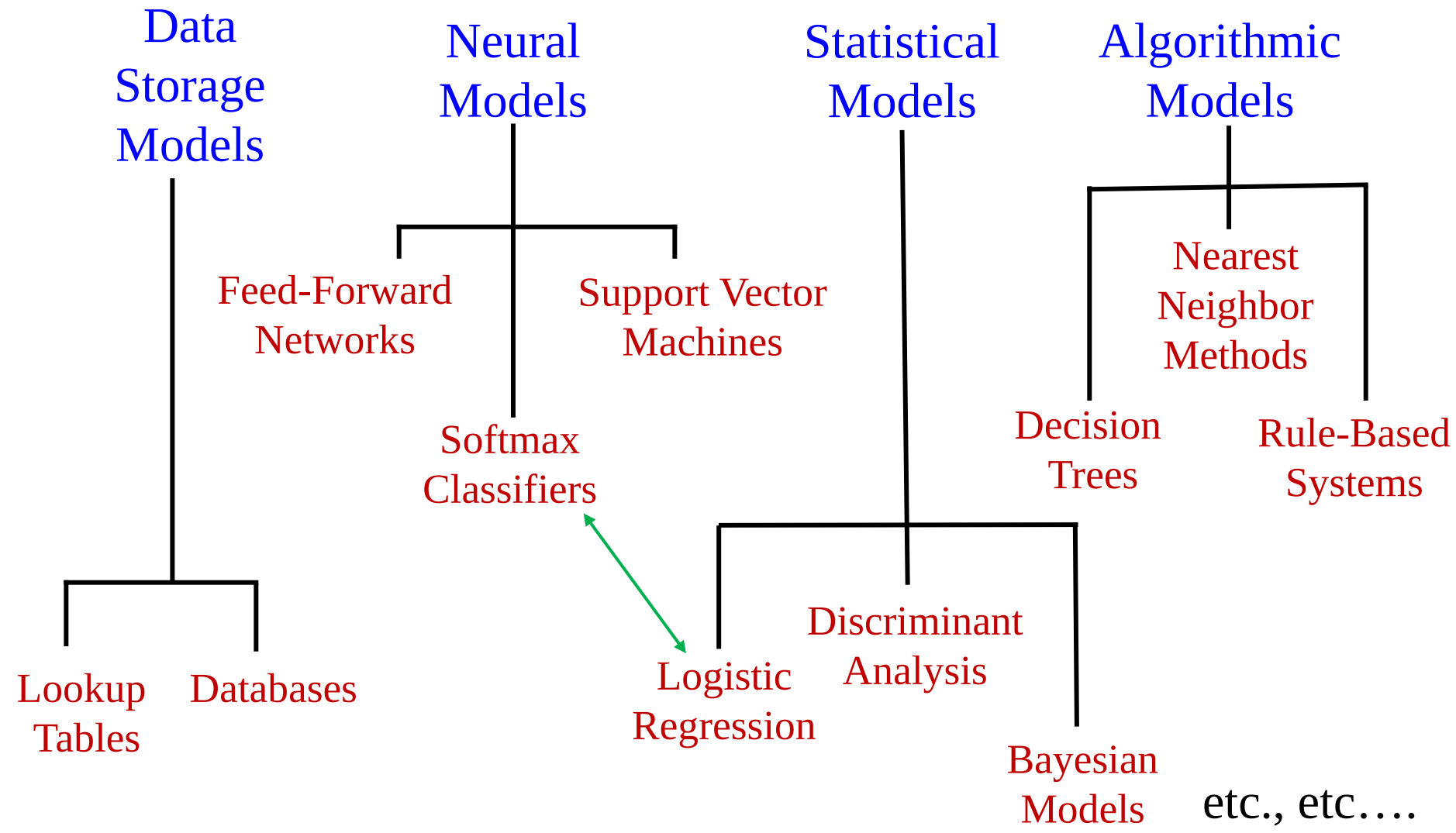
A Very Difficult Classification Problem: Two Spirals



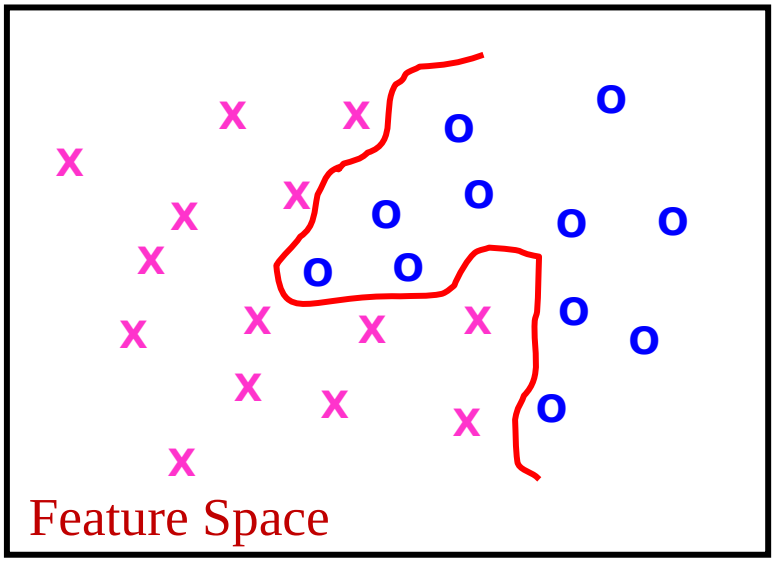


José R. Álvarez (1999) Injecting Knowledge into the Solution of th
Two-Spiral Problem, *Neural Computing & Appl*ications
DOI:10.1007/s005210050029

Selecting a Classifier



Finding Class Boundaries



o = A
x = B
Training Set
= class boundary

Feature space is usually a BIG place

→ The system can *never* see all exemplars of a class

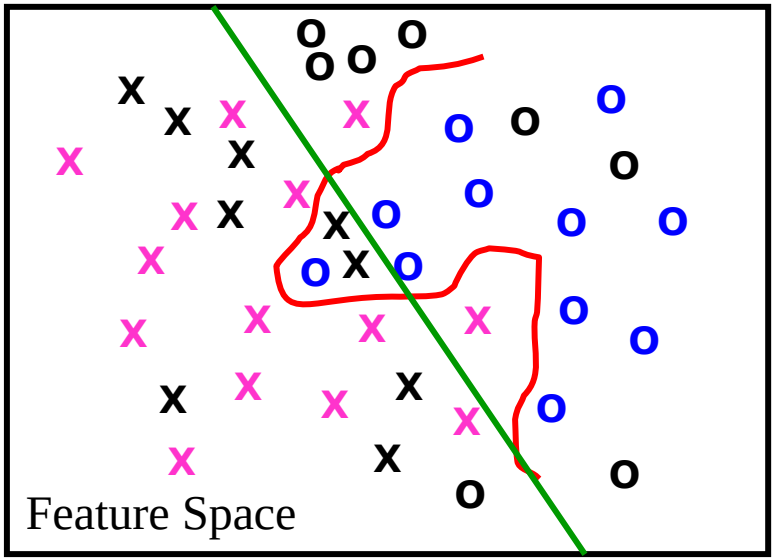
→ It must **generalize** from a limited sample

The curse of dimensionality:

The number of examplars needed to determine class boundaries increases exponentially with data space dimension.

→ Reduce the dimension of data space

Validating Performance



o = A
x = B
} Training Set
o x = Test Set
= complicated class boundary
= better, simpler boundary

Fitting training data too well can be a liability

→ Go for simple inductions (**Occam's Razor**)

→ Mistakes are often the price of intelligence

Intelligence requires making good choices, not perfection.

→ Know how to tell good choices from bad
(common sense, intuition, experience, wisdom)

Assessing the Quality of Classification

Assessing the performance of a classifier must account for two important factors:

- **Generalization** to novel data.
- The effect of class sizes.

Generalization is checked by validation and testing

The set of labeled data is divided into three sets:

- **Training Set:** Used to train the classifier.
- **Validation Set:** Used to check generalization ***during*** learning.
- **Test Set:** Used to check generalization ***after*** learning.

Typically, the Validation and Test Sets are smaller than the Training Set, especially if data is scarce.

More complex methods such as **cross-validation** are used in many cases.

Binary Classification Performance Metrics

Given: 1) Labeled data from two classes 0 (NO) and 1 (YES)
2) 0/1 labels assigned by a classifier to each data point

Define:

Q_{00} = Number of correct 0 outputs – true negatives

Q_{11} = Number of correct 1 outputs – true positives

Q_{01} = Number of 1 outputs that should have been 0 – false positives

Q_{10} = Number of 0 outputs that should have been 1 – false negatives

These four quantities can be used to define performance metrics for the classifier.

Sensitivity (Recall, True Positive Rate): Fraction of positives identified.

$$\textbf{Sensitivity} = \frac{\text{true positives}}{\text{all positives in data}} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} = \frac{Q_{11}}{Q_{11} + Q_{10}}$$

Specificity (Selectivity, True Negative Rate): Fraction of negatives identified.

$$\textbf{Specificity} = \frac{\text{true negatives}}{\text{all negatives in data}} = \frac{\text{true negatives}}{\text{false positives} + \text{true negatives}} = \frac{Q_{00}}{Q_{01} + Q_{00}}$$

Positive Predictive Value (Precision):

Fraction of identified positives that are correct (how accurate are the positives?).

$$\textbf{PPV} = \frac{\text{true positives}}{\text{all classified as positive}} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} = \frac{Q_{11}}{Q_{11} + Q_{01}}$$

Negative Predictive Value:

Fraction of identified negatives that are correct.

$$\textbf{NPV} = \frac{\text{true negatives}}{\text{all classified as negative}} = \frac{\text{true negatives}}{\text{true negatives} + \text{false negatives}} = \frac{Q_{00}}{Q_{00} + Q_{10}}$$

Accuracy

Accuracy: The fraction of data that is correctly classified.

$$\textbf{Accuracy} = \frac{\text{true positives} \ + \ \text{true negatives}}{\text{all data}} = \frac{Q_{11} + Q_{00}}{Q_{11} + Q_{10} + Q_{01} + Q_{00}}$$

The accuracy can be misleading when the data is unbalanced, i.e., there are many more positives than negatives, or many more negatives than positives.

In that case, we can use **balanced accuracy**:

$$\textbf{Balanced Accuracy} = \frac{\text{sensitivity} \ + \ \text{specificity}}{2}$$

F_1 Score: Measures both precision and recall.

$$\mathbf{F}_1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}}$$

F_β Score: Weights precision higher or lower than recall.

$$\mathbf{F}_\beta = (1+\beta^2) \times \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}} = \frac{(1+\beta^2)}{\frac{1}{(\beta^2 \times \text{precision})} + \frac{1}{\text{recall}}}$$

- High precision indicates absence of false positives (negative seen as positive).
- High recall indicates absence of false negatives (positive seen as negative).

If precision is more important, i.e., avoiding negatives is more important than missing positives, use $\beta < 1$
e.g., investing in a startup, accepting a job offer.

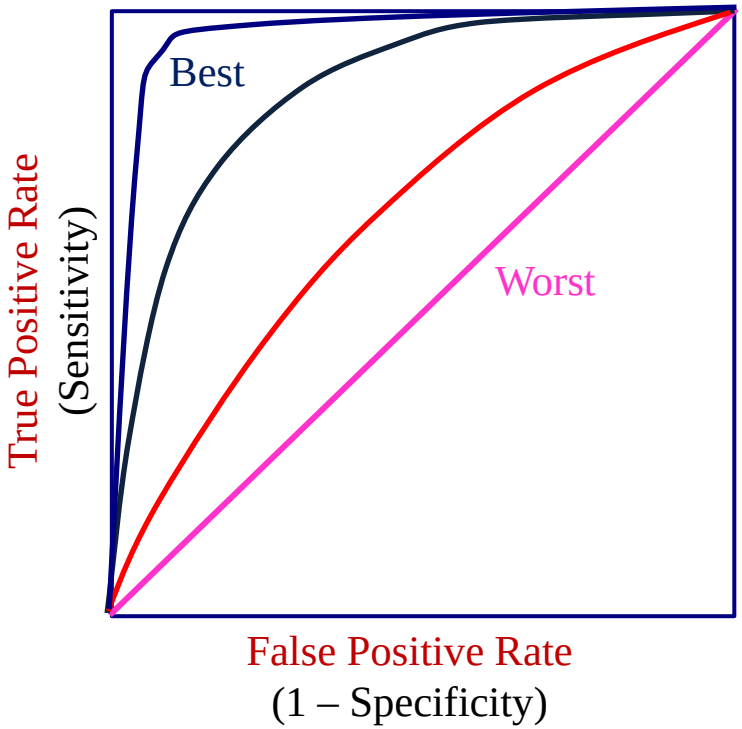
If recall is more important, i.e., identifying positives is more important than avoiding negatives, use $\beta > 1$
e.g., deciding whether a person is critically ill.

The ROC Curve

Receiver Operating Characteristic Curve

In many binary classification problems, the classifier depends on a parameter θ, which determines whether datapoint x is classified as C_1 or C_0. For example:

$x \in C_1$ if $g(x) > \theta$ else $x \in C_0$



As θ changes:

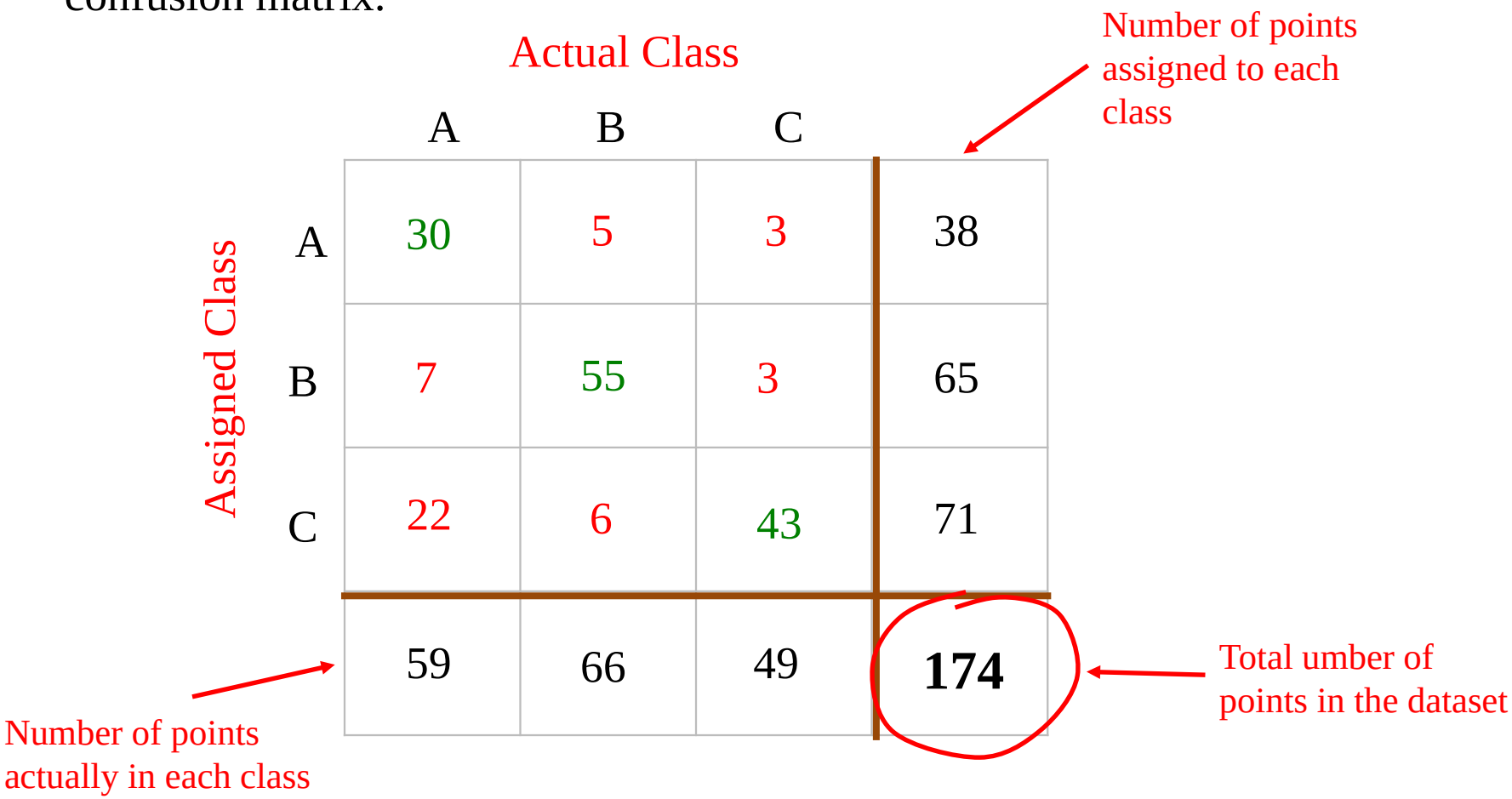
- If we pick up more true positives
- We also pick up more false positives

A good classifier should pick up the most true positives at the cost of the fewest false positives.

- The **area under the ROC curve (AUC)** is a measure of classifier quality.
- Optimal θ is where the ROC slope becomes less than 1.

The Confusion Matrix

For multi-class problems, it is important to consider how much of data from each class is classified into each of the classes, which is represented by the confusion matrix.



Meaningful Validation

It is possible that the system may do well on both the training and testing sets, but still not be a good model. Why????

Because the test set – which is chosen randomly – may be "easy" by accident.

How can we tell that "successful" learning is actually valid?

n-Fold Cross-Validation

- Do *n* learning trials, choosing the training and test sets randomly each time.
- Get the validation performance for each trial.
- If the validation results on all (or almost all) trials are good and consistent:
  The results on any of the good trials are likely to be valid → Keep the best/all.
- If the validation results are good on very few trials and bad on most:
  The good trials are likely flukes → This probably isn't a good model for this task.

Should we expect perfect classification?

No, because we will never have complete information.

Is this imperfection a bad thing?

Of course, we would like to be correct as often as possible, but the ability to make "sufficiently correct" choices based on incomplete information is far more important.

Why?

Because it allows useful choices to be made in *real time* and with limited data. In a complex environment, waiting for sufficient information to make the perfect choice is unaffordable.

- Intelligence does not require optimal choices, just ones that are "sufficiently good" – ***Satisficing*** (Simon)
- The *willingness* to make errors is a "feature", not a "bug" for an intelligent system trying to survive in a complex environment.
- The *preference* not to make an error is also a feature.

Decision Theoretic Methods

Let x be an input pattern to be classified into one of m classes, C_1 ..., C_m

Define $g_1(x),\ g_2(x), \ldots,\ g_m(x)$

such that

$$x \in C_i \Rightarrow g_i(x) > g_j(x)$$

$$\forall\ j = 1, 2, ..., m$$

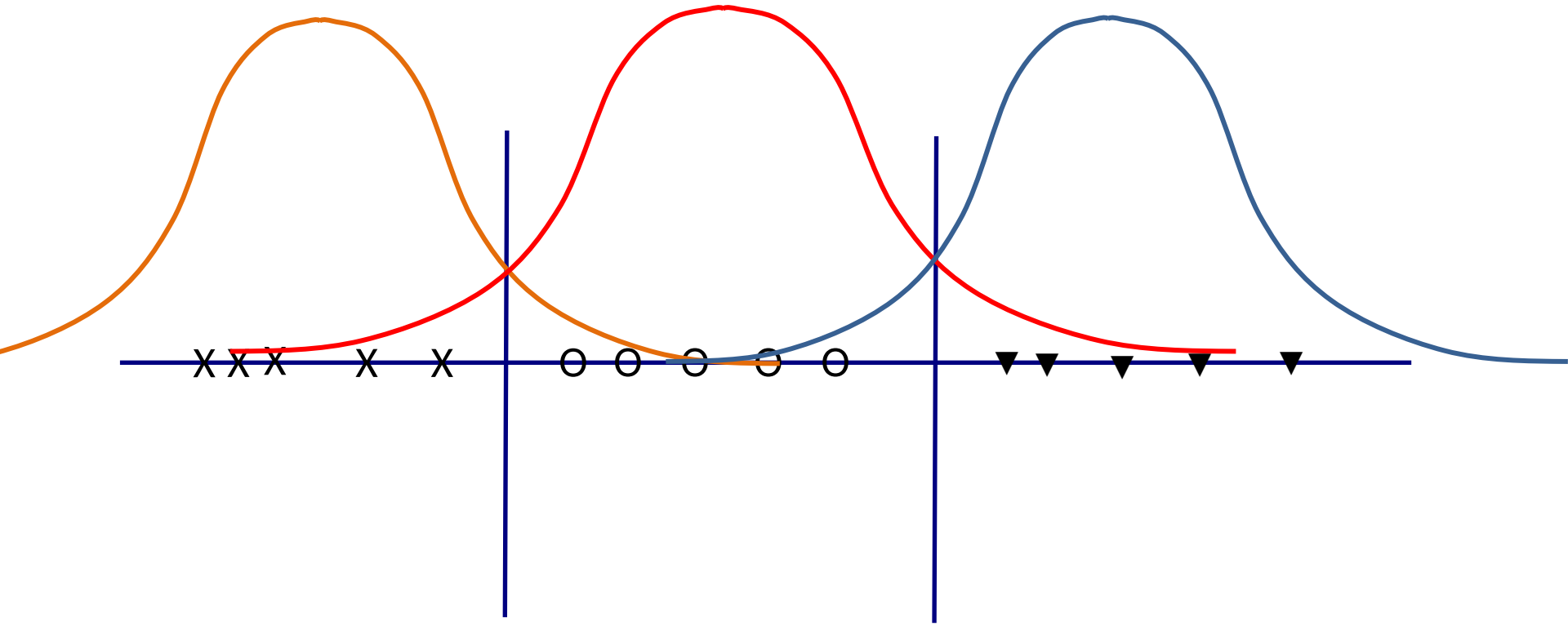
$$j \neq i$$

$$d_{ij} \equiv \{x : g_i(x) - g_j(x) = 0\}$$

d_{ij} is the decision boundary between C_i and C_j

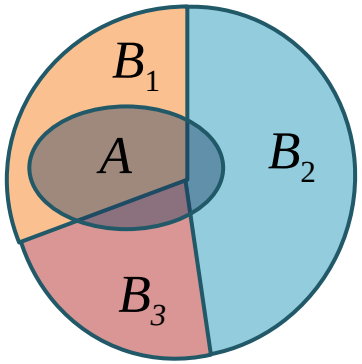
The $g_i(x)$ are called ***decision functions*** or ***discriminant functions***

Example: Minimum distance classifier



Each function is higher than all others for data within its class

Bayes Rule



If $S = \bigcup_{k=1}^{n} B_k$ and $A \subset S$

B_k form a partition of S if:

$$B_i \cap B_j = \Phi \qquad \forall i \neq j$$

$$P(B_j \mid A) = \frac{P(A \cap B_j)}{P(A)} = \frac{P(A \mid B_j)\,P(B_j)}{\sum_{k=1}^{n} P(A \mid B_k)\,P(B_k)}$$

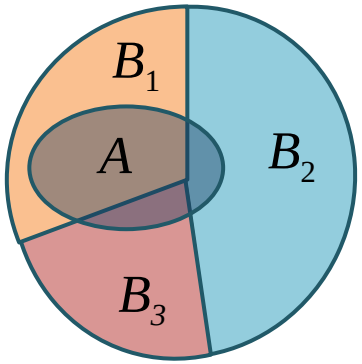
Note that this cannot be more than this

The form

$$P(B_j \mid A) = \frac{P(A \mid B_j)\,P(B_j)}{P(A)}$$

is widely used in estimation

Bayes Rule



If $S = \bigcup_{k=1}^{n} B_k$ and $A \subset S$

B_k form a partition of S if:

$$B_i \cap B_j = \Phi \qquad \forall i \neq j$$

$$P(B_j \mid A) = \frac{P(A \cap B_j)}{P(A)} = \frac{P(A \mid B_j)\,P(B_j)}{\sum_{k=1}^{n} P(A \mid B_k)\,P(B_k)}$$

Note that this cannot be more than this

The form

Posterior $P(B_j \mid A)$ = Likelihood $P(A \mid B_j)$ Prior $P(B_j)$ / Evidence $P(A)$

is widely used in estimation

The Bayes Classifier

Given:

- Classes $C_1, C_2, \ldots, C_m$
- Data point $x \in \mathfrak{I}$

Bayes' Rule: $P(C_k \mid x) = \dfrac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}} = \dfrac{P(x \mid C_k)\, P(C_k)}{P(x)}$

$$= \frac{P(x \mid C_k)\, P(C_k)}{\sum_{j=1}^{m} P(x \mid C_j)\, P(C_j)}$$

Bayes Classifier: Choose class C_i if $P(C_i \mid x) > P(C_k \mid x) \quad \forall\, k \neq i$

$\Rightarrow$ $g_k(x) = P(x|C_k)\, P(C_k)$ ← Decision Function

because $P(x)$ is common for all classes

Often, $g_k(x) = \log P(x|C_k) + \log P(C_k)$ is used

$P(C_k)$ and $P(x|C_k)$ are *estimated* using the training set.

Suppose a satellite camera observes patches of the Earth and tries to classify each patch based on its color as:

C_1 = Water; C_2 = Rock; C_3 = Sand; C_4 = Vegetation; C_5 = Snow/Ice

The possible observations are: $x \in$ {Blue, Brown, Yellow, Green, White}

Suppose the observation is x = Green

What is the probability that the patch is Water?

$$P(\text{Water} \mid \text{Green}) = \frac{P(\text{Green} \mid \text{Water})\, P(\text{Water})}{P(\text{Green})} = \frac{P(\text{Green} \cap \text{Water})}{P(\text{Green})}$$

- Prior: How likely is it that a random patch on the Earth is Water? – say 0.75 = P(Water)
- Likelihood: Given that a patch is Water, how likely is it to be Green? – say 0.1 = P(Green | Water)
- Evidence: How likely is it that a random patch on the Earth is Green? – say 0.2 = P(Green)

With these values P(Water | Green) = (0.1)(0.75)/0.2 = 0.375

Note that, given the first two probabilities, the third cannot be chosen arbitrarily. For example, we cannot have P(Green) = 0.05, because if Green Water has a probability of $0.75 \times 0.1 = 0.075$, even if all Green is Water, its probability cannot be less than 0.075. If some of the Green is not Water, its probability must be higher.

Three Simple Classifiers

Minimum Distance Classifier

(Minimum Distance to Mean Classifier)

Let class centroids be

$$u^i = \frac{1}{N_i} \sum_{x^k \in C_i} x^k \qquad i = 1, 2, ..., m$$

Note that x^k and u^i are n-dimensional vectors,
where $n \equiv$ dimension of the feature space

N_i = # of patterns of class i in the training set.

Given a ***test pattern***, x

We assign it to the class whose centroid it is closest to.

http://homepages.inf.ed.ac.uk/rbf/HIPR2/classify.htm

Define distance from centroid as:

$$D_i(x) = \left\| x - u^i \right\| \qquad i = 1, \ldots, m$$

where $\|a\| = \sqrt{a^T a}$ (Euclidean norm)

so:

$$D_i(x) = \sqrt{[x - u^i]^T [x - u^i]}$$

$$= \sqrt{x^T x - 2u^{iT} x + u^{iT} u^i}$$

$$= \sqrt{x^T x - 2\left[u^{iT} x - \frac{1}{2} u^{iT} u^i\right]}$$

1. $x^T x$ is in all $D_i(x)$
2. $2\left[u^{iT} x - \frac{1}{2} u^{iT} u^i\right] \le x^T x$

since $[x - u^i]^T [x - u^i] \ge 0$

Our hypothesis is

$$x \in C_i \mid D_i(x) < D_j(x) \quad \forall \quad j \neq i$$

i.e. we need to find the smallest $D_i(x)$

$1, 2 \Rightarrow$ the smallest $D_i(x)$ is the one with the largest

$$u^{iT}x - \frac{1}{2}u^{iT}u^i$$

$$\Rightarrow \boxed{g_i(x) = u^{iT}x - \frac{1}{2}u^{iT}u^i}$$

are decision functions for $i = 1, \ldots, m$

i.e. $x \in C_i$ if $g_i(x) > g_j(x) \quad \forall\ j \neq i$

Decision boundary :

$$[u^i - u^j]^{\mathrm{T}} x - \frac{1}{2}\left[u^{iT}u^i - u^{jT}u^j\right] = 0$$

Perpendicular bisector of the two centroids

Example:

Given patterns

$$x^1 = [10 \;\; 10]^T \in C_1$$
$$x^2 = [4 \;\; 3]^T \in C_2$$
$$x^3 = [11 \;\; 12]^T \in C_1$$
$$x^4 = [3 \;\; 2]^T \in C_2$$

The feature space is 2-d $\Im = (x_1, x_2)$

so $x_1^1 = 10$, $x_2^1 = 10$

$x_1^2 = 4$, $x_2^2 = 3$ etc.

Represent classes by output :

$$y = 0 \Rightarrow C_1$$
$$y = 1 \Rightarrow C_2$$

Training set

x_1	x_2	y
10	10	0
4	3	1
11	12	0
3	2	1

$$u^1 = \frac{1}{2}\left(\begin{bmatrix}10\\10\end{bmatrix} + \begin{bmatrix}11\\12\end{bmatrix}\right) = \begin{bmatrix}10.5\\11\end{bmatrix}$$

$$u^2 = \frac{1}{2}\left(\begin{bmatrix}4\\3\end{bmatrix} + \begin{bmatrix}3\\2\end{bmatrix}\right) = \begin{bmatrix}3.5\\2.5\end{bmatrix}$$

$$g_1(x) = u^{1T}x - \frac{1}{2}u^{1T}u^1$$

$$= 10.5x_1 + 11x_2 - \frac{1}{2}\begin{bmatrix}10.5 & 11\end{bmatrix}\begin{bmatrix}10.5\\11\end{bmatrix}$$

$$g_1(x) = 10.5x_1 + 11x_2 - 115.62$$

similarly

$$g_2(x) = u^{2T}x - \frac{1}{2}u^{2T}u^2$$

$$g_2(x) = 3.5x_1 + 2.5x_2 - 9.25$$

Given test pattern $x = \begin{bmatrix}5 & 2\end{bmatrix}^T$

we calculate

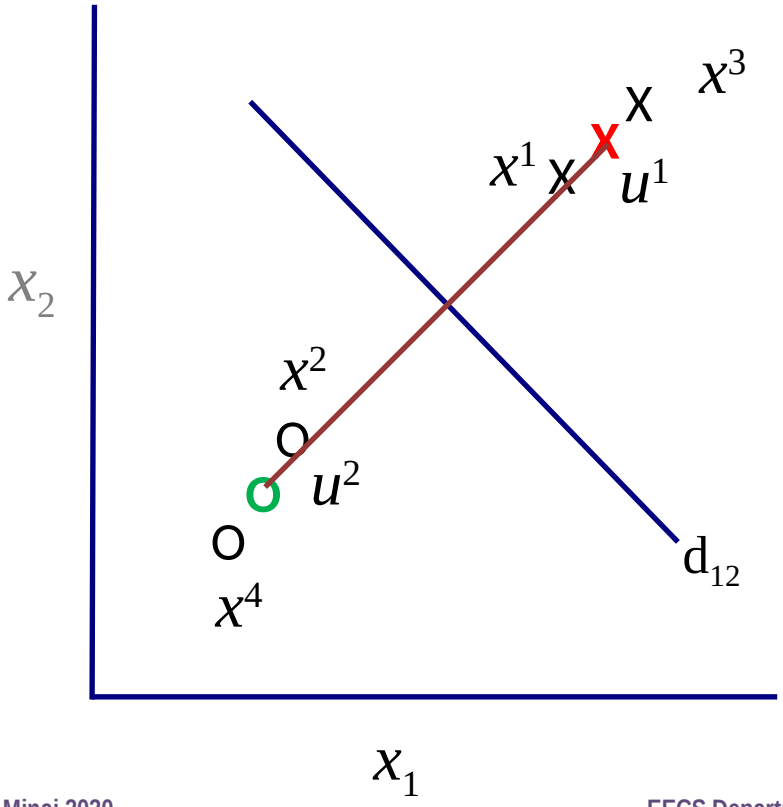
$$g_1(x) = 10.5(5) + 11(2) - 115.62$$
$$= -41.12$$
$$g_2(x) = 3.5(5) + 2.5(2) - 9.25$$
$$= 13.25$$
$$\Rightarrow \quad x \in C_2$$

Decision boundary:

$$d_{12} = \{ x : \; g_1(x) - g_2(x) = 0 \}$$

$$10.5x_1 + 11x_2 - 115.62 - \left[3.5x_1 + 2.5x_2 - 9.25 \right] = 0$$

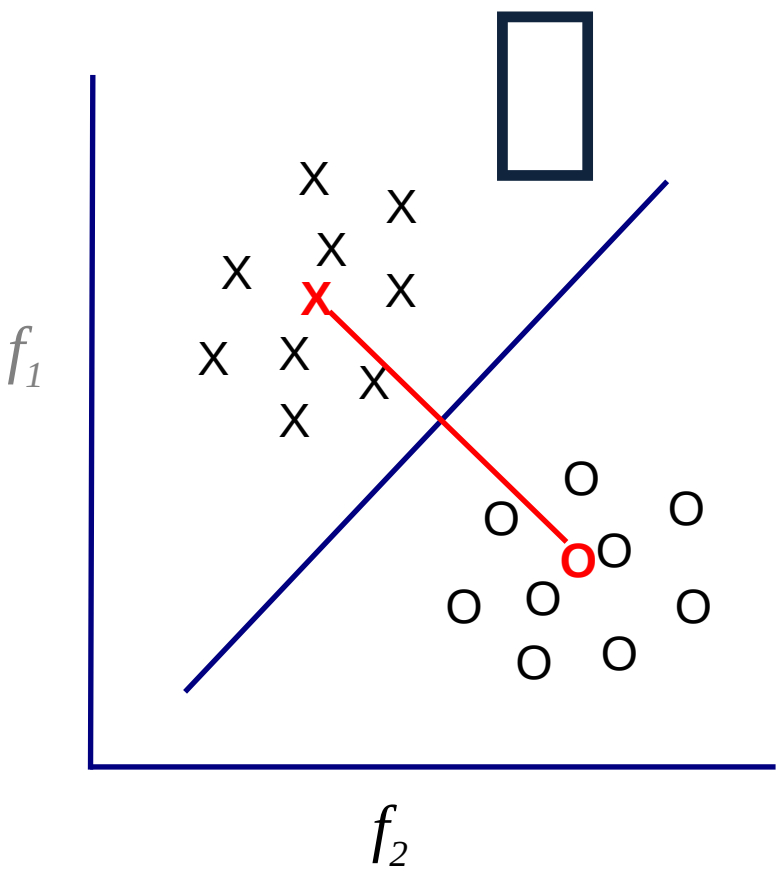
$$7x_1 + 8.5x_2 - 106.37 = 0$$

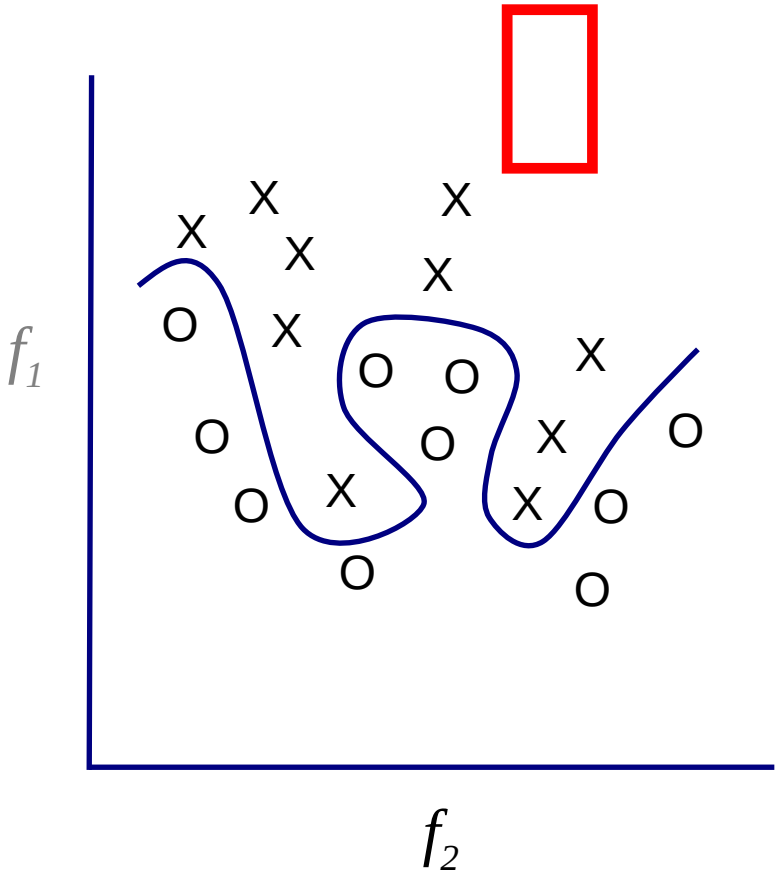


d_{12} is the $\perp$ bisector of the line joining u^1 and u^2

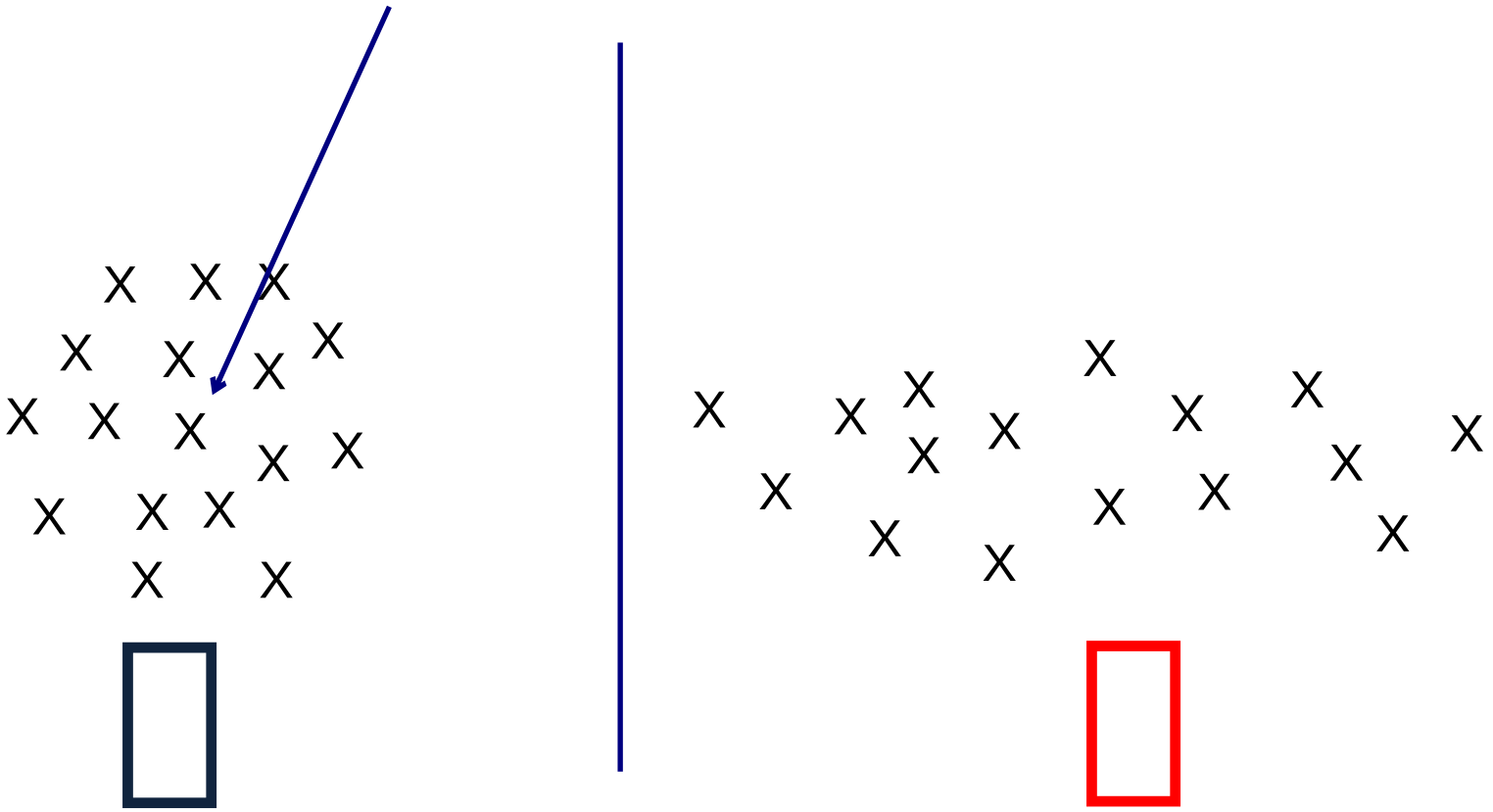
Critique of MDC's

A. Pairs of classes must be linearly separable – since d_{ij} is a line

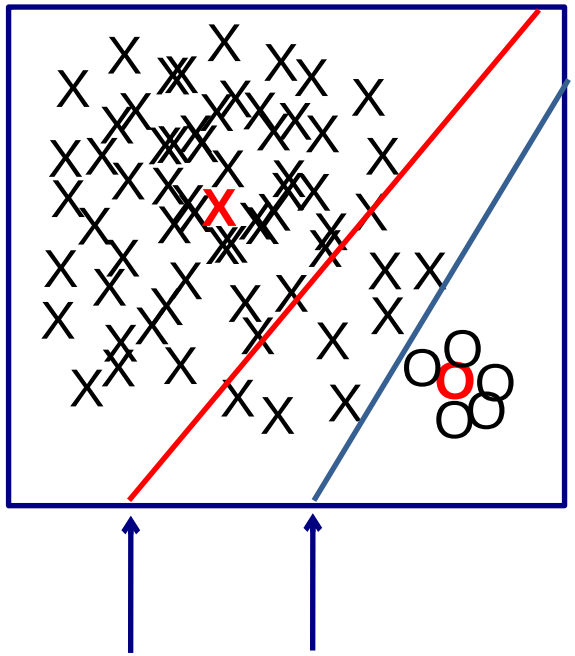




B. MDC's work best when patterns in each class are distributed in a "spherical hypercloud" around the class mean.



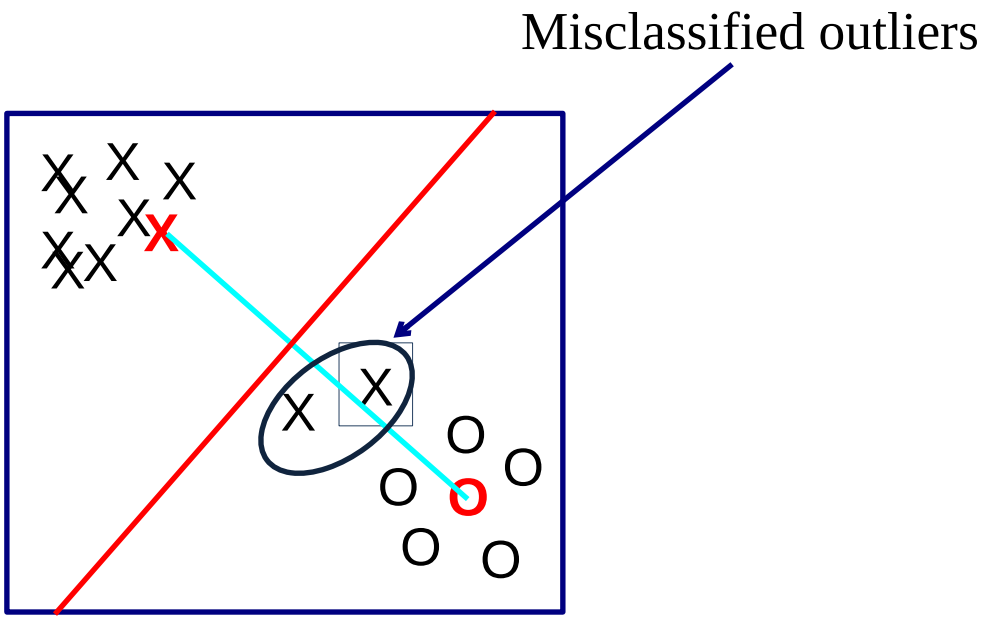
C. MDC's can have problems if nearby classes have very different spreads



Inferred Decision Boundary

Ideal Decision Boundary

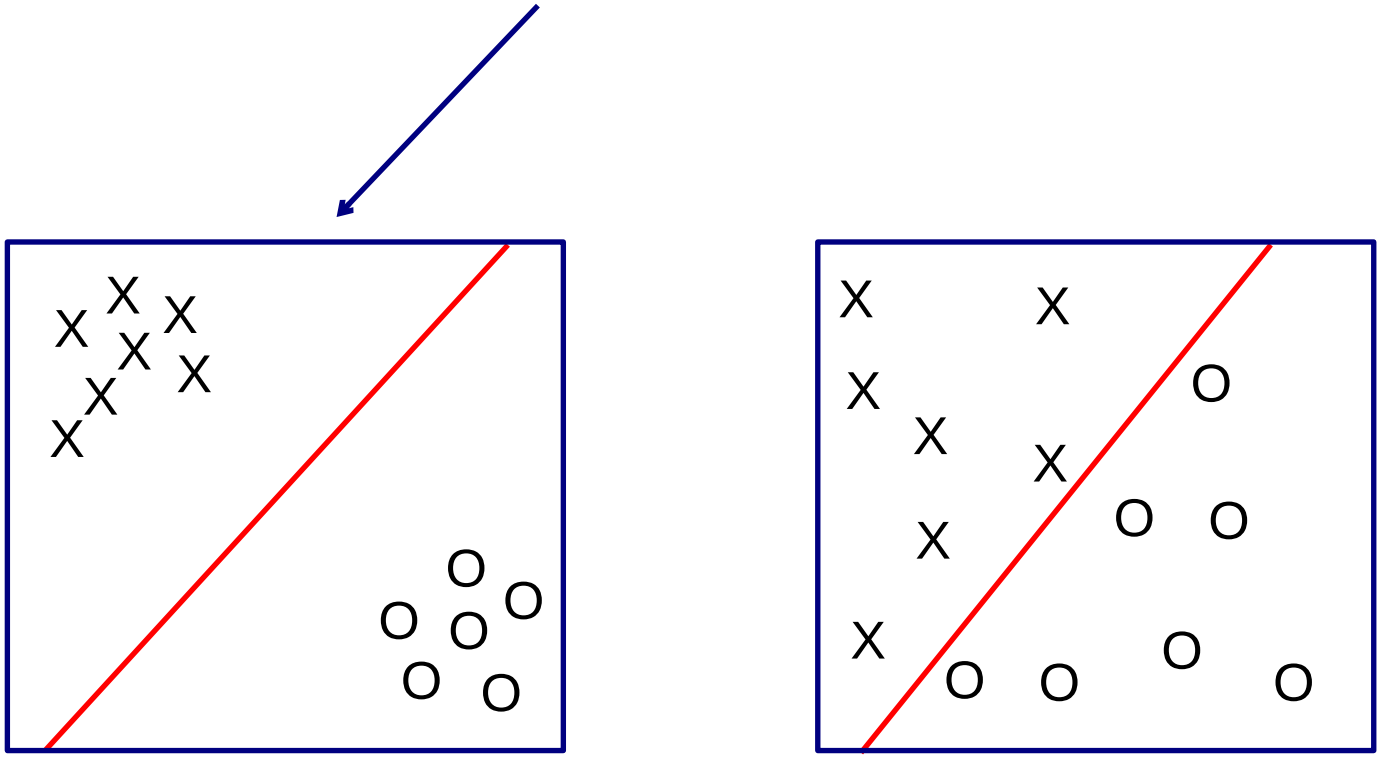
D. Outliers can cause problems for minimum distance classifiers



In general, it should not be possible for exemplars of class *i* to be closer to some u^j , $j \neq i$ than to u^i

Note that in this case, the classes are linearly separable, but the classifier still makes errors.

E. Like most classifiers, minimum distance classifiers work well if interclass distances are large compared to class spreads.



Well-separated classes

Cutting it close – new data may fall on the wrong side.

The minimum distance classifier is a Bayes classifier with the following assumptions:

- All classes have Gaussian (normal) distributions.
- All classes have the same variance.
- All classes have diagonal covariance matrices (features are independent).
- All classes have the same prior probability of occurrence.

If some of these assumptions are not true, the classifier fails, and a more complicated classifier must be used.

For example, a quadratic classifier:

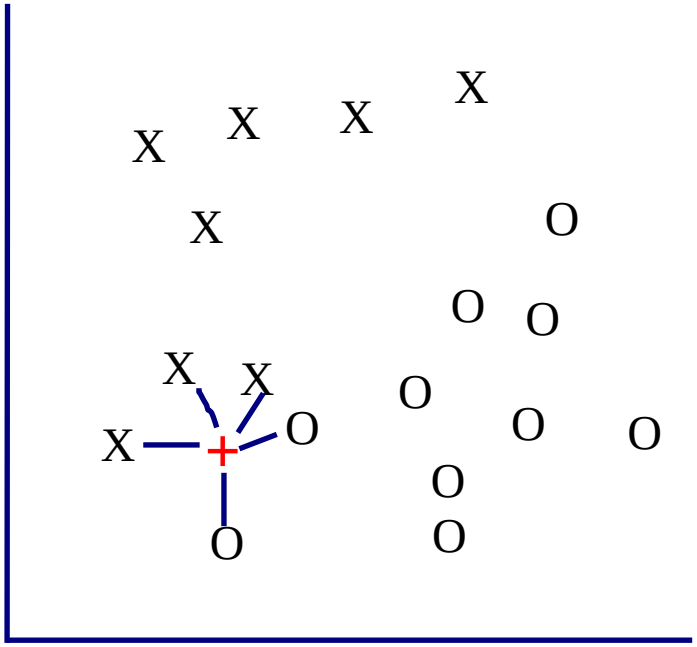
$$\Rightarrow \quad g_i(x) = K_i^{-1} u^{iT} x - \frac{1}{2} u^{iT} K_i^{-1} u^i + \log P(C_i)$$

where K_i is the covariance matrix of features in class C_i

The k-Nearest Neighbors (KNN) Classifier

Given set $X = \{x^1, x^2, ..., x^n\}$ $x^k \in \Im$ of patterns with known classes and novel pattern $x \in \Im$

1. Find $x^{q(1)}, ..., x^{q(k)} \in X$ that are closest to x in $\Im$
2. Check the classes of the $x^{q(m)}$, $m = 1, ..., k$
   Assign x to the class with the highest count



X = Class A
O = Class B

5 nearest neighbors
→ 3A 2B
→ + is classified as A

Pros and Cons of KNN

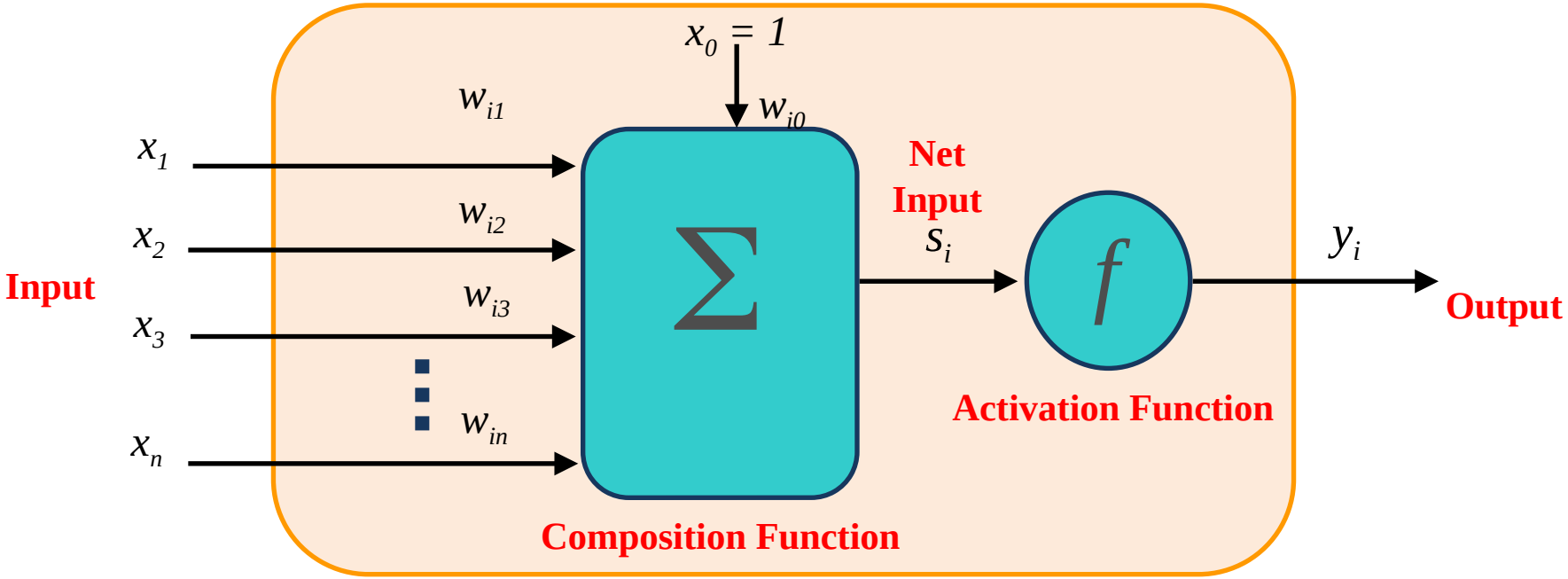
Pros

- Very simple to implement.
- Makes intuitive sense.
- Requires no optimization or training (at least in this simple form).
- Easy to use.
- Can handle nonlinearly separable classes.

Cons

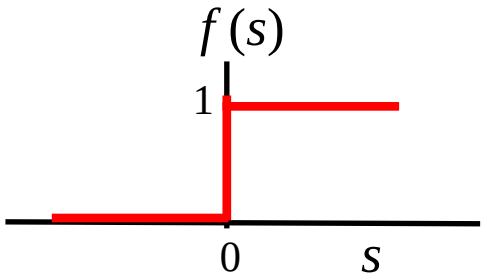
- Class boundaries are implicit, i.e., not explicitly defined by an equation as in MDC.
- Outliers can lead to poor results.

The Perceptron



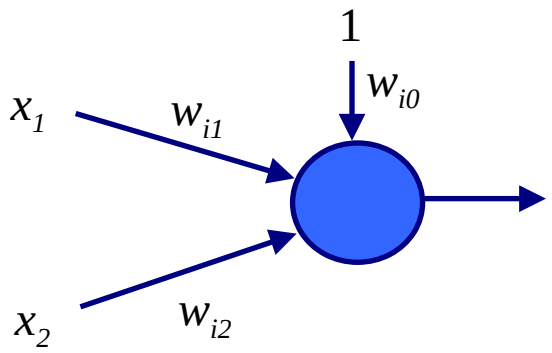
$$s_i = \sum_j w_{ij} x_j \qquad f(s) = \begin{cases} 1 & if\ s > 0 \\ 0 & else \end{cases}$$

$$y_i = f(s_i)$$



The perceptron is a Linear Threshold Unit (LTU)

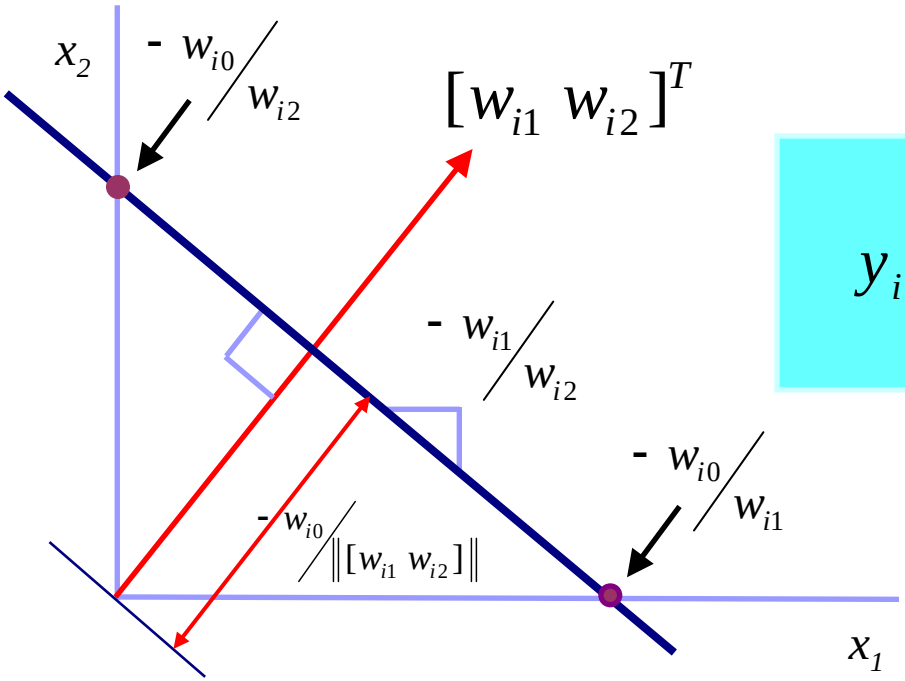
2-Dimensional Case



Decision Boundary:

$$w_{i1}x_1 + w_{i2}x_2 + w_{i0} = 0$$

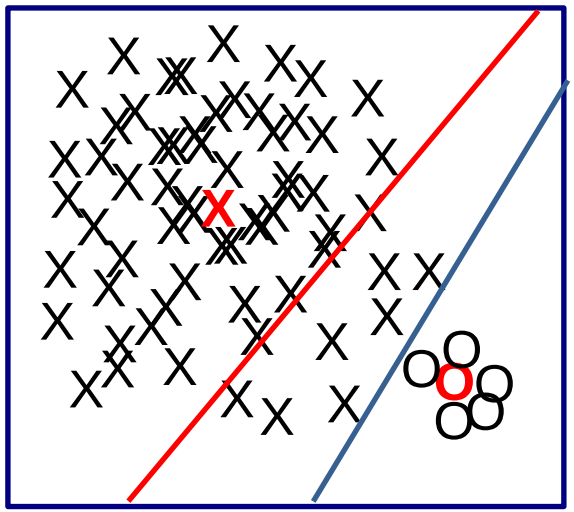
$$x_2 = -\frac{w_{i1}}{w_{i2}}x_1 - \frac{w_{i0}}{w_{i2}}$$



$$y_i = \begin{cases} 1 & \text{if } w_{i1}x_1 + w_{i2}x_2 > -w_{i0} \\ 0 & \text{if } w_{i1}x_1 + w_{i2}x_2 \leq -w_{i0} \end{cases}$$

Perceptron Decision Boundary

The perceptron is a linear classifier. It can draw ***any*** linear boundary in feature space.



Minimum Distance Classifier Decision Boundary

Perceptron Decision Boundary

The Perceptron Learning Rule

Given

Input vectors	$X = \{\bar{x}^1, \bar{x}^2, ..., \bar{x}^m\}$	$\bar{x}^k \in \Re^n$
Outputs	$Y_i = \{y_i^1, y_i^2, ..., y_i^m\}$	$y_i^k \in \{0,1\}$

The task is to set $\bar{w}_i = \{w_{i1}, w_{i2}, ..., w_{in}\}$ and w_{i0} such that all $\bar{x}^k$ are classified "as well as possible".

If the $\bar{x}^k$ are linearly separable, perfect classification is possible. Otherwise not.

Basic Approach

- Begin with an initial $\bar{w}_i$
- Present each $(\bar{x}^k, y_i^k)$ pair and change $\bar{w}_i$ to make a correct y^k more likely
- Iterate till done (or a limit is reached)

Algorithm

1. Initialize $\overline{w}_i$ and w_{i0} to random values.

2. Check if stopping criterion is met:

   if yes $\rightarrow$ stop

Possible stopping criteria:
- All inputs classified correctly
- A time limit is reached

3. For $k = 1$ to m

   a. Present $\overline{x}^k$ to the perceptron, and

   calculate $\hat{y}_i^k = f\left(\overline{w}_i^T \overline{x}^k + w_{i0}\right)$

   This is the actual output.

   b. Change weights as:

$$w_{ij} = w_{ij} + \eta \left[y_i^k - \hat{y}_i^k \right] x_j^k$$

$$w_{i0} = w_{i0} + \eta \left[y_i^k - \hat{y}_i^k \right]$$

$$w_{ij} = w_{ij} + \eta \left[y_i^k - \hat{y}_i^k \right] x_j^k$$

η is called the learning rate

4. Repeat from step 2.

Why it works….

Before weight adjustment, $\bar{x}^k$ produces

$$\hat{y}_i^k = f\left(\bar{w}_i^T \bar{x}^k + w_{i0}\right) \qquad = f(s_k)$$

After adjustment

$$\tilde{y}_i^k = f\left\{ \left[\bar{w}_i + \left(y_i^k - \hat{y}_i^k\right)\bar{x}^k\right]^T x^k + w_{io} + \left(y_i^k - \hat{y}_i^k\right)\right\}$$

$$= f\left\{ \left[\bar{w}_i^T \bar{x}^k + w_{io}\right] + \left(y_i^k - \hat{y}_i^k\right)\left(1 + \bar{x}^{kT}\bar{x}^k\right)\right\}$$

δ

> 0

$$= f(s_k + \delta) \qquad \text{if } \hat{y}_i^k = y_i^k,\ \delta = 0$$

Case 1 $\quad y_i^k = 0, \hat{y}_i^k = 1 \Rightarrow \delta < 0 \Rightarrow s_k + \delta < s_k$

$\Rightarrow \tilde{y}_i^k$ is likelier to be 0

Case 2 $\quad y_i^k = 1, \hat{y}_i^k = 0 \Rightarrow \delta > 0 \Rightarrow s_k + \delta > s_k$

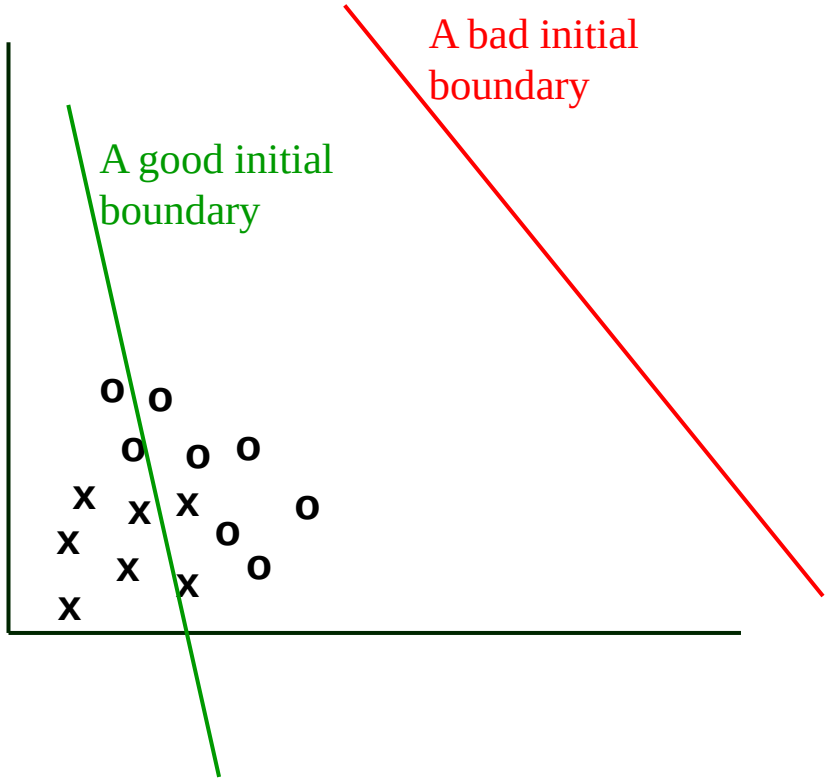
$\Rightarrow \tilde{y}_i^k$ is likelier to be 1

If the $\bar{x}^k$ that produce $y_i^k = 0$ and 1 are linearly separable:
The perceptron algorithm will find a correct $\bar{w}_i$ in a finite number of steps.

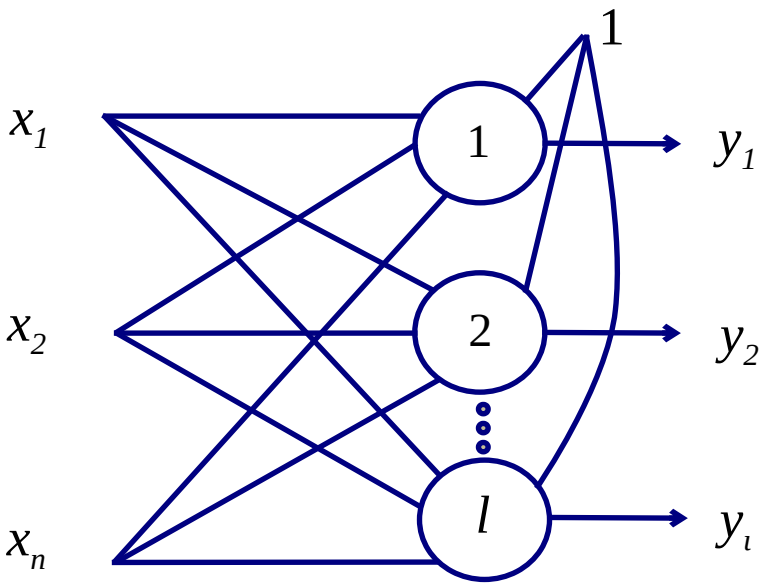
If the sets are not linearly separable:
The algorithm will not converge
→ STOP after a certain number of steps is reached.

It is very important to choose:
- The right initial weights.
- An appropriate learning rate.

Learning rate should generally be small to avoid overshoot and oscillation…..
……. but this only works if the initial boundary (= weights) is chosen well.



Multiple Output Case



Output vector

$$\hat{\bar{y}}^k = \left[\hat{y}_1^k \;\; y_2^k \;\; \ldots \; y_\ell^k\right]^T$$

Each $\hat{y}_i$ depends separately on $\bar{x}$.

⇒ Each perceptron can be trained separately and simultaneously using the perceptron algorithm without interactions.