Predicting the Future Classification

Presented by:
Dr Noureddin Sadawi

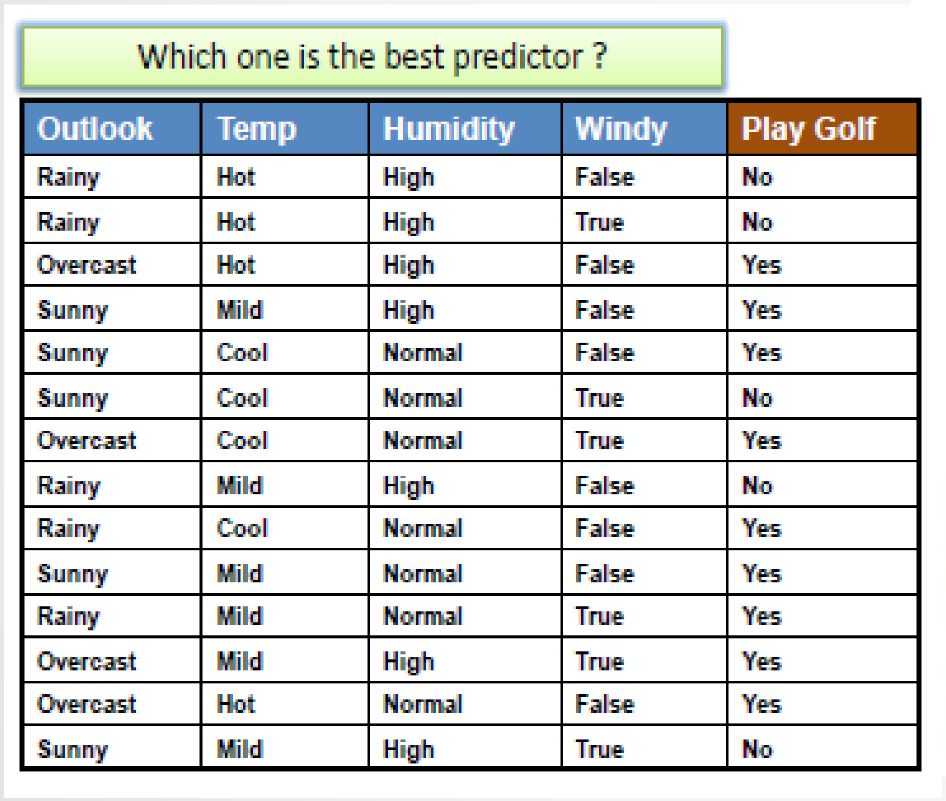
Material belongs to Prof Saed Sayad
http://www.saedsayad.com

Classification

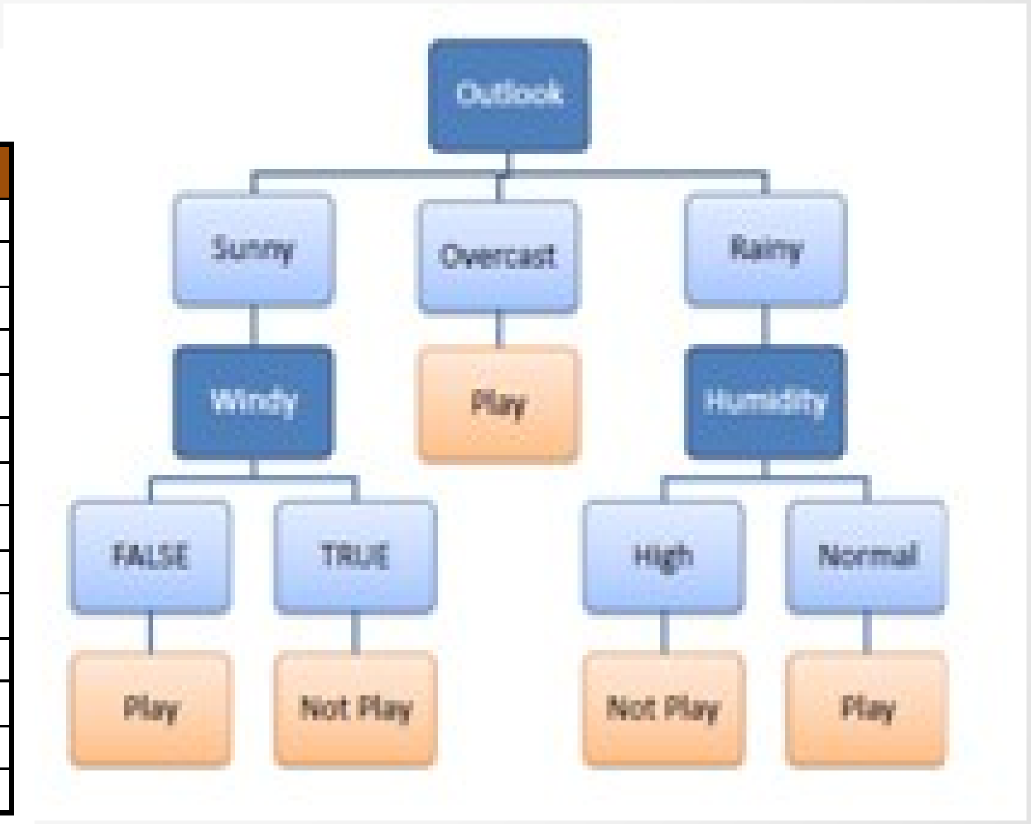
- Classification is a data mining task of predicting the value of a categorical variable (target or class)
- This is done by building a model based on one or more numerical and/or categorical variables (predictors, attributes or features)

Remember!

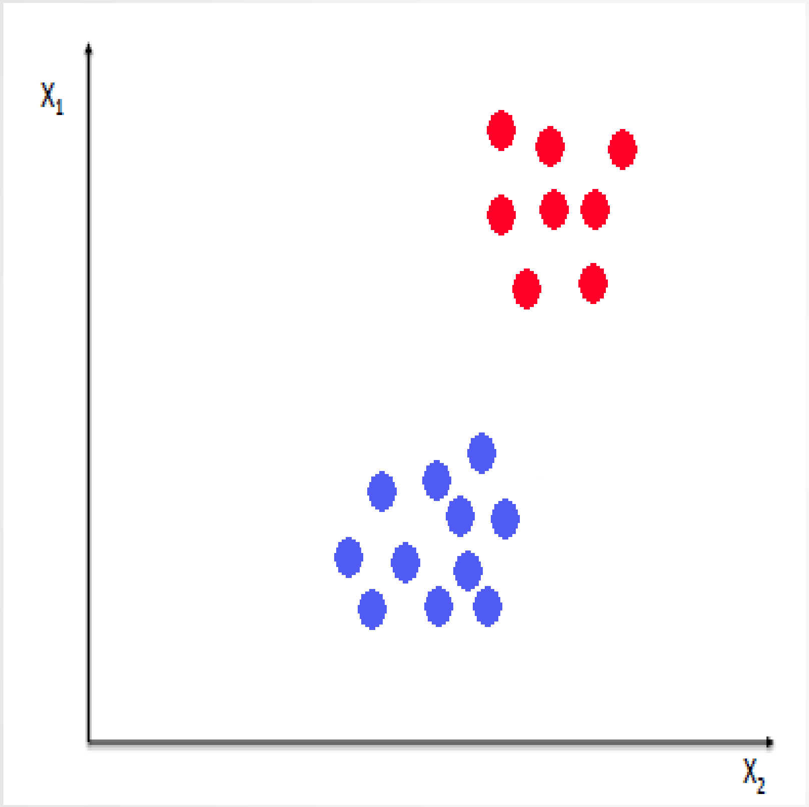
- Transforming Attributes
  - Numerical to Categorical (Binning or Discretization)
  - Categorical to Numerical (Encoding or Continuization)

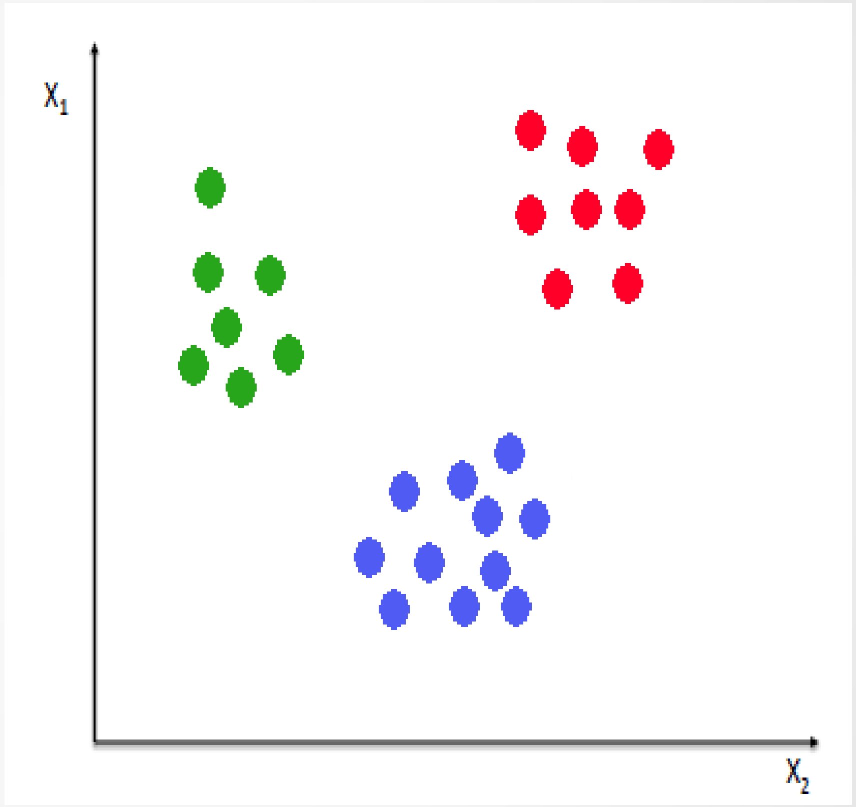

Which one is the best predictor ?

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



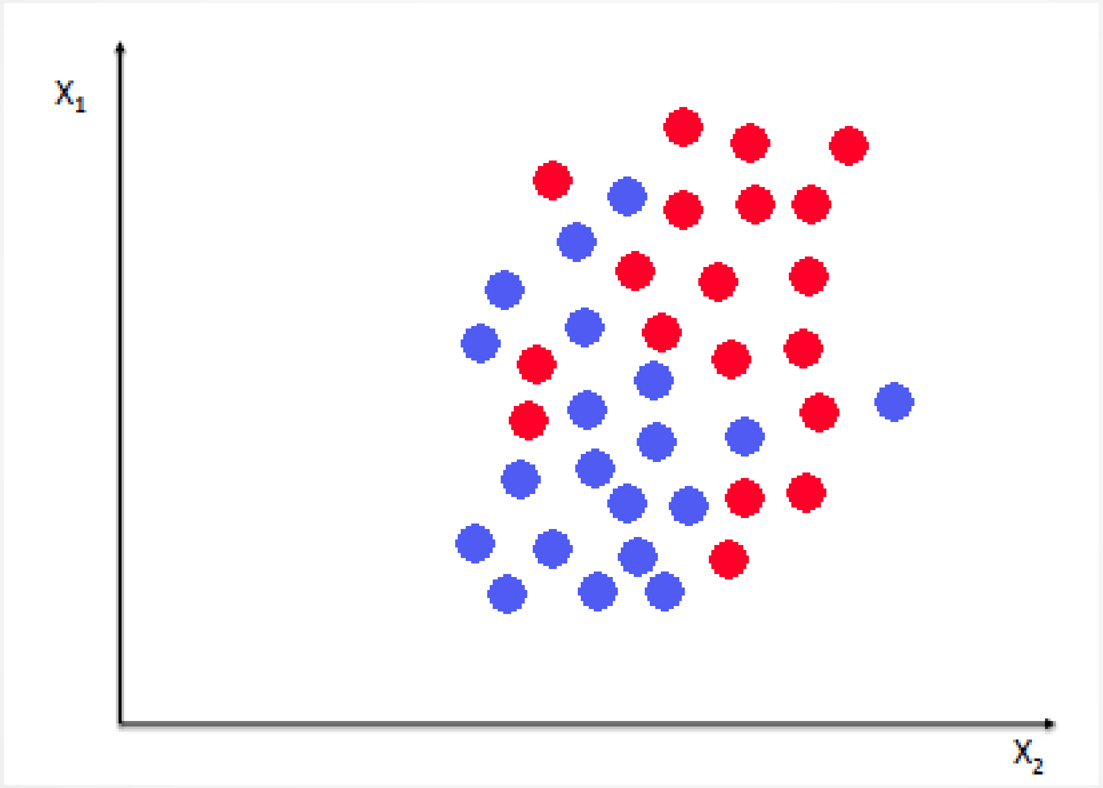
Linear/Non-linear Separability



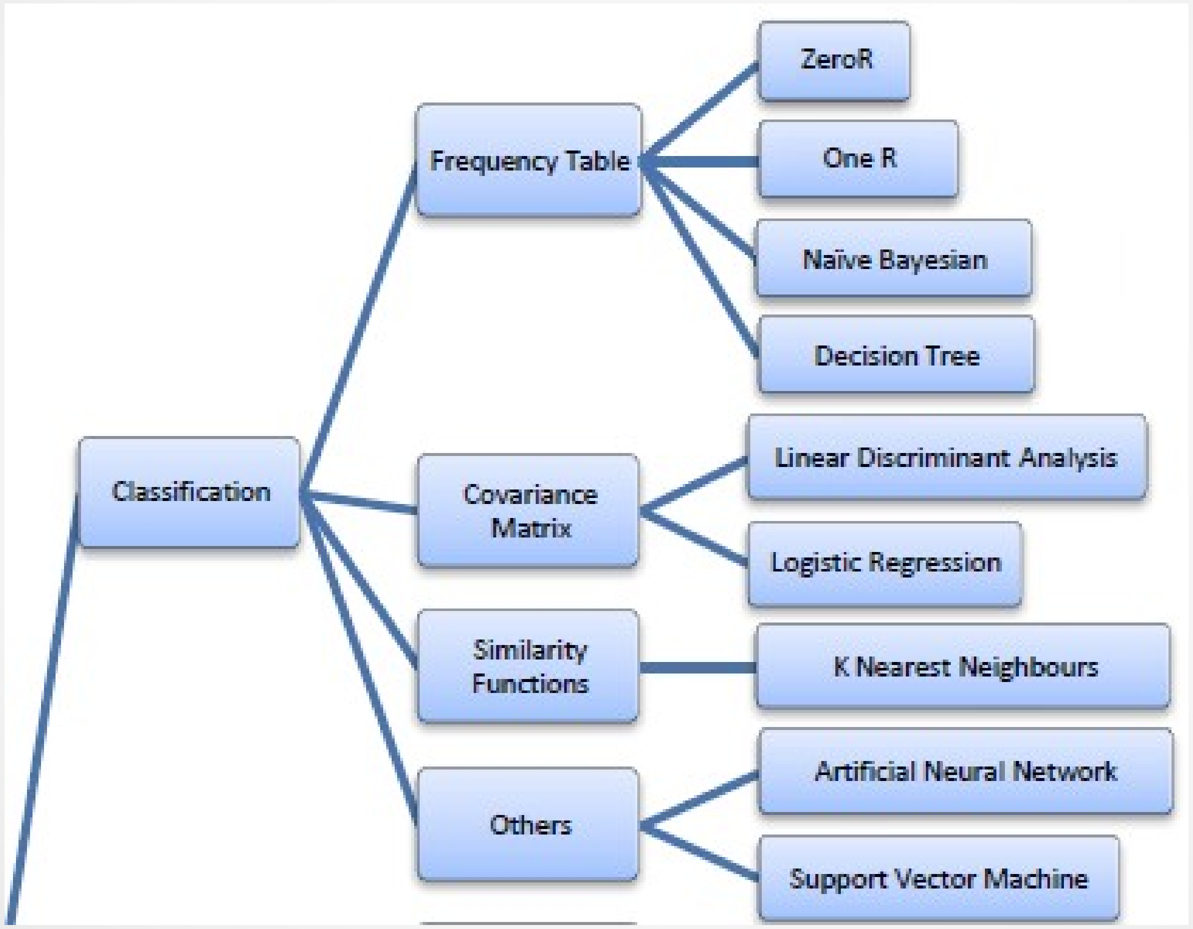


Important to use Diagrams

Linear/Non-linear Separability



Important to use Diagrams


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

Classification

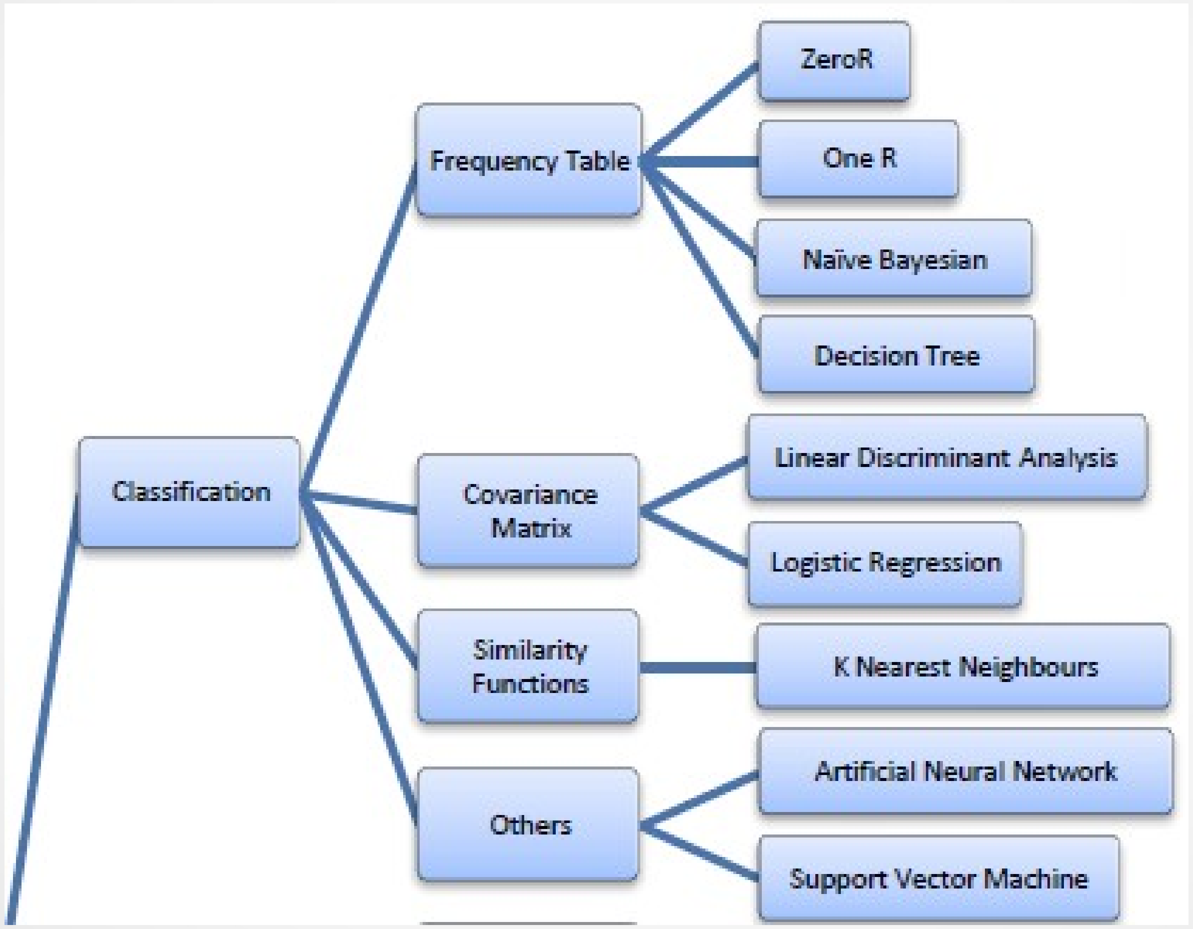
Four main groups of classification algorithms are:

- **Frequency Table**
  - ZeroR
  - OneR
  - Naive Bayesian
  - Decision Tree
- **Covariance Matrix**
  - Linear Discriminant Analysis
  - Logistic Regression
- **Similarity Functions**
  - K Nearest Neighbours
- **Others**
  - Artificial Neural Network
  - Support Vector Machine

The ZeroR Classifier

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

ZeroR

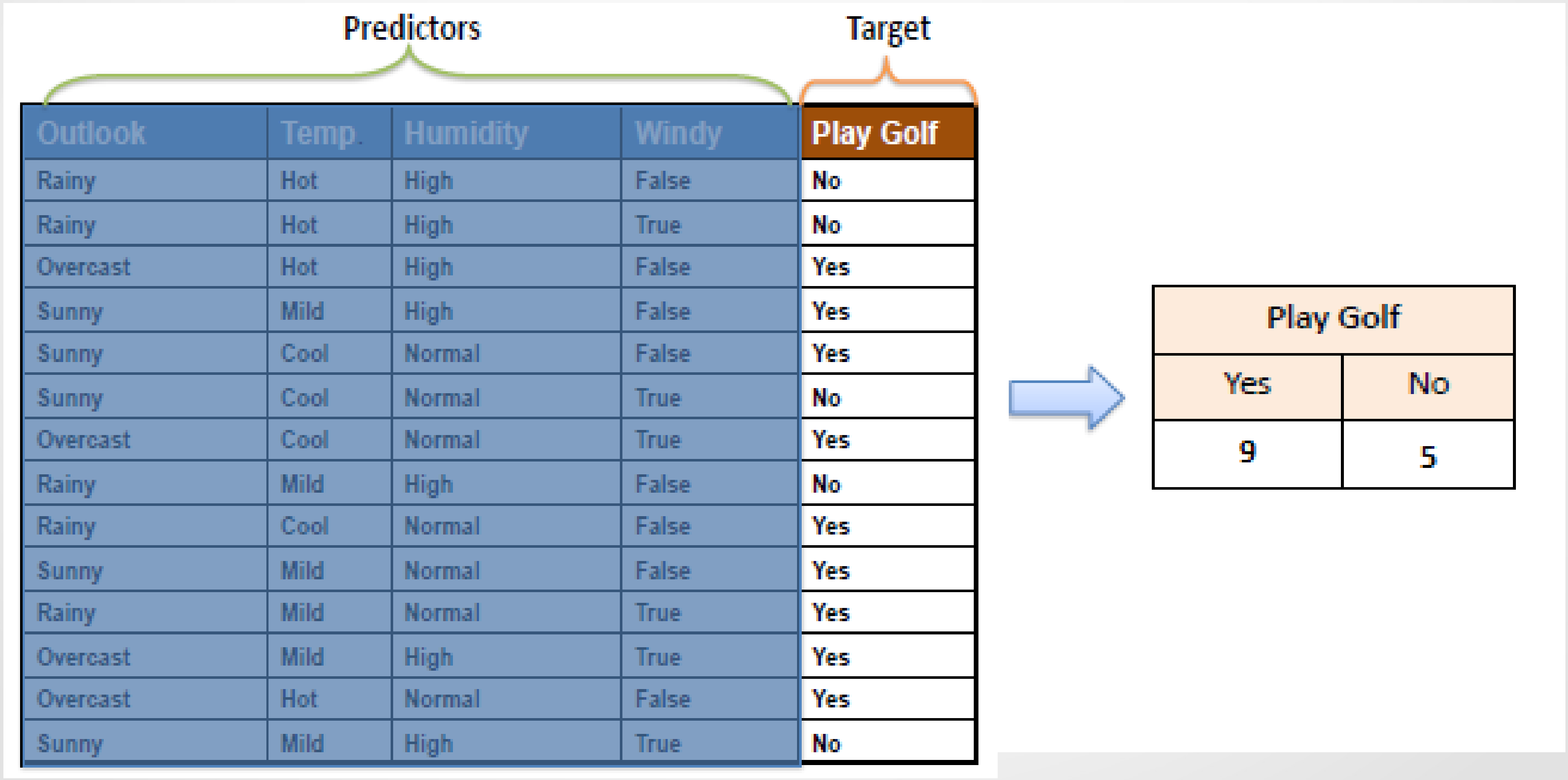
- ZeroR is the simplest classification method *which relies on the target and ignores all predictors*
- ZeroR classifier simply predicts the majority category (class)
- Although there is no predictability power in ZeroR, it is useful for determining a baseline performance as a benchmark for other classification methods

How it works

- Construct a frequency table for the target and select its most frequent value

ZeroR Example

- "Play Golf = Yes" is the ZeroR model for the following dataset with an accuracy of 0.64

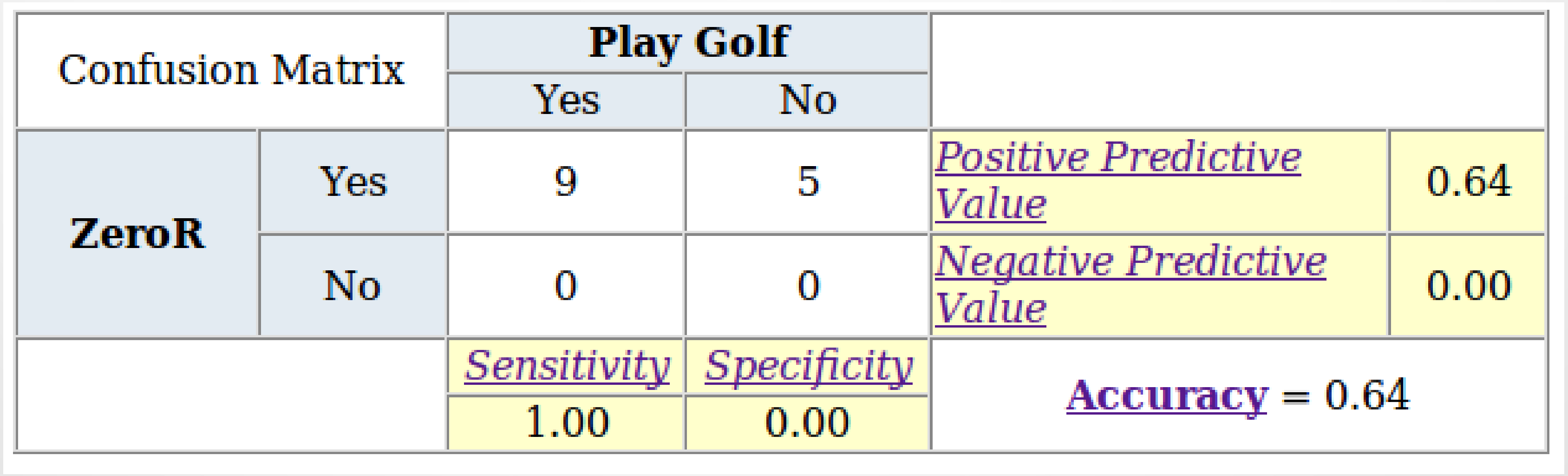

Predictors
Target

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Play Golf	
Yes	No
9	5

ZeroR Model Evaluation

- The following Confusion Matrix shows that **ZeroR** only predicts the majority class correctly
- As mentioned before, ZeroR is only useful for determining a baseline performance for other classification methods

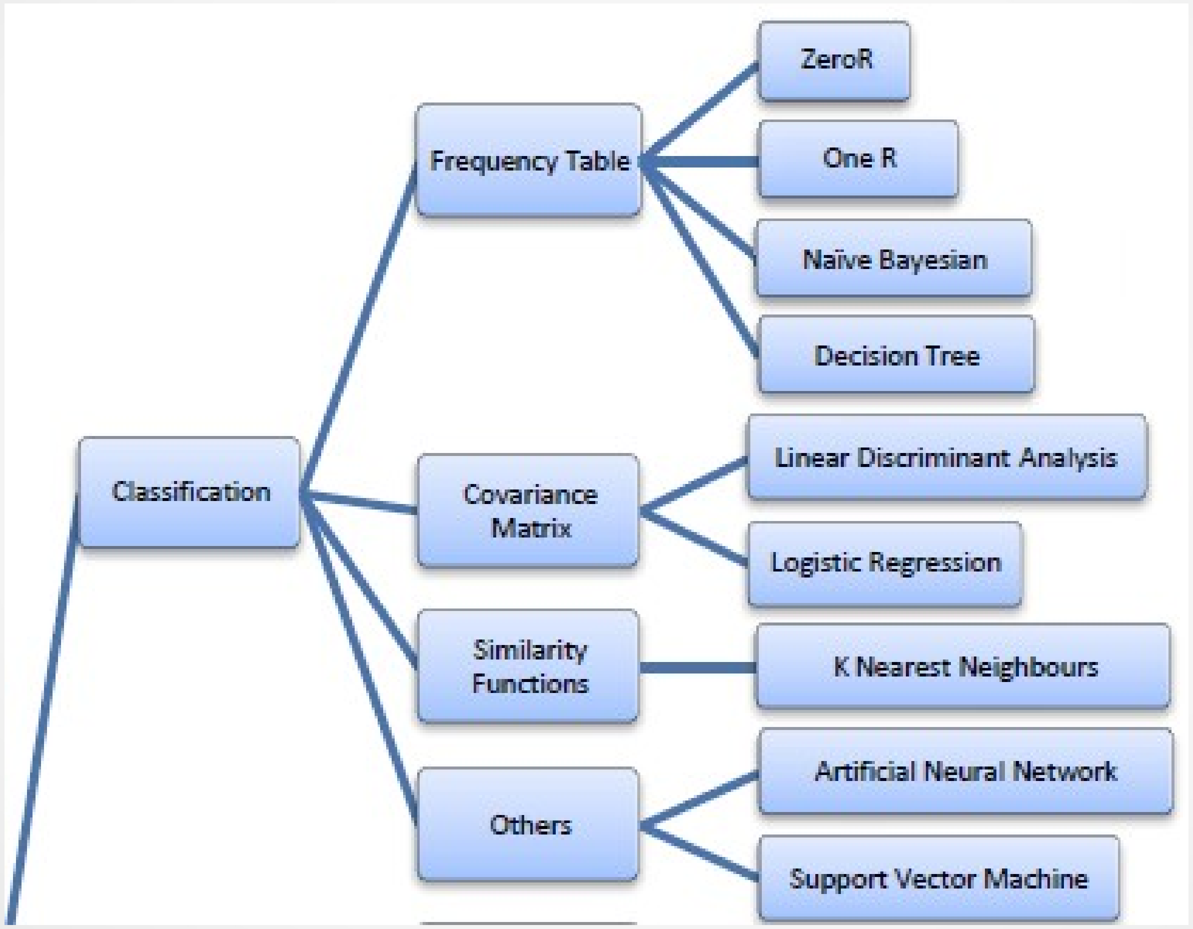


Confusion Matrix		**Play Golf**			
		Yes	No		
ZeroR	Yes	9	5	*Positive Predictive Value*	0.64
	No	0	0	*Negative Predictive Value*	0.00
		Sensitivity	*Specificity*	**Accuracy** = 0.64	
		1.00	0.00		

The OneR Classifier

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

OneR

- OneR, short for "One Rule", is a simple, yet accurate, classification algorithm

  How it works:

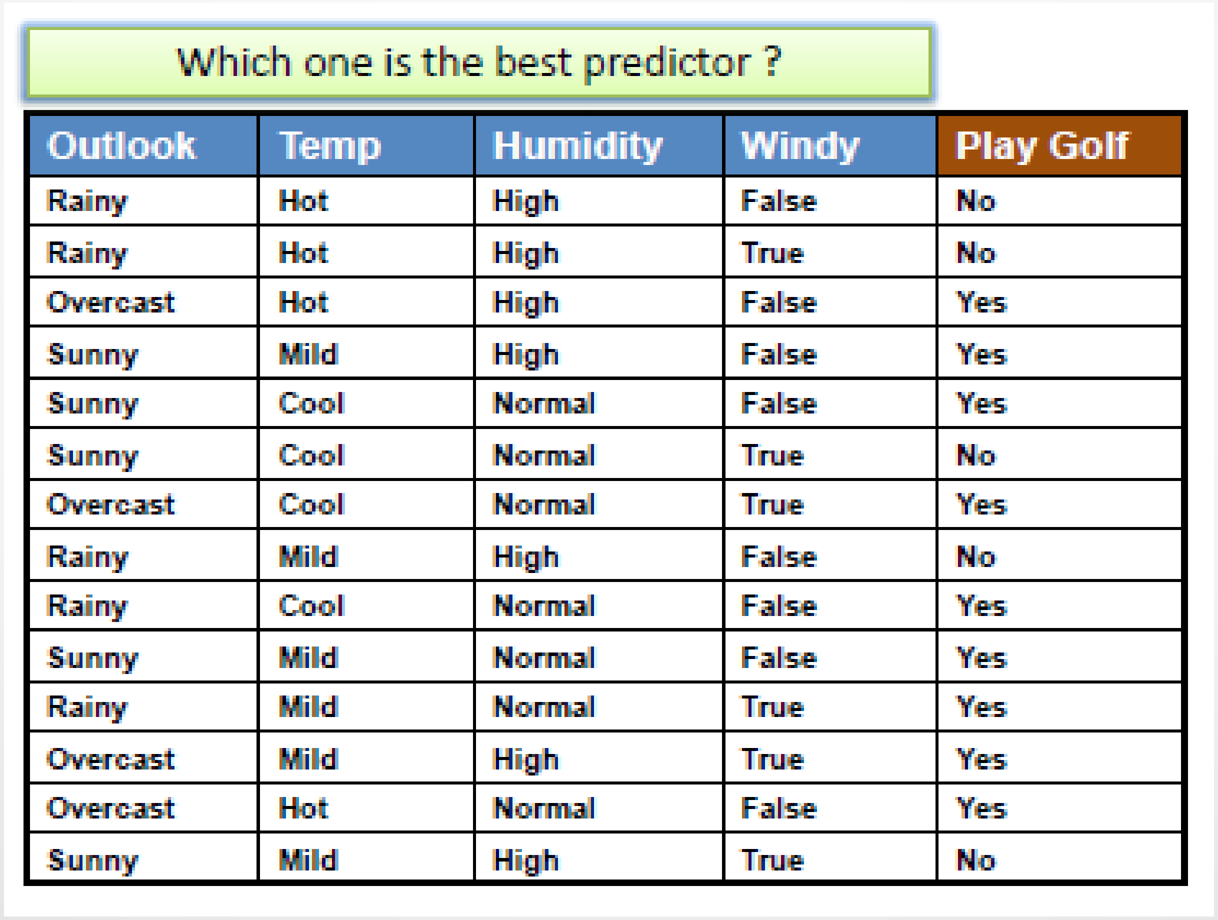
- It generates one rule for each predictor in the data, then selects the rule with the smallest total error as its "one rule"
- To create a rule for a predictor, we construct a frequency table for each predictor against the target
- It has been shown that OneR produces rules only slightly less accurate than state-of-the-art classification algorithms while producing rules that are simple for humans to interpret

OneR Algorithm

```
For each predictor,
    For each value of that predictor, make a rule as follows;
        Count how often each value of target (class) appears
        Find the most frequent class
        Make the rule assign that class to this value of the
           predictor
    Calculate the total error of the rules of each predictor
Choose the predictor with the smallest total error
```

Example

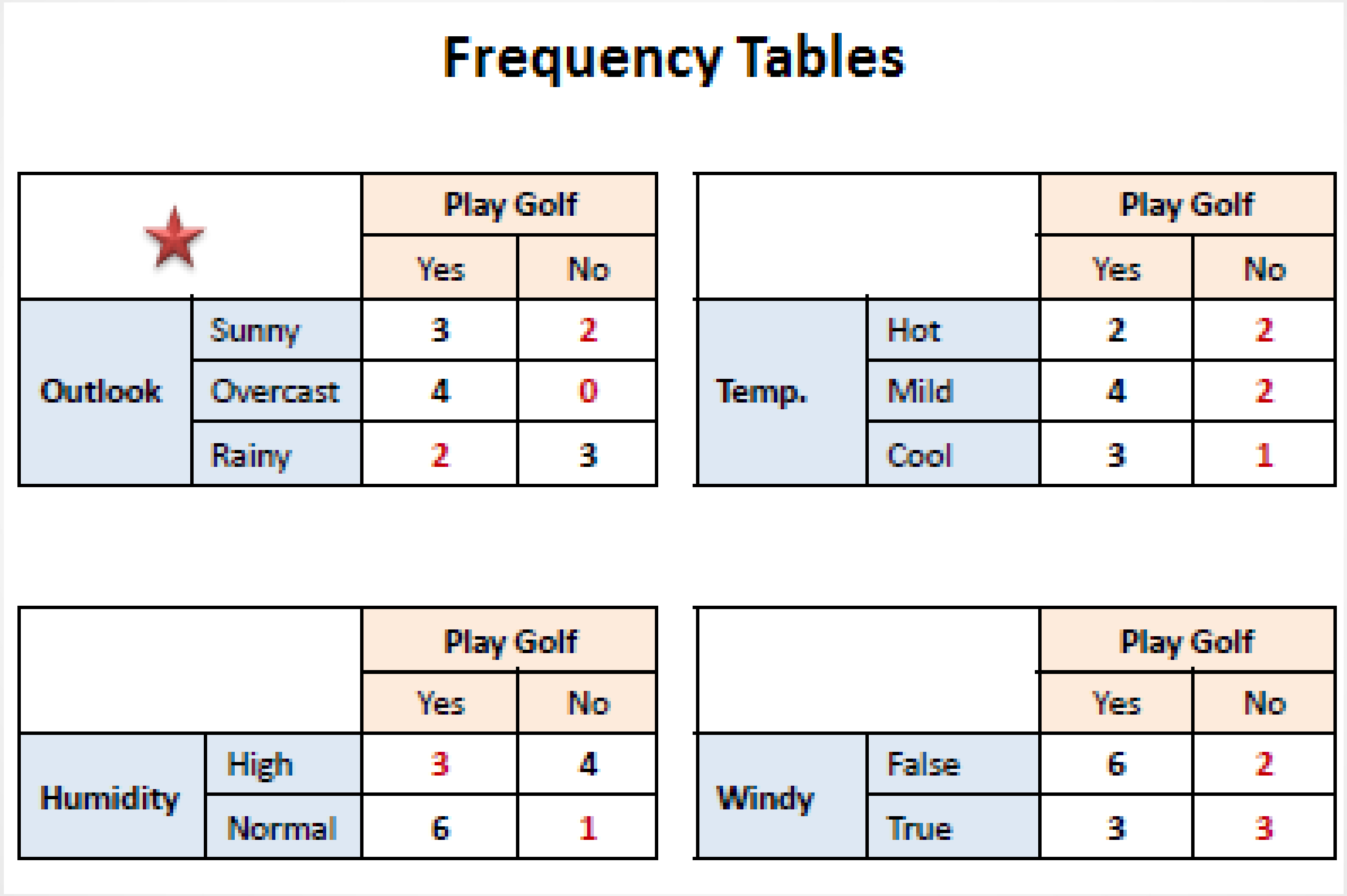
- Finding the best predictor with the smallest total error using OneR algorithm based on related frequency tables



Which one is the best predictor ?

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Example



Frequency Tables

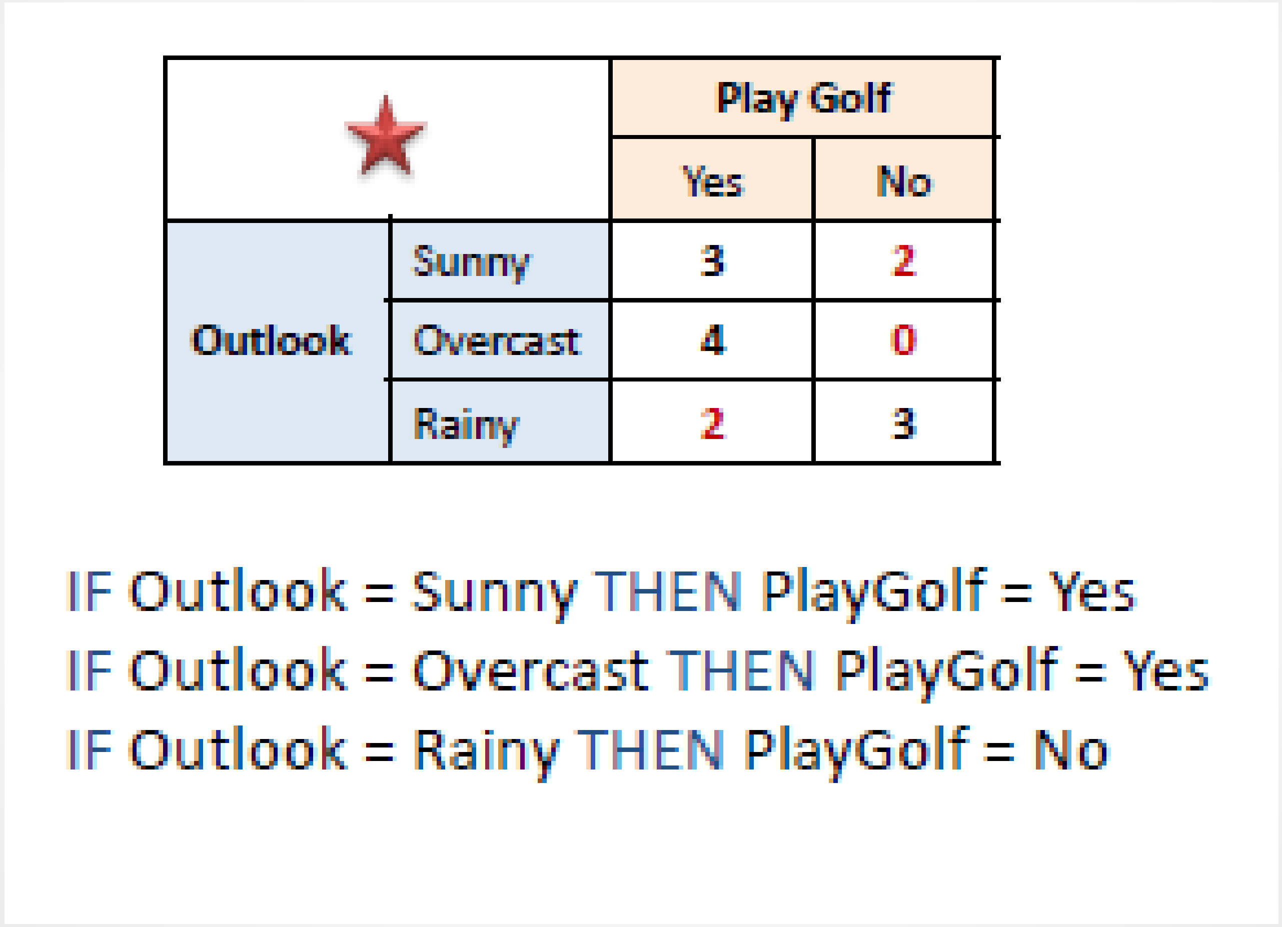
★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

The best predictor is:



★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

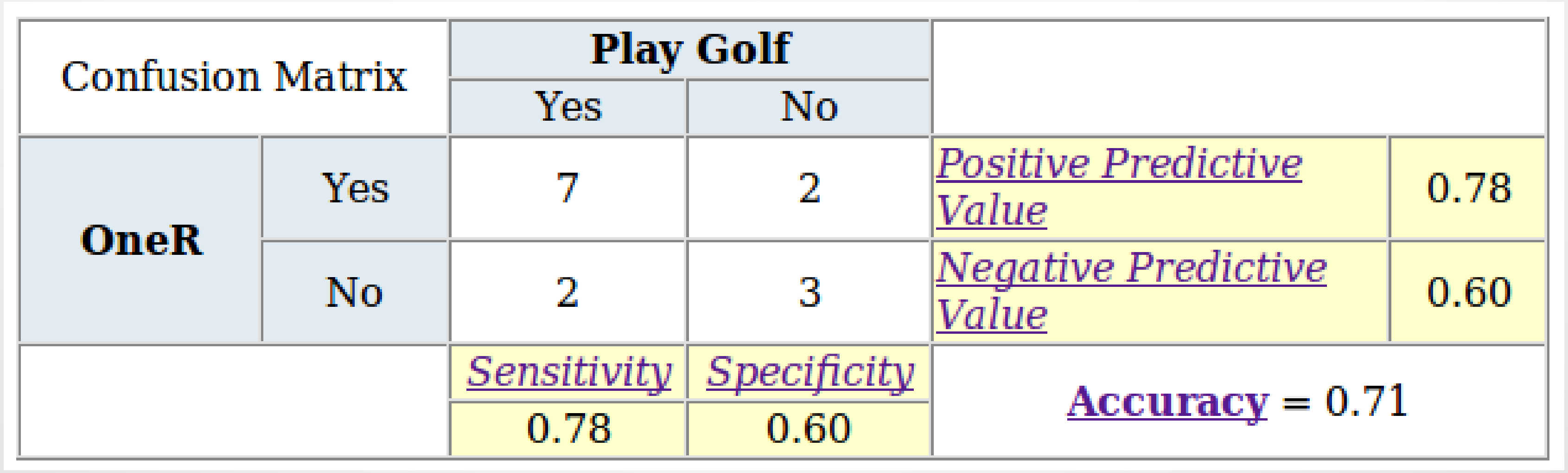
IF Outlook = Sunny THEN PlayGolf = Yes
IF Outlook = Overcast THEN PlayGolf = Yes
IF Outlook = Rainy THEN PlayGolf = No

Predictors Contribution

- Simply, the total error calculated from the frequency tables is the measure of each predictor contribution
- A low total error means a higher contribution to the predictability of the model

Model Evaluation

- The following confusion matrix shows significant predictability power
- OneR does not generate score or probability, which means evaluation charts (Gain, Lift, K-S and ROC) are not applicable



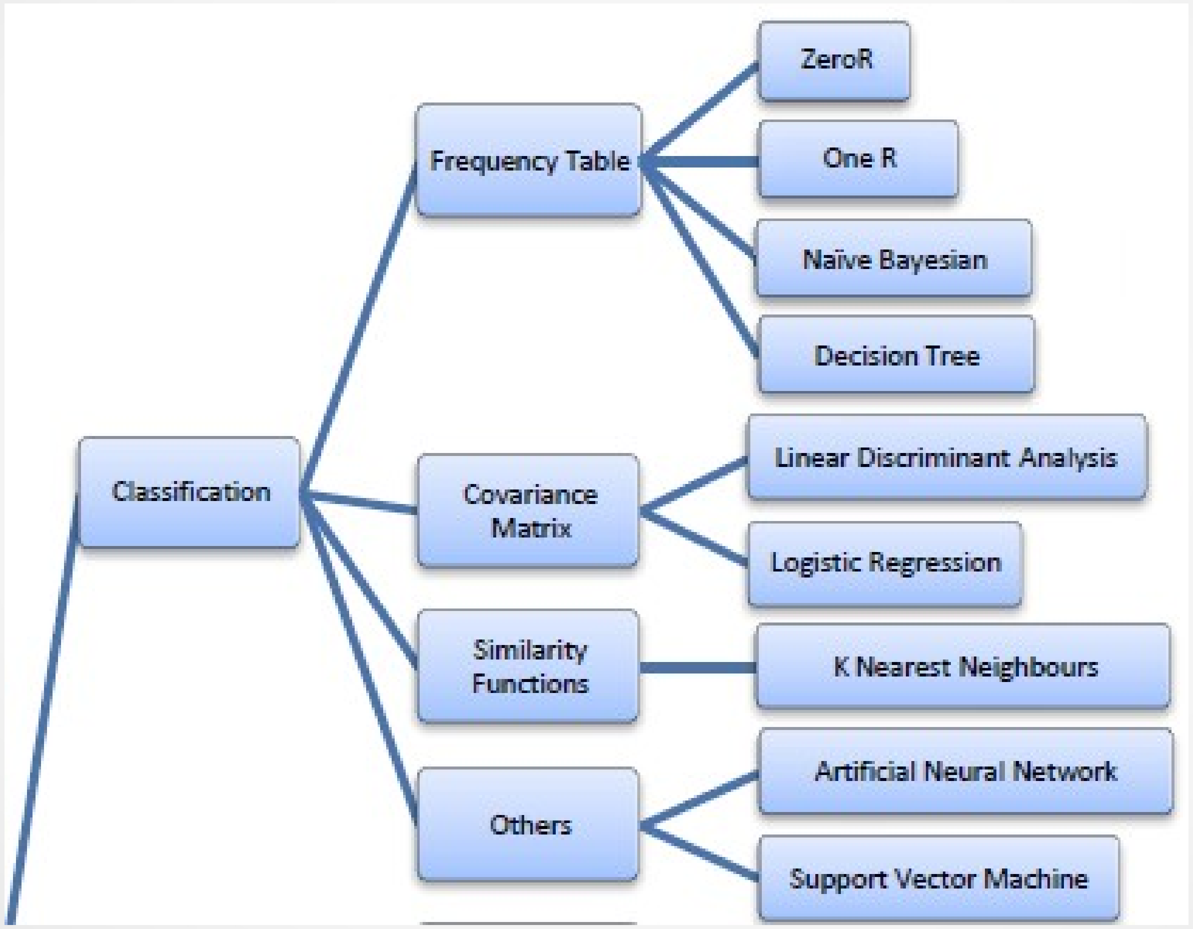
Confusion Matrix		Play Golf			
		Yes	No		
OneR	Yes	7	2	Positive Predictive Value	0.78
	No	2	3	Negative Predictive Value	0.60
		Sensitivity	Specificity	**Accuracy** = 0.71	
		0.78	0.60		

The Naive Bayesian Classifier

Numerical Attributes

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com

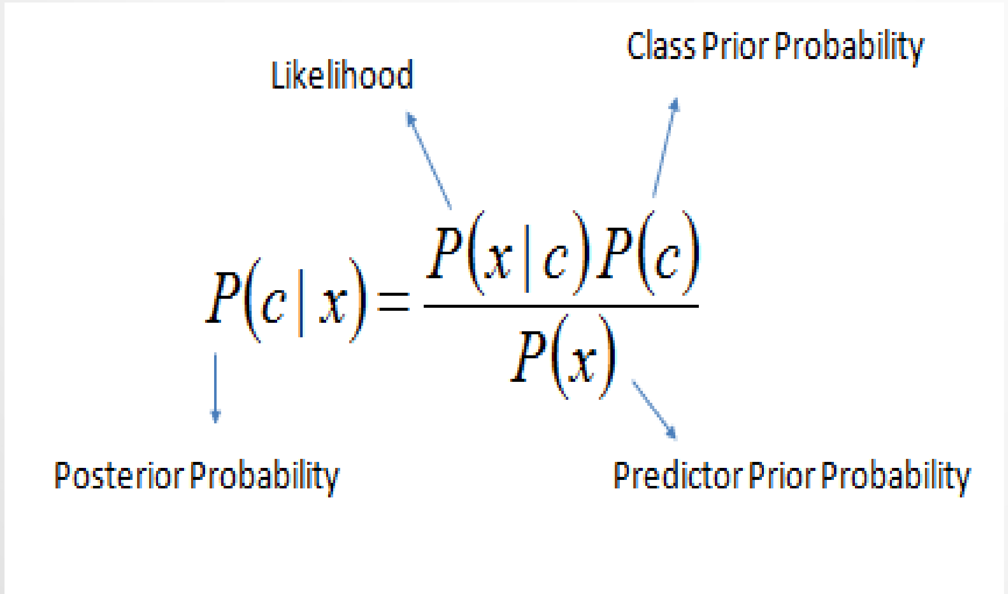

Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

The Naive Bayesian Classifier

- The Naive Bayesian classifier is based on Bayes' theorem with independence assumptions between predictors
- A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets
- Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods

How it works

- Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c)
- NB classifier assumes that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence



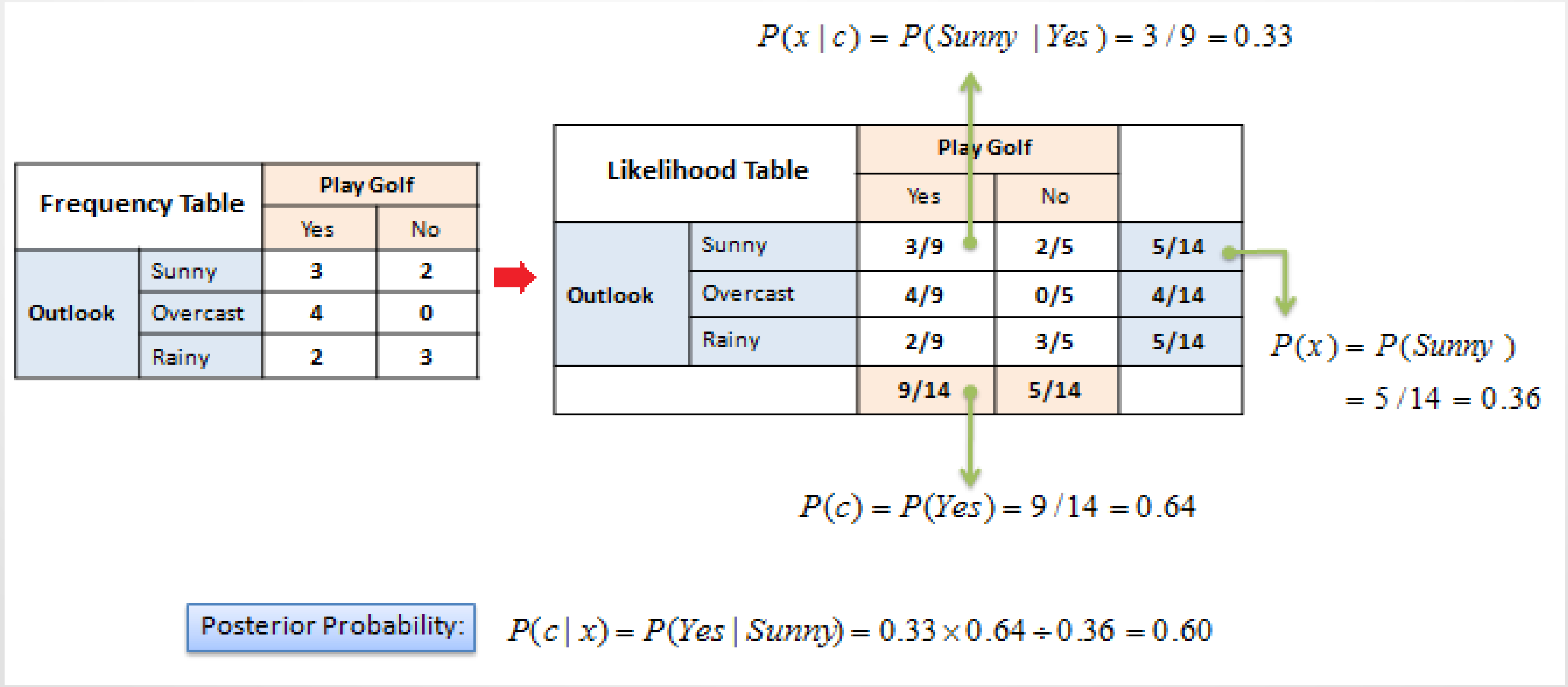
$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

- P(c|x) is the posterior probability of class (target) given predictor (attribute)
- P(c) is the prior probability of class
- P(x|c) is the likelihood which is the probability of predictor given class
- P(x) is the prior probability of predictor

Example

- The posterior probability can be calculated by first, constructing a frequency table for each attribute against the target
- Then, transforming the freq. tables to likelihood tables and finally using the Naive Bayesian equation to calculate the posterior probability for each class
- The class with the highest posterior probability is the outcome of prediction



Frequency Table		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

Likelihood Table		Play Golf		
		Yes	No	
Outlook	Sunny	3/9	2/5	5/14
	Overcast	4/9	0/5	4/14
	Rainy	2/9	3/5	5/14
		9/14	5/14	

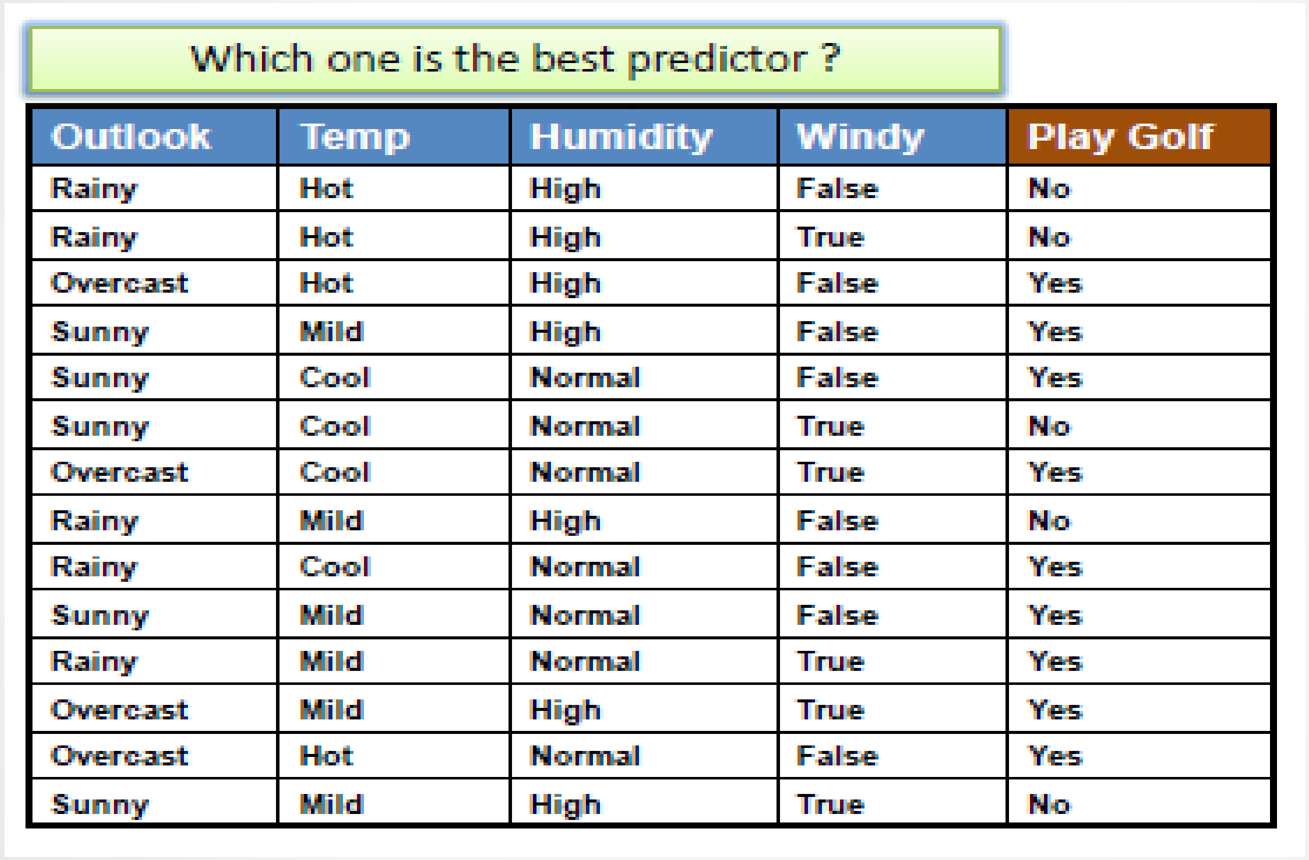
$$P(x \mid c) = P(Sunny \mid Yes) = 3/9 = 0.33$$

$$P(x) = P(Sunny) = 5/14 = 0.36$$

$$P(c) = P(Yes) = 9/14 = 0.64$$

Posterior Probability: $P(c \mid x) = P(Yes \mid Sunny) = 0.33 \times 0.64 \div 0.36 = 0.60$

Example

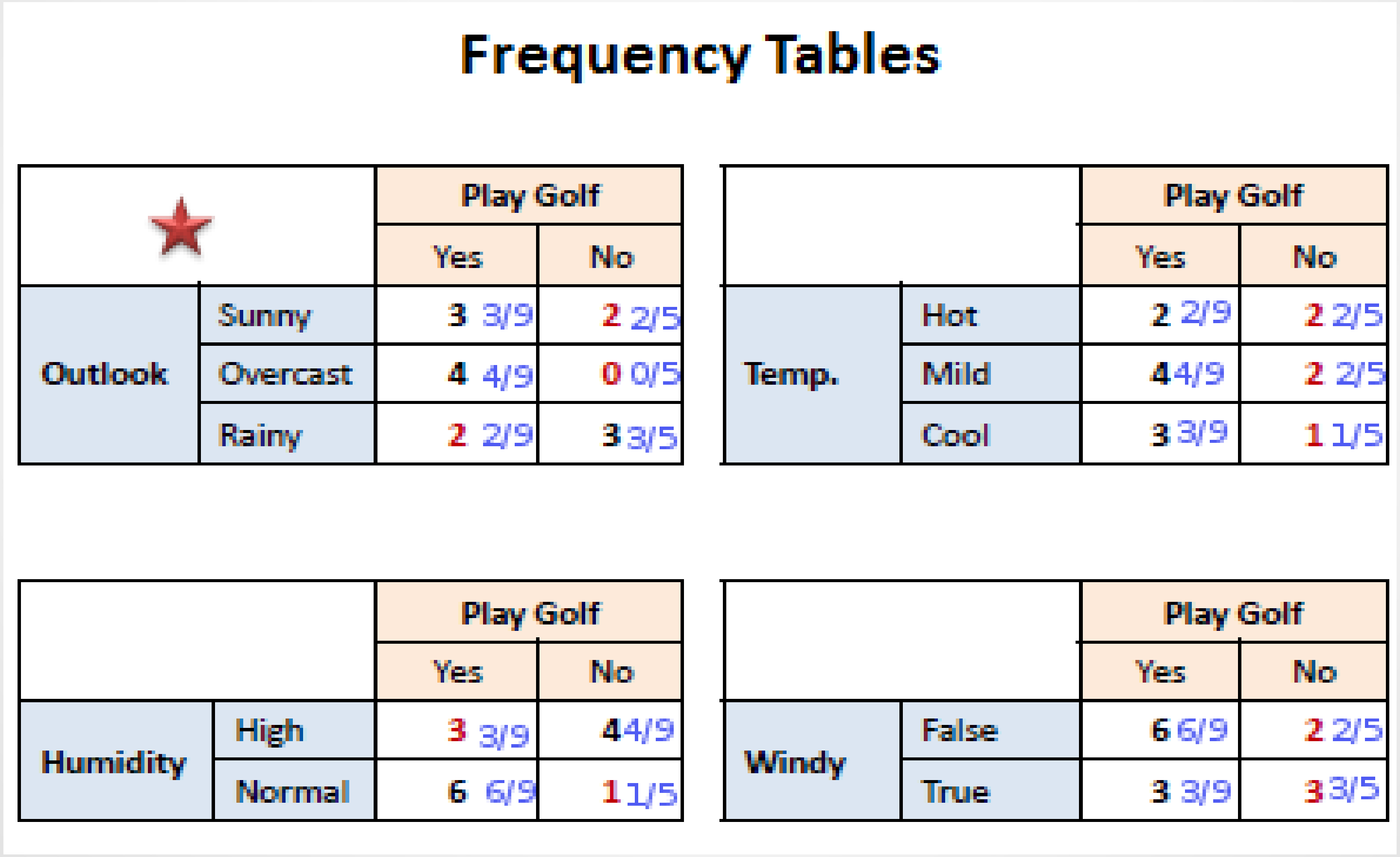


Which one is the best predictor ?

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

P(Yes) = 9 / 14
P(No) = 5 / 14

Example



Frequency Tables

★		Play Golf	
		Yes	No
Outlook	Sunny	3 3/9	2 2/5
	Overcast	4 4/9	0 0/5
	Rainy	2 2/9	3 3/5

		Play Golf	
		Yes	No
Temp.	Hot	2 2/9	2 2/5
	Mild	4 4/9	2 2/5
	Cool	3 3/9	1 1/5

		Play Golf	
		Yes	No
Humidity	High	3 3/9	4 4/9
	Normal	6 6/9	1 1/5

		Play Golf	
		Yes	No
Windy	False	6 6/9	2 2/5
	True	3 3/9	3 3/5

Example

- Let's assume we have a day with:

  Outlook = Rainy

  Temp = Mild

  Humidity = Normal

  Windy = True

  Likelihood of Yes = P(Outlook=Rainy|Yes)*P(Temp=Mild|Yes)*P(Humidity=Normal|Yes)*P(Windy=True|Yes)*P(Yes) =

  2/9 * 4/9 * 6/9 * 3/9 * 9/14 = 0.014109347

  Likelihood of No = P(Outlook=Rainy|No)* P(Temp=Mild|No)*P(Humidity=Normal|No)*P(Windy=True|No)*P(No) =

  3/5 * 2/5 * 1/5 * 3/5 * 5/14 = 0.010285714

- Now we normalize:

  P(Yes) = 0.014109347/(0.014109347+0.010285714) = 0.578368999

  P(No) = 0.010285714/(0.014109347+0.010285714) = 0.421631001

Example

- Let's assume we have a day with:

  Outlook = Rainy

  Temp = Mild

  Humidity = Normal

  Windy = True

  Likelihood of Yes = P(Outlook=Rainy|Yes)*P(Temp=Mild|Yes)*P(Humidity=Normal|Yes)*P(Windy=True|Yes)*P(Yes) =

  2/9 * 4/9 * 6/9 * 3/9 * 9/14 = 0.014109347

  Likelihood of No = P(Outlook=Rainy|No)* P(Temp=Mild|No)*P(Humidity=Normal|No)*P(Windy=True|No)*P(No) =

  3/5 * 2/5 * 1/5 * 3/5 * 5/14 = 0.010285714

- Now we normalize:

  P(Yes) = 0.014109347/(0.014109347+0.010285714) = 0.578368999

  P(Yes) = 0.010285714/(0.014109347+0.010285714) = 0.421631001

The zero-frequency problem

- When an attribute value (Outlook=Overcast) doesn’t occur with every class value (Play Golf=no)
- Add 1 to all the counts

Numerical Predictors

- Numerical variables need to be transformed to their categorical counterparts (binning) before constructing their frequency tables
- The other option we have is using the distribution of the numerical variable to have a good guess of the frequency
- For example, one common practice is to assume normal distributions for numerical variables

Normal Distribution

- The probability density function for the normal distribution is defined by two parameters (mean and standard deviation)

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Mean

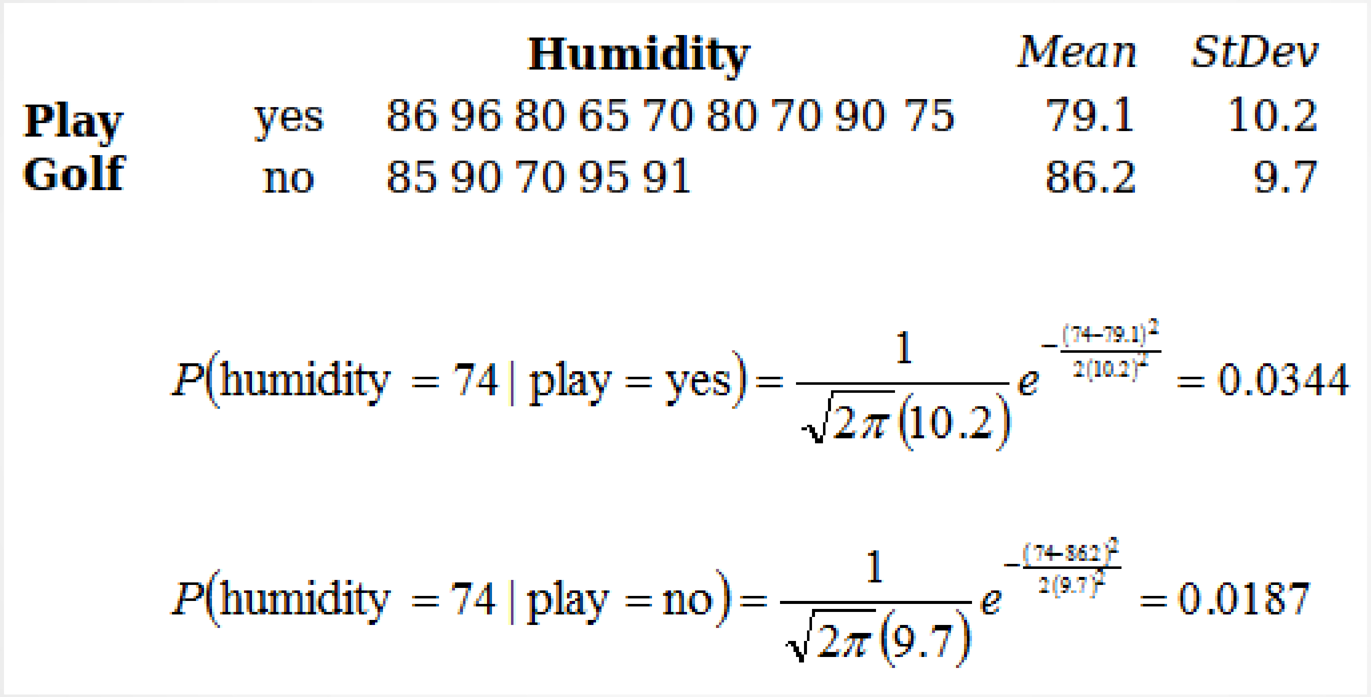
$$\sigma = \left[\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2\right]^{0.5}$$

Standard deviation

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Normal distribution

Example of Numerical Predictors



		Humidity	*Mean*	*StDev*
Play Golf	yes	86 96 80 65 70 80 70 90 75	79.1	10.2
	no	85 90 70 95 91	86.2	9.7

$$P(\text{humidity} = 74 \mid \text{play} = \text{yes}) = \frac{1}{\sqrt{2\pi}(10.2)} e^{-\frac{(74-79.1)^2}{2(10.2)^2}} = 0.0344$$

$$P(\text{humidity} = 74 \mid \text{play} = \text{no}) = \frac{1}{\sqrt{2\pi}(9.7)} e^{-\frac{(74-86.2)^2}{2(9.7)^2}} = 0.0187$$

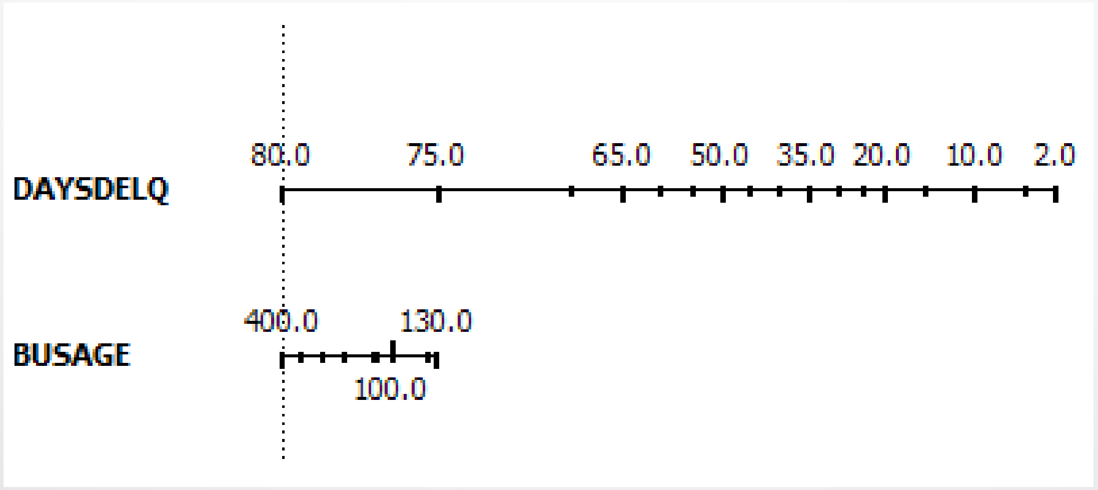
Predictors Contribution

- Kononenko's information gain as a sum of information contributed by each attribute can offer an explanation on how values of the predictors influence the class probability

$$log_2 P(c|x) - log_2 P(c)$$

Nomograms

- The contribution of predictors can also be visualized by plotting nomograms
- Nomogram plots log odds ratios for each value of each predictor
- Lengths of the lines correspond to spans of odds ratios, suggesting importance of the related predictor
- It also shows impacts of individual values of the predictor

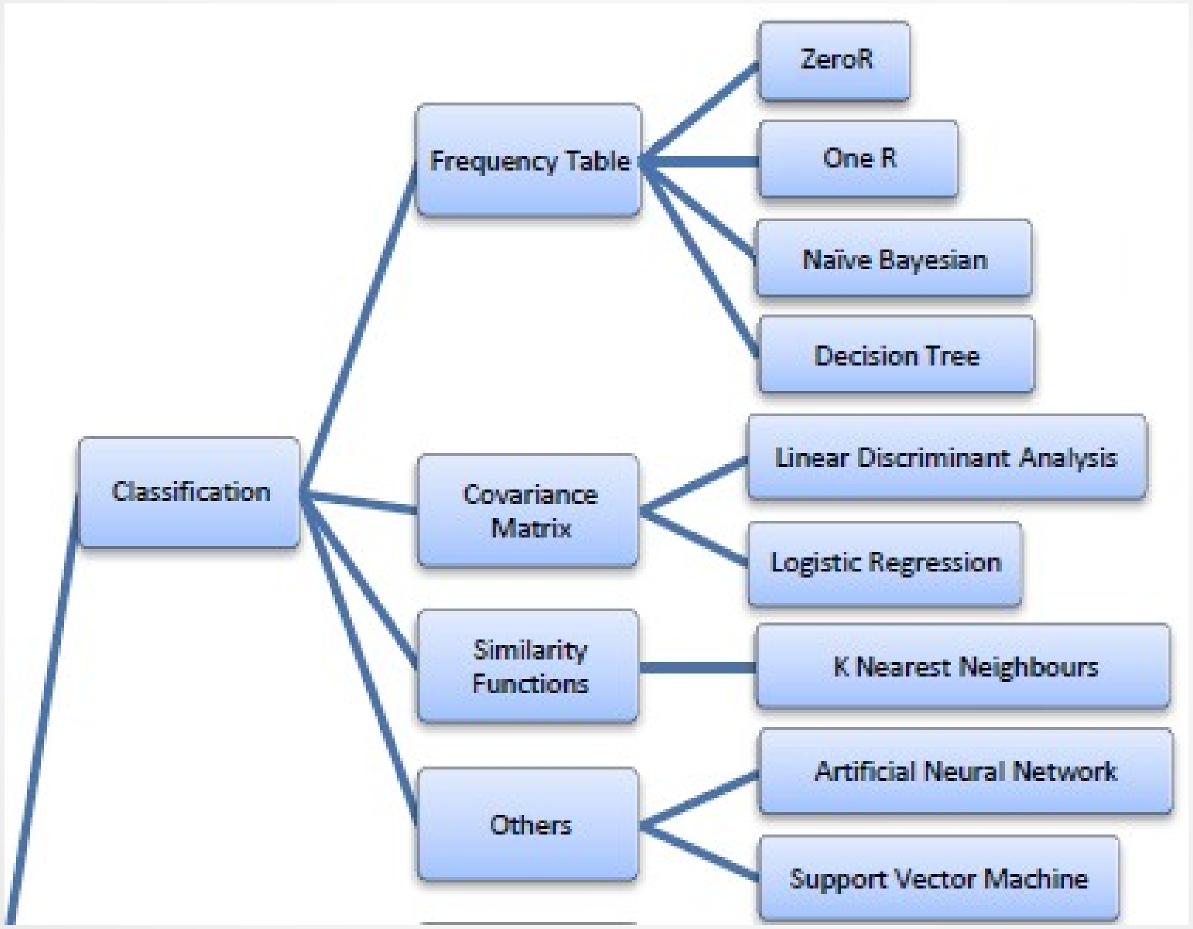


The Decision Tree Classifier

Example

Presented by:
Dr Noureddin Sadawi

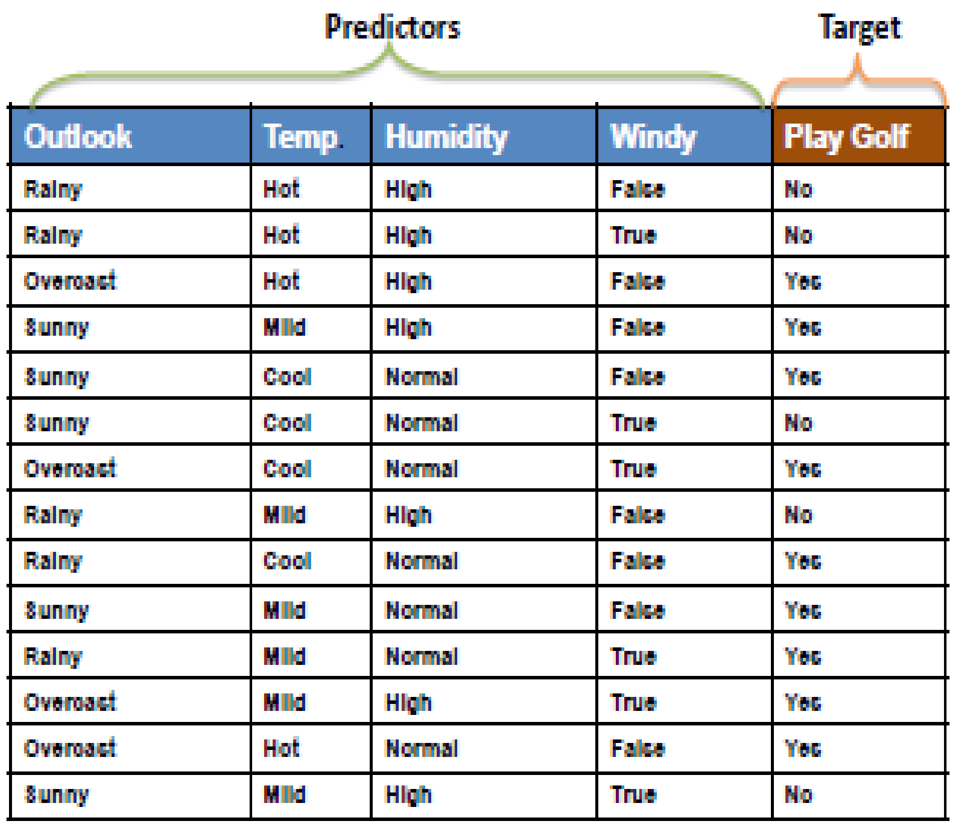
Material belongs to Prof Saed Sayad
http://www.saedsayad.com


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

Decision Tree

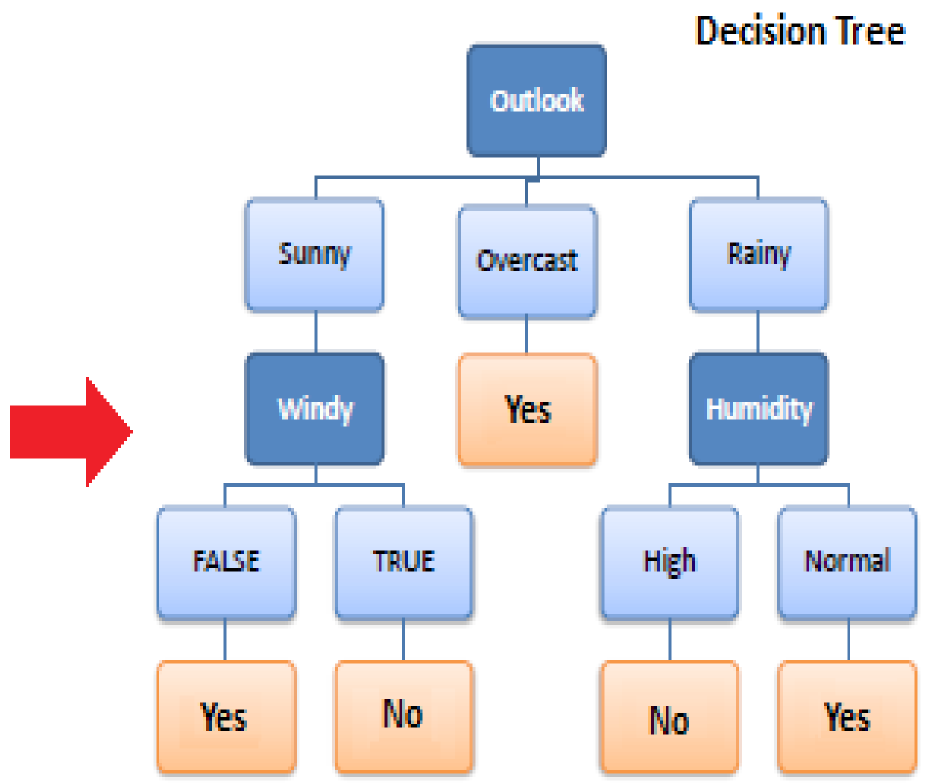
- Decision tree builds classification or regression models in the form of a tree structure
- It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed
- The final result is a tree with decision nodes and leaf nodes.
  - A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy)
  - Leaf node (e.g., Play) represents a classification or decision
- The topmost decision node in a tree which corresponds to the best predictor called root node
- Decision trees can handle both categorical and numerical data

Decision Tree for Weather Data



Predictors | Target

Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

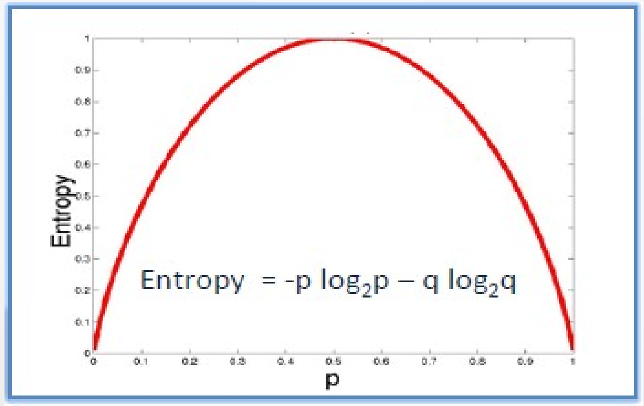


How it works

- The core algorithm for building decision trees called ID3 by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking
- ID3 uses Entropy and Information Gain to construct a decision tree

Entropy

- A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogeneous)
- ID3 algorithm uses entropy to calculate the homogeneity of a sample
- If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one



$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

$$\text{Entropy} = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

Compute Two Types of Entropy

- To build a decision tree, we need to calculate two types of entropy using frequency tables as follows:
- a) Entropy using the frequency table of one attribute (Entropy of the Target):

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5

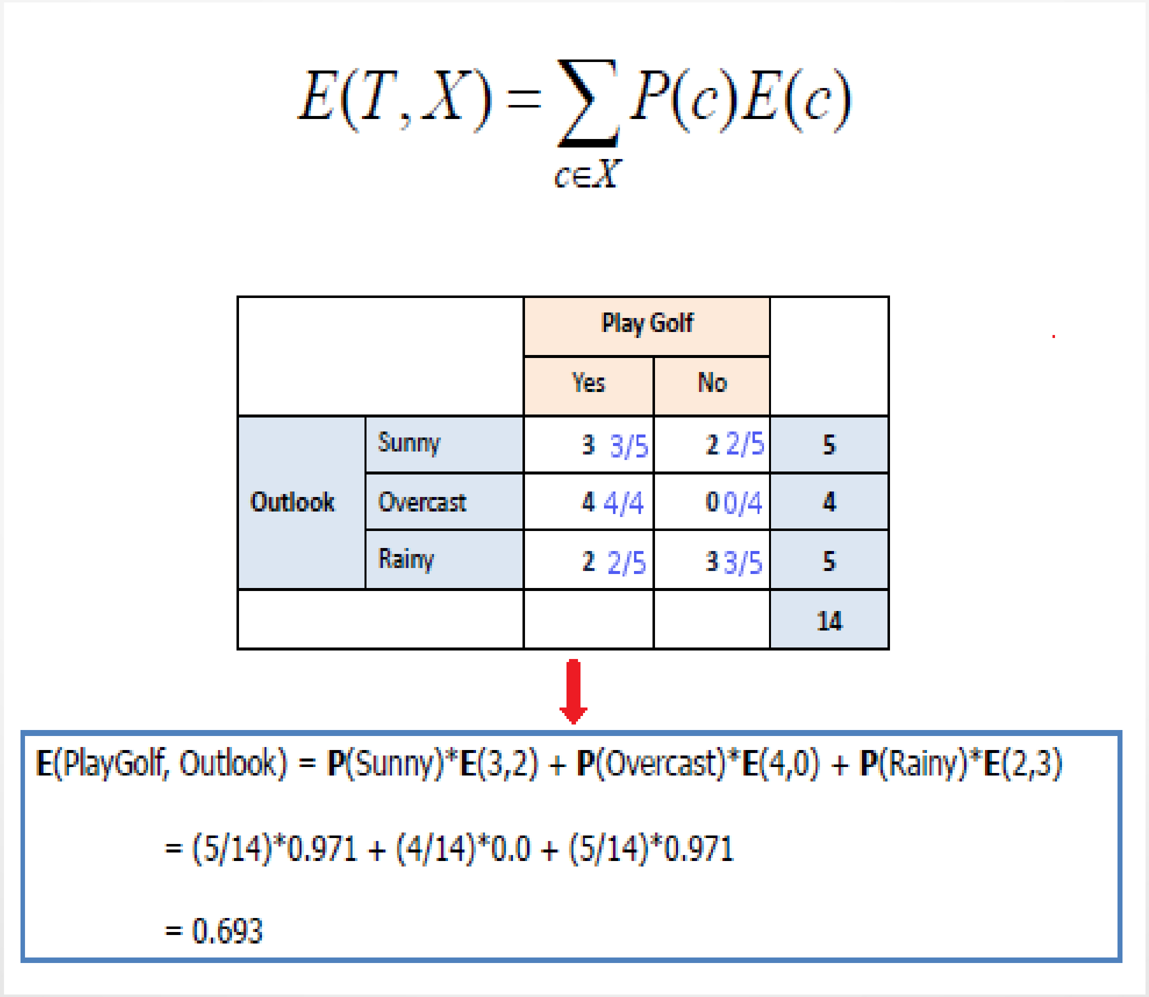
Entropy(PlayGolf) = Entropy (5,9)

$= \text{Entropy}(0.36, 0.64)$

$= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64)$

$= 0.94$

- b) Entropy using the frequency table of two attributes:



$$E(T,X)=\sum_{c\in X}P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3 3/5	2 2/5	5
	Overcast	4 4/4	0 0/4	4
	Rainy	2 2/5	3 3/5	5
				14

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Information Gain

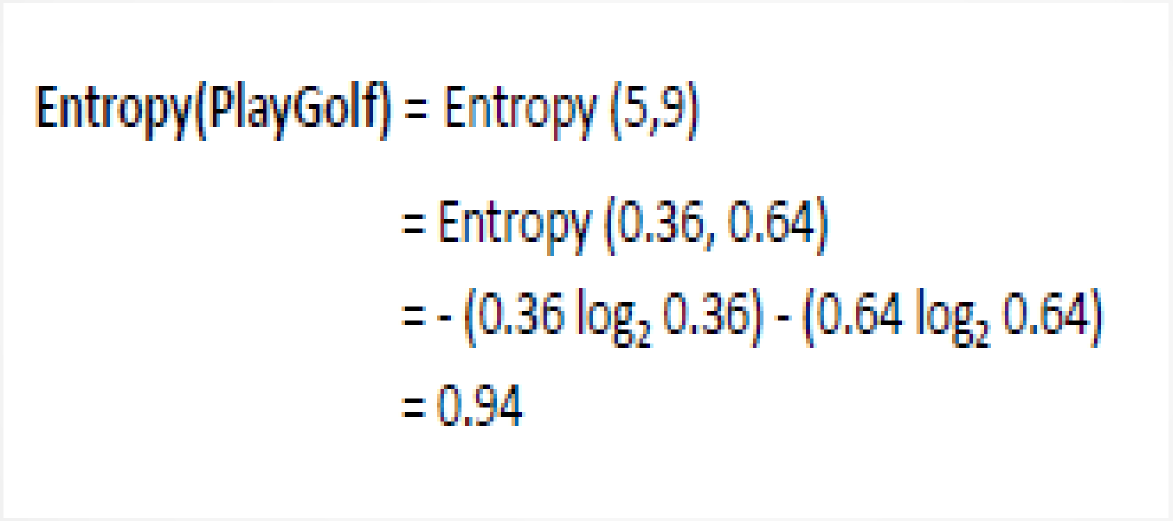
- The information gain is based on the decrease in entropy after a dataset is split on an attribute

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

- Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches)

Example

- Step 1: Calculate entropy of the target.

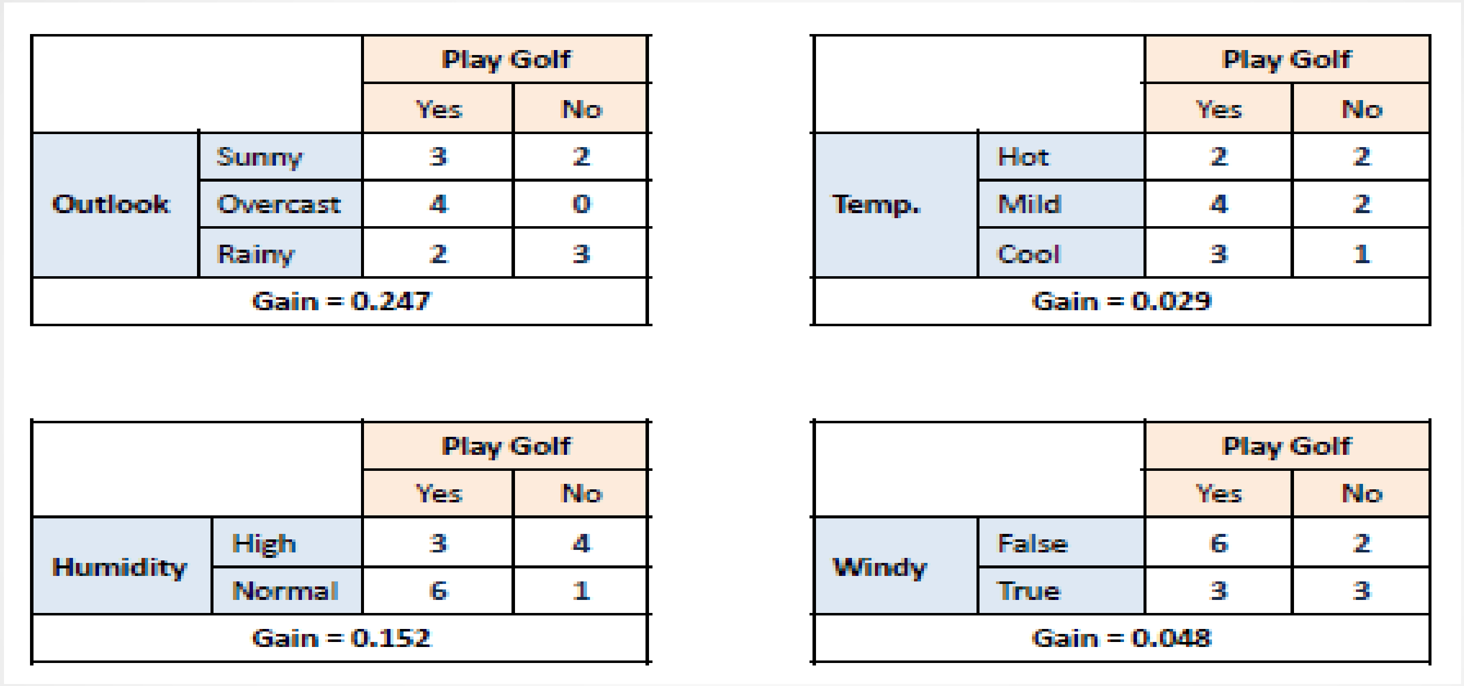


$$\begin{aligned}\text{Entropy(PlayGolf)} &= \text{Entropy }(5,9)\\ &= \text{Entropy }(0.36, 0.64)\\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64)\\ &= 0.94\end{aligned}$$

Example

- Step 2: The dataset is then split on the different attributes. The entropy for each branch is calculated.
- Then it is added proportionally, to get total entropy for the split.
- The resulting entropy is subtracted from the entropy before the split.
- The result is the Information Gain, or decrease in entropy.

Example

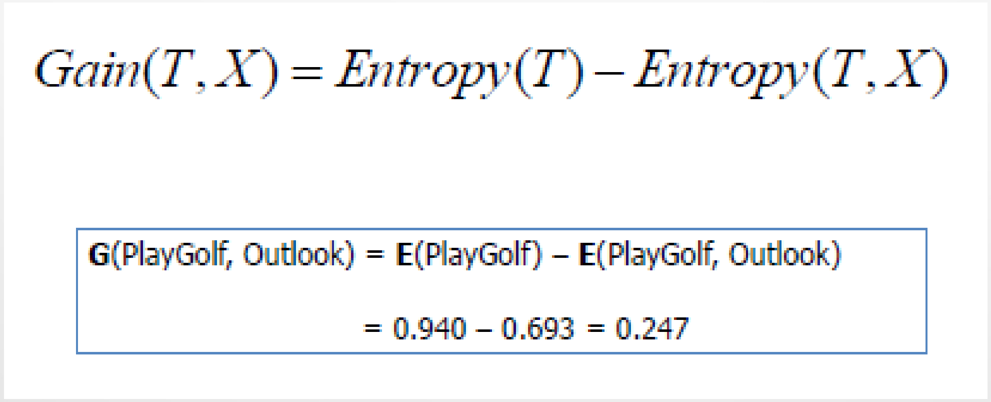


		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			



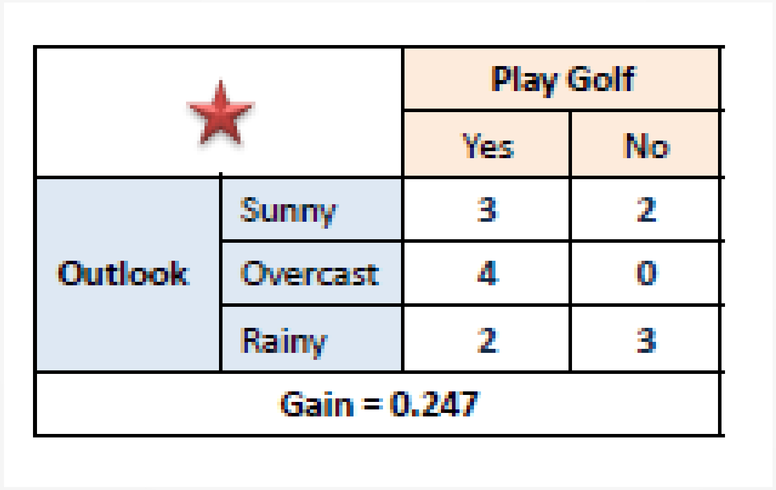
$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Example

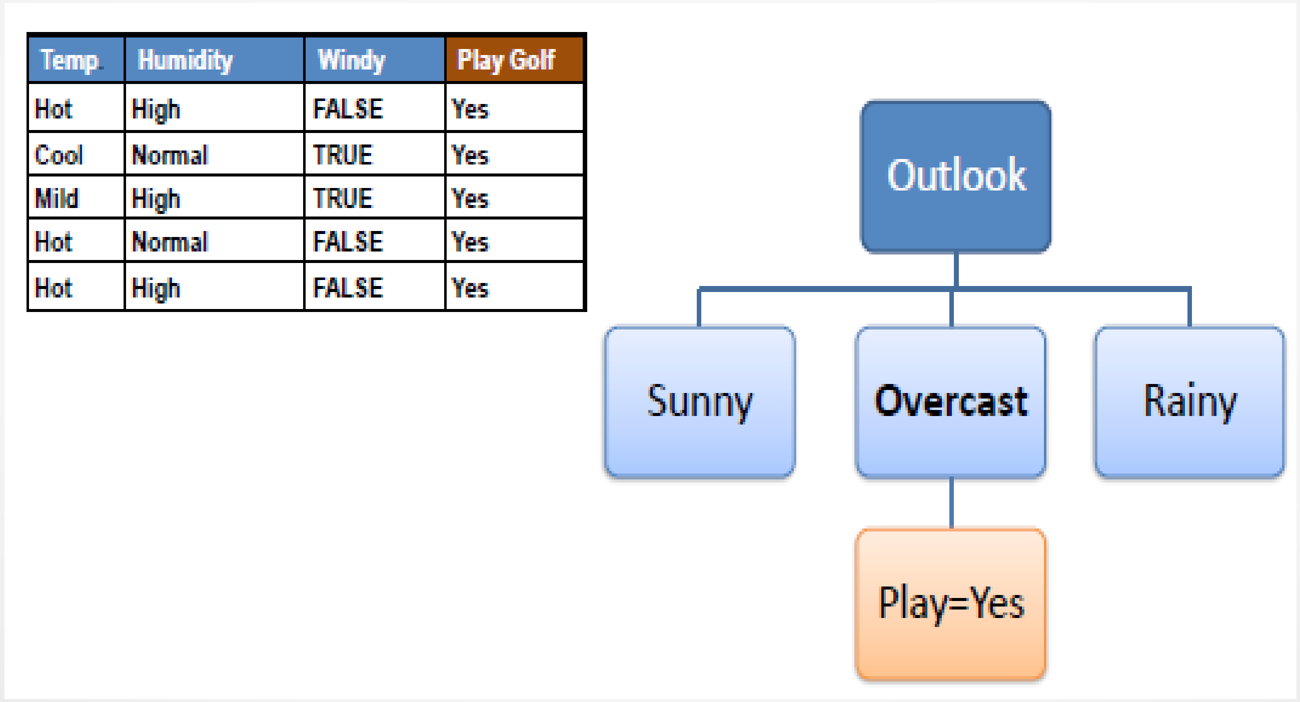
- Step 3: Choose attribute with the largest information gain as the decision node.



★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

Example

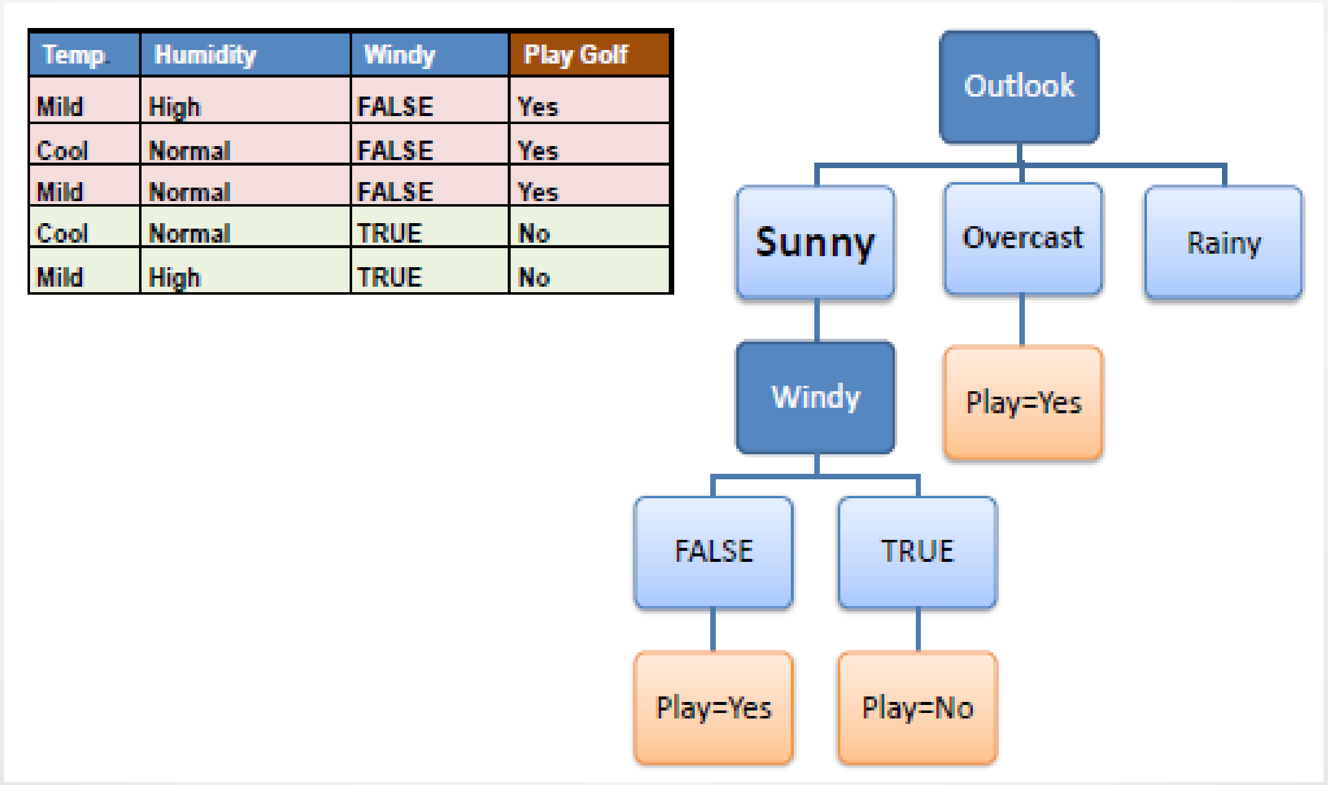
- Step 4a: A branch with entropy of 0 is a leaf node.



Temp.	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes
Hot	High	FALSE	Yes

Example

- Step 4b: A branch with entropy more than 0 needs further splitting.



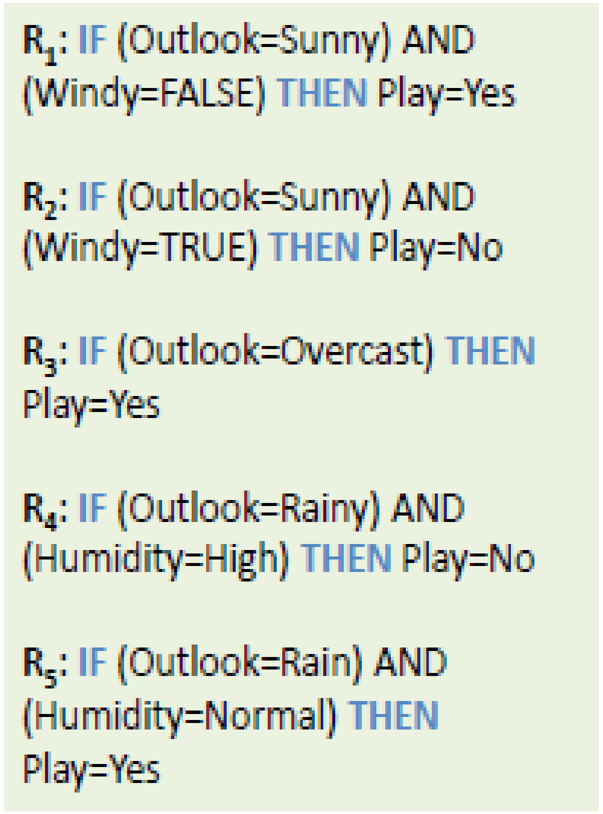
Temp.	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No

Example

- Step 5: The ID3 algorithm is run recursively on the non-leaf branches, until all data is classified.

Decision Tree to Decision Rules

- A decision tree can easily be transformed to a set of rules by mapping from the root node to the leaf nodes one by one.



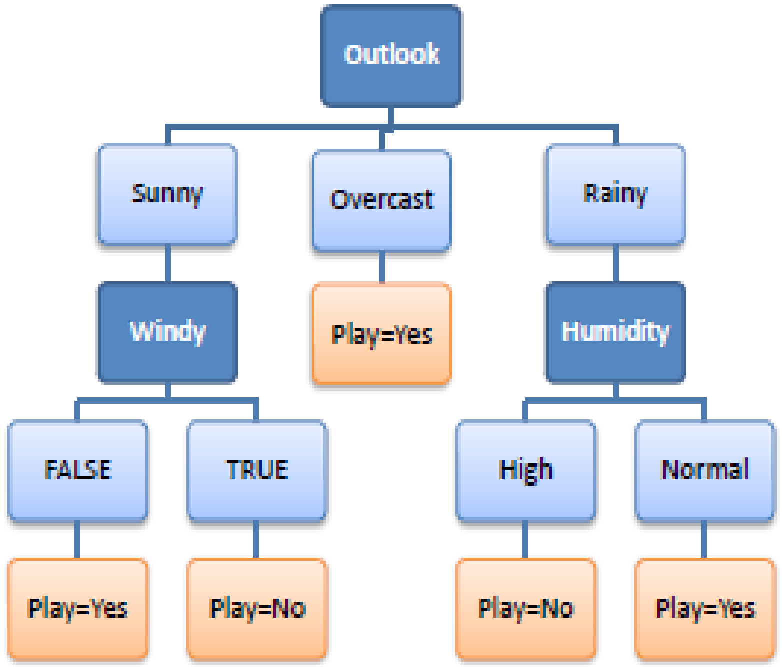
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



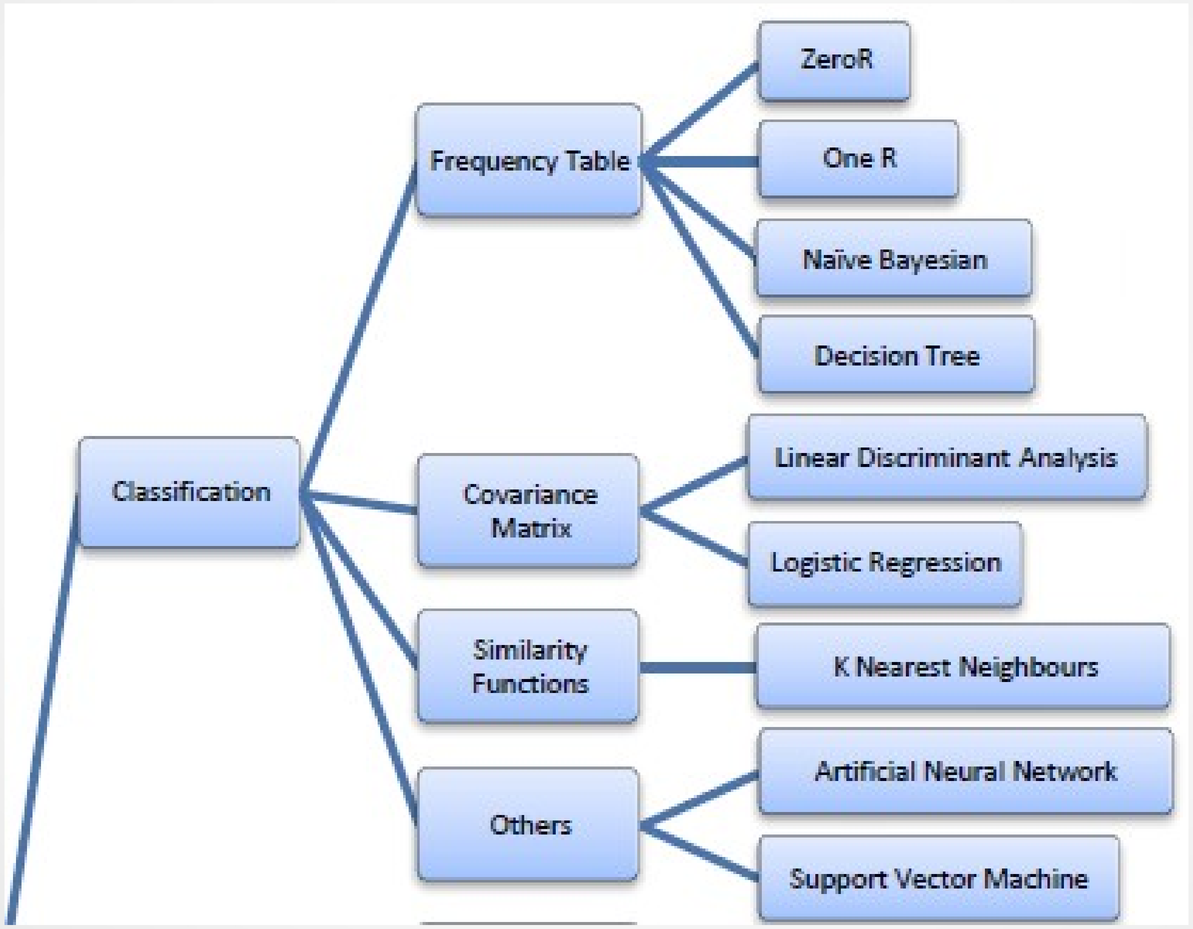
Decision Trees - Issues

- Working with continuous attributes (binning)
- Avoiding overfitting
- Super Attributes (attributes with many values)
- Working with missing values

Linear Discriminant Analysis
Classification - Example

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

- Linear Discriminant Analysis (LDA) is a classification method originally developed in 1936 by R. A. Fisher
- It is simple, mathematically robust and often produces models whose accuracy is as good as more complex methods

Algorithm

- LDA is based upon the concept of searching for a linear combination of variables (predictors) that best separates two classes (targets)
- To capture the notion of separability, Fisher defined the following score function:

$$Z = \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_d x_d$$

$$S(\beta) = \frac{\beta^T \mu_1 - \beta^T \mu_2}{\beta^T C \beta}$$

Score function

$$\Big\downarrow$$

$$S(\beta) = \frac{\overline{Z}_1 - \overline{Z}_2}{\text{Variance of Z within groups}}$$

- Given the score function, the problem is to estimate the linear coefficients that maximize the score which can be solved by the following equations:

$$\beta = C^{-1}(\mu_1 - \mu_2)$$ *Model coefficients*

$$C = \frac{1}{n_1 + n_2}(n_1 C_1 + n_2 C_2)$$ *Pooled covariance matrix*

Where:

β : *Linear model coefficients*

C_1, C_2 : *Covariance matrices*

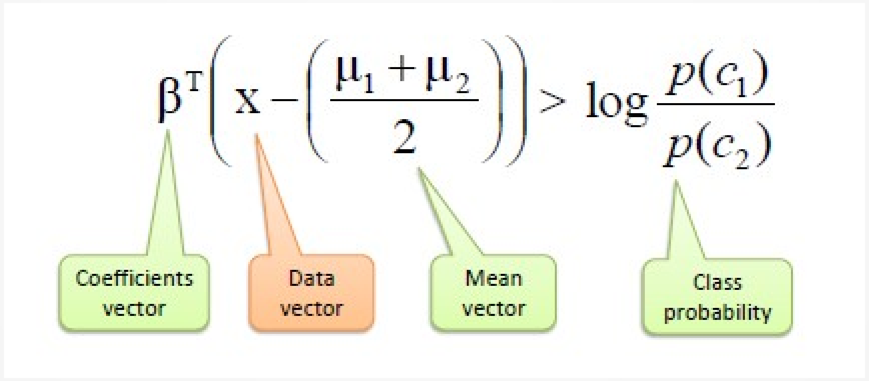
μ_1, μ_2 : *Mean vectors*

- One way of assessing the effectiveness of the discrimination is to calculate the Mahalanobis distance between two groups
- A distance greater than 3 means that in two averages differ by more than 3 standard deviations
- It means that the overlap (probability of misclassification) is quite small

$$\Delta^2 = \beta^T(\mu_1 - \mu_2)$$

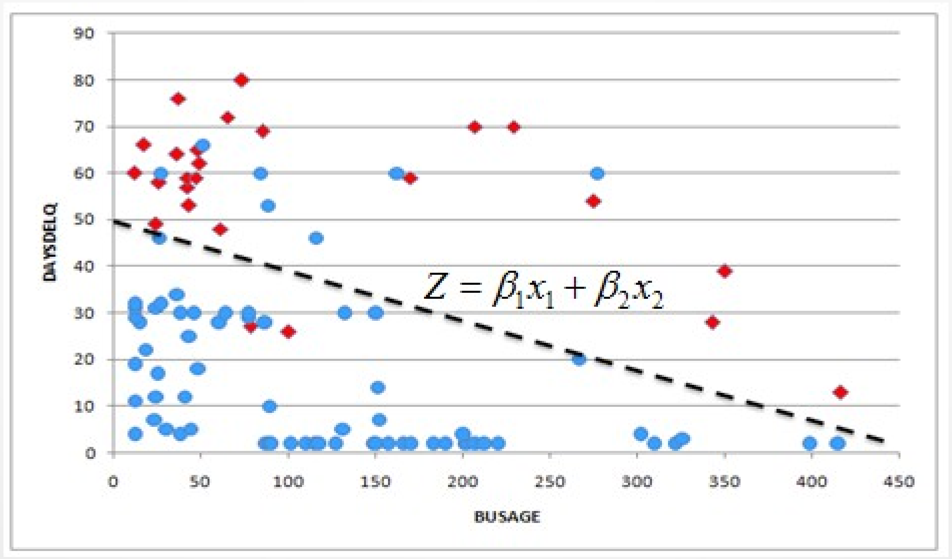
Δ : *Mahalanobis distance between two groups*

- Finally, a new point is classified by projecting it onto the maximally separating direction and classifying it as C1 if:

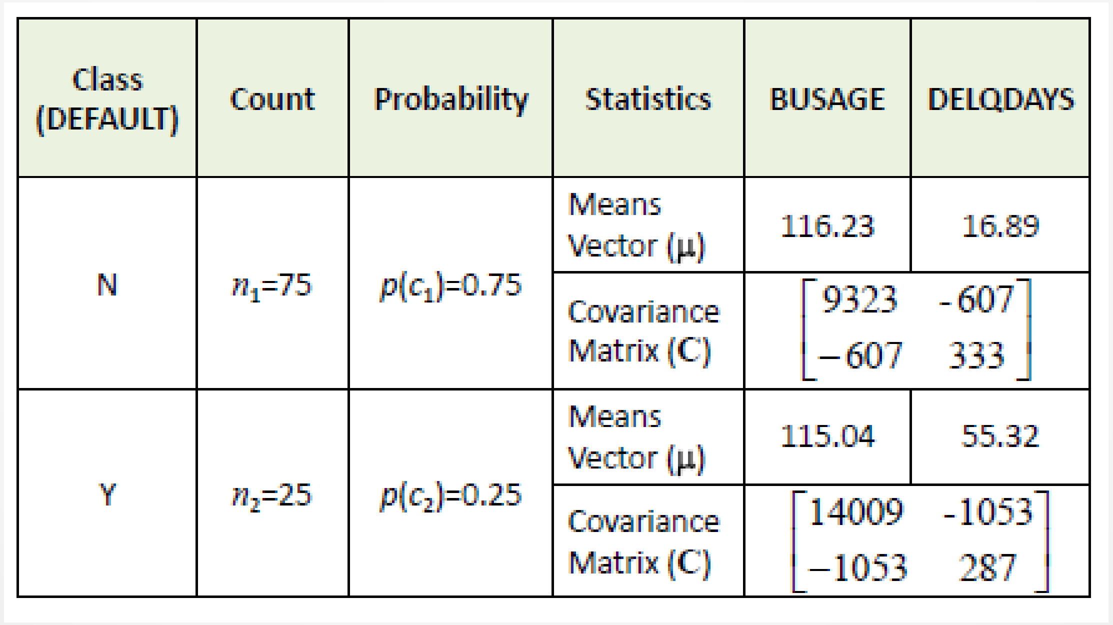

$$\beta^T\left(x-\left(\frac{\mu_1+\mu_2}{2}\right)\right) > \log\frac{p(c_1)}{p(c_2)}$$

LDA Example

- Suppose we received a dataset from a bank regarding its small business clients who defaulted (red square) and those that did not (blue circle) separated by delinquent days (DAYSDELQ) and number of months in business (BUSAGE).
- We use LDA to find an optimal linear model that best separates two classes (default and non-default).



- The first step is to calculate the mean (average) vectors, covariance matrices and class probabilities



Class (DEFAULT)	Count	Probability	Statistics	BUSAGE	DELQDAYS
N	n_1=75	$p(c_1)$=0.75	Means Vector (μ)	116.23	16.89
			Covariance Matrix (C)	$\begin{bmatrix} 9323 & -607 \\ -607 & 333 \end{bmatrix}$	
Y	n_2=25	$p(c_2)$=0.25	Means Vector (μ)	115.04	55.32
			Covariance Matrix (C)	$\begin{bmatrix} 14009 & -1053 \\ -1053 & 287 \end{bmatrix}$	

- Then, we calculate pooled covariance matrix and finally the coefficients of the linear model.

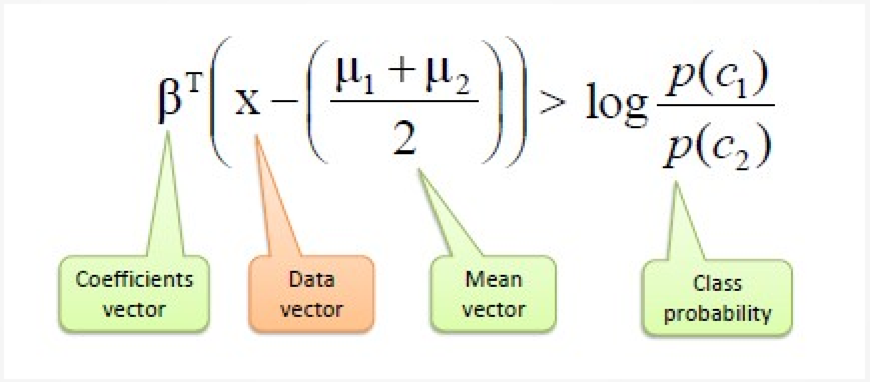
$$C = \frac{1}{n_1 + n_2}(n_1 C_1 + n_2 C_2) = \begin{bmatrix} 10495 & -718 \\ -718 & 322 \end{bmatrix}$$

$$\beta = C^{-1}(\mu_1 - \mu_2) = \begin{bmatrix} -0.0095 & -0.1408 \end{bmatrix}$$

$$Z = -0.0095\ \mathit{BUSAGE} - 0.1408\ \mathit{DAYSDELQ}$$

- Assume we have a point with BUSAGE = 111 and DAYSDELQ = 24

  x = [111   24]


$\beta^T\left(x-\left(\frac{\mu_1+\mu_2}{2}\right)\right) > \log\frac{p(c_1)}{p(c_2)}$
Coefficients vector
Data vector
Mean vector
Class probability

$$\begin{bmatrix} -0.0095 \\ -0.1408 \end{bmatrix} \left(\begin{bmatrix} 111 & 24 \end{bmatrix} \left(\frac{\begin{bmatrix} 116.23 & 16.89 \end{bmatrix} + \begin{bmatrix} 115.04 & 55.32 \end{bmatrix}}{2} \right) \right)$$

$$\overset{?}{>} \log \frac{0.75}{0.25}$$

- A Mahalanobis distance of 2.32 shows a small overlap between two groups which means a good separation between classes by the linear model.

$$\Delta^2 = \beta^T(\mu_1 - \mu_2) = 5.40$$
$$\Delta = 2.32$$

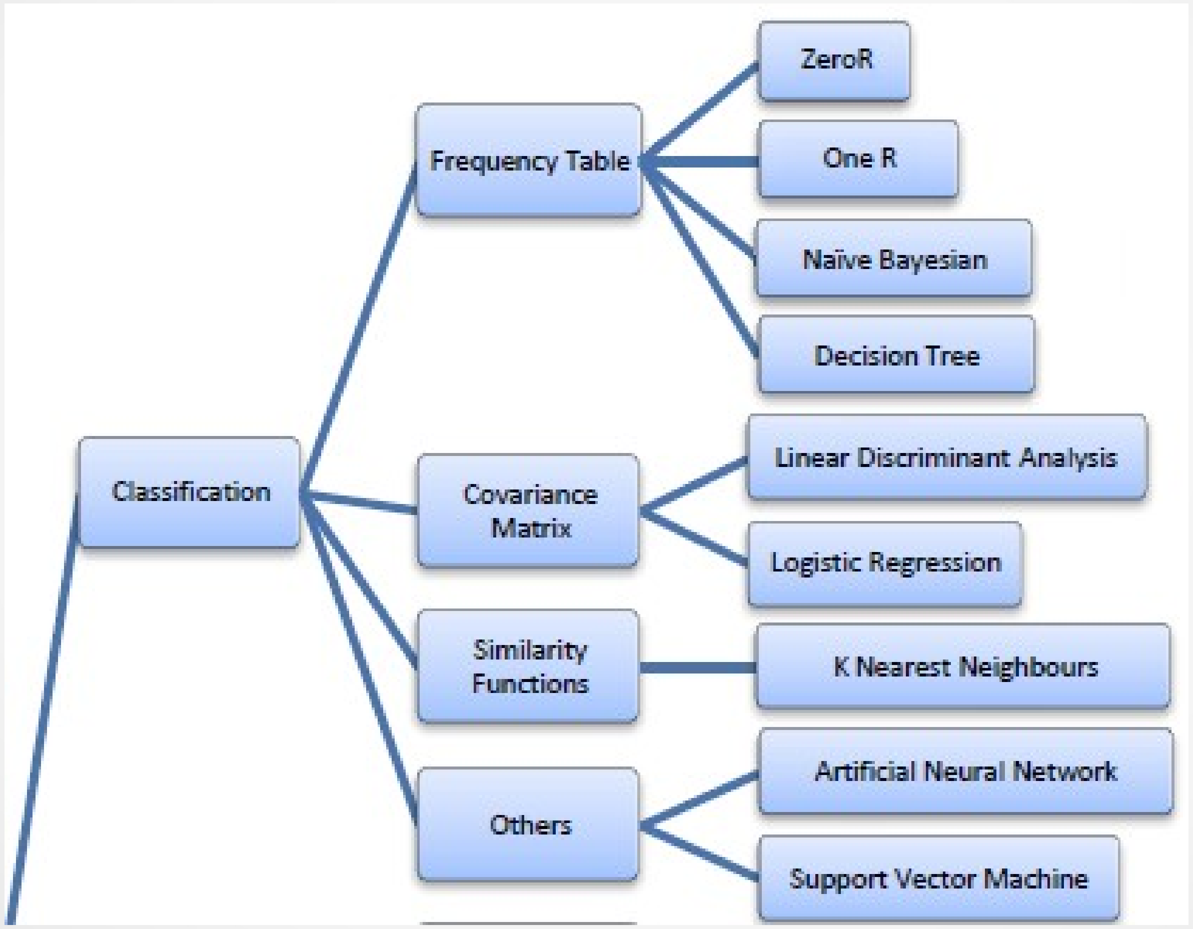
Predictors Contribution

- A simple linear correlation between the model scores and predictors can be used to test which predictors contribute significantly to the discriminant function.
- Correlation varies from -1 to 1, with -1 and 1 meaning the highest contribution but in different directions and 0 means no contribution at all.

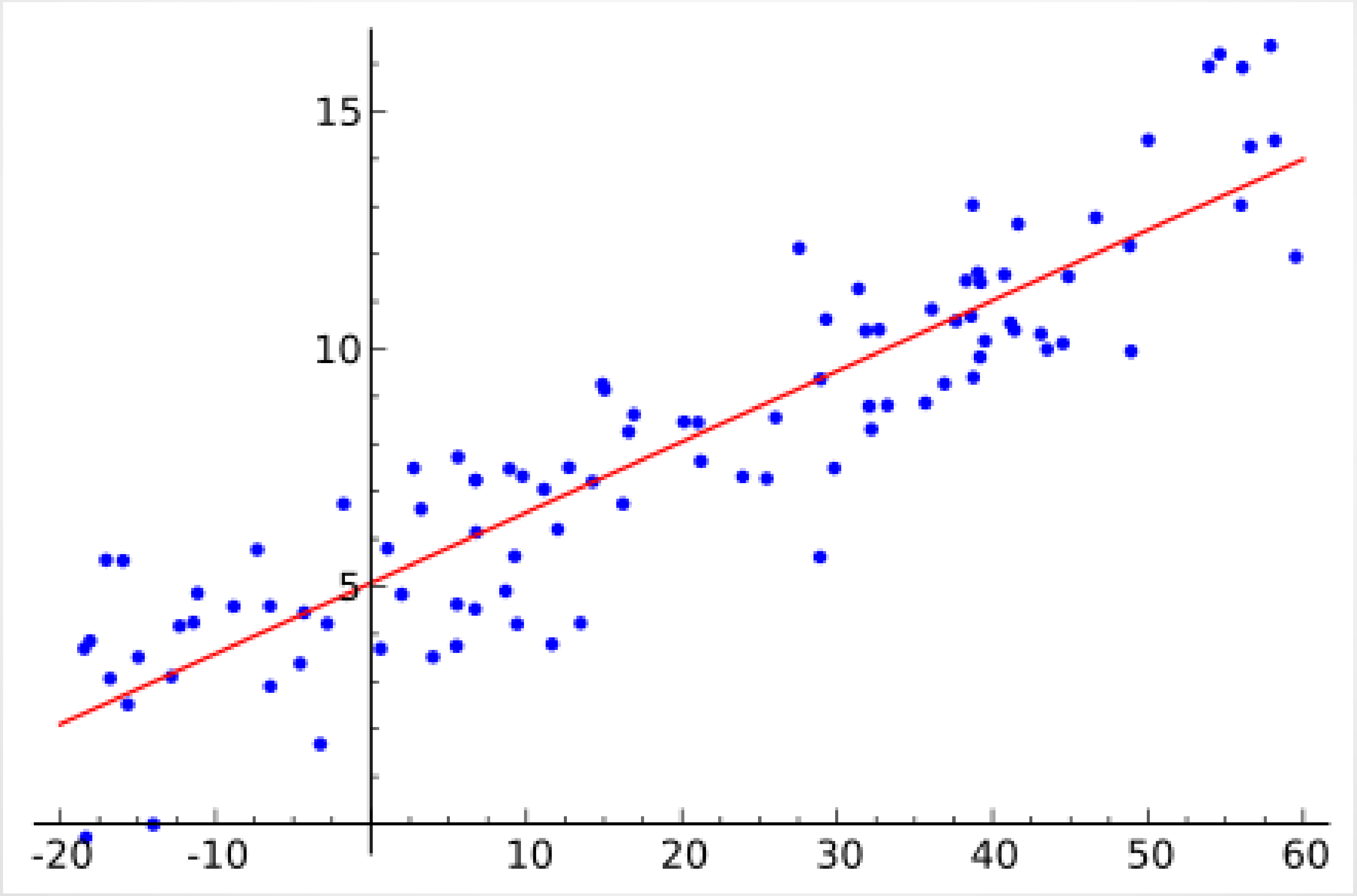
Logistic Regression

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

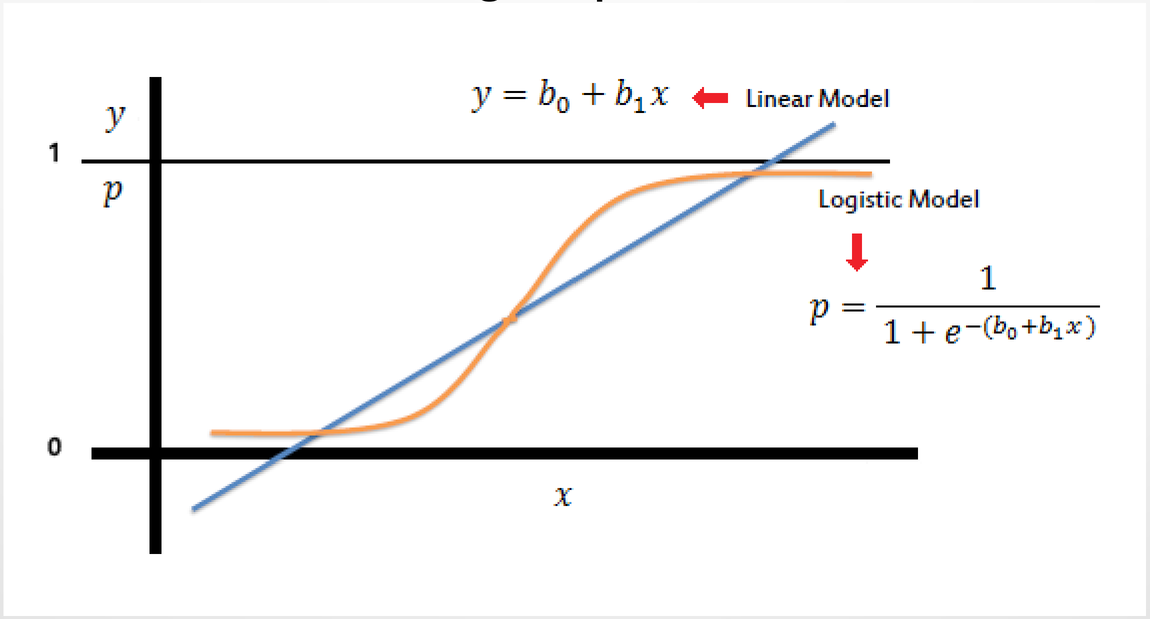
Linear Regression



Logistic Regression Overview

- Logistic regression predicts the probability of an outcome that can only have two values (i.e. a dichotomy)
- The prediction is based on the use of one or several predictors (numerical and categorical)
- A linear regression is not appropriate for predicting the value of a binary variable for two reasons:
  - A linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1)
  - Since the dichotomous experiments can only have one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line

- Logistic regression produces a logistic curve, which is limited to values between 0 and 1
- Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the “odds” of the target variable, rather than the probability
- Moreover, the predictors do not have to be normally distributed or have equal variance in each group.



- In the logistic regression the constant (b0) moves the curve left and right and the slope (b1) defines the steepness of the curve
- By simple transformation, the logistic regression equation can be written in terms of an odds ratio:

$$\frac{p}{1-p} = \exp(b_0 + b_1 x)$$

- Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors

$$ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

- The coefficient (b1) is the amount the logit (log-odds) changes with a one unit change in x

- As mentioned before, logistic regression can handle any number of numerical and/or categorical variables

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}}$$

Linear Regression & Logistic Regression

- There are several analogies between linear regression and logistic regression
- Just as ordinary least square regression is the method used to estimate coefficients for the best fit line in linear regression, logistic regression uses maximum likelihood estimation (MLE) to obtain the model coefficients that relate predictors to the target
- After this initial function is estimated, the process is repeated until LL (Log Likelihood) does not change significantly

$$\beta^1 = \beta^0 + [X^T W X]^{-1} . X^T (y - \mu)$$

β is a vector of the logistic regression coefficients.

W is a square matrix of order N with elements $n_i \pi_i (1 - \pi_i)$ on the diagonal and zeros everywhere else.

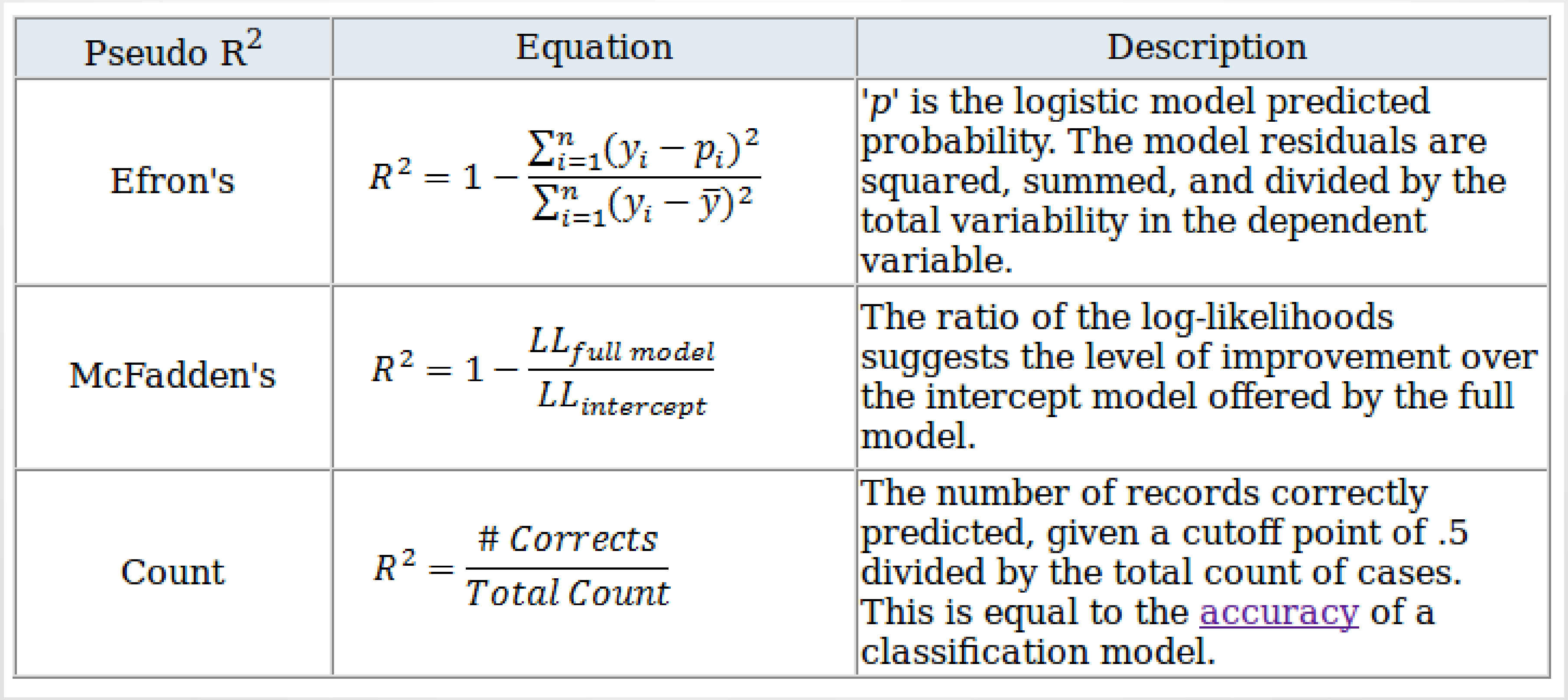
μ is a vector of length N with elements $\mu_i = n_i \pi_i$.

Pseudo R-squared

- A pseudo R-squared value is also available to indicate the adequacy of the regression model
- Likelihood ratio test is a test of the significance of the difference between the likelihood ratio for the baseline model minus the likelihood ratio for a reduced model
- This difference is called "model chi-square“
- Wald test is used to test the statistical significance of each coefficient (b) in the model (i.e., predictors contribution)

Pseudo R-squared

- There are several measures intended to mimic the R-squared analysis to evaluate the goodness-of-fit of logistic models, but they cannot be interpreted as one would interpret an R-squared and different pseudo R-squared can arrive at very different values. Here we discuss three pseudo R-squared measures



Pseudo R^2	Equation	Description
Efron's	$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - p_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$	'*p*' is the logistic model predicted probability. The model residuals are squared, summed, and divided by the total variability in the dependent variable.
McFadden's	$R^2 = 1 - \frac{LL_{full\ model}}{LL_{intercept}}$	The ratio of the log-likelihoods suggests the level of improvement over the intercept model offered by the full model.
Count	$R^2 = \frac{\#\ Corrects}{Total\ Count}$	The number of records correctly predicted, given a cutoff point of .5 divided by the total count of cases. This is equal to the accuracy of a classification model.

Likelihood Ratio Test

- The likelihood ratio test provides the means for comparing the likelihood of the data under one model (e.g., full model) against the likelihood of the data under another, more restricted model (e.g., intercept model)

$$LL = \sum_{i=1}^{n} y_i ln(p_i) + (1 - y_i) ln(1 - p_i)$$

where 'p' is the logistic model predicted probability

- The next step is to calculate the difference between these two log-likelihoods

$$2(LL_1 - LL_2)$$

- The difference between two likelihoods is multiplied by a factor of 2 in order to be assessed for statistical significance using standard significance levels (Chi-squared test)
- The degrees of freedom for the test will equal the difference in the number of parameters being estimated under the models (e.g., full and intercept)

Wald test

- A Wald test is used to evaluate the statistical significance of each coefficient (b) in the model

$$W_j = \frac{b_j}{SE_{b_j}}$$

- where W is the Wald's statistic with a normal distribution (like Z-test), b is the coefficient and SE is its standard error
- The W value is then squared, yielding a Wald statistic with a chi-square distribution

$$SE_j = sqrt(diag(H^{-1})_j)$$

$$H = [X^T W X]$$

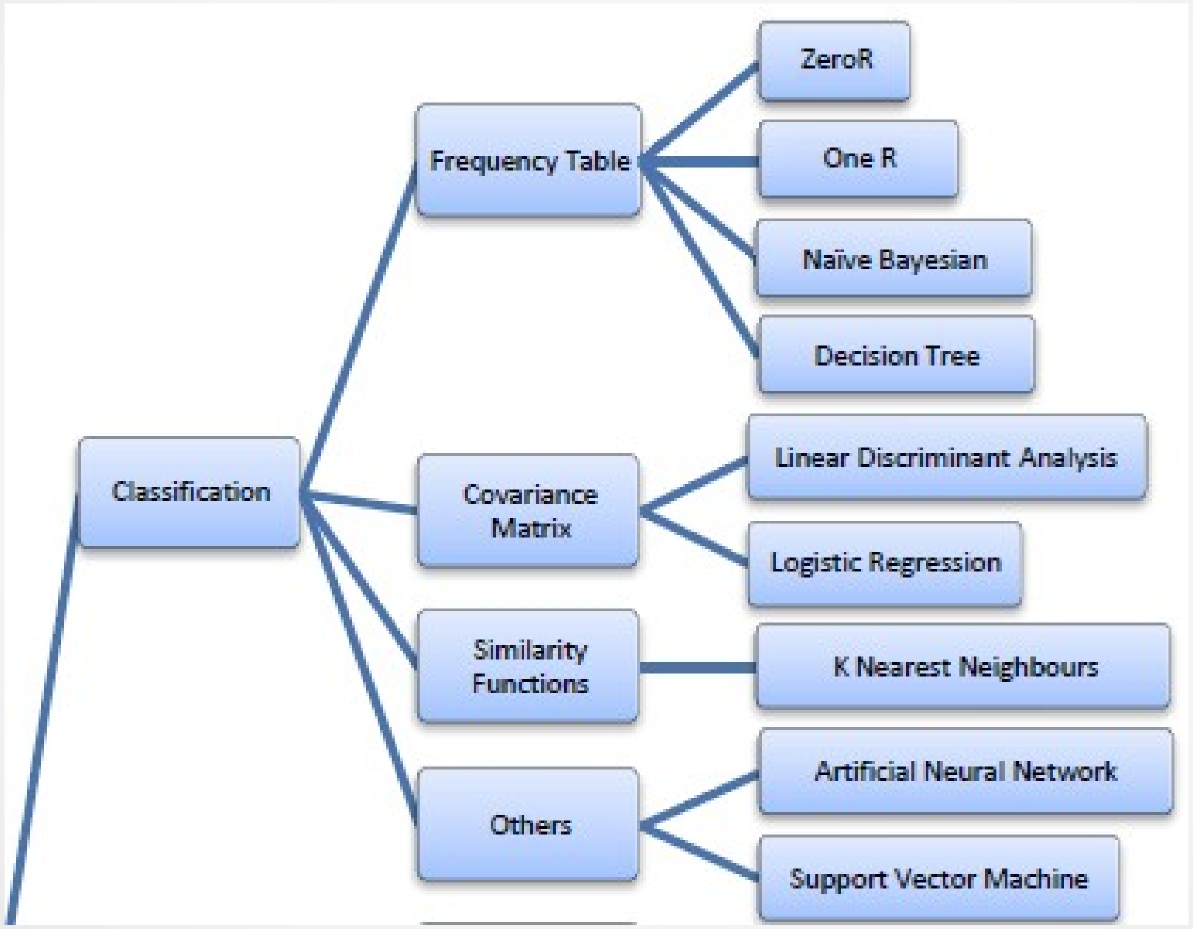
Predictors Contributions

- The Wald test is usually used to assess the significance of prediction of each predictor
- Another indicator of contribution of a predictor is exp(b) or odds-ratio of coefficient which is the amount the logit (log-odds) changes, with a one unit change in the predictor (x)

K Nearest Neighbors Classification

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com

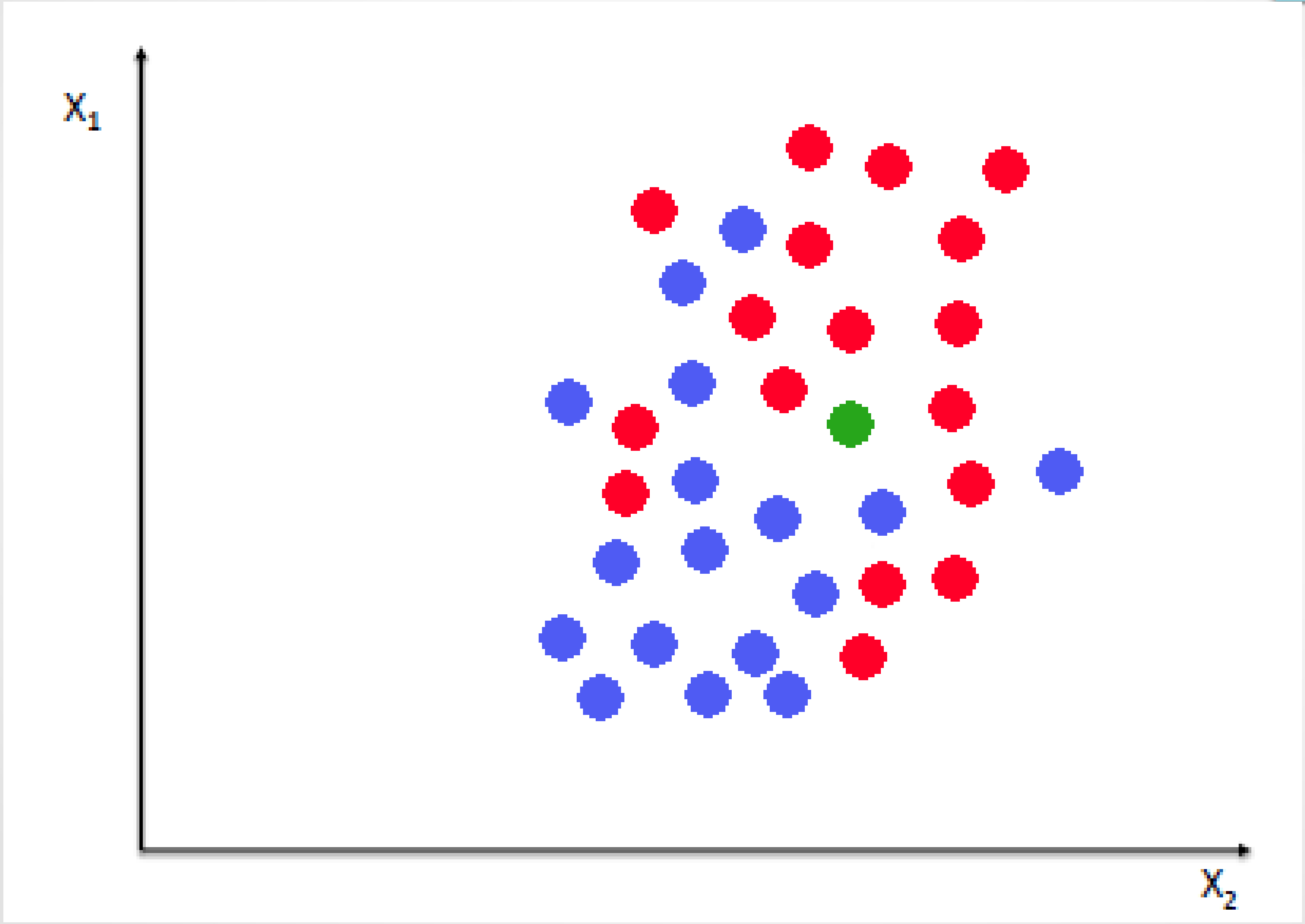

Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

- K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions)
- KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's

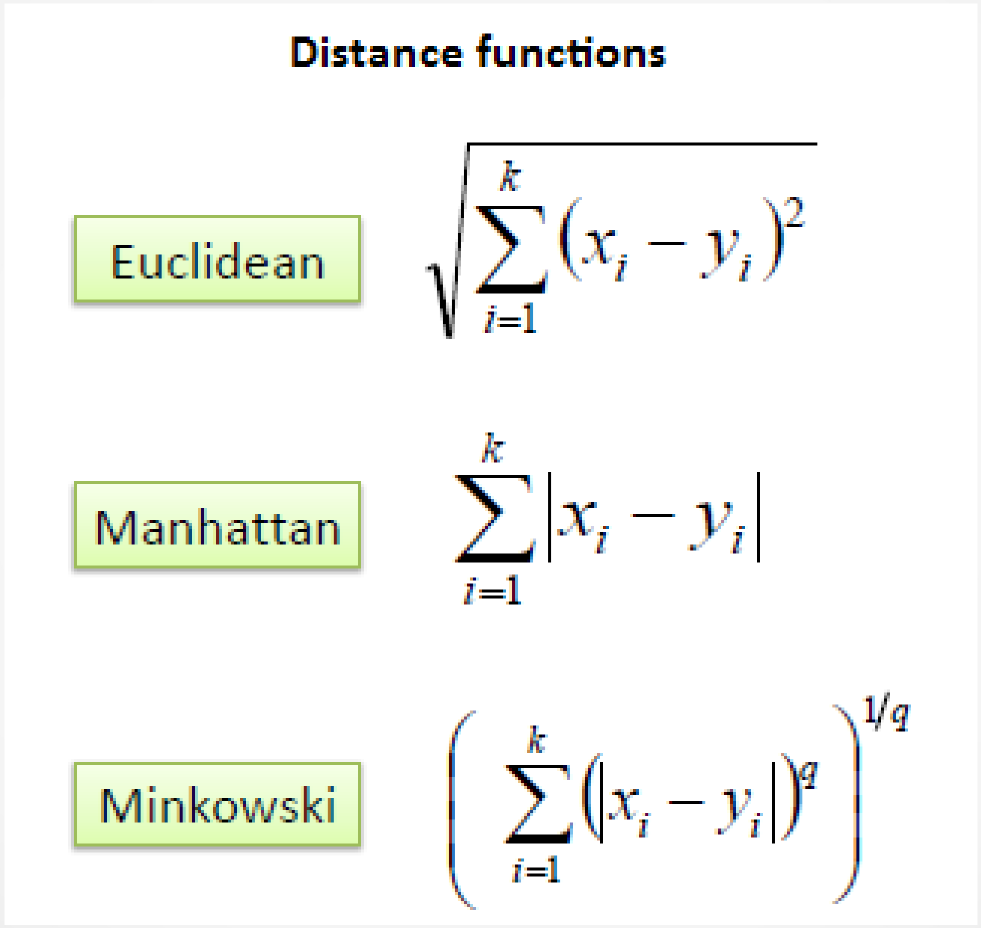
Algorithm

- A case is classified by a majority vote of its neighbors, with the case being assigned to the class most common amongst its K nearest neighbors measured by a distance function
- If K = 1, then the case is simply assigned to the class of its nearest neighbor

Diagram



Distance measures for cont. variables



Distance functions

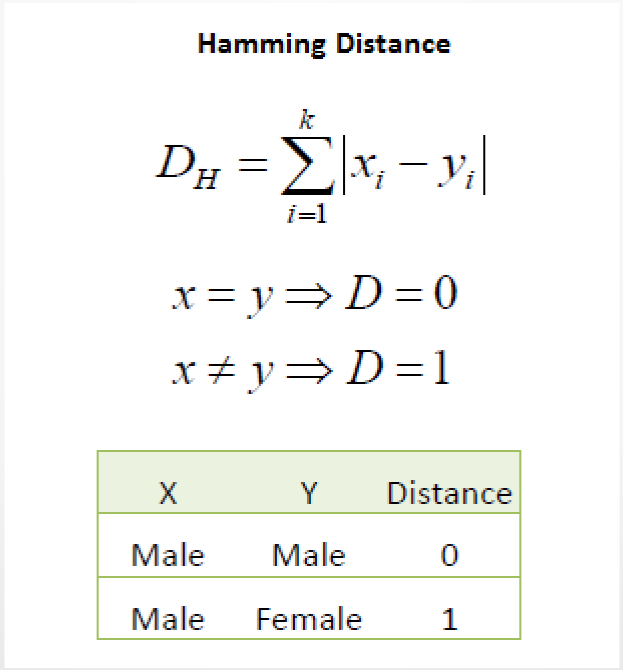
Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$$

Categorical Variables

- In the instance of categorical variables the Hamming distance must be used
- It also brings up the issue of standardization of the numerical variables between 0 and 1 when there is a mixture of numerical and categorical variables in the dataset



Hamming Distance

$$D_H = \sum_{i=1}^{k} \left| x_i - y_i \right|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

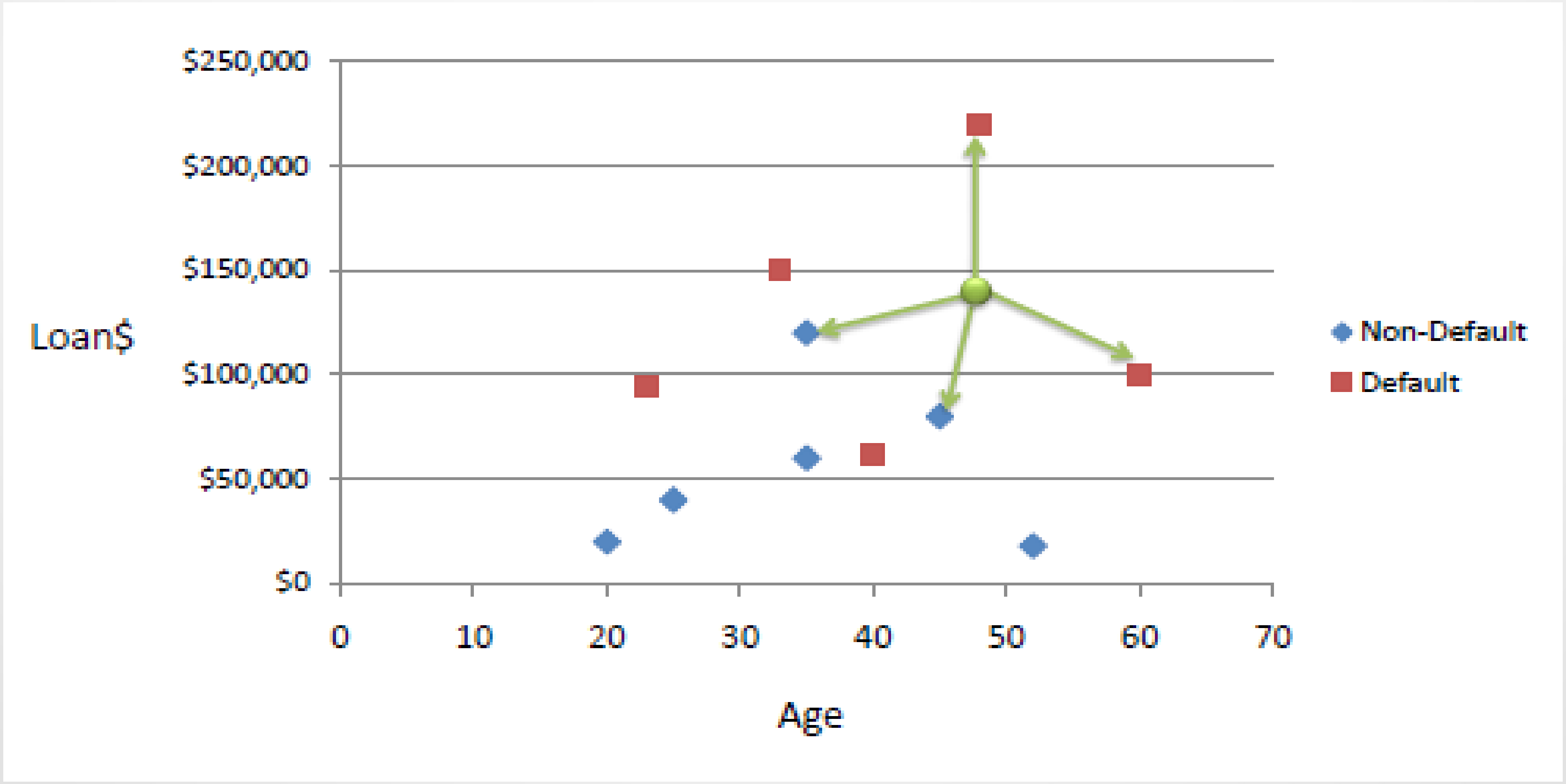
X	Y	Distance
Male	Male	0
Male	Female	1

How many neighbors?

- Choosing the optimal value for K is best done by first inspecting the data
- In general, a large K value is more precise as it reduces the overall noise but there is no guarantee
- Cross-validation is another way to retrospectively determine a good K value by using an independent dataset to validate the K value
- Historically, the optimal K for most datasets has been between 3-10. That produces much better results than 1NN

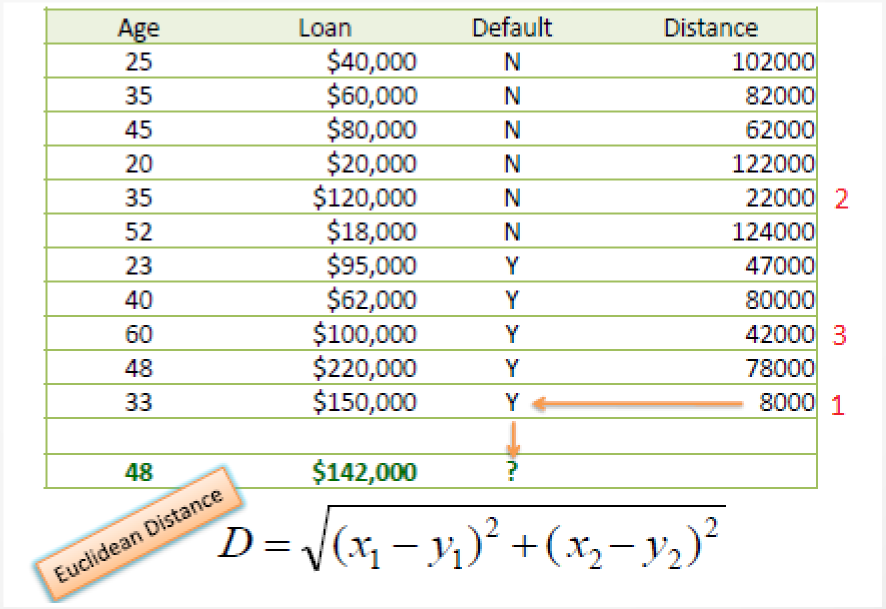
Example:

- Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target



Example

- We can now use the training set to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance.
- If K=1 then the nearest neighbor is the last case in the training set with Default=Y.
- D = Sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01 >> Default=Y



Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	**$142,000**	?		

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

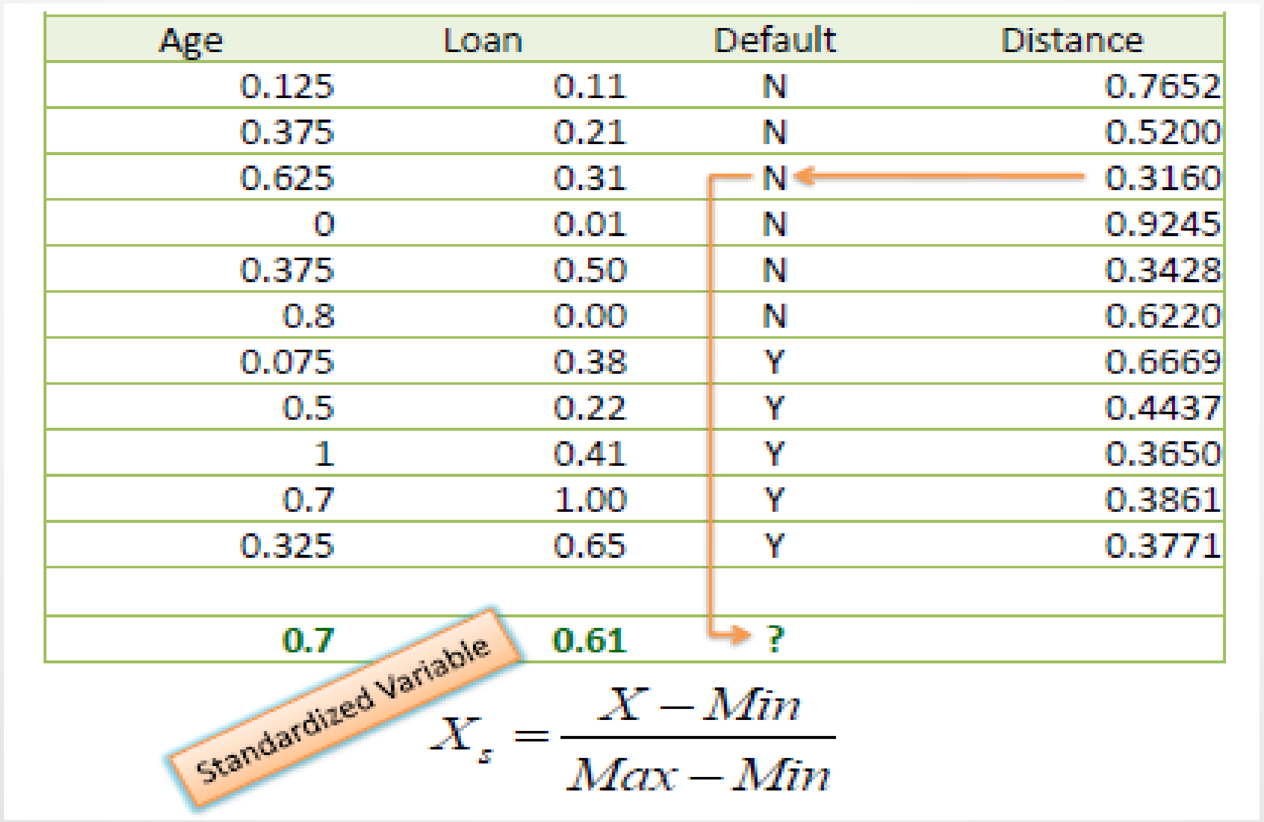
- With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y

Standardized Distance

- One major drawback in calculating distance measures directly from the training set is in the case where variables have different measurement scales or there is a mixture of numerical and categorical variables.
- For example, if one variable is based on annual income in dollars, and the other is based on age in years then income will have a much higher influence on the distance calculated.
- One solution is to standardize the training set

Standardized Distance

Using the standardized distance on the same training set, the unknown case returned a different neighbor which is not a good sign of robustness.



Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	**0.61**	**?**	

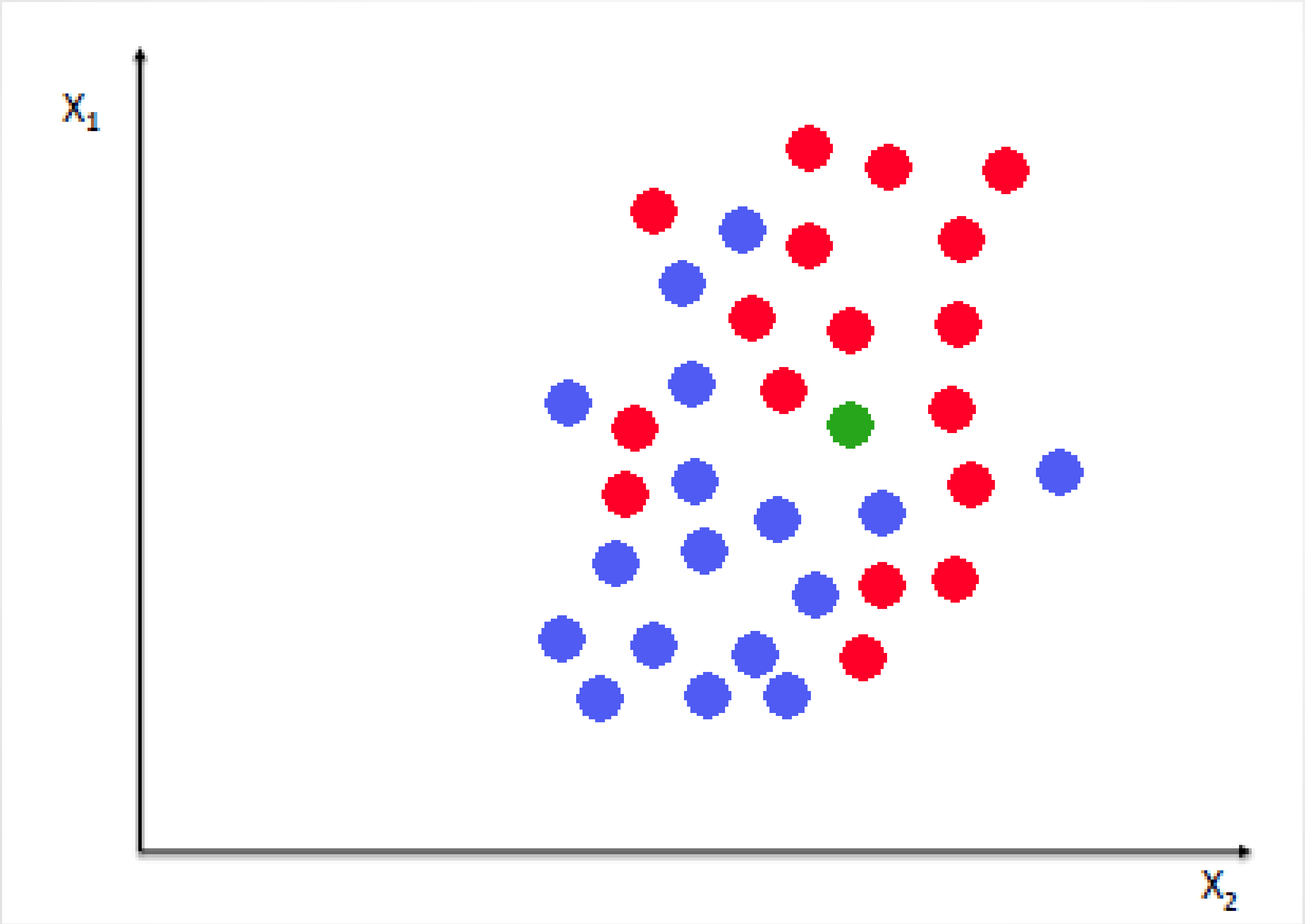
Standardized Variable

$$X_s = \frac{X - Min}{Max - Min}$$

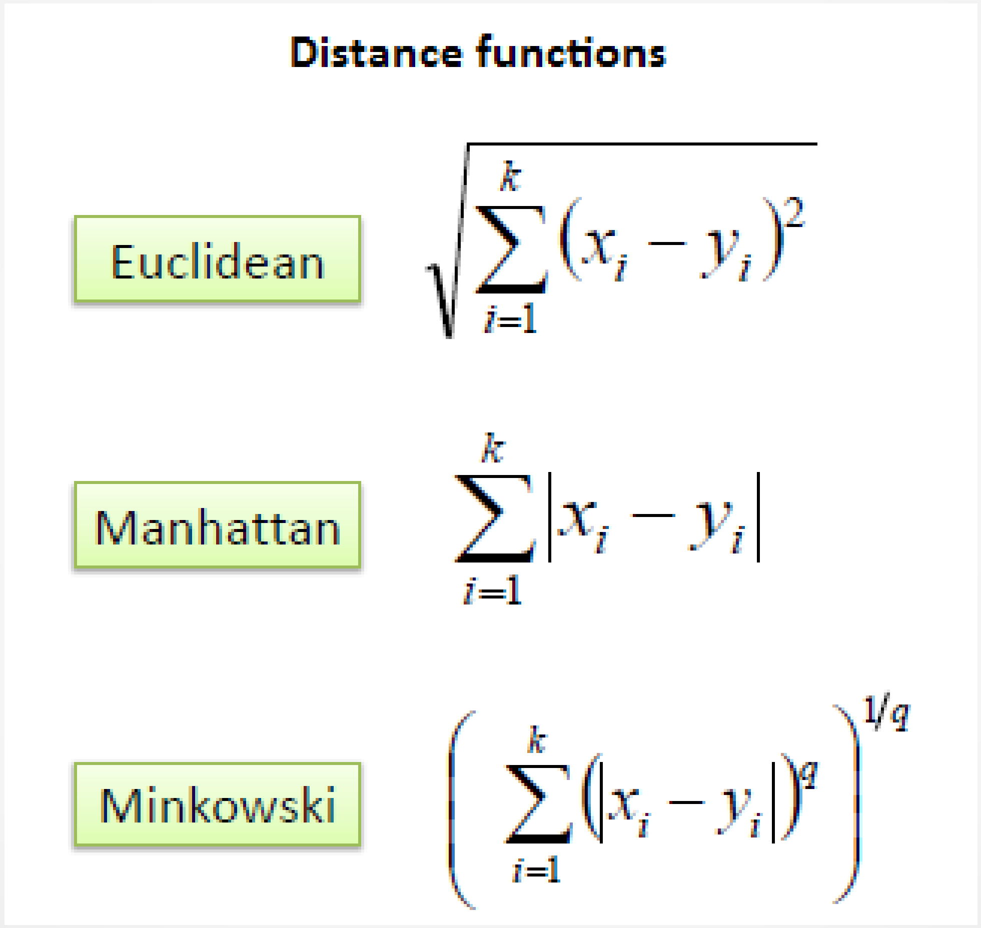
KNN Classification
Java Implementation
Part 2

Presented by:
Dr Noureddin Sadawi

Diagram



Distance measures for cont. variables



Distance functions

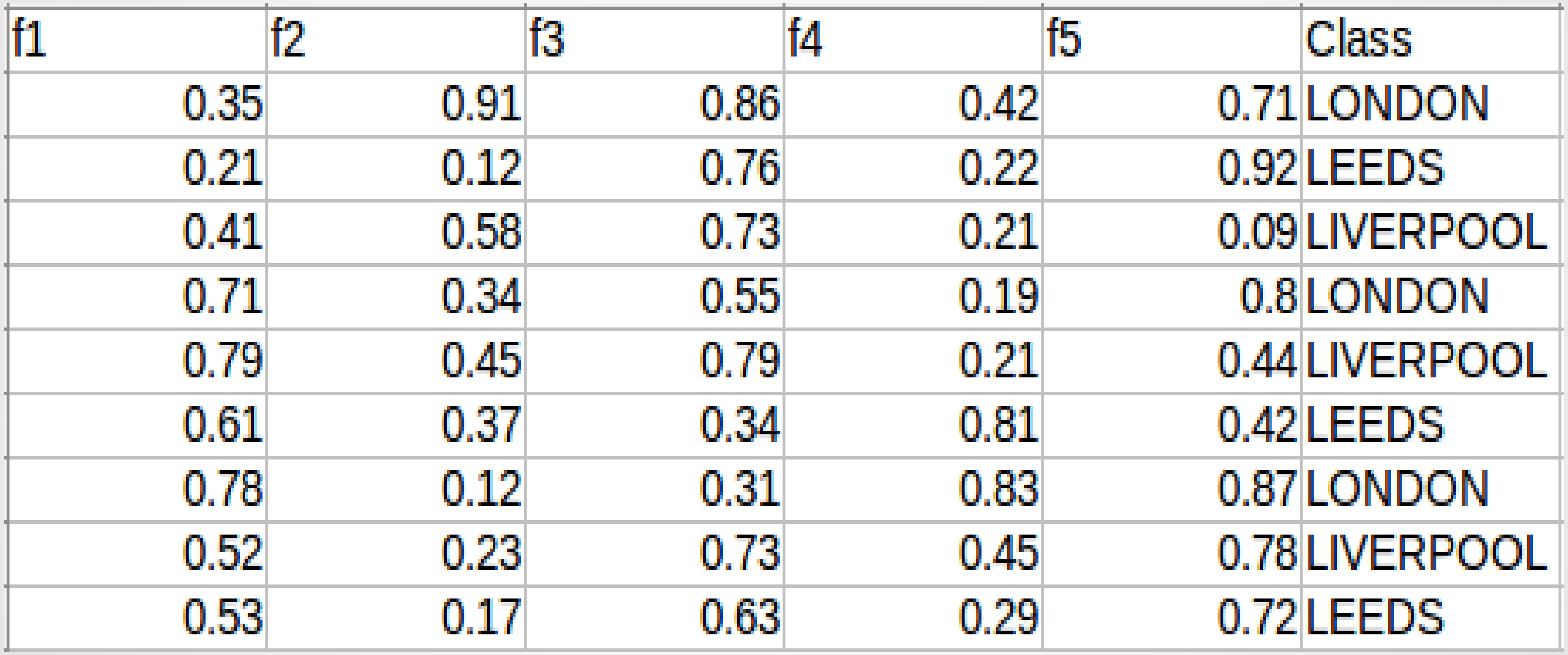
Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}\left|x_i - y_i\right|$$

Minkowski $$\left(\sum_{i=1}^{k}(\left|x_i - y_i\right|)^q\right)^{1/q}$$

p1=(w1,x1,y1,z1), p2=(w2,x2,y2,z2)
Euc. Dist. (p1,p2) = sqrt((w1-w2)^2+(x1-x2)^2+(y1-y2)^2+(z1-z2)^2)

Example Dataset



f1	f2	f3	f4	f5	Class
0.35	0.91	0.86	0.42	0.71	LONDON
0.21	0.12	0.76	0.22	0.92	LEEDS
0.41	0.58	0.73	0.21	0.09	LIVERPOOL
0.71	0.34	0.55	0.19	0.8	LONDON
0.79	0.45	0.79	0.21	0.44	LIVERPOOL
0.61	0.37	0.34	0.81	0.42	LEEDS
0.78	0.12	0.31	0.83	0.87	LONDON
0.52	0.23	0.73	0.45	0.78	LIVERPOOL
0.53	0.17	0.63	0.29	0.72	LEEDS

Query 0.65 0.78 0.21 0.29 0.58 Class = ?

How to find Majority Class

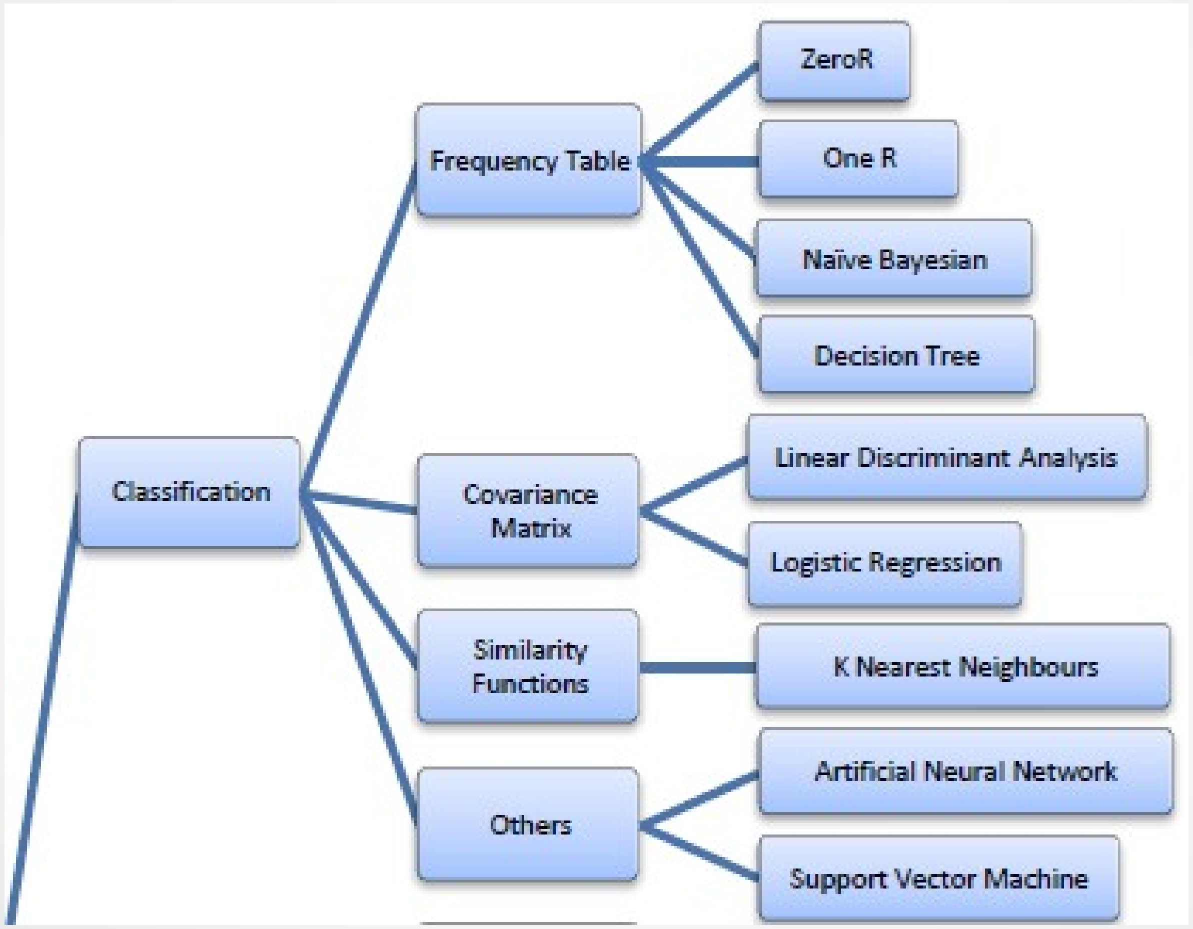
- Build an array of the unique String values *(same order as in the original array)*
- Create histogram of unique values in the original array *(counts for unique strings)*
- Find the max value of counts *(in the counts array)*
- Find the frequency of max in the counts array and this will give us how many times the most frequent classes appear *(in case of multiple majority classes)*
- If frequency is 1 *(max occurs once)*, then we have one majority class
  - We find the index of this "max" in the counts array and return the element at that index in the uniquevalues array
- If frequency is > 1 (max occurs more than once) then we have more than one majority class (a tie) and we need to break it at random
  - We create an array of size frequency (call it ix), in it we save indices of max in the counts array
  - We choose a random number rIndex s.t. 0<= rIndex < size of ix
  - We retrieve the value *at rIndex in array ix* (call this value nIndex)
  - We return the element at nIndex in the uniquevalues array

Artificial Neural Networks

The Perceptron

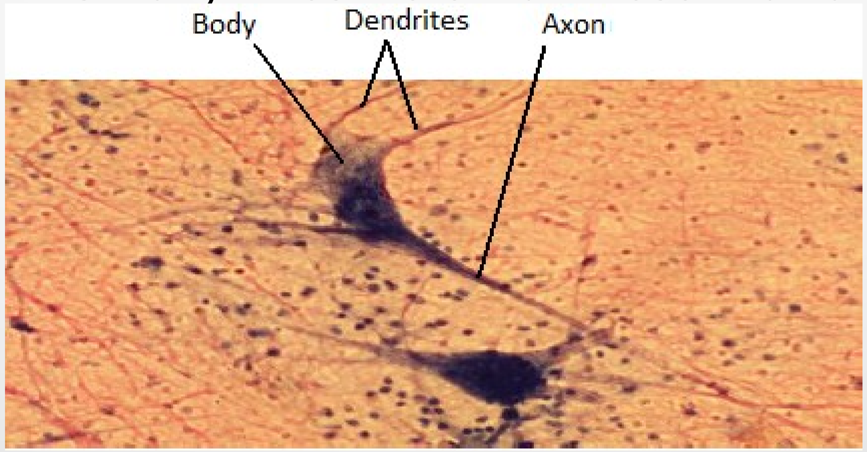
Part 1

Presented by:
Dr Noureddin Sadawi


Classification
Frequency Table
ZeroR
One R
Naïve Bayesian
Decision Tree
Covariance Matrix
Linear Discriminant Analysis
Logistic Regression
Similarity Functions
K Nearest Neighbours
Others
Artificial Neural Network
Support Vector Machine

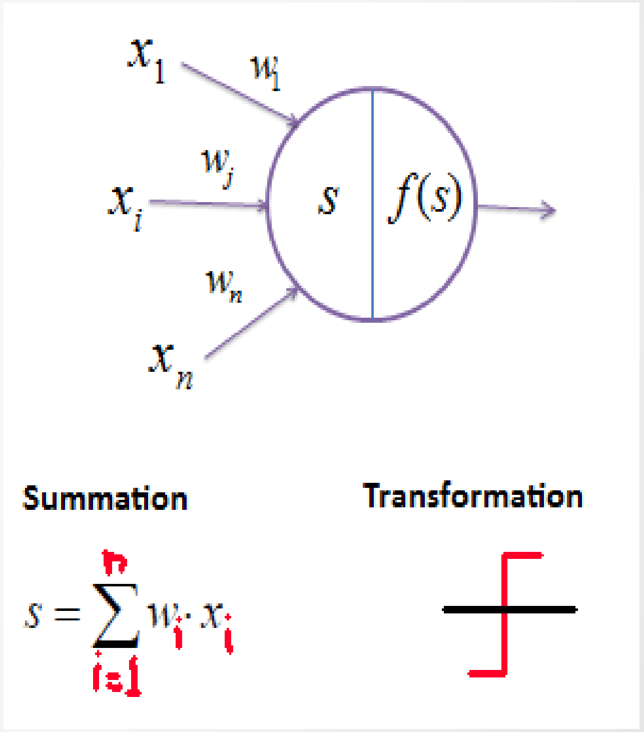
Biological Background

- An artificial neutral network (ANN) is a system that is based on the biological neural network, such as the brain
- The brain has approximately 100 billion neurons, which communicate through electro-chemical signals (the neurons are connected through junctions called synapses)
- Each neuron receives thousands of connections with other neurons, constantly receiving incoming signals to reach the cell body
- If the resulting sum of the signals surpasses a certain threshold, a response is sent through the axon
- The ANN attempts to recreate the computational mirror of the biological neural network, although it is not comparable since the number and complexity of neurons used in a biological neural network is many times more than those in an artificial neutral network

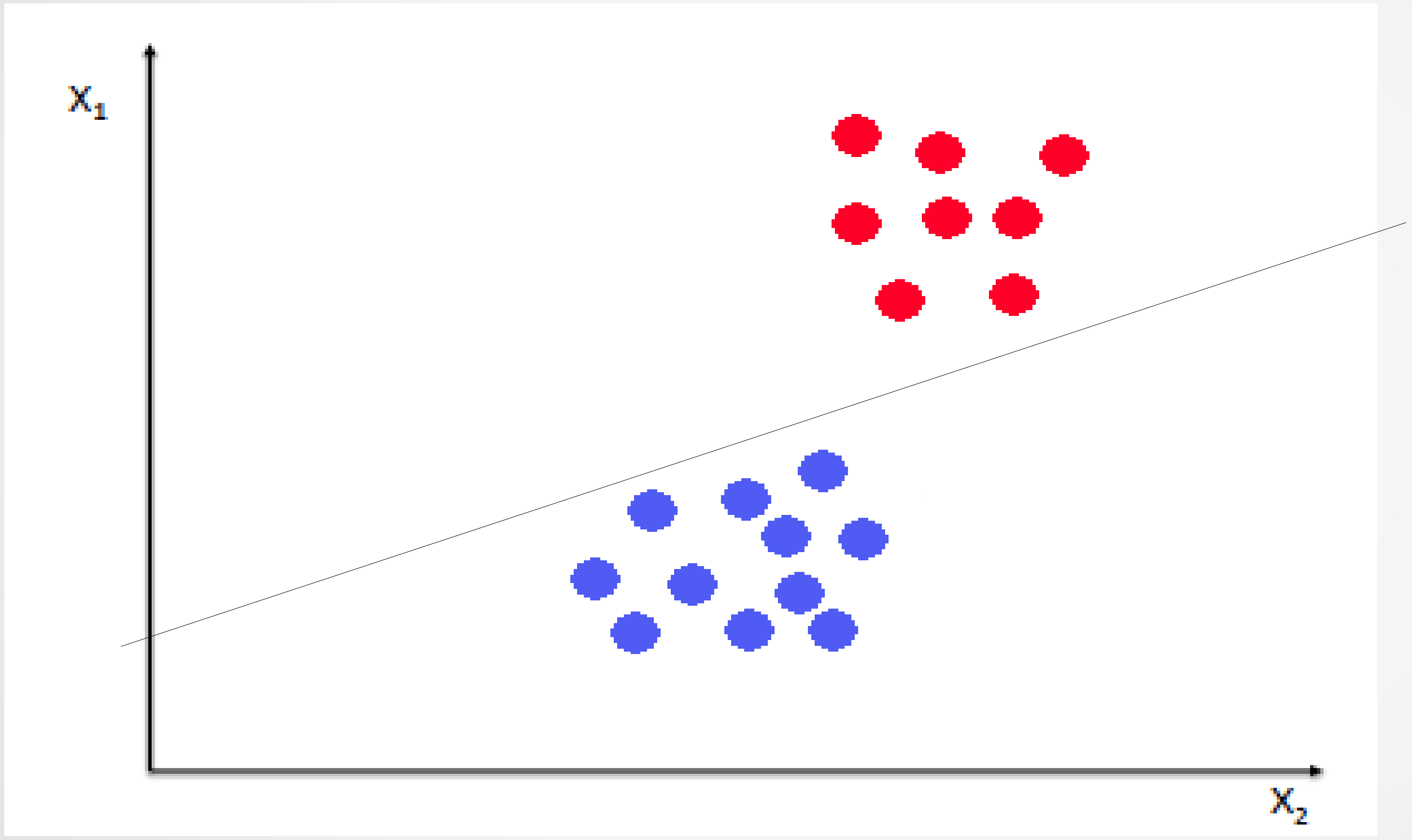


Perceptron (an artificial neuron)

- A perceptron models a neuron
- It receives n inputs (corresponding to features)
- It sums those inputs, checks the result and produces an output
- It is used to classify linearly separable classes
- Often for binary classification



Linear/Non-linear Separability



An Artificial Neuron

- The perceptron consists of weights, the summation processor, and an activation function
- A perceptron takes a weighted sum of inputs and outputs:

  1 if the sum is > some adjustable threshold value (theta)

  0 otherwise

Output

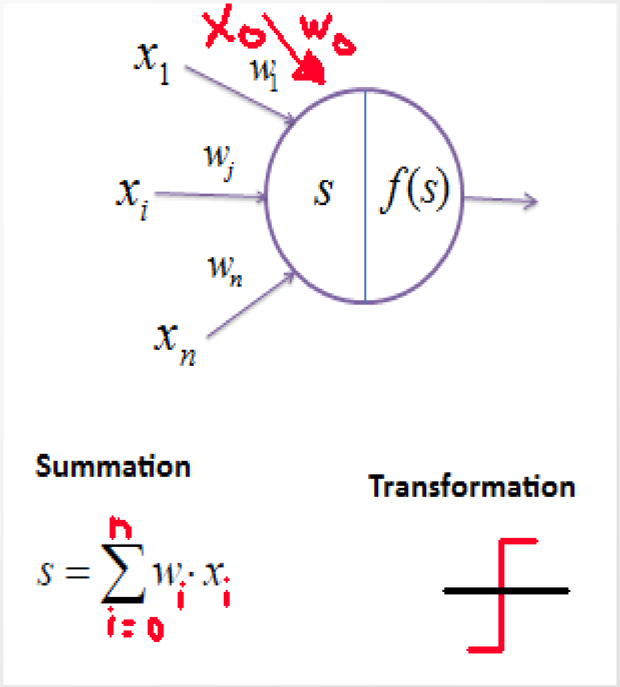
$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \theta \Longrightarrow 1$$

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq \theta \Longrightarrow 0$$

- The inputs and connection weights are typically real values

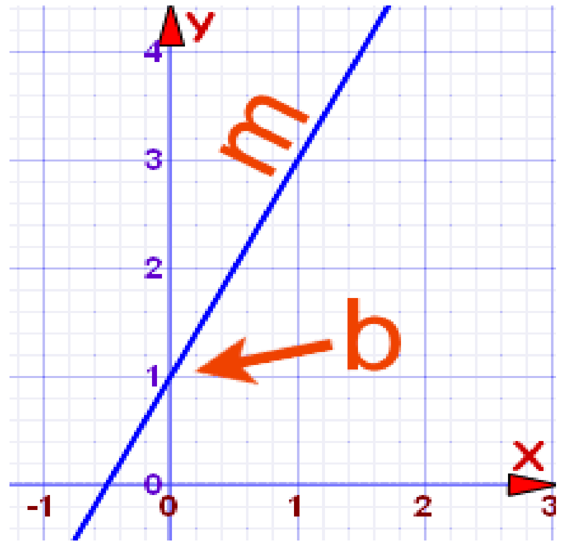
The Role of Weights and Bias

- The perceptron can have another input *known as the bias*
- It is normal practice is to treat the bias as just another input
- The bias allows us to shift the transfer function curve horizontally (left/right) along the input axis while leaving the shape/curvature unaltered
- The weights determine the slope



Remember Equation of a Straight Line ?

What does it stand for?



$$y = \boxed{mx} + \boxed{b}$$

Slope (or Gradient) Y Intercept

y = how far up

x = how far along

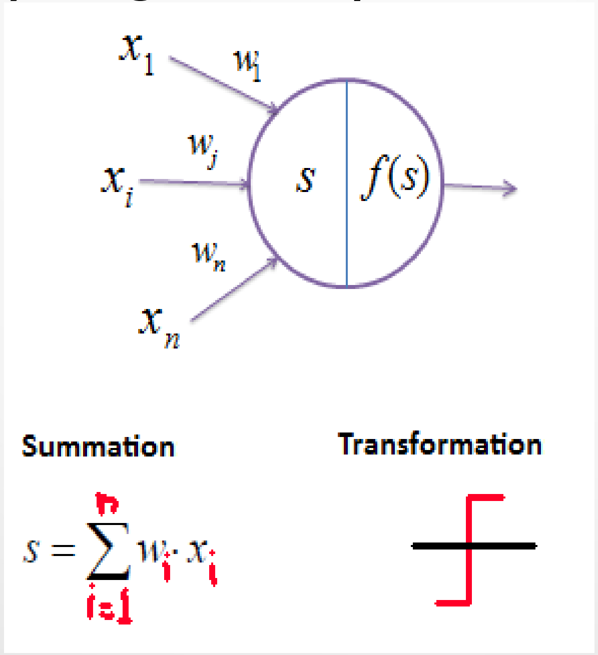
m = Slope or Gradient (how steep the line is)

b = the Y Intercept (where the line crosses the Y axis)

Source: http://www.mathsisfun.com/equation_of_line.html

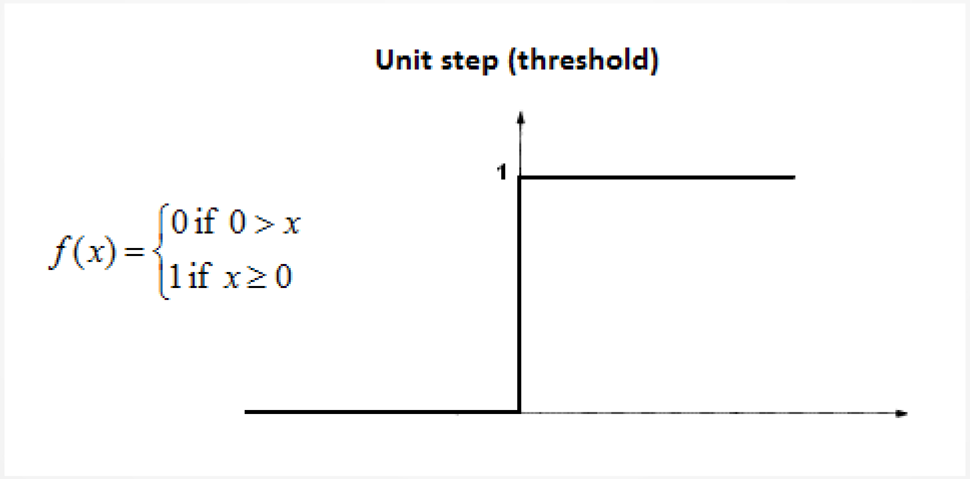
Transfer (Activation) Functions

- The transfer function translates the input signals to output signals
- It uses a threshold to produce an output
- Four types of transfer functions are commonly used, Unit step (threshold), sigmoid, piecewise linear, and Gaussian



Unit step (threshold)

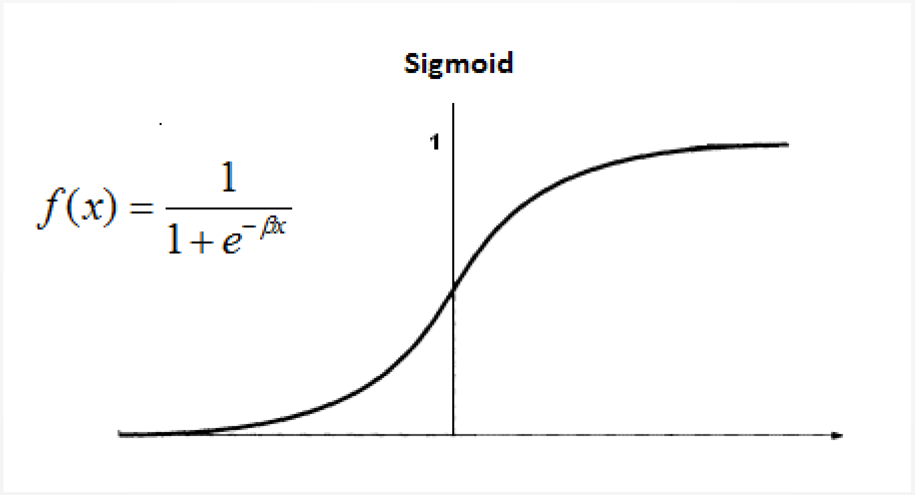
- The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value.



$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$

Sigmoid

- The sigmoid function consists of 2 functions, logistic and tangential. The values of logistic function range from 0 and 1 and -1 to +1 for tangential function.

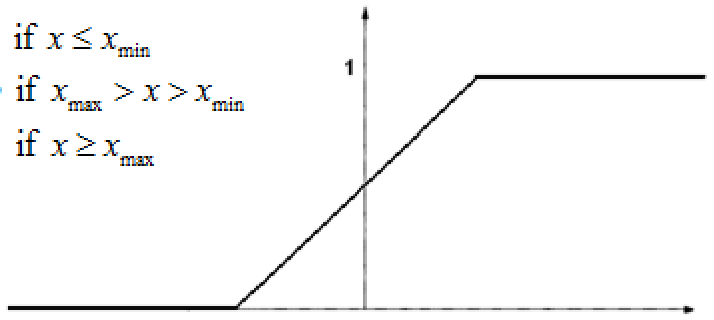


Piecewise Linear

- The output is proportional to the total weighted output.

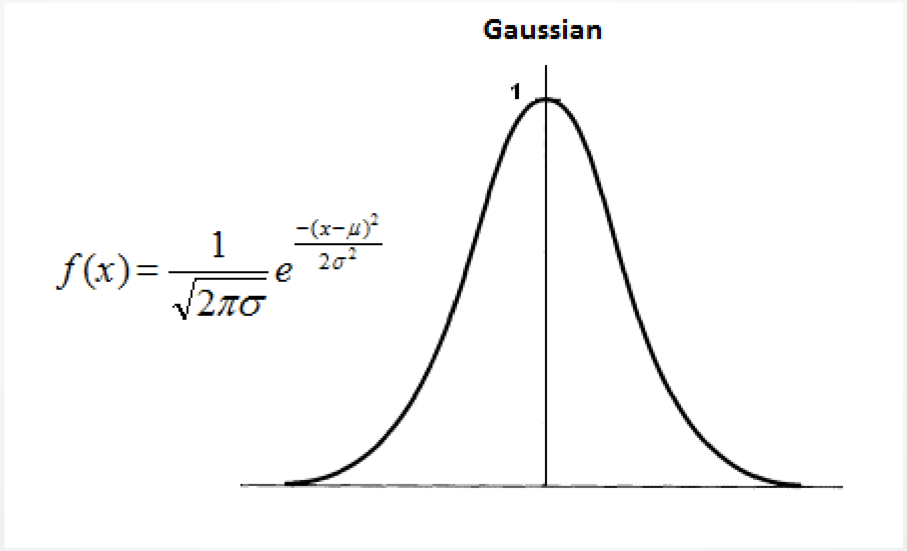
Piecewise Linear

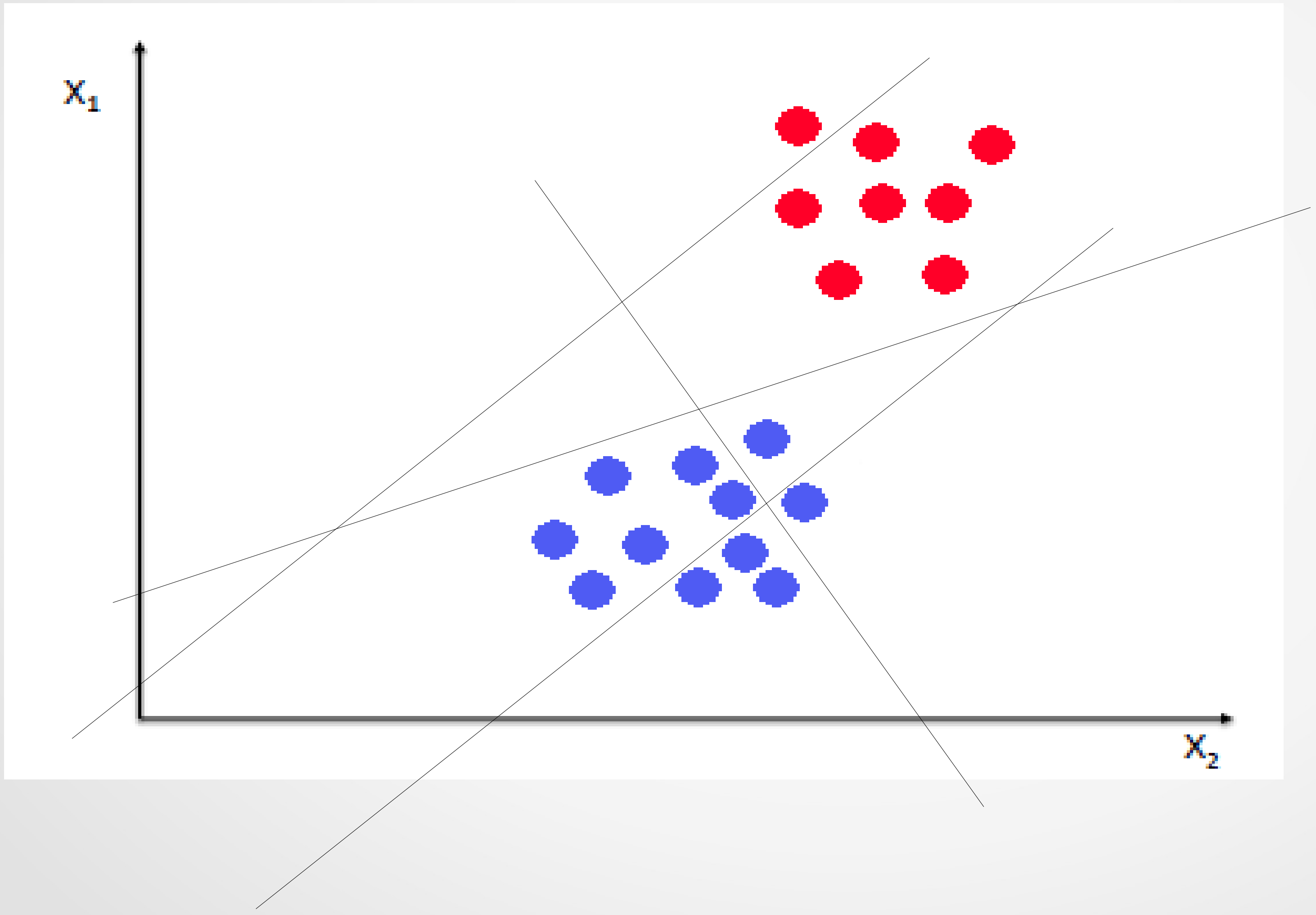
$$f(x) = \begin{cases} 0 & \text{if } x \le x_{\min} \\ mx + b & \text{if } x_{\max} > x > x_{\min} \\ 1 & \text{if } x \ge x_{\max} \end{cases}$$



Gaussian

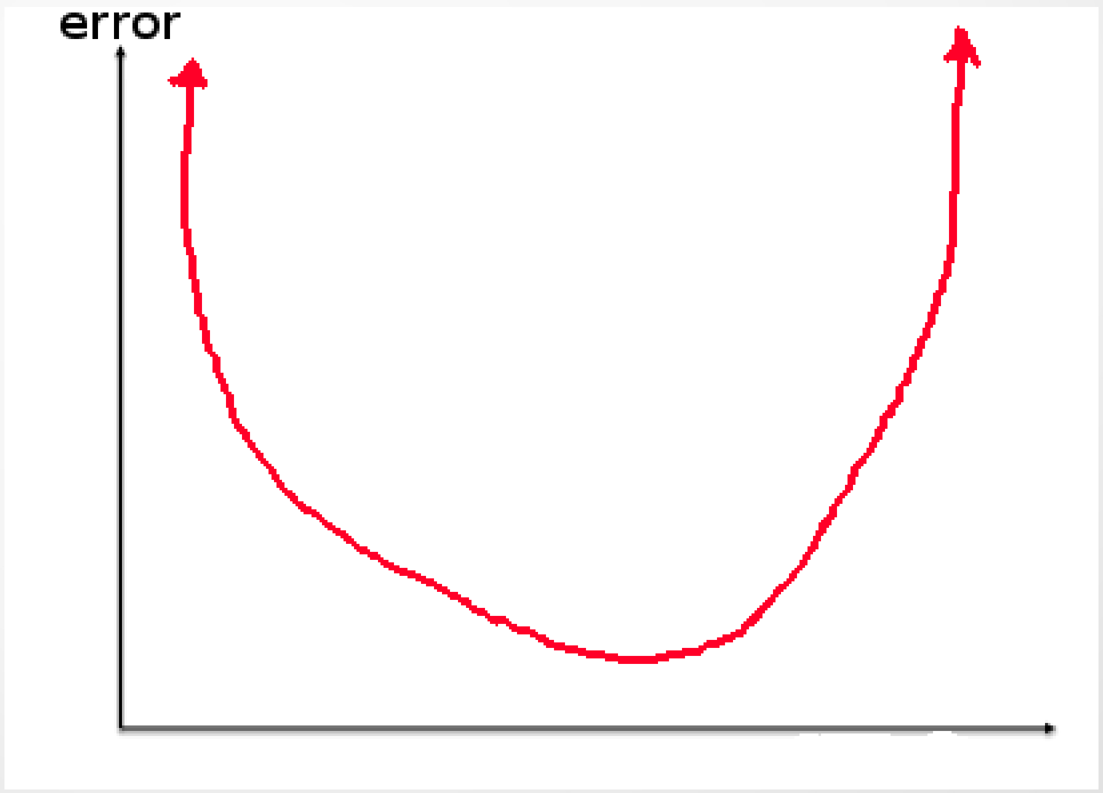
- Gaussian functions are bell-shaped curves that are continuous. The node output (high/low) is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen value of average.




X_1
X_2

The learning rate

- We would like to update the weights and bias in order to get a smaller error
- The learning rate helps us control how much we change the weight and bias



So … In Plain English

- If we have n variables, then we need to find n+1 weight values (n variables + the bias)
- These will be the coeffecients in the equation of the separation line|plane|hyperpalne
- For example, if we have 4 inputs, the equation becomes:

$$w0x0 + w1x1 + w2x2 + w3x3 + w4x4 = 0$$

which is equivalent to:

$$w1x1 + w2x2 + w3x3 + w4x4 + b = 0$$

where wi is the weight of input I and b is the bias (w0 with input value (x0) of 1)

Training/Learning in perceptrons

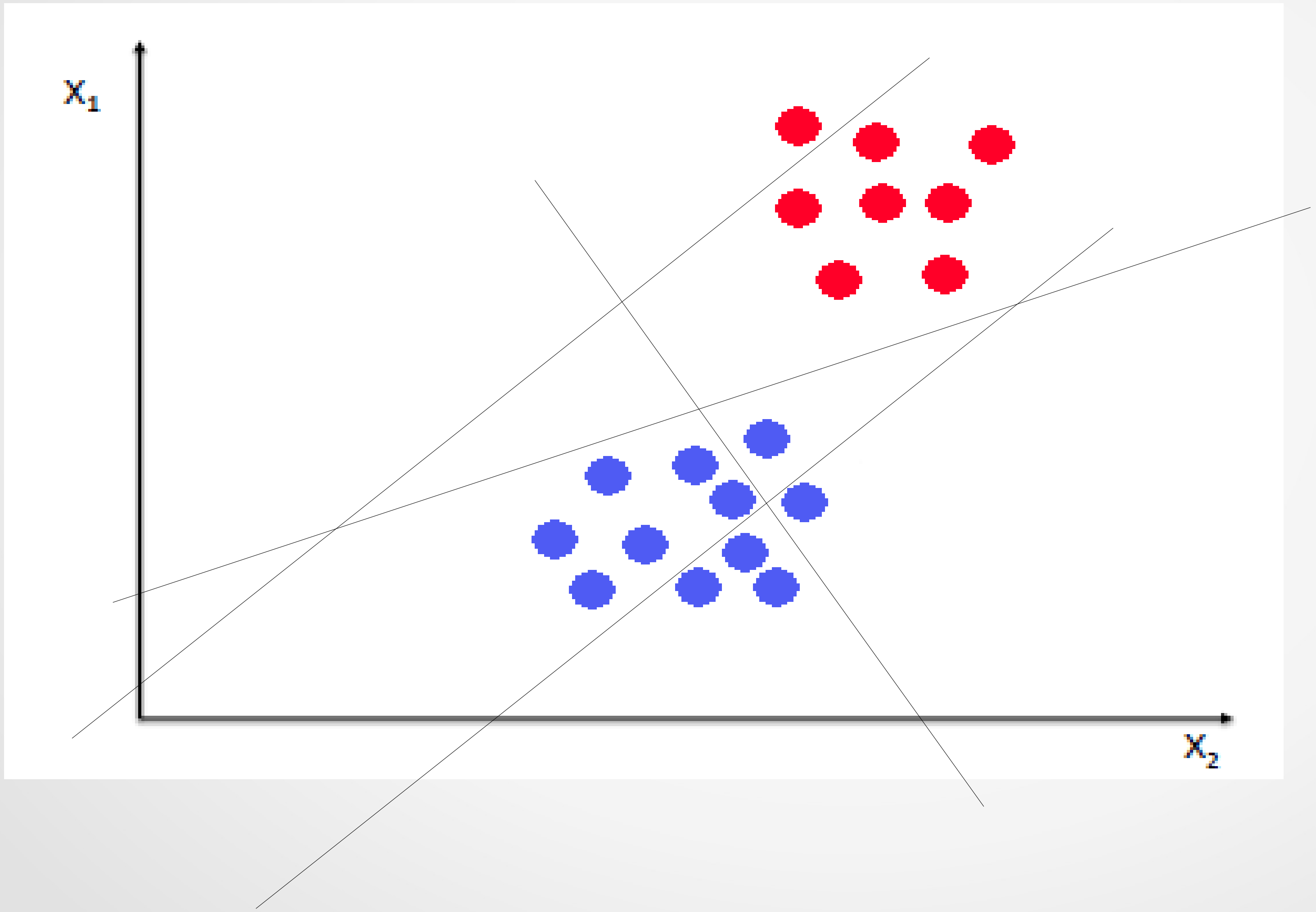
- We are trying to find a line|plane|hyperplane to correctly separate two classes by adjusting the weights and bias
- We train the perceptron to respond to each input vector with a corresponding target value of 0 or 1
- If a solution exists, it will be found in finite time

Procedure

- Initialize the weights (either to zero or to a small random value)
- Pick a learning rate m (this is a number between 0 and 1)
- Do the following until stopping condition is satisfied
- For each training instance (x, actual): epoch
  - compute activation output = f(w x)
  - The Learning Rule:

```
Find error = output - actual
b = b + m * error
For all inputs i:
    W(i) = W(i) + error * m * x(i)
```

Where W is the vector of weights, x is the input vector presented to the network, output is the predicted class, actual is the actual output of the neuron, and b is the bias



X_1
X_2

Training

- We present vectors from our training set to the network one after another
- If the network's output is correct, no change is made
- Otherwise, we use the perceptron learning rule to update the weights and biases
- training is complete If we finish an entire pass through all of the input training vectors of without error *(the term epoch is used to refer to an entire pass through all of the input training vectors)*
- Now we can present any input training vector to the network and it will output the correct output
- If a vector, P, not in the training set is presented to the network, the network will tend to exhibit generalization by responding with an output similar to target vectors for input vectors close to the previously unseen input vector P

Artificial Neural Networks

The Perceptron

Java Implementation

Presented by:
Dr Noureddin Sadawi

Java Implementation

- We will have three variables x, y and z (features)
- Each instance will belong to either class 1 or 0
- We will have 100 randomly generated instances (50 of class 0 and 50 of class 1) you can read instances from an input file if you wish
- We start with random weights and bias
- We loop through instances and update weights and bias (the process involves computing local & global error)
- We continue until stopping condition is satisfied (a solution is found OR max # of iterations is reached)
- I have modified Richard Knop's C code which can be found here: https://github.com/RichardKnop/ansi-c-perceptron

Transfer (Activation) Functions

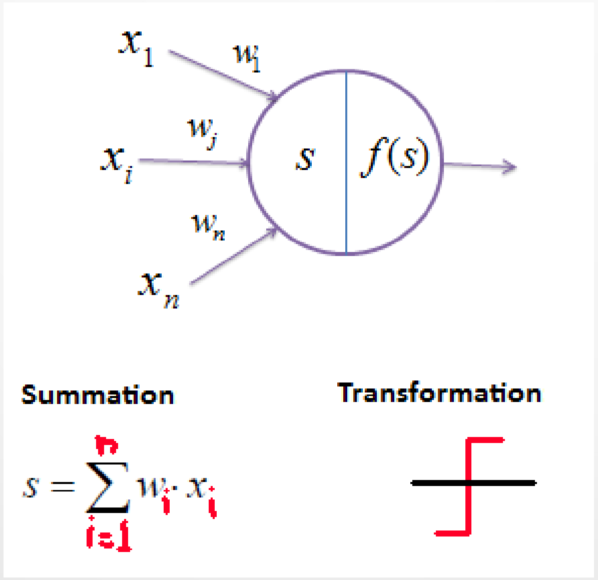
In Artifician Neural Networks

Presented by:
Dr Noureddin Sadawi

Most material belongs to Prof Saed Sayad
http://www.saedsayad.com

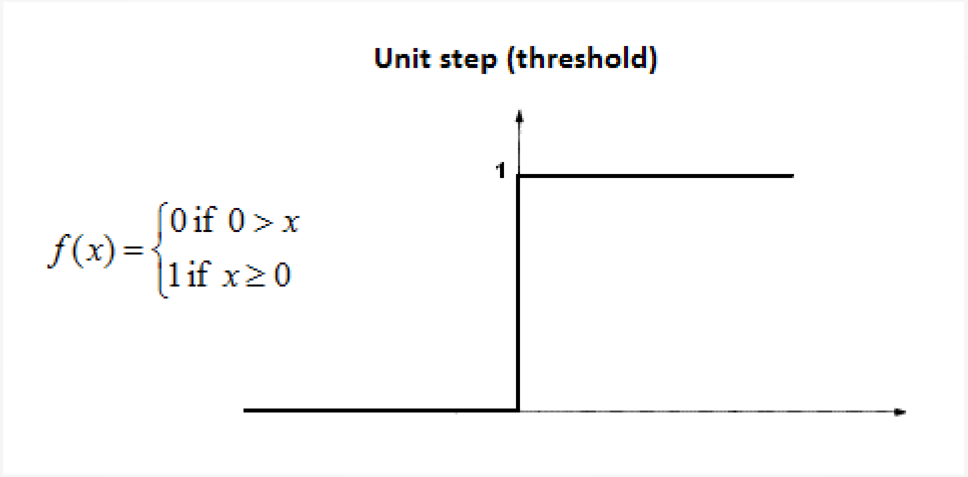
Transfer (Activation) Functions

- The transfer function translates the input signals to output signals
- Hence, it influences the behaviour of an ANN
- Four types of transfer functions are commonly used, Unit step (threshold), piecewise linear, sigmoid, and Gaussian

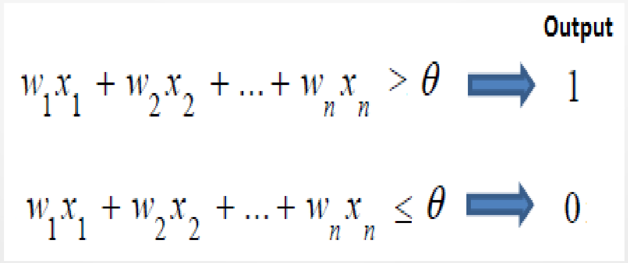


Unit Step (Threshold)

- The output is set at one of two levels, depending on whether the total input is greater than or less than some threshold value



$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



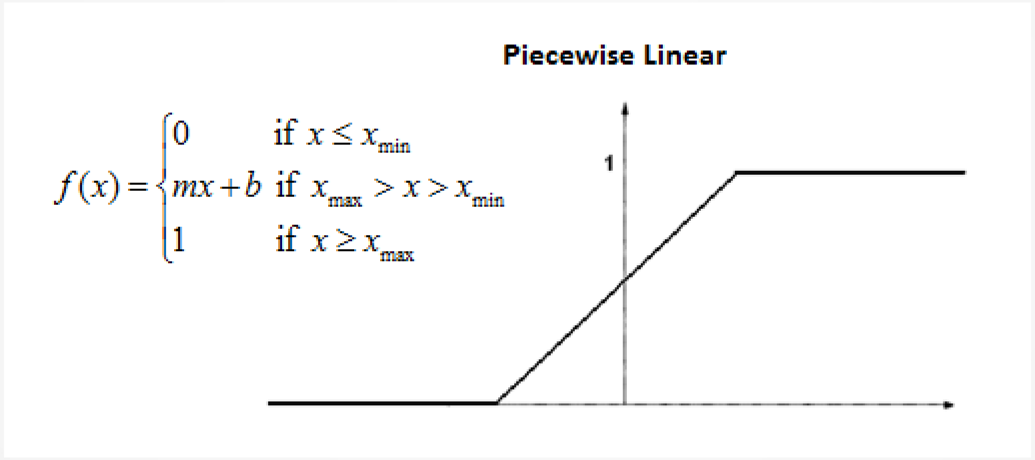
Output

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \theta \Rightarrow 1$$

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq \theta \Rightarrow 0$$

Piecewise Linear

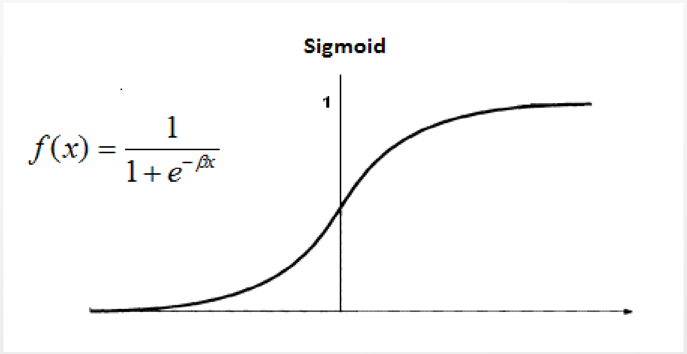
- The output is proportional to the total weighted output



$$f(x) = \begin{cases} 0 & \text{if } x \leq x_{\min} \\ mx + b & \text{if } x_{\max} > x > x_{\min} \\ 1 & \text{if } x \geq x_{\max} \end{cases}$$

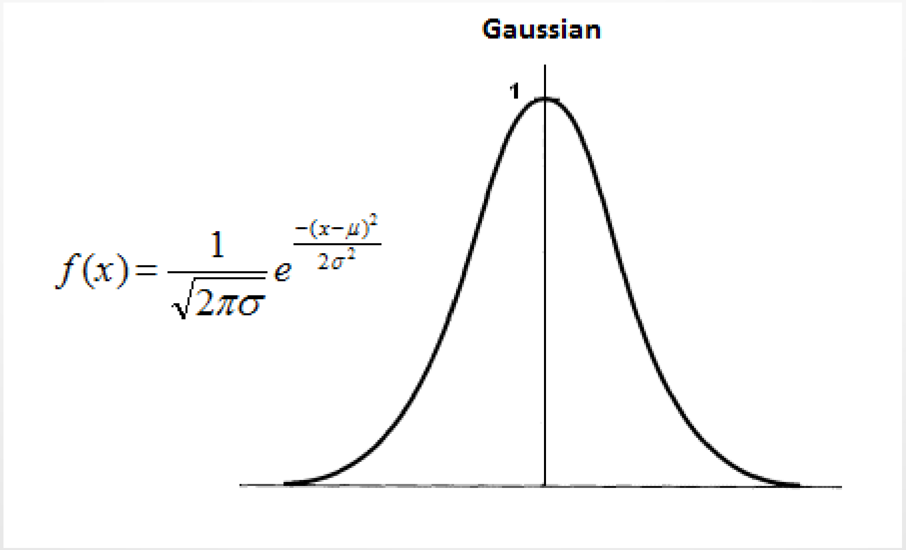
Sigmoid

- The output of this function changes continuously as the input changes (observe not linearly)
- It generates outputs between 0 and 1 as the input gradually changes from -ve to +ve infinity
- It is used when the output is expected to be a positive number
- It is considered a reasonable approximation of real neurones
- It is differentiable!



Gaussian

- Gaussian functions are bell-shaped curves that are continuous
- The node output (high/low) is interpreted in terms of class membership (1/0), depending on how close the net input is to a chosen value of average
- It can be used when finer control is needed over the activation range
- The derivative of the gaussian function can be obtained; hence, it can be used with propagation training



$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Other Activation Functions

- BiPolar: used with neural networks that require bipolar numbers (true or false)
- Log: uses an algorithm based on the log function
- Sin: based on the sine function, useful for data that periodically changes over time
- Tanh: uses the hyperbolic tangent function, commonly used activation function, as it works with both negative and positive numbers

Source: http://www.heatonresearch.com

Artificial Neural Networks

The Multi-Layer Perceptron

MLP

Presented by:
Dr Noureddin Sadawi

Parts of the Material belong to Prof Saed Sayad
http://www.saedsayad.com

An Artificial Neuron

- The perceptron consists of weights (including bias), the summation processor, and an activation function
- A perceptron takes a weighted sum of inputs and outputs:

  1 if the sum is > some adjustable threshold value (theta)

  0 otherwise

Output

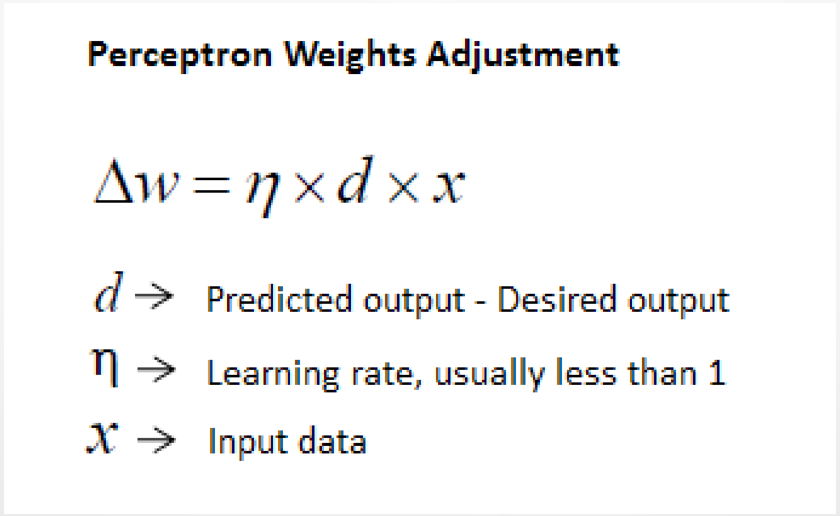
$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \theta \Longrightarrow 1$$

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n \leq \theta \Longrightarrow 0$$

- The inputs and connection weights are typically real values

How it Works

- The input values are presented to the perceptron, and if the predicted output is the same as the desired output, then the performance is considered satisfactory and no changes to the weights are made
- However, if the output does not match the desired output, then the weights need to be changed to reduce the error



Perceptron Weights Adjustment

$$\Delta w = \eta \times d \times x$$

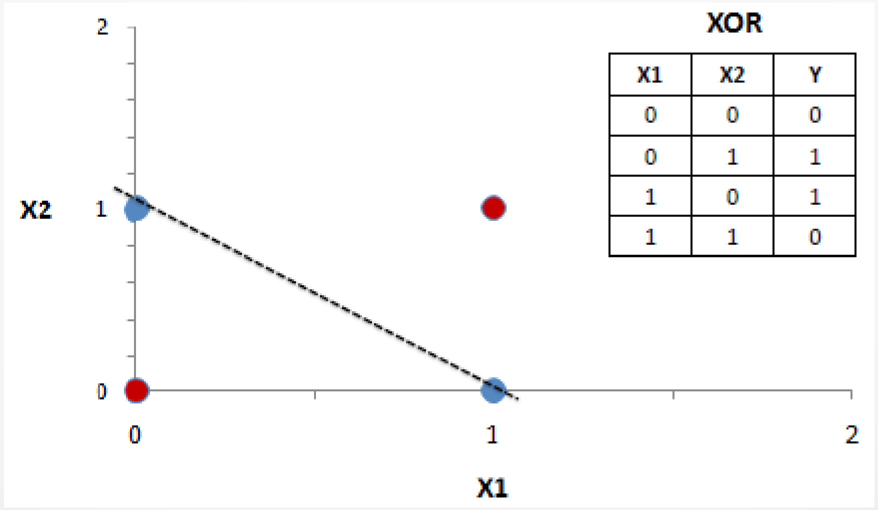
$d \rightarrow$ Predicted output - Desired output

$\eta \rightarrow$ Learning rate, usually less than 1

$x \rightarrow$ Input data

Non-Linearly Separable Data

- Because SLP is a linear classifier and if the cases are not linearly separable the learning process will never reach a point where all the cases are classified properly
- The most famous example of the inability of perceptron to solve problems with linearly non-separable cases is the XOR problem



XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

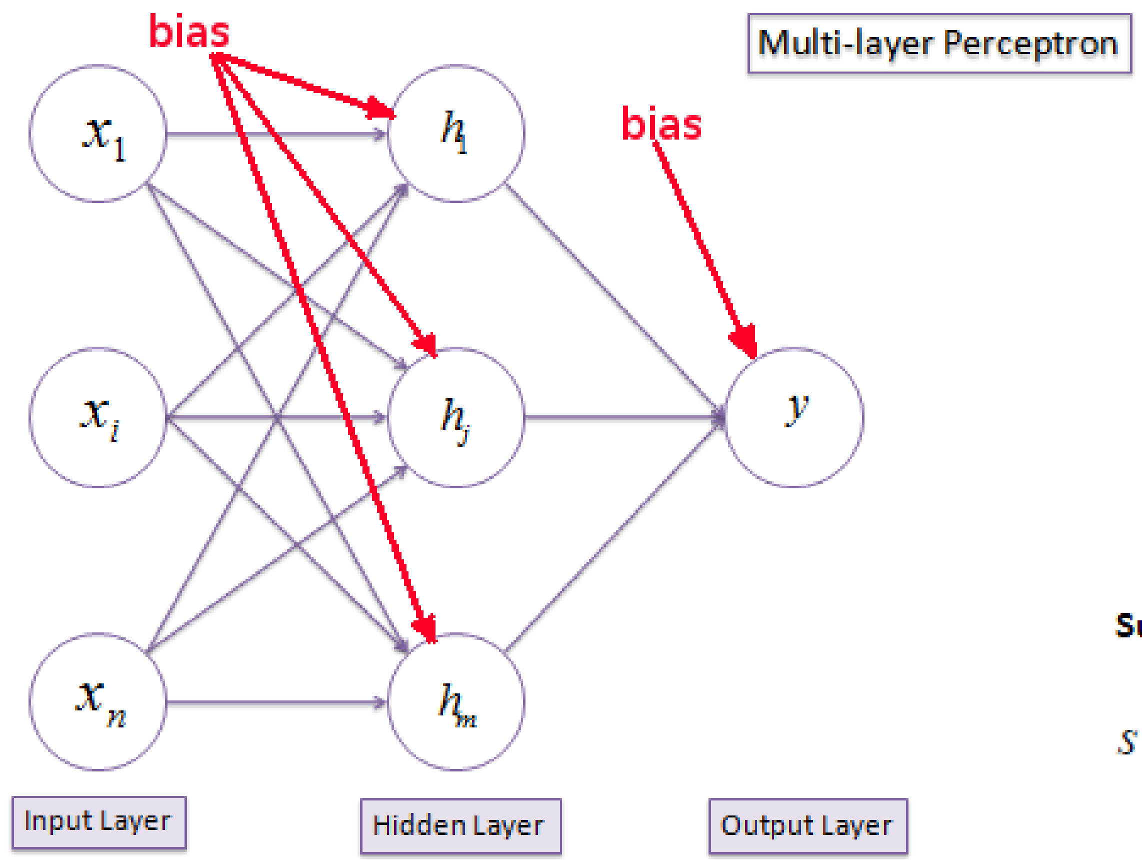
- However, a multi-layer perceptron using the backpropagation algorithm can successfully classify the XOR data

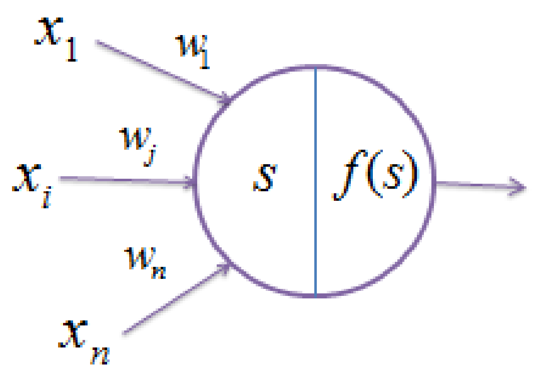
Exercise

- Download my java implementation of the Perceptron algorithm
- Modify the code such that the inputs and outputs are:
  - Inputs: double[] x = {0.0,0.0,1.0,1.0}; double[] y = {0.0,1.0,0.0,1.0};
  - //AND gate   int[] outputs = {0,0,0,1};
  - //OR gate   int[] outputs = {0,1,1,1};
  - //XOR gate   int[] outputs = {0,1,1,0};
- You need to modify the weights array and the calculateOutput method as we now have 2 inputs
- Also modify the number of instances to be 4
- Run it for the AND, OR and XOR gates and see what happens!

The Multi-Layer Perceptron   MLP

- A multi-layer perceptron (MLP) has the same structure of a single layer perceptron with one or more hidden layers
- Let's assume we have one input layer, one hidden layer, and one output layer
- We use the inputs and weights to work out the activation for any node as we learned before *(i.e. weighted sum and transfer function)*
- This is easily achieved for the hidden layer as it has direct links to the actual input layer
- We use the output from the hidden layer nodes to work out the activation for an output node (they are the inputs to the output layer nodes), observe that the output layer is not directly connected to the input layer, so it knows nothing about it
- Think of it as forward passing something from one layer to the next (propagate inputs by adding all the weighted inputs and then computing outputs using sigmoid threshold)





Summation

$$s = \sum w \cdot x$$

Transformation

$$f(s) = \frac{1}{1+e^{-s}}$$

The Back Propagation

- In BP we use output error, to adjust the weights of inputs at the output layer
- We can also calculate the error at the previous layer, and use it to adjust the weights arriving there
- We repeat this process of back-propagating errors through any number of layers
- This is made possible by using a sigmoid as the non-linear transfer function
- The sigmoid is used because it is differentiable

Artificial Neural Networks

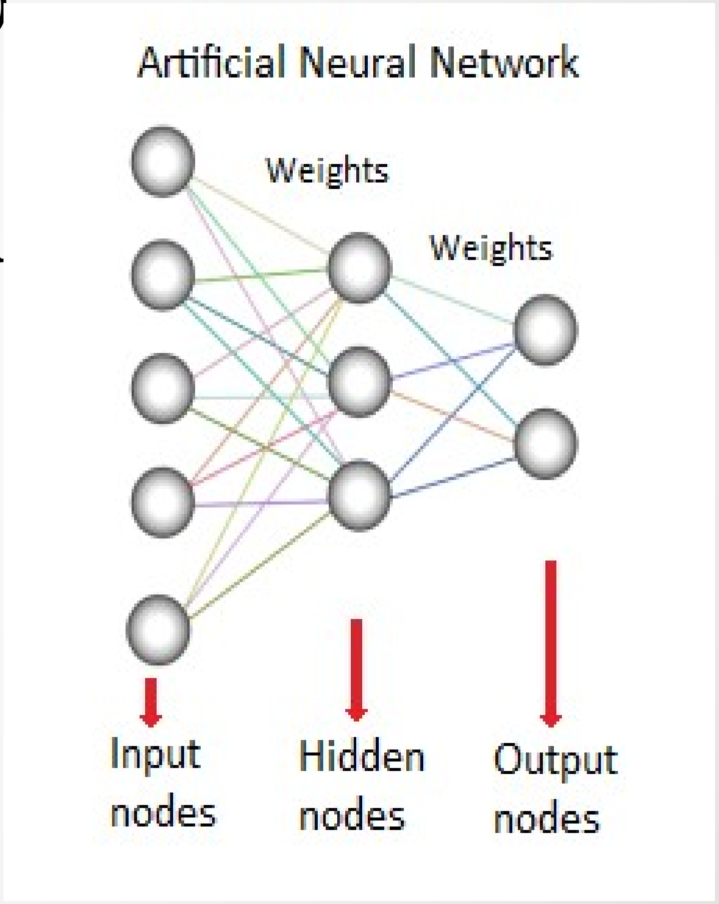
Feed-Forward & Feed-Back ANN

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com

Components of ANN

- An ANN is comprised of a network of artificial neurons (also known as "nodes")
- These nodes are connected to each other, and the strength of their connections to one another is assigned a value based on their strength
- If the value of the connection is high, then it indicates that there is a strong connection
- Within each node's design, a transfer function is built in
- There are three types of neurons in an ANN, input nodes, hidden nodes, and output nodes



Flow of ANN

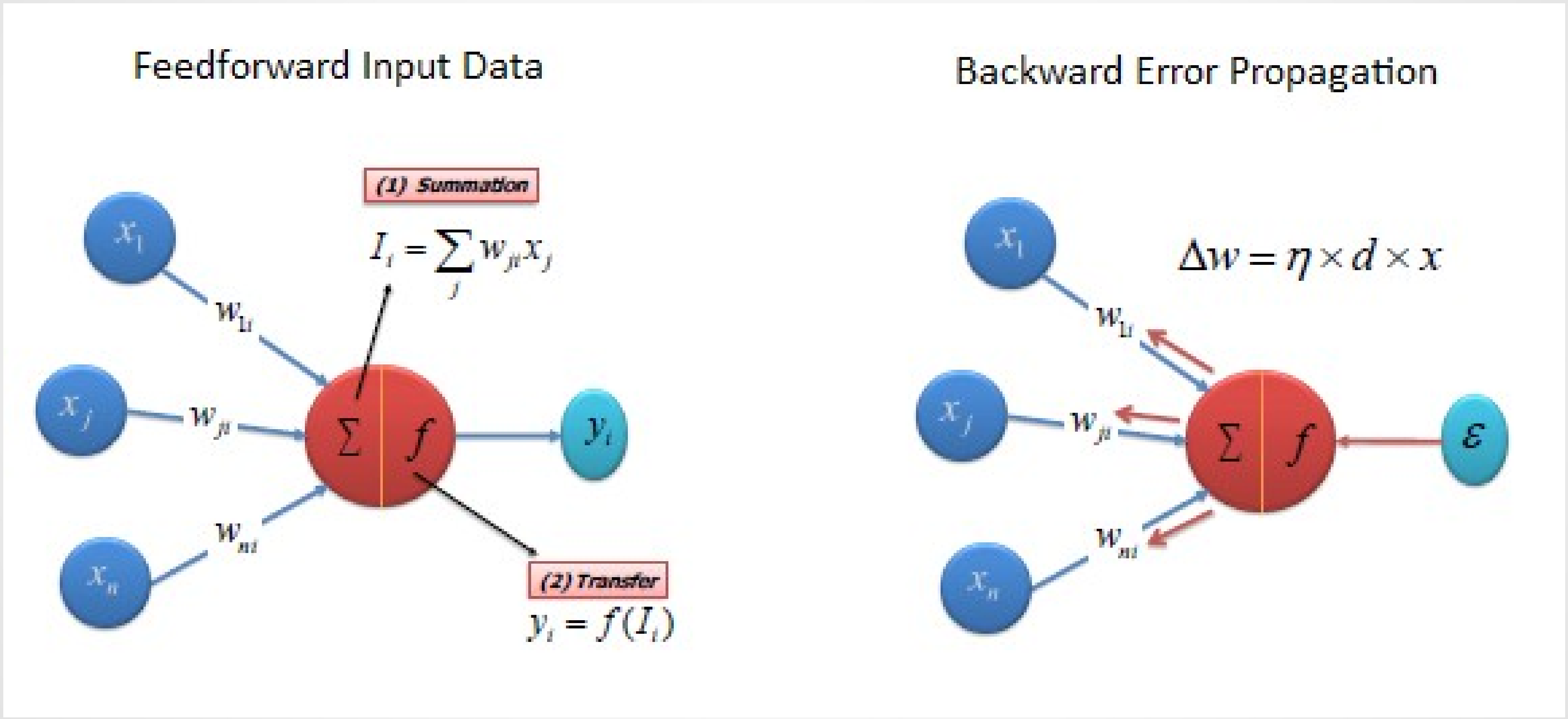
- The input nodes take in information, in the form which can be numerically expressed
- The information is presented as activation values, where each node is given a number, the higher the number, the greater the activation
- This information is then passed throughout the network
- Based on the connection strengths (weights), inhibition or excitation, and transfer functions, the activation value is passed from node to node
- Each of the nodes sums the activation values it receives; it then modifies the value based on its transfer function
- The activation flows through the network, through hidden layers, until it reaches the output nodes
- The output nodes then reflect the input in a meaningful way to the outside world

Algorithm

- There are different types of neural networks, but they are generally classified into feed-forward and feed-back networks

Feed-Forward ANN

- A FF network is a non-recurrent network which contains inputs, outputs, and hidden layers; the signals can only travel in one direction
- Input data is passed onto a layer of processing elements where it performs calculations
- Each processing element makes its computation based upon a weighted sum of its inputs
- The new calculated values then become the new input values that feed the next layer
- This process continues until it has gone through all the layers and determines the output
- A threshold transfer function is sometimes used to quantify the output of a neuron in the output layer
- FF networks include Perceptron (linear and non-linear) and Radial Basis Function networks
- Feed-forward networks are often used in data mining


Feedforward Input Data
Backward Error Propagation
(1) Summation
$I_i = \sum_j w_{ji} x_j$
x_1
x_j
x_n
w_{1i}
w_{ji}
w_{ni}
Σ
f
y_i
(2) Transfer
$y_i = f(I_i)$
$\Delta w = \eta \times d \times x$
ε

Feed-Back ANN

- A FB network has feed-back paths meaning they can have signals traveling in both directions using loops
- All possible connections between neurons are allowed
- Since loops are present in this type of network, it becomes a non-linear dynamic system which changes continuously until it reaches a state of equilibrium
- FB networks are often used in associative memories and optimization problems where the network looks for the best arrangement of interconnected factors

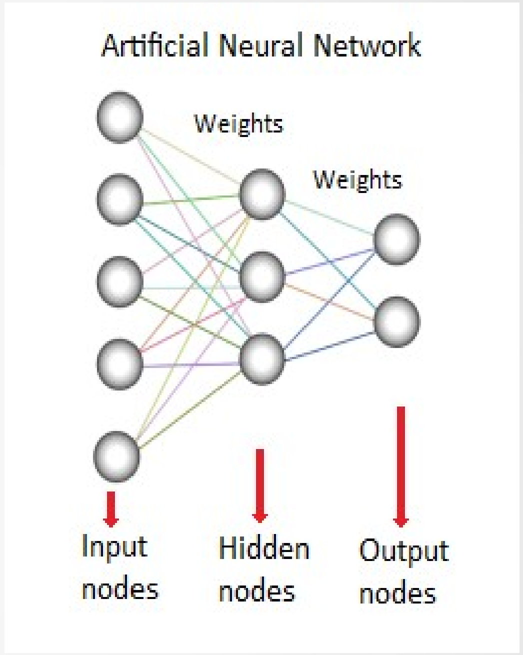
Radial Basis Function Networks (RBF)

Presented by:
Dr Noureddin Sadawi

Material belongs to Prof Saed Sayad
http://www.saedsayad.com

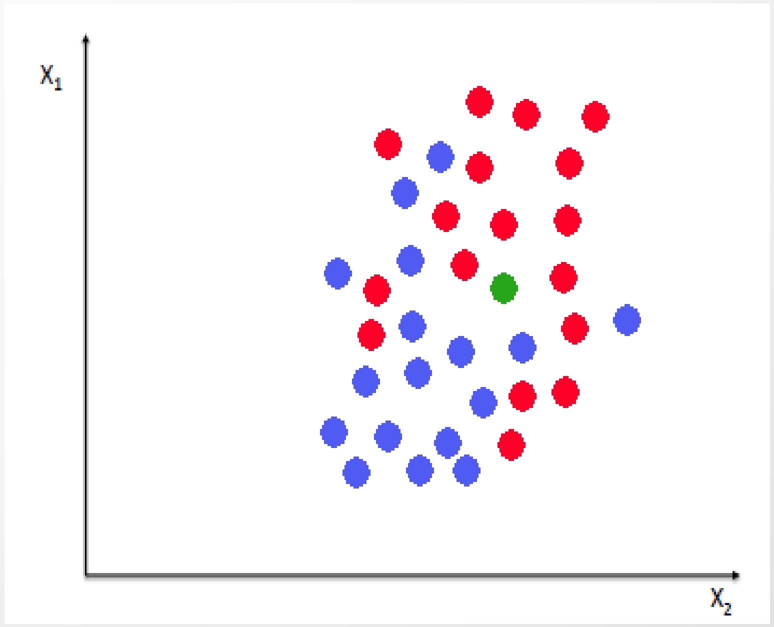
Overview of RBF Networks

- RBF networks have three layers: input layer, hidden layer and output layer
- Used for function approximation *(i.e. their output is a real value)*
- One neuron in the input layer corresponds to each predictor variable
- Each neuron in the hidden layer consists of a radial basis function (e.g. Gaussian) centered on a point with the same dimensions as the predictor variables
- The output layer has a weighted sum of outputs from the hidden layer to form the network outputs



How RBF networks work 1/2

- RBF neural networks are conceptually similar to K-Nearest Neighbor (k-NN) models
- The basic idea is that a predicted target value of an item is likely to be about the same as other items that have close values of the predictor variables

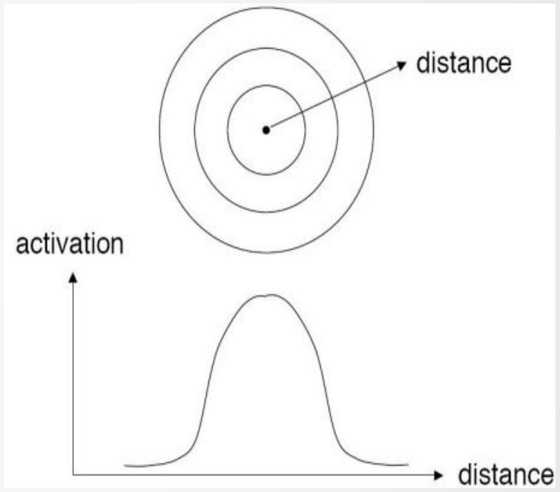


How RBF networks work 2/2

- An RBF network positions one or more RBF neurons in the space described by the predictor variables (x1,x2 for example)
- This space has as many dimensions as there are predictor variables
- The Euclidean distance is computed from the point being evaluated (e.g., the gree circle) to the center of each neuron, and a radial basis function (RBF) (also called a kernel function) is applied to the distance to compute the weight (influence) for each neuron
- The radial basis function is so named because the radius distance is the argument to the function

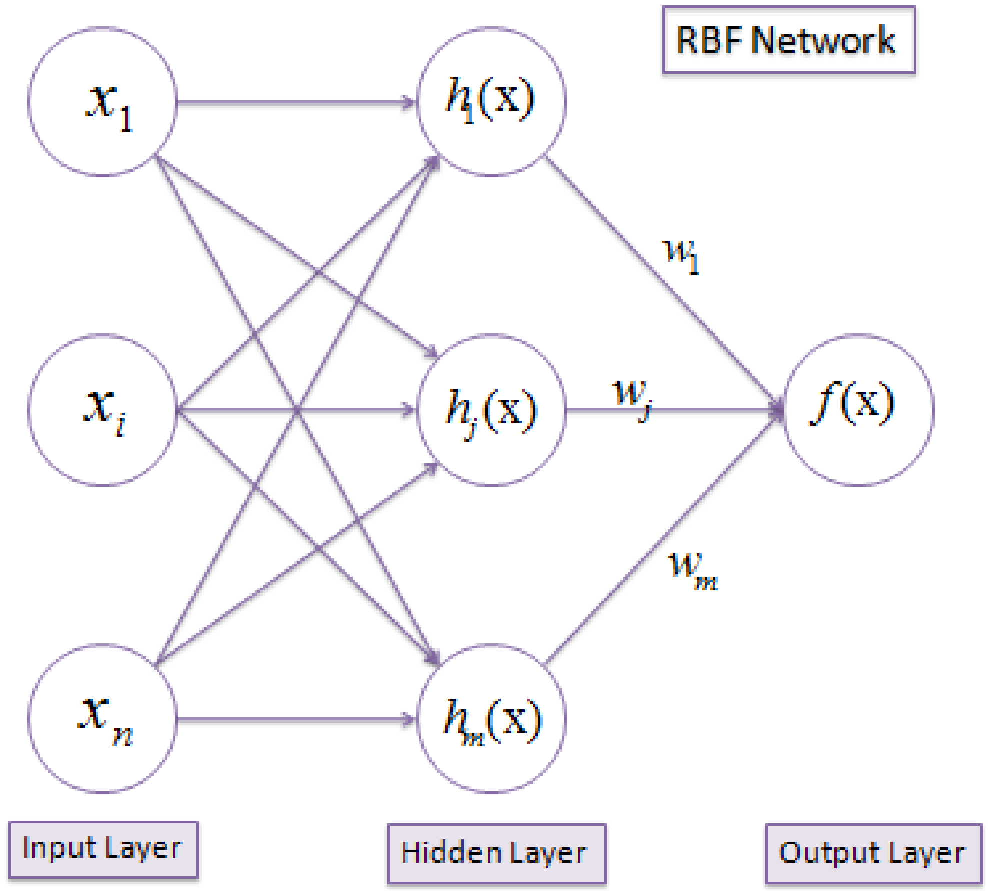
Weight = RBF(distance)

- The further a neuron is from the point being evaluated, the less influence it has



Source: http://www.dtreg.com

Structure of RBF Networks



$$f(x) = \sum_{j=1}^{m} w_j h_j(x)$$

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2} \right)$$

Algorithm

- h(x) is the Gaussian activation function with the parameters r (the radius or standard deviation) and c (the center or average taken from the input space) defined separately at each RBF unit
- The learning process is based on adjusting the parameters of the network to reproduce a set of input-output patterns
- There are three types of parameters; the weight w between the hidden nodes and the output nodes, the center c of each neuron of the hidden layer and the unit width r

Unit Center (c)

- Any clustering algorithm can be used to determine the RBF unit centers (e.g., K-means clustering)
- A set of clusters each with r-dimensional centers is determined by the number of input variables or nodes of the input layer
- The cluster centers become the centers of the RBF units
- The number of clusters is a design parameter and determines the number of nodes in the hidden layer
- The K-means clustering algorithm proceeds as follows:

  1- Initialize the center of each cluster to a different randomly selected training pattern

  2- Assign each training pattern to the nearest cluster. This can be accomplished by calculating the Euclidean distances between the training patterns and the cluster centers

  3- When all training patterns are assigned, calculate the average position for each cluster center. They then become new cluster centers

  4- Repeat steps 2 and 3, until the cluster centers do not change during the subsequent iterations

Unit width (r) … a.k.a the *spread*

- When the RBF centers have been established, the width of each RBF unit can be calculated using the K-nearest neighbors algorithm
- A number K is chosen, and for each center, the K nearest centers are found
- The root-mean squared distance between the current cluster center and its K nearest neighbors is calculated, and this is the value chosen for the unit width (r)
- So, if the current cluster center is cj, the r value is:

$$r_j = \sqrt{\frac{\sum_{i=1}^{k} (c_j - c_i)^2}{k}}$$

Weights (w)

- For each node in the hidden layer, now we have obtained the parameters r and c
- Now we compute the Euclidean distance from the point being evaluated to the center c of each neuron
- We then apply a radial basis function (RBF) (also called a kernel function) to the distance to compute the weight (influence) for each neuron

Three Learning Algorithms

- Choose the centers randomly from the training set, compute the spread for the RBF function using the normalization method and Find the weights using the pseudo-inverse method
- Use Clustering for finding the centers, normalization to choose Spreads and LMS algorithm for finding the weights (*Hybrid Learning Process*)
- Apply the gradient descent method for finding centers, spread and weights, by minimizing the (instantaneous) squared error

Support Vector Machine (SVM) Classification

Part 3

Presented by:
Dr Noureddin Sadawi

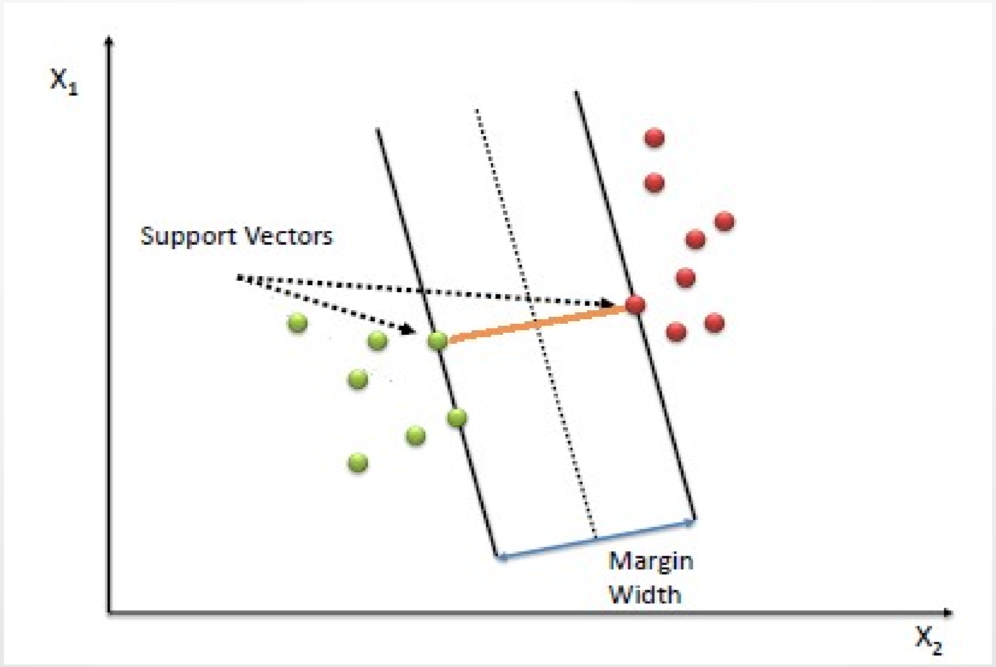
Contents adapted from:
http://www.saedsayad.com
And
Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, ***Introduction to Information Retrieval,*** *Cambridge University Press. 2008.*

Overview of SVM

- A Support Vector Machine (SVM) performs classification by finding the hyperplane that maximizes the margin between the two classes

Notice: linearly separable and binary

- *It draws the widest channel, or street, between the two classes*
- The two class labels are +1 (positive examples) and -1 (negative examples)



Remember

- In 2D we are seeking a line
- In 3D we are seeking a plane (surface)
- In > 3D we are seeking a hyperpane

- We will focus on 2D because it is easy to draw
- Whatever we can do in 2D, we should be able to do it in higher dimensions!

Intuition behind SVM

- Points (instances) are like vectors   p = (x1,x2,...,xn)
- SVM finds the closest two points from the two classes (see figure), that support (define) the best separating line|plane
- Then SVM draws a line connecting them (the orange line in the figure)
- After that, SVM decides that the best separating line is the line that bisects, and is perpendicular to, the connecting line

Requires basic knowledge of Vectors and Vector Algebra

Vectors

- We mentioned that each point will be a vector of the form

$$p = (x1,x2,...,xn)$$

- So our dataset can be represented as:

  $D=\{(x_1,y_1),(x_2,y_2,...,(x_n,y_n))\}$ **n** is the number of instances

  - x_i is the *ith* point (example or vector) and y_i is the class associated with that point
  - y_i can either be -1 or +1

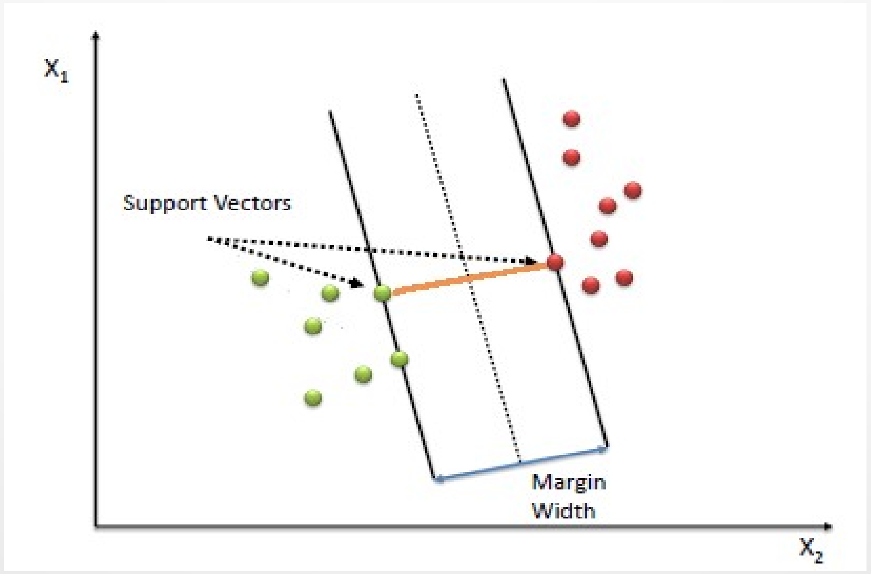
- A key concept required here is the dot product between two vectors (a.k.a inner,or scalar, product)

$$A = [a1, a2,...,an], B = [b1,b2,...,bn]$$

$$A.B = a1*b1 + a2*b2 + \dots + an*bn$$

Remember:

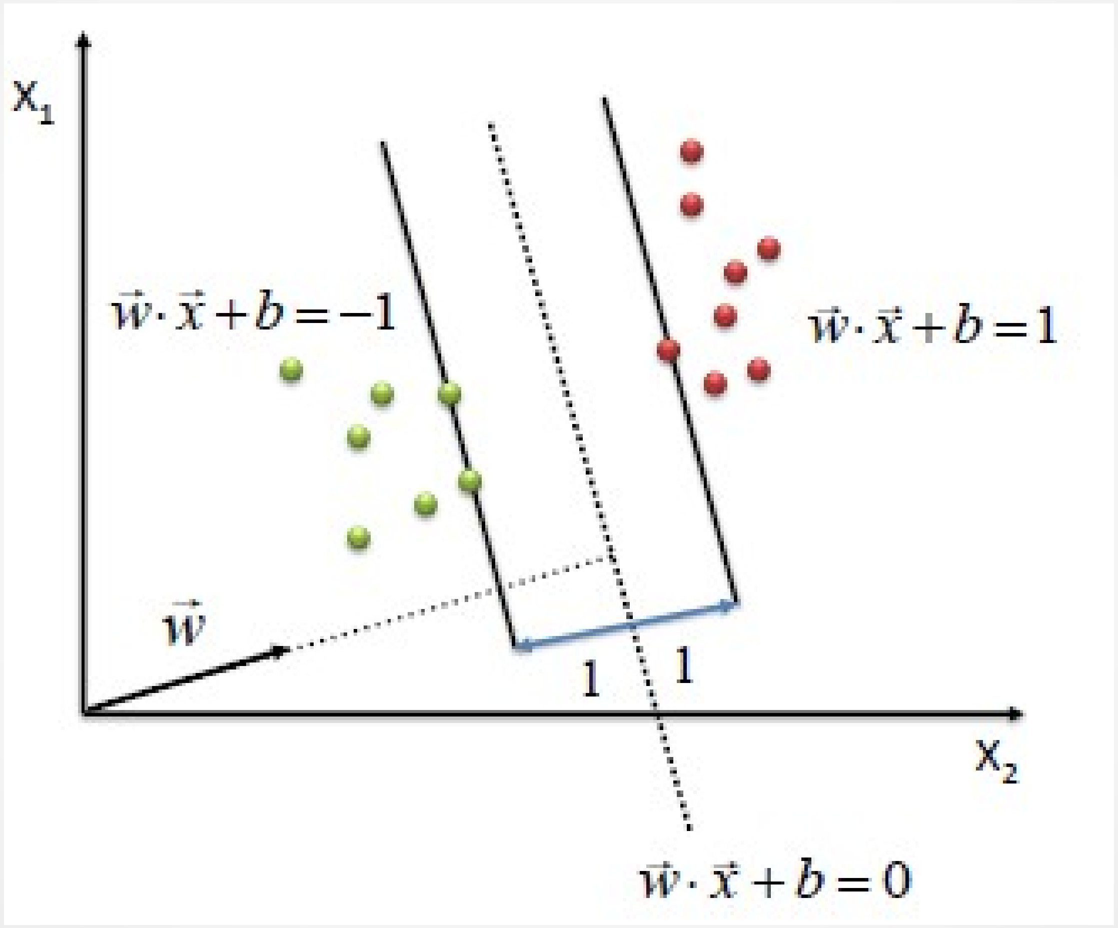
- Every nonzero vector has a corresponding unit vector, which has the same direction as that vector but a magnitude of 1
- We can compute the distance of a point to a plane
- The shortest distance between a point and a hyperplane is perpendicular to the plane



SVM more formally

- We can define a separating (decision) hyperplane in terms of an intercept term ***b*** and a normal vector $\vec{w}$ which is perpendicular to the hyperplane (commonly referred to as the weight vector)
- To choose among all the hyperplanes that are perpendicular to the normal vector, we specify the intercept term ***b***
- All points $\vec{x}$ on the hyperplane satisfy $\vec{w}^T\vec{x} = -b$ as the hyperplane is perpendicular to the normal vector
- We represent the training dataset as $\mathbb{D} = \{(\vec{x}_i, y_i)\}$ as a pair of a point and a class label corresponding to it
- Now the linear classifier becomes:

$$f(\vec{x}) = \text{sign}(\vec{w}^T\vec{x} + b)$$


X_1
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}$
1
1
X_2
$\vec{w}\cdot\vec{x}+b=0$

More Details

- The Euclidean distance r between a point $\vec{x}$ and the decision boundary is parallel to $\vec{w}$ with a unit vector in this direction is $\vec{w}/|\vec{w}|$
- So, the green line in the next diagram is a translation of the vector $r\vec{w}/|\vec{w}|$
- $\vec{x}'$ is the point on the plane closest to $\vec{x}$ and therefore:

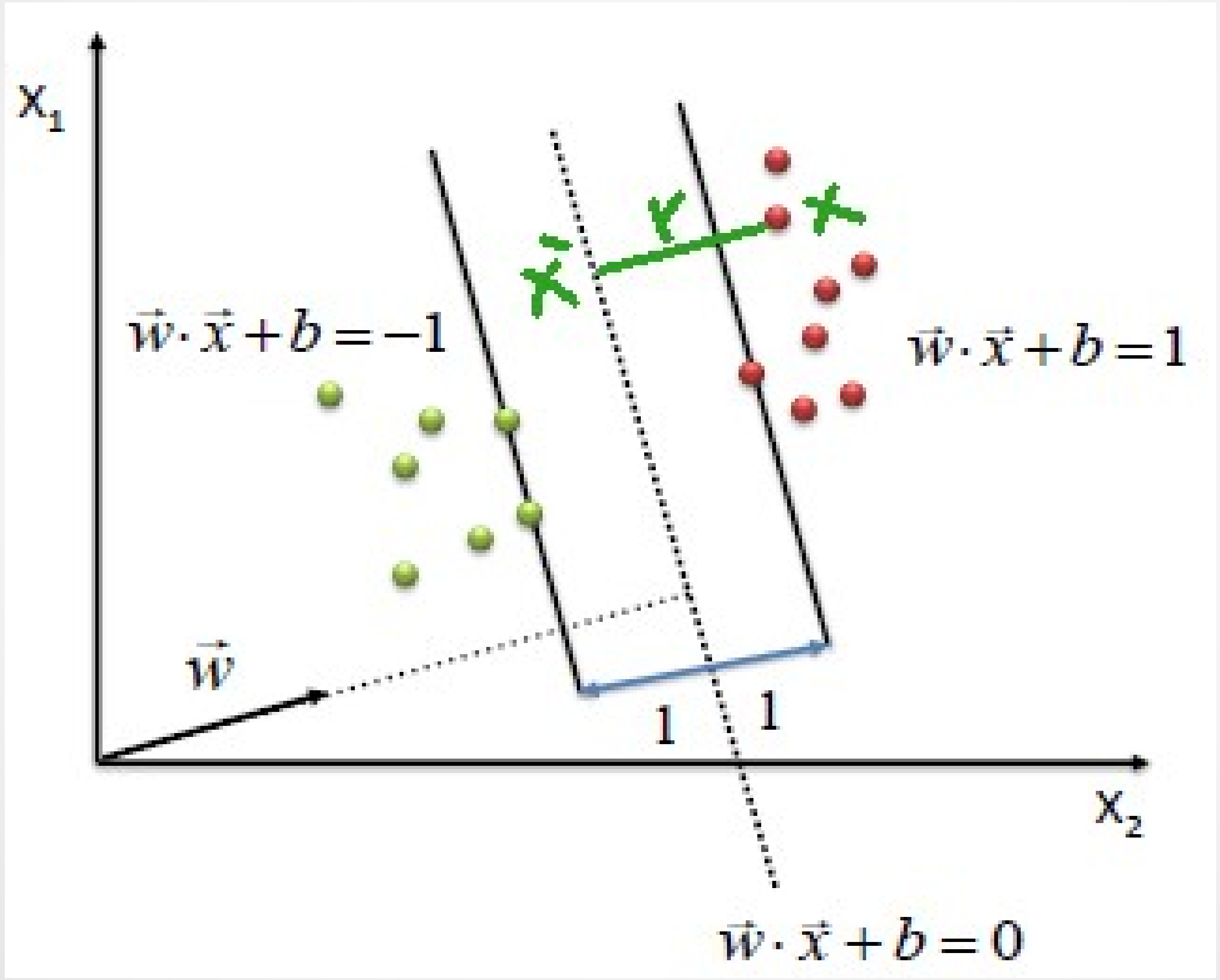
$$\vec{x}' = \vec{x} - yr\frac{\vec{w}}{|\vec{w}|}$$

  multiplying by y just changes the sign for the two cases of $\vec{x}$ being on either side of the decision surface

- Since $\vec{x}'$ lies on the decision boundary, it satisfies: $\vec{w}^T\vec{x}' + b = 0$
- Therefore:

$$\vec{w}^T\left(\vec{x} - yr\frac{\vec{w}}{|\vec{w}|}\right) + b = 0$$

- We solve for r and get: $r = y\frac{\vec{w}^T\vec{x} + b}{|\vec{w}|}$



x_1
x_2
$\vec{w}$
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}\cdot\vec{x}+b=0$
1
1
x^-
r
x

- If we compute **r** for a support vector, then the margin width is

2*r

- The margin is invariant to scaling of parameters because it is normalized by the length of $\vec{w}$
  - Therefore, we could use unit vectors by requiring that

$$|\vec{w}| = 1$$

- Also we can require that the margin of all data points is at least 1 and that it is equal to 1 for at least one data vector
  - That is, for all items in the data: $y_i(\vec{w}^T\vec{x}_i + b) \geq 1$

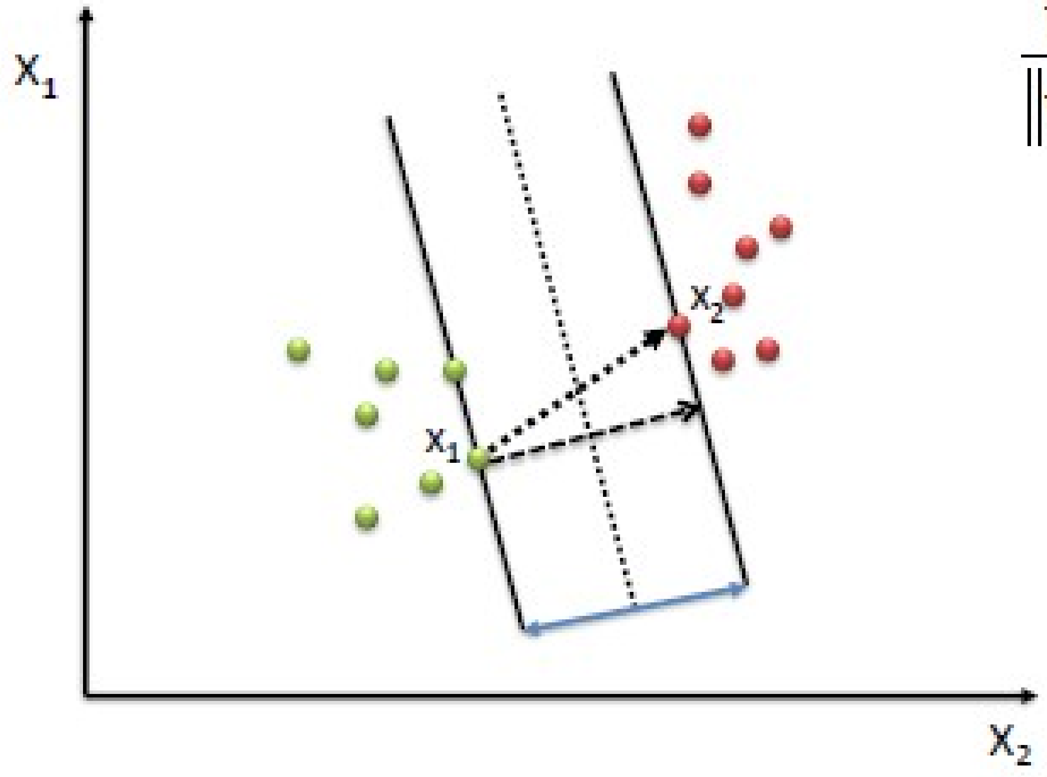
- Since each example's distance from the hyperplane is:

$$r_i = y_i(\vec{w}^T\vec{x}_i + b)/|\vec{w}|$$

- Using all the equations we saw in the last few slides, we conclude that the total margin width is:

$$2/|\vec{w}|$$

Another Derivation



$$\frac{w}{\|w\|}\cdot(x_2 - x_1) = \text{width} = \frac{2}{\|w\|}$$

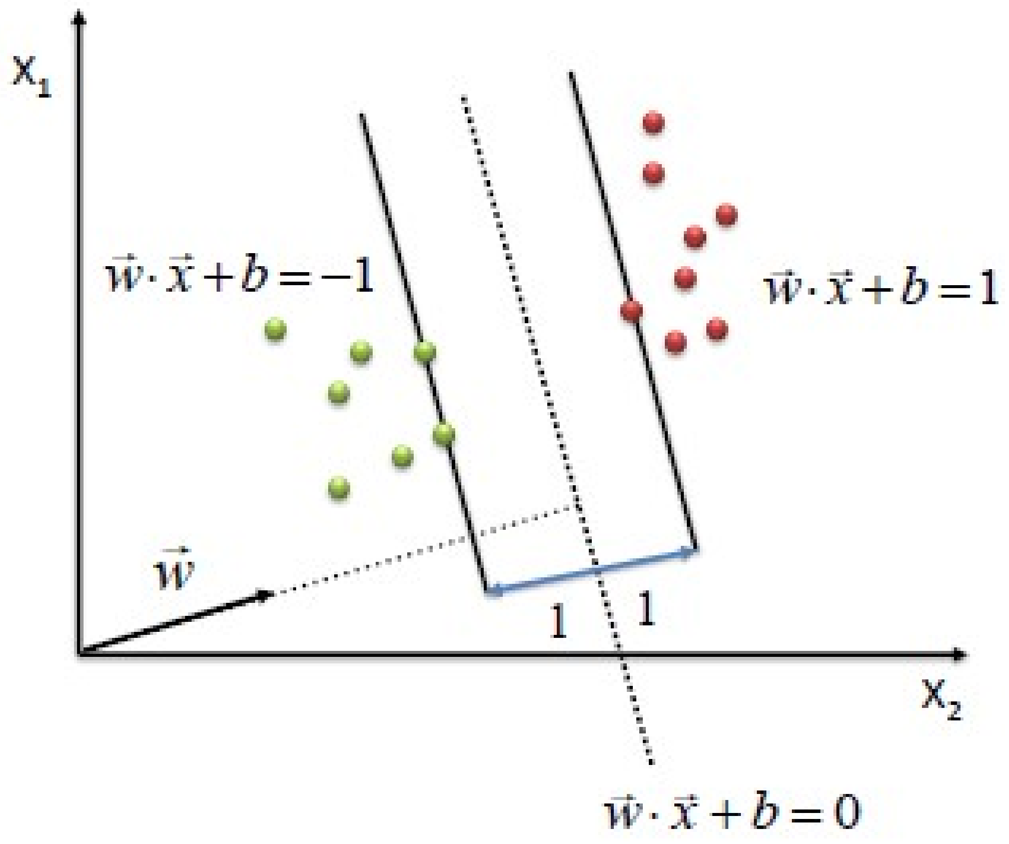
$$w \cdot x_2 + b = 1$$

$$w \cdot x_1 + b = -1$$

$$w \cdot x_2 + b - w \cdot x_1 - b = 1 - (-1)$$

$$w \cdot x_2 - w \cdot x_1 = 2$$

$$\frac{w}{\|w\|}(x_2 - x_1) = \frac{2}{\|w\|}$$


X_1
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}$
1
1
X_2
$\vec{w}\cdot\vec{x}+b=0$

$$\boxed{\max \frac{2}{|w|}}$$

s.t.

$(w\cdot x+b)\geq 1, \forall x$ of class 1

$(w\cdot x+b)\leq -1, \forall x$ of class 2

SVM as a Minimization Problem

- Maximizing $2/|\vec{w}|$ is the same as minimizing $|\vec{w}|/2$
- Hence SVM becomes a minimization problem:

$$\min \frac{1}{2}\|w\|^2$$
$$s.t.\ y_i(w \cdot x_i + b) \geq 1,\ \forall x_i$$

- We are now optimizing a quadratic function subject to linear constraints
- Quadratic optimization problems are a standard, well-known class of mathematical optimization problems, and many algorithms exist for solving them

SVM Algorithm

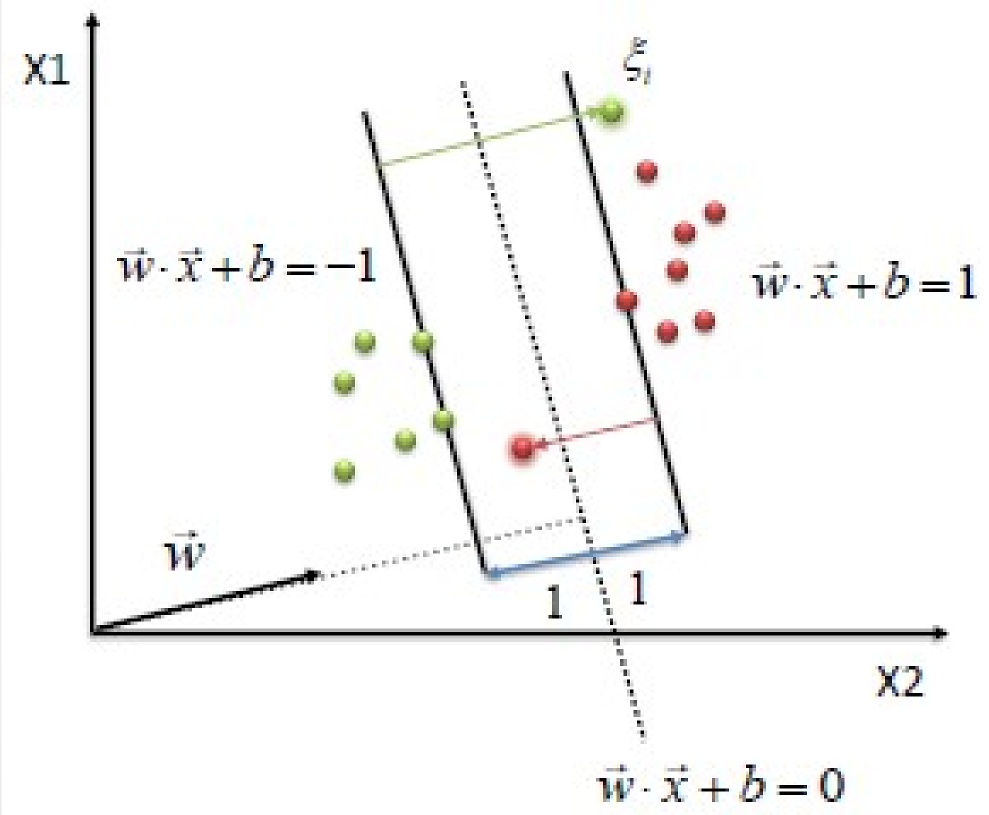
1- Define an optimal hyperplane: maximize margin

2- Extend the above definition for non-linearly separable problems: have a penalty term for misclassifications

3- Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data is mapped implicitly to this space

The beauty of SVM

- The beauty of SVM is that if the data is linearly separable, there is a unique global minimum value
- An ideal SVM analysis should produce a hyperplane that completely separates the vectors (cases) into two non-overlapping classes
- However, perfect separation may not be possible, or it may result in a model with so many cases that the model does not classify correctly
- In this situation SVM finds the hyperplane that maximizes the margin and minimizes the misclassifications



X1
X2
ξ_i
$\vec{w}\cdot\vec{x}+b=-1$
$\vec{w}\cdot\vec{x}+b=1$
$\vec{w}\cdot\vec{x}+b=0$
$\vec{w}$
1
1

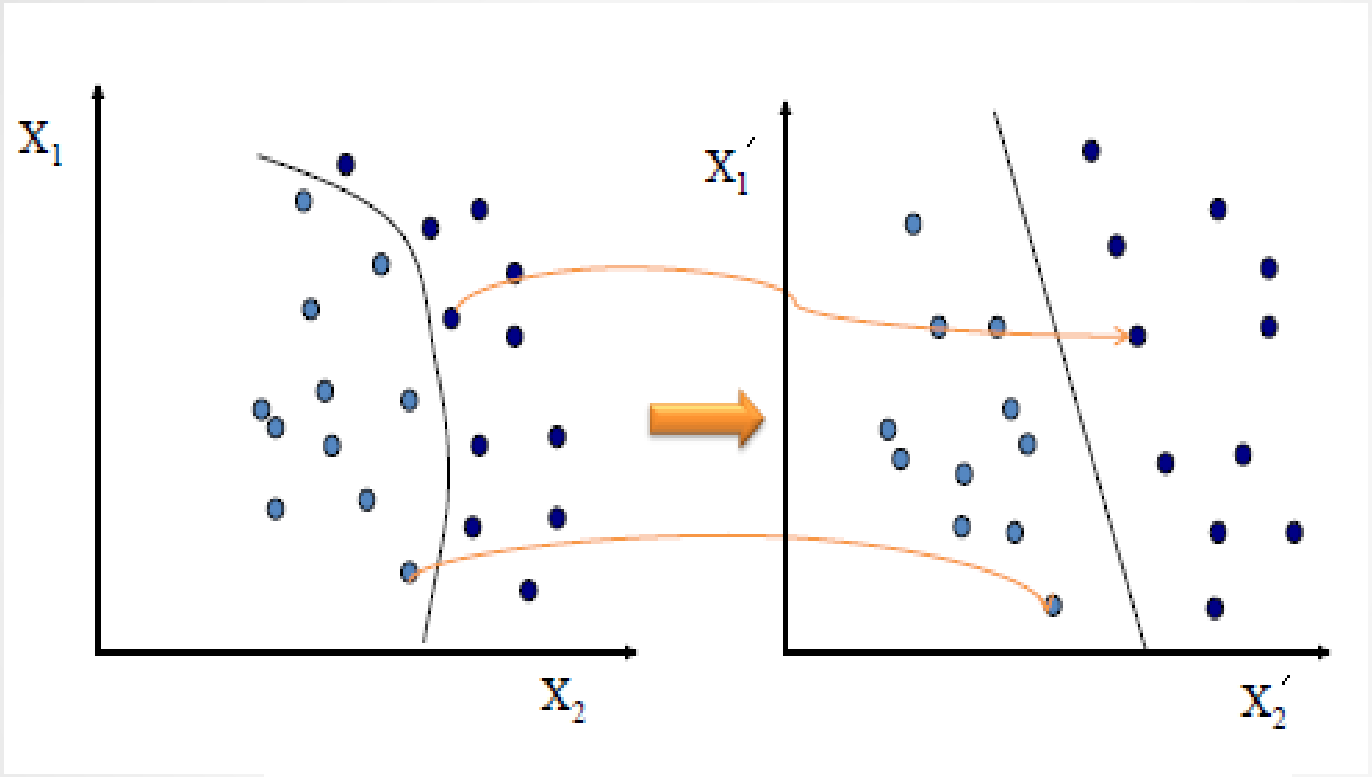
slack variable:

$$\xi_i$$

Allow some instances to fall off the margin, but penalize them

SVM for Nonlinear Separability

- The simplest way to separate two groups of data is with a straight line, flat plane an N-dimensional hyperplane
- However, there are situations where a nonlinear region can separate the groups more efficiently
- SVM handles this by using a kernel function (nonlinear) to map the data into a *different space* where a hyperplane (linear) cannot be used to do the separation
- It means a non-linear function is learned by a linear learning machine in a high-dimensional feature space while the capacity of the system is controlled by a parameter that does not depend on the dimensionality of the space
- This is called **kernel trick** which means the kernel function transform the data into a higher dimensional feature space to make it possible to perform the linear separation



Linear SVM: $x_i \cdot x_j$

Non-linear SVM: $\phi(x_i) \cdot \phi(x_j)$

Kernel function: $k(x_i \cdot x_j)$

Kernel Functions

- Map data into new space, then take the inner product of the new vectors
- The image of the inner product of the data is the inner product of the images of the data
- Two kernel functions are shown below

Polynomial

$$k(\mathrm{x}_i, \mathrm{x}_j) = (\mathrm{x}_i . \mathrm{x}_j)^d$$

Gaussian Radial Basis function

$$k(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(-\frac{\left\| \mathrm{x}_i - \mathrm{x}_j \right\|^2}{2\sigma^2} \right)$$