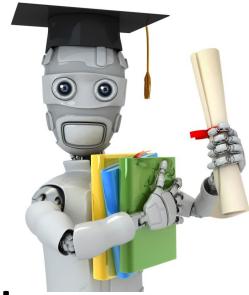


# INTRODUCTION TO MACHINE LEARNING

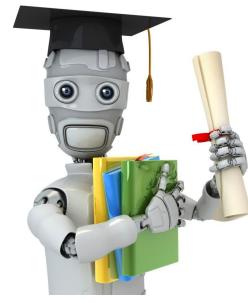
Dr. Srikanth Allamsetty



# Basic Understanding of Machine Learning

- Traditional Programming: Data and program is run on the computer to produce the output.
- Machine Learning: Data and output is run on the computer to create a program. This program can be used in traditional programming.
- Machine Learning is an application of artificial intelligence where a computer/machine learns from the past experiences and makes future predictions.
  - The past experience is developed through the data collected.
  - In order to perform the task, the system learns from the data-set provided.
  - A data-set is a collection of many examples.
  - An example is a collection of features.
- Machine Learning refers to the techniques involved in dealing with vast data in the most intelligent fashion (by developing algorithms) to derive actionable insights.

# Basic Understanding of Machine Learning



## Brief History

- **1950**
- Samuel developed checker playing program and he coined the term Machine learning
- **1960**
- Neural Network- Rosenblatt perceptron
- Pattern Recognition
- Minsky and Papert proved limitation of Perceptron
- **1970**
- Decision tree – J.R Quinlan
- Natural Language processing
- **1980**
- Advanced decision tree and rule based learning
- **Resurgence** of neural network- Multilayer perceptron and neural network specific back propagation algorithm was developed
- PAC- probably approximate correct learning
- **1990 ( Machine learning embraced statistics to a large extent)**
- SVM- 1995
- Data mining
- Adaptive agents and web applications
- Text learning
- Reinforcement learning
- Ensembles
- Bayes learning

# Basic Understanding of Machine Learning

## APPLICATIONS OF MACHINE LEARNING



INTELLIGENT AGENTS,  
NATURAL LANGUAGE  
PROCESSING ETC.

Virtual Assistant



SENTIMENT ANALYSIS,  
FILTERING SPAM ETC.

Social Media



CUSTOMER  
SUPPORT,  
PRODUCT  
RECOMMENDATION  
, ADVERTISING,

eCommerce



Machine  
Learning  
Applications

SAFETY  
MONITORING,  
AIR TRAFFIC  
CONTROL ETC.

Transport



Healthcare



DRUG DISCOVERY,  
DISEASE DIAGNOSIS,  
ROBOTIC SURGERY

Financial Services



ALGORITHMIC TRADING,  
PORTFOLIO MANAGEMENT,  
FRAUD DETECTION

# Basic Understanding of Machine Learning



## Many domains and application

- **Medicine**
  - Diagnose a disease
    - Input: Symptoms , Lab measurement, test results, DNA tests,.....
    - Output: one of set of possible disease , or none of the above
  - Data mine historical medical records to learn which future patients will respond best to which treatment.
- **Robot control**
  - Design autonomous mobile robots that learn to navigate from their own experience
- **Financial**
  - Predict if a stock will rise or fall in few millisec
  - Predict if a user will click on an ad or not in order to decide which ad to show.
- **Application in Business intelligence**
  - Robustly forecasting product sale quantities taking seasonality and trend into account
  - Identify price sensitivity of a consumer product and identify the optimum price point that maximizes the net profit.
- **Some other applications**
  - Fraud detection: Credit card providers determines whether or not someone will default.
  - Understanding the consumer sentiment based on unstructured text data.
  - Self customized program
    - e.g. Amazon, Netflix product recommendation
    - Algorithm learns by itself to customize.

# Basic Understanding of Machine Learning

- Examples of applications in diverse fields
- Machine Vision
- Biometric Recognition
- Handwriting Recognition
- Medical Diagnosis
- Alignment of Biological Sequences
- Drug Design
- Speech Recognition
- Text Mining
- Natural Language Processing
- Fault Diagnostics
- Load Forecasting
- Control and Automation
- Business Intelligence

# Basic Understanding of Machine Learning

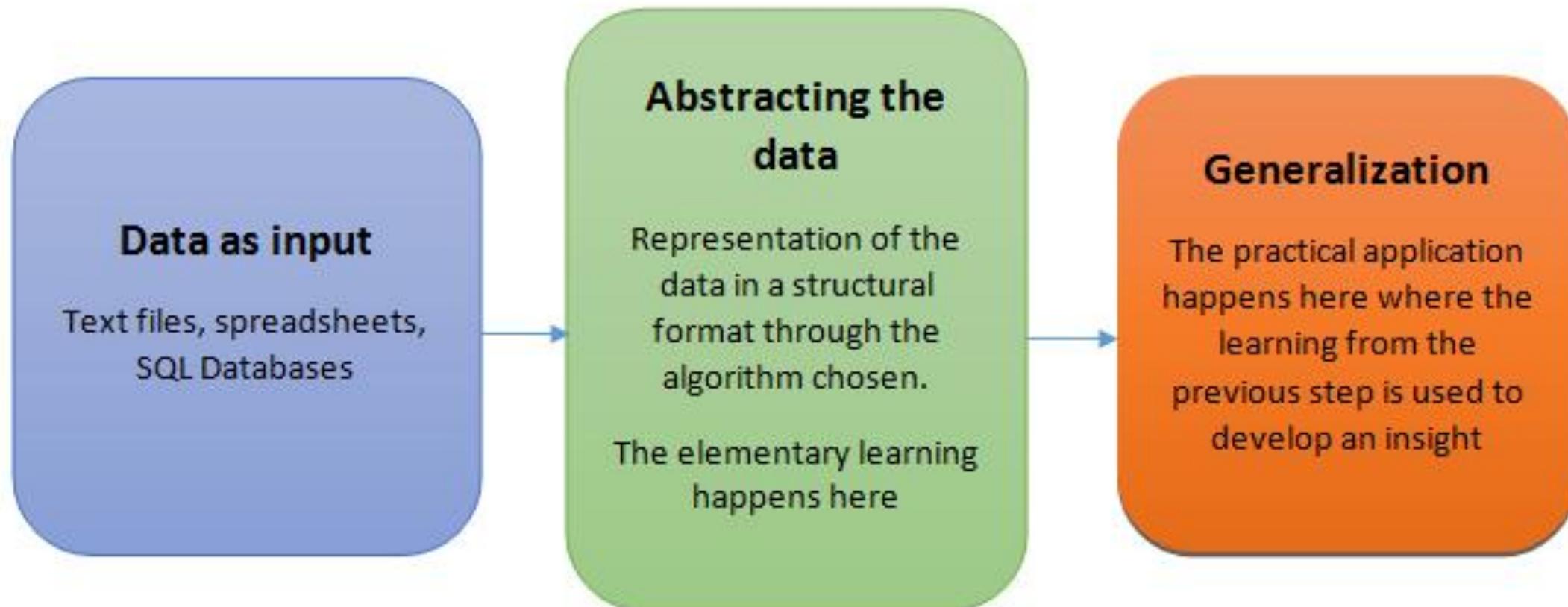
- The field of machine learning is concerned with the question of how to construct computer programs that automatically **improve with experience**.
- In recent years many successful machine learning applications have been developed, ranging from **data-mining programs** that learn to detect fraudulent credit card transactions, to **information-filtering systems** that learn users' reading preferences, to **autonomous vehicles** that learn to drive on public highways.
- At the same time, there have been important **advances** in the theory and algorithms that form the foundations of this field.
- Machine learning **draws on concepts and results from** many fields, including statistics, artificial intelligence, philosophy, information theory, biology, cognitive science, computational complexity, and control theory.

# Basic Understanding of Machine Learning

- **How is machine learning different from \_\_\_\_\_**
- **Artificial Intelligence:** It refers to the procedure of programming a computer (machine) to take rational.
  - to make the machine behave in an excellent fashion in lieu of human guidance.
  - ML is a subset of AI.
- **Statistics:** utilizes data to carry out the analysis and present inferences.
  - regression, variance, standard deviation, conditional probability etc.
  - Machine learning algorithms uses statistical concepts to execute machine learning
- **Deep Learning:** associated with a ML algorithm (ANN) which uses the concept of human brain to facilitate the modeling of arbitrary functions.
  - ANN requires a vast amount of data and this algorithm is highly flexible when it comes to model multiple outputs simultaneously.
- **Data Mining:** deals with searching specific information
  - ML solely concentrates on performing a given task

# Basic Understanding of Machine Learning

- The process of teaching machines can be broken down into 3 parts



# Basic Understanding of Machine Learning

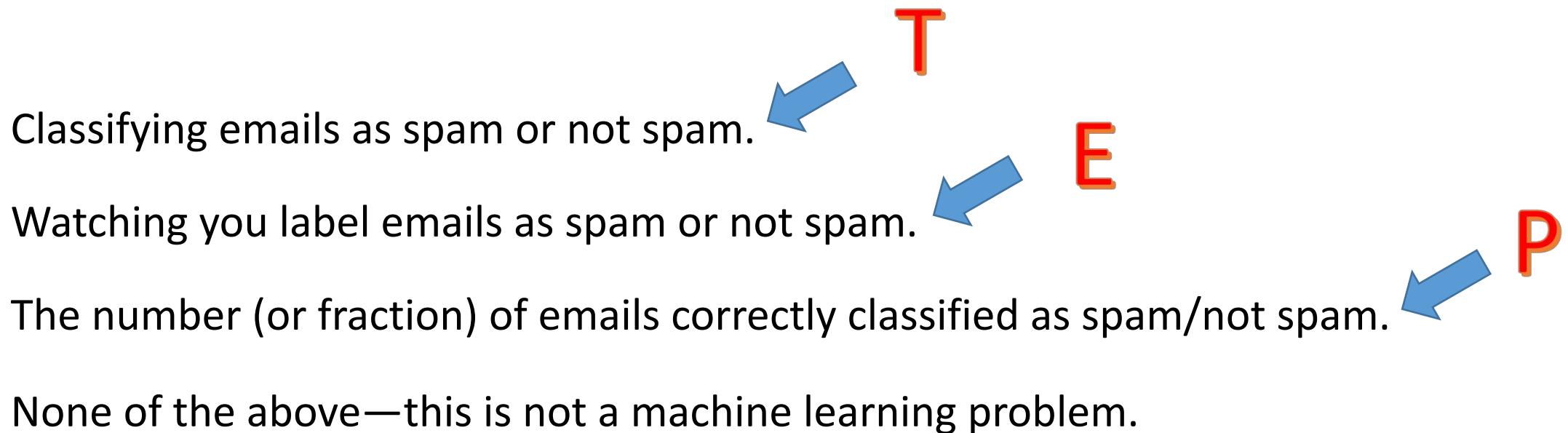
- If we could understand how to program to learn (to improve automatically with experience) the impact would be dramatic.
- Imagine computers
  - learning from medical records which treatments are most effective for new diseases,
  - houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants,
  - personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper.
- A successful understanding of how to make computers learn would open up many new uses of computers and new levels of competence and customization.
- algorithms have been invented that are effective for certain types of learning tasks, and a theoretical understanding of learning is beginning to emerge.

# Basic Understanding of Machine Learning

- Machine learning addresses the question of how to build computer programs that improve their performance at some task through experience.
- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E

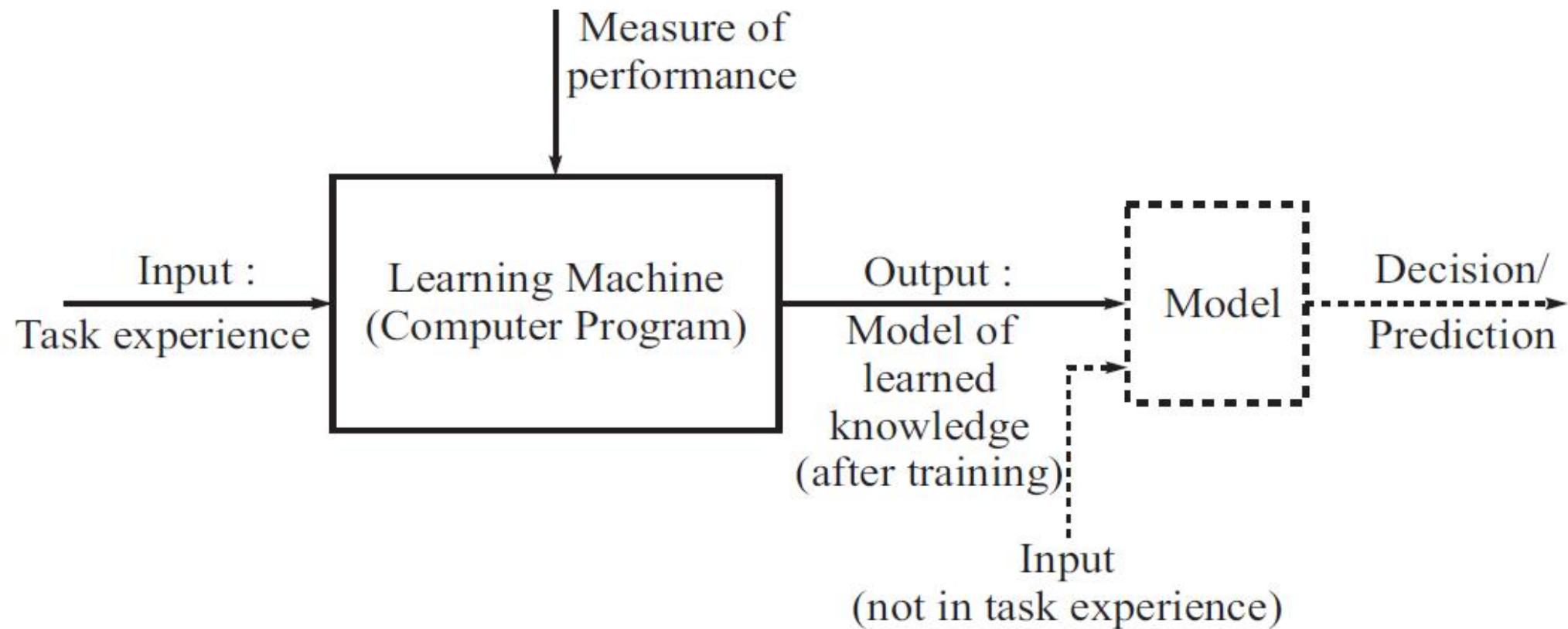
“A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.”

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?

- 
- The diagram illustrates the components of machine learning. At the top right is a red letter 'T'. A blue arrow points from the text 'Classifying emails as spam or not spam.' to 'T'. Below 'T' is a red letter 'E'. A blue arrow points from the text 'Watching you label emails as spam or not spam.' to 'E'. To the right of 'E' is a red letter 'P'. A blue arrow points from the text 'The number (or fraction) of emails correctly classified as spam/not spam.' to 'P'.
- Classifying emails as spam or not spam. 
  - Watching you label emails as spam or not spam. 
  - The number (or fraction) of emails correctly classified as spam/not spam. 
  - None of the above—this is not a machine learning problem.

# Basic Understanding of Machine Learning

- Block diagrammatic representation of a learning machine



# Basic Understanding of Machine Learning

- Steps used in Machine Learning

1. Collecting data: Be it the raw data from excel, access, text files etc., this step (gathering past data) forms the foundation of the future learning.
  - The better the variety, density and volume of relevant data, better the learning prospects for the machine becomes.
2. Preparing the data: Any analytical process thrives on the quality of the data used.
  - One needs to spend time determining the quality of data and then taking steps for fixing issues such as missing data and treatment of outliers.
3. Training a model: This step involves choosing the appropriate algorithm and representation of data in the form of the model.
  - The cleaned data is split into two parts – train and test (proportion depending on the prerequisites);
  - the first part (training data) is used for developing the model.
  - The second part (test data), is used as a reference.

# Basic Understanding of Machine Learning

- Steps used in Machine Learning

4. Evaluating the model: To test the accuracy, the second part of the data (holdout / test data) is used.

- This step determines the precision in the choice of the algorithm based on the outcome.
- A better test to check accuracy of model is to see its performance on data which was not used at all during model build.

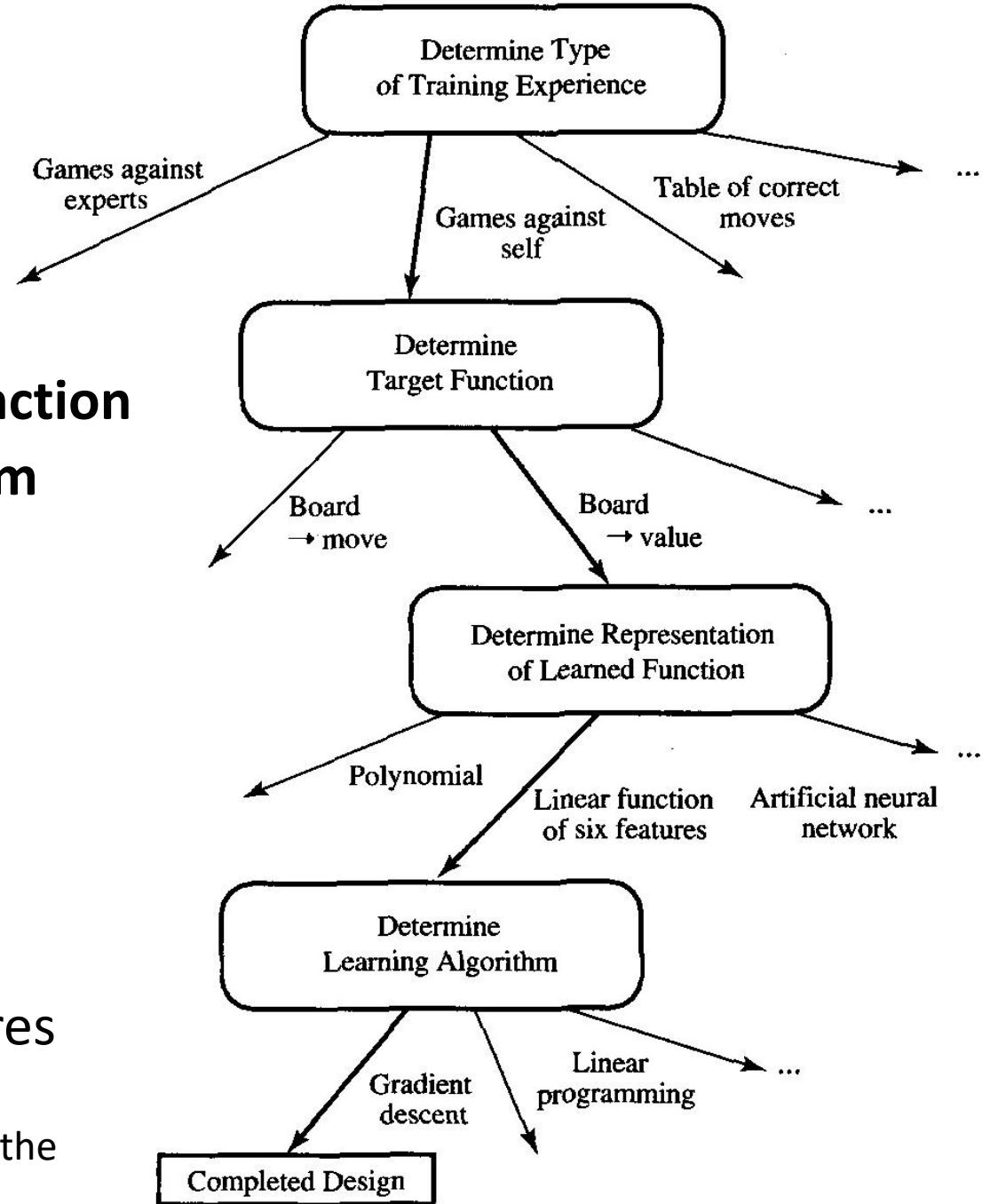
5. Improving the performance: This step might involve choosing a different model altogether or introducing more variables to augment the efficiency.

- That's why significant amount of time needs to be spent in data collection and preparation.

# Formulating a Machine Learning Problem and Models: Special Emphasis on Target Function

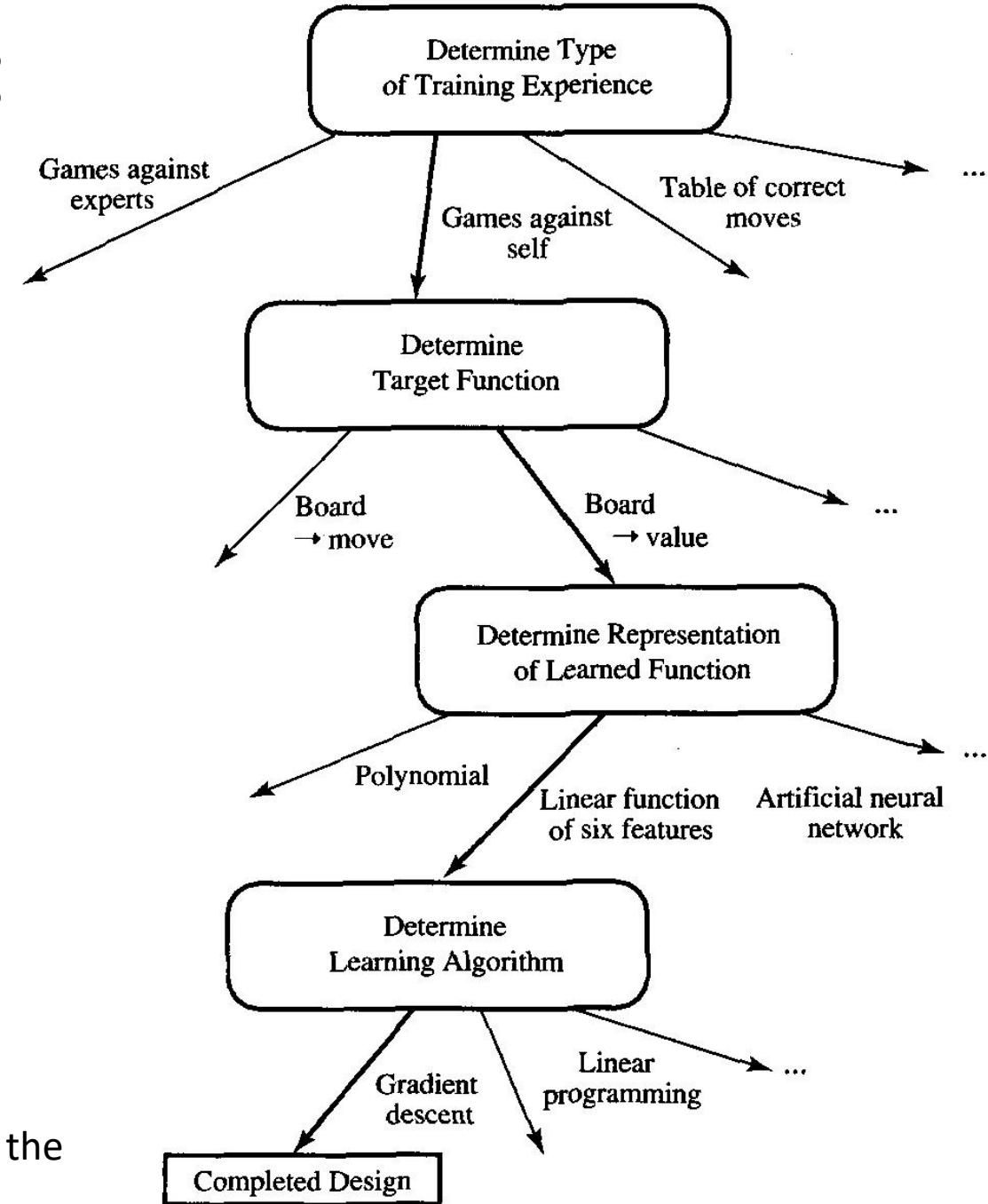
# A checkers learning problem:

- Designing any learning system includes
  - **Choosing the Training Experience**
  - **Choosing the Target Function**
  - **Choosing a Representation for the Target Function**
  - **Choosing a Function Approximation Algorithm**
    - **ESTIMATING TRAINING VALUES**
    - **ADJUSTING THE WEIGHTS**
  - **The Final Design**
- Few Other Examples:
  - Learning to recognize spoken words
  - Learning to drive an autonomous vehicle
  - Learning to classify new astronomical structures
  - Learning to play world-class backgammon
    - Summary of choices in designing the checkers learning program



# A checkers learning problem:

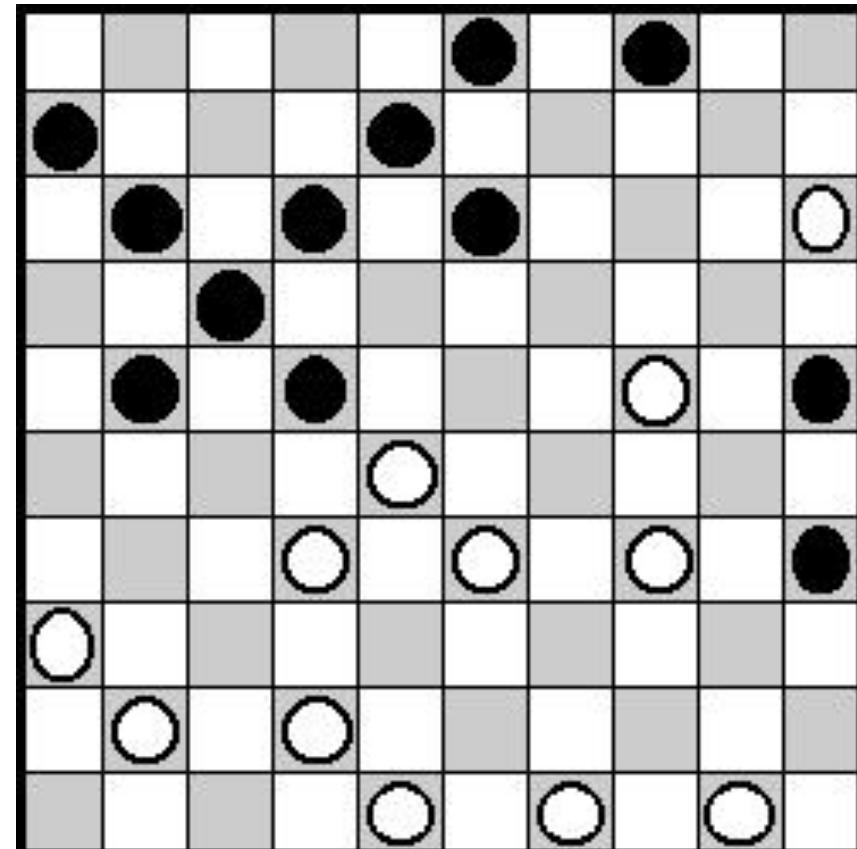
- A checkers learning problem:
  - **Task T: playing checkers**
  - **Performance measure P: percent of games won against opponents**
  - **Training experience E: playing practice games against itself**
- In order to complete the design of the learning system, we must now choose
  - the exact type of knowledge to be learned
  - a representation for this target knowledge
  - a learning mechanism
- Summary of choices in designing the checkers learning program



# Designing a program to learn to play checkers

## Choosing the Training Experience:

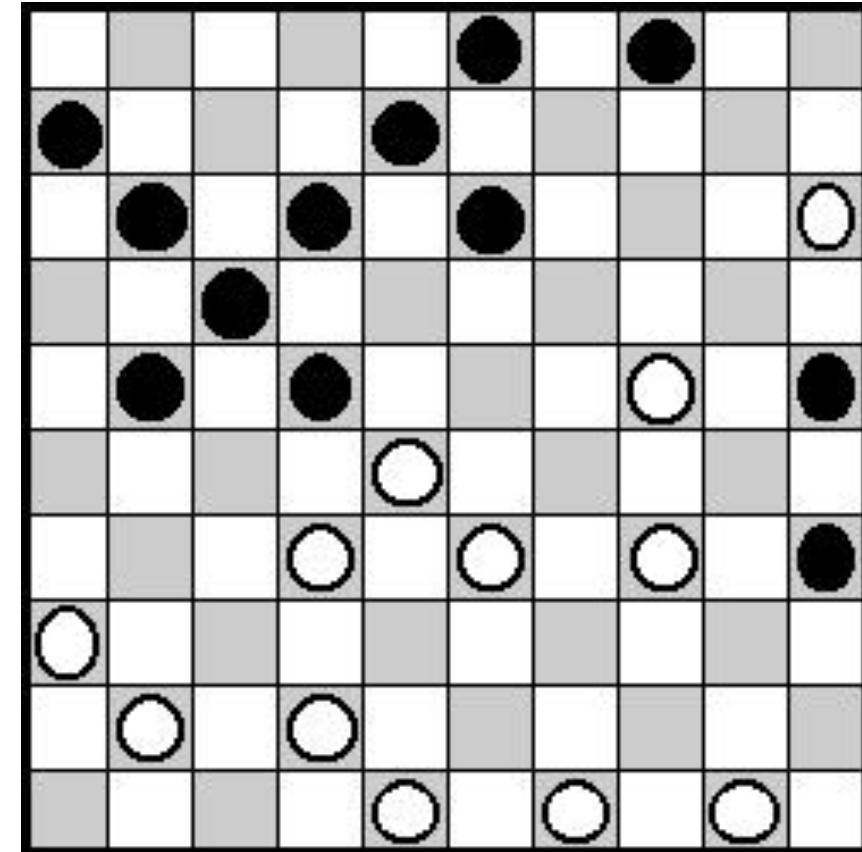
- The system might learn from direct training examples consisting of individual checkers board states and the correct move for each.
- Alternatively, it might have available only indirect information consisting of the move sequences and final outcomes of various games played.
- In this later case, information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.
- Due to Credit assignment problem, learning from direct training feedback is typically easier than learning from indirect feedback.



# Designing a program to learn to play checkers

## Choosing the Training Experience:

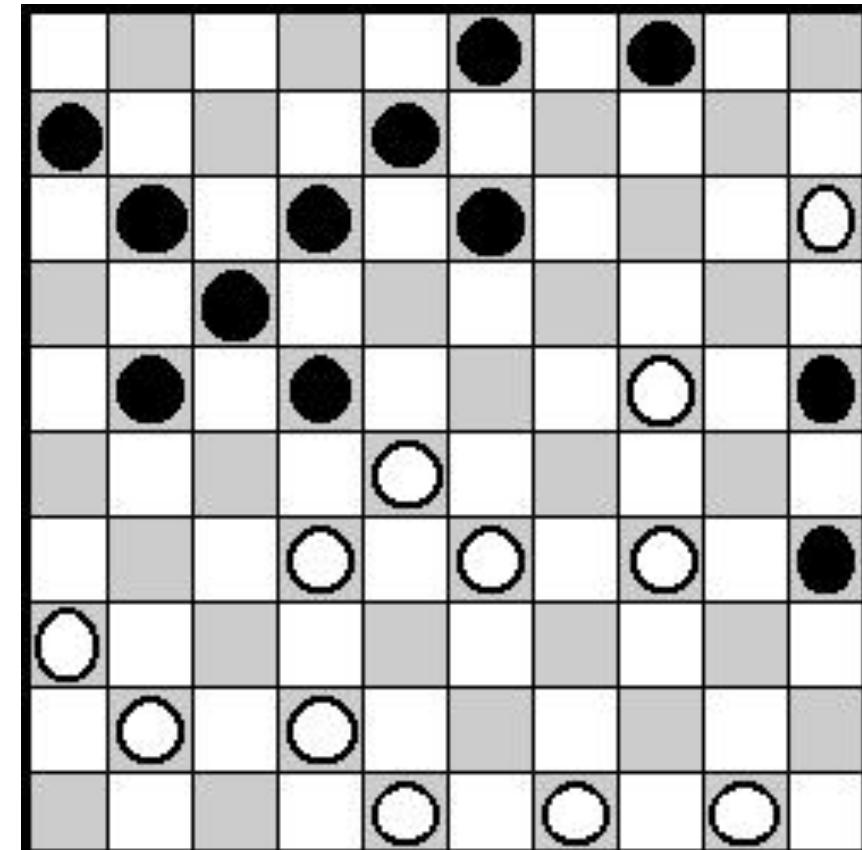
- A second important attribute of the training experience is the degree to which the learner controls the sequence of training examples.
- A third **important attribute of the training experience is how well it represents the distribution of examples over which the final system performance  $P$  must be measured.**
- To proceed with our design, let us decide that our system will train by playing games against itself.
- This has the advantage that no external trainer need be present, and it therefore allows the system to generate as much training data as time permits.



# Designing a program to learn to play checkers

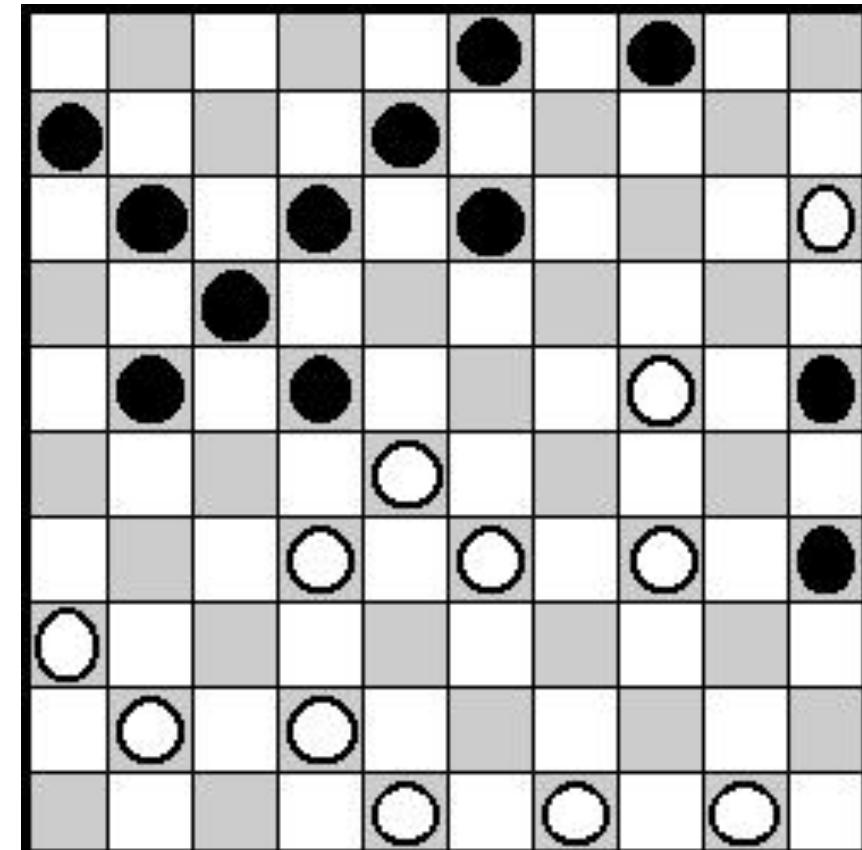
## Choosing the Target Function:

- The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.
- Let us begin with a checkers-playing program that can generate the legal moves from any board state.
- **The program needs only to learn how to choose the best move from among these legal moves.**
- This learning task is representative of a large class of tasks for which the legal moves that define some large search space are known a priori, but for which the best search strategy is not known.



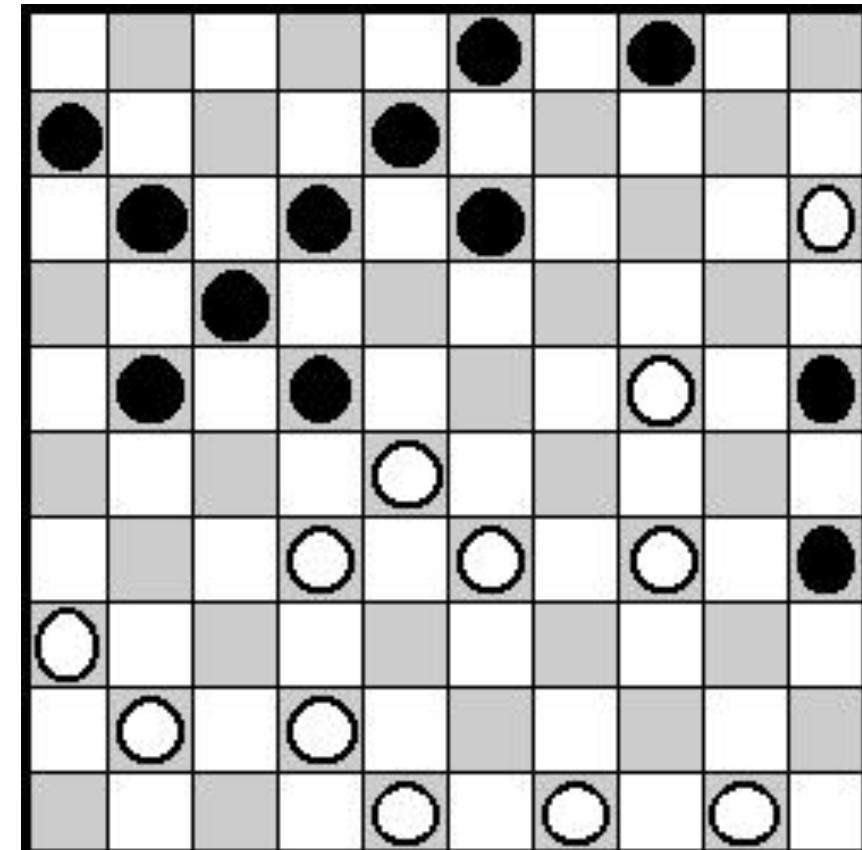
# Designing a program to learn to play checkers

- **Choosing the Target Function:**
- The most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.
- Another choice is an **evaluation function** that assigns a **numerical score** to any given board state (**higher scores to better board states**).
- learning task in this case to the problem of discovering an operational description of the ideal target function  $V$ .
- **learning algorithms acquires only some approximation to the target function, thus the process of learning the target function is often called function approximation.**



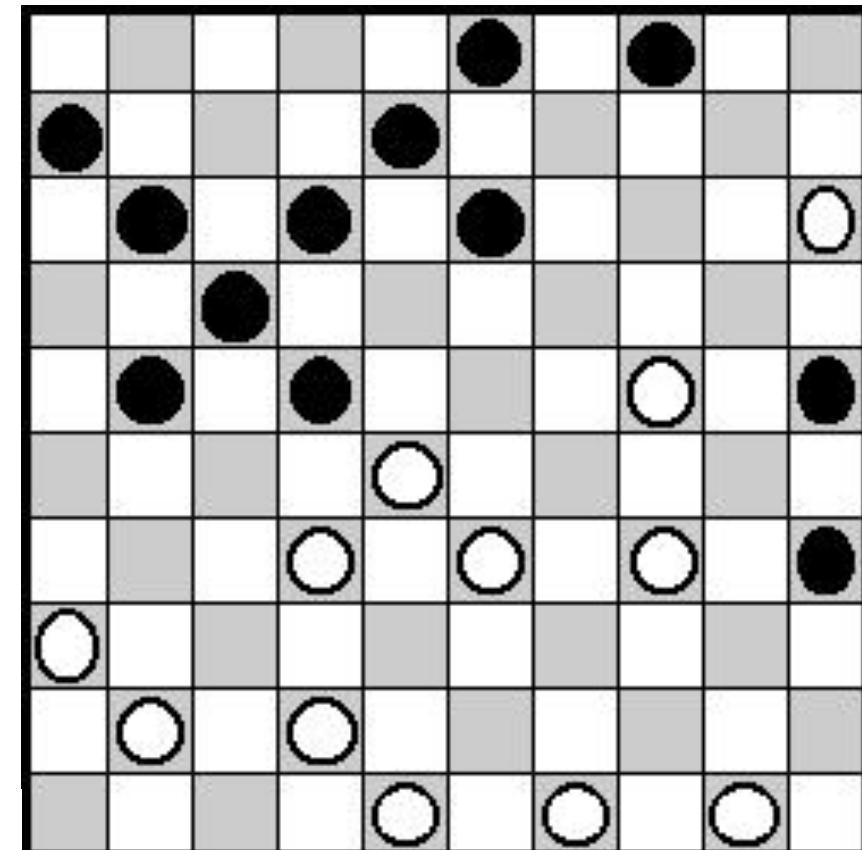
# Designing a program to learn to play checkers

- Choosing a Representation for the Target Function:
- We again have many options to represent TF:
  - using a large table with a distinct entry specifying the value for each distinct board state.
  - using a collection of rules that match against features of the board state
  - using a quadratic polynomial function of predefined board features
  - using an artificial neural network.
- We wish to pick a very expressive representation to allow representing **as close an approximation as possible to the ideal target function**.
- The more expressive the representation, the more training data the program will require.



# Designing a program to learn to play checkers

- Choosing a Representation for the Target Function:
  - for example, we can calculate the function  $\hat{V}$  as a linear combination of the following board features:
    - $x_1$ : the number of black pieces on the board
    - $x_2$ : the number of white pieces on the board
    - $x_3$ : the number of black kings on the board
    - $x_4$ : the number of white kings on the board
    - $x_5$ : the number of black pieces threatened by white (i.e., which can be captured on white's next turn)
    - $x_6$ : the number of white pieces threatened by black
- $$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$



where  $w_0$  through  $w_6$  are numerical coefficients, or weights, to be chosen by the learning algorithm.

# Designing a program to learn to play checkers

- Choosing a Function Approximation Algorithm:

- Estimating training values

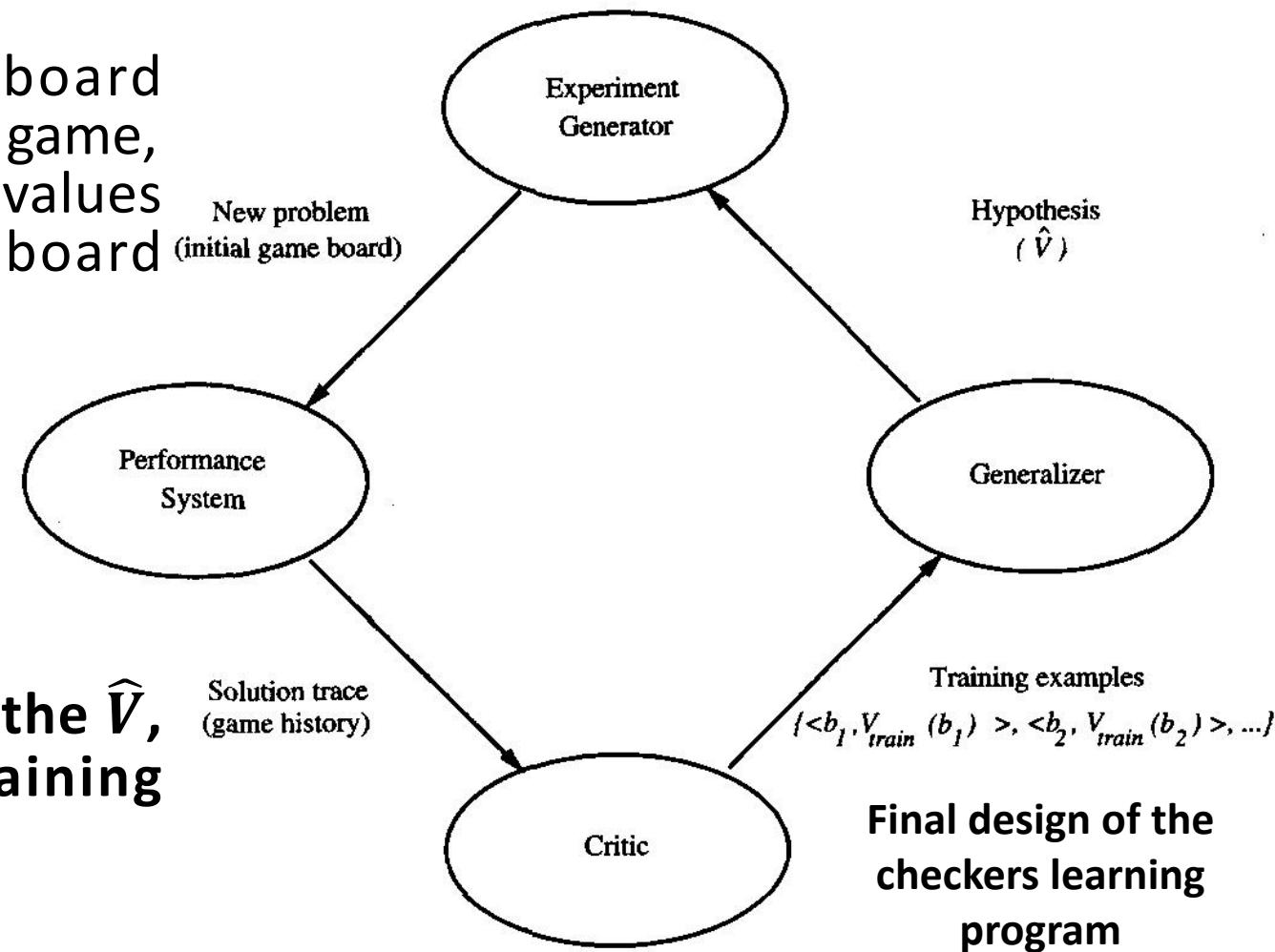
- While it is easy to assign a value to board states that correspond to the end of the game, it is less obvious how to assign training values to the more numerous intermediate board states that occur before the game's end.

- Simple solution:  $V_{train}(b) \leftarrow \hat{V}(\text{Successor}(b))$

- Adjusting the weights

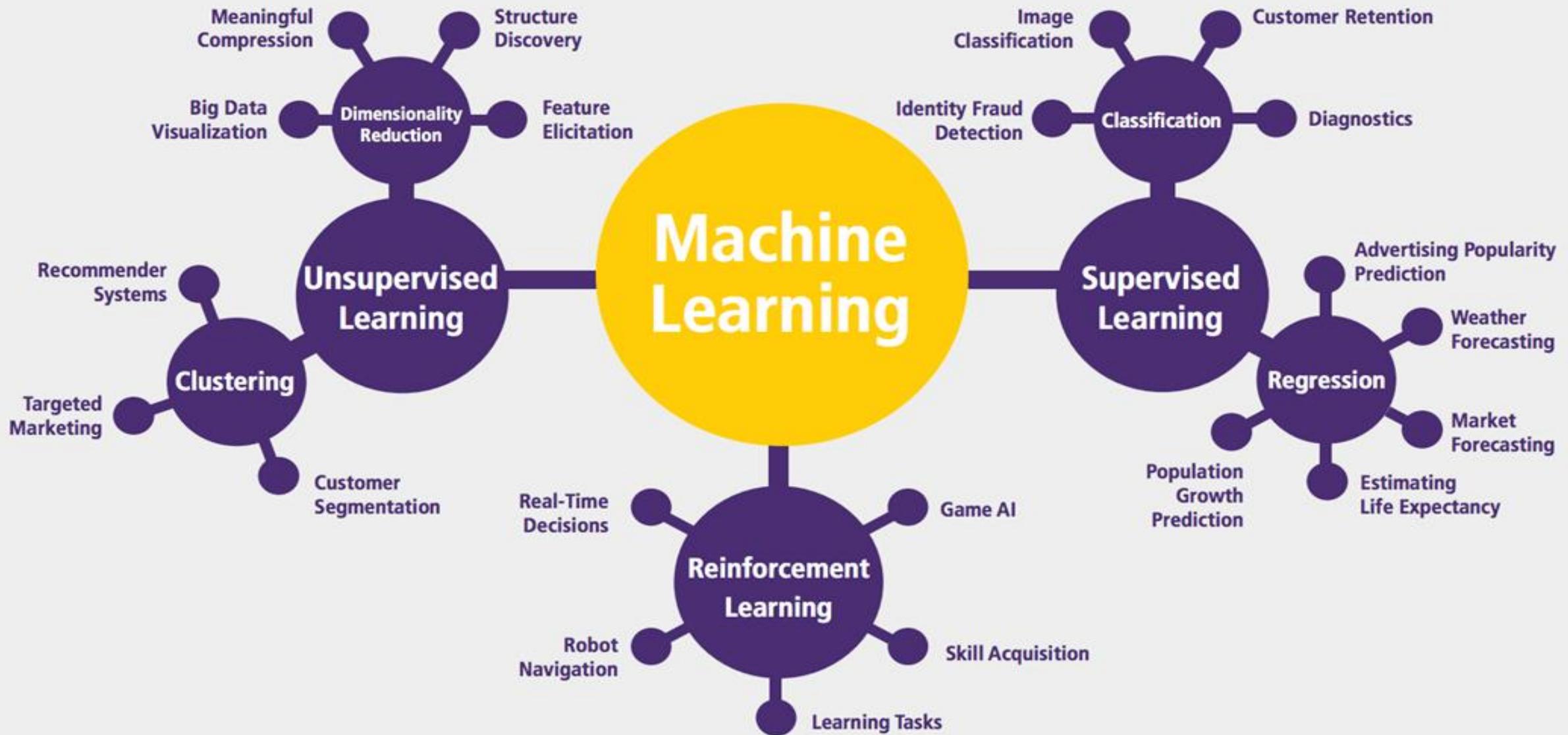
$$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in \text{training examples}} (V_{train}(b) - \hat{V}(b))^2$$

- We seek the weights, or equivalently the  $\hat{V}$ , that minimize E for the observed training examples.



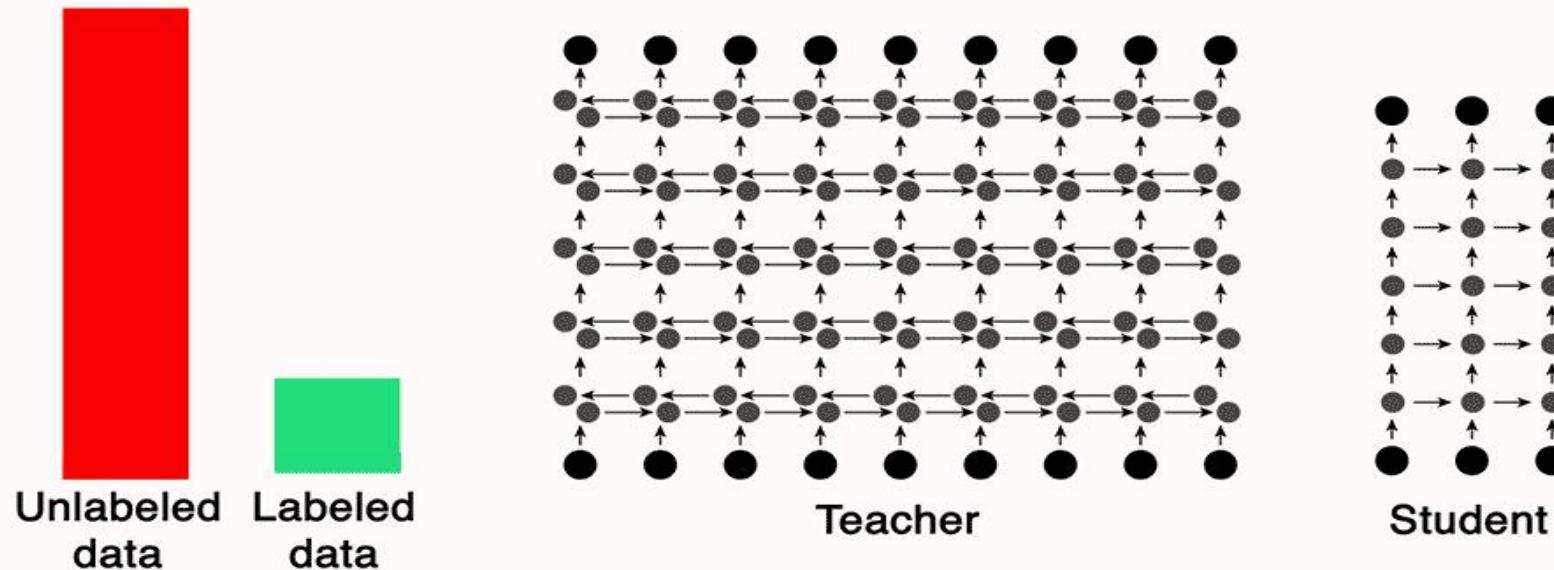
# Type of Machine Learning Problem: Supervised, Unsupervised and Reinforced

# Types of Machine Learning algorithms



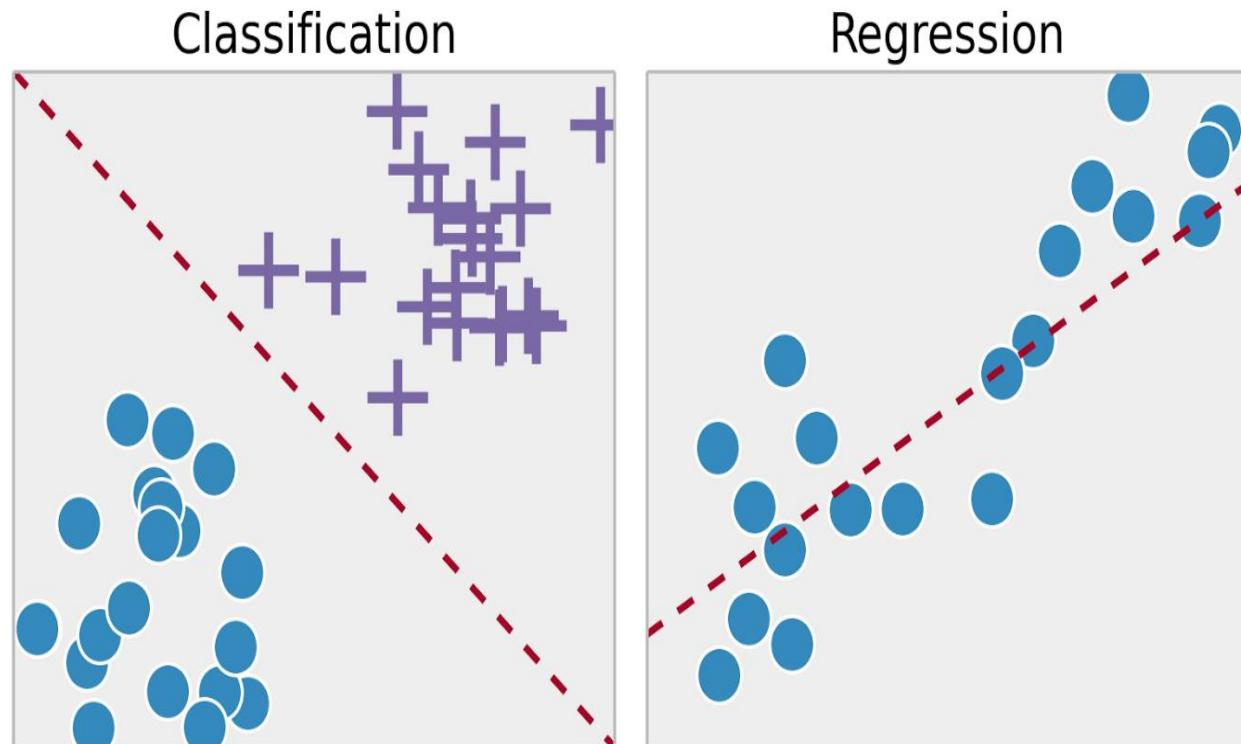
# Supervised/Directed Learning

- In supervised learning the machine experiences the examples along with the labels or targets for each example.
- The labels in the data help the algorithm to correlate the features.



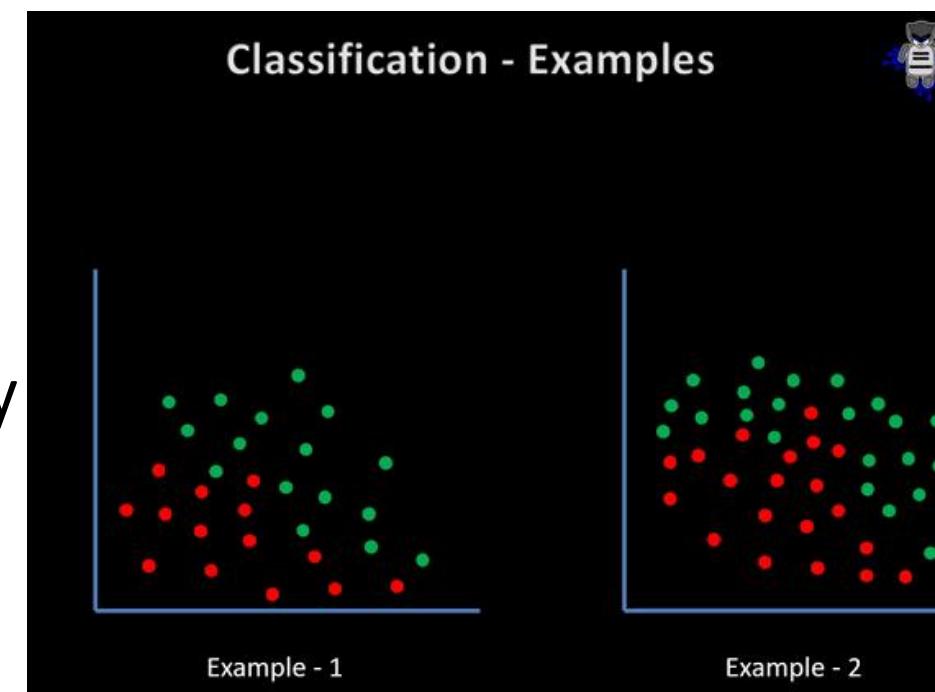
# Supervised/Directed Learning

- Two of the most common supervised machine learning tasks are:
- **Classification:** machine learns to predict discrete values
  - predicting whether a stock's price will rise or fall
  - deciding if a news article belongs to the politics or leisure section
- **Regression:** machine predicts the value of a continuous response variable
  - predicting the sales for a new product
  - the salary for a job based on its description



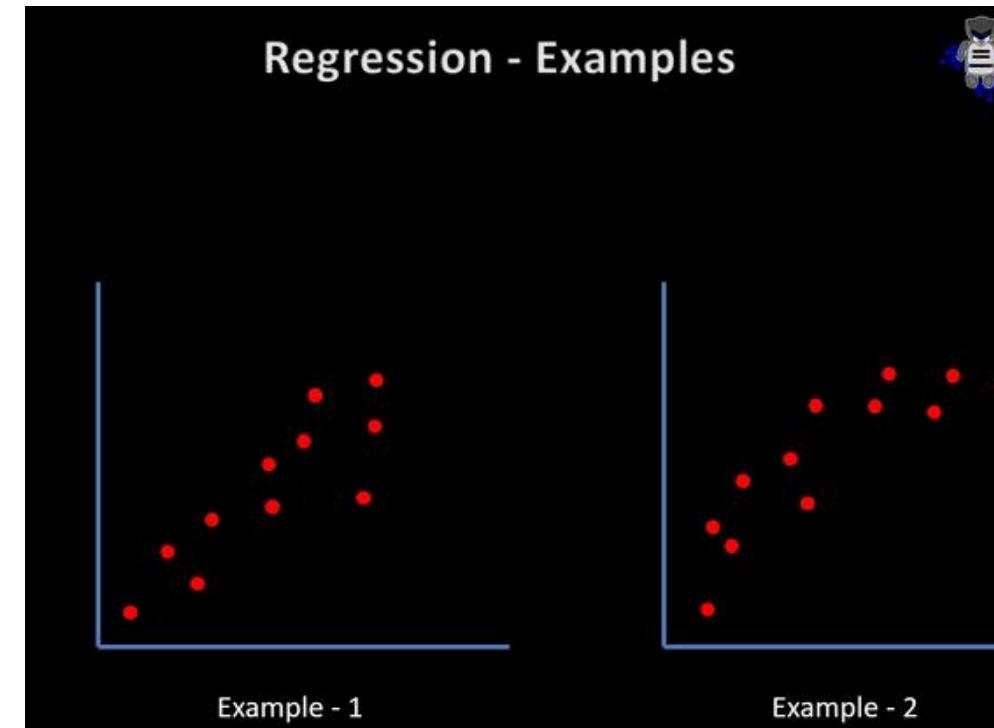
# Supervised/Directed Learning

- **Classification:**
- A classification problem is when the output variable is a category/class and the goal is to classify the input into the known categories/classes.
- It basically aims to predict which class the input corresponds to.
- Some examples include systems where we seek a yes-or-no prediction, such as:
  - “Is this tumour cancerous?”,
  - “Does this product meet our quality standards?”



# Supervised/Directed Learning

- **Regression:**
- A regression problem is when the output variable is a real and continuous value. Here, the goal is to predict the output value given an input  $x$ .
- Some examples include systems where the value being predicted falls somewhere on a continuous spectrum.
  - predicting the stock price of a company
  - predicting the temperature tomorrow based on historical data.



# Supervised/Directed Learning

- **Commonly used algorithms for classification and regression**

## Regression

Simple Linear Regression

Multiple Linear Regression

Polynomial Linear Regression

Support Vector Regression

Decision Tree Regression

Random Forest Regression

Neural Network

## Classification

Logistic Regression

K Nearest Neighbors

Support Vector Machine

Naïve Bayes

Decision Tree Classification

Random Forest Classification

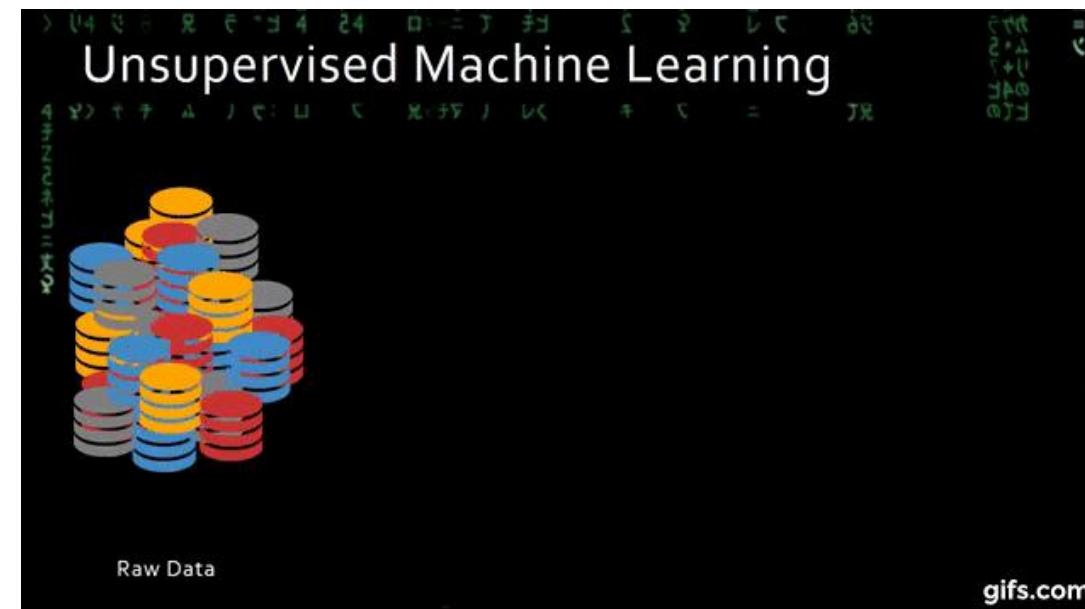
Neural Network

# Unsupervised/Undirected Learning

- When we have unclassified and unlabeled data, the system attempts to uncover patterns from the data .
- There is no label or target given for the examples.
- One common task is to group similar examples together called clustering.
- clustering and association learning belong to this category

# Unsupervised/Undirected Learning

- **Clustering:**
- Here the goal is to find similarities in the dataset and group similar data points together.
- Cluster is the collection of data objects which are similar to one another within the same group (class or category) and are different from the objects in the other clusters.
- This method can also be used to detect anomalies that do not fit to any group.
  - In marketing, customers are segmented according to similarities to carry out targeted marketing.
  - Given a collection of text, we need to organize them, according to the content similarities to create a topic hierarchy.
  - Detecting distinct kinds of pattern in image data (Image processing).
  - It's effective in biology research for identifying the underlying patterns.



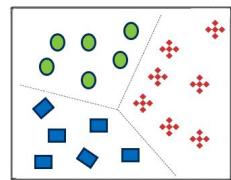
# Unsupervised/Undirected Learning

- **Association learning:**
- In association learning, any relation between observations is required, not merely association capable of predicting a specific class value.
- Here the aim is to discover rules that describe large portions of our data, such as people that buy X also tend to buy Y.
- A classic example of association rules would be market basket analysis, like, a person who buys biryani and burgers usually buys a soft drink too.

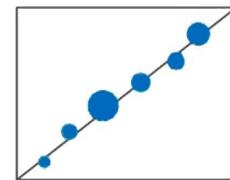
Of the following examples, which would you address using an unsupervised learning algorithm? (Check all that apply.)

- Given email labeled as spam/not spam, learn a spam filter.
- Given a set of news articles found on the web, group them into set of articles about the same story.
- Given a database of customer data, automatically discover market segments and group customers into different market segments.
- Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.

# SUPERVISED LEARNING

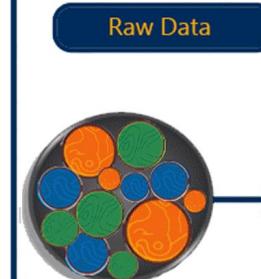


**Classification**  
Is this data input red, blue or green?

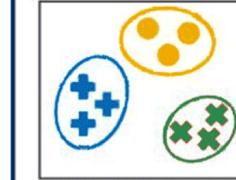


**Regression**  
What is the impact of product price on number of sales?  
What is the impact of years of experience on salary?

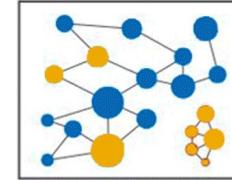
# UNSUPERVISED LEARNING



## EXAMPLES



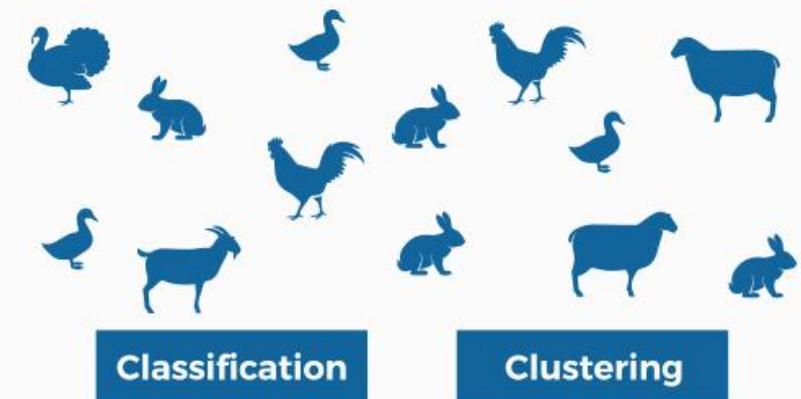
**CLUSTERING**  
Identifies similarities in groups:  
Are there patterns in the data that indicate which groups to target?



**ANOMALY DETECTION**  
Identifies abnormalities in dataset:  
Is the user behaving as it should? Is a hacker intruding the network?

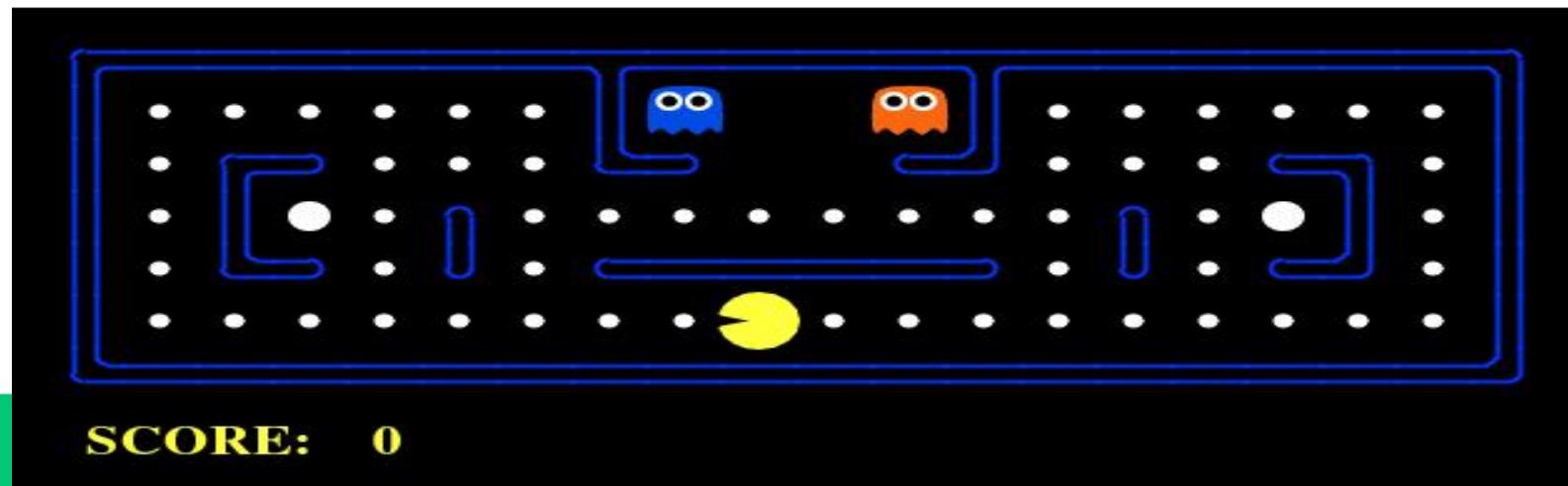
# Clustering vs classification

- In the case of Classification, there are predefined labels assigned to each input instance according to their properties
  - whereas in clustering those labels are missing.
- The process of classifying the input instances based on their corresponding class labels is known as classification
  - whereas grouping the instances based on their similarity without the help of class labels is known as clustering.



# Reinforcement Learning

- Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective (goal) or maximize along a particular dimension over many steps.
- This method allows machines to automatically determine the ideal behavior within a specific context in order to maximize its performance.
- Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.
- For example, maximize the points won in a game over many moves.



# Semi-Supervised Learning

- Problems where you have a large amount of input data ( $X$ ) and only some of the data is labeled ( $Y$ ) are called semi-supervised learning problems.
- Many real world machine learning problems fall into this area.
- This is because it can be expensive or time-consuming to label data as it may require access to domain experts, whereas, unlabeled data is cheap and easy to collect and store.
- You can also use supervised learning techniques to make best guess predictions for the unlabeled data, feed that data back into the supervised learning algorithm as training data and use the model to make predictions on new unseen data.

# FUNDAMENTALS OF LEARNING

Dr. Srikanth Allamsetty

# Least Square Method

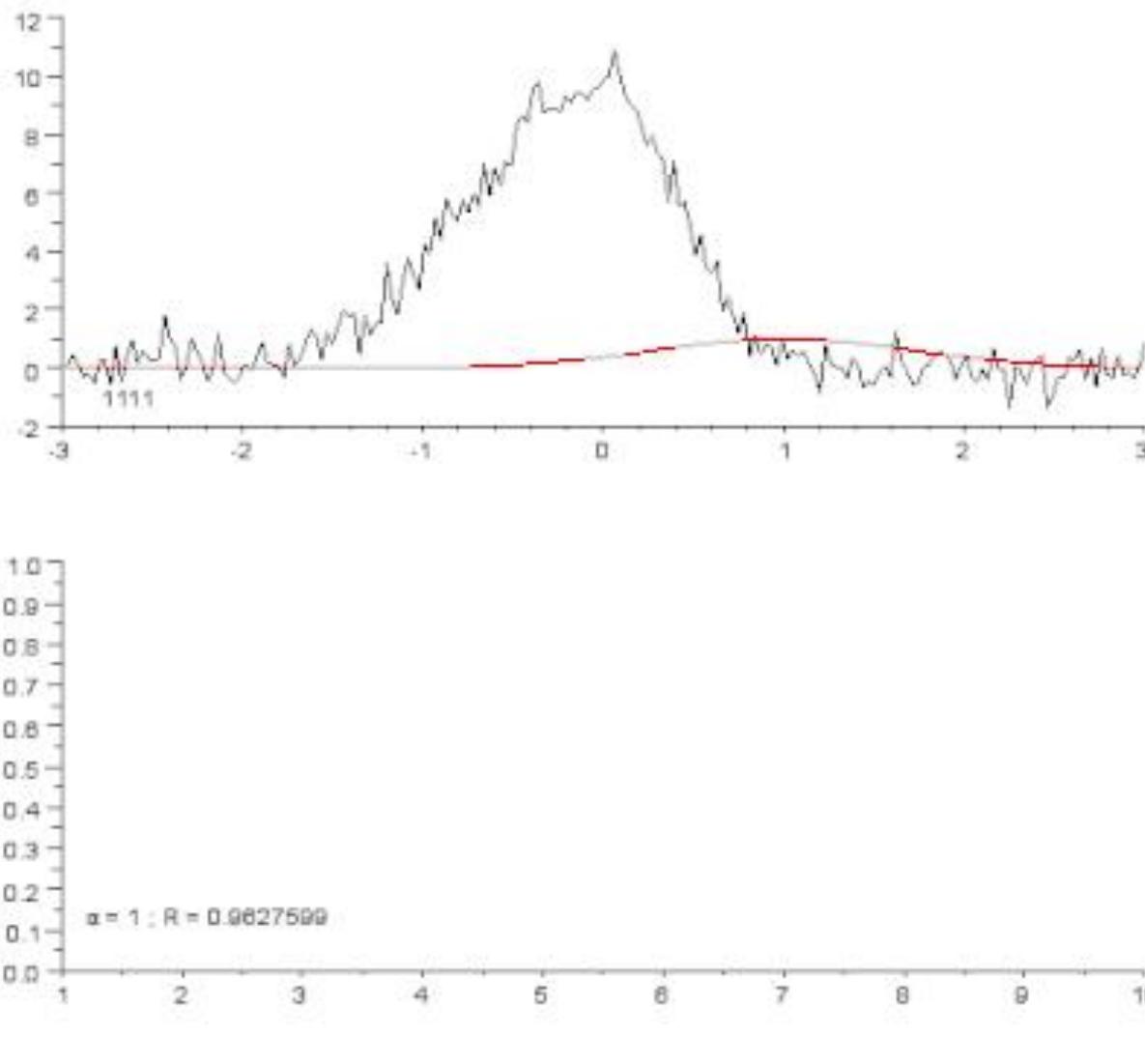
# What is Least Square Method

- The least-squares method is a statistical method that is practised to find a regression line or a best-fit line for the given pattern.
- The method of least squares is used in regression.
- In regression analysis, this method is said to be a standard approach for the approximation of sets of equations having more equations than the number of unknowns (overdetermined systems).
- It is used to approximate the solution by minimizing the sum of the squares of the residuals made in the results of each individual equation.
  - Residual: the difference between an observed value and the fitted value provided by a model
- The method of least squares assumes that the best fit curve of a given type is the curve that has the minimal sum of deviations, i.e., least square error from a given set of data.

# Curve fitting and **Least Square Method**

- Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints.
- Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing, in which a "smooth" function is constructed that **approximately** fits the data.
- The least squares method is one way to compare the deviations.

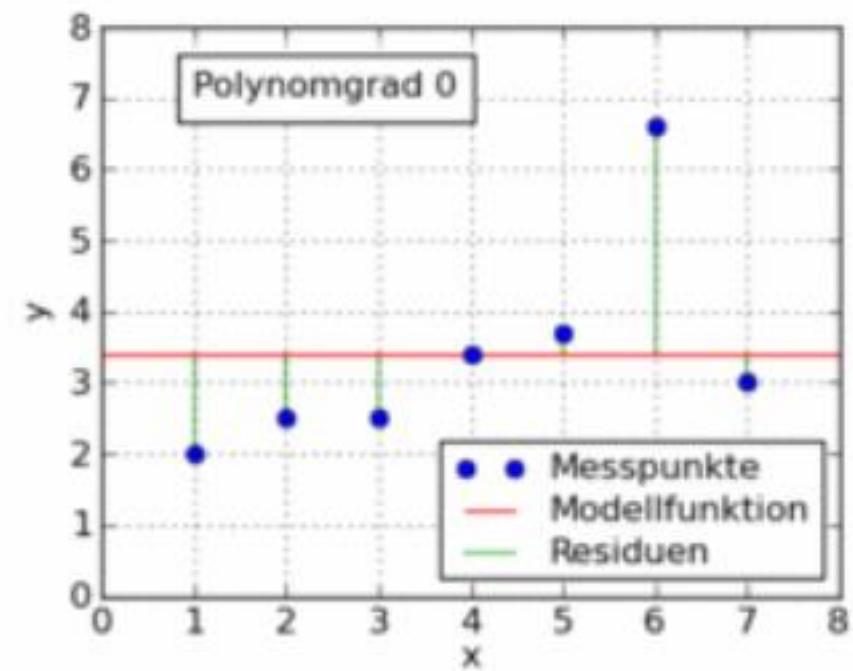
# Curve fitting and Least Square Method



Fitting of a noisy curve by an asymmetrical peak model, with an iterative process (Gauss–Newton algorithm with variable damping factor  $\alpha$ ).

Top: raw data and model.

Bottom: evolution of the normalised sum of the squares of the errors.



# Best fitting and Least Square Method

- Suppose that the data points are  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  where  $x$  is the independent variable and  $y$  is the dependent variable. The fitting curve  $f(x)$  has the deviation (error)  $e_i$  from each data point, as follows:

$$e_1 = y_1 - f(x_1),$$

$$e_2 = y_2 - f(x_2),$$

⋮

$$e_n = y_n - f(x_n)$$

- According to the method of least squares, the best fitting curve has the property that  $\sum_1^n e_i^2 = \sum_1^n [y_i - f(x_i)]^2$  is minimum.

# Least square line $y=a+bx$ for the given data

- The method of least squares can be applied to determine the estimates of 'a' and 'b' in the simple linear regression equation using the given data by minimizing

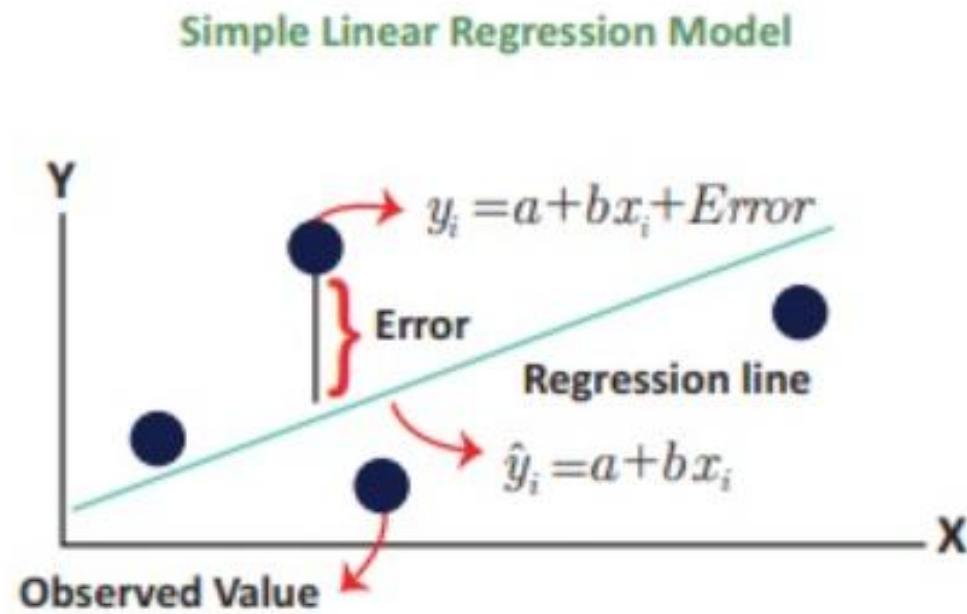
$$E(a,b) = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

i.e.,  $E(a,b) = \sum_{i=1}^n (y_i - a - bx_i)^2$ .

- Differentiation of  $E(a,b)$  with respect to 'a' and 'b' and equating them to zero constitute a set of two equations as described below:

$$\frac{\partial E(a,b)}{\partial a} = -2 \sum_{i=1}^n (y_i - a - bx_i) = 0$$

$$\frac{\partial E(a,b)}{\partial b} = -2 \sum_{i=1}^n x_i(y_i - a - bx_i) = 0$$



# Least square line $y=a+bx$ for the given data

$$\frac{\partial E(a,b)}{\partial a} = -2 \sum_{i=1}^n (y_i - a - bx_i) = 0 \rightarrow na + b \sum_{i=1}^n x_i = \sum_{i=1}^n y_i \rightarrow na + b \sum x = \sum y$$

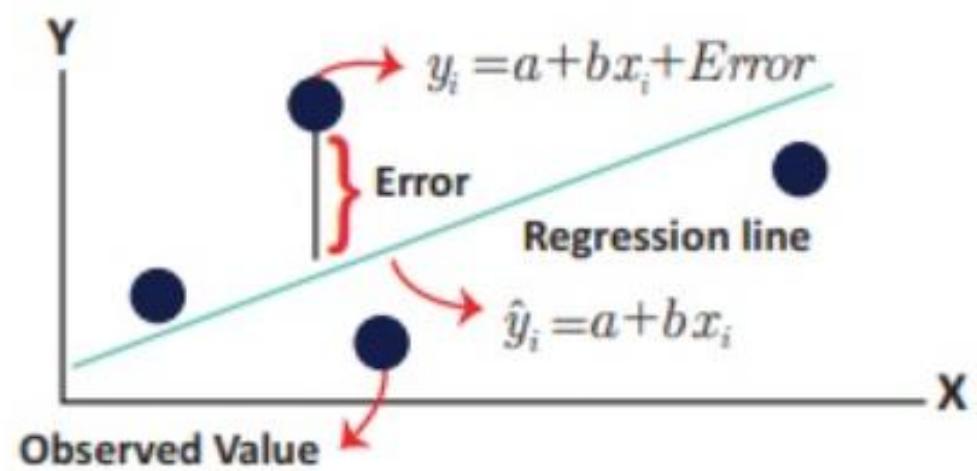
$$\frac{\partial E(a,b)}{\partial b} = -2 \sum_{i=1}^n x_i(y_i - a - bx_i) = 0 \rightarrow a \sum_{i=1}^n x_i + b \sum_{i=1}^n x_i^2 = \sum_{i=1}^n x_i y_i \rightarrow a \sum x + b \sum x^2 = \sum xy$$

- These equations are popularly known as normal equations. Solving these equations for 'a' and 'b' yield the estimates  $\hat{a}$  and  $\hat{b}$ .

$$\hat{a} = \frac{1}{n} \sum_{i=1}^n y_i - \hat{b} \frac{1}{n} \sum_{i=1}^n x_i$$

$$\hat{b} = \frac{\frac{1}{n} \sum_{i=1}^n x_i y_i - (\bar{x} \times \bar{y})}{\frac{1}{n} \sum_{i=1}^n x_i^2 - \bar{x}^2}$$

Simple Linear Regression Model



# Example 1

- Find the least square line  $y=a+bx$  for the data:

x	-2	-1	0	1	2
y	1	2	3	3	4

**Solution:**

x	y	$x^2$	$xy$
-2	1	4	-2
-1	2	1	-2
0	3	0	0
1	3	1	3
2	4	4	8
$\sum x = 0$		$\sum y = 13$	$\sum x^2 = 10$
			$\sum xy = 7$

The normal equations are

$$na + b \sum x = \sum y$$

$$a \sum x + b \sum x^2 = \sum xy$$

# Example 1

The normal equations are

$$na + b \sum x = \sum y$$

$$a \sum x + b \sum x^2 = \sum xy$$

Putting the values of  $n$ ,  $\sum x$ ,  $\sum y$ ,  $\sum x^2$ ,  $\sum xy$  in the above equation, we get

$$5a + b \cdot 0 = 13 \Rightarrow 5a = 13$$

$$a \cdot 0 + b(10) = 7 \Rightarrow 10b = 7$$

From Eqs. (12.3) and (12.4) we get

$$a = \frac{13}{5} = 2.6, \quad b = \frac{7}{10} = 0.7$$

The required of least square line is

$$y = 2.6 + (0.7)x$$

## Example 2

- Fit a straight line trend by the method of least square from the following data and find the trend values.

Year	1958	1959	1960	1961	1962
Sales (in lakhs of units)	65	95	80	115	105

- Solution:

Year	Sales	x	$x^2$	$xy$
1958	65	-2	4	-130
1959	95	-1	1	-95
1960	80	0	0	0
1961	115	1	1	115
1962	105	2	4	210
Total	$\sum y = 460$	$\sum x = 0$	$\sum x^2 = 10$	$\sum xy = 100$

## Example 2

- Solution:

$$a = \frac{\sum y}{n} \Rightarrow a = \frac{460}{5} = 92$$

$$b = \frac{\sum xy}{\sum x^2} = \frac{100}{10} = 10$$

∴ The equation of the straight line trend is

$$y_c = a + bx \Rightarrow y_c = 92 + 10x$$

Year	Sales	Trend value $y_c$
1958	65	72
1959	95	82
1960	80	92
1961	115	102
1962	105	112

# Example 3

- Fit a straight line trend by the method of least square from the following data and find the trend values.

Year	1950	1951	1952	1953	1954	1955	1956	1957
Value	346	411	392	512	626	640	611	796

Year	x	y	$x^2$	xy	Trend: $y=541.75+(59.60)x$
1950	-3.5	346	12.25	-1211.00	333.15
1951	-2.5	411	6.25	-1027.50	392.75
1952	-1.5	392	2.25	-588.00	452.35
1953	-0.5	512	0.25	-256.00	511.95
1954	0.5	626	0.25	313.00	571.95
1955	1.5	640	2.25	960.00	631.15
1956	2.5	611	6.25	1527.50	690.75
1957	3.5	796	12.25	2786.00	750.35
Total	0	4334	42.00	2504.00	

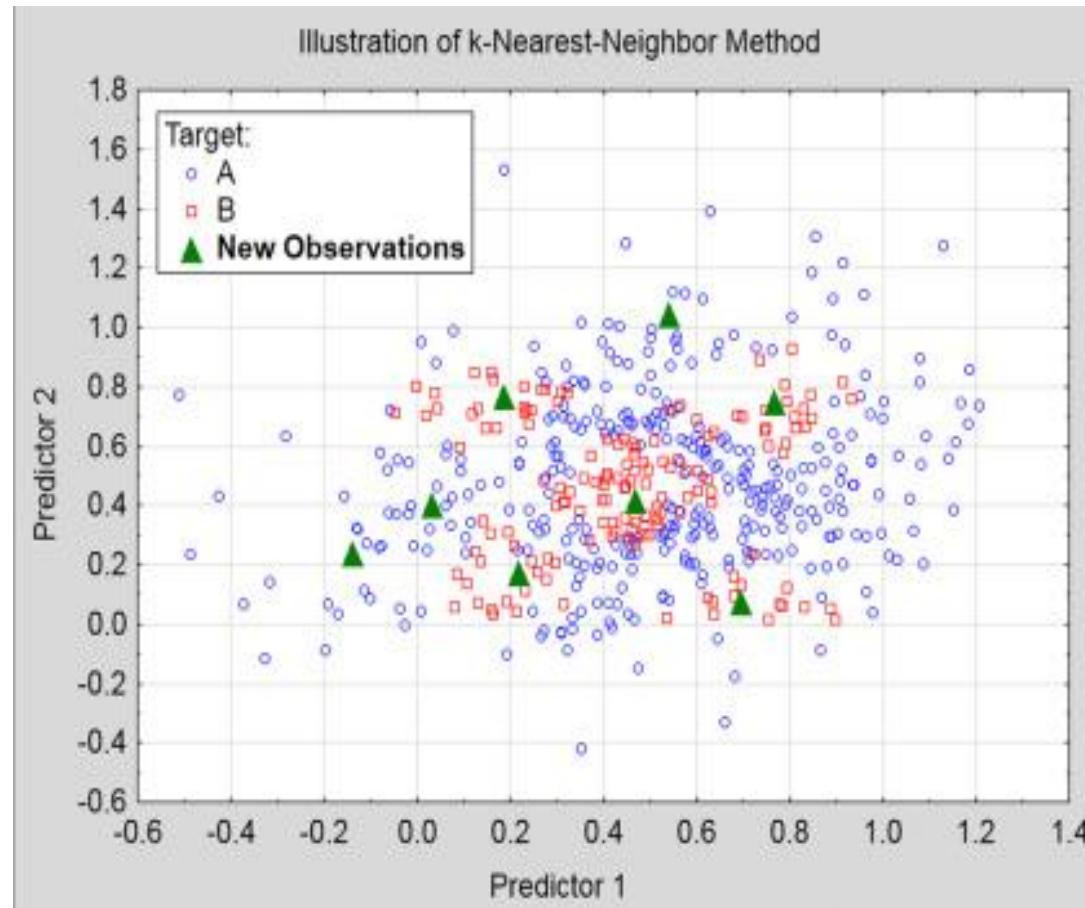
# Nearest Neighbor Method

# What is Nearest Neighbor Method

- "Nearest Neighbour" is merely "k-Nearest Neighbours" with k=1.
- The k-nearest neighbor (KNN) method is a good example of a “general approximator” that is entirely based on patterns in the data, without any specific “statistical model” that must be estimated.
- In fact, KNN methods do not rely on any “models” at all but instead simply use the existing data (or a sample of exemplars from the existing data) to “predict” new observations.
- Specifically, when predicting a new observation, the algorithm finds the most similar observations among the exemplars with respect to the available predictors and then makes the prediction that the new observation will have the same outcome (same predicted y, or predicted classification).

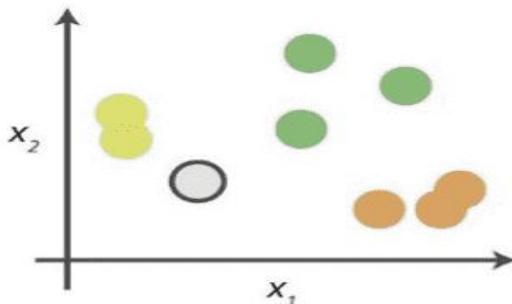
# What is Nearest Neighbor Method

- For example, consider the classification problem illustrated in this figure.
- In this case there are two predictors—Predictor 1 and Predictor 2—and a number of exemplars or individual square (red) and round (blue) points that belong to category A or B, respectively.
- Now consider the problem of predicting new observations: triangles (green) to belong either to category A or B.



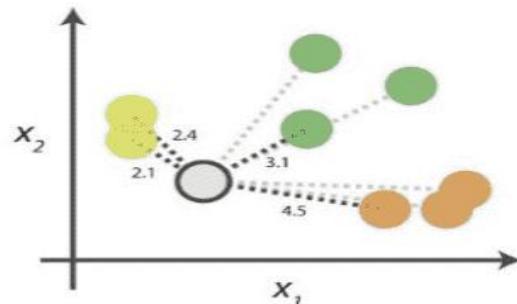
# Steps in KNN Classification

## 0. Look at the data



Say you want to classify the grey point into a class. Here, there are three potential classes - lime green, green and orange.

## 1. Calculate distances



Start by calculating the distances between the grey point and all other points.

## 2. Find neighbours

### Point Distance

● ... ●	2.1	→ 1st NN
● ... ●	2.4	→ 2nd NN
● ... ●	3.1	→ 3rd NN
● ... ●	4.5	→ 4th NN

Next, find the nearest neighbours by ranking points by increasing distance. The nearest neighbours (NNs) of the grey point are the ones closest in dataspace.

## 3. Vote on labels

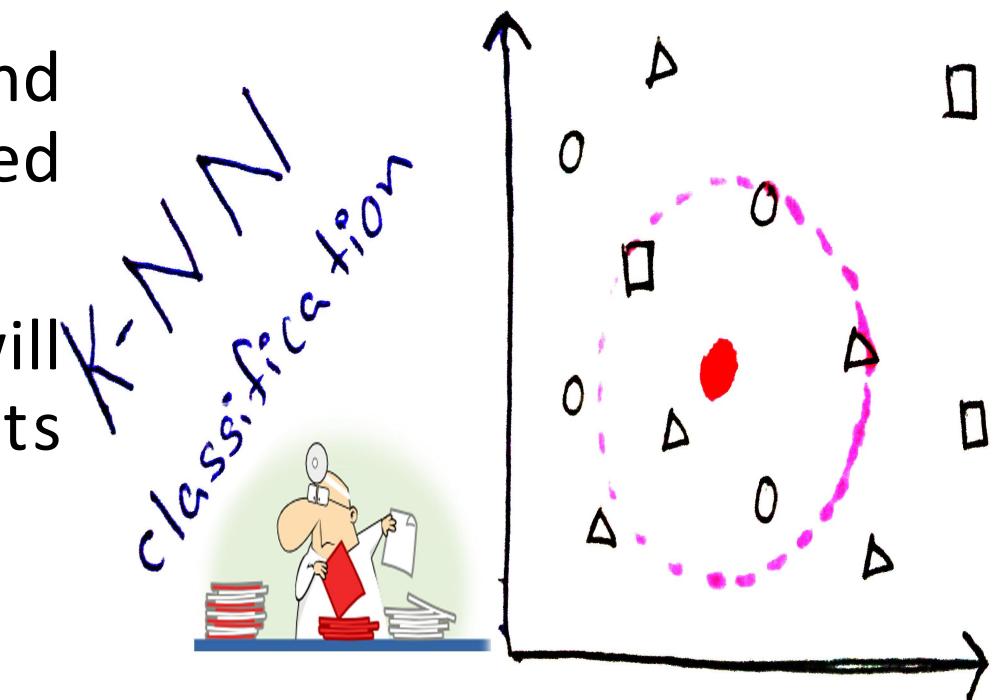
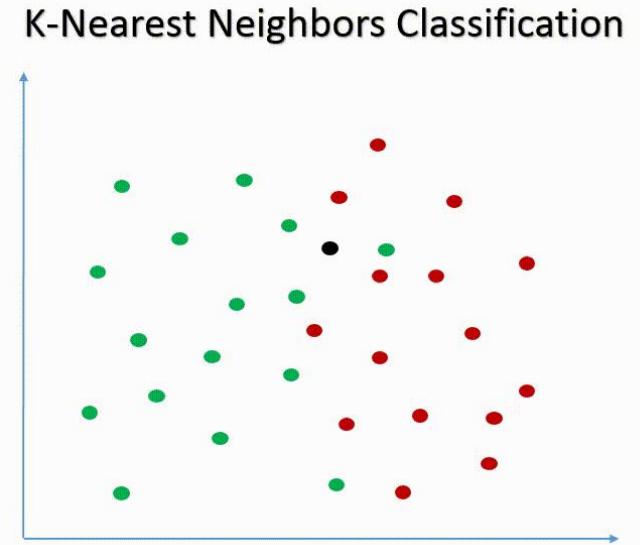
Class	# of votes
●	2
●	1
●	1

Class ● wins the vote!  
Point ● is therefore predicted to be of class ●.

Vote on the predicted class labels based on the classes of the  $k$  nearest neighbours. Here, the labels were predicted based on the  $k=3$  nearest neighbours.

# Application of KNN Classification

- K is the most important hyperparameter in the KNN algorithm.
- KNN makes predictions based on the K nearest instances in the training data.
- KNN can be used for both classification and regression; both supervised and unsupervised learning (K-Means clustering).
- As it has all the training data in memory, it will simply compare the new data with k of its nearest data points.

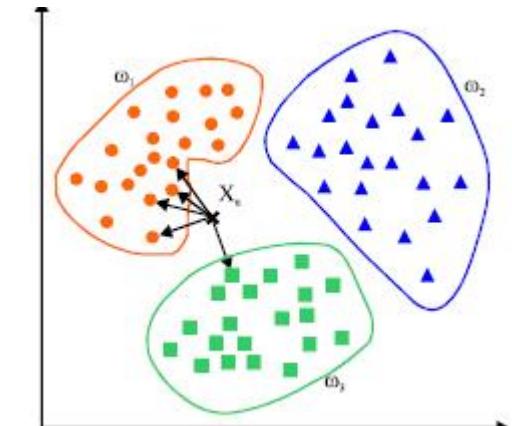
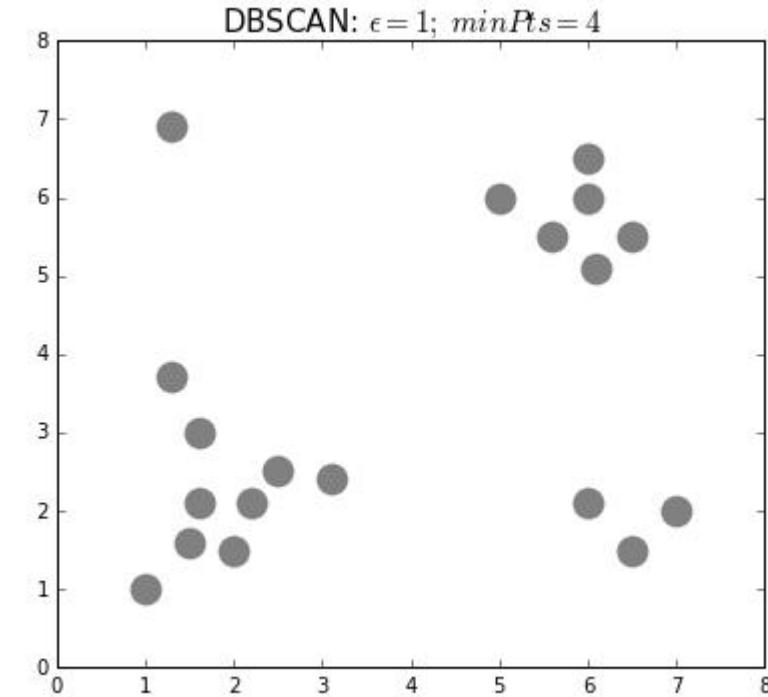


# Application of KNN Classification

- The k-NN algorithm is a classification technique that does not make assumptions about the model between the class membership ( $y$ ) and the features ( $x_1, x_2, \dots, x_n$ ).
- This is a non-parametric technique as it does not involve estimation of parameters in an assumed model.
- Rather, this method pulls out information from **similarities** existing amongst the values of the patterns in the dataset.
- This model-free algorithm finds similar past patterns or instances from the training set with the use of a suitable distance measure and interpolates from them the correct output.
- In machine learning, these techniques are also known as instance-based or memory-based learning algorithms.

# Differences between K-means and K-NN?

- **K-means** is a clustering algorithm that tries to partition a set of points into K sets (clusters) such that the points in each cluster tend to be near each other.
  - It is unsupervised because the points have no external classification.
- **K-nearest neighbors** is a classification (or regression) algorithm that in order to determine the classification of a point, combines the classification of the K nearest points.
  - It is supervised because you are trying to classify a point based on the known classification of other points.



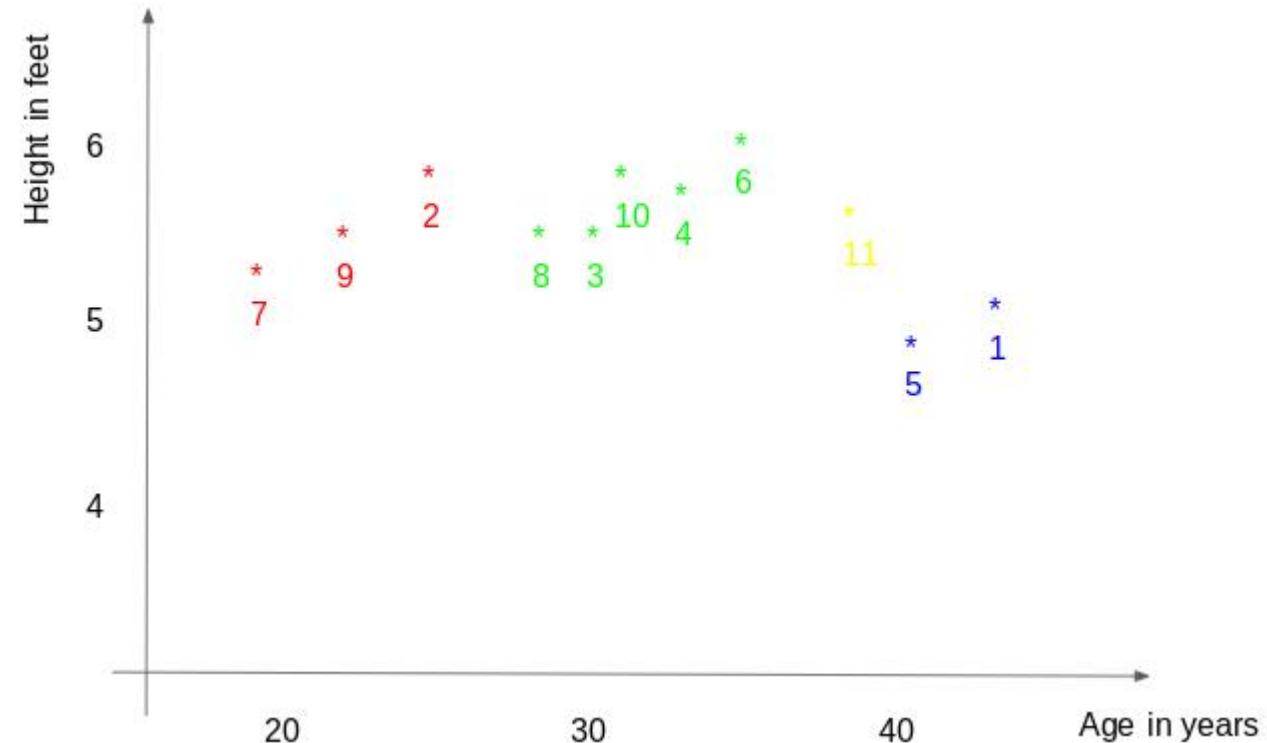
# Example

- Predict the weight of the person with ID 11 based on his height and age.

ID	Height	Age	Weight
1	5	45	77
2	5.11	26	47
3	5.6	30	55
4	5.9	34	59
5	4.8	40	72
6	5.8	36	60
7	5.3	19	40
8	5.8	28	60
9	5.5	23	45
10	5.6	32	58
11	5.5	38	?

Solution:

below is the plot of height versus age from the given table



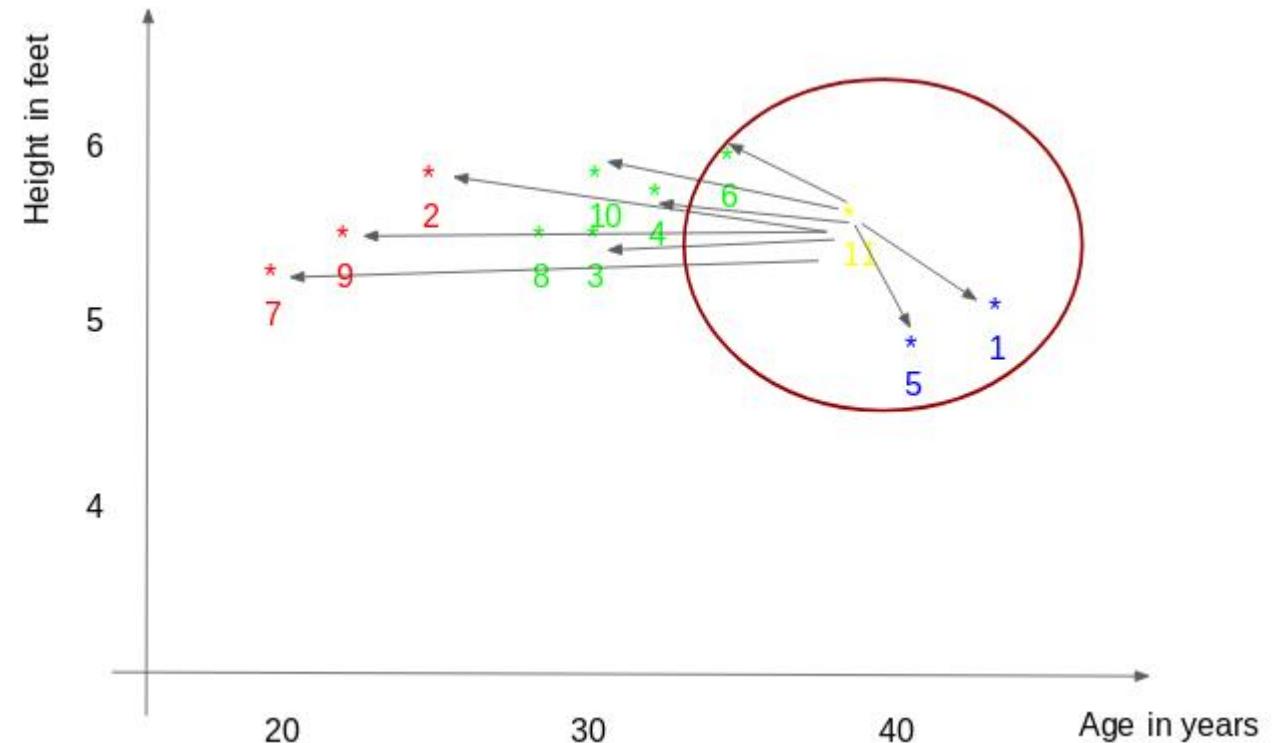
# Example

- Predict the weight of the person with ID 11 based on his height and age.

ID	Height	Age	Weight
1	5	45	77
2	5.11	26	47
3	5.6	30	55
4	5.9	34	59
5	4.8	40	72
6	5.8	36	60
7	5.3	19	40
8	5.8	28	60
9	5.5	23	45
10	5.6	32	58
11	5.5	38	?

## Solution:

for a value  $k = 3$ , the closest points are ID1, ID5 and ID6.



Here, we get weight of ID11 =  $(77+72+60)/3 = 69.66$  kg.

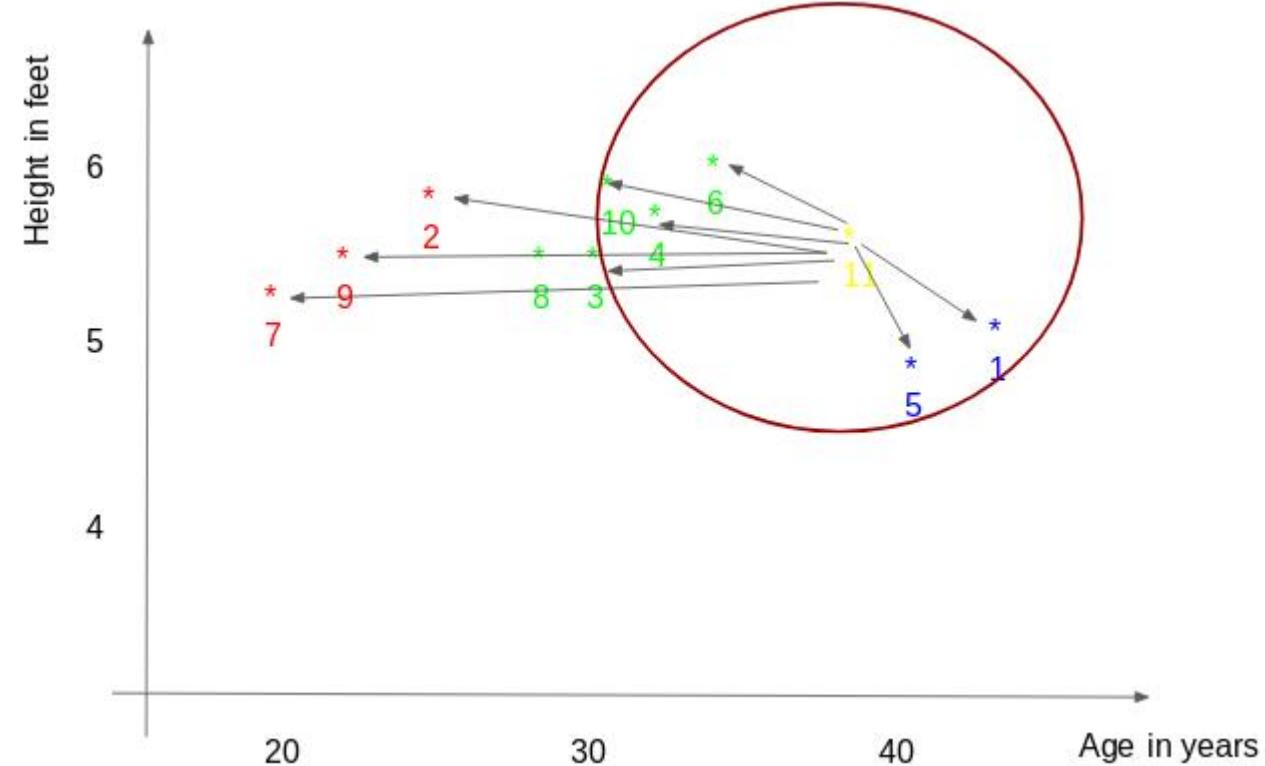
# Example

- Predict the weight of the person with ID 11 based on his height and age.

ID	Height	Age	Weight
1	5	45	77
2	5.11	26	47
3	5.6	30	55
4	5.9	34	59
5	4.8	40	72
6	5.8	36	60
7	5.3	19	40
8	5.8	28	60
9	5.5	23	45
10	5.6	32	58
11	5.5	38	?

Solution:

for a value  $k = 5$ , the closest points are ID1, ID4, ID5, ID6, ID10.



Here, we get weight of ID11 =  $(77+59+72+60+58)/5 = 65.2$  kg.

# Constraints

- Based on the k value, the final result tends to change; Then how can we figure out the optimum value of k?
  - through finding training and validating/testing errors
- KNN works best when the dataset is not too large.
- The main reason being that the whole dataset is stored in memory. Therefore, when the number of samples increases, then KNN can become inefficient/slow.
- Another thing to notice is that KNN models are not really ‘trained’; The dataset is just memorized.
- So, with large datasets, the main problem begins when finding the neighbors for a new data point, that is the testing phase.
- The most serious shortcoming of nearest-neighbor methods is that they are very sensitive to the presence of irrelevant parameters.

# Distance Based Learning

# What is **Distance Based Learning**

- As the name implies, distance-based learning/models work on the concept of distance.
- A few of the More popular machine learning algorithms that use distance measures at their core is
  - **K-Nearest Neighbors (KNN)**
  - **Learning Vector Quantization (LVQ)**: A type of neural network that lets you choose how many training instances to hang onto.
  - **Self-Organizing Map (SOM)**: A neural network-based dimensionality reduction algorithm.
  - **K-Means Clustering**
  - **kernel-based methods such as Support Vector Machine (SVM) algorithm**

# Distance as measure of similarity

- Characterizing the similarity of the patterns in state space can be done through some form of metric (distance) measure: distance between two vectors is a measure of similarity between two corresponding patterns.
- Let us use  $d_{il}$  to depict a distance metric or dissimilarity measure, between patterns  $i$  and  $l$ . For pattern  $i$ , we have the vector of  $n$  measurements  $(x_1^{(i)}, x_2^{(i)}, \dots, x_n^{(i)})$ , while for pattern  $l$ , we have the vector of  $n$  measurements  $(x_1^{(l)}, x_2^{(l)}, \dots, x_n^{(l)})$ .

# Properties of Distances

- Distances can be defined in multiple ways, but in general the following properties are required:

*Nonnegativity:*  $d_{il} \geq 0$

*Self-proximity:*  $d_{ii} = 0$

*Symmetry:*  $d_{il} = d_{li}$

*Triangle Inequality:*  $d_{il} \leq d_{ik} + d_{kl}$

# Types of Distances

- Euclidean distance
- Statistical distance
- Manhattan distance
- Minkowski metric
- Hamming distance

# Euclidean distance

**Euclidean distance:** The most popular distance measure is the *Euclidean distance*,  $d_{il}$ , which between two patterns  $i$  and  $l$ , is defined by,

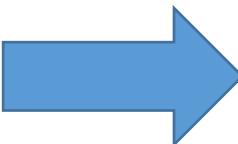
$$d_{il} = \sqrt{(x_1^{(i)} - x_1^{(l)})^2 + (x_2^{(i)} - x_2^{(l)})^2 + \dots + (x_n^{(i)} - x_n^{(l)})^2}$$

Despite Euclidean distance being the most commonly used similarity measure, there are three primary characteristics that need to be taken into consideration.

- (i) It depends highly on scale, and variables that possess bigger scales impact the total distance to a great extent. Thus, we first *normalize* continuous measurements and only then calculate the Euclidean distance. This transforms all measurements to the same scale.
- (ii) Euclidean distance entirely ignores the relationship between measurements. If there is a strong correlation between measurements, a *statistical distance* measure seems to be a better candidate.
- (iii) Euclidean distance is sensitive to outliers, and if the data comprises outliers, a preferred choice is the use of robust distances like *Manhattan distance*.

# Normalized data

	x1	x2	x3		x1n	x2n	x3n
1	53	5	4646		1	0.833333333	0.388461538
2	53	5	11960		1	0.833333333	1
3	50	6	534		0.980392157	0.000820345	0.048090778
4	34	4	6688		0.666666667	0.000350079	0.602305476
5	45	4	6023	x <sub>ij</sub> =x <sub>ij</sub> /max(x <sub>i</sub> )	0.882352941	0.000350079	0.542417147
6	29	6	9014	(normalized)	0.568627451	0.000525118	0.811779539
7	33	6	6251		0.647058824	0.000525118	0.562950288
8	31	6	1093		0.607843137	0.000525118	0.098432997
9	21	5	6259		0.411764706	0.000437598	0.563670749
10	41	4	8715		0.803921569	0.000350079	0.784852305
11	51	5	10877		1	0.000437598	0.979556916
12	21	6	11104		0.6	0.000525118	1
13	35	6	10238		1	0.000525118	1
14	20	6	4985		1	0.000525118	1



d12	7314	0.611538462
d23	11426.00044	1.264750375
d34	6154.021124	0.636849931
d45	665.0909712	0.223846333
d56	2991.043463	0.413497052

# Euclidean distance

- Example:
- $\text{row1} = [10, 20, 15, 10, 5]$
- $\text{row2} = [12, 24, 18, 8, 7]$
- $\text{euclidean\_distance}(\text{row1}, \text{row2}) = 6.082762530298219$

# Statistical distance

**Statistical distance:** This metric takes into consideration the correlation between measurements. With this metric, measurements extremely correlated with other measurements do not contribute as much as the uncorrelated or less correlated. The *statistical distance*, also referred to as the *Mahalanobis distance*, between patterns  $i$  and  $l$  is defined as,

$$d_{il} = \sqrt{(\mathbf{x}^{(i)} - \mathbf{x}^{(l)})^T \Sigma^{-1} (\mathbf{x}^{(i)} - \mathbf{x}^{(l)})}$$

where  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(l)}$  are  $n$ -dimensional vectors of measurement values of patterns  $i$  and  $l$ , respectively, and  $\Sigma$  is the covariance matrix of these vectors.

# Manhattan distance

**Manhattan distance:** This distance looks at the absolute differences rather than squared differences, and is defined by,

$$d_{il} = \sum_{j=1}^n |x_j^{(i)} - x_j^{(l)}|$$

- Example:
- `row1 = [10, 20, 15, 10, 5]`
- `row2 = [12, 24, 18, 8, 7]`
- `manhattan_distance(row1, row2) = 13`

# Minkowski metric

**Minkowski metric:** One general class of metrics for  $n$ -dimensional patterns is the Minkowski metric (also referred to as the  $L_p$  norm):

$$L_p(\mathbf{x}^{(i)}, \mathbf{x}^{(l)}) = \left( \sum_{j=1}^n |x_j^{(i)} - x_j^{(l)}|^p \right)^{1/p}$$

where  $p \geq 1$  is a selectable parameter. Setting  $p = 2$  gives the familiar Euclidean distance ( $L_2$  norm) and setting  $p = 1$  gives the Manhattan distance ( $L_1$  norm). With respect to Eqns given before

$$L_p(\mathbf{x}^{(i)}, \mathbf{x}^{(l)}) = \|\mathbf{x}^{(i)} - \mathbf{x}^{(l)}\|_p = \left( \sum_{j=1}^n |x_j^{(i)} - x_j^{(l)}|^p \right)^{1/p}$$

(Minkowski norm)

$$\|\mathbf{x}^{(i)} - \mathbf{x}^{(l)}\|_2 = [(x_1^{(i)} - x_1^{(l)})^2 + (x_2^{(i)} - x_2^{(l)})^2 + \dots + (x_n^{(i)} - x_n^{(l)})^2]^{1/2}$$

(Euclidean norm)

$$\|\mathbf{x}^{(i)} - \mathbf{x}^{(l)}\|_1 = \sum_{j=1}^n |x_j^{(i)} - x_j^{(l)}|$$

(Manhattan norm)

# Hamming distance

- Hamming distance calculates the distance between two binary vectors, also referred to as binary strings or bitstrings
- **For a One-hot encoded string**

Hamming Distance = sum for i to N abs(v1[i] – v2[i])

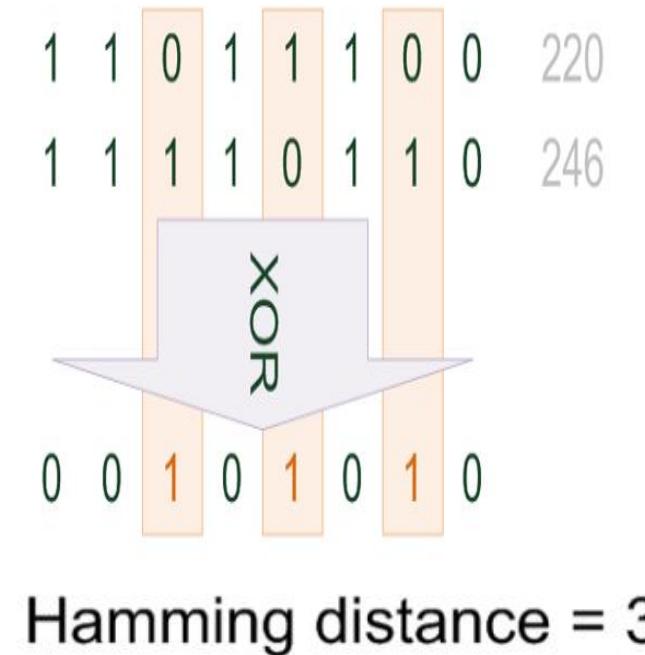
- **For bitstrings that may have many 1 bits**

Hamming Distance = (sum for i to N abs(v1[i] – v2[i]))/N

example: row1 = [0,0,0,0,0,1]

row2 = [0,0,0,0,1,0]

hamming\_distance(row1, row2)=0.3333



# Scope for Research (just an example)

<https://www.sciencedirect.com/science/article/pii/S1568494620307936>



Applied Soft Computing

Volume 98, January 2021, 106855



## Standardized Variable Distances: A distance-based machine learning method

Abdullah Elen <sup>a</sup>  Emre Avuçlu <sup>b</sup> 

Show more 

 Add to Mendeley  Share  Cite

<https://doi.org/10.1016/j.asoc.2020.106855>

[Get rights and content](#)

### Highlights

- The Standardized Variable Distances (SVD) is a novel machine learning algorithm.
- The SVD is an improved version of the Minimum Distance Classifier (MDC) algorithm.

# LINEAR MODEL - LINEAR REGRESSION

Dr. Srikanth Allamsetty

# Formulation & Mathematical Foundation of Regression Problem

# What is Regression

- **Regression** – predict value of *response variable* from attribute **variables**.
- Variables – continuous numeric values
- **Regression analysis** – a set of **statistical processes** for estimating the relationships between a dependent variable and one or more independent variables.
  - Dependent variables are often called the 'predictand', 'outcome' or 'response' variable;
  - Independent variables are often called 'predictors', 'covariates', 'explanatory variables' or 'features'.
  - Regression analysis is a way of mathematically sorting out which of those variables does indeed have an impact.
  - Used for modeling the future relationship between the variables.
- **Statistical process** – a **science** of collecting, exploring, organizing, analyzing, interpreting data and exploring patterns and trends to answer questions and make decisions (Broad area).

# Basics of Regression Models

- Regression models predict a value of the  $Y$  variable given known values of the  $X$  variables.
- Prediction within the range of values in the dataset used for model-fitting is known as interpolation.
- Prediction outside this range of the data is known as extrapolation.
- First, a model to estimate the outcome need to be fixed.
- Then the parameters of that model need to be estimated using any chosen method (e.g., least squares).

# Formulation of Regression Models

- Regression models involve the following components:
- The unknown parameters, often denoted as  $\beta$  or  $\omega$  or  $w$ .
- The independent variables, which are observed in data and are often denoted as a vector  $X_i$  (where  $i$  denotes a row of data).
- The dependent variable, which are observed in data and often denoted using the scalar  $Y_i$ .
- The error terms, which are not directly observed in data and are often denoted using the scalar  $e_i$ .

# Formulation of **Regression Models**

- Most regression models propose that  $Y_i$  is a function of  $X_i$  and  $\beta$ , with  $e_i$  representing an additive error term that may stand in for a random statistical noise.

$$Y_i = f(X_i, \beta) + e_i$$

- Our objective is to estimate the function  $f(X_i, \beta)$  that most closely fits the data.
- To carry out regression analysis, the form of the function  $f$  must be specified.
- Sometimes the form of this function is based on knowledge about the relationship between  $Y_i$  and  $X_i$ .
- If no such knowledge is available, a flexible or convenient form for  $f$  is chosen.

# Formulation of Regression Models

- You may start with a simple univariate linear regression:

$$f(X_i, \beta) = \beta_0 + \beta_1 X_i$$

- It indicates that you believe that a reasonable approximation for  $Y_i$  is:

$$Y_i = \beta_0 + \beta_1 X_i + e_i$$

- Now the next objective is to estimate the parameters  $\beta$ 
  - may be using least squares method
  - may go with other alternatives such as least absolute deviations, Least trimmed squares, quantile regression estimator, Theil–Sen estimator, M-estimation (maximum likelihood type) or S-estimation (scale).

# Formulation of Regression Models

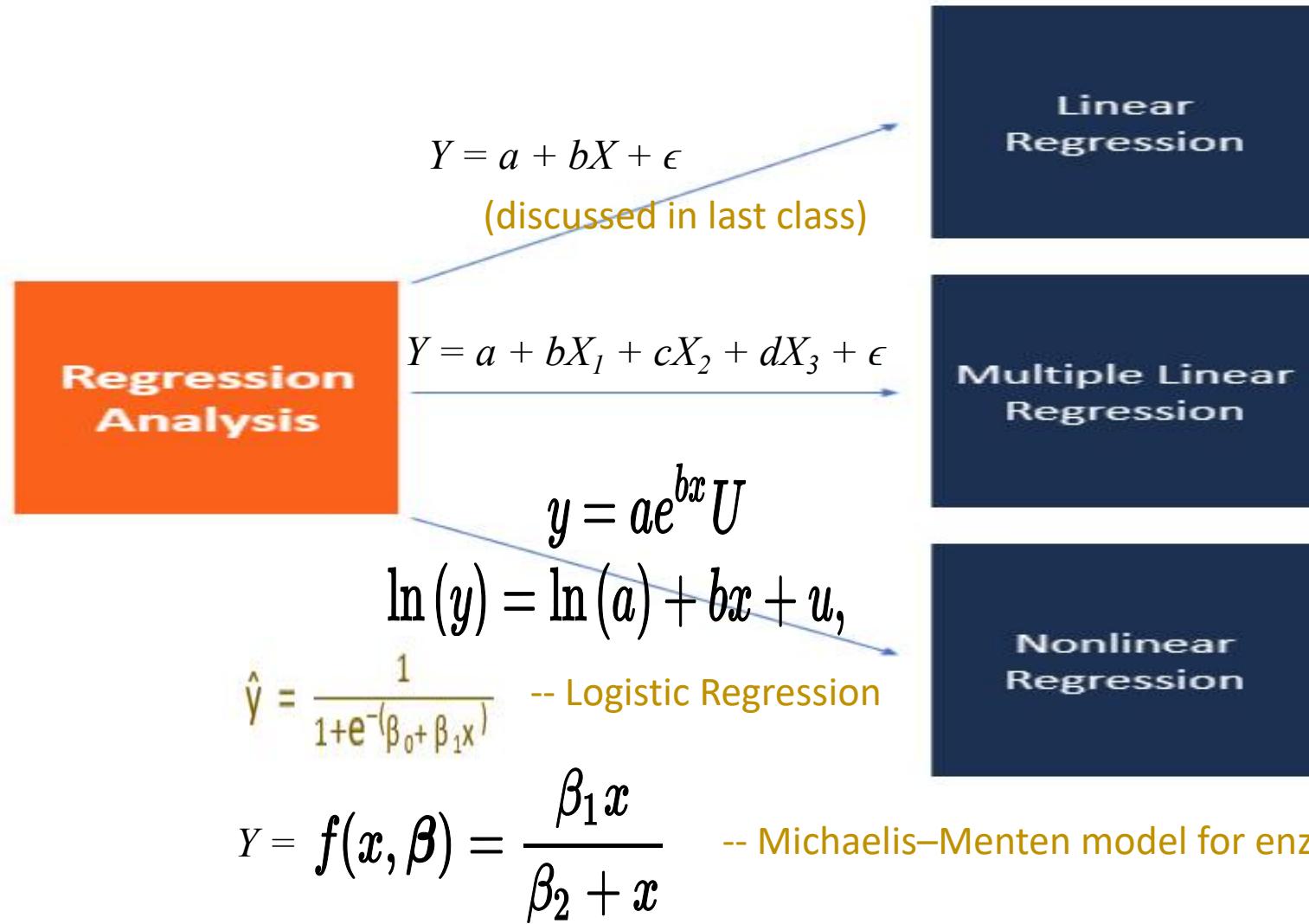
- Find the value of  $\beta$  that minimizes the sum of squared errors

$$\sum_i (Y_i - f(X_i, \beta))^2$$

- A given regression method will ultimately provide an estimate of  $\beta$ , usually denoted  $\hat{\beta}$ .
- Using this estimate, you can then find the fitted value for prediction or to assess the accuracy of the model in explaining the data.

$$\hat{Y}_i = f(X_i, \hat{\beta})$$

# Variants in Regression Models



- The most common models are simple linear and multiple linear.
- Nonlinear regression analysis is commonly used for more complicated data sets in which the dependent and independent variables show a nonlinear relationship.

# Multiple Linear Regression

	$x$		$y$		
$i \downarrow$	$j \rightarrow$				
	Size (feet <sup>2</sup> )	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
1	2104	5	1	45	460
2	1416	3	2	40	232
3	1534	3	2	30	315
4	852	2	1	36	178
N	...	...	...	...	...

Notation:

$n$  = number of features

$x^{(i)}$  = input (features) of  $i^{th}$  training example.

$x_j^{(i)}$  = value of feature  $j$  in  $i^{th}$  training example.

$N$  = number of training examples

# Multiple Linear Regression

$$\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + \dots + w_n x_n^{(i)}; i = 1, \dots, N$$

Notation:

$n$  = number of features

$x^{(i)}$  = input (features) of  $i^{th}$  training example.

$x_j^{(i)}$  = value of feature  $j$  in  $i^{th}$  training example.

$N$  = number of training examples

Use the following data set to answer the questions:

Sample	Y	X1	X2	X3
1	35	8.4	8	1
2	10	2	6.5	8.5
3	9	3.5	6.2	6.5
4	30	10.4	5	1.5
5	20	6.5	6.5	7.5
6	23	6.2	7.3	4.5
7	28	12.4	6.4	4
8	8	7	6	10
9	29	5.8	6.1	3
10	4	3	5.4	11
11	18	6	7.3	4.5
12	14	5.5	6.6	5.5
13	32	9	6.5	2.5
14	6	1.1	5.8	7
15	8	2.1	7.1	9
16	37	10	8.5	2
17	25	7	5.5	3
18	15	5	5	4.5
19	30	9.3	7.9	3
20	10	4.4	4.5	7.9

We are interested in modeling Y based on X1, X2 and X3.

Y= length of larvae in the water

X1: depth of the water

X2: amount of dissolved oxygen

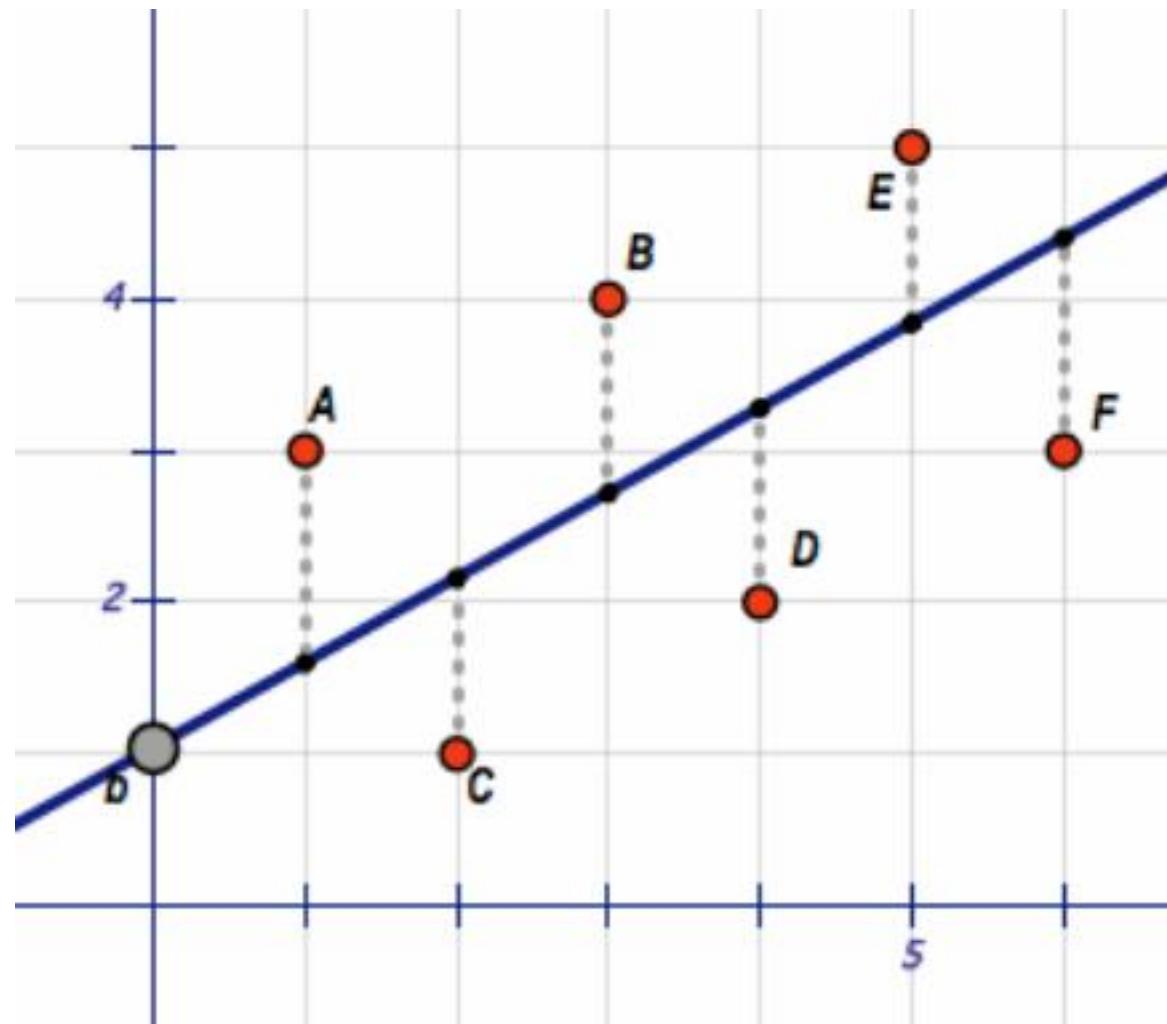
X3: brackishness of water.

# The Regression Model, & The Concepts of Least Squares

# What is Least Square Method

- The least-squares method is a statistical method that is practised to find a regression line or a best-fit line for the given pattern.
- The method of least squares is used in regression.
- In regression analysis, this method is said to be a standard approach for the approximation of sets of equations having more equations than the number of unknowns (overdetermined systems).
- It is used to approximate the solution by minimizing the sum of the squares of the residuals made in the results of each individual equation.
  - Residual: the difference between an observed value and the fitted value provided by a model
- The problem of finding a linear regressor function will be formulated as a problem of minimizing a criterion function.
- The widely-used criterion function for regression purposes is the sum-of-error-squares.

# Least Square Method with Linear Regression



# Least Square Method with Linear Regression

- In general, regression methods are used to predict the value of response (dependent) variable from attribute (independent) variables,
- Linear regressor model fits a linear function (relationship) between dependent (output) variable and independent (input) variables.

$$\hat{y}^{(i)} = w_0 + w_1 x_1^{(i)} + \dots + w_n x_n^{(i)}; i = 1, \dots, N$$

- where  $\{w_0, w_1, \dots, w_n\}$  are the parameters of the model.
- The method of linear regression is to choose the  $(n + 1)$  coefficients  $w_0, w_1, \dots, w_n$ , to minimize the residual sum of squares of these differences over all the  $N$  training instances.

# Least Square Method with Linear Regression

Residual error

$$e^{(i)} = y^{(i)} - \hat{y}^{(i)} = y^{(i)} - \sum_{j=0}^n w_j x_j^{(i)}; x_0^{(i)} = 1$$

Residual sum-of-error-squares

$$E = \sum_{i=1}^N (e^{(i)})^2 = \sum_{i=1}^N \left( y^{(i)} - \sum_{j=0}^n w_j x_j^{(i)} \right)^2; x_0^{(i)} = 1$$

- The method of linear regression is to choose the  $(n + 1)$  coefficients  $w_0, w_1, \dots, w_n$ , to minimize the residual sum of squares of these differences over all the  $N$  training instances.

# Minimal Sum-of-Error-Squares

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(N)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(N)} \\ \vdots & \vdots & & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(N)} \end{bmatrix} = [\bar{\mathbf{x}}^{(1)} \ \bar{\mathbf{x}}^{(2)} \ \dots \ \bar{\mathbf{x}}^{(N)}] \quad \mathbf{y} = [y^{(1)} \ y^{(2)} \ \dots \ y^{(N)}]^T$$
$$\bar{\mathbf{w}} = [w_0 \ w_1 \ w_2 \ \dots \ w_n]^T$$

$$\bar{\mathbf{x}} = [1 \ x_1 \ x_2 \ \dots \ x_n]^T$$

- For an optimum solution for  $\mathbf{w}$ , the following equations need to be satisfied:

$$y^{(1)} = \bar{\mathbf{w}}^T \bar{\mathbf{x}}^{(1)}$$

$$\vdots$$

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

# Minimal Sum-of-Error-Squares

Therefore,

$$[y^{(1)} \ y^{(2)} \dots y^{(N)}] = \bar{\mathbf{w}}^T [\bar{\mathbf{x}}^{(1)} \ \bar{\mathbf{x}}^{(2)} \dots \bar{\mathbf{x}}^{(N)}] = \bar{\mathbf{w}}^T \mathbf{X}$$

or

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

The vector of residual errors becomes

$$\mathbf{y} - (\bar{\mathbf{w}}^T \mathbf{X})^T$$

Hence the error function can be written as,

$$\begin{aligned} E(\bar{\mathbf{w}}) &= [\mathbf{y} - (\bar{\mathbf{w}}^T \mathbf{X})^T]^T [\mathbf{y} - (\bar{\mathbf{w}}^T \mathbf{X})^T] \\ &= \bar{\mathbf{w}}^T [\mathbf{X} \mathbf{X}^T] \bar{\mathbf{w}} - 2\bar{\mathbf{w}}^T \mathbf{X} \mathbf{y} + \mathbf{y}^T \mathbf{y} \end{aligned}$$

# Minimal Sum-of-Error-Squares

$$E(\bar{\mathbf{w}}) = \bar{\mathbf{w}}^T [\mathbf{X}\mathbf{X}^T] \bar{\mathbf{w}} - 2\bar{\mathbf{w}}^T \mathbf{X}\mathbf{y} + \mathbf{y}^T \mathbf{y}$$

- In this least-squares estimation task, the objective is to find the optimal  $\bar{\mathbf{w}}^*$  that minimizes  $E(\bar{\mathbf{w}})$ .
- The solution to this classic problem in calculus is found by setting the gradient of  $E(\bar{\mathbf{w}})$ , with respect to  $\bar{\mathbf{w}}$ , to zero.

$$\frac{\partial E(\bar{\mathbf{w}})}{\partial \bar{\mathbf{w}}} = 2(\mathbf{X}\mathbf{X}^T) \bar{\mathbf{w}} - 2\mathbf{X}\mathbf{y} = \mathbf{0}$$

This gives

$$\bar{\mathbf{w}}^* = (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y}$$

The fitted output values at the training data are

$$\hat{\mathbf{y}} = \mathbf{X}^T \bar{\mathbf{w}}^* = \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y}$$

# Minimal Sum-of-Error-Squares

$$\hat{\mathbf{y}} = \mathbf{X}^T \bar{\mathbf{w}}^* = \mathbf{X}^T (\mathbf{X}\mathbf{X}^T)^{-1} \mathbf{X}\mathbf{y}$$

- The  $(n + 1) \times N$  matrix  $\mathbf{X}^+ = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}$  is called the pseudoinverse matrix of the matrix  $\mathbf{X}^T$ . Thus, the optimal solution is

$$\bar{\mathbf{w}}^* = \mathbf{X}^+\mathbf{y}$$

# Unique solution?

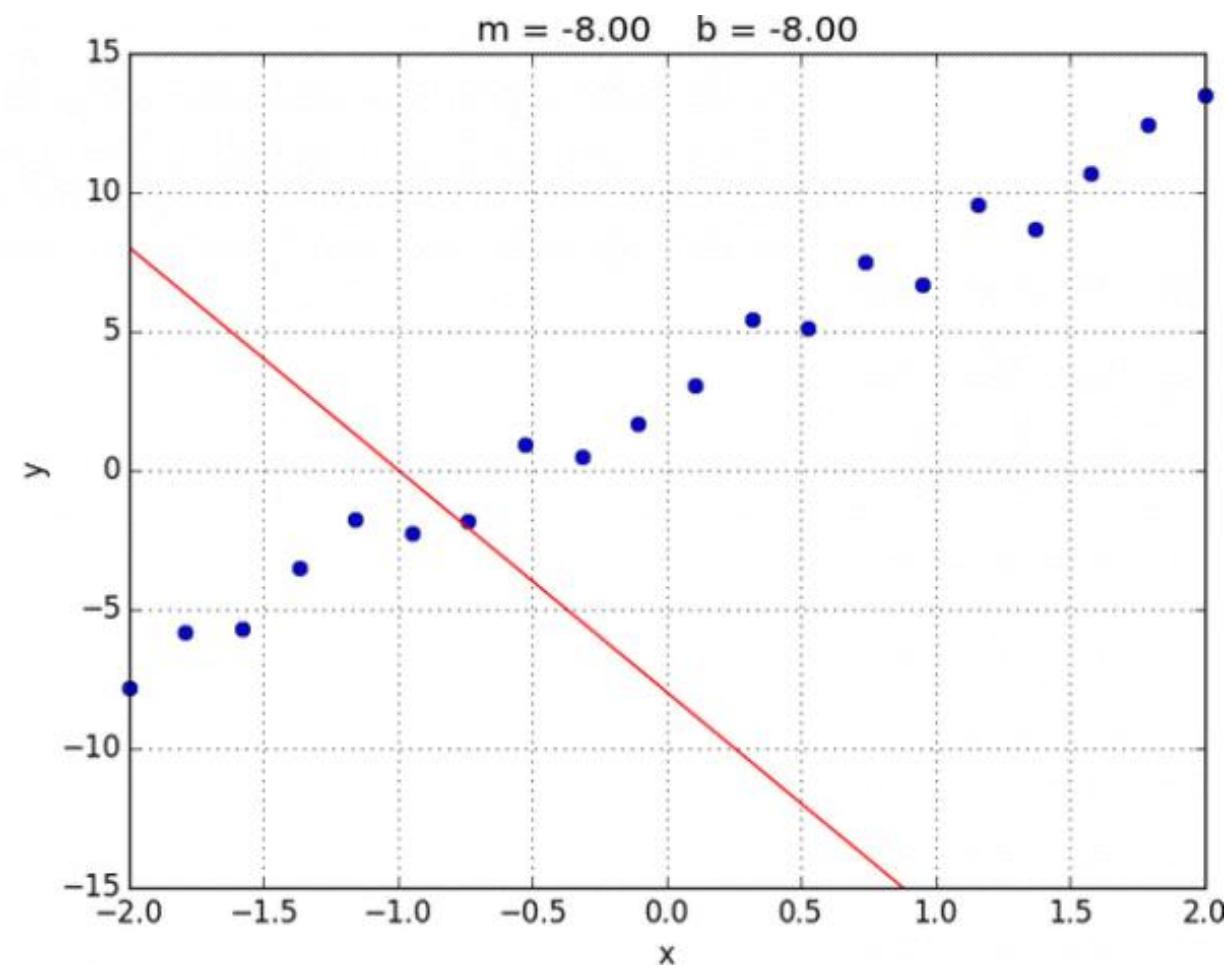
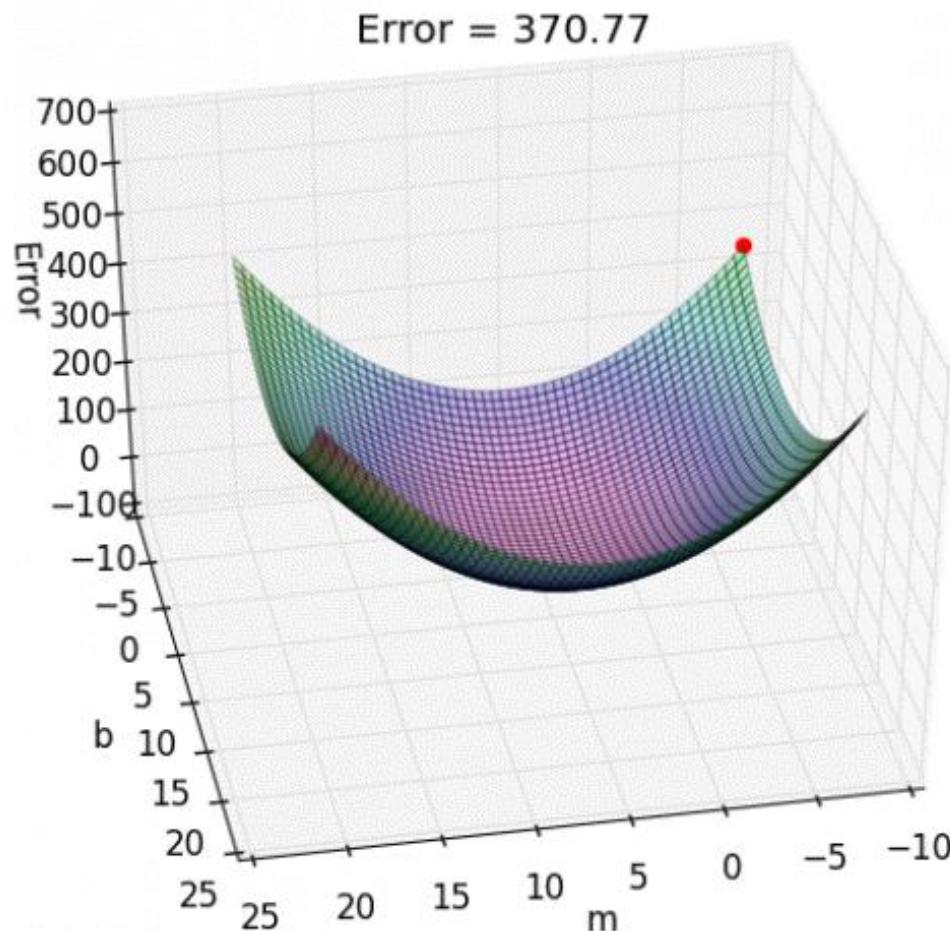
- It might happen that the columns of  $X$  are not linearly independent.
- Then  $XX^T$  is singular and the least squares coefficients  $\bar{w}^*$  are not uniquely defined.
- The singular case occurs most often when two or more inputs were perfectly correlated.
- A natural way to resolve the non-unique representation is by dropping redundant columns in  $X$ .
- Most regression software packages detect these redundancies and automatically implement some strategy for removing them.

# Error Reduction-Gradient Descent

# Basics of Gradient Descent

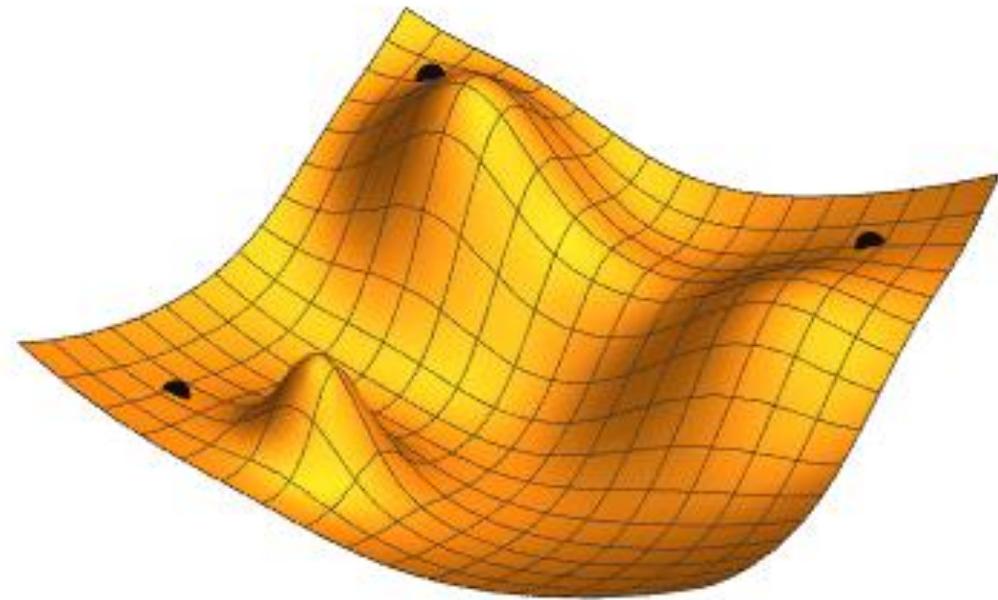
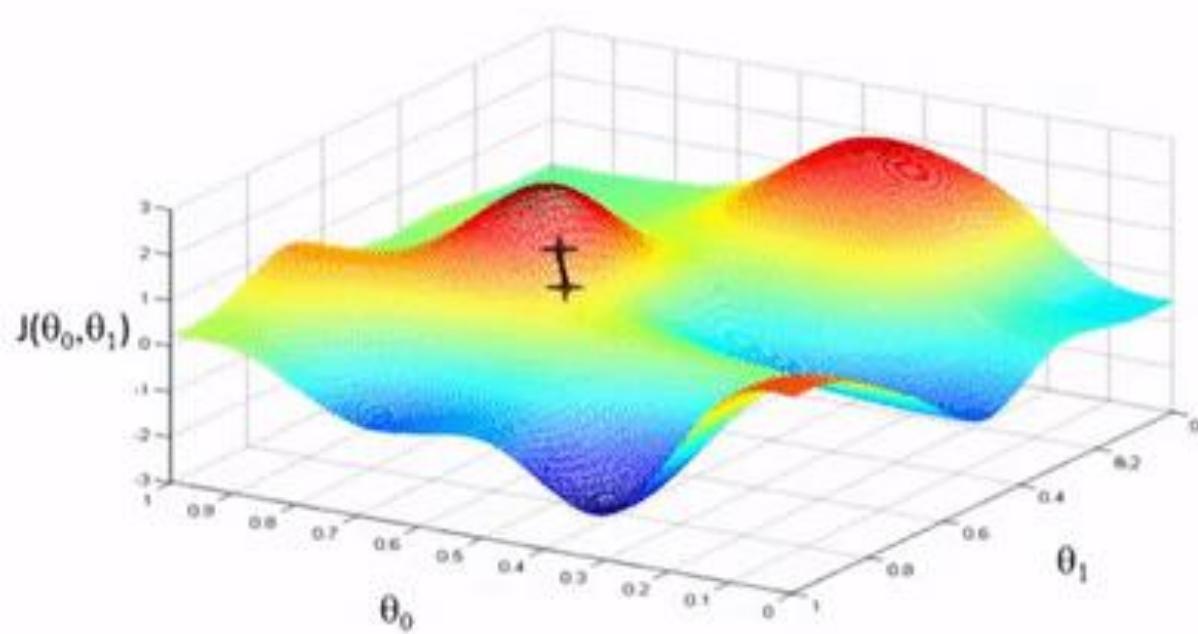
- Gradient descent search helps determine a weight vector that minimizes  $E$  by starting with an arbitrary initial weight vector and then altering it again and again in small steps.
- **Batch gradient descent:** When the weight update is calculated based on all examples in the training dataset, it is called as batch gradient descent.
- **Stochastic gradient descent:** When the weight update is calculated incrementally after each training example or a small group of training example, it is called as stochastic gradient descent.
- Gradient descent procedure has two advantages over merely computing the pseudoinverse:
  - (1) it avoids the problems that arise when  $XX^T$  is singular (it always yields a solution regardless of whether or not  $XX^T$  is singular);
  - (2) it avoids the need for working with large matrices.

# Basics of Gradient Descent



# Basics of Gradient Descent

- The error surface may have multiple local minimums but a single global minimum.
- The objective would be to find out global minimum.



# What is Gradient Descent

## Gradient Descent Optimization Schemes

- Optimization method *Gradient Descent Method* used for minimization tasks. Changes of the weights are made according to the following algorithm:

$$\bar{\mathbf{w}}_{k+1} = \bar{\mathbf{w}}_k - \eta \frac{\partial E}{\partial \bar{\mathbf{w}}} |_k$$

where  $\eta$  denotes the *learning rate*, and  $k$  stands for the actual iteration step.

### Note:

- Need to choose  $\eta$ .
- Needs many iterations.
- Works well even when  $n$  is large.
- Gradient descent serves as the basis for learning algorithms that search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.

# What is Gradient Descent

## Gradient Descent Optimization Schemes

- Optimization method *Gradient Descent Method* used for minimization tasks. Changes of the weights are made according to the following algorithm:

$$\bar{\mathbf{w}}_{k+1} = \bar{\mathbf{w}}_k - \eta \frac{\partial E}{\partial \bar{\mathbf{w}}} |_k$$

where  $\eta$  denotes the *learning rate*, and  $k$  stands for the actual iteration step.

## Approaches for deciding the iteration step:

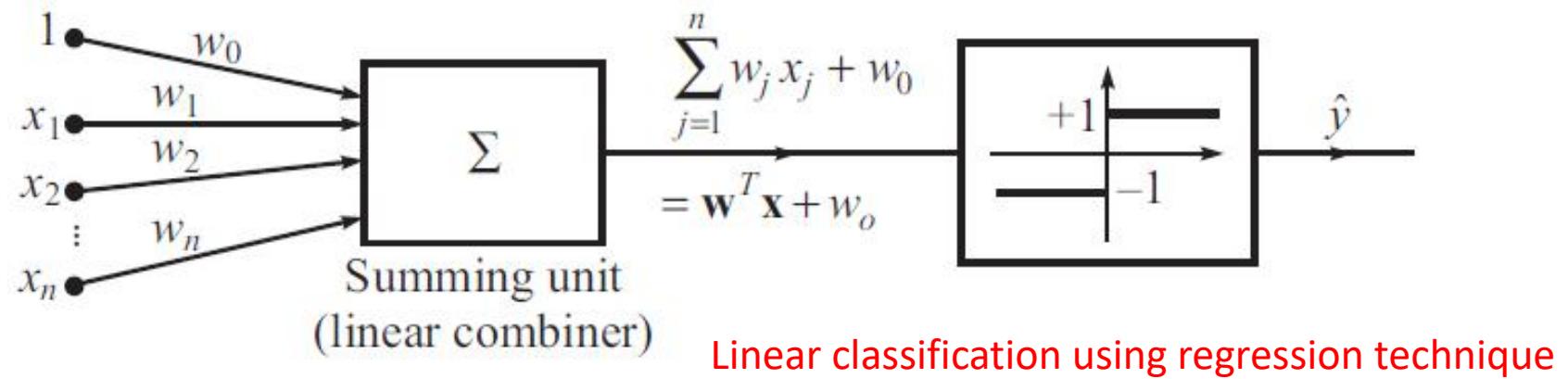
1. *Batch methods* use all the data in one shot.

- $k^{th}$  iteration step means the  $k^{\text{th}}$  presentation of training dataset.
- Gradient is calculated across the entire set of training patterns.

2. *Online methods* is where

- $k$  – iteration step after single data pair is presented.
- Share almost all good features of recursive least square algorithm with reduced computational complexity.

# The gradient descent training rule



$$g(\mathbf{x}) = \sum_{j=1}^n w_j x_j + w_0 = \mathbf{w}^T \mathbf{x} + w_0$$

Let us define a single weight vector  $\bar{\mathbf{w}}$  for the weights  $(\mathbf{w}, w_0)$ :

$$\bar{\mathbf{w}}^T = [w_0 \ w_1 \ w_2 \ \dots \ w_n]^T$$

In terms of the weight vector  $\bar{\mathbf{w}}$ , the output

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

where

$$\bar{\mathbf{x}}^T = [x_0 \ x_1 \ x_2 \ \dots \ x_n]^T; x_0 = 1$$

Performance Criterion:

$$E(\bar{\mathbf{w}}) = \frac{1}{2} \sum_{i=1}^N (y^{(i)} - \bar{\mathbf{w}}^T \bar{\mathbf{x}}^{(i)})^2$$

---To be minimized.  
Called as cost function.

used for computational convenience

# The **gradient descent** training rule

- The error surface is parabolic with a single global minimum.
- The specific parabola will depend on the particular set of training examples.
- The direction of steepest descent along the error surface can be found by computing the derivative of  $E$  with respect to each component of the vector  $\bar{w}$ .
- This vector-derivative is called the gradient of  $E$  w.r.t  $\bar{w}$ , written  $\nabla E(\bar{w})$ .

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

Remember, it can be applied to any objective function, not just for squared distances.

- The negative of this vector gives the direction of steepest decrease.
- Therefore, the training rule for gradient descent is,  $\bar{w} \leftarrow \bar{w} + \Delta \bar{w}$

**Performance Criterion:**

$$E(\bar{w}) = \frac{1}{2} \sum_{i=1}^N (y^{(i)} - \bar{w}^T \bar{x}^{(i)})^2$$

---To be minimized.  
Called as cost function.

$$\Delta \bar{w} = -\eta \nabla E(\bar{w})$$

here,  $\eta$  is learning rate, a +ve constant, determines the step size in the search.

# The **gradient descent** training rule

- This training rule can also be written in its component form:

$$w_j \leftarrow w_j + \Delta w_j; j = 0, 1, 2, \dots, n$$

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

- which shows that steepest descent is achieved by altering each component  $w_j$  of  $w$  in proportion to  $\frac{\partial E}{\partial w_j}$
- starting with an arbitrary initial weight vector, is changed in the direction producing the steepest descent along the error surface.
- The process goes on till the global minimum error is attained.

**Performance Criterion:**

$$E(\bar{\mathbf{w}}) = \frac{1}{2} \sum_{i=1}^N (y^{(i)} - \bar{\mathbf{w}}^T \bar{\mathbf{x}}^{(i)})^2$$

---To be minimized.  
Called as cost function.

# The **gradient** with respect to weight $w_j$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \left[ \frac{1}{2} \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right)^2 \right]$$

The error  $e^{(i)}$  for the  $i^{\text{th}}$  sample of data is given by  $e^{(i)} = y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right)$ .  
It follows that

$$\frac{1}{2} \sum_{i=1}^N \left[ \frac{\partial}{\partial w_j} (e^{(i)})^2 \right] = \sum_{i=1}^N e^{(i)} \frac{\partial e^{(i)}}{\partial w_j}$$

$$\& \quad \frac{\partial e^{(i)}}{\partial w_j} = -x_j^{(i)}$$

$$\longrightarrow \frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \left[ \frac{1}{2} \sum_{i=1}^N (e^{(i)})^2 \right]$$

$$= - \sum_{i=1}^N e^{(i)} x_j^{(i)}$$

$$= - \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right) x_j^{(i)}$$

**Performance Criterion:**

$$E(\bar{\mathbf{w}}) = \frac{1}{2} \sum_{i=1}^N (y^{(i)} - \bar{\mathbf{w}}^T \bar{\mathbf{x}}^{(i)})^2$$

**--To be minimized.  
Called as cost function.**

# The **gradient** with respect to weight $w_j$

$$\frac{\partial E}{\partial w_j} = \frac{\partial}{\partial w_j} \left[ \frac{1}{2} \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right)^2 \right]$$

The gradient with respect to bias,

The error  $e^{(i)}$  for the  $i^{\text{th}}$  sample of data is given by  $\epsilon$

It follows that

$$\frac{1}{2} \sum_{i=1}^N \left[ \frac{\partial}{\partial w_j} (e^{(i)})^2 \right] = \sum_{i=1}^N e^{(i)} \frac{\partial e^{(i)}}{\partial w_j}$$

$$\& \quad \frac{\partial e^{(i)}}{\partial w_j} = -x_j^{(i)}$$

$$\begin{aligned} \xrightarrow{\text{ }} \frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j} \left[ \frac{1}{2} \sum_{i=1}^N (e^{(i)})^2 \right] \\ &= - \sum_{i=1}^N e^{(i)} x_j^{(i)} \end{aligned}$$

$$\frac{\partial E}{\partial w_0} = - \sum_{i=1}^N e^{(i)} = - \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right)$$

Therefore, the weight update rule for gradient descent becomes

$$w_j \leftarrow w_j + \eta \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right) x_j^{(i)}$$

$$w_0 \leftarrow w_0 + \eta \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right)$$

An epoch is a complete run through all the  $N$  associated pairs.

$$= - \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right) x_j^{(i)}$$

# The gradient with respect to weight $w_j$

The gradient with respect to bias,

$$\frac{\partial E}{\partial w_0} = - \sum_{i=1}^N e^{(i)} = - \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right)$$

Therefore, the weight update rule for gradient descent becomes

$$w_j \leftarrow w_j + \eta \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right) x_j^{(i)}$$

$$w_0 \leftarrow w_0 + \eta \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j x_j^{(i)} + w_0 \right) \right)$$

- Once an epoch is completed, the pair  $(x^{(1)}, y^{(1)})$  is presented again and a run is performed through all the pairs again.
- After several epochs, the output error is expected to be sufficiently small.

In terms of iteration index  $k$ , the weight update equations are

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \eta \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j(k) x_j^{(i)} + w_0(k) \right) \right) \mathbf{x}^{(i)}$$

$$w_0(k+1) = w_0(k) + \eta \sum_{i=1}^N \left( y^{(i)} - \left( \sum_{j=1}^n w_j(k) x_j^{(i)} + w_0(k) \right) \right)$$

- k corresponds to the epoch number, the number of times the set of N pairs is presented and cumulative error is compounded.**

# GENERALISATION

Dr. Srikanth Allamsetty

# Over-fitting, Bias and Variance Relationship

# Generalization

- Given a collection of examples  $(x^{(i)}, f(x^{(i)}))$ ;  $i = 1, 2, \dots, N$ , of a function  $f(x)$ , return a function  $h(x)$  that approximates  $f(x)$ .
- $h(x)$  is called hypothesis function and  $f(x)$  is called true function.
- It is not easy to tell whether any particular  $h(\cdot)$  is a good approximation of  $f(\cdot)$ .
- A good approximation will generalize well—that is, will predict novel patterns correctly.
- Generalization performance is, thus, the fundamental problem in inductive learning (Inductive Learning, also known as Concept Learning, is how AI systems attempt to use a generalized rule to carry out observations).
- The off-training set error—the error on points not in the training set, will be used as a measure of generalization performance (generally called testing error).

# Generalization

- If one algorithm appears to outperform another in a specific situation, it is as a result of its fit to the specific learning problem, not the general supremacy of the algorithm.
- Even if the algorithms are widely used and grounded in terms of theory, they will not perform well on certain problems.

# Bias and variance

- A measure of how close the mapping function  $h(x; \mathcal{D}_j)$  is to the desired one is, therefore, given by the error function,

$$\text{error}_{\mathcal{D}_j}[h] = [h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2$$

- The value of this quantity will depend on the dataset  $\mathcal{D}_j$  on which it is trained.
- We write the average over the complete ensemble of datasets as,

$$\text{error}_{\mathcal{D}}[h] = \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2\}$$

Expected value;  
Arithmetic mean of a large number of independent realizations

- It may be that the hypothesis function  $h(\cdot)$  is on average, different from the regression function  $f(x)$ . This is called bias.
- It may be that the hypothesis function is very sensitive to the particular dataset  $\mathcal{D}_j$ , so that for a given  $x$ , it is larger than the required value for some datasets, and smaller for other datasets. This is called variance.

# Bias and variance

$$\begin{aligned}\text{error}_{\mathcal{D}}[h] &= \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2\} \\ &= \underbrace{[\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} - f(\mathbf{x})]^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}]^2\}}_{\text{variance}}\end{aligned}$$

(w.r.t original) (range of the results of hypothesis fn)

- The bias measures the level to which the average (over all datasets) of the hypothesis function is different from the desired function  $f(x)$ .
- The bias term reflects the approximation error that the hypothesis  $h(\cdot)$  is expected to have on average when trained on datasets of same finite size.
- In general, higher the complexity of the hypothesis function (more flexible function with large number of adjustable parameters), the lower is the approximation error.

# Bias and variance

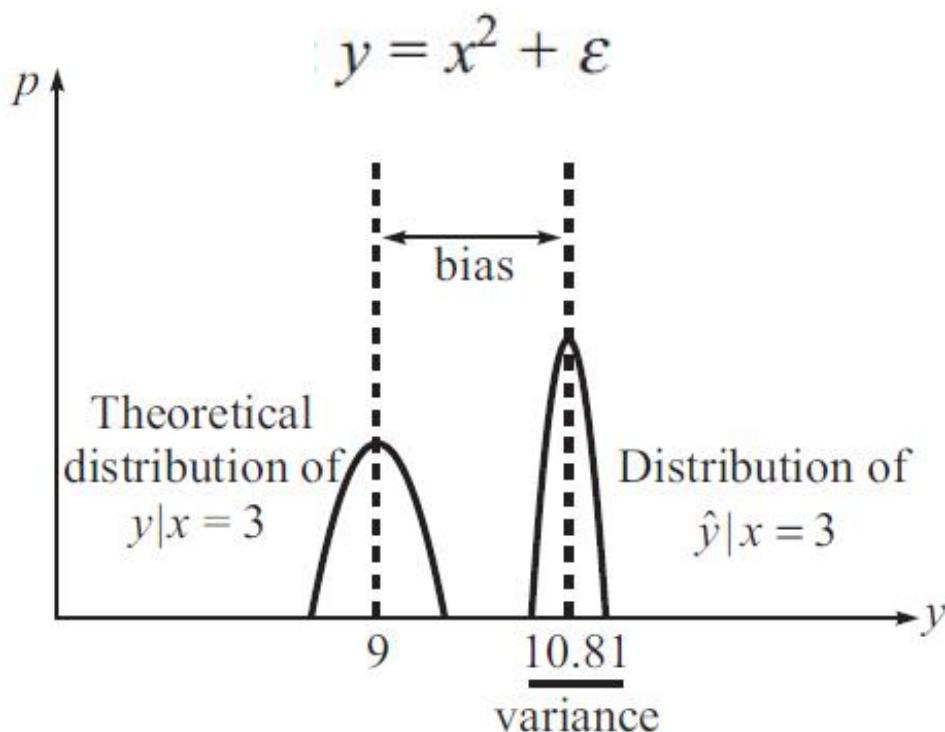
$$\begin{aligned}\text{error}_{\mathcal{D}}[h] &= \mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - f(\mathbf{x})]^2\} \\ &= \underbrace{[\mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\} - f(\mathbf{x})]^2}_{(\text{bias})^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\{[h(\mathbf{x}; \mathcal{D}_j) - \mathbb{E}_{\mathcal{D}}\{h(\mathbf{x}; \mathcal{D}_j)\}]^2\}}_{\text{variance}}\end{aligned}$$

(w.r.t original) (range of the results of hypothesis fn)

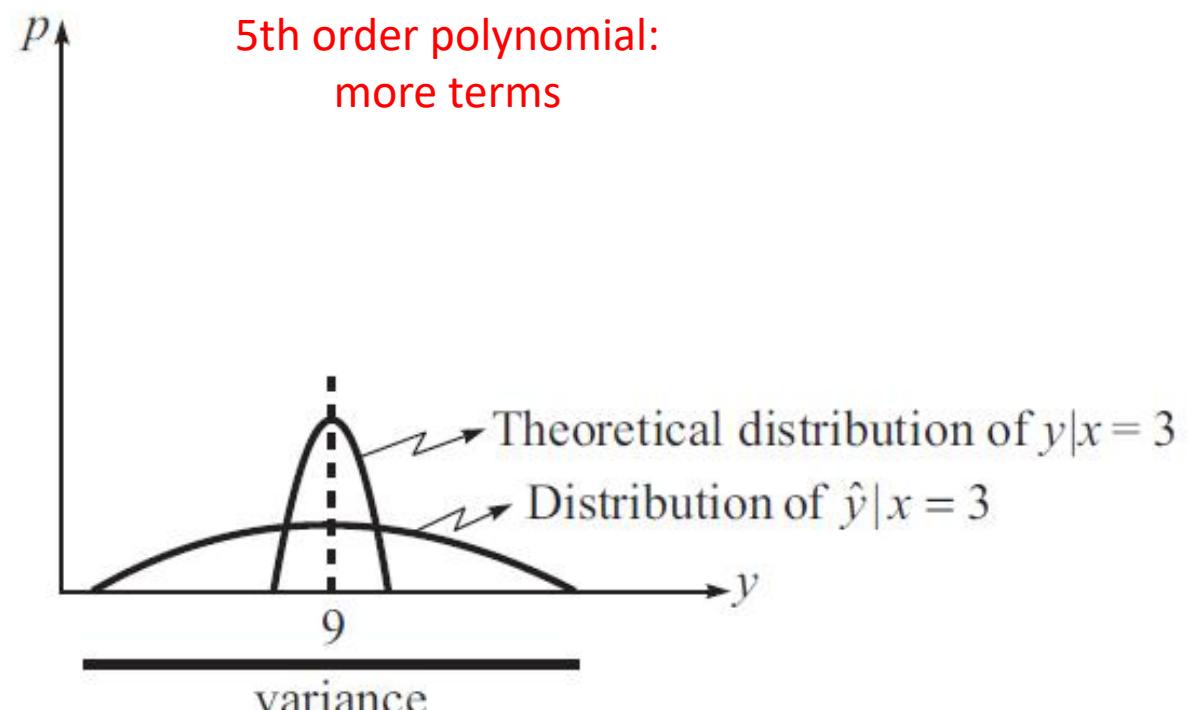
- The variance is a measure of the level to which the hypothesis function  $h(\mathbf{x}; \mathcal{D}_j)$  is sensitive to the specific selection of the dataset.
- The variance term reflects the capability of the trained model on a data sample to generalize to other data samples.
- Low variance means that the estimate of  $f(\mathbf{x})$  based on a data sample does not change much on the average as the data sample varies.
- Unfortunately, the higher the complexity of the hypothesis function (which results in low bias/low approximation error), the higher is the variance.

# Bias and variance

- To minimize the overall mean-square-error, we need a hypothesis that results in low bias and low variance.
- This is known as bias-variance dilemma or bias-variance trade-off.



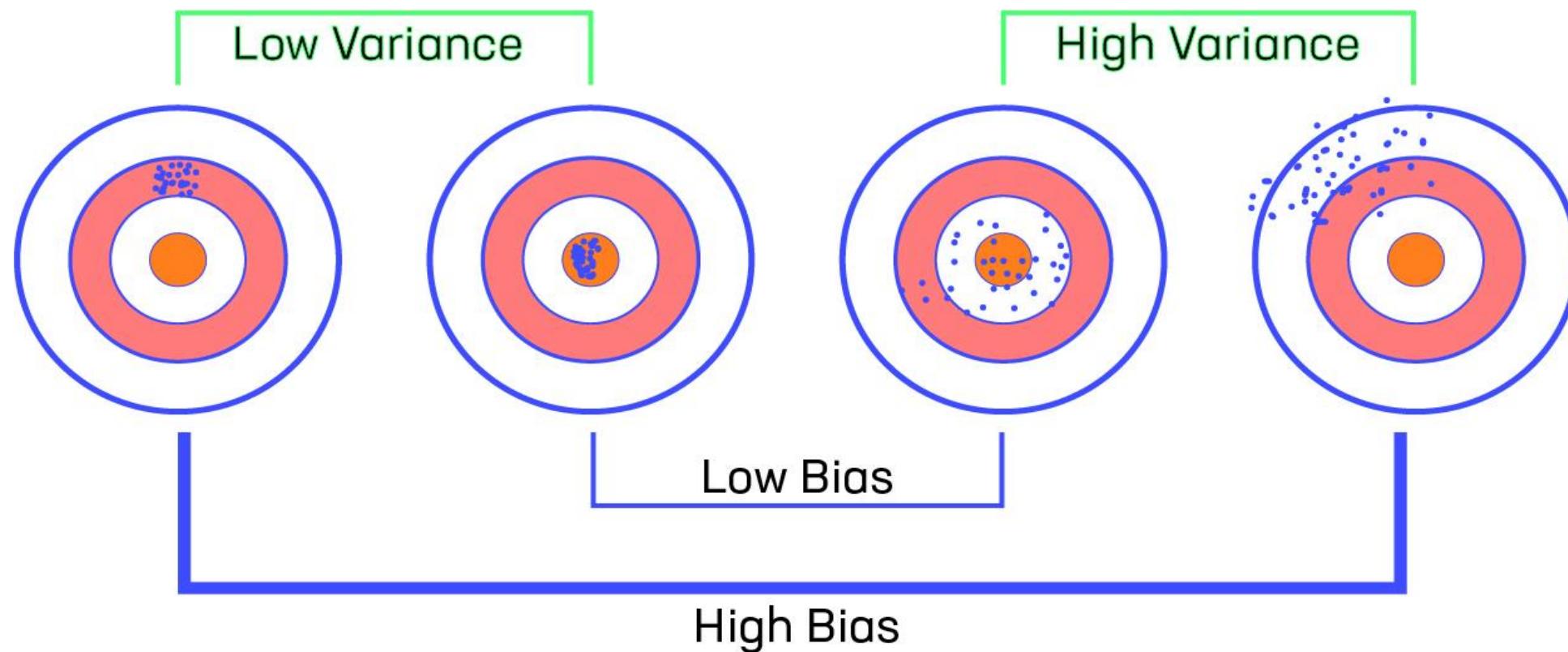
(a) Linear curve fitting



(b) Higher-order polynomial fitting

# Bias and variance

- To minimize the overall mean-square-error, we need a hypothesis that results in low bias and low variance.
- This is known as bias-variance dilemma or bias-variance trade-off.



# Bias and variance

- To minimize the overall mean-square-error, we need a hypothesis that results in low bias and low variance.
- This is known as bias-variance dilemma or bias-variance trade-off.

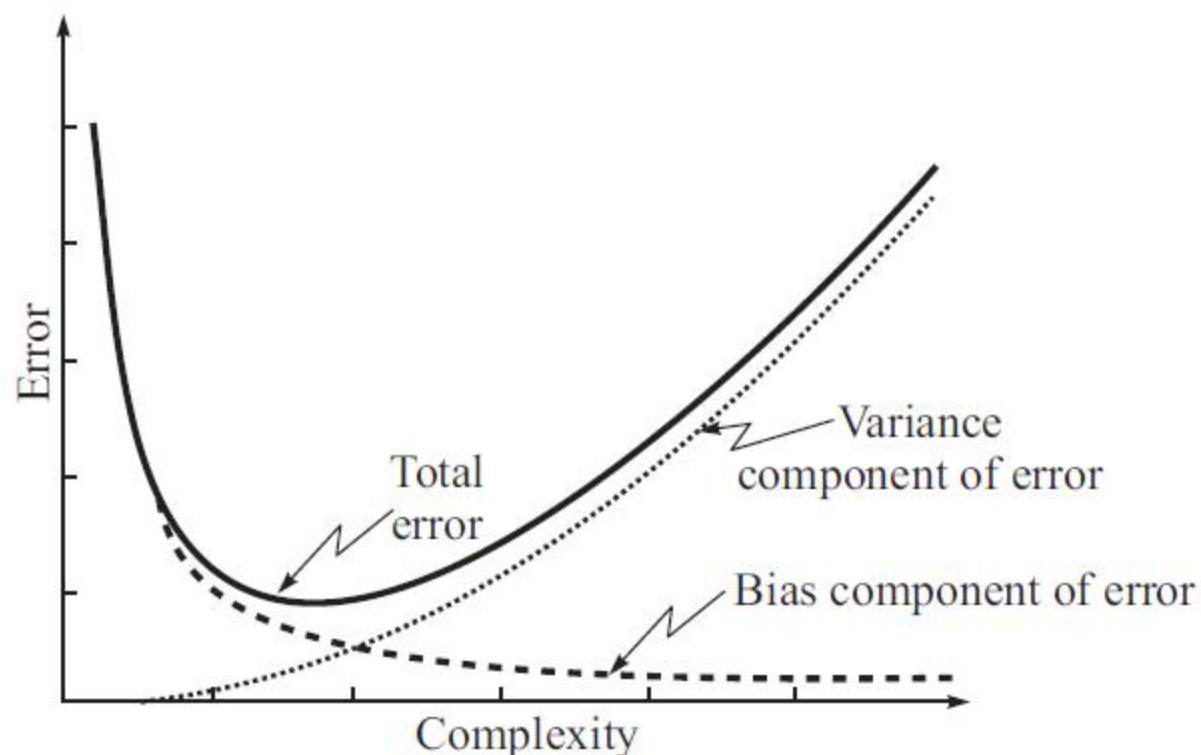
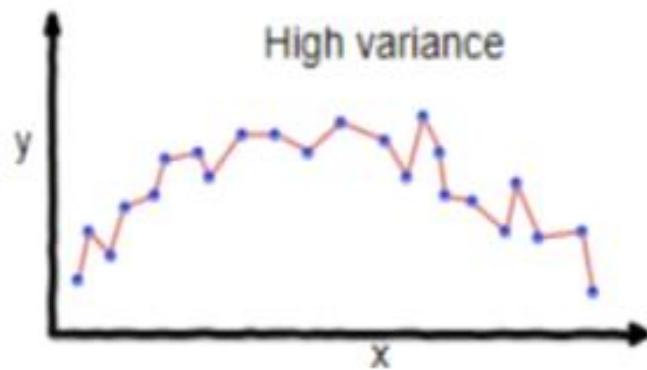
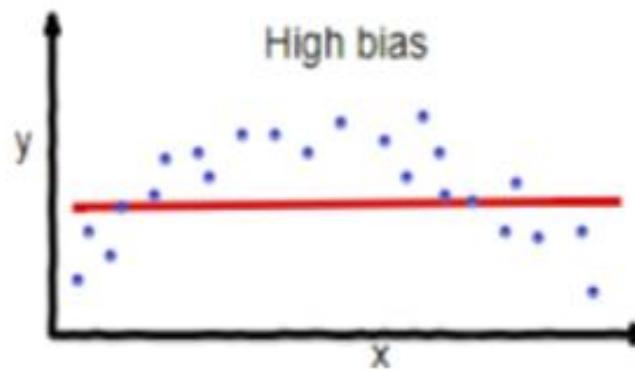


Figure 2.4 Bias-variance trade-off

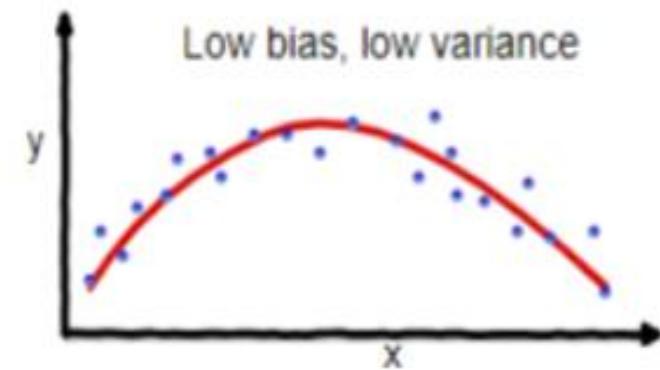
# Over-fitting



overfitting

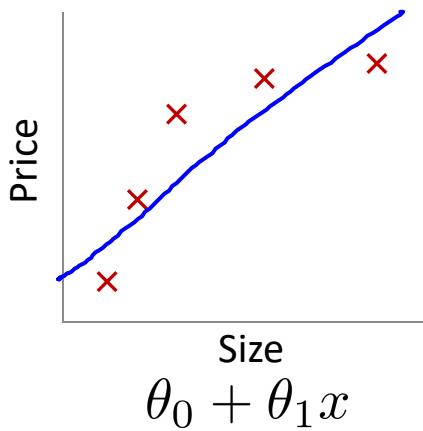


underfitting

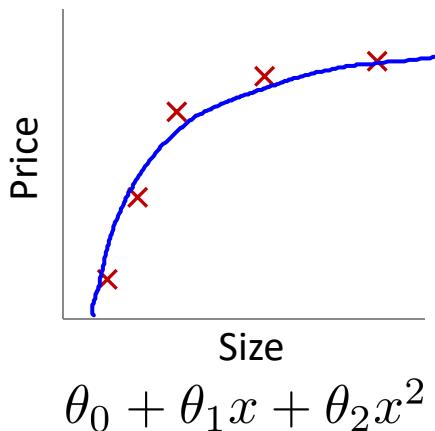


Good balance

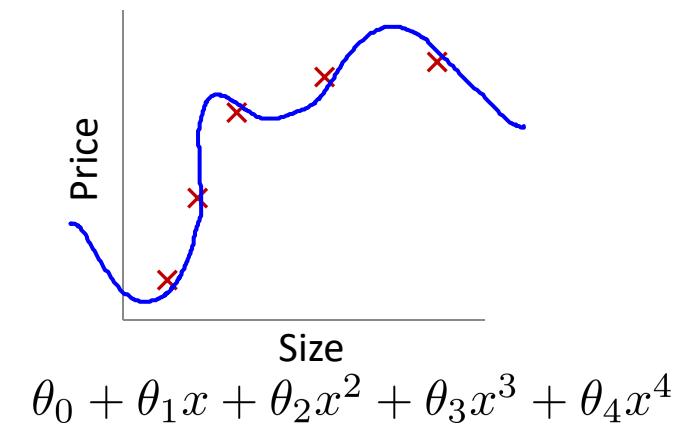
# Over-fitting



High bias  
(underfit)  
 $d=1$



"Just right"  
 $d=2$

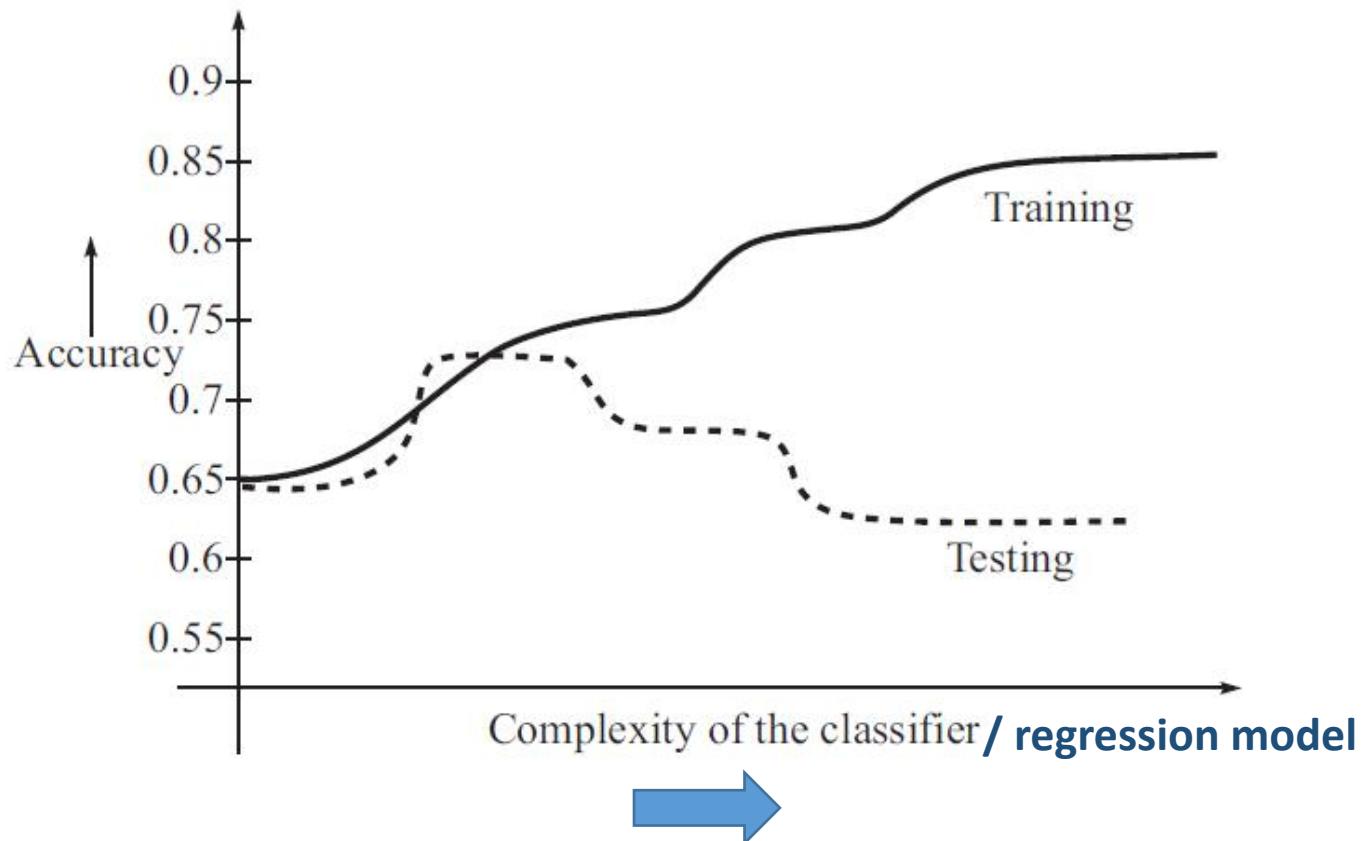


High variance  
(overfit)  
 $d=4$

# Over-fitting

- **Occam's Razor Principle:** The simpler explanations are more reasonable, and any unnecessary complexity should be shaved off.
- ‘simpler’ may imply needing lesser parameters, lesser training time, fewer attributes for data representation, and lesser computational complexity.
- Occam’s razor principle suggests hypothesis functions that avoid overfitting of the training data.
- A learning machine is said to **overfit** the training examples if certain other learning machine that fits the training examples less well, but actually performs better over the total distribution of patterns (i.e., including patterns beyond the training set).

# Over-fitting



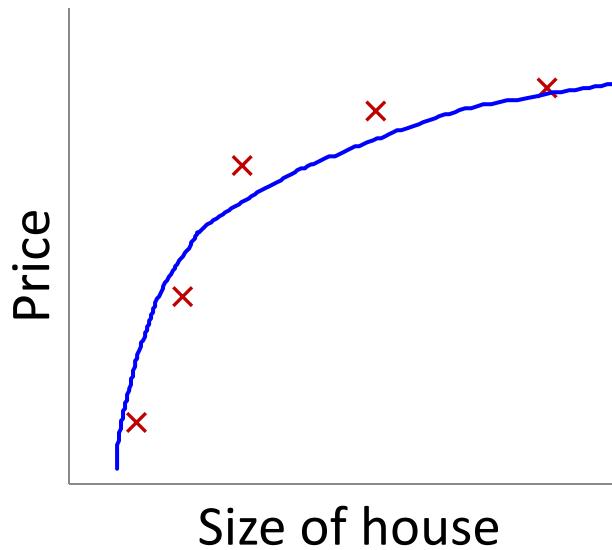
(more number of weights in the neural network, large number of nodes in the decision tree, more number of rules in a fuzzy logic model, etc.)

# Addressing overfitting:

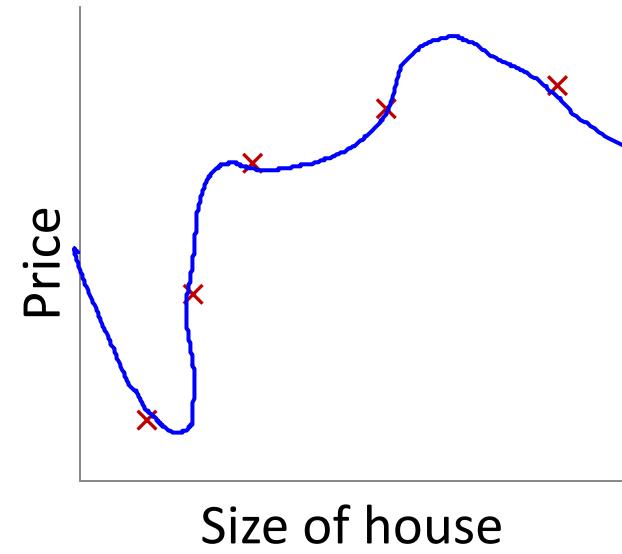
Options:

1. Reduce number of features.
  - Manually select which features to keep.
  - Model selection algorithm (later in course).
2. Regularization.
  - Keep all the features, but reduce magnitude/values of parameters  $\theta_j$  or  $\beta$ .
  - Works well when we have a lot of features, each of which contributes a bit to predicting  $y$ .

# Regularization



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

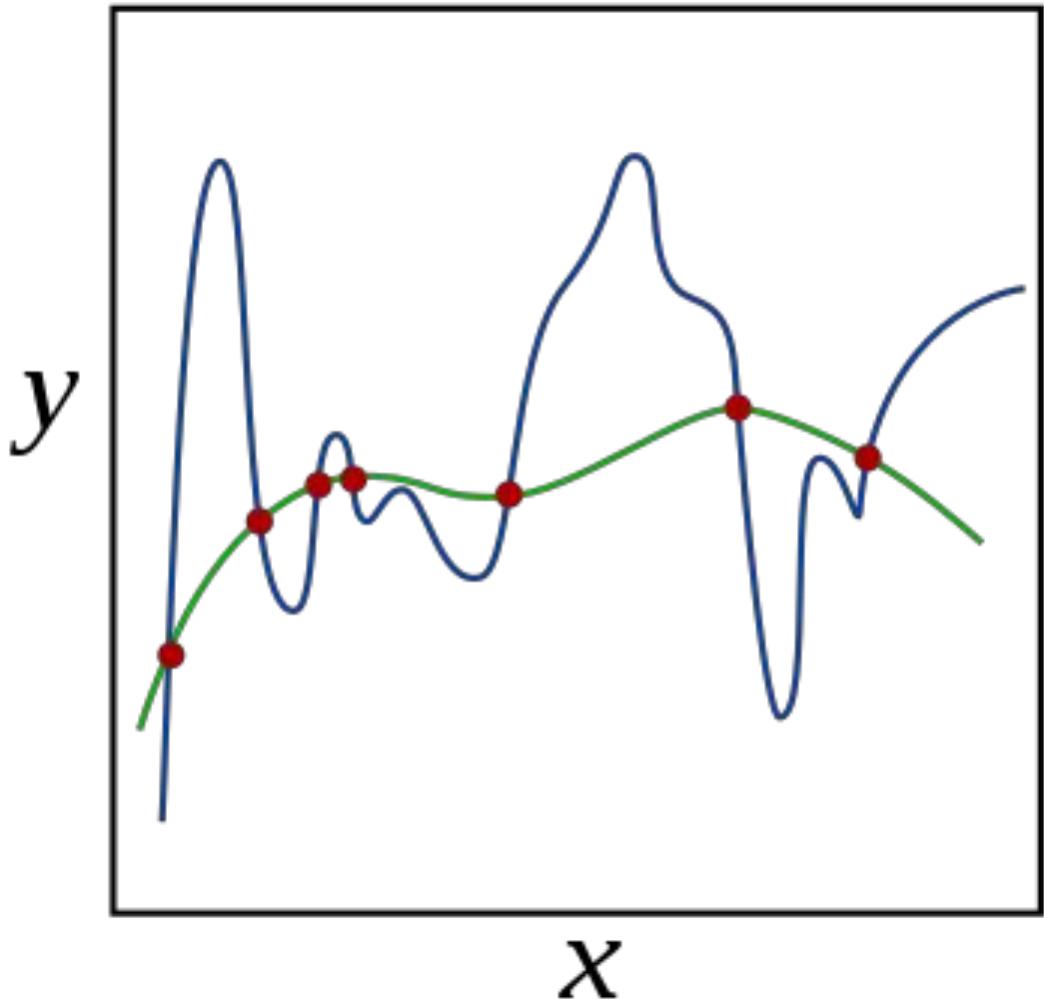


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Over-fitting can be avoided if suppose we penalize and make  $\theta_3$ ,  $\theta_4$  really small.

# Regularization

- This is a form of regression, that constrains/ regularizes or shrinks the coefficient estimates towards zero.
- In other words, this technique discourages learning a more complex or flexible model, so as to avoid the risk of overfitting.



# Regularization

- $Y \approx \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p$
- Here  $Y$  represents the learned relation and  $\beta$  represents the coefficient estimates for different variables or predictors ( $X$ ).
- The fitting procedure involves a loss function, known as residual sum of squares or RSS.
- The coefficients are chosen, such that they minimize this loss function.

$$\text{RSS} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

# Regularization

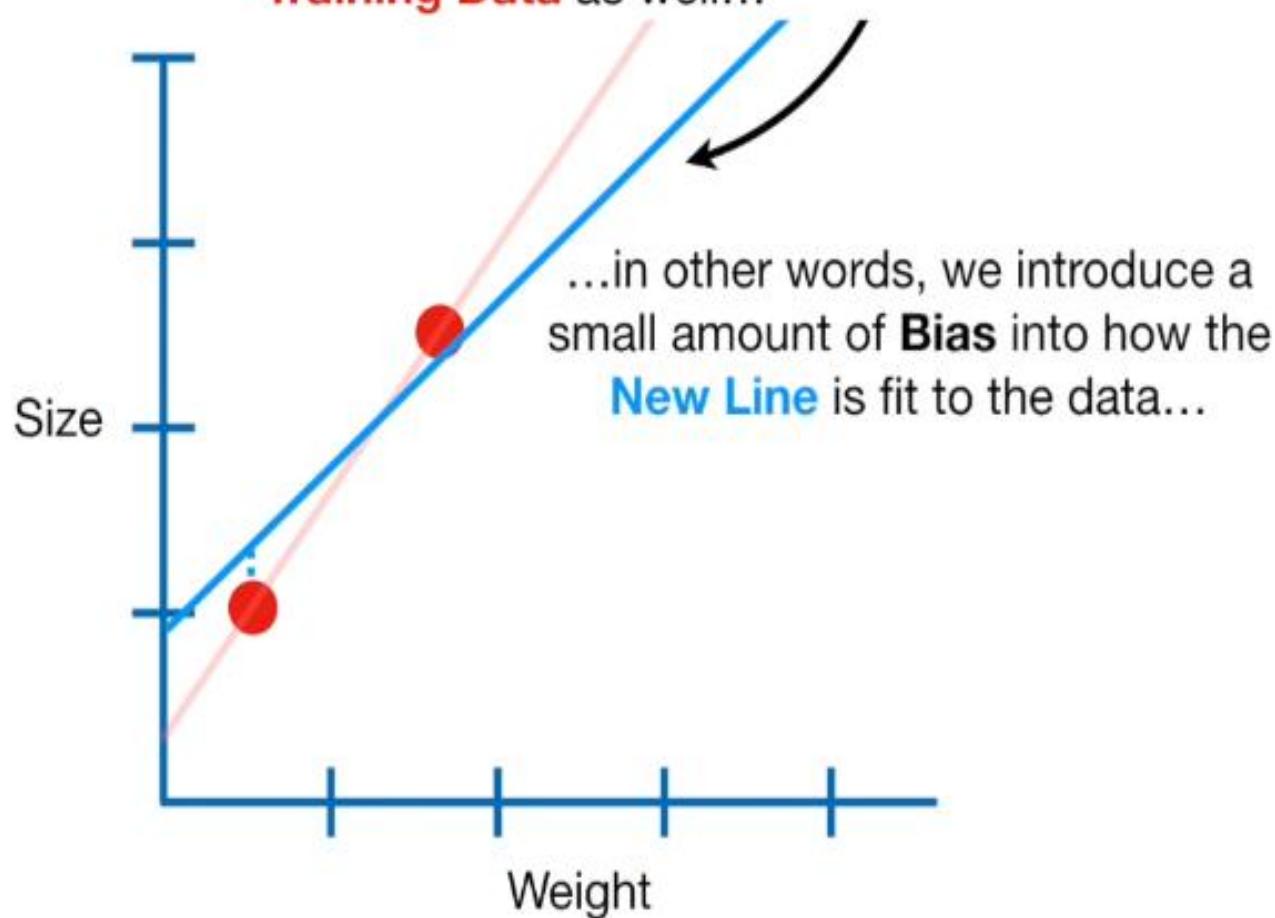
$$\text{RSS} = \sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2.$$

- Now, this will adjust the coefficients based on your training data.
- If there is noise in the training data (or more terms in the equation), then the estimated coefficients won't generalize well to the future data.
- This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.
- Regularization, significantly reduces the variance of the model, without substantial increase in its bias.

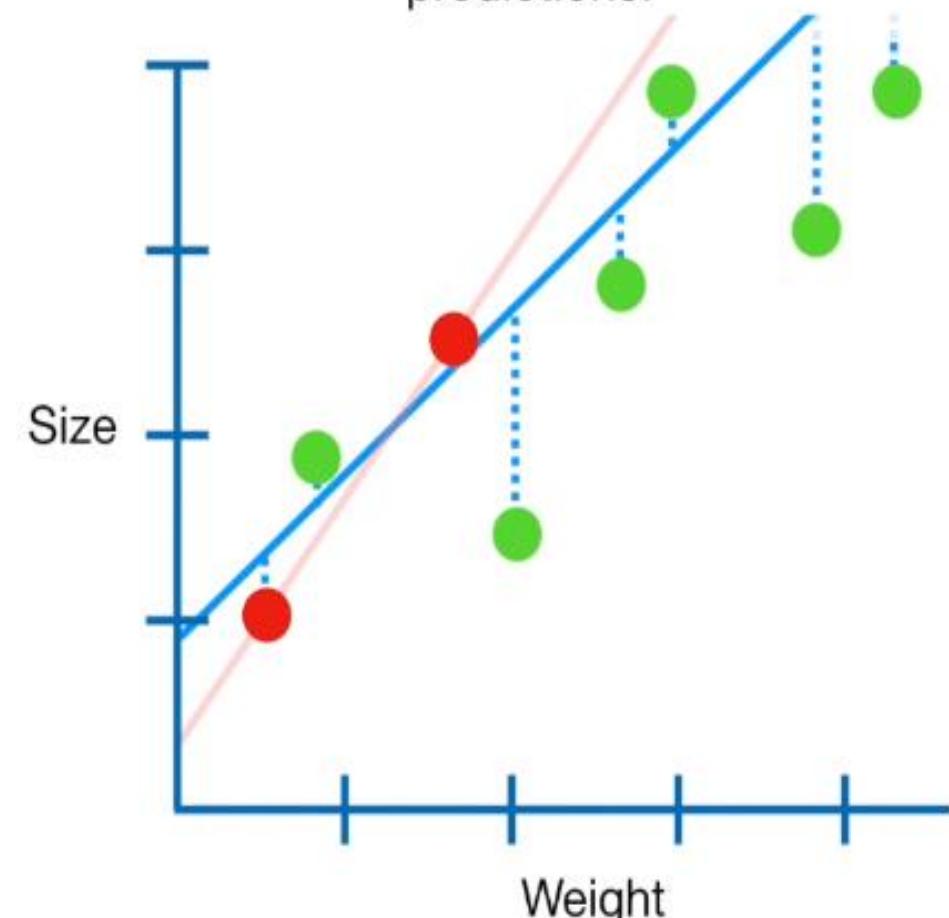
# RIDGE Regression

# Basics

The main idea behind **Ridge Regression** is to find a **New Line** that doesn't fit the **Training Data** as well...



In other words, by starting with a slightly worse fit, **Ridge Regression** can provide better long term predictions.



# Basics

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 = \text{RSS} + \lambda \sum_{j=1}^p \beta_j^2$$

- Above image shows ridge regression, where the RSS is modified by adding the shrinkage quantity.
- Now, the coefficients are estimated by minimizing this function.
- Here,  $\lambda$  is the tuning parameter that decides how much we want to penalize the flexibility of our model.
- The increase in flexibility of a model is represented by increase in its coefficients
- if we want to minimize the above function, then these coefficients need to be small.

# Basics

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 = \text{RSS} + \lambda \sum_{j=1}^p \beta_j^2$$

- This is how the Ridge regression technique prevents coefficients from rising too high.
- Also, notice that we shrink the estimated association of each variable with the response, except the intercept  $\beta_0$ . This intercept is a measure of the mean value of the response when  $x_{i1} = x_{i2} = \dots = x_{ip} = 0$ .
- When  $\lambda = 0$ , the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares.
- However, as  $\lambda \rightarrow \infty$ , the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero.

# RIDGE vs Ordinary Least Squares

$$L_{ridge}(\hat{\beta}) = \sum_{i=1}^n (y_i - x_i' \hat{\beta})^2 + \lambda \sum_{j=1}^m \hat{\beta}_j^2 = \|y - X\hat{\beta}\|^2 + \lambda \|\hat{\beta}\|^2.$$

- Solving this for  $\hat{\beta}$  gives the ridge regression estimates

$$\hat{\beta}_{ridge} = (X'X + \lambda I)^{-1}(X'Y)$$

$I$  - Identity matrix

- You might have remembered that the optimal solution in case of Ordinary least Squares (OLS) is

$$\bar{\beta}^* = (X X^T)^{-1} X y$$

# RIDGE vs Ordinary least Squares

- The  $\lambda$  parameter is the regularization penalty. Notice that:
  - As  $\lambda \rightarrow 0$ ,  $\hat{\beta}_{ridge} \rightarrow \hat{\beta}_{OLS}$ ;
  - As  $\lambda \rightarrow \infty$ ,  $\hat{\beta}_{ridge} \rightarrow 0$ .
- So, setting  $\lambda$  to 0 is the same as using the OLS, while the larger its value, the stronger is the coefficients' size penalized.

# Constraints

- As can be seen, selecting a good value of  $\lambda$  is critical.
- Cross validation comes in handy for this purpose.
- The larger LAMBDA means, our prediction became less sensitive to the independent variables.
- The coefficient estimates produced by this method are also known as the L2 norm.
- The coefficients that are produced by the standard least squares method are scale equivariant, i.e. if we multiply each input by  $c$  then the corresponding coefficients are scaled by a factor of  $1/c$ .
- Therefore, regardless of how the predictor is scaled, the multiplication of predictor and coefficient ( $X_j\beta_j$ ) remains the same.

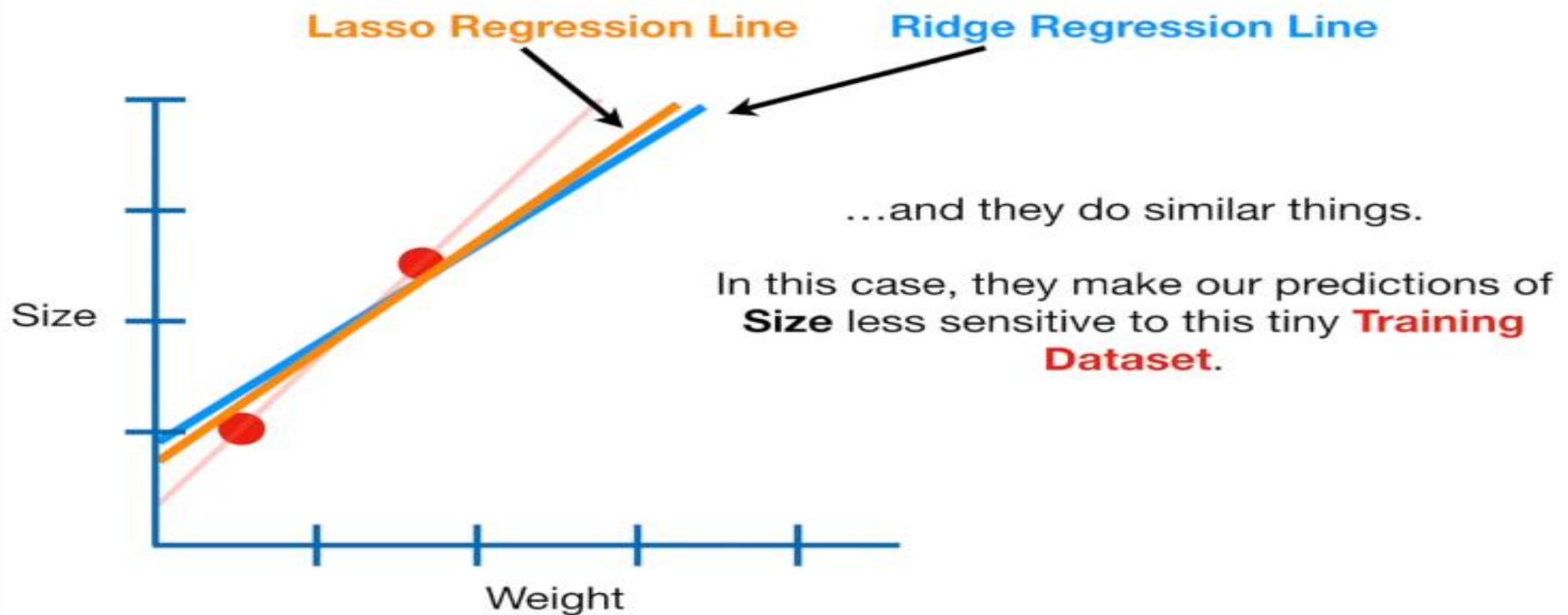
# Standardizing the predictors

- However, this is not the case with ridge regression, and therefore, we need to standardize the predictors or bring the predictors to the same scale before performing ridge regression.
- The formula used to do this is given below.

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n} \sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}}, \quad j=1,2,3,\dots,p \text{ (no. of features)}$$
$$i=1,2,3,\dots,n \text{ (no. of examples)}$$

# LASSO Regression

# Basics



# Basics

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = \text{RSS} + \lambda \sum_{j=1}^p |\beta_j|.$$

- Lasso is a variation, in which the above function is minimized.
- It's clear that this variation differs from ridge regression only in penalizing the high coefficients.
- It uses  $|\beta_j|$  (modulus) instead of squares of  $\beta$ , as its penalty.
- In statistics, this is known as the L1 norm.

# Basics

$$\sum_{i=1}^n \left( y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| = \text{RSS} + \lambda \sum_{j=1}^p |\beta_j|.$$

- Lasso regression stands for Least Absolute Shrinkage and Selection Operator.
- It also adds penalty term to the cost function.
- This term is the absolute sum of the coefficients.

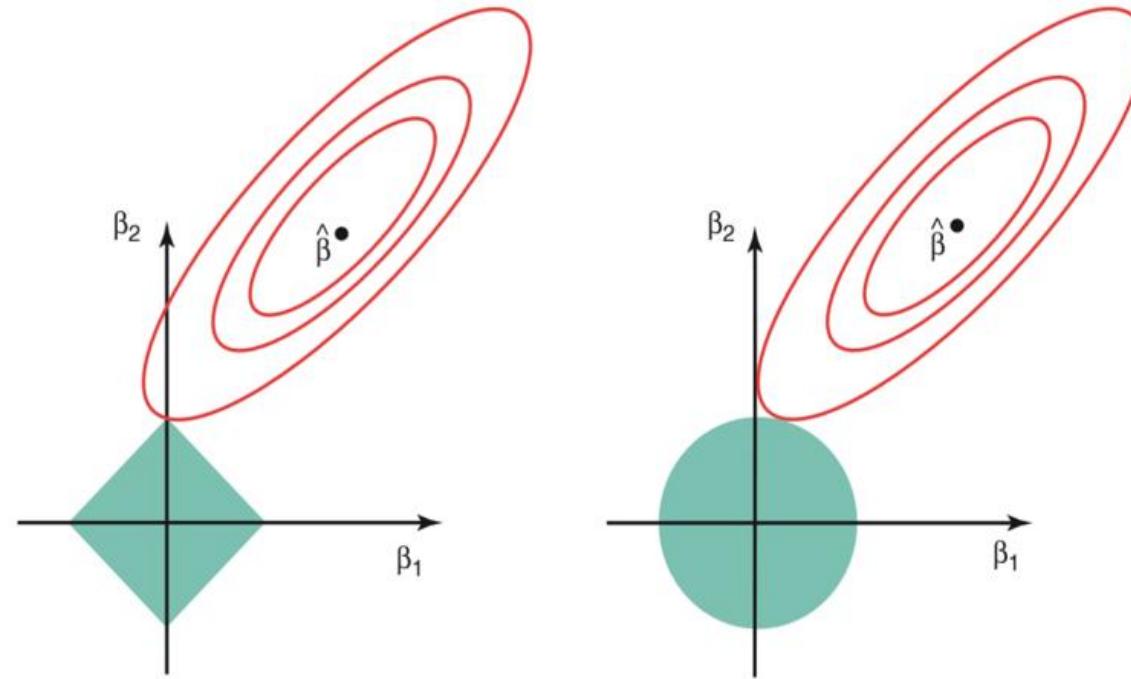
# LASSO vs RIDGE

- Lasso regression differs from ridge regression in a way that it uses absolute values within the penalty function, rather than that of squares.
- This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero.
- The more penalty is applied, the more the estimates get shrunk towards absolute zero.
- This helps to variable selection out of given range of n variables.

# LASSO vs RIDGE

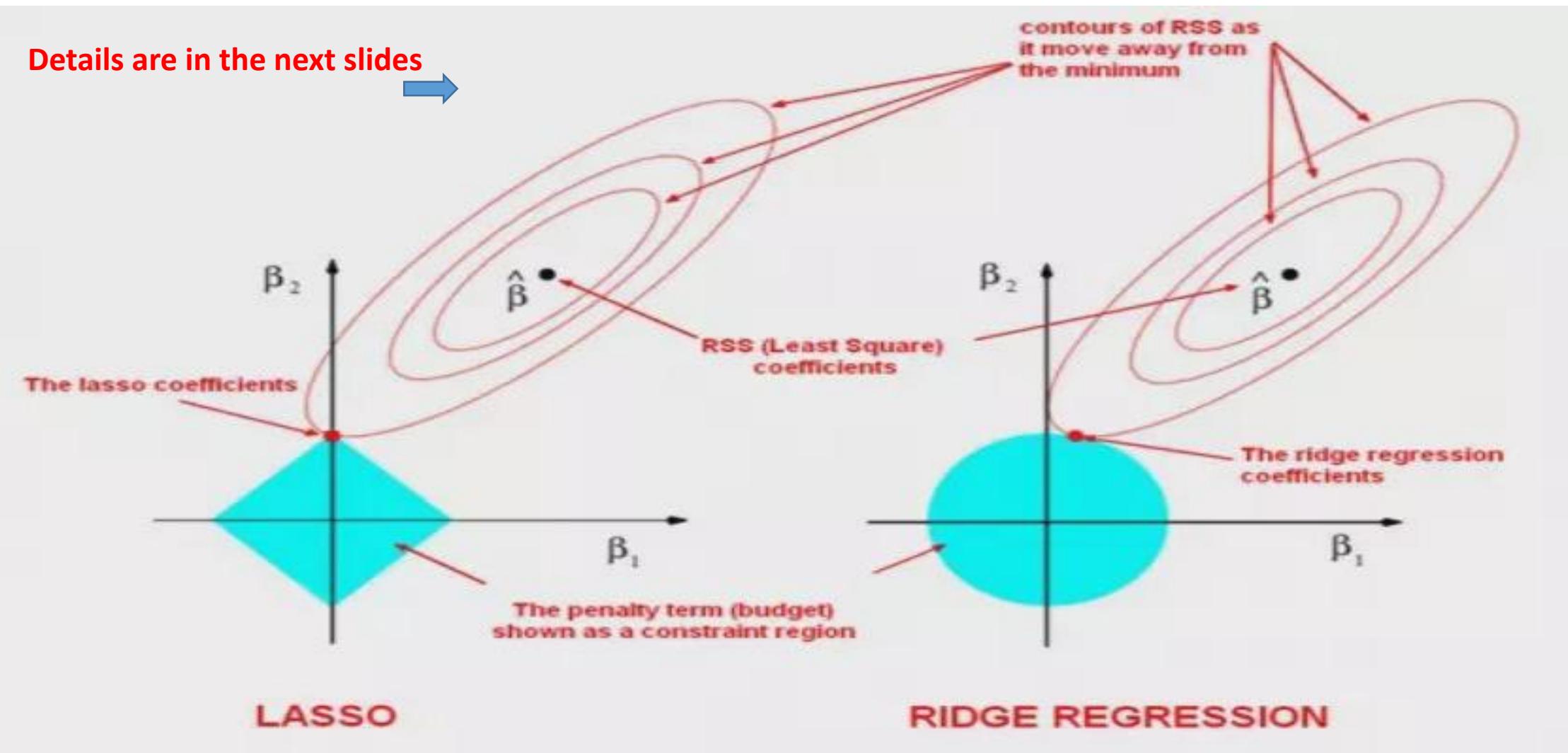
- The ridge regression can be thought of as solving an equation, where summation of squares of coefficients is less than or equal to  $s$ .
- And the Lasso can be thought of as an equation where summation of modulus of coefficients is less than or equal to  $s$ .
- Here,  $s$  is a constant that exists for each value of shrinkage factor  $\lambda$ .
- Consider there are 2 parameters in a given problem.
- Then according to above formulation, lasso regression is expressed by  $|\beta_1| + |\beta_2| \leq s$  and the ridge regression is expressed by  $\beta_1^2 + \beta_2^2 \leq s$ .
- This implies that ridge regression coefficients have the smallest RSS (loss function) for all points that lie within the circle given by  $\beta_1^2 + \beta_2^2 \leq s$ .
- Similarly, the lasso coefficients have the smallest RSS (loss function) for all points that lie within the diamond given by  $|\beta_1| + |\beta_2| \leq s$ .

# LASSO vs RIDGE

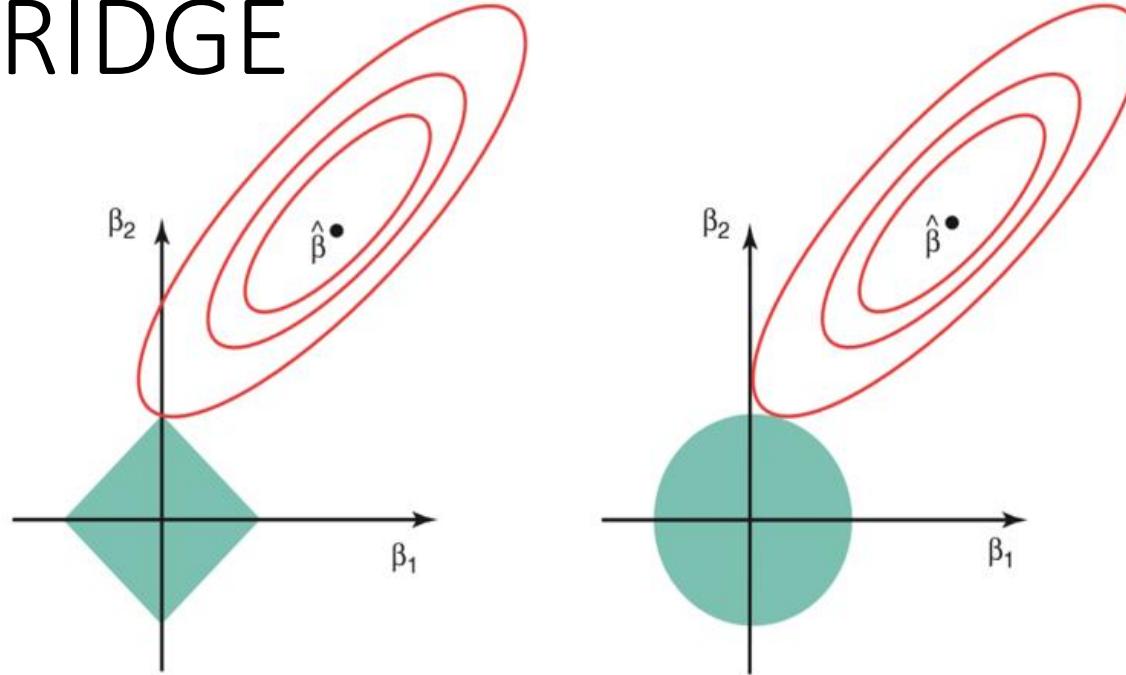


- This implies that ridge regression coefficients have the smallest RSS (loss function) for all points that lie within the circle given by  $\beta_1^2 + \beta_2^2 \leq s$ .
- Similarly, the lasso coefficients have the smallest RSS (loss function) for all points that lie within the diamond given by  $|\beta_1| + |\beta_2| \leq s$ .

# LASSO vs RIDGE

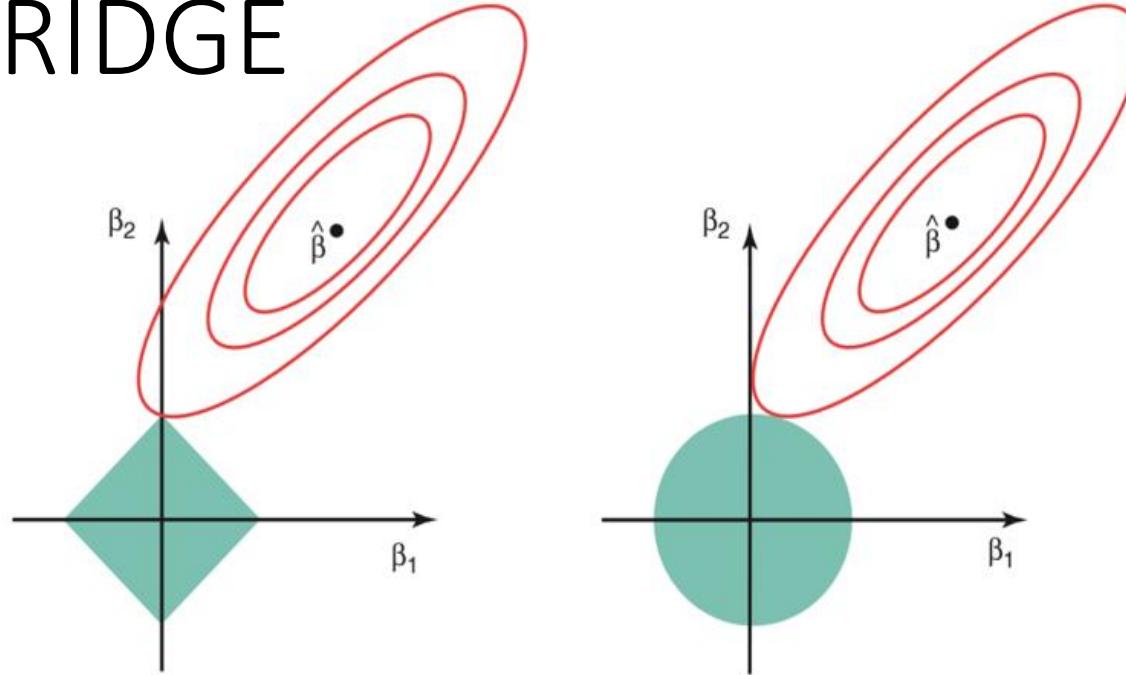


# LASSO vs RIDGE



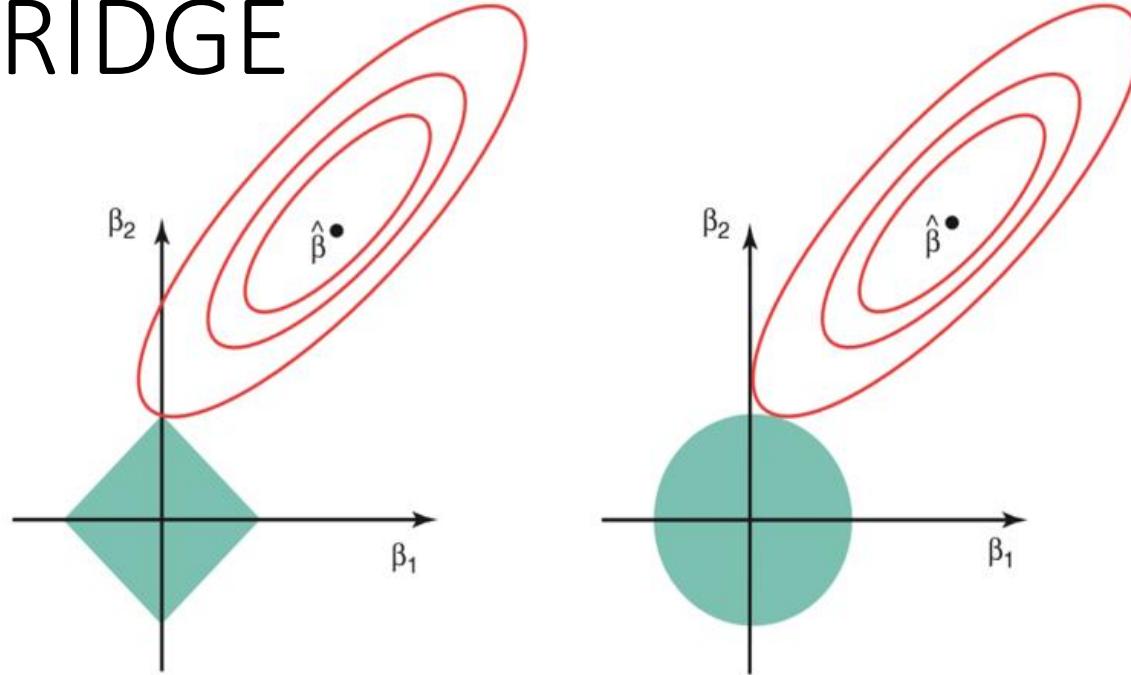
- The above image shows the constraint functions(green areas), for lasso(left) and ridge regression(right), along with contours for RSS (red ellipse).
- Points on the ellipse share the value of RSS.

# LASSO vs RIDGE



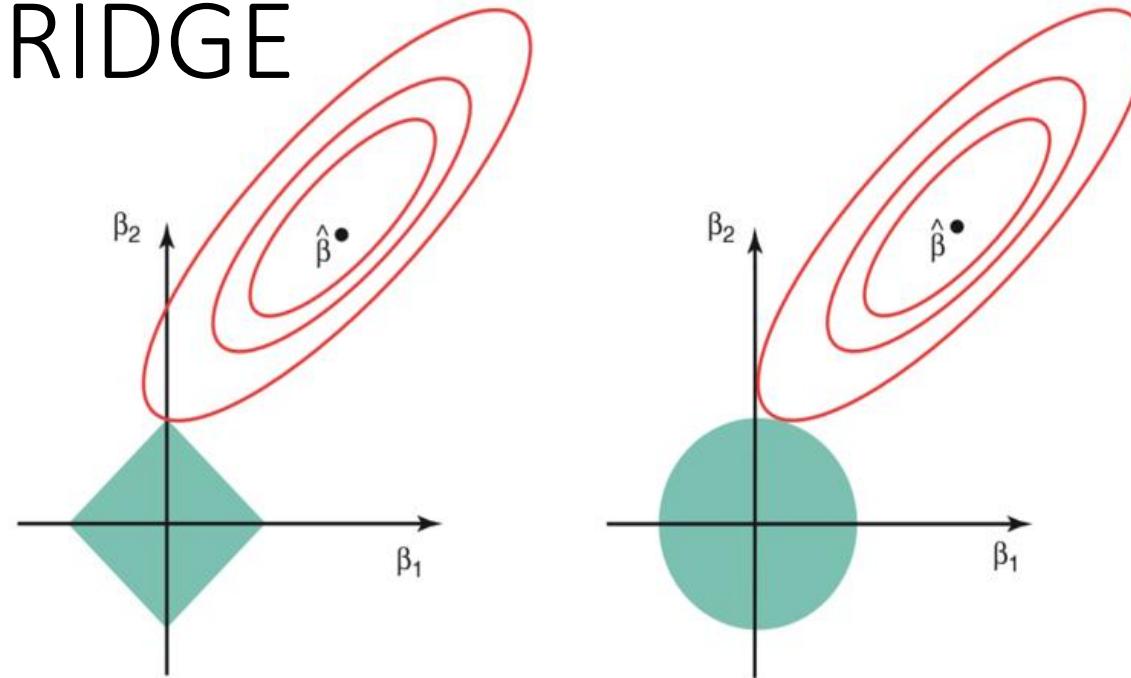
- For a very large value of  $s$ , the green regions will contain the center of the ellipse, making coefficient estimates of both regression techniques, equal to the least squares estimates.
- But, in this above case, the lasso and ridge regression coefficient estimates are given by the first point at which an ellipse contacts the constraint region.

# LASSO vs RIDGE



- Since ridge regression has a circular constraint with no sharp points, this intersection will not generally occur on an axis, and so the ridge regression coefficient estimates will be exclusively non-zero.
- However, the lasso constraint has corners at each of the axes, and so the ellipse will often intersect the constraint region at an axis.

# LASSO vs RIDGE



- When this occurs, one of the coefficients will equal zero.
- In higher dimensions (where parameters are much more than 2), many of the coefficient estimates may equal zero simultaneously.
- That means, in the case of the lasso, the L1 penalty has the effect of forcing some of the coefficient estimates to be exactly equal to zero when the tuning parameter  $\lambda$  is sufficiently large.

# LASSO vs RIDGE

- This sheds light on the obvious disadvantage of ridge regression, which is model interpretability.
- It will shrink the coefficients for least important predictors, very close to zero.
- But it will never make them exactly zero.
- In other words, the final model will include all predictors.
- Therefore, the lasso method also performs variable selection and is said to yield sparse models.
- You may visit <https://www.pluralsight.com/guides/linear-lasso-ridge-regression-scikit-learn> to see a numerical example.

# What does **Regularization** achieve?

- The tuning parameter  $\lambda$ , used in the regularization techniques described above, controls the impact on bias and variance.
- As the value of  $\lambda$  rises, it reduces the value of coefficients and thus reducing the variance.
- Till a point, this increase in  $\lambda$  is beneficial as it is only reducing the variance (hence avoiding overfitting), without loosing any important properties in the data.
- But after certain value, the model starts loosing important properties, giving rise to bias in the model and thus underfitting.
- Therefore, the value of  $\lambda$  should be carefully selected.

## Debugging a learning algorithm:

Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples  $\rightarrow$  fixes high variance
- Try smaller sets of features  $\rightarrow$  fixes high variance
- Try getting additional features  $\rightarrow$  fixes high bias
- Try adding polynomial features ( $x_1^2, x_2^2, x_1x_2$ , etc)  $\rightarrow$  fixes high bias.
- Try decreasing  $\lambda \rightarrow$  fixes high bias
- Try increasing  $\lambda \rightarrow$  fixes high variance

# CLASSIFICATION

Dr. Srikanth Allamsetty

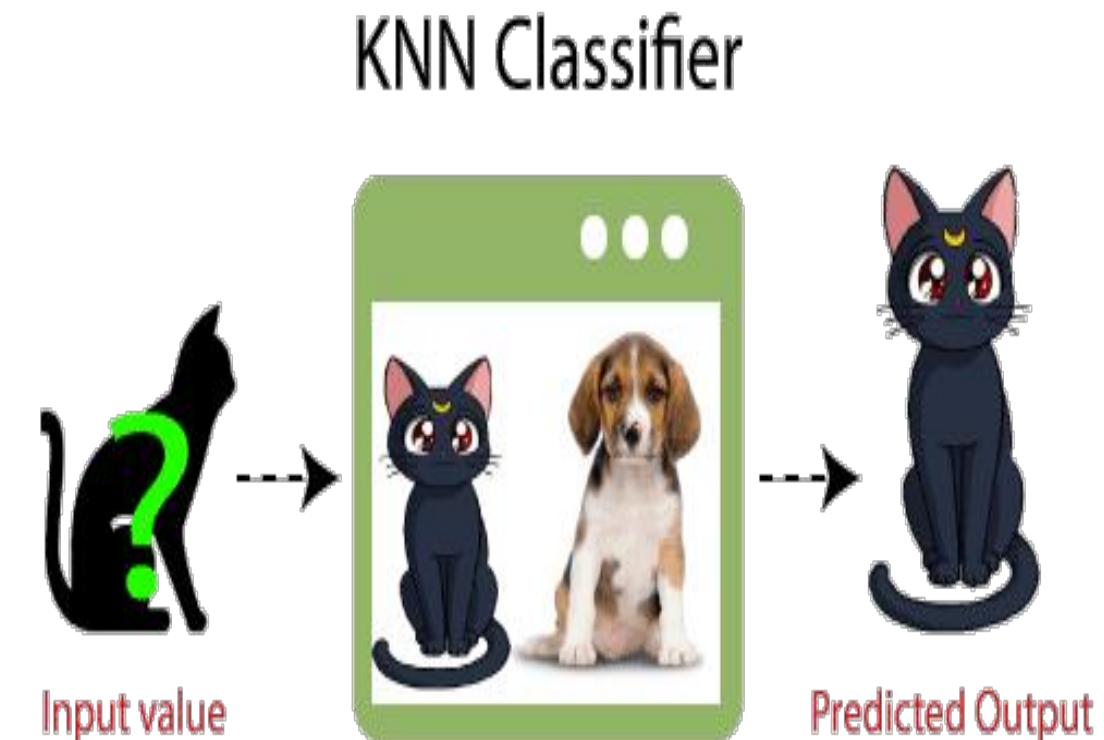
# Nearest Neighbor Learning

# Basics

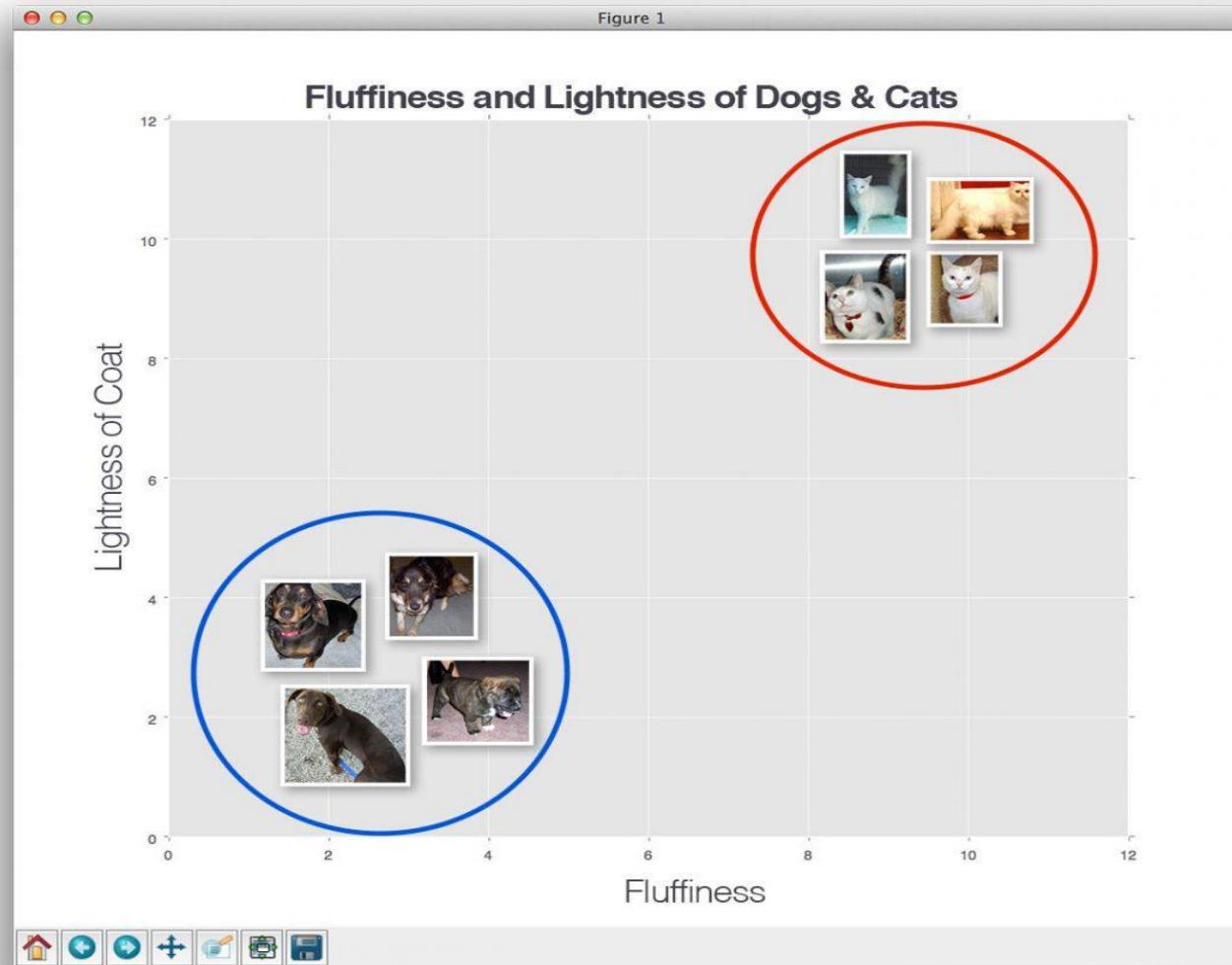
- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- For any given problem, a small value of k will lead to a large variance in predictions; Alternatively, setting k to a large value may lead to a large model bias.

# Using it for pattern recognition

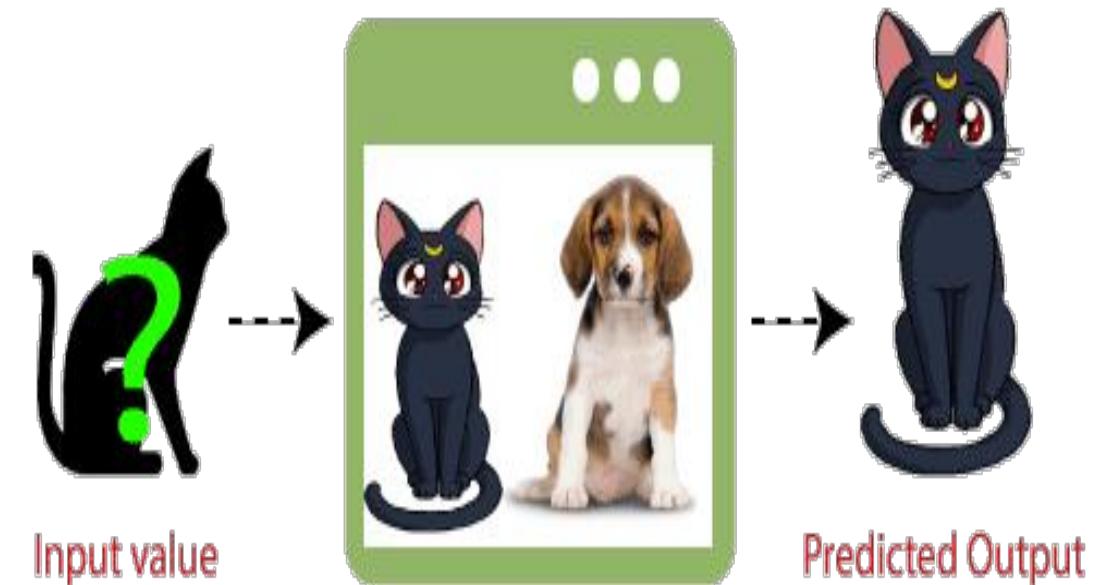
- Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog.
- So for this identification, we can use the KNN algorithm, as it works on a similarity measure.
- Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



# Using it for pattern recognition



KNN Classifier



# Using it for estimating the density functions

- There are several types of non-parametric methods of interest in pattern recognition.
- It consists of procedures for estimating the density functions  $p(x|y_q)$  from sample patterns (without the assumption that forms of underlying densities are known; and with the assumption that attributes are statistically independent).
- In probability theory, a probability density function is a function whose value at any given sample (or point) in the sample space (the set of possible values) can be interpreted as providing a relative likelihood that the value of the random variable would be close to that sample.

# Estimation of probability density function

- Assume that the sample  $\{x^{(i)}\}; i = 1, \dots, N$ , is drawn independently from some unknown probability density  $p(\cdot)$ .
  - $\hat{p}(\cdot)$  is our estimate of  $p(\cdot)$ .
  - We start with the univariate case where  $x(i)$  are scalars ( $n = 1$ ), and later generalize to the multidimensional case.
- For the non-parametric estimation, the oldest and the most popular method is the histogram, where the input space is divided into equal-sized intervals, named bins.
- The corresponding probability is approximated by the frequency ratio:

$$\hat{P}(x) = \frac{\# \{x^{(i)} \text{ in the same bin as } x\}}{N}$$

where  $\#\{\cdot\}$  denotes the number of training instances  $x^{(i)}$  that lie in the same bin as the test instance  $x$ .

# Estimation of probability density function

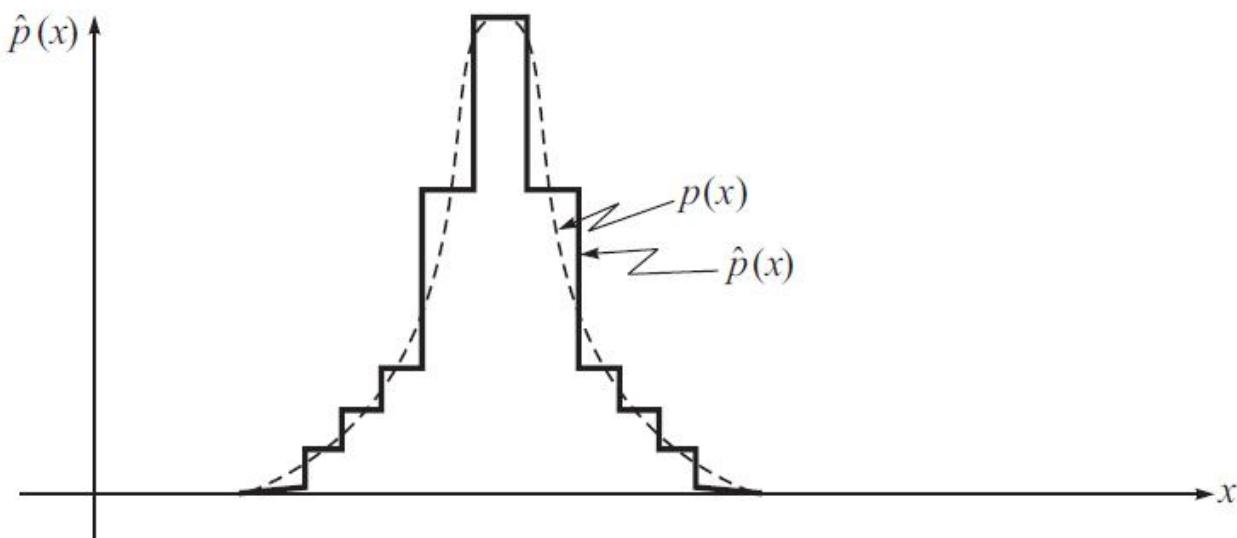
- This approximation converges to the true probability as N approaches infinity.
- The corresponding probability density function value is assumed constant throughout the bin and is approximated as,

$$\hat{p}(x) \equiv \hat{p}(\bar{x}) = \left[ \frac{\# \{x^{(i)} \text{ in the same bin as } x\}}{N} \right] \frac{1}{h}$$

- where  $\bar{x}$  is the midpoint of the bin

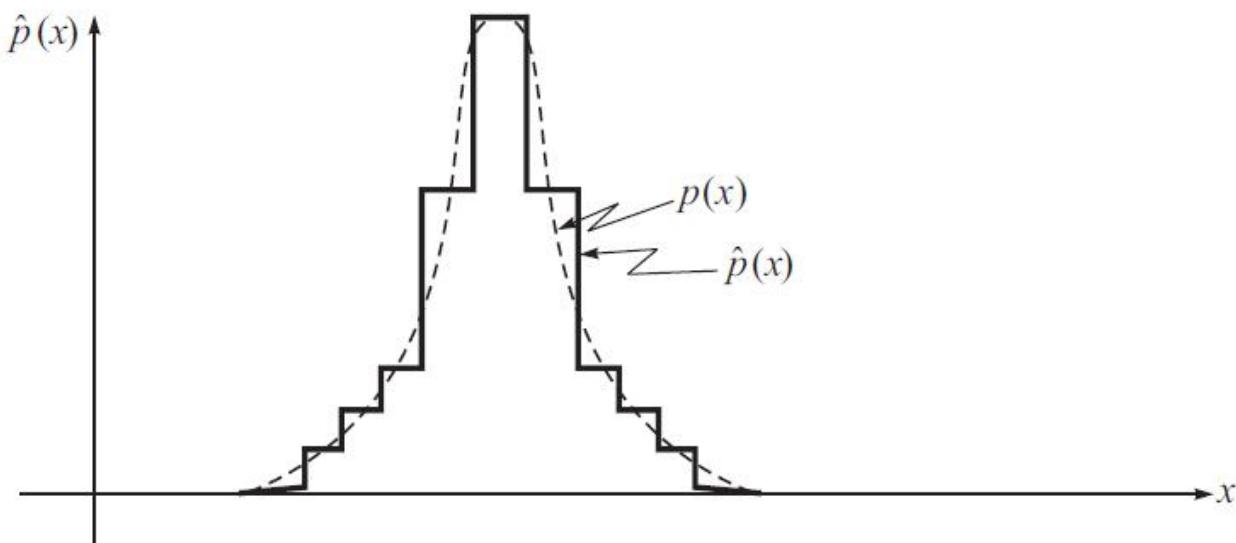
$$\left( |x - \bar{x}| \leq \frac{h}{2} \right)$$

- and h is the width of the bin.



# Estimation of probability density function

- In constructing the histogram, we have to choose both an origin and a bin width  $h$ .
- With small bins, the estimate is spiky, and with larger bins, the estimate is smoother.
- There are discontinuities at the boundaries.
- For a large enough number of samples, the smaller the  $h$ , the better the accuracy of the resulting estimate.
- Usually, a large  $N$  is necessary for acceptable performance.



# Estimation of probability density function

- The nearest-neighbor class of estimators adopts the reasonable amount of smoothing to the local density of the data.
- The degree of smoothing is controlled by  $k$  (the number of neighbors taken into account), which is much smaller than  $N$  (the sample size).
- For each  $x$  ( $n = 1$ ), we define  $d_1(x) \leq d_2(x) \leq \dots \leq d_N(x)$  distances from  $x$  to the points in the sample.
- The  $k$ -nearest neighbor (k-NN) density estimate is  $\hat{p}(x) = \frac{k}{2N d_k(x)}$
- instead of fixing  $h$  and checking how many samples fall in the bin, we fix  $k$ , the number of observations to fall in the bin, and compute the bin size.
- Where density is high, bins are small, and where density is low, bins are larger.

# Estimation of probability density function

- Generalizing to multivariable data,  $\hat{p}(\mathbf{x}) = \frac{k}{NV_k(\mathbf{x})}$
- where  $V_k(x)$  is the volume of the n-dimensional hyperspace centered at  $x$  with radius  $r = ||x - x(k)||$ ;  $x(k)$  is the  $k^{\text{th}}$  nearest observation to  $x$  among the neighbors.
- When used for classification, we need the class-conditional densities,  $p(x|y_q)$  of the feature vector distributions.
- The estimator of the class-conditional density is given as,  $\hat{p}(\mathbf{x}|y_q) = \frac{k_q}{N_q V_k(\mathbf{x})}$
- where  $k_q$  is the number of neighbors out of  $k$  nearest that belong to class  $y_q$ ; and  $N_q$  is the number of labeled instances belonging to class  $y_q$ .

# Estimating a posteriori probabilities using KNN

A reasonable estimate for  $P(y_q|\mathbf{x})$  is

( $y_q$  is the class for sample  $x$ )

$$\hat{P}(y_q|\mathbf{x}) = \frac{\hat{p}(\mathbf{x}|y_q) \hat{P}(y_q)}{\hat{p}(\mathbf{x})}$$

$$\hat{p}(\mathbf{x}|y_q) = \frac{k_q}{N_q V_k(\mathbf{x})}$$

$$\hat{P}(y_q) = N_q / N$$

$$\hat{p}(\mathbf{x}) = \frac{k}{N V_k(\mathbf{x})}$$

Therefore,

$$\begin{aligned}\hat{P}(y_q|\mathbf{x}) &= \left( \frac{k_q}{N_q V_k(\mathbf{x})} \times \frac{N_q}{N} \right) \Bigg/ \left( \frac{k}{N V_k(\mathbf{x})} \right) \\ &= k_q / k\end{aligned}$$

Thus, the estimate of the a posteriori probability that  $y_q$  is the class for sample  $x$  is just the fraction of the samples within the cell  $V_k(\mathbf{x})$  that are labeled  $y_q$ .

# KNN Classification

# Computer Vision: Car detection



Cars



Not a car

Testing:



What is this?

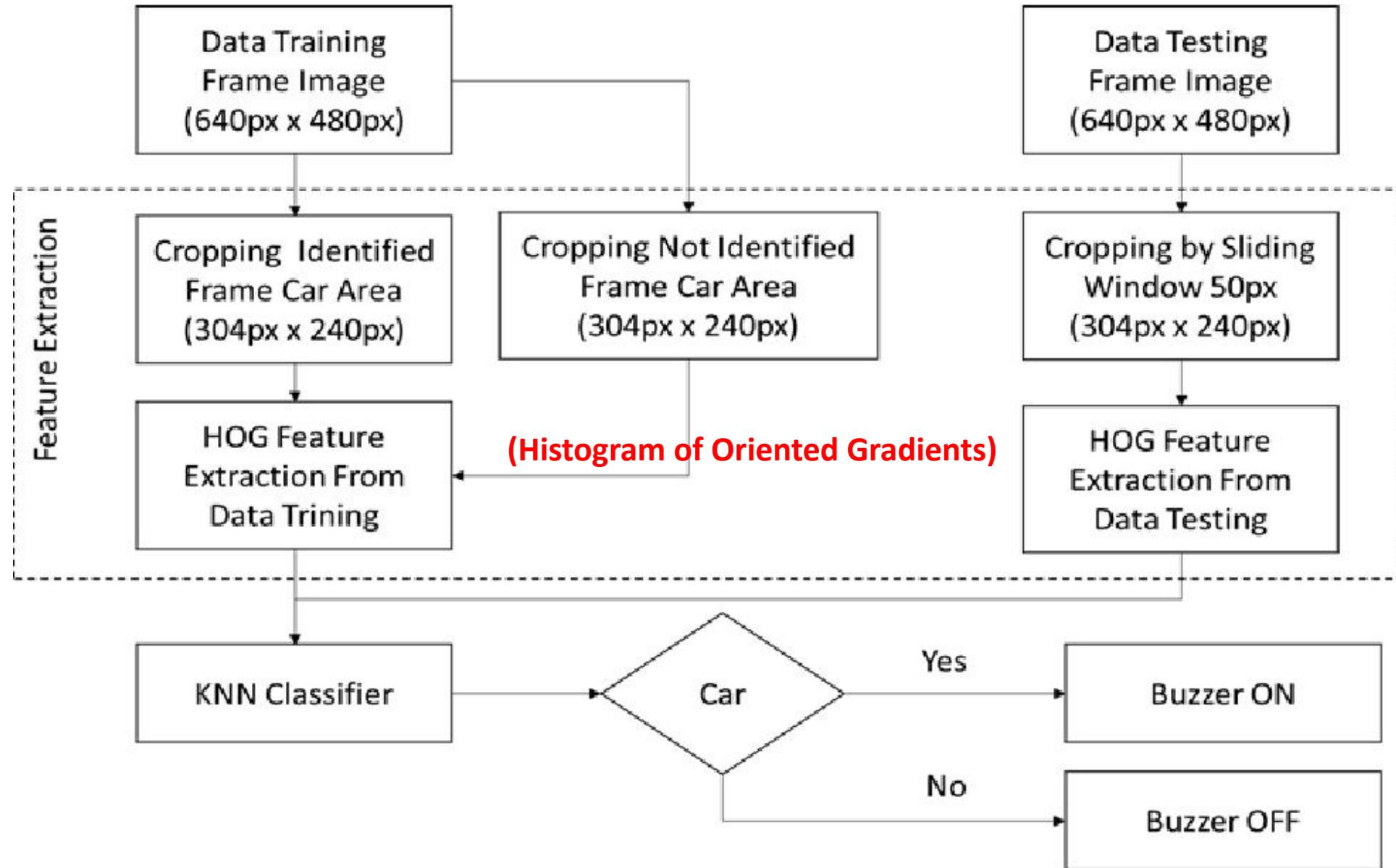
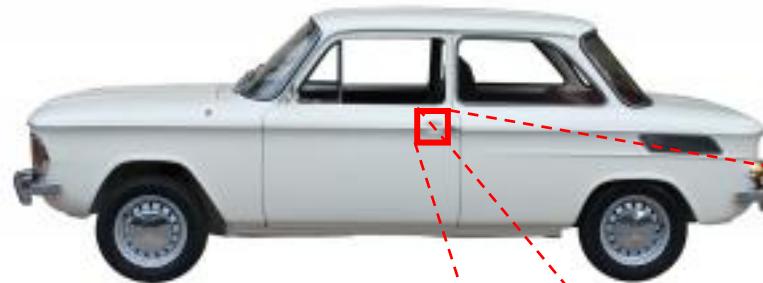


Figure 1 Block System

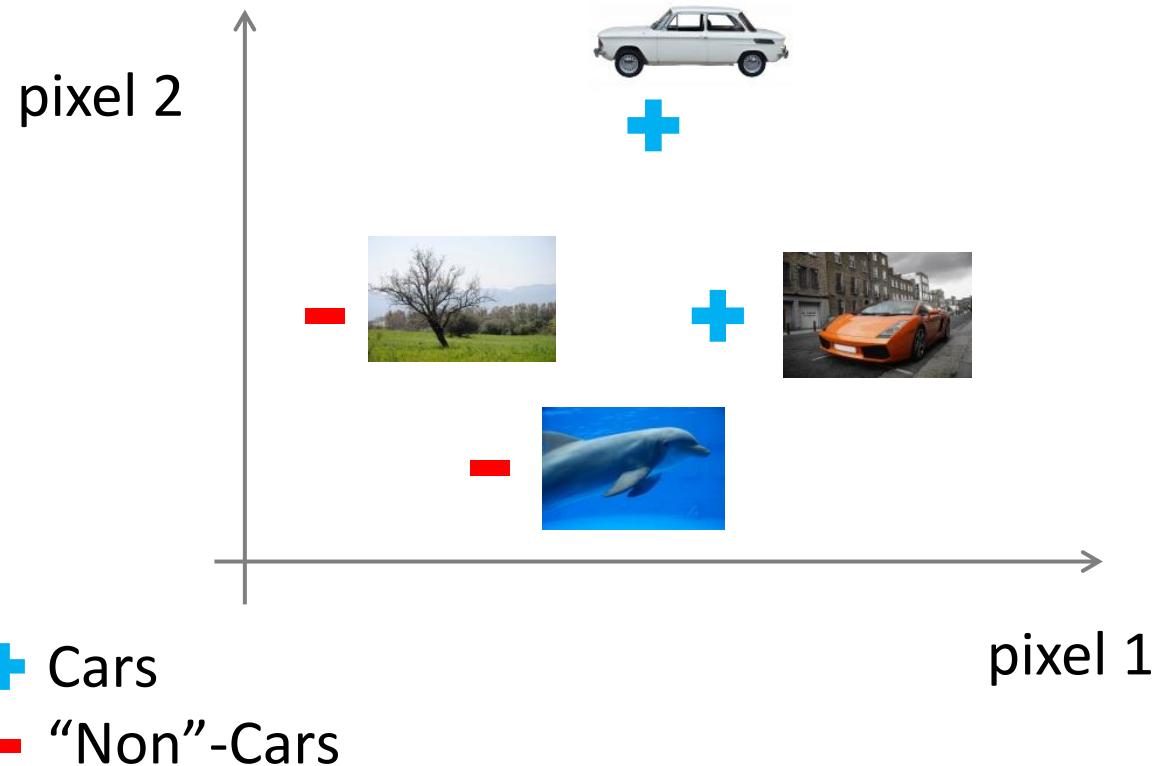
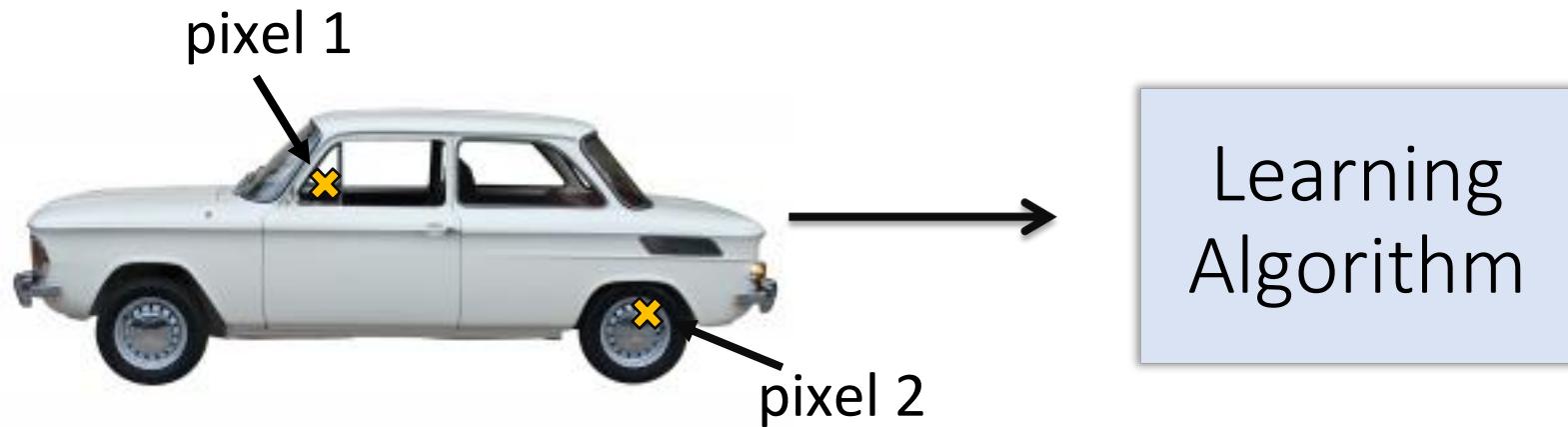
# What is this?

You see this:



But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50





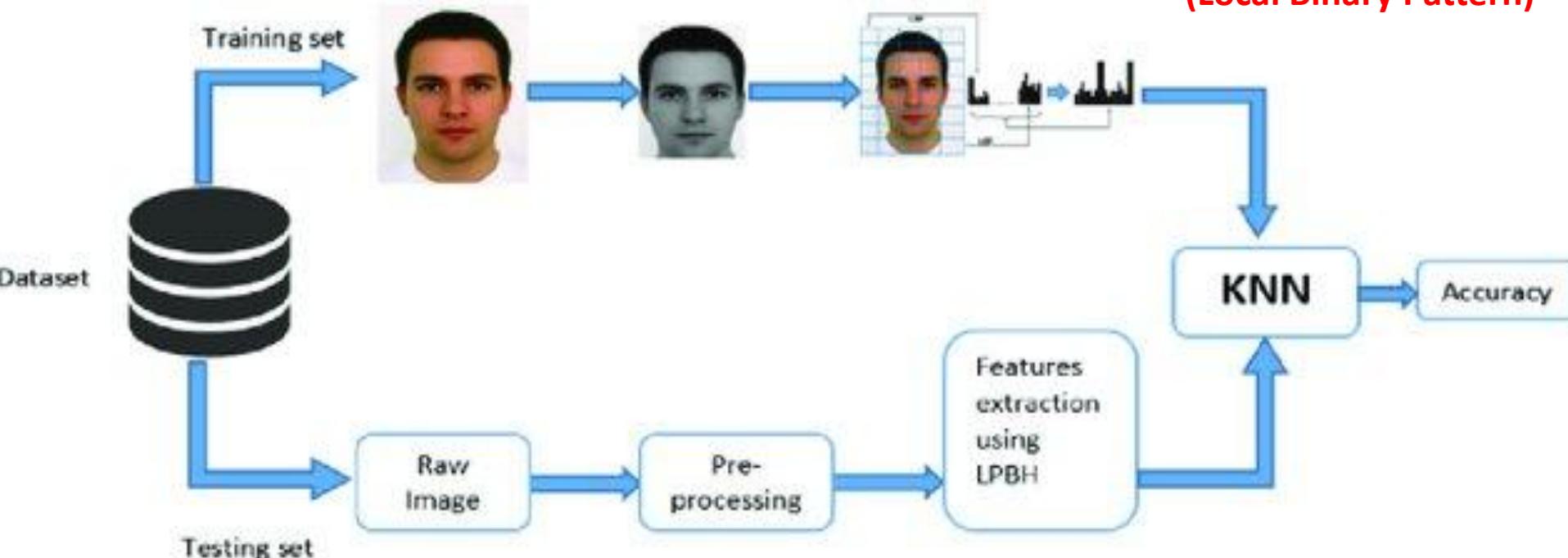
Face image



The face image is divided into blocks

LBP histogram from each block  
(Local Binary Pattern)

Feature histogram



To find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram (based on distance).

# How does it work?

- We first determine the  $k$  points that are closest neighbors of  $x$  with the help of a specific distance metric.
- The categorization of  $x$  is then given by the class label found in most of the  $k$  neighbors.
- All neighbors have equal vote, and the class with the most number of votes among the  $k$  neighbors is selected.
- Ties are randomly broken or weighted vote is taken.
- $k$  is usually taken an odd number to reduce ties: confusion exists usually between two neighboring classes.
- Note that  $k = 1$  is usually not sufficient for determining the class of  $x$  due to noise and outliers in the data.
- A set of nearest neighbors is needed to accurately decide the class.

# Distance function?

- The key issue of k-NN algorithm is the distance/similarity function, which is selected on the basis of applications and nature of the data.
- The cautious selection of an appropriate distance function is a crucial step in the use of k-NN.
- A validation set can be made use of to pick the appropriate distance function for a given dataset by applying all candidates to gauge which one gives better results.

# Value of k?

- Another question is the number of neighbors to select.
- Again, investigation of varying numbers of neighbors with the help of the validation set can facilitate establishing the optimal number, as the number is dependent on the data distribution and relies heavily on the problem being solved.
- Selecting the appropriate training set is probably the most important step in k-NN process.
- The training set should include enough examples of all probable categories, that is, it should be a balanced training set having more or less the same number of instances for all classes.
- Generally speaking, the size of the training set should have at least thousands, if not hundreds of thousands or millions, of training examples.

# Value of k?

- The data is partitioned into training, validation, and test sets.
- Validation set is used to compare error rates for various values of k/various similarity measures, and test set is used to evaluate the performance.
- In the k-NN procedure, all neighbors have equal vote.
- In an alternative method: the Parzon-window approach, also called Probabilistic Neural Network (PNN), the weight of the vote is given by a kernel function, typically giving more weight to closer instances.
- Low k-value is sensitive to outliers and a higher K-value is more resilient to outliers as it considers more voters to decide prediction.

# Value of k?

- Cross-validation is a smart way to find out the optimal K value.
- It estimates the validation error rate by holding out a subset of the training set from the model building process.
- Cross-validation (let's say 10 fold validation) involves randomly dividing the training set into 10 groups, or folds, of approximately equal size.
- 90% data is used to train the model and remaining 10% to validate it.
- The misclassification rate is then computed on the 10% validation data.
- This procedure repeats 10 times.
- Different group of observations are treated as a validation set each of the 10 times.
- It results to 10 estimates of the validation error which are then averaged out.

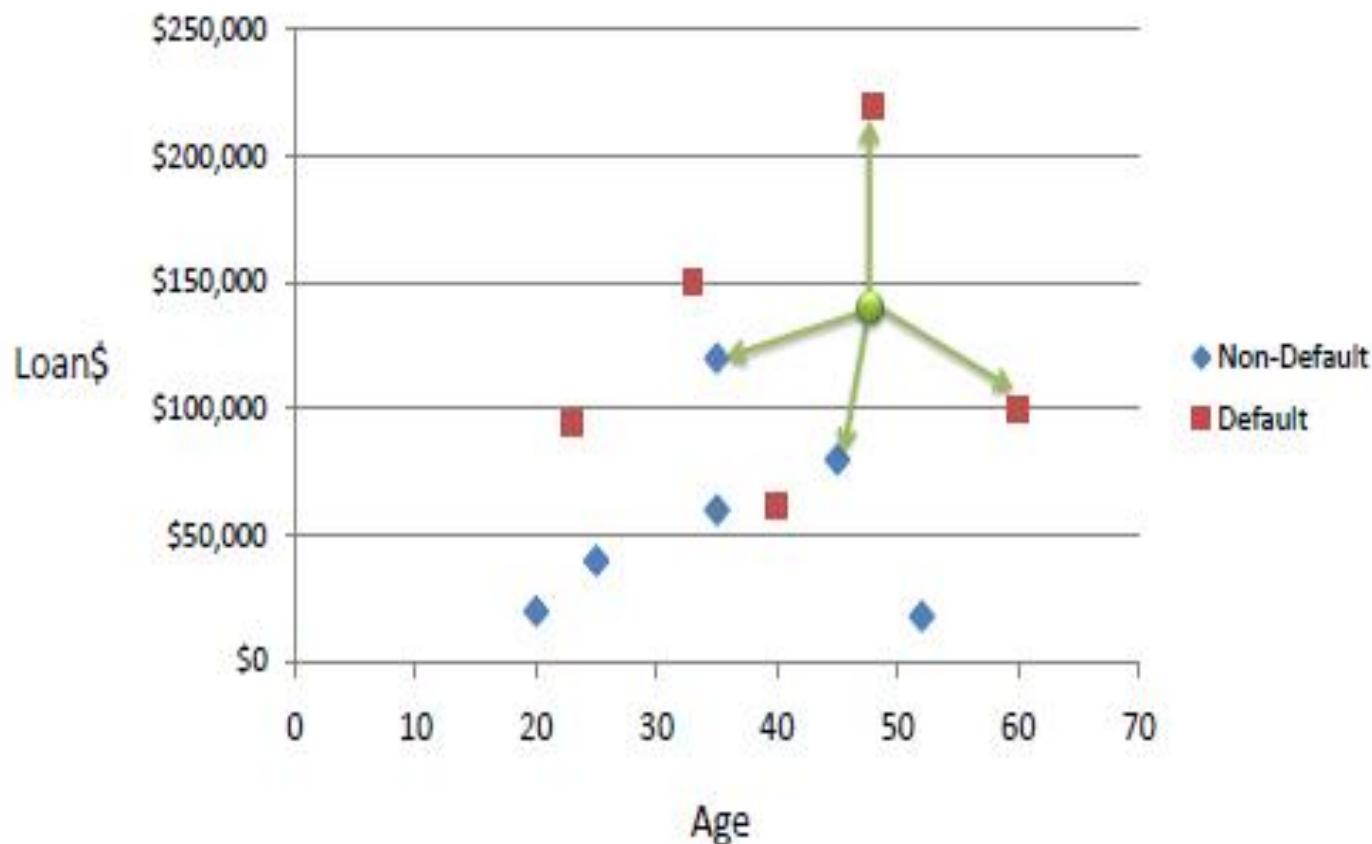
# Pros and Cons of KNN

Pros	Cons
Easy to understand	Memory Intensive / Computationally expensive
No assumptions about data	Sensitive to scale of data
Can be applied to both classification and regression	Not work well on rare event (skewed) target variable
Works easily on multi-class problems	Struggle when high number of independent variables

# Numerical Discussion

## Example 1:

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



# Example 1:

- 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.
- $D = \text{Sqrt}[(48-33)^2 + (142000-150000)^2] = 8000.01 >> \text{Default}=Y$
- 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.

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
52	\$18,000	N	124000
23	\$95,000	Y	47000
40	\$62,000	Y	80000
60	\$100,000	Y	42000
48	\$220,000	Y	78000
33	\$150,000	Y	8000
48	\$142,000	?	

Euclidean Distance

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

## Example 1:

- 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 as shown here.

### Standardized Distance

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

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

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

## Example 2:

- Suppose we have height, weight and T-shirt size of some customers and we need to predict the T-shirt size of a new customer given only height and weight information we have.
- Data including height, weight and T-shirt size information is shown here
- **Step 1 : Calculate Similarity based on distance function**
- There are many distance functions but Euclidean is the most commonly used measure.
- It is mainly used when data is continuous. Manhattan distance is also very common for continuous variables.

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L
168	62	L
168	63	L
168	66	L
170	63	L
170	64	L
170	68	L

## Example 2:

- The idea to use distance measure is to find the distance (similarity) between new sample and training cases and then finds the k-closest customers to new customer in terms of height and weight.
- New customer named 'Monica' has height 161cm and weight 61kg.
- Euclidean distance between first observation and new observation (monica) is as follows -  $d = \sqrt{(161-158)^2 + (61-58)^2}$
- Similarly, we will calculate distance of all the training cases with new case and calculates the rank in terms of distance.
- The smallest distance value will be ranked 1 and considered as nearest neighbor.

Euclidean :

$$d(x, y) = \sqrt{\sum_{i=1}^m (x_i - y_i)^2}$$

Manhattan / city - block :

$$d(x, y) = \sum_{i=1}^m |x_i - y_i|$$

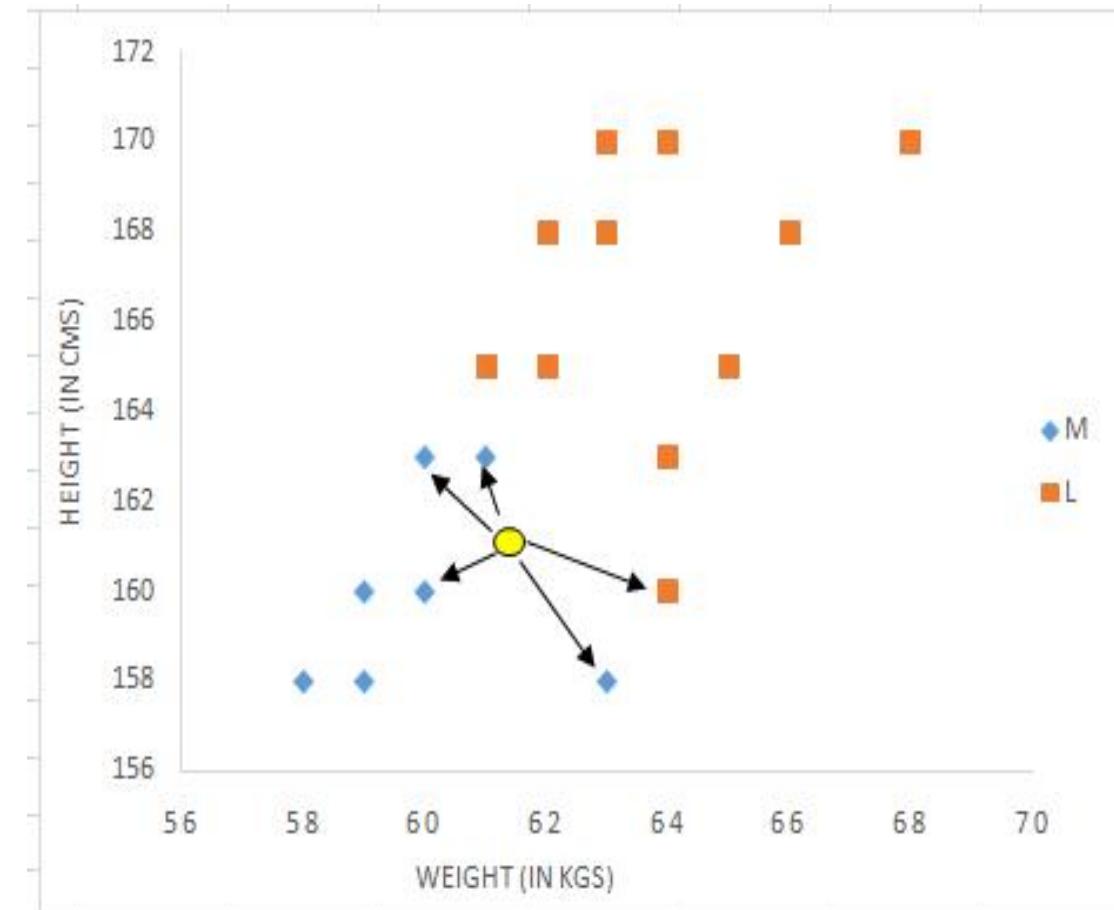
## Example 2:

- **Step 2 : Find K-Nearest Neighbors**
- Let k be 5.
- Then the algorithm searches for the 5 customers closest to Monica, i.e. most similar to Monica in terms of attributes, and see what categories those 5 customers were in.
- If 4 of them had ‘Medium T shirt sizes’ and 1 had ‘Large T shirt size’ then your best guess for Monica is ‘Medium T shirt’.
- See the calculation shown in the figure.

	A	B	C	D	E
1	Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
2	158	58	M	4.2	
3	158	59	M	3.6	
4	158	63	M	3.6	
5	160	59	M	2.2	3
6	160	60	M	1.4	1
7	163	60	M	2.2	3
8	163	61	M	2.0	2
9	160	64	L	3.2	5
10	163	64	L	3.6	
11	165	61	L	4.0	
12	165	62	L	4.1	
13	165	65	L	5.7	
14	168	62	L	7.1	
15	168	63	L	7.3	
16	168	66	L	8.6	
17	170	63	L	9.2	
18	170	64	L	9.5	
19	170	68	L	11.4	
20					
21	161	61			

## Example 2:

- In the graph, binary dependent variable (T-shirt size) is displayed in blue and orange color.
- 'Medium T-shirt size' is in blue color and 'Large T-shirt size' in orange color.
- New customer information is exhibited in yellow circle.
- Four blue highlighted data points and one orange highlighted data point are close to yellow circle.
- So the prediction for the new case is blue highlighted data point which is Medium T-shirt size.



## Example 2:

- **Standardization**
- In order to make the values comparable we need to standardize them which can be done by any of the following methods :

$$x_s = \frac{X - \text{mean}}{\text{s. d.}}$$

$$x_s = \frac{X - \text{mean}}{\text{max} - \text{min}}$$

$$x_s = \frac{X - \text{min}}{\text{max} - \text{min}}$$

- After standardization, 5th closest value got changed as height was dominating earlier before standardization. Hence, it is important to standardize predictors before running K-nearest neighbor algorithm.

A	B	C	D	E
Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
1				
2	-1.39	-1.64	M	1.3
3	-1.39	-1.27	M	1.0
4	-1.39	0.25	M	1.0
5	-0.92	-1.27	M	0.8 4
6	-0.92	-0.89	M	0.4 1
7	-0.23	-0.89	M	0.6 3
8	-0.23	-0.51	M	0.5 2
9	-0.92	0.63	L	1.2
10	-0.23	0.63	L	1.2
11	0.23	-0.51	L	0.9 5
12	0.23	-0.13	L	1.0
13	0.23	1.01	L	1.8
14	0.92	-0.13	L	1.7
15	0.92	0.25	L	1.8
16	0.92	1.39	L	2.5
17	1.39	0.25	L	2.2
18	1.39	0.63	L	2.4
19	1.39	2.15	L	3.4
20				
21	-0.7	-0.5		

# CLUSTERING

Dr. Srikanth Allamsetty

# Introduction to Unsupervised Learning, Distance Metrics used

# Unsupervised Learning

- It is a form of machine learning tasks where output  $y^{(i)}$  is not available in training data.
- In this type of problem, we are given a set of feature vectors  $x^{(i)}$ , and the goal is to unravel the underlying similarities.
- Unsupervised learning is more about creative endeavours—exploration, understanding and refinements that do not lend themselves to a specific set of steps, i.e, there is no specific methodology.
- There is no right or wrong answer; no simple statistical measure that summarizes the goodness of results.
- Instead, descriptive statistics and visualization are key parts of the process.

# Unsupervised Learning

- The requirement of dataset partitioning—into training, validation and test sets—is of little importance.
- There are two types:
  - **Cluster Analysis**
  - **Association Rules**
- Use the labeled data to train a classifier.
- The next step is to apply the same to the unlabeled data so that it is labeled with class probabilities.
- The third step is to train another classifier with the help of labels for all the data.
- Fourth, repeat till convergence (the termination of the optimization algorithm) is achieved.

# Cluster Analysis

- Cluster analysis is employed to create groups or clusters of similar records on the basis of many measurements made for these records.
- A primary issue in clustering is that of defining ‘similarity’ between feature vectors  $x^{(i)}$ ;  $i = 1, 2, \dots, N$ , representing the records.
- Another important issue is the selection of an algorithmic scheme that will cluster (group) the vectors based on the accepted similarity measure.
- The level of similarity and dissimilarity are evaluated on the basis of the characteristics of the variables that describe the objects or components.
- This assessment often involves distance measures.

Already discussed in previous class



**Euclidean distance**  
**Statistical distance**  
**Manhattan distance**  
**Minkowski metric**  
**Hamming distance**

# Cluster Analysis

- Clustering can be used for data exploration, to understand the structure of the data.
- A natural way to make sense of complex data is to break the data into smaller clusters of data; then finding patterns within each cluster is often possible.
- Another use of clustering methods is in outlier detection which is finding instances that do not lie in any of the main clusters and are exceptions.
- The outliers may be recording errors that should be detected and discarded in data cleansing process.
- While selecting the right clustering algorithm, we should make use of the knowledge of the problem the dataset describes.

# Cluster Analysis

- Data partition must usually have two features:
  - Clusters should be homogeneous within: Data within each cluster should strongly resemble each other.
  - Compactness is indicated by the variance of patterns in a cluster.
    - There should be heterogeneity between clusters: Data should differ from one cluster to another cluster as much as possible.
  - The Euclidean distance between cluster centroids indicates the cluster separation.

# Association Rules

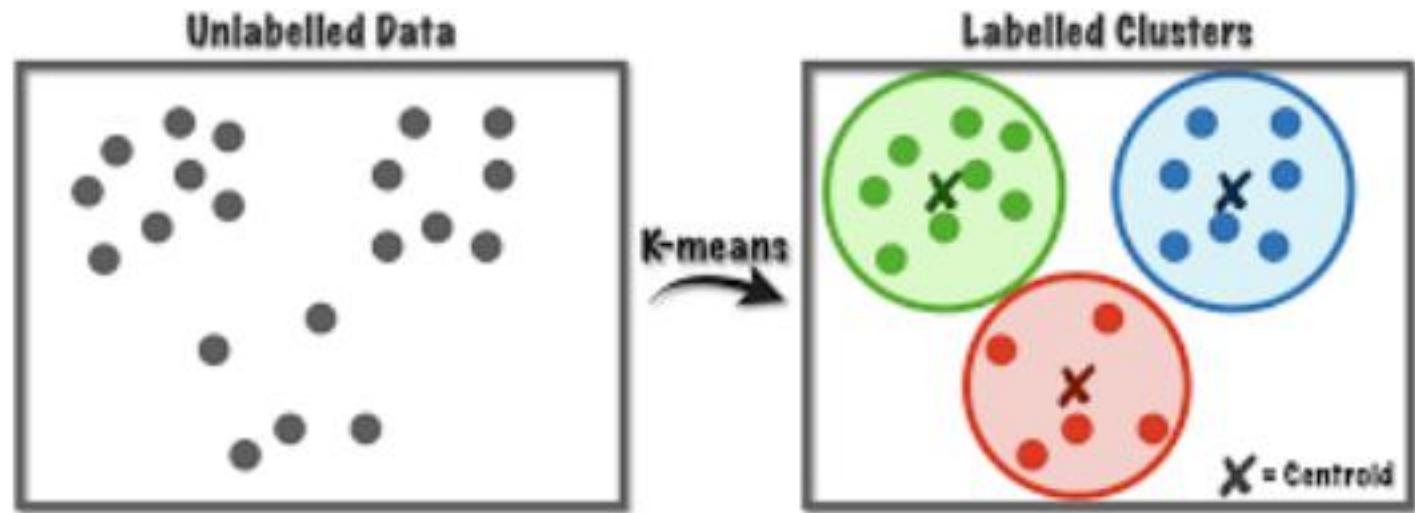
- Frequent patterns detection and generating association rules is another application of unsupervised learning.
- Mining frequent patterns is originated with the study of customer transactions databases to determine association between purchases of different items/service offerings.
- This popular area of application is called market basket analysis, which studies customers' buying habits for products that are purchased together.
- This application is commonly encountered in online recommender systems where customers examining product(s) for possible purchase are shown other products that are frequently purchased in conjunction with the desired product(s); display from Amazon.com, for example.

# Association Rules

- Other than market basket data, association analysis can also be applied to web mining, medical diagnosis, text mining, scientific data analysis, and other application domains.
- A medical researcher wishes to find out about the symptoms that match the confirmed diagnosis.
- While analysing Earth science data, the association patterns often disclose interesting links among the ocean, land and atmospheric pressures.
- The association approach, in terms of text documents, can be used to discover word co-occurrence relationships (used to find linguistic patterns).
- Association analysis can be used to discover web usage patterns.

# K-Means Approach for Clustering

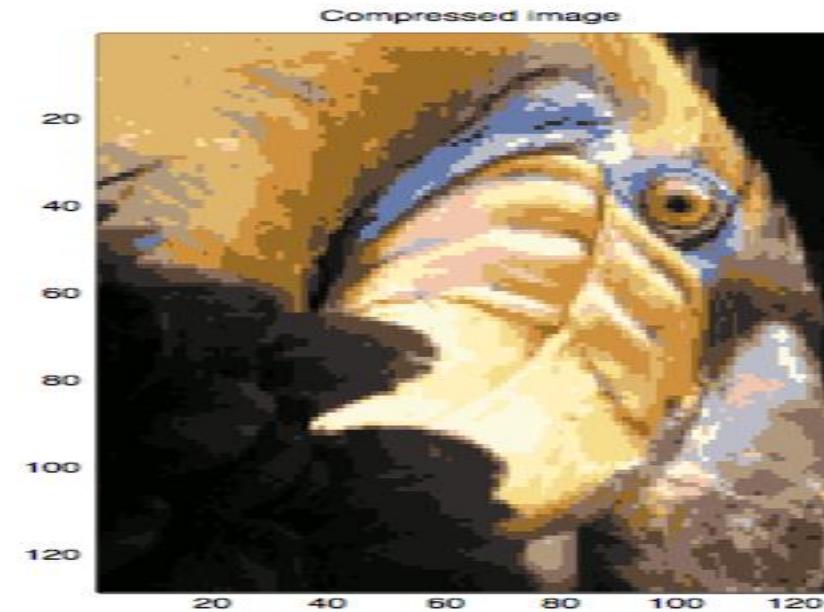
# Basics



- The most general of the heuristic clustering techniques is the K-means clustering.
- It is amongst the widely used clustering algorithms.
- K-means clustering characteristically espouses exclusive cluster separation:
  - The set of all clusters comprises all data vectors.
  - Each object belongs to exactly one group.
    - None of the clusters is empty and none of them contain the entire dataset X. The clusters are not joined.

# Image compression using **clustering**

- In an image, if we decide to color code shades of the same group with a single color, say their average, then we are actually quantizing the image.
- If 24 bit pixels represent 16 million colors for an image, and there are shades of merely 64 main colors, we will require 6 bits for each pixel rather than 24.



# Image compression using **clustering**

- K-means can be applied on the pixel values to get the resultant compressed image.
- Now, these ‘k’ cluster centroids will replace all the color vectors in their respective clusters.
- we have only reduced the number of colors to represent a colored digital image known as Color Quantization.
- However, there can be several different ways to achieve this target.
- Few other methods can be reducing the size of the image or reducing the intensity ranges of pixels or reducing the frequency of an image.

# How does it work

- The goal of clustering algorithm is to find K points that make good cluster centers.
- These centers define the clusters.
- Each object is assigned to the cluster defined by the nearest cluster center.
- The best assignment of cluster centers could be defined as the one that minimizes the sum of distances (or the distance-squared) from every point to its nearest cluster center.
- K points are randomly selected as cluster centers, which gives us cluster seeds.

# Cluster Interpretability

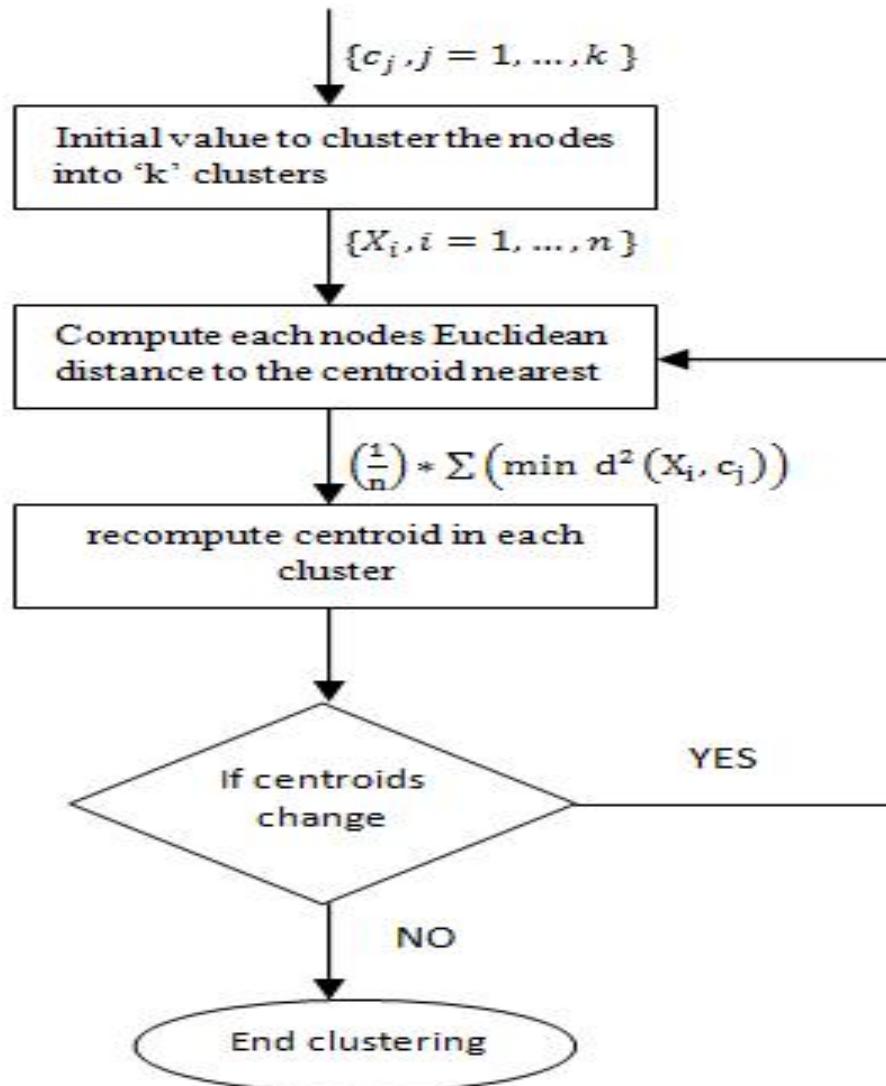
- The characteristics of each cluster to be able to understand the clusters by:
  - first getting the summary statistics from each cluster;
  - then, establishing if the dataset provided has a nonrandom structure, which may result in meaningful clusters;
  - and finally attempting to allocate a name or label to each cluster.
- The most simple and commonly used criterion function for clustering is the sum-of-squared-error criterion.
- Let  $N_k$  be the no. of samples in cluster  $k$  and  $\mu_k$  be the mean of those samples:

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=1}^{N_k} \mathbf{x}^{(i)}$$

Then the sum of squared errors is defined by,

$$J_e = \sum_{k=1}^K \sum_{i=1}^{N_k} \left\| \mathbf{x}^{(i)} - \boldsymbol{\mu}_k \right\|^2 \text{ where K stands for number of data partitionings.}$$

# Flow Chart



# Procedure

- All objects are allocated to their nearest cluster center as per the Euclidean distance metric.
- allocation of all objects to the nearest seed, all that is required is the calculation of the distance between each object and each seed.
- Then comes the calculation of the centroid or mean of the objects in each cluster, which is the ‘means’ part of the algorithm.
- These centroids are considered as new cluster-center values for their respective clusters.
- The entire procedure is iterated with the new cluster centers.
- Repetition goes on until the same points are assigned to each cluster in successive rounds.
- At this stage, the cluster centers become stable.

# Points to be understood

- Entirely different arrangements can be derived from small changes in primary random selection of the seeds, and some may be more beneficial than others.
- To increase the chance of discovering the solution that has the maximum benefit, the algorithm may require to be run many times with various seeds and then the best final result may be selected.
- Similar situation arises in the choice of K.
- Often, nothing is known about the likely number of clusters, and the whole point of clustering is to find it out.
- A heuristic way is to try different values and choose the best.
- Try normalizing/standardizing the data as it involves with distance measures.

# Performance Evaluation, and Stopping Criteria for K Means

# Form two clusters for given data.

- We need to form 2 clusters, So for that we consider two data points of our data (randomly or based on i value) and assign them as a centroid for each cluster.
- Now we need to assign each and every data point of our data to one of these clusters based on Euclidean distance calculation.

Euclidean Distance :  $\sqrt{(X_0 - X_c)^2 + (Y_0 - Y_c)^2}$

Height	Weight
185	72
170	56
168	60
179	68
182	72
188	77
180	71
180	70
183	84
180	88
180	67
177	76

# Form two clusters for given data.

- Lets consider the 2nd data point i.e. (170,56) and check its distance with the centroid of both clusters.

$$K_1 = \sqrt{(170-185)^2 + (56-72)^2}$$
$$= 21.93$$

$$K_2 = \sqrt{(170-180)^2 + (56-71)^2}$$
$$= 18.03$$

Height	Weight
185	72
170	56
168	60
179	68
182	72
188	77
180	71
180	70
183	84
180	88
180	67
177	76

- Now we can see from calculations that 3rd data point(168,60) is more closer to k2(cluster 2), so we assign it to k2.
- We shall continue this procedure for all data points.

# Form two clusters for given data.

Height	Weight	K1 (185,72)	K2 (180,71)	Closer Centroid		Height	Weight	k1_1 (184,80.25)	k2_1 (177,67.25)	Closer Centroid
185	72	0.00	5.10	K1	1st iteration	185	72	8.31	9.18	K1
170	56	21.93	18.03	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	188	77	5.15	14.53	K1
168	60	20.81	16.28	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	183	84	3.88	17.56	K1
179	68	7.21	3.16	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	180	88	8.72	20.72	K1
182	72	3.00	2.24	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	180	71	10.08	4.61	K2
188	77	5.83	10.00	K1	$K_{1-1} = \frac{185+188+183+180}{4}$	170	56	28.00	13.46	K2
180	71	5.10	0.00	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	168	60	25.81	11.72	K2
180	70	5.39	1.00	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	179	68	13.23	2.06	K2
183	84	12.17	13.34	K1	$K_{1-1} = \frac{185+188+183+180}{4}$	182	72	8.49	6.73	K2
180	88	16.76	17.00	K1	$K_{1-1} = \frac{185+188+183+180}{4}$	180	70	11.00	3.91	K2
180	67	7.07	4.00	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	180	67	13.84	3.04	K2
177	76	8.94	5.83	K2	$K_{1-1} = \frac{185+188+183+180}{4}$	177	76	8.19	8.50	K1

[https://docs.google.com/spreadsheets/d/1xOFIBp\\_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true](https://docs.google.com/spreadsheets/d/1xOFIBp_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true)

# Form two clusters for given data.

Height	Weight	k1_1 (184,80.25)	k2_1 (177,67.25)	Closer Centroid		Height	Weight	k1_2 (182.6,79.4)	k2_2 (177,66.29)	Closer Centroid
185	72	8.31	9.18	K1	2nd iteration	185	72	7.78	9.83	K1
188	77	5.15	14.53	K1	$K_{1\_2} = \frac{185+188+183+180+177}{5}$	188	77	5.91	15.36	K1
183	84	3.88	17.56	K1	$K_{1\_2} = \frac{182.6}{5}$	183	84	4.62	18.70	K1
180	88	8.72	20.72	K1	$K_{1\_2} = \frac{72+77+84+88+76}{5}$	180	88	8.98	21.92	K1
					$(weight) = 79.4$	177	76	6.55	9.71	K1
180	71	10.08	4.61	K2	$K_{2\_2} = \frac{180+170+168+....+180}{7}$	180	71	8.79	5.59	K2
170	56	28.00	13.46	K2	$K_{2\_2} = \frac{177}{7}$	170	56	26.58	12.44	K2
168	60	25.81	11.72	K2	$K_{2\_2} = 177$	168	60	24.28	10.98	K2
179	68	13.23	2.06	K2	$K_{2\_2} = \frac{71+56+60+....+67}{7}$	179	68	11.95	2.63	K2
182	72	8.49	6.73	K2	$K_{2\_2} = \frac{66.29}{7}$	182	72	7.42	7.59	K1
180	70	11.00	3.91	K2		180	70	9.75	4.77	K2
180	67	13.84	3.04	K2		180	67	12.67	3.08	K2
177	76	8.19	8.50	K1						

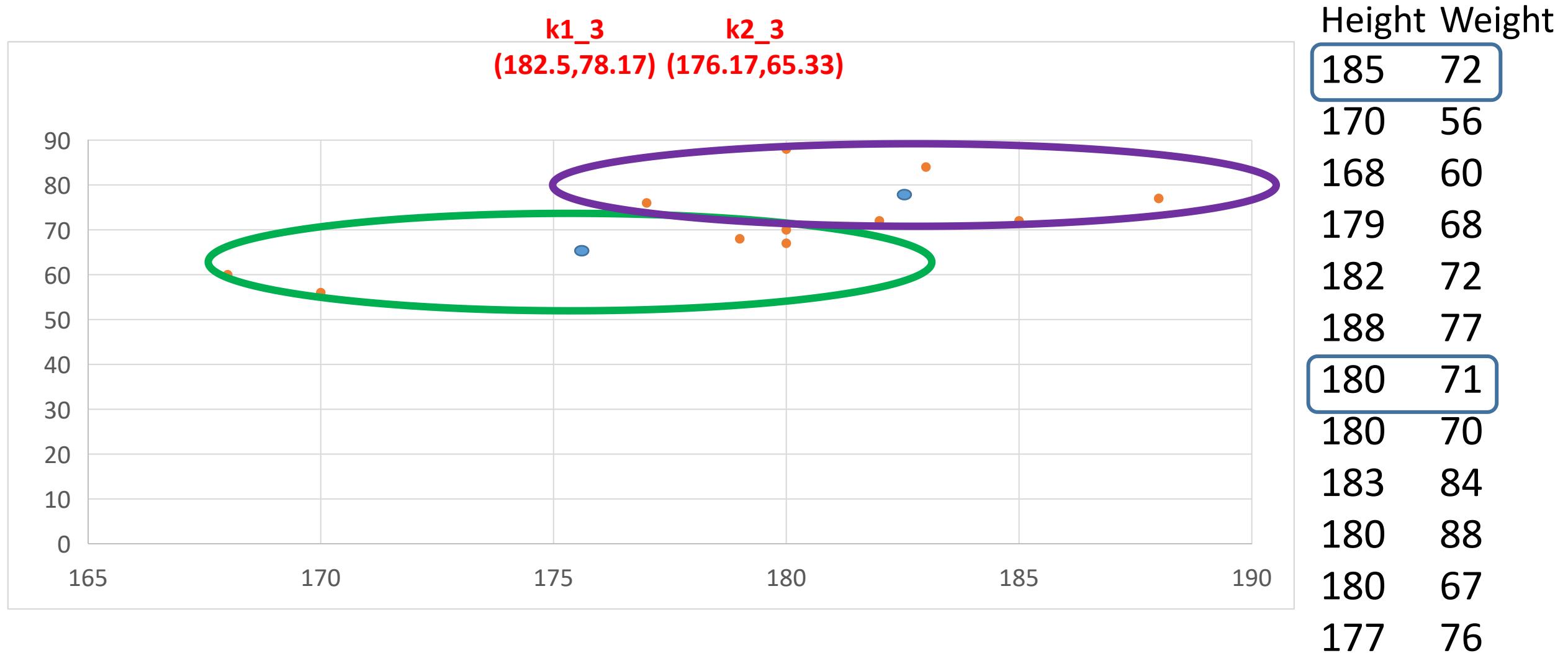
[https://docs.google.com/spreadsheets/d/1xOFIBp\\_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true](https://docs.google.com/spreadsheets/d/1xOFIBp_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true)

# Form two clusters for given data.

Height	Weight	k1_2 (182.6,79.4)	k2_2 (177,66.29)	Closer Centroid		Height	Weight	k1_3 (182.5,78.17)	k2_3 (176.17,65.33)	Closer Centroid
185	72	7.78	9.83	K1	3rd iteration	185	72	6.65	11.07	K1
188	77	5.91	15.36	K1	$K_{1-3} = \frac{185+188+183+\dots+182}{6}$	188	77	5.62	16.62	K1
183	84	4.62	18.70	K1	$K_{1-3} = 182.5$	183	84	5.85	19.88	K1
180	88	8.98	21.92	K1	$K_{1-3} = \frac{72+77+84+\dots+72}{6}$	180	88	10.15	22.99	K1
177	76	6.55	9.71	k1 (weight)	$K_{1-3} = \frac{78.17}{6}$	177	76	5.91	10.70	K1
						182	72	6.19	8.86	K1
180	71	8.79	5.59	K2	$K_{2-3} = \frac{180+170+169+\dots+180}{6}$	180	71	7.59	6.84	K2
170	56	26.58	12.44	K2	$K_{2-3} = 176.17$	180	71	7.59	6.84	K2
168	60	24.28	10.98	K2	$K_{2-3} = \frac{71+56+60+\dots+67}{6}$	170	56	25.45	11.19	K2
179	68	11.95	2.63	K2	$K_{2-3} = \frac{71+56+60+\dots+67}{6}$	168	60	23.24	9.75	K2
182	72	7.42	7.59	K1	$= 65.33$	179	68	10.75	3.89	K2
180	70	9.75	4.77	K2		180	70	8.54	6.04	K2
180	67	12.67	3.08	K2		180	67	11.44	4.18	K2

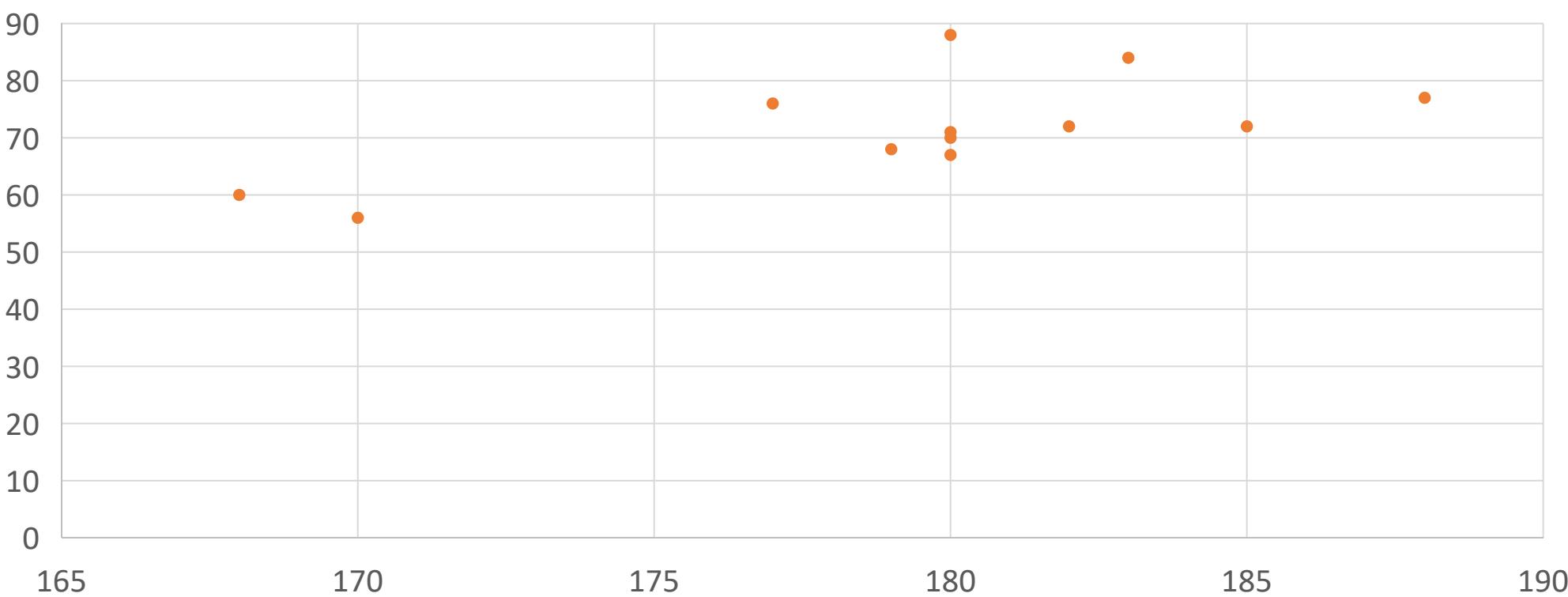
[https://docs.google.com/spreadsheets/d/1xOFIBp\\_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true](https://docs.google.com/spreadsheets/d/1xOFIBp_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true)

# How to know your result is correct or not?



# How to know your result is correct or not?

- Change the initial centroids; or chose them on the basis of their appearance in the graph.



Height	Weight
185	72
170	56
168	60
179	68
182	72
188	77
180	71
180	70
183	84
180	88
180	67
177	76

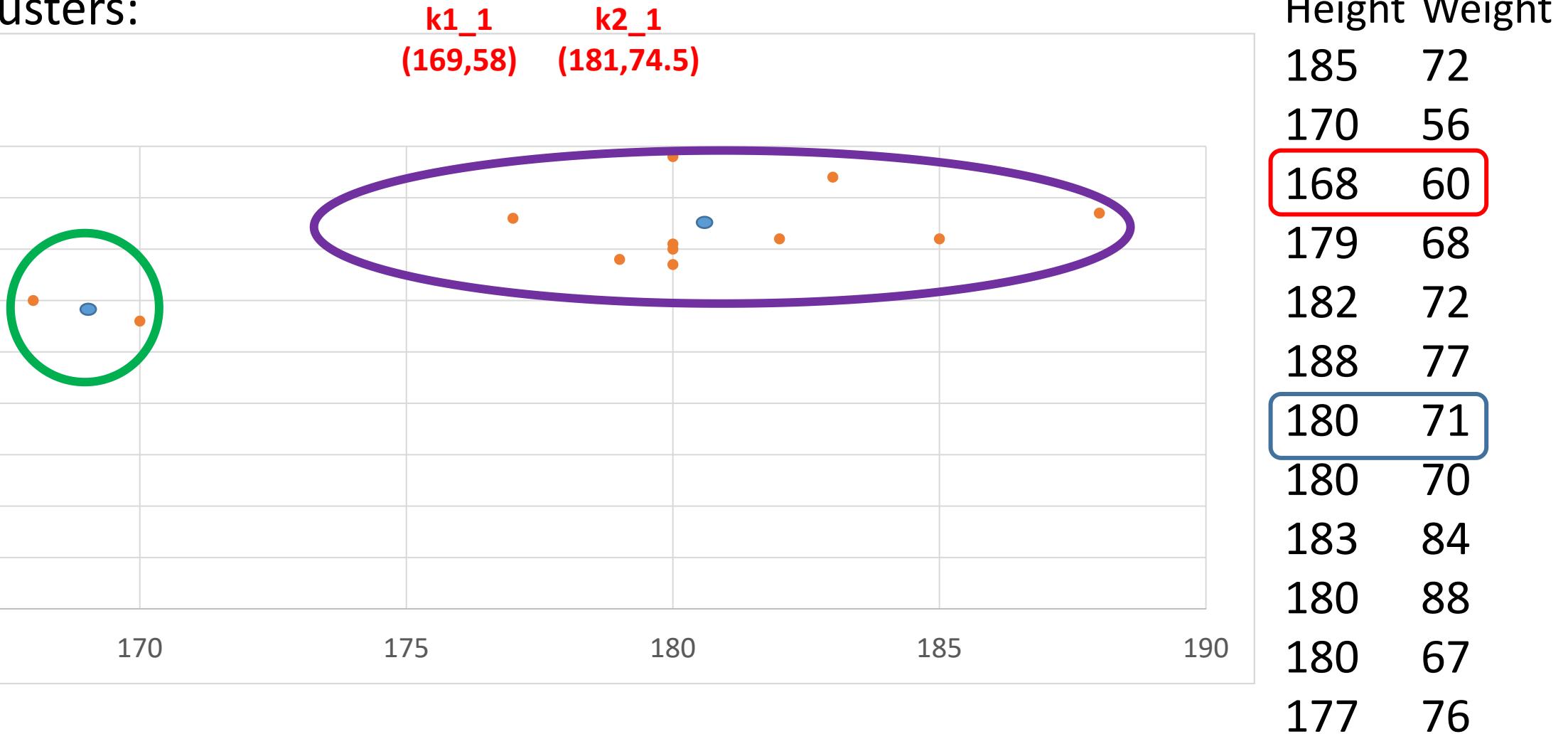
# Form two clusters for given data.

Height	Weight	K1 (168,60)	K2 (180,71)	Closer Centroid		Height	Weight	k1_1 (169,58)	k2_1 (181,74.5)	Closer Centroid
185	72	20.81	5.10	K2		170	56	2.24	21.73	K1
170	56	4.47	18.03	K1		168	60	2.24	19.74	K1
168	60	0.00	16.28	K1						
179	68	13.60	3.16	K2		185	72	21.26	4.38	K2
182	72	18.44	2.24	K2	<b>Stopping Criteria: No Change in centroids (happens when no exchange of data points takes place between clusters.)</b>	179	68	14.14	6.93	K2
188	77	26.25	10.00	K2		182	72	19.10	2.57	K2
		178.39				188	77	26.87	7.06	K2
180	71	16.28	0.00	K2		180	71	17.03	3.77	K2
180	70	15.62	1.00	K2		180	70	16.28	4.71	K2
183	84	28.30	13.34	K2		183	84	29.53	9.63	K2
180	88	30.46	17.00	K2		180	88	31.95	13.57	K2
180	67	13.89	4.00	K2		180	67	14.21	7.63	K2
177	76	18.36	5.83	K2		177	76	19.70	4.65	K2

[https://docs.google.com/spreadsheets/d/1xOFIBp\\_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true](https://docs.google.com/spreadsheets/d/1xOFIBp_sV-WnNclz6VvjNaw-Hr4sw8M1/edit?usp=sharing&ouid=118186460474919932244&rtpof=true&sd=true)

# How to know your result is correct or not?

- New Clusters:



# Points to be understood (once again)

- Entirely different arrangements can be derived from small changes in primary random selection of the seeds, and some may be more beneficial than others.
- To increase the chance of discovering the solution that has the maximum benefit, the algorithm may require to be run many times with various seeds and then the best final result may be selected.
- Similar situation arises in the choice of K.
- Often, nothing is known about the likely number of clusters, and the whole point of clustering is to find it out.
- A heuristic way is to try different values and choose the best.
- Try normalizing/standardizing the data as it involves with distance measures.

# GENERALISED LINEAR MODEL

Dr. Srikanth Allamsetty

# Limitation of Linear Model and Max Likelihood Learning

# Limitation of Linear Model

- Logistic regression is a way to adopt regression models for Modeling a binary outcome such as yes/no or 1/0.
- Using the linear regression model to predict  $y$  (binary outcome) yields predictions that are not necessarily 0 and 1.
- $y$  for task such as “what is the probability that an instance belongs to class q?” restricted between 0 and 1.
- Linear regression is inappropriate for such responses.
- For binary classification problem, it is assumed that

$$\log(\text{odds}) = \log \frac{P(\text{Class 1}|\mathbf{x})}{1 - P(\text{Class 1}|\mathbf{x})} = w_0 + w_1 x_1 + \dots + w_n x_n$$

- The odds; ratio of the proportions for the two possible outcomes; is a number between 0 and infinity.

# what is Logistic function?

$w_0, w_1, \dots, w_n$  are  $(n + 1)$  parameters of linear logit function.

Solving for probability  $P(\text{Class } 1|\mathbf{x})$  from Eqn (3.81) requires a bit of algebra.

$$P(\text{Class } 1|\mathbf{x}) = \frac{1}{1 + e^{-(w_0 + w_1x_1 + \dots + w_nx_n)}}$$

$$P(\text{Class } 2|\mathbf{x}) = 1 - P(\text{Class } 1|\mathbf{x})$$

- This is the logistic function that never gets smaller than 0 and never gets larger than 1, but takes on every value in between.
- maximum likelihood criterion is used for estimating parameters of logistic regression model.

# Parameters of logistic regression model

We are given a set of observed data:

$$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}; i = 1, \dots, N$$

$$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T \in \Re^n$$

is  $\{0, 1\}$

The proposed model, given by (3.82), is

$$P(y=1|\mathbf{x}) = \frac{1}{1 - e^{-(w_0 + w_1 x_1 + \dots + w_n x_n)}}$$

$$= \frac{1}{1 - e^{-(\mathbf{w}^T \mathbf{x} + w_0)}}$$

Where

$$\mathbf{w}^T = [w_1 \ w_2 \ \dots \ w_n]$$

$$P(y=0|\mathbf{x}) = 1 - P(y=1|\mathbf{x})$$

# Maximum Likelihood Estimation

- The maximum likelihood approach answers the question: for all possible values of the parameters  $\{w, w_0\}$  that the distribution might have, which of these are **most likely**.
- We call this '**most likely**' estimate, the maximum likelihood estimate, which is computed as follows:
  - For each of our observed data points, we compute a likelihood as a function of the parameters we seek to estimate.
    - This likelihood is just the value of the corresponding probability distribution evaluated at that particular point.
    - We then compute the likelihood function, which is the combination of likelihoods for all the data points we observed.
    - Thereafter, we estimate the set of parameters which will maximize the likelihood function.

# Maximum Likelihood function

$$\mathcal{L}(\{\mathbf{w}, w_0\}, \mathcal{D}) = \mathcal{L}(\bar{\mathbf{w}}, \mathcal{D}) = \prod_{i=1}^N P(y^{(i)}|\mathbf{x}^{(i)})$$

- In the maximum likelihood problem, our goal is to find  $w$  that maximizes  $\mathcal{L}$ . That is, we wish to find  $\bar{w}^*$  where

$$\bar{\mathbf{w}}^* = \arg \max_{\bar{\mathbf{w}}} \mathcal{L}(\bar{\mathbf{w}}, \mathcal{D})$$

- Often we maximize  $\log(\mathcal{L}(w, D))$  instead because it is analytically easier.
- Because the logarithm is monotonically increasing, the  $\bar{w}^*$  that maximizes log-likelihood, also maximizes the likelihood.

$$\begin{aligned}\log(\mathcal{L}(\bar{\mathbf{w}}|\mathcal{D})) &= \log\left(\prod_{i=1}^N P(y^{(i)}|\mathbf{x}^{(i)})\right) \\ &= \sum_{i=1}^N \log P(y^{(i)}|\mathbf{x}^{(i)})\end{aligned}$$

# logistic regression model in terms of discriminant fn g

Assuming  $y^{(i)}$ , given  $\mathbf{x}^{(i)}$ , is Bernoulli<sup>1</sup> with  $P(y^{(i)}|\mathbf{x}^{(i)}) = g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})$ ; we model log-likelihood as,

$$P\{Y=y\} = p^y(1-p)^{1-y}; y \in [0, 1]$$

$$\begin{aligned}\log(\mathcal{L}(\bar{\mathbf{w}}|\mathcal{D})) &= \log \left( \prod_{i=1}^N P(y^{(i)}|\mathbf{x}^{(i)})^{y^{(i)}} (1-P(y^{(i)}|\mathbf{x}^{(i)}))^{1-y^{(i)}} \right) \\ &= \sum_{i=1}^N \log(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})^{y^{(i)}} (1-g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}))^{1-y^{(i)}})\end{aligned}$$

To maximize log-likelihood with respect to  $\bar{\mathbf{w}}$ , let us look at each example:

$$\begin{aligned}\log(\mathcal{L}_i(\bar{\mathbf{w}}|\mathcal{D})) &= \log(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})^{y^{(i)}} (1-g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}))^{1-y^{(i)}}) \\ &= y^{(i)} \log g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}) + (1-y^{(i)}) \log(1-g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}))\end{aligned}$$

Taking derivative of  $\log(\mathcal{L}_i(\cdot))$  with respect to  $\bar{\mathbf{w}}$ , we have

$$\nabla_{\bar{\mathbf{w}}}(\log(\mathcal{L}_i(\cdot))) = \frac{y^{(i)}}{g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})} \nabla_{\bar{\mathbf{w}}}(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})) - \frac{1-y^{(i)}}{1-g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})} \nabla_{\bar{\mathbf{w}}}(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}))$$

# Derivative of $\log(\mathcal{L}_i(\cdot))$ with respect to $\bar{\mathbf{w}}$

$$\nabla_{\bar{\mathbf{w}}}(\log(\mathcal{L}_i(\cdot))) = \frac{y^{(i)}}{g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})} \nabla_{\bar{\mathbf{w}}}(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})) - \frac{1-y^{(i)}}{1-g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})} \nabla_{\bar{\mathbf{w}}}(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}))$$

Now

$$g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}) = \frac{1}{1 + e^{-a}}; a = (\bar{\mathbf{w}}^T \mathbf{x}^{(i)})$$

Therefore,

$$\begin{aligned}\nabla_{\bar{\mathbf{w}}}(g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})) &= \frac{\partial}{\partial a} \left( \frac{1}{1 + e^{-a}} \right) \frac{\partial a}{\partial \bar{\mathbf{w}}} \\ &= \left( \frac{1}{1 + e^{-a}} \right) \left( 1 - \frac{1}{1 + e^{-a}} \right) \bar{\mathbf{x}}^{(i)} \\ &= g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}) (1 - g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})) \bar{\mathbf{x}}^{(i)}\end{aligned}$$

This gives

$$\nabla_{\bar{\mathbf{w}}}(\log(\mathcal{L}_i(\bar{\mathbf{w}}|\mathcal{D}))) = (y^{(i)} - g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})) \mathbf{x}^{(i)}$$

# Gradient ascent (maximization problem)

- Considering all training examples, we get,

$$\nabla_{\bar{\mathbf{w}}} (\log(\mathcal{L}(\bar{\mathbf{w}}|\mathcal{D}))) = \sum_{i=1}^N (y^{(i)} - g(\mathbf{x}^{(i)}, \bar{\mathbf{w}})) \mathbf{x}^{(i)}$$

- The updated equation for gradient ascent (maximization problem):

$$\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \eta \sum_{i=1}^N (y^{(i)} - g(\bar{\mathbf{x}}^{(i)}, \bar{\mathbf{w}})) \mathbf{x}^{(i)}$$

- Once training is complete and we have the final  $\mathbf{w}$  and  $w_0$ , during testing, given  $\mathbf{x}^{(k)}$ , we calculate  $y^{(k)} = g(\mathbf{x}^{(k)}, \mathbf{w}, w_0)$
- we choose Class 1 if  $y^{(k)} > 0.5$  and choose Class 2 otherwise.
- Instead of 0.5, the cut-off value can be chosen based on overall accuracy computed for various values.

# Link function and its Role in Handling Non Normal Kind of Data Distribution

# Non-normal data Distribution

- Normal distribution is a probability distribution that is symmetric about the 'mean'.
- That means the data near the 'mean' are more frequent in occurrence than data far from the mean.
- In a normal distribution the mean is zero and the standard deviation is 1.

**Normal Distribution** is a distribution that has most of the data in the center with decreasing amounts evenly distributed to the left and the right.



## Non-normal Distributions

**Skewed Distribution** is distribution with data clumped up on one side or the other with decreasing amounts trailing off to the left or the right.

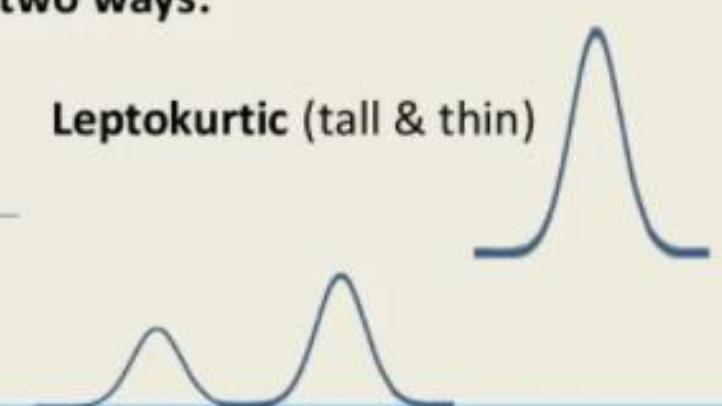


Distributions can have Kurtosis in two ways:

**Platykurtic** (very flat)

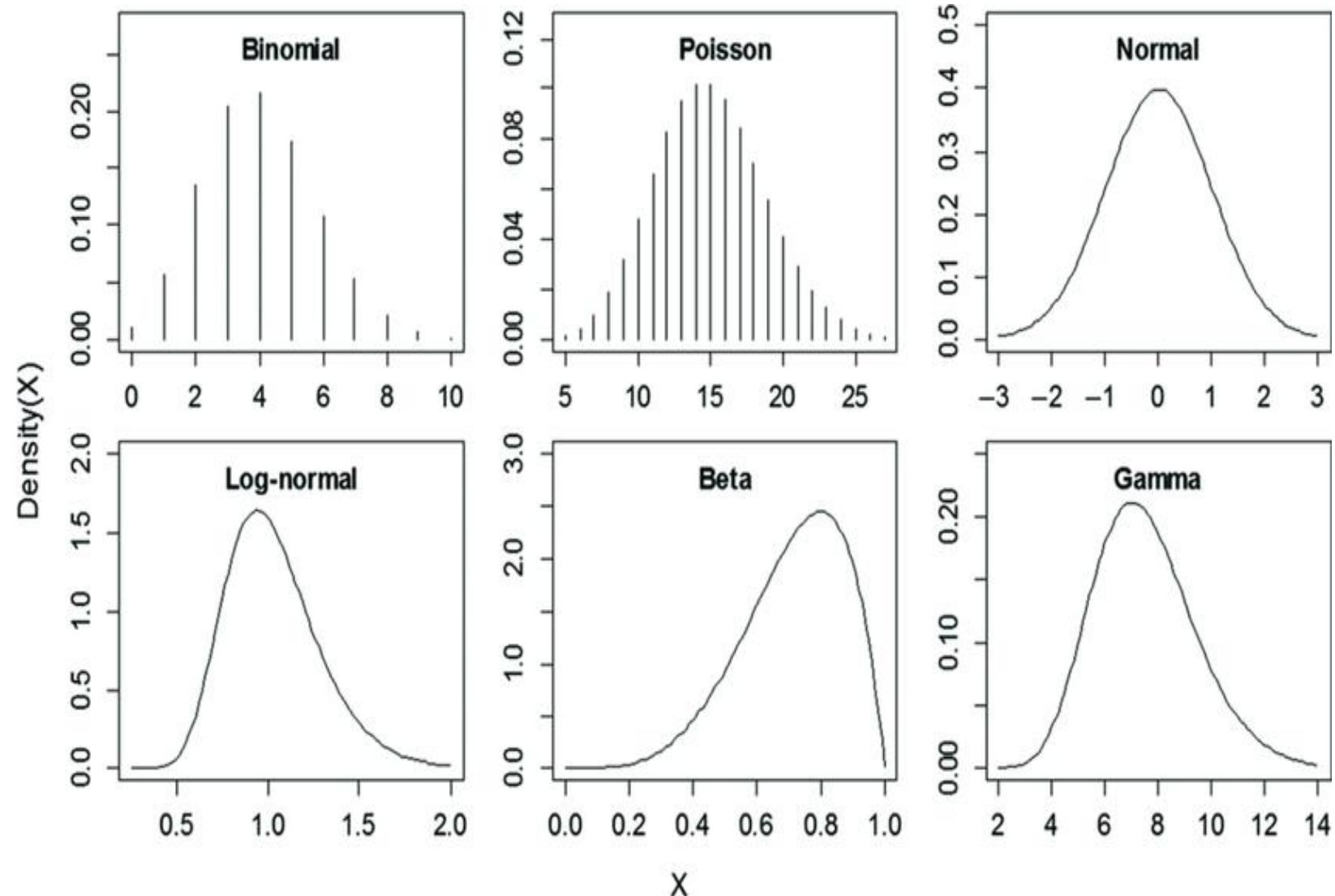


**Leptokurtic** (tall & thin)



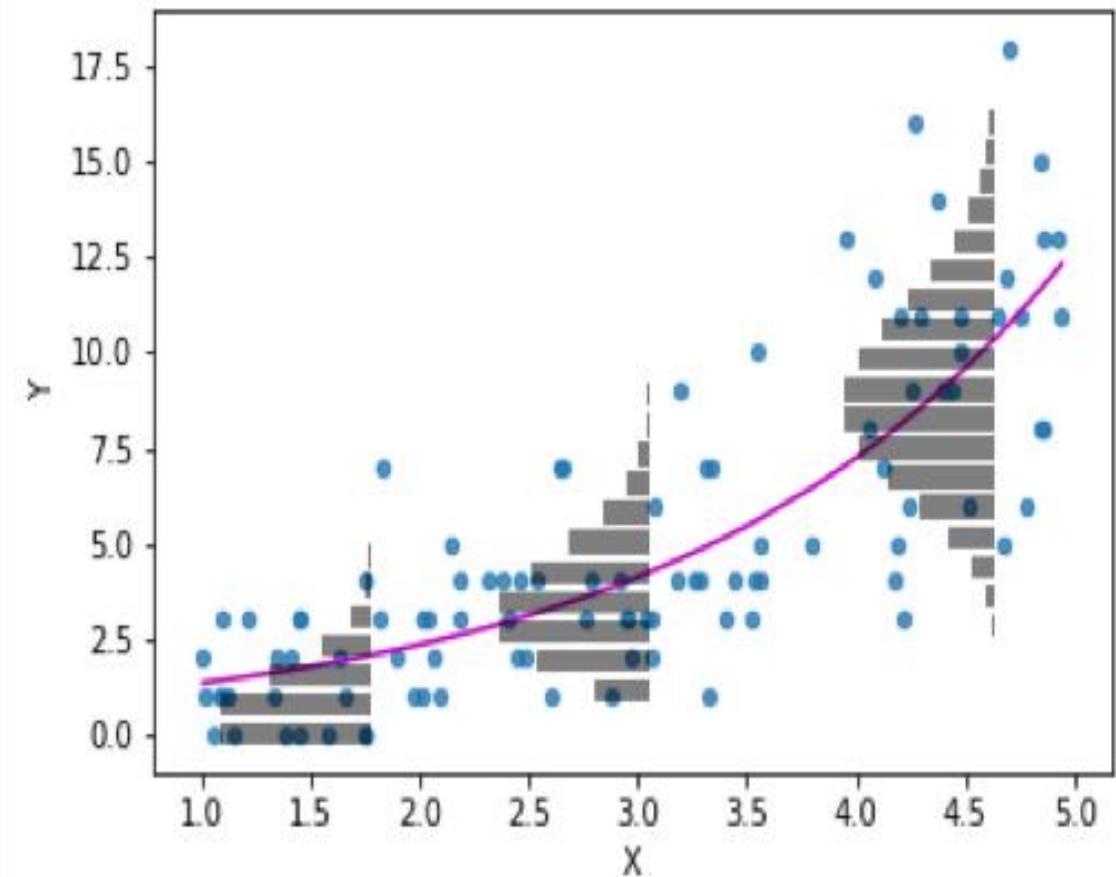
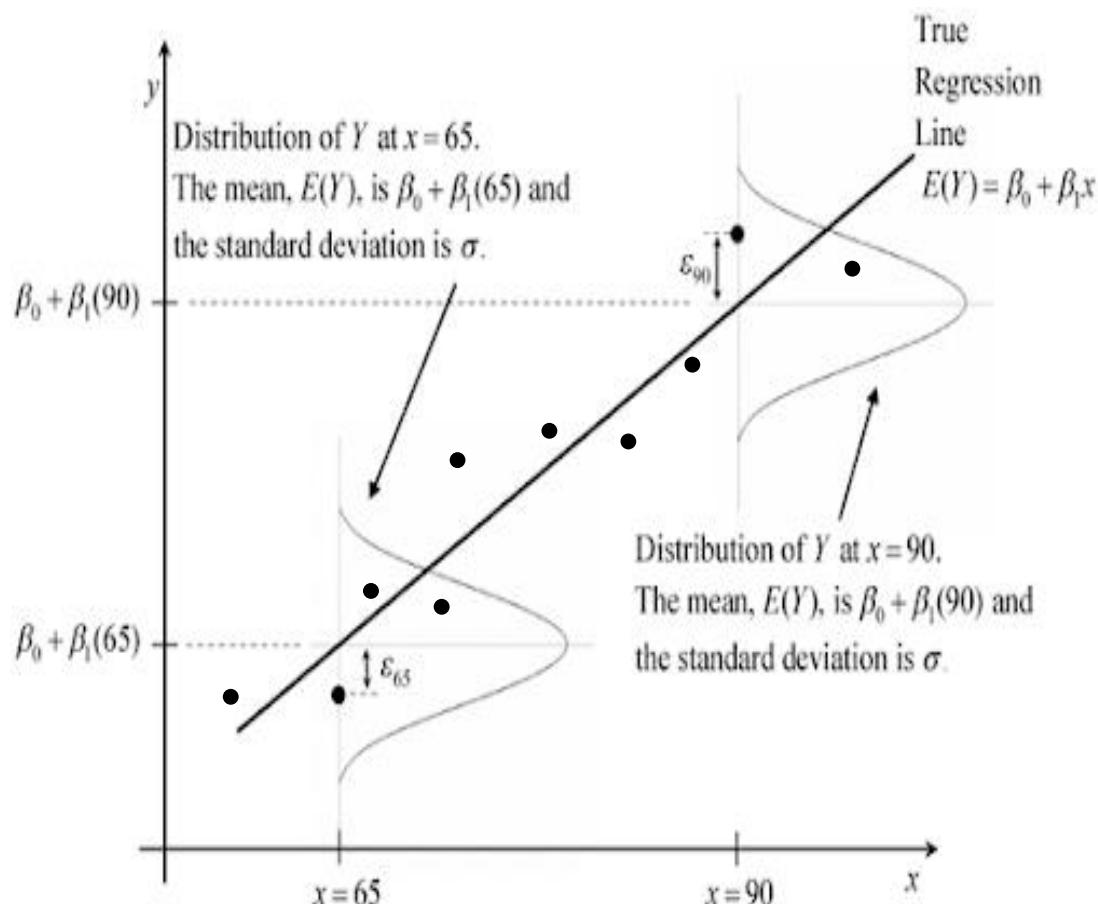
# Non-normal data Distribution

- Non-normality is a way of life, since no characteristic (height, weight, etc.) will have exactly a normal distribution (also known as the Gaussian distribution).
- One strategy to make non-normal data resemble normal data is by using a transformation.



# Generalized linear models vs Linear model

- Difference in data distribution can be understood from figures given below.
- Accordingly, GLM or LM can be chosen.



# Generalized linear models

- **Generalized linear models** represent the class of regression models which models the response variable,  $Y$ , and the random error term ( $\epsilon$ ) based on exponential family of **distributions such as Poisson, Gamma, Binomial, inverse Gaussian etc.**
- GLM assumes that the distribution of the response variable is a member of the exponential family of distribution.
- This is different from the general **linear models** (linear regression / ANOVA) where response variable,  $Y$ , and the random error term ( $\epsilon$ ) have to be based solely on the **normal distribution**.

# Generalized linear models and **Link functions**

- There are **three** components in generalized linear models.

1  **Link function** **Linear predictor**

$$\ln \lambda_i = b_0 + b_1 x_i$$

$$y_i \sim \text{Poisson}(\lambda_i)$$

**(In the case of Poisson regression)**

**Probability distribution**

- Link function literally “links” the linear predictor and the parameter for probability distribution.

# Generalized linear models and **Link functions**

- The generalized linear models (GLM) generalizes linear regression by allowing the linear model to be related to the response variable via a link function.
- In GLM, a link function maps a nonlinear relationship to a linear one so that a linear model can be fit.
- For example, in logistic regression, we want to find the probability of success:  $P(Y = 1)$
- After  $\log(\text{odds})$  is fit to a linear model, then it can be mapped back to probabilities.
- $\log(\text{odds})$  is the link function for logistic regression.

$$\log(\text{odds}) = \log \frac{P(\text{Class 1}|\mathbf{x})}{1 - P(\text{Class 1}|\mathbf{x})} = w_0 + w_1 x_1 + \cdots + w_n x_n$$

# Link functions

- The link function for linear regression is the identity function.
  - An identity function maps every element in a set to itself.
  - In other words, the linear model directly predicts the outcome.
  - Other regressions use different link functions to transform the data.
- Some common link functions and their uses:

Response Variable Type	Distribution	Support	Link name	Link function
Only two possible outcomes	Bernoulli	Integers: {0, 1}	Logit	$XB = \ln(\mu / 1 - \mu)$
Categorical	Binomial	integers: {0, 1, ..., N}	Logit	$XB = \ln(\mu / n - \mu)$
Continuous symmetric	Normal Distribution	real: $(-\infty, \infty)$	Identity	$XB = \mu$
Time to event	Exponential Distribution	real: $(0, \infty)$	Negative inverse	$XB = -\mu^{-1}$
Time to event	Gamma Distribution	real: $(0, \infty)$	Negative inverse	$XB = -\mu^{-1}$
Count	Poisson Distribution	integers: 0, 1, 2, ...	Log	$XB = \ln(\mu)$

# When to use Which GLM Model?

- If the response variable represents counts (non-negative integer valued) or relative frequencies (non-negative), Poisson regression with log-link is used.
- If the response variable values are positive valued and skewed, Gamma Regressor with log-link can be tried.
- If the response variable values seem to be heavier tailed than a Gamma distribution, one may try an Inverse Gaussian distribution based regressor.
- If the response variable is related to binary outcome, Logistic regression with Logit link can be used.

# Logistic Regression Introduction

# Basics

- Logistic Regression is used when the data is linearly separable and the outcome is binary or dichotomous in nature.
- That means Logistic regression is usually used for Binary classification problems.
- It can be extended to solve multiclass classification problems.
- Multinomial Logistic Regression: The output variable is discrete in three or more classes with no natural ordering.
  - ex:- Food texture: Crunchy, Mushy, Crispy  
Hair colour: Blonde, Brown, Brunette, Red
- Ordered Logistic Regression: The output variable is discrete in three or more classes with the ordering of the levels.
  - ex:- Customer Rating: extremely dislike, dislike, neutral, like, extremely like;  
Income level: low income, middle income, high income

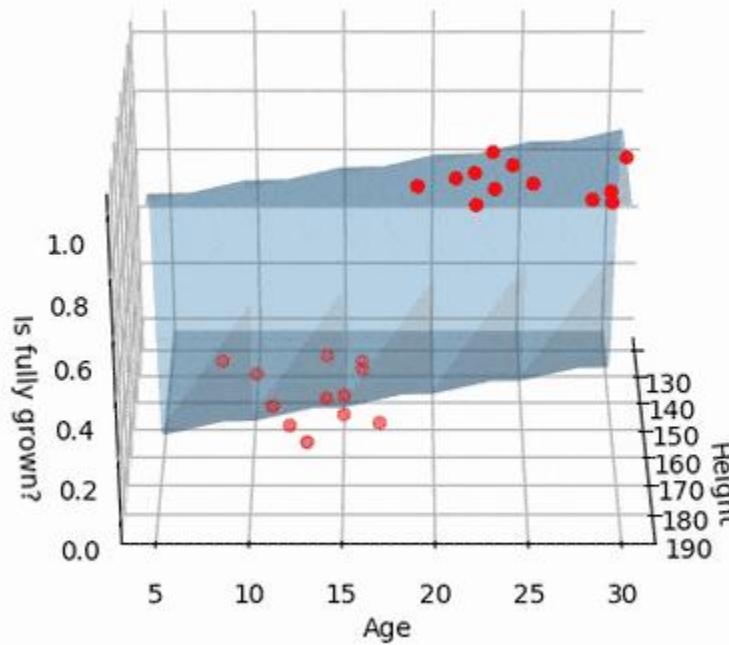
# Basics

- Logistic regression is a “Supervised machine learning” algorithm that can be used to model the probability of a certain class or event.
- Logistic regression is a nonlinear regression problem.
- As we use least-squares method in linear regression maximum likelihood method is used in logistic regression.
- A logistic regression model applies a logit transformation to the probabilities. The logit is the natural log of the odds.
- $P$  is the probability of the event
- $\ln$  is the natural log (to the base e)
- Logit is also denoted as  $\ln$

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right)$$

$$p = \frac{1}{1 + e^{-\text{logit}(p)}}$$

# Example for Binary classification



# Binary classification problems

Email: Spam / Not Spam?

Online Transactions: Fraudulent (Yes / No)?

Tumor: Malignant / Benign ?

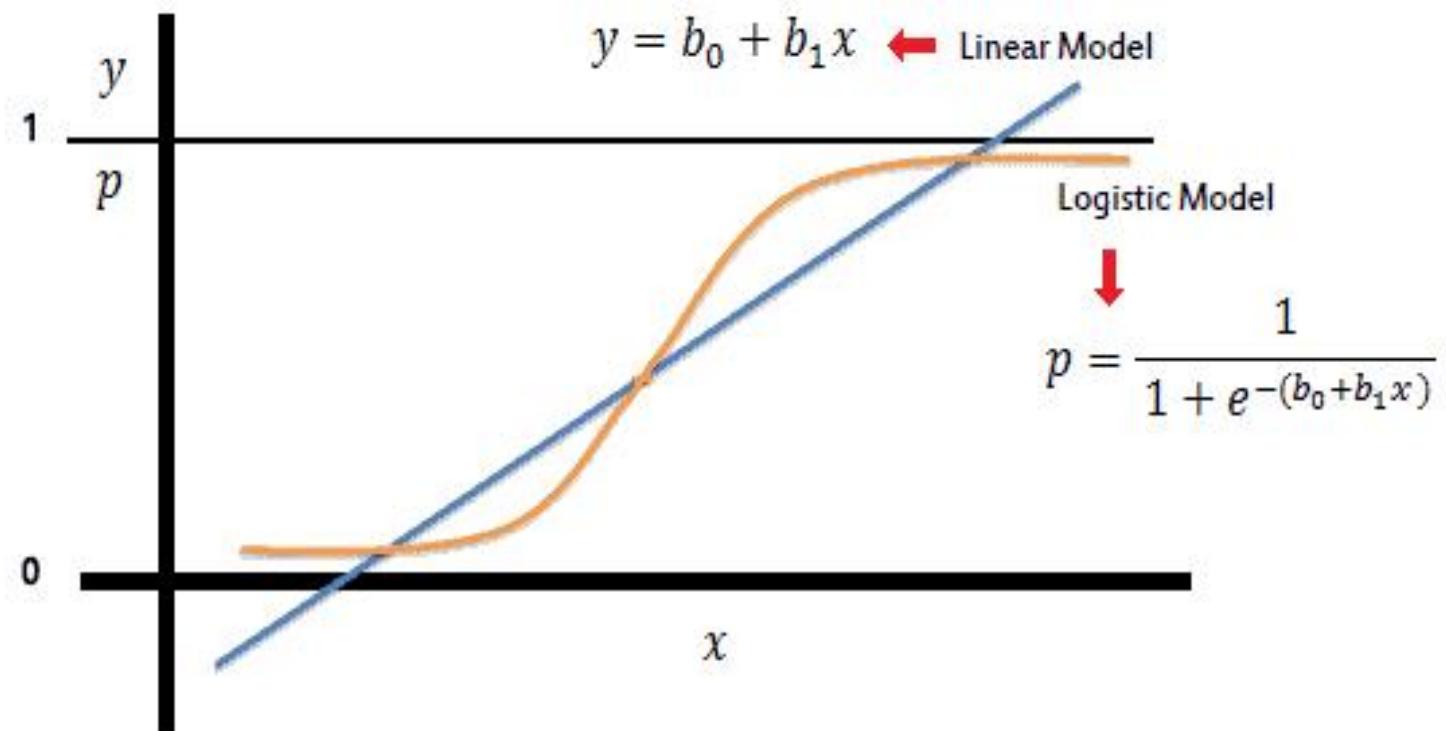
$$y \in \{0, 1\}$$

0: “Negative Class” (e.g., benign tumor)

1: “Positive Class” (e.g., malignant tumor)

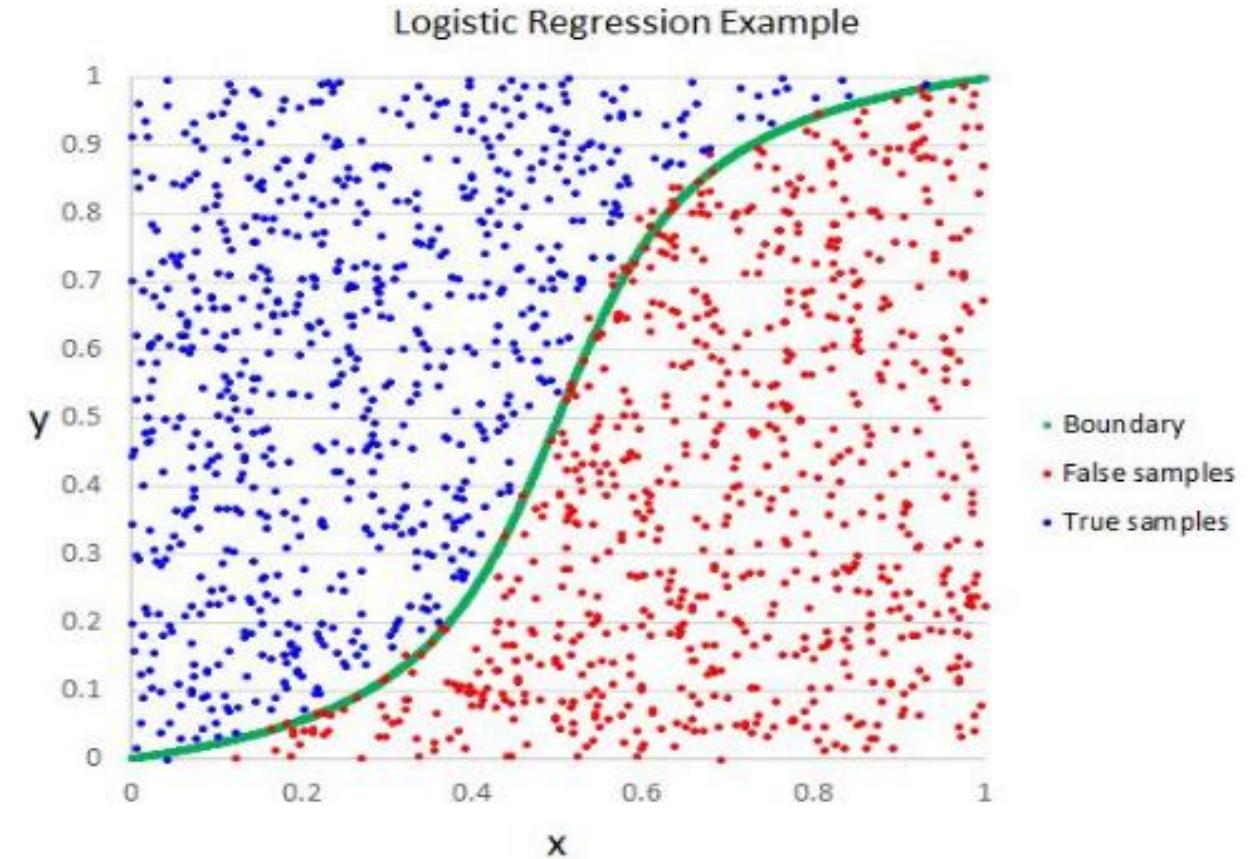
# Logistic Regression vs Linear Regression

- The plot shows a model of the relationship between a continuous predictor and the probability of an event or outcome.
- The linear model clearly does not fit if this is the true relationship between  $X$  and the probability.
- In order to model this relationship directly, you must use a nonlinear function.
- The plot displays one such function: The S-shaped curvy function is known as sigmoid.



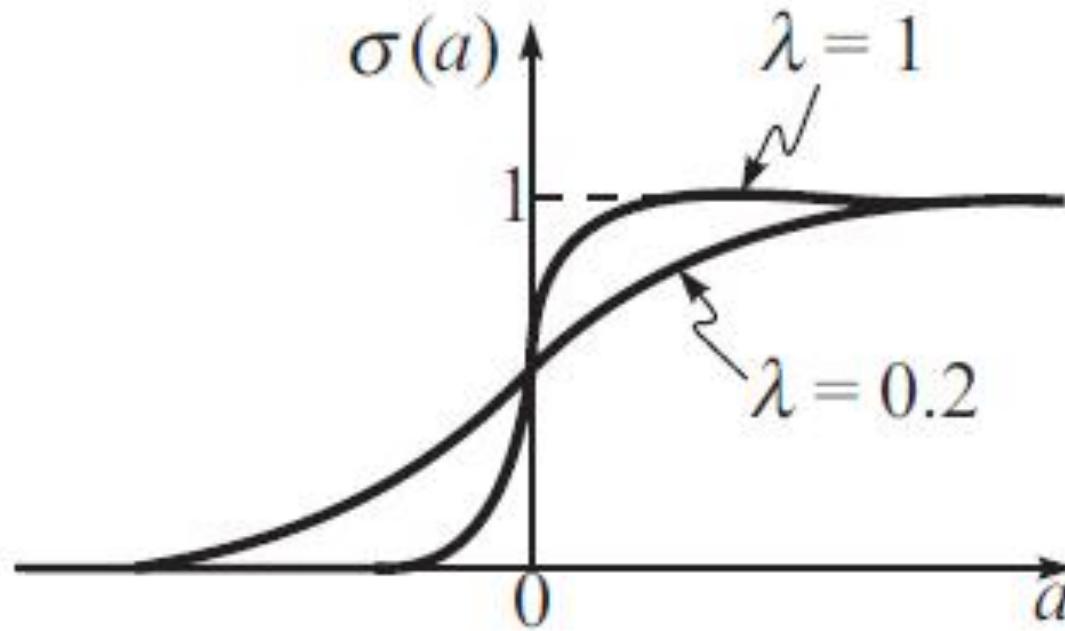
# Logistic Regression vs Linear Regression

- The plot shows a model of the relationship between a continuous predictor and the probability of an event or outcome.
- The linear model clearly does not fit if this is the true relationship between X and the probability.
- In order to model this relationship directly, you must use a nonlinear function.
- The plot displays one such function: The S-shaped curvy function is known as sigmoid.



# Sigmoid function

$$\sigma(a) = \frac{1}{1 + e^{-\lambda a}}$$



- It is continuous and varies monotonically from 0 to 1 as  $a$  varies from  $-\infty$  to  $\infty$ .
- The gain of the sigmoid,  $\lambda$ , determines the steepness of the transition region.
- One of the advantages of the sigmoid is that it is differentiable.

# Problem Formulation

Training set:  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

m examples

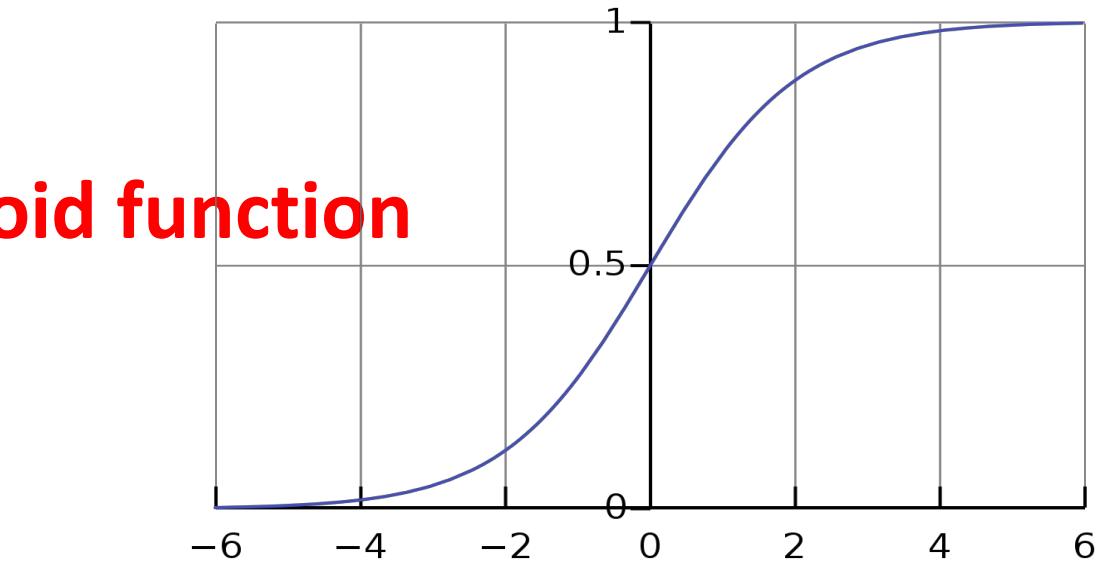
$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}$$
$$x_0 = 1, y \in \{0, 1\}$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters  $\theta$ ?

Max Likelihood Estimation (**already discussed**)

Sigmoid function



Threshold classifier output  $h_{\theta}(x)$  at 0.5:

If  $h_{\theta}(x) \geq 0.5$ , predict "y = 1"

If  $h_{\theta}(x) < 0.5$ , predict "y = 0"

# LOGISTIC REGRESSION

Dr. Srikanth Allamsetty

# Likelihood Vs Probability

# Probability vs Statistics

- In probability theory we consider some underlying process which has some randomness or uncertainty modeled by random variables, and we figure out what happens.
- In statistics we observe something that has happened, and try to figure out what underlying process would explain those observations.
- Likelihood function is a fundamental concept in statistical inference.
- It indicates how likely a particular population is to produce an observed sample.
- Probability points to chances while likelihood denotes a possibility.

# Likelihood Vs Probability

- Probability is simply how likely something is to happen.
- The occurrence of discrete values  $y_k$  is expressed by the probability  $P(y_k)$ .
- The distribution of all possible values of discrete random variable  $y$  is expressed as probability distribution.
- We assume that there is some *a priori probability* (or simply prior)  $P(y_k)$  that the next feature vector belongs to the class  $k$ .
- $P(x|y_k)$  is called the class *likelihood* and is the conditional probability that a pattern belonging to class  $y_k$  has the associated observation value  $x$ .
- Any class that maximizes  $P(x|y_q)$  is called Maximum Likelihood (ML) class.

$$y_{\text{ML}} \equiv \arg \max_q P(\mathbf{x}|y_q)$$

# Likelihood Vs Probability

- Probability follows clear parameters and computations while a likelihood is based merely on observed factors/data.

$$L(\mu, \sigma; \text{data}) = P(\text{data}; \mu, \sigma)$$

- $P(\text{data}; \mu, \sigma)$  It means “the probability density of observing the data with model parameters  $\mu$  and  $\sigma$ ”. It’s worth noting that we can generalise this to any number of parameters and any distribution.
- On the other hand  $L(\mu, \sigma; \text{data})$  means “the likelihood of the parameters  $\mu$  and  $\sigma$  taking certain values given that we’ve observed a bunch of data.”
- But despite these two things being equal, the likelihood and the probability density are fundamentally asking different questions — one is asking about the data and the other is asking about the parameter values.

# Example of **Probability**

- Consider a dataset containing the heights of the people of a particular country. Let's say the mean of the data is 170 & the standard deviation is 3.5.
- When Probability has to be calculated of any situation using this dataset, then the dataset features will be constant i.e. mean & standard deviation of the dataset will be constant, they will not be altered.
- Let's say the probability of height  $> 170$  cm has to be calculated for a random record in the dataset, then that will be calculated using the information shown below:

$$P(\text{height} > 170\text{cm} | \mu = 170, \sigma = 3.5)$$

- While calculating probability, feature value can be varied, but the characteristics(mean & Standard Deviation) of the data distribution cannot be altered.

# Example of Likelihood

- Likelihood calculation involves calculating the best distribution or best characteristics of data given a particular feature value or situation.
- Consider the exactly same dataset example as provided above for probability, if their likelihood of height > 170 cm has to be calculated then it will be done using the information shown below:

$$\text{Likelihood}(\mu = 170, \sigma = 3.5 | \text{height} > 170\text{cm})$$

- In the calculation of the Likelihood, the equation of the conditional probability flips as compared to the equation in the probability calculation.
- Here, the dataset features will be varied, i.e. Mean & Standard Deviation of the dataset will be varied to get the maximum likelihood for height > 170 cm.
- The likelihood in very simple terms means to increase the chances of a particular situation to happen/occur by varying the characteristics of the dataset distribution.

# Logistic Regression Implementation

# Hypothetical function

- In Logistic Regression, we apply the sigmoid activation function on the hypothetical function of linear regression.
- So the resultant hypothetical function for logistic regression is given below:

$$h(x) = \text{sigmoid}(wx + b)$$

Here,  $w$  is the weight vector.

$x$  is the feature vector.

$b$  is the bias.

$$\text{sigmoid}(z) = 1 / (1 + e^{(-z)})$$

# Cost function

- The cost function of linear regression (mean square error) can't be used in logistic regression because it is a non-convex function of weights.
- Optimizing algorithms like i.e gradient descent only converge convex function into a global minimum.
- So, the simplified cost function we use :

$$J = -y \log(h(x)) - (1 - y) \log(1 - h(x)) \text{ (it's derived in last class)}$$

here,  $y$  is the real target value

$$h(x) = \text{sigmoid}(wx + b)$$

$$\text{For } y = 0, J = -\log(1 - h(x))$$

$$\text{and } y = 1, J = -\log(h(x))$$

# Gradient Descent Calculation

```
repeat until convergence {  
    tmpi = wi - alpha * dwi  
    wi = tmpi  
}
```

where alpha is the learning rate.

- The chain rule is used to calculate the gradients like i.e  $dw_i$ .

$$\frac{\partial J}{\partial w} = \frac{\partial J}{\partial a} * \frac{\partial a}{\partial z} * \frac{\partial z}{\partial w}$$

- here,  $a = \text{sigmoid}(z)$  and  $z = wx + b$ .

# Next?

- Update weights in an iterative process
- After completing all iterations, calculate Hypothetical function  $h(x)$

Threshold classifier output  $h(x)$  at 0.5:

If  $h(x) \geq 0.5$ , predict "y = 1"

If  $h(x) < 0.5$ , predict "y = 0"

# Logistic Regression Numerical

# Example 1

- Some samples of two classes of articles: Technical (1) and Non-technical (0) are given.
- Each class has two features:
  - Time, which represent the average time required to read an article in hours,
  - Sentences, representing a number of sentences in a book
- first, we need to train our logistic regression model.

Time (Hr)	Sentences (1= 1000)	Article Type
2.7	2.5	0
1.4	2.3	0
3.3	2.4	0
1.3	1.8	0
3	3	0
7.6	2.7	1
5.9	2.2	1
6.9	1.8	1
8.6	3.5	1
7.7	3.5	1

# Example 1

- Training involves finding optimal values of coefficients which are  $\beta_0$ ,  $\beta_1$ , and  $\beta_2$ .
- While training, we find some value of coefficients in the first step and use those coefficients in another step to optimize their value.
- We continue to do it until we get consistent accuracy from the model.

$$\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \eta \sum_{i=1}^N (y^{(i)} - g(\bar{\mathbf{x}}^{(i)}, \bar{\mathbf{w}})) \mathbf{x}^{(i)}$$

$$g(\mathbf{x}^{(i)}, \bar{\mathbf{w}}) = P(y^{(i)} | \mathbf{x}^{(i)})$$

Time (Hr)	Sentences (1= 1000)	Article Type
2.7	2.5	0
1.4	2.3	0
3.3	2.4	0
1.3	1.8	0
3	3	0
7.6	2.7	1
5.9	2.2	1
6.9	1.8	1
8.6	3.5	1
7.7	3.5	1

1.9

3.1

?

# Example 1

- After 20 iteration, we get:

$$B_0 = -0.1068913$$

$$B_1 = 0.41444855$$

$$B_3 = -0.2486209$$

- Thus, the decision boundary is given as:

$$Z = B_0 + B_1 * X_1 + B_2 * X_2$$

$$Z = -0.1068913 + 0.41444855 * \text{Time} - 0.2486209 * \text{Sentences}$$

Time (Hr)	Sentences (1= 1000)	Article Type
2.7	2.5	0
1.4	2.3	0
3.3	2.4	0
1.3	1.8	0
3	3	0
7.6	2.7	1
5.9	2.2	1
6.9	1.8	1
8.6	3.5	1
7.7	3.5	1

# Example 1

- For,  $X_1 = 1.9$  and  $X_2 = 3.1$ , we get:

$$Z = -0.101818 + 0.41444855 * 1.9 - 0.2486209 * 3.1$$

$$Z = -0.085090545$$

- Now, we use sigmoid function to find the probability and thus predicting the class of given variables.

$$y = \frac{1}{1 + e^{-Z}} \quad \text{or,} \quad y = \frac{1}{1 + e^{-(0.085090545)}}$$

- As  $y=0.477$ , is less than 0.5, we can safely classify given sample to class Non-technical.

Time (Hr)	Sentences (1= 1000)	Article Type
2.7	2.5	0
1.4	2.3	0
3.3	2.4	0
1.3	1.8	0
3	3	0
7.6	2.7	1
5.9	2.2	1
6.9	1.8	1
8.6	3.5	1
7.7	3.5	1

1.9

3.1

?

# Examples 2 & 3

- Can be seen here in the following links
- <https://machinelearningmastery.com/logistic-regression-tutorial-for-machine-learning/>
- <https://courses.lumenlearning.com/introstats1/chapter/introduction-to-logistic-regression/>

# TREE BASED LEARNER

Srikanth Allamsetty

# Idea of a tree based learner

# Introduction

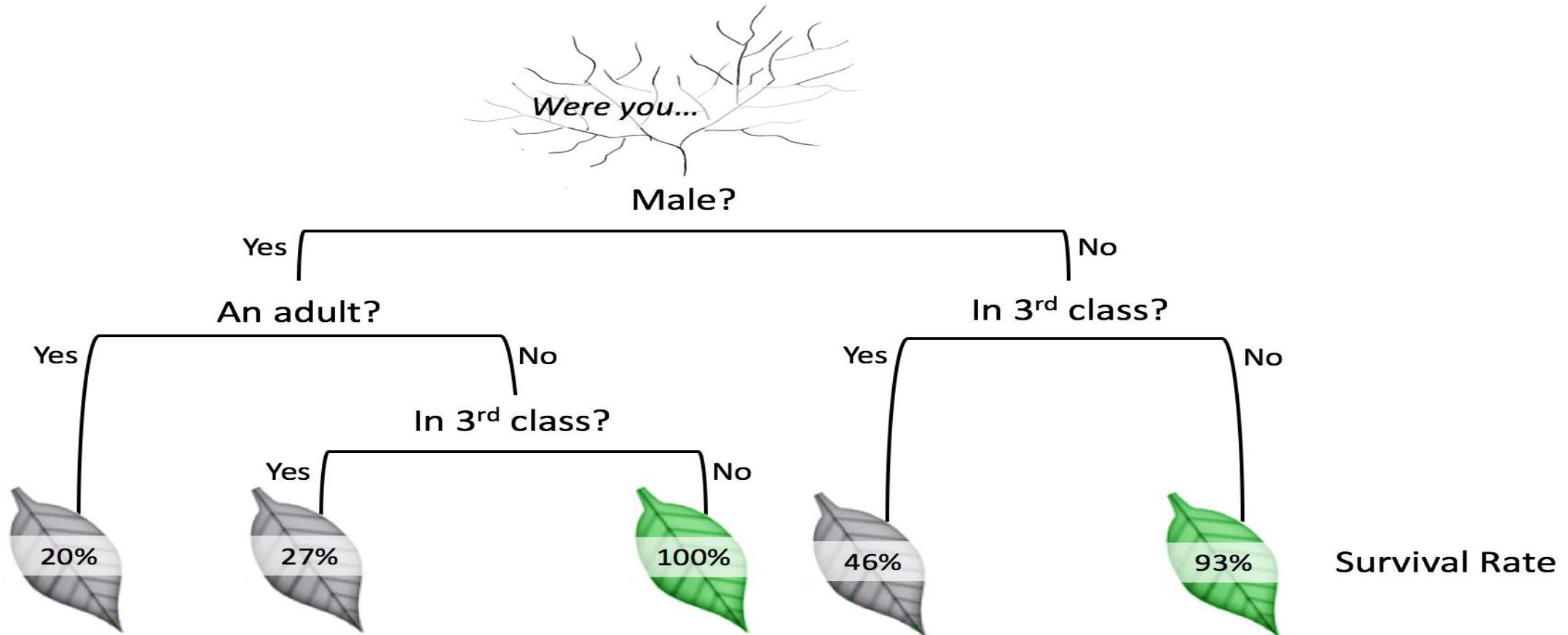
- Unlike linear models, Tree based algorithms map non-linear relationships quite well.
- Methods like decision trees, random forest, gradient boosting are being popularly used in all kinds of data science problems.
- Decision Trees (DTs) are a non-parametric supervised learning method used for classification and regression.
- The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
- A tree can be seen as a piecewise constant approximation.

# Introduction

- A decision tree can be said to be a map of reasoning process.
- It uses a structure resembling that of a tree to describe a dataset and solutions can be visualized by following different pathways through the tree.
- It is a hierarchical set of rules explaining the way in which a large set of data can be divided into smaller data partitions.
- Each time a split takes place, the components of the resulting partitions become increasingly similar to one another with regard to the target.
- Capable of performing well across a wide range of situations.

# Introduction

- Can you imagine the data we have for making this decision tree?

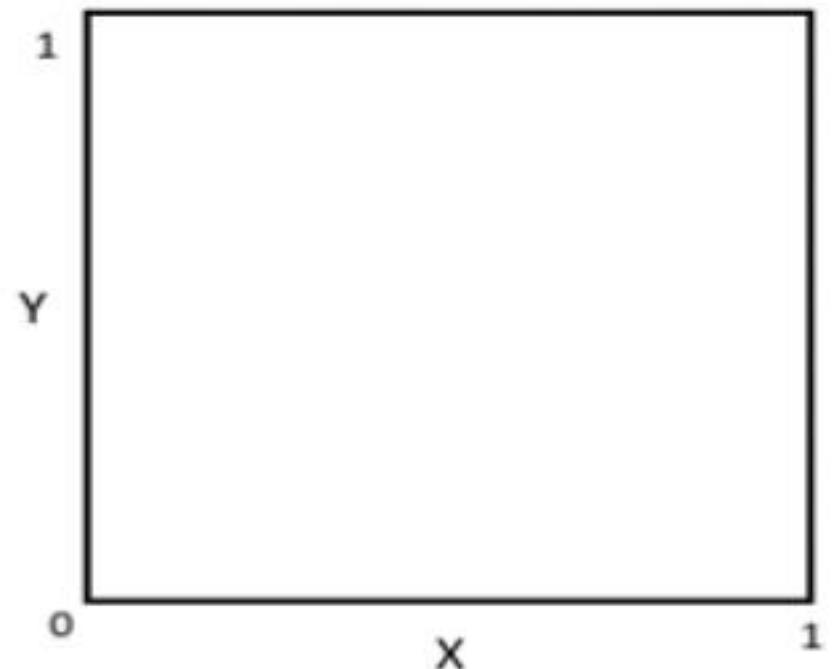
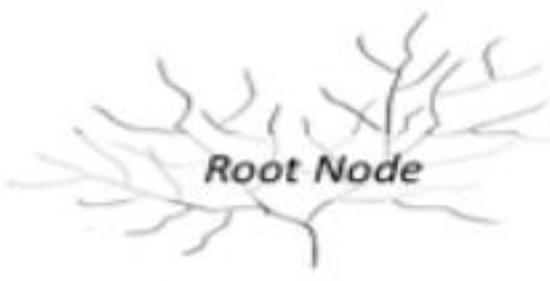


# Introduction

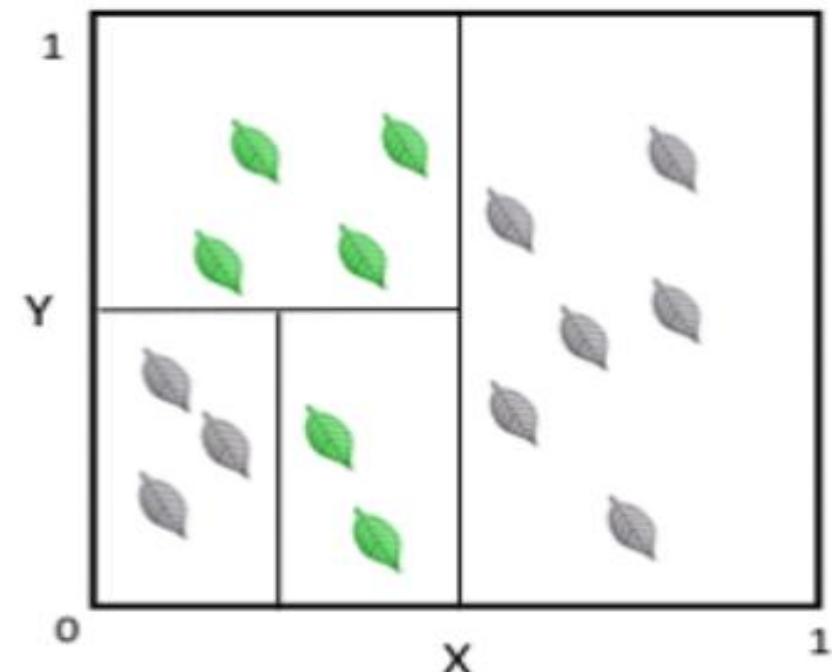
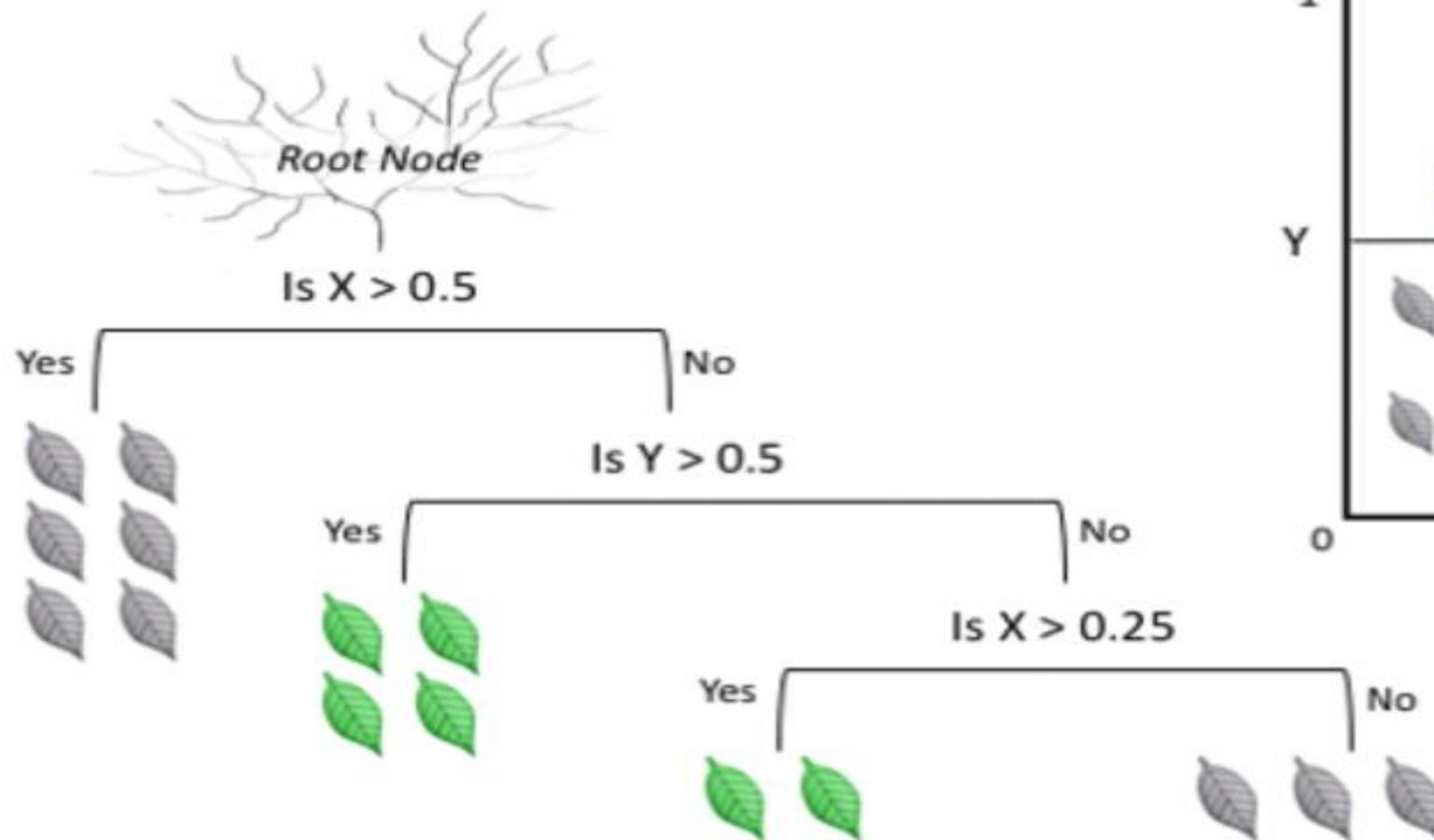
- Decision-tree learning is generally best suited to problems with the following characteristics:
  - Patterns are described by a fixed set of attributes  $x_j; j=1,2,\dots,n$  and each attribute  $x_j$  takes on a small number of disjoint possible values (categorical or numeric)  $v_{lx_j}; l = 1, 2, \dots, d_j$
  - The output variable  $y$  is Boolean-valued function (binary classification problems) defined over the set  $S$  or patterns  $\{s^{(i)}\} \equiv \{\mathbf{x}^{(i)}\}; i = 1,2, \dots, N$ . That is,  $y$  take on values  $y_q; q = 1, 2$ . For instance, if  $y_1 \equiv 0$  and  $y_2 \equiv 1$ , then  $y: S \rightarrow [0, 1]$ .
  - The training data is described by the dataset  $\mathcal{D}$  of  $N$  patterns with corresponding observed outputs:

$$\mathcal{D} = \{s^{(i)}, y^{(i)}\} = \{\mathbf{x}^{(i)}, y^{(i)}\}; i = 1,2, \dots, N$$

# Example for Decision tree based Classification



# Example for Decision tree based Classification



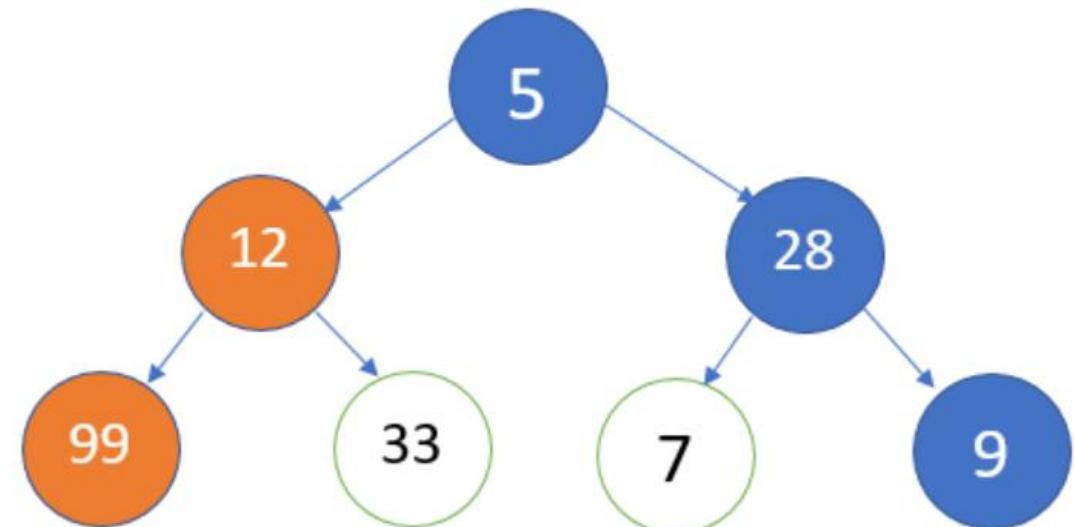
- Extension of the basic decision-tree learning algorithm allow handling of continuous- valued attributes, and learning functions with more than two possible output values (multiclass classification problems).
- Extensions also allow learning target functions with continuous-valued outputs (regression problems), though the application of decision trees in this setting is not very common.

### Basic characteristics of decision-tree building:

- Hierarchical model for supervised learning.
- Training set is portioned into smaller subsets in a sequence of recursive splits as the tree is being built. The building follows a top-down hierarchical approach.
- Tree-learning algorithm are greedy; best split (nonbacktracking) is desired.
- It picks the best immediate output, but does not consider the big picture, hence it is considered greedy.
- Tree with no error and smallest number of nodes is desired.
- Divide and conquer, a frequently used heuristic, is the tree building strategy.

# Disadvantage of being greedy

- For example: Take the path with the largest sum overall.
- A greedy algorithm would take the blue path, as a result of shortsightedness, rather than the orange path, which yields the largest sum.



# Steps in Decision Tree and Construction

# An Example of a Classification Decision Tree

- Sample Data

Instances	Outlook $x_1$	Temperature $x_2$	Humidity $x_3$	Wind $x_4$	PlayTennis $y$
$s^{(1)}$	Sunny	Hot	High	Weak	No
$s^{(2)}$	Sunny	Hot	High	Strong	No
$s^{(3)}$	Overcast	Hot	High	Weak	Yes
$s^{(4)}$	Rain	Mild	High	Weak	Yes
$s^{(5)}$	Rain	Cool	Normal	Weak	Yes
$s^{(6)}$	Rain	Cool	Normal	Strong	No
$s^{(7)}$	Overcast	Cool	Normal	Strong	Yes
$s^{(8)}$	Sunny	Mild	High	Weak	No
$s^{(9)}$	Sunny	Cool	Normal	Weak	Yes
$s^{(10)}$	Rain	Mild	Normal	Weak	Yes
$s^{(11)}$	Sunny	Mild	Normal	Strong	Yes
$s^{(12)}$	Overcast	Mild	High	Strong	Yes
$s^{(13)}$	Overcast	Hot	Normal	Weak	Yes
$s^{(14)}$	Rain	Mild	High	Strong	No

In the sample data set:

- The input variables are:
  - $x_1 = \text{outlook}$ ;  $x_2 = \text{Temperature}$
  - $x_3 = \text{humidity}$ ;  $x_4 = \text{wind}$
  - Target variable  $y = \text{PlayTennis}$ .

Target function to be learnt as:

$$\hat{y}: S \rightarrow [0, 1]$$

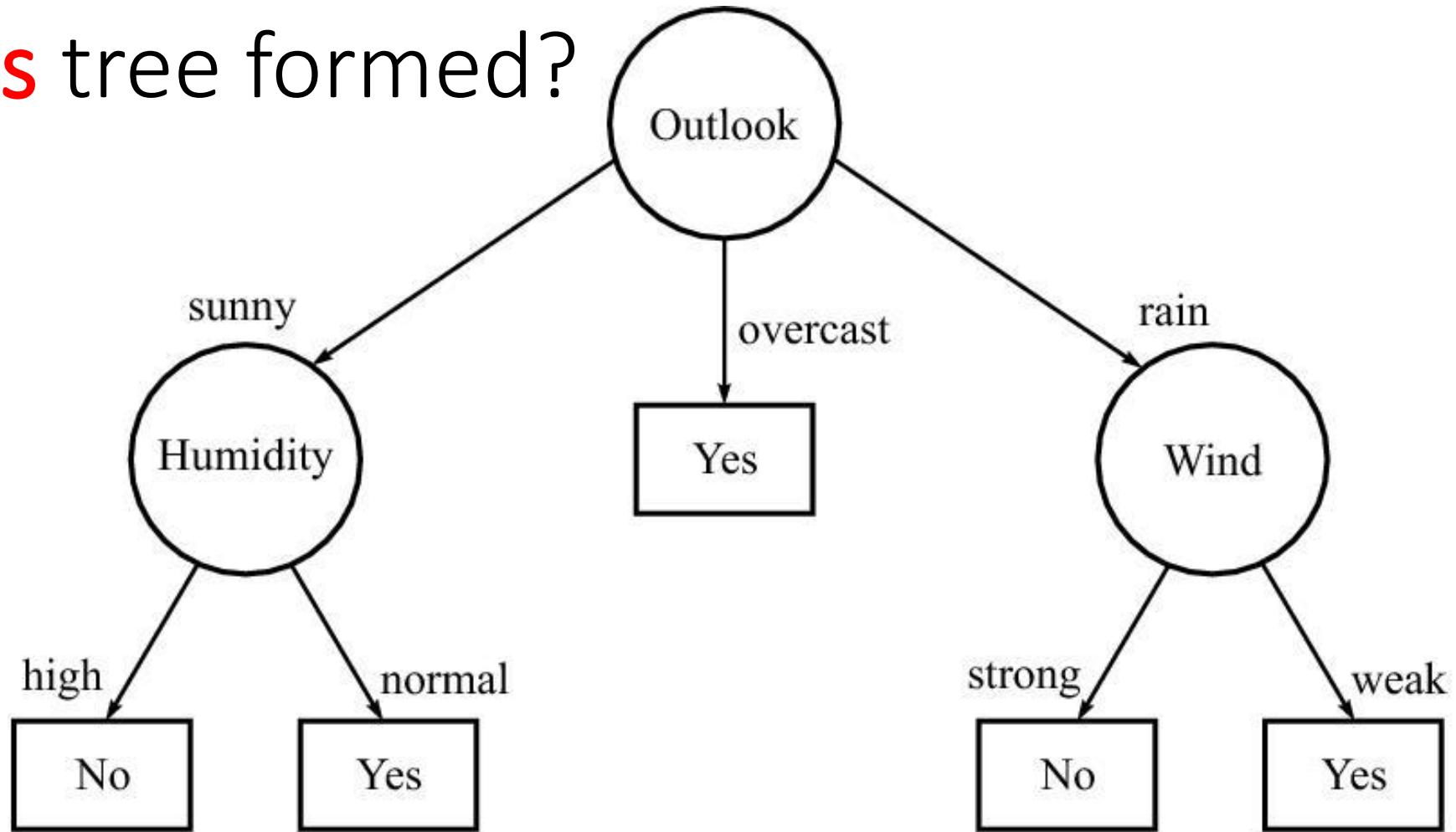
$\hat{y}^{(i)} \equiv \hat{y}(s^{(i)}) = 1$  if  $\text{PlayTennis} = \text{yes}$  and  $\hat{y}^{(i)} \equiv \hat{y}(s^{(i)}) = 0$  if  $\text{PlayTennis} = \text{No}$ .

- The set  $S$  of pattern has 14 data samples:

$$S = \{s^{(i)}\}; i = 1, \dots, 14$$

- The circle at the top of the diagram is the root node.
- It contains all the training data used to grow the tree

# How is **this** tree formed?



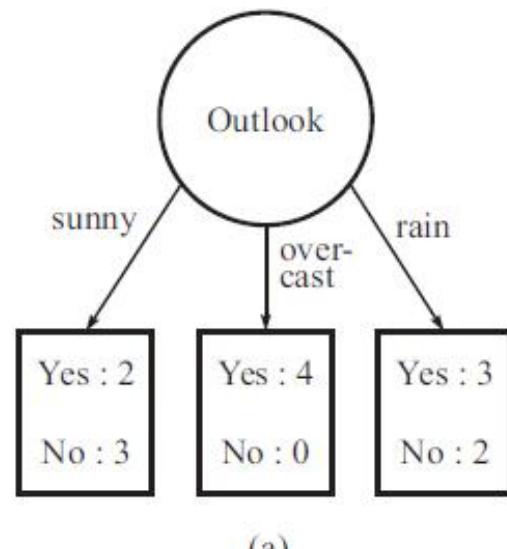
Decision Tree for the weather data

# How is the tree formed?

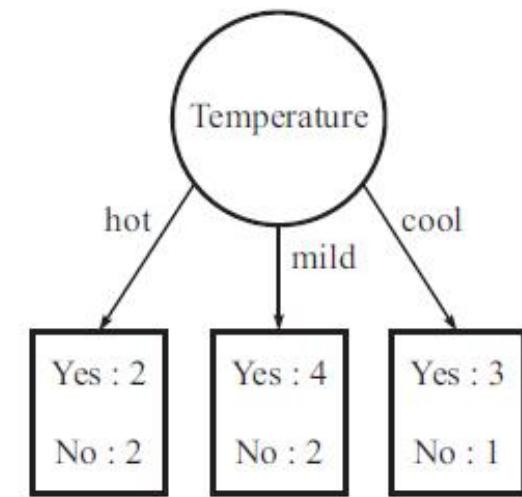
- The tree always starts from the root node and grows by splitting the data at each level into new nodes (daughter nodes).
- The root node (parent node) contains the entire data and daughter nodes (internal nodes) hold respective subsets of the data.
- All the nodes are connected by branches shown by the line segments.
- The nodes that are at the end of the branches are called terminal nodes or leaf nodes, shown by boxes.
- The leaf nodes in this figure are class labels.

# How is the tree formed?

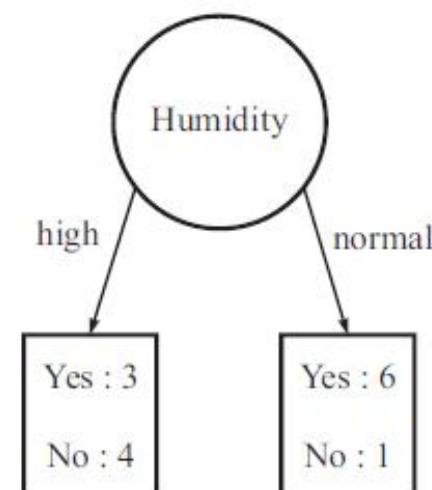
- Because we seek small trees, we would like the leaf nodes (all instances at the node having same classification, Yes or No) to come up as soon as possible,
- **so that further splits are reduced.**
- Therefore, we select Outlook as the splitting attribute at the root of the tree.
- because it is the only choice for which one daughter node becomes a leaf node.



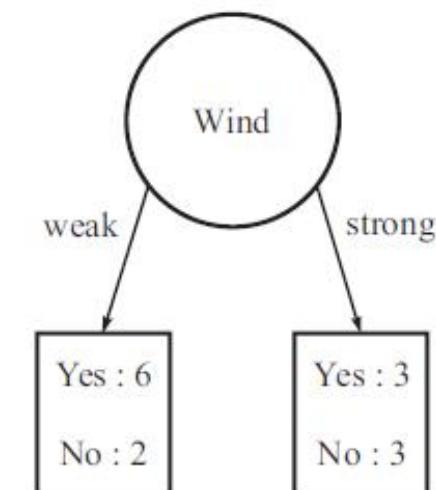
(a)



(b)



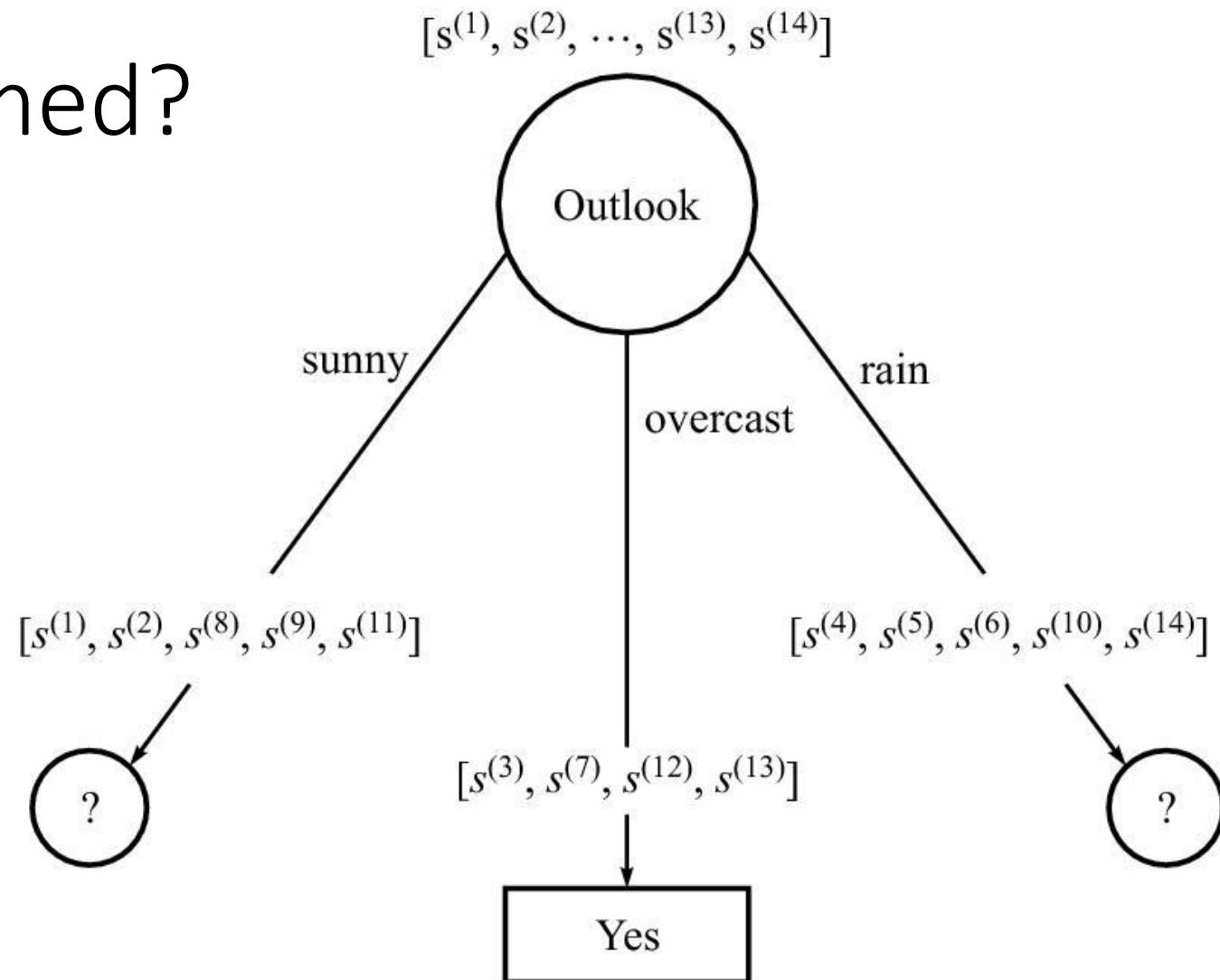
(c)



(d)

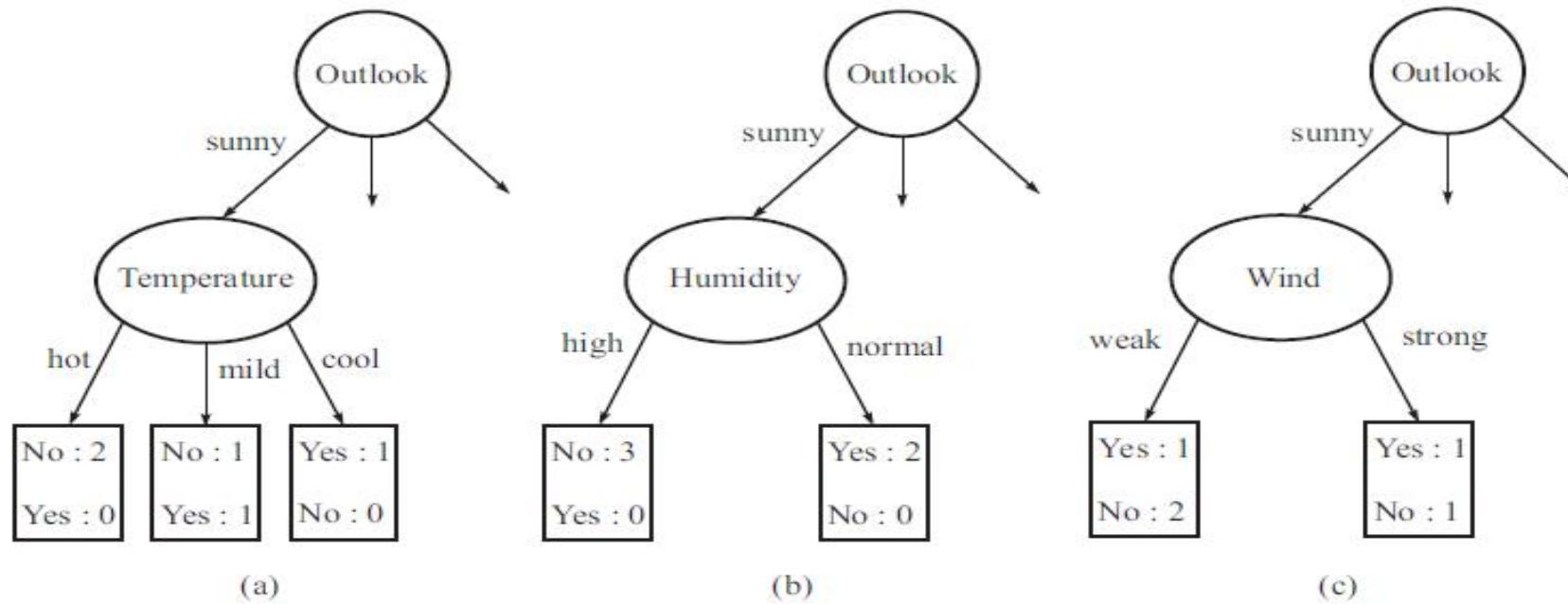
# How is the tree formed?

- Figure gives the data distribution at the root node and daughter nodes when Outlook is selected as the root node.



Partially learned decision tree: the training examples are sorted to corresponding descendant nodes

# How is **this** tree formed?



- Figure shows the possibilities for further branching at the node reached when **Outlook** is **sunny**.
- As if these were ‘root node’, we repeat the procedure described earlier.
- Again, intuitively, **Humidity** is the best to select.

# Parameters of Decision Tree Performance

# Measures of Impurity for Evaluating Splits in Decision Trees.

- An impurity is a heuristic for selecting the splitting criterion that “best” separates a given dataset  $\mathcal{D}$  of class labeled training tuples into individual classes.
  - a tuple is a finite ordered list of elements.
- If  $\mathcal{D}$  were split into smaller partitions according to the outcome of the splitting criterion, ideally each partition would be pure.
- There are three popular impurity measures—information gain, gain ratio, and Gini index.
- If the training data  $\mathcal{D}$  available is very less in numbers, as in the previous example, no need to use any of these measures.
- But this is not the case in real-time applications.

## Information gain/entropy reduction

(entropy measures how disorganized a system is...)

- The expected information needed to classify a pattern in  $\mathcal{D}$  is given by

$$Info(\mathcal{D}) = - \sum_{q=1}^2 P_q \log_2(P_q); \text{ where } P_q = \frac{freq(y_q, \mathcal{D})}{|\mathcal{D}|};$$

where  $freq(y_q, \mathcal{D})$  stands for the number of patterns in  $\mathcal{D}$  that belong to class  $y_q$  and  $|\mathcal{D}|$  denotes the total number of patterns in  $\mathcal{D}$  ( $|\mathcal{D}| = N$ ).

- A log function to the base 2 is used because information is encoded in bits.
- Info is the average amount of information needed to identify the class label.
- It can also be expressed as  $Entropy(\mathcal{D}) = - \sum_{q=1}^2 P_q \log_2(P_q)$
- $Info(\mathcal{D})$  is between 0 and 1.
  - 0 if all members belong to the same class; 1 when the collection contains an equal number of Class 1 and 2
- Thus, it is a measure of impurity of the collection of examples.

## Information gain/entropy reduction

- More the impurity (more the heterogeneity in the dataset), more the entropy, more the expected amount of information that would be needed to classify a new pattern.
- More the purity (more the homogeneity in the dataset), less the entropy, less the expected amount of information that would be needed to classify a new pattern.
- To illustrate, we consider training set given in Table (Weather Data).
- It has nine examples of class Yes, and five examples of class No.
- Therefore,  $\text{Info}(\mathcal{D}) = \text{Entropy}(\mathcal{D}) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14}$ 
$$= 0.94 \text{ bits}$$
- Root node with dataset  $\mathcal{D}$  will therefore be a highly impure node.

- Attribute  $x_j$  has distinct values  $v_{lx_j}; l = 1, \dots, d_j$ , as observed from the training data  $\mathcal{D}$ .
- Attribute  $x_j$  can be used to split data into  $l; l = 1, \dots, d_j$ , partitions or subsets  $\{\mathcal{D}_1, \mathcal{D}_2, \dots, \mathcal{D}_{d_j}\}$  where  $\mathcal{D}_l$  contains those patterns in  $\mathcal{D}$  that have distinct values  $v_{lx_j}$  of  $x_j$ .
- These partitions would correspond to branches grown from the node.
- It is quite likely that partitions will be impure.
- The amount of more information required (in order to arrive at an exact classification), is measured by

$$Info(\mathcal{D}, x_j) = \sum_{l=1}^{d_j} \frac{|\mathcal{D}_l|}{\mathcal{D}} \times Info(D_l)$$

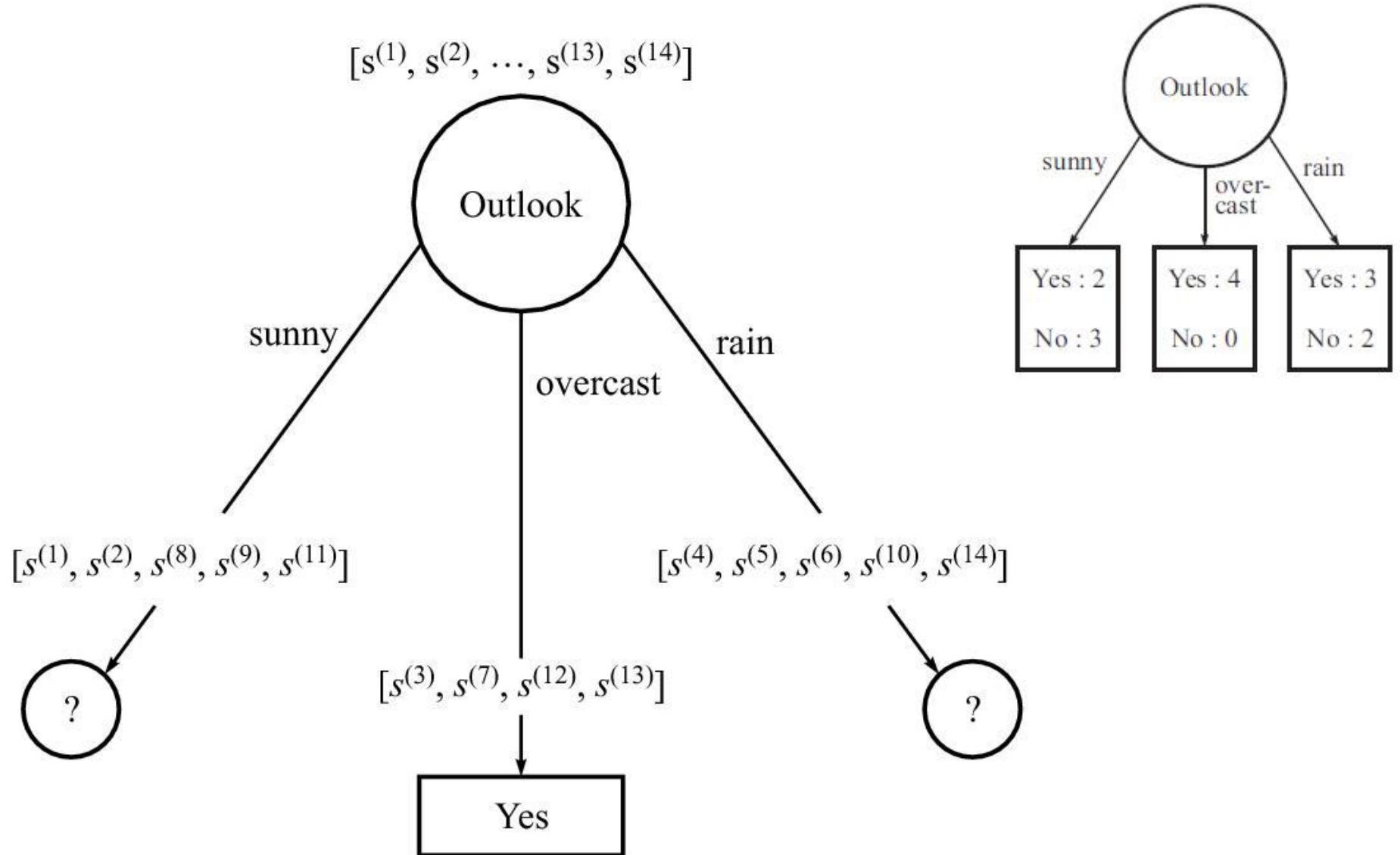
- The term  $|\mathcal{D}_l| / |\mathcal{D}|$  acts as the weight of  $l^{th}$  partition.
- $Info(\mathcal{D}_l)$  is given by:

$$Info(\mathcal{D}_l) = - \sum_{q=1}^2 P_{ql} \log_2(P_{ql})$$

- Where  $P_{ql}$  is the probability that the arbitrary sample in subset  $\mathcal{D}_l$  belongs to class  $y_q$  and is estimated as

$$P_{ql} = \frac{freq(y_q, \mathcal{D}_l)}{|\mathcal{D}_l|}$$

- $Info(\mathcal{D}, x_j)$  is expected information required to classify a pattern from  $\mathcal{D}$  based on the partitioning by  $x_j$ .
- The smaller the expected information (still) required, the greater the purity of the pattern.



Partially learned decision tree: the training examples are sorted to corresponding descendant nodes

$$Info(\mathcal{D}, x_1) = \sum_{l=1}^{d_1} \frac{|\mathcal{D}_l|}{|\mathcal{D}|} \times Info(\mathcal{D}_l); d_1 = 3$$

(for Outlook)

$$Info(\mathcal{D}_1) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.97$$

$$Info(\mathcal{D}_2) = 0$$

$$Info(\mathcal{D}_3) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.97$$

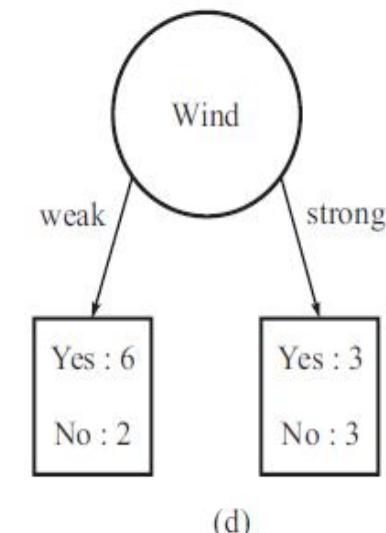
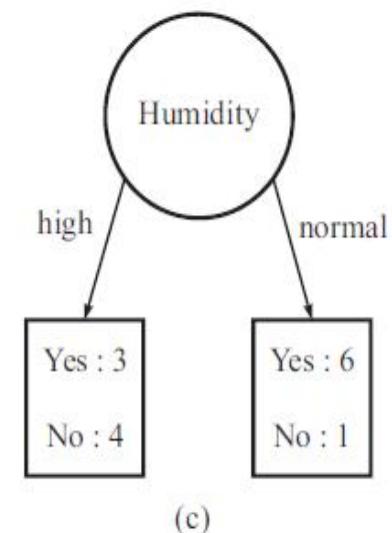
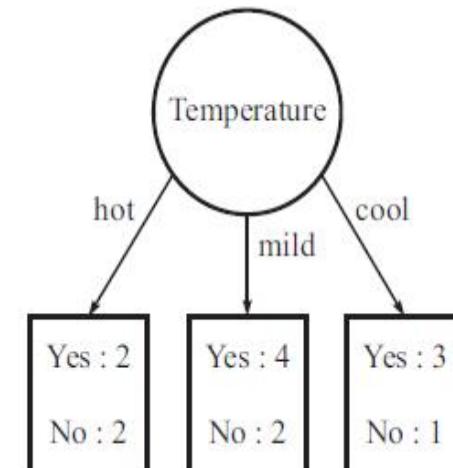
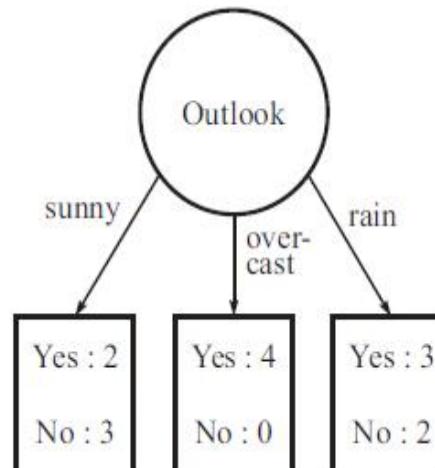
$$Info(\mathcal{D}, x_1) = \frac{5}{14} \times 0.97 - \frac{5}{14} \times 0.97 = 0.693$$

Similarly, for other tree stumps of figure, we obtain

$$Info(\mathcal{D}, x_2) = 0.911 \text{ (Temperature)}$$

$$Info(\mathcal{D}, x_3) = 0.788 \text{ (Humidity)}$$

$$Info(\mathcal{D}, x_4) = 0.892 \text{ (Wind)}$$



Tree stumps for the weather data

- We see that expected required information for classification is the least if we select attribute Outlook for the root node. Humidity is the next best choice.

## Information gain/entropy reduction

- Information gain is defined as the difference between the original information requirement and the new requirement

$$Gain(\mathcal{D}, x_j) = Info(\mathcal{D}) - Info(\mathcal{D}, x_j)$$

- The attribute  $x_j$  with the **highest** information gain,  $Gain(\mathcal{D}, x_j)$ , is chosen as the splitting attribute at the root node.
- We have selected  $x_1 = \text{Outlook}$  as the splitting attribute at the root node, for which  $Gain(\mathcal{D}, x_1) = 0.94 - 0.693 = 0.247$
- The same strategy is applied recursively to each subset of training instances.
- It measures the expected reduction in entropy, caused by partitioning the patterns in dataset  $\mathcal{D}$  according to the attribute  $x_j$ .

$$Entropy\ Reduction(\mathcal{D}, x_j) = Entropy(\mathcal{D}) - \sum_{l=1}^{d_j} \frac{|\mathcal{D}_l|}{|\mathcal{D}|} \times Entropy(\mathcal{D}_l)$$

## Information gain/entropy reduction

- For ID3, decision-tree tool developed by Quinlan, the selection of partitioning was made on the basis of the information gain/entropy reduction.
- It ran into trouble for applications having some attributes  $x_j$  with large number of possible values  $v_{l|x_j}$ ;  $l = 1, \dots, d_j$ ; giving rise to multiway splitting with many daughter nodes.
- The attribute with large number of values will get selected at root itself and may lead to all leaf nodes, resulting in a too simple hypothesis model unable to capture the structure of the data.

# Gain Ratio

- C4.5, a successor of ID3, uses an extension of information gain: gain ratio.
- It applies a kind of **normalization** to information gain using a ‘split information’ value defined analogously with Info ( $\mathcal{D}, x_j$ ) as

$$GainRatio(\mathcal{D}, x_j) = \frac{Gain(\mathcal{D}, x_j)}{SplitInfo(\mathcal{D}, x_j)}$$

$$SplitInfo(\mathcal{D}, x_j) = - \sum_{l=1}^{d_j} \frac{|\mathcal{D}_l|}{\mathcal{D}} \times \log_2 \frac{|\mathcal{D}_l|}{\mathcal{D}}$$

- The attribute with the maximum gain ratio is selected as the splitting attribute.

$$SplitInfo(\mathcal{D}, x_1) = - \sum_{l=1}^3 \frac{|\mathcal{D}_l|}{|\mathcal{D}|} \log_2 \frac{|\mathcal{D}_l|}{|\mathcal{D}|}$$

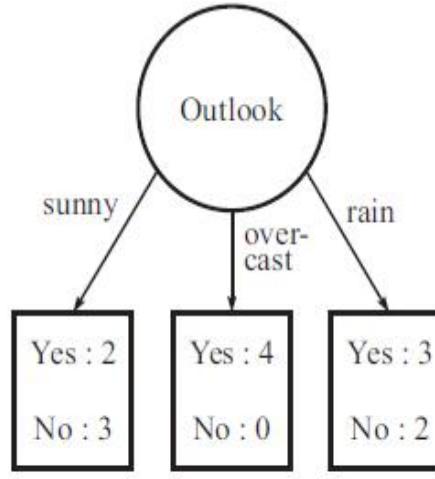
(for Outlook)

$$= - \frac{5}{14} \log_2 \frac{5}{14} - \frac{4}{14} \log_2 \frac{4}{14} - \frac{5}{14} \log_2 \frac{5}{14}$$

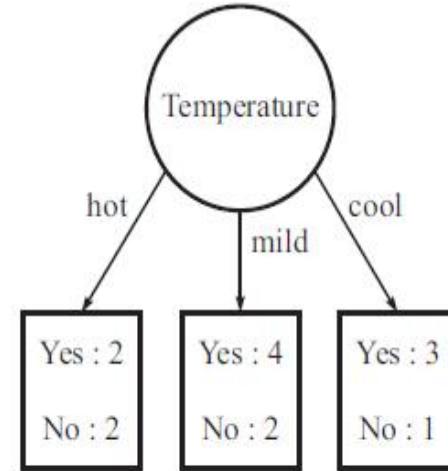
$$= 1.577$$

$$GainRatio(\mathcal{D}, x_1) = \frac{Gain(\mathcal{D}, x_1)}{SplitInfo(\mathcal{D}, x_1)}$$

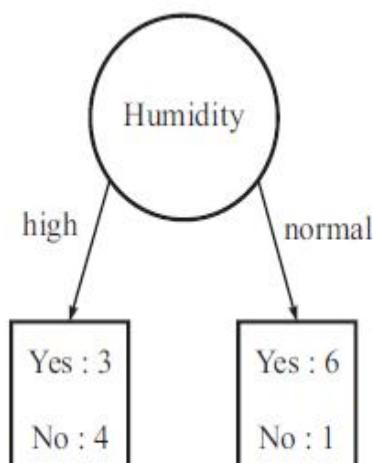
$$= \frac{0.247}{1.577} = 0.156$$



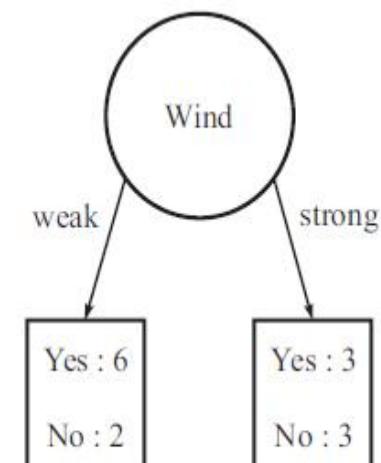
(a)



(b)



(c)



(d)

Temperature :  $Gain(\mathcal{D}, x_2) = 0.029$ ,  $SplitInfo(\mathcal{D}, x_2) = 1.362$ ,  $GainRatio(\mathcal{D}, x_2) = 0.019$

Tree stumps for the weather data

Humidity :  $Gain(\mathcal{D}, x_3) = 0.152$ ,  $SplitInfo(\mathcal{D}, x_3) = 1.000$ ,  $GainRatio(\mathcal{D}, x_3) = 0.152$

**Humidity now is a much closer contender because it splits the data into two subsets instead of three.**

Wind :  $Gain(\mathcal{D}, x_4) = 0.048$ ,  $SplitInfo(\mathcal{D}, x_4) = 0.985$ ,  $GainRatio(\mathcal{D}, x_4) = 0.049$

# Gini Index

- Gini Index is used in CART (Classification and Regression Tree).

$$Gini(\mathcal{D}) = 1 - \sum_{q=1}^M P_q^2$$

Where  $P_q$  is the probability that a tuple in  $\mathcal{D}$  belongs to class  $y_q$ , and is estimated by

$$P_q = \frac{freq(y_q, \mathcal{D})}{|\mathcal{D}|}$$

- Gini index considers a binary split for each attribute.
- First consider the case where  $x_j$  is continuous-valued attribute having  $d_j$  distinct values;  $l = 1, 2, \dots, d_j$ .
- It is common to take mid-point between each pair of (sorted) adjacent values as a possible split-point.
- The point giving the **minimum** Gini index for the attribute  $x_j$  is taken as its split point.

- For a possible split-point of  $x_j$ ,  $\mathcal{D}_l$  is the number of tuples in  $\mathcal{D}$  satisfying  $x_j \leq \text{split\_point}$ , and  $\mathcal{D}_2$  is the set of tuples satisfying  $x_j \geq \text{split\_point}$ .
- The reduction in impurity that would be incurred by a binary split on  $x_j$  is:

$$\Delta Gini(x_j) = Gini(\mathcal{D}) - Gini(\mathcal{D}, x_j)$$

$$Gini(\mathcal{D}, x_j) = \frac{|\mathcal{D}_1|}{\mathcal{D}} Gini(\mathcal{D}_1) + \frac{|\mathcal{D}_2|}{\mathcal{D}} Gini(\mathcal{D}_2)$$

- The attribute that maximized the reduction in impurity is selected as the splitting attribute.
- Then one of these two parts ( $\mathcal{D}_1, \mathcal{D}_2$ ) is divided in a similar manner by choosing a variable again and a split value for the variable.
- This process is continued till we get pure leaf nodes.

- To determine the best binary split on Outlook, we examine all possible subsets that can be formed using categories of Outlook: {sunny, overcast, rain}, {sunny, overcast}, {sunny, rain}, {overcast, rain}, {sunny}, {overcast}, {rain}, and {}.
- We exclude the powerset {sunny, overcast, rain}, and the empty set {} from consideration since, conceptually, they do not represent a split.
- Therefore, there are  $2^\gamma - 2$  possible ways to form two partitions of dataset D, based on the binary splits on  $x_j$  having  $\gamma$  categorical values.
- Each of the possible binary splits is considered; the subset that gives the minimum Gini index for attribute  $x_j$  is selected as its splitting subset.

# Numerical on Decision Tree

# The CART Decision Tree

- CART: Classification and Regression Tree
- a lawn-tractor manufacturer would like to find a way of classifying households in a city into:
  - those likely to purchase a lawn tractor
  - those not likely to buy one.

$s^{(i)}$	$x_1$	$x_2$	
1	60	18.4	Owner
2	75	19.6	Nonowner
3	85.5	16.8	Owner
4	52.8	20.8	Nonowner
5	64.8	21.6	Owner
6	64.8	17.2	Nonowner
7	61.5	20.8	Owner
8	43.2	20.4	Nonowner
9	87	23.6	Owner
10	84	17.6	Nonowner
11	110.1	19.2	Owner
12	49.2	17.6	Nonowner
13	108	17.6	Owner
14	59.2	16	Nonowner
15	82.8	22.4	Owner
16	66	18.4	Nonowner
17	69	20	Owner
18	47.4	16.4	Nonowner
19	93	20.8	Owner
20	33	18.8	Nonowner
21	51	22	Owner
21	51	14	Nonowner
23	81	20	Owner
24	63	14.8	Nonowner

Random Sample of Households in a city with respect to ownership of a lawn tractor

# The CART Decision Tree

- Input variables:
  - $x_1$  = Income
  - $x_2$  = Lawn size
- they are recorded for 24 households;
- the target variable:
- $y$  = Ownership of a lawn tractor
- it is assigned to each household.

$s^{(i)}$	$x_1$	$x_2$	
1	60	18.4	Owner
2	75	19.6	Nonowner
3	85.5	16.8	Owner
4	52.8	20.8	Nonowner
5	64.8	21.6	Owner
6	64.8	17.2	Nonowner
7	61.5	20.8	Owner
8	43.2	20.4	Nonowner
9	87	23.6	Owner
10	84	17.6	Nonowner
11	110.1	19.2	Owner
12	49.2	17.6	Nonowner
13	108	17.6	Owner
14	59.2	16	Nonowner
15	82.8	22.4	Owner
16	66	18.4	Nonowner
17	69	20	Owner
18	47.4	16.4	Nonowner
19	93	20.8	Owner
20	33	18.8	Nonowner
21	51	22	Owner
21	51	14	Nonowner
23	81	20	Owner
24	63	14.8	Nonowner

Random Sample of Households in a city with respect to ownership of a lawn tractor

## Finding midpoints

- When searching for a binary split on a continuous-valued input variable, midpoints between the consecutive values may be treated as candidate values for the split.
- That means, after sorting  $x_1$  in ascending order, we get  $\{33, 43.2, 47.4, \dots, 108, 110.1\}$ , the midpoints between the consecutive values need to be calculated.
  - for example, midpoint between 33 and 43.2 is 38.1; and that between 43.2 and 47.4 is 45.3.
- Thus, the candidate split points for the variable  $x_1$  (Income) are  $\{38.1, 45.3, 50.1, \dots, 109.5\}$ ,
- similarly, the candidate split points for  $x_2$  (Lawn size) are  $\{14.4, 15.4, 16.2, \dots, 23\}$ .

## Finding midpoints

33	x1
43.2	38.1
47.4	45.3
49.2	48.3
51	50.1
51	51
52.8	51.9
59.2	56
60	59.6
61.5	60.75
63	62.25
64.8	63.9
64.8	64.8
66	65.4
69	67.5
75	72
81	78
82.8	81.9
84	83.4
85.5	84.75
87	86.25
93	90
108	100.5
110.1	109.05

14	x2
14.8	14.4
16	15.4
16.4	16.2
16.8	16.6
17.2	17
17.6	17.4
17.6	17.6
17.6	17.6
18.4	18
18.4	18.4
18.8	18.6
19.2	19
19.6	19.4
20	19.8
20	20
20.4	20.2
20.8	20.6
20.8	20.8
20.8	20.8
21.6	21.2
22	21.8
22.4	22.2
23.6	23

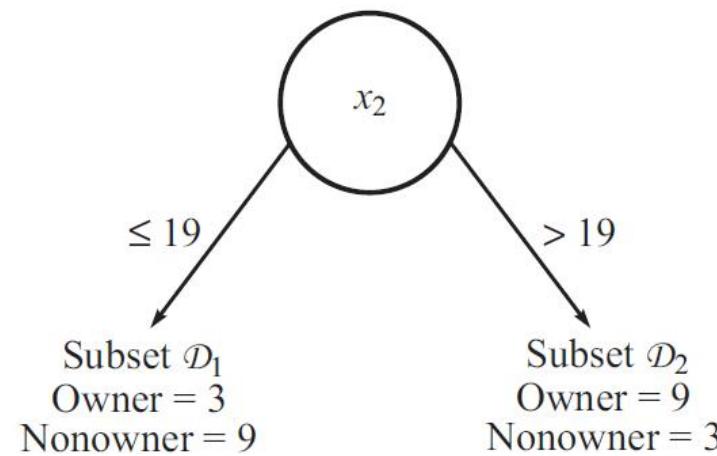
## Calculate Gini index and rank the candidate split points

- We need to rank the candidate split points according to how much they reduce impurity (heterogeneity) in the resulting subsets after the split.
- With respect to Gini index as impurity measure,

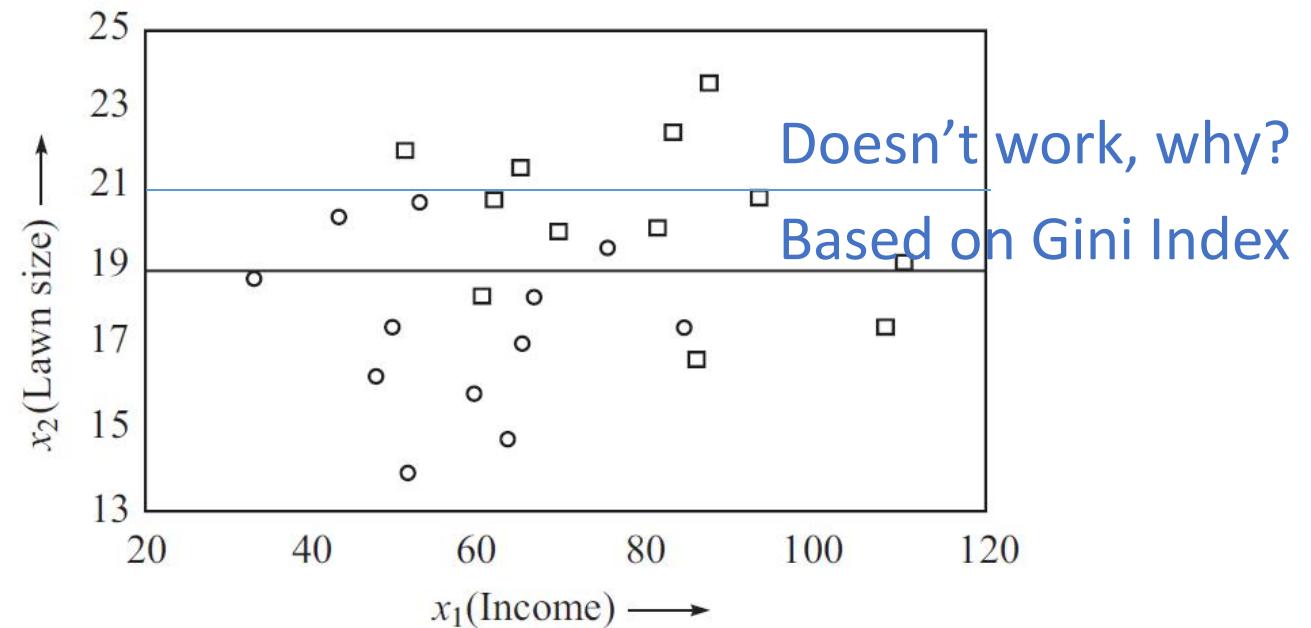
$$\begin{aligned}Gini(\mathcal{D}) &= 1 - \sum_{q=1}^2 P_q^2; P_q = \frac{freq(y_q, \mathcal{D})}{|\mathcal{D}|} \\&= 1 - (0.5)^2 - (0.5)^2 \\&= 0.5\end{aligned}$$

- Note that the Gini index impurity measure is at its peak when  $P_q = 0.5$ , i.e., when the data is perfectly balanced (containing equal numbers of Owner/Nonowner households).
- Calculate Gini index for all the candidate split points for both  $x_1$  and  $x_2$  variables, and rank them according to how much they reduce impurity

Try splitting at 21.2



Tree stumps after first split



Scatter plot after first split

- we choose  $x_2$  (Lawn size) for the first split with a splitting value of 19 (mid point for entire dataset w.r.t  $x_2$ ).
- The  $(x_1, x_2)$  space is now divided into two rectangles, one with  $x_2 \leq 19$  and the other with  $x_2 > 19$ .

$$\begin{aligned}
 Gini(\mathcal{D}, x_2) &= \frac{|\mathcal{D}_1|}{|\mathcal{D}|} \times Gini(\mathcal{D}_1) + \frac{|\mathcal{D}_2|}{|\mathcal{D}|} \times Gini(\mathcal{D}_2) \\
 &= \frac{12}{24} \times Gini(\mathcal{D}_1) + \frac{12}{24} \times Gini(\mathcal{D}_2) \\
 &= \frac{12}{24} \left(1 - \left(\frac{3}{12}\right)^2 - \left(\frac{9}{12}\right)^2\right) + \frac{12}{24} \left(1 - \left(\frac{9}{12}\right)^2 - \left(\frac{3}{12}\right)^2\right) \\
 &= \frac{1}{2} \times 0.375 + \frac{1}{2} \times 0.375 = 0.375
 \end{aligned}$$

- Thus, the Gini impurity index decreased from 0.5 before the split to 0.375 after the split.
- Each of the rectangles created by the split is more homogeneous than the rectangle before the split.
- The upper rectangle contains points that are mostly Owners and the lower rectangle contains mostly Nonowners.

Try splitting at 21.2

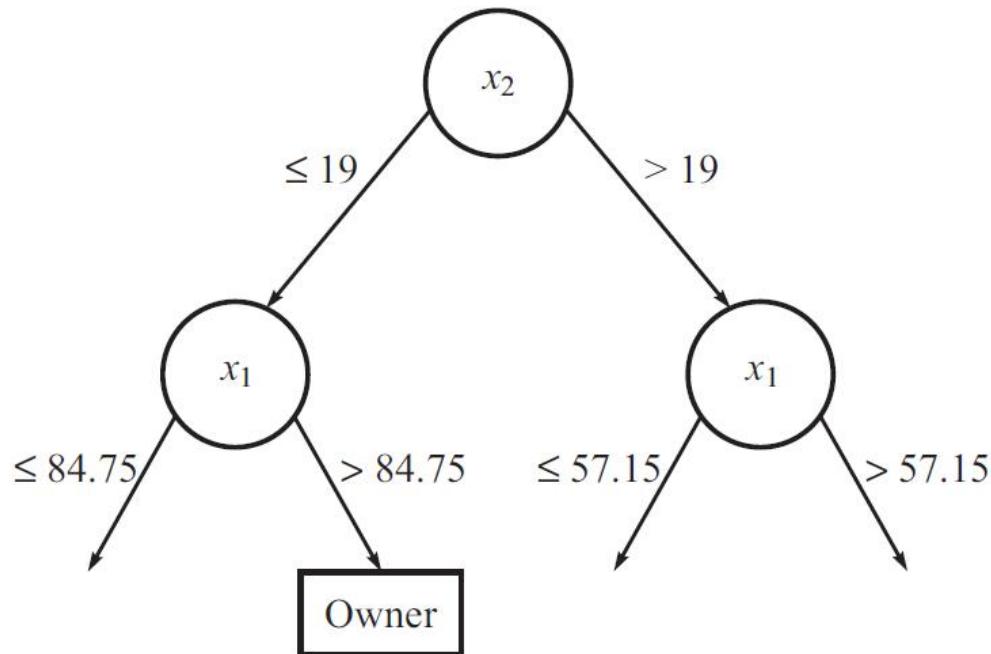
$$\begin{aligned}
 &\frac{4}{24} \left(1 - \left(\frac{4}{4}\right)^2\right) + \frac{20}{24} \left(1 - \left(\frac{12}{20}\right)^2 - \left(\frac{8}{20}\right)^2\right) \\
 &= \frac{5}{6} \left(1 - \frac{9}{25} - \frac{4}{25}\right) \\
 &= \frac{5}{6} \left(1 - \frac{13}{25}\right) \\
 &= \frac{5}{6} \times \frac{12}{25} = 0.4
 \end{aligned}$$

- Notice that 19 is the best splitting value.
- Following is the formula, if you want to calculate GINI Index in Excel:
 

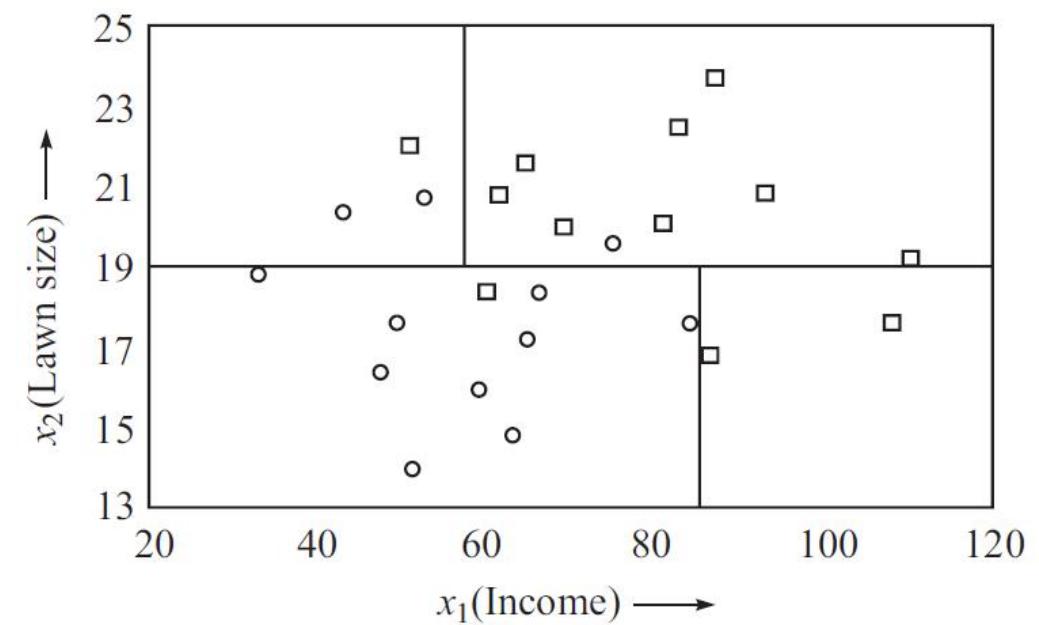
```
= (COUNTA(F$2:F13)/24) * (1 - ((COUNTIF(F$2:F13,"Owner")/COUNTA(F$2:F13))^2) - ((COUNTIF(F$2:F13,"Nonowner")/COUNTA(F$2:F13))^2)) + (COUNTA(F14:F$25)/24) * (1 - ((COUNTIF(F14:F$25,"Owner")/COUNTA(F14:F$25))^2) - ((COUNTIF(F14:F$25,"Nonowner")/COUNTA(F14:F$25))^2))
```

	A	B	C	D	E	F
1	S	x1	x2	mid_x2	y	
2	20	33	14		Nonowner	
3	8	43.2	14.8	14.4	Nonowner	0.478
4	18	47.4	16	15.4	Nonowner	0.455
5	12	49.2	16.4	16.2	Nonowner	0.429
6	21	51	16.8	16.6	Owner	0.400
7	21	51	17.2	17	Nonowner	0.453
8	4	52.8	17.6	17.4	Nonowner	0.426
9	14	59.2	17.6	17.6	Nonowner	
10	1	60	17.6	17.6	Owner	
11	7	61.5	18.4	18	Owner	0.407
12	24	63	18.4	18.4	Nonowner	0.443
13	5	64.8	18.8	18.6	Nonowner	0.413
14	6	64.8	19.2	19	Owner	0.375
15	16	66	19.6	19.4	Nonowner	0.413
16	17	69	20	19.8	Owner	
17	2	75	20	20	Owner	0.407
18	23	81	20.4	20.2	Nonowner	0.438
19	15	82.8	20.8	20.6	Nonowner	0.395
20	10	84	20.8	20.8	Owner	
21	3	85.5	20.8	20.8	Owner	
22	9	87	21.6	21.2	Owner	0.400
23	19	93	22	21.8	Owner	0.429
24	13	108	22.4	22.2	Owner	0.455
25	11	110.1	23.6	23	Owner	0.478

- The next split is found to be on the  $x_1$  (Income) variable at the value 84.75 to increase the purity of the resulting rectangles.
- The left lower rectangle ( $x_1 \leq 84.75$ ,  $x_2 \leq 19$ ) has all points that are Nonowners with one exception, whereas the right lower rectangle ( $x_1 > 84.75$ ,  $x_2 \leq 19$ ) consists exclusively of Owners.



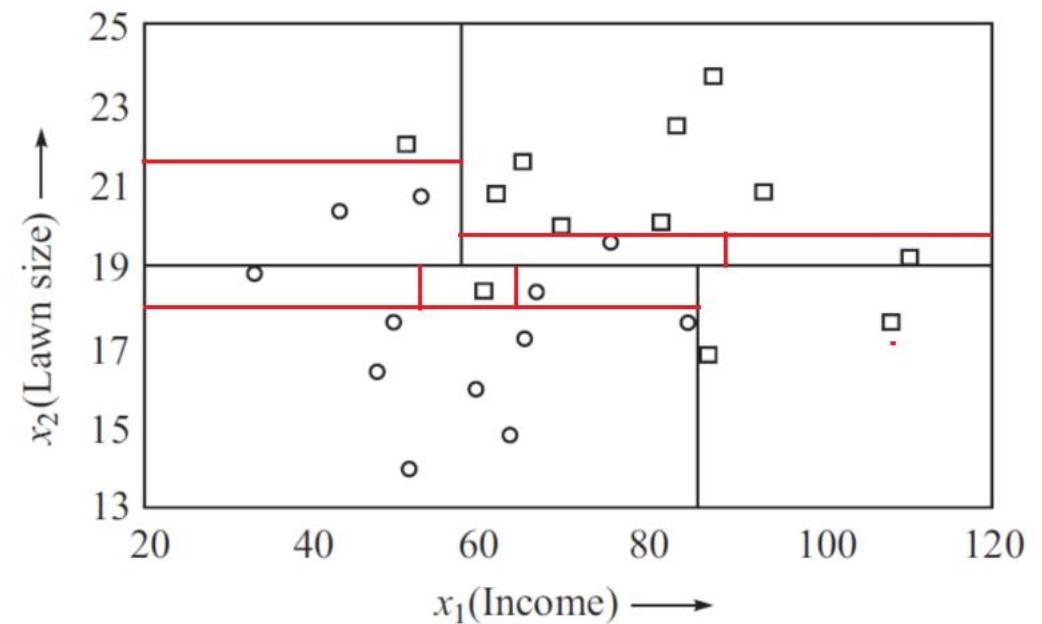
Tree stumps after first three splits



Scatter plot after first three splits

- If the partitioning is continued till all the branches hit leaf nodes, each rectangle will have data points from just one of the two classes

- There are so many other ways to write an algorithm and build a tree using same GINI Index concept.
- There are researchers throughout the world working on modifying the algorithms to get better results.
- **Solutions are not unique (see next slide).**



Probable final tree

- If the partitioning is continued till all the branches hit leaf nodes, each rectangle will have data points from just one of the two classes

- You may Choose  $x_1$  mid points for first split.
- By comparing the reduction in impurity across all possible splits in all possible attributes, the next split is chosen.
- For example, if  $x_1 \leq 64.8$  is chosen:

$$\begin{aligned}
 Gini(D, x_1) &= \frac{12}{24} \times Gini(D_1) + \frac{12}{24} \times Gini(D_2) \\
 &= \frac{12}{24} \left(1 - \left(\frac{4}{12}\right)^2 - \left(\frac{8}{12}\right)^2\right) + \frac{12}{24} \left(1 - \left(\frac{8}{12}\right)^2 - \left(\frac{4}{12}\right)^2\right) \\
 &= \frac{1}{2} \times 0.444 + \frac{1}{2} \times 0.444 = 0.444
 \end{aligned}$$

- You may Choose 78 or 59.6 as first split according to the Gini index values calculated.
- It gives a completely different solution.

s	$x_1$	mid_x1	$x_2$	y	
20	33		18.8	Nonowner	0.478
8	43.2	38.1	20.4	Nonowner	0.455
18	47.4	45.3	16.4	Nonowner	0.429
12	49.2	48.3	17.6	Nonowner	0.400
21	51	50.1	22	Owner	0.453
21	51	51	14	Nonowner	0.426
4	52.8	51.9	20.8	Nonowner	0.395
14	59.2	56	16	Nonowner	0.359
1	60	59.6	18.4	Owner	0.407
7	61.5	60.75	20.8	Owner	0.443
24	63	62.25	14.8	Nonowner	0.413
5	64.8	63.9	21.6	Owner	0.444
6	64.8	64.8	17.2	Nonowner	0.413
16	66	65.4	18.4	Nonowner	0.371
17	69	67.5	20	Owner	0.407
2	75	72	19.6	Nonowner	0.359
23	81	78	20	Owner	0.426
15	82.8	81.9	22.4	Owner	0.368
10	84	83.4	17.6	Nonowner	0.400
3	85.5	84.75	16.8	Owner	0.429
9	87	86.25	23.6	Owner	0.455
19	93	90	20.8	Owner	0.478
13	108	100.5	17.6	Owner	
11	110.1	109.05	19.2	Owner	

# Stopping Criteria in Tree and Over-fitting Avoidance

# Pruning the Tree

- Elimination of any leaf nodes will simply lead to an increase in the error rate of the tree on the training set.
- But this certainly does not mean that the entire tree with pure leaf nodes also performs the best on new data!
- The strategy of building tree can result in problems when there is random noise in the data.
- It is said that a decision tree overfits the training examples if there is some other tree that doesn't fit the training examples that well actually ends up performing better over the entire distribution of instances (including instances beyond the training set)

# Pruning the Tree

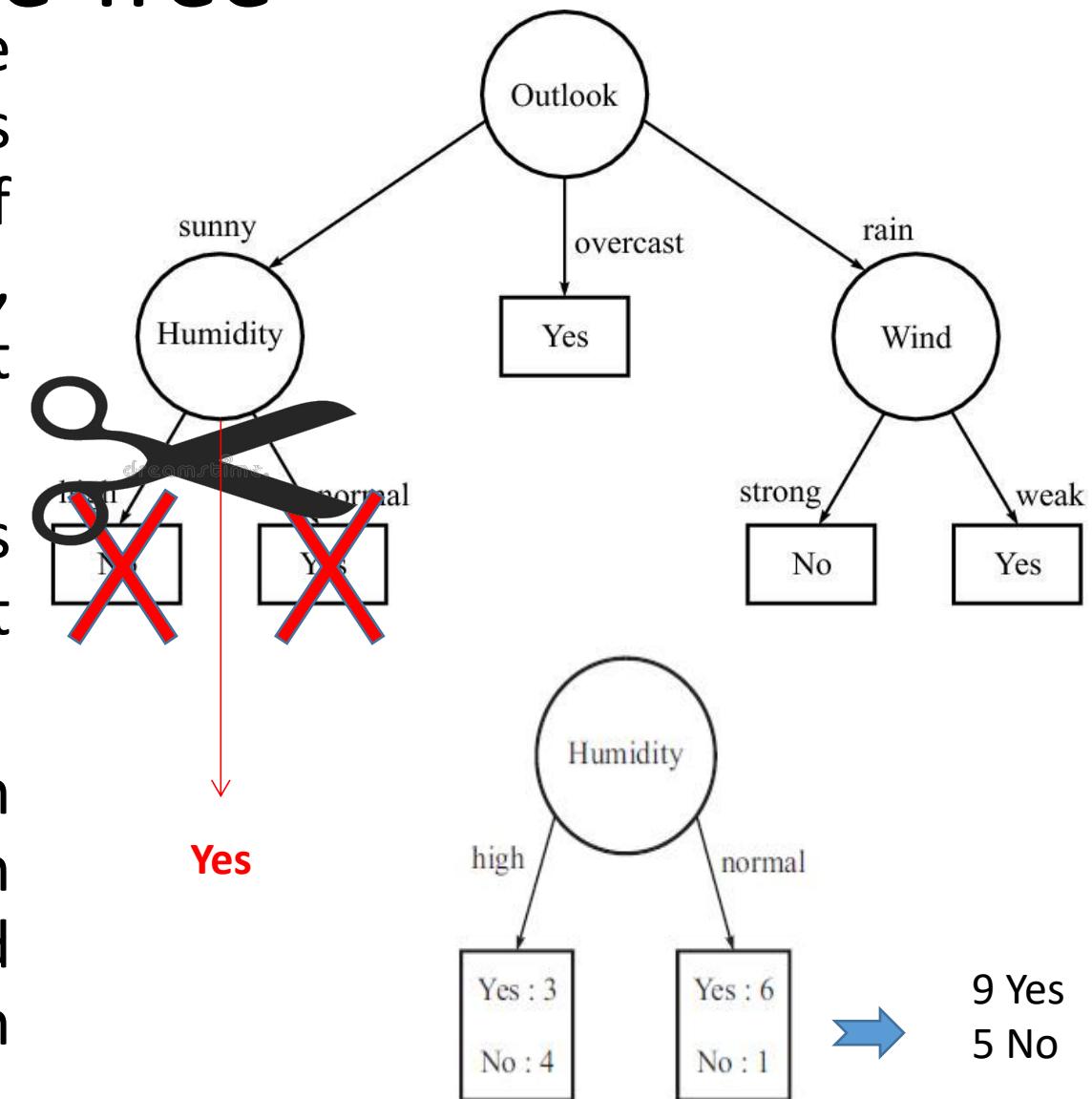
- To avoid overfitting in decision tree learning:

Two main groups:

- Prepruning: Growing of the tree is stopped before it reaches the point where it perfectly classifies the training data.
- Postpruning: The tree is allowed to grow to perfectly classify the training examples, and then post-prune is done.
- The second approach of postpruning overfit trees has been found to be more successful in practice.
- This is because the first approach finds it difficult to estimate accurately as to when the growing of the tree can be stopped.
- weakest branches of large tree overfitting the training data, which hardly reduce the error rate, are to be removed.

# Pruning the Tree

- If the partitioning of tuples at a node results in a division that meets prespecified threshold (may be in terms of Information Gain/Gain Ratio/Gini Index), then no more partitioning of the subset takes place.
- On stopping the growth, the node turns into a leaf, which may hold the most frequent class among the subset tuples.
- However, it is not easy to select an appropriate threshold. While high thresholds may lead to oversimplified trees, low thresholds may hardly result in any simplification at all.



# Pruning the Tree

What is the criterion for deciding on the right final tree size?

- Make use of the training and test set approach, wherein data available is separated into two groups of examples:
  - (i) a training set, with the help of which the learned decision tree is formed, and
  - (ii) a separate test set, with the help of which the evaluation of accuracy of this tree over subsequent data is done, especially, evaluation of the effect of pruning this tree (The approach is effective when huge volumes of data are available).
- Instead of pruning trees on the basis of estimated error rates, we can prune trees according to adjusted error rate, which is equal to the misclassification error of a tree plus a penalty factor on the tree size.

# Pruning the Tree

- The CART algorithm recognizes a group of such subtrees as candidate models.
- On applying these candidate subtrees to the test set, the tree showing the lowest test set **misclassification** error is chosen as the final model.
- The chosen model may have leaf nodes, which are impure, whereas the original tree had pure leaf nodes.

		Predicted +	Predicted -	
Target +	True Positive	a	b	False Negative
	False Positive	c	d	True Negative
Target -				

$a + d = \text{good predictions}$   
 $b + c = \text{bad predictions}$

# Pruning the Tree

## Cost complexity measure?

- It is used to identify weak branches and mark them for pruning.
- For a tree  $T$  that has  $L(T)$  leaf nodes, the cost complexity can be written as

$$CC(T) = Err(T) + \alpha L(T)$$

- where  $Err(T)$  is the fraction of training data observations that are misclassified by tree  $T$  and  $\alpha$  is a penalty factor for tree size.
- When  $\alpha = 0$ , the fully grown unpruned tree is the best tree.
- If  $\alpha$  is too large, the best tree may merely be the tree with the root node.
- Raise the penalty factor slowly (from zero) till a point is reached where the  $CC(T)$  is more than that of the subtree obtained by replacing an internal node at the next higher level with a leaf node.

# Pruning the Tree

Cost complexity measure?

- The same process is then repeated on the subtree.
- The process continues and as a result, a succession of trees is generated with a decreasing number of nodes.
- From the sequence of trees, it appears natural to select the one that gives the least misclassification error on the test dataset.

# Ensemble Learning and Random Forest

# Ensemble Learning

- For the given amount and quality of training data, the output of one hypothesis function may be inappropriate for the problem at hand.
- The ideal model to make more reliable decisions is to create a combination of outputs of many different hypotheses.
- Many machine learning algorithms do this by learning an ensemble of hypothesis and employing them in a combined form.
- Bagging and boosting are the most frequently used among these schemes.
- These general methods may be applied to classification and regression problems
- They frequently increase predictive performance over a single hypothesis.

# Ensemble Learning

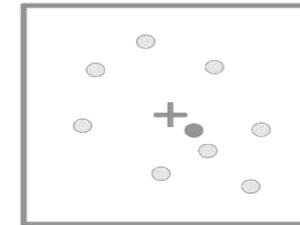
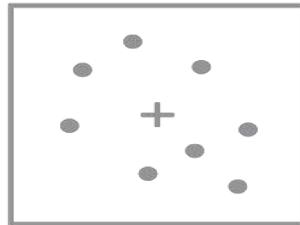
- By combining the decisions of various hypotheses, we amalgamate the different outputs into a single prediction.
- For classification problems, it is done through voting
  - may be a weighted vote
- whereas in case of regression problems, the average is computed
  - may be a weighted average

# Bagging and Boosting

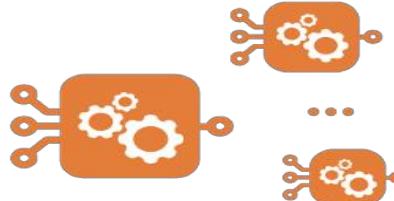
- In the bagging technique, individual approaches are constructed separately,
- whereas in boosting, each new model is impacted by the performance of those built earlier.
- Ada Boost (an abbreviation for Adaptive Boosting) is a widely used algorithm for boosting.
- Bagging is the most straightforward and basic technique of pooling or integrating the outputs of component classifiers.

+ target      ● predicted      ..... error

*predictions  
visualisation*

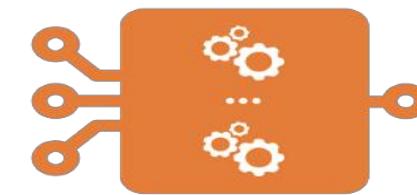


*models  
representation*



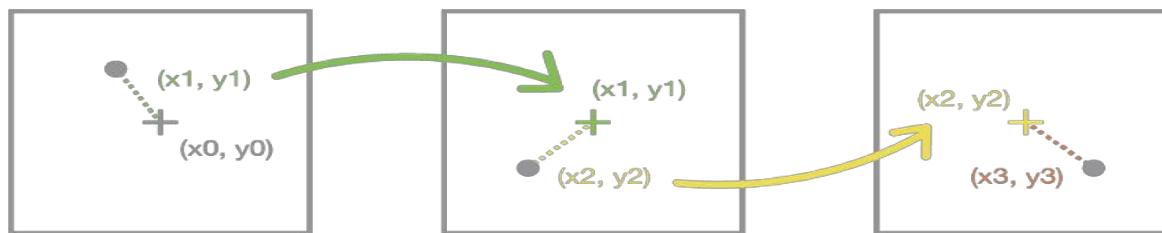
several single weak learners  
with **low bias but high variance**

“average”  
weak learners

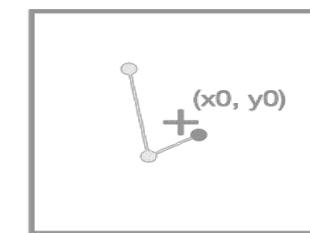


ensemble model with a **lower variance than its components**

### LOW BIAS HIGH VARIANCE WEAK LEARNERS



### LOW VARIANCE HIGH BIAS WEAK LEARNERS



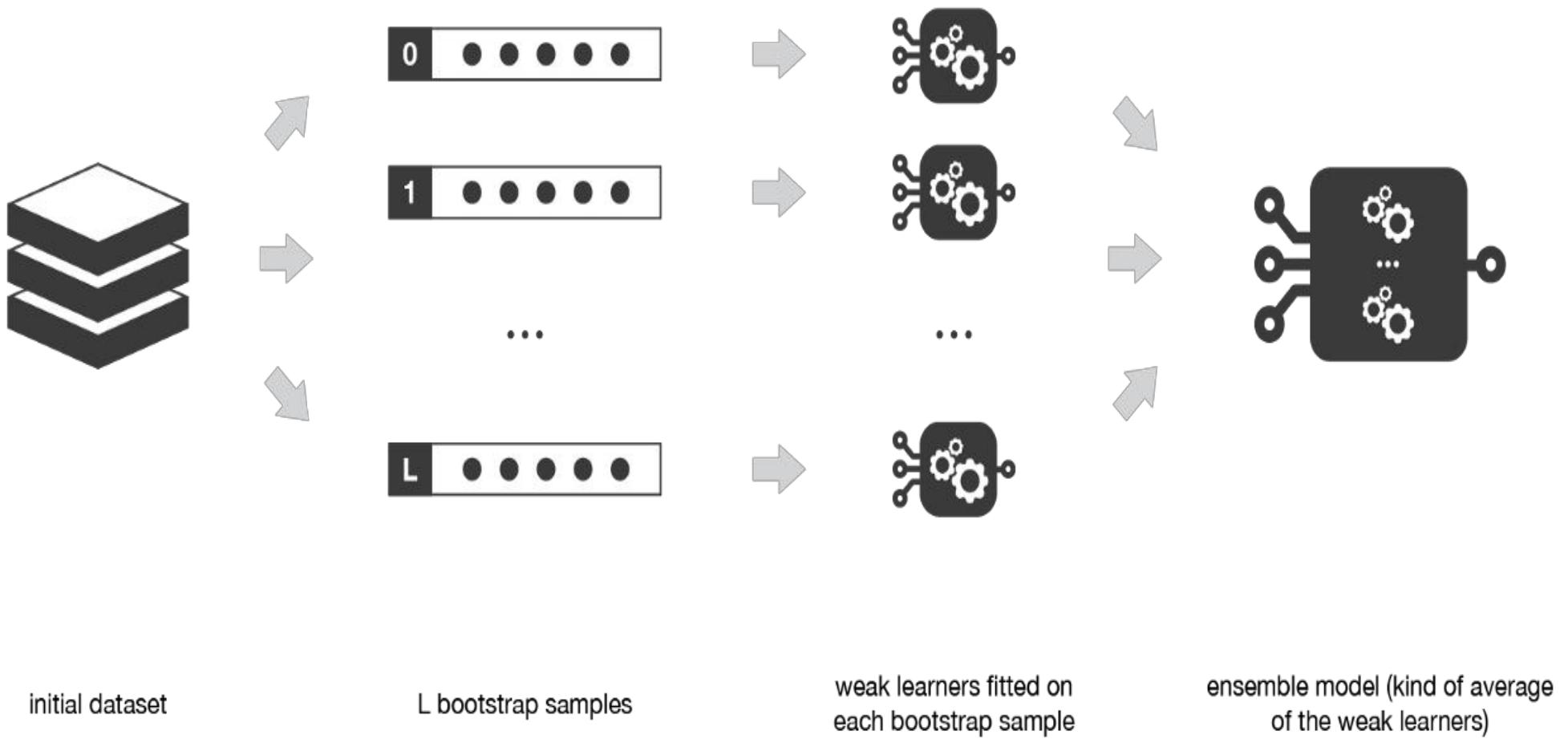
“compose”  
weak learners



several single weak learners with **high bias but low variance**:  
each model target the error of the previous one

ensemble model with a **lower bias than its components**

# Bagging



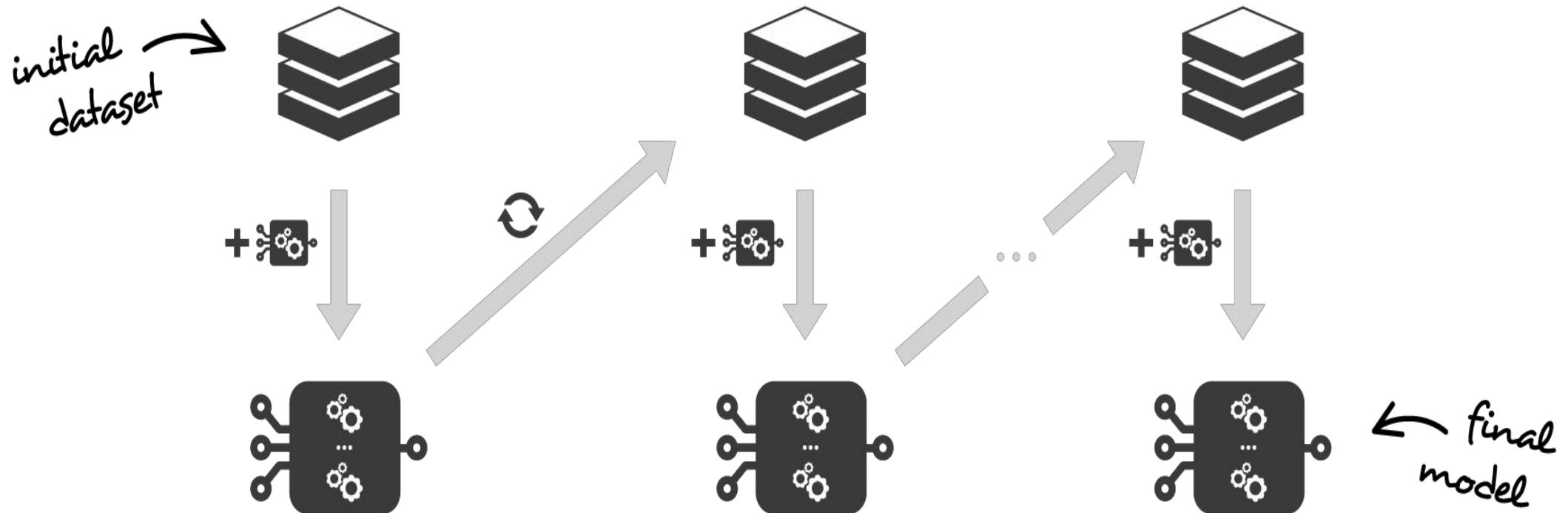
# Boosting



train a weak model  
and aggregate it to  
the ensemble model

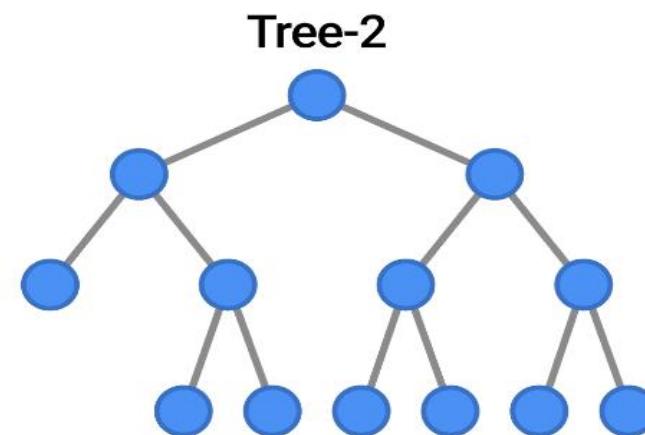
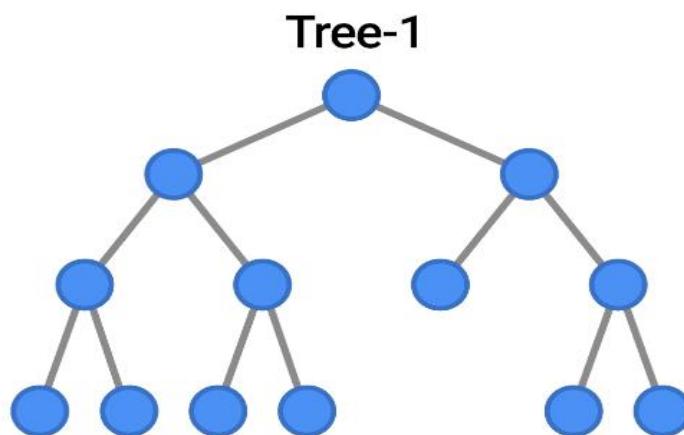


update the training dataset  
(values or weights) based on the  
current ensemble model results

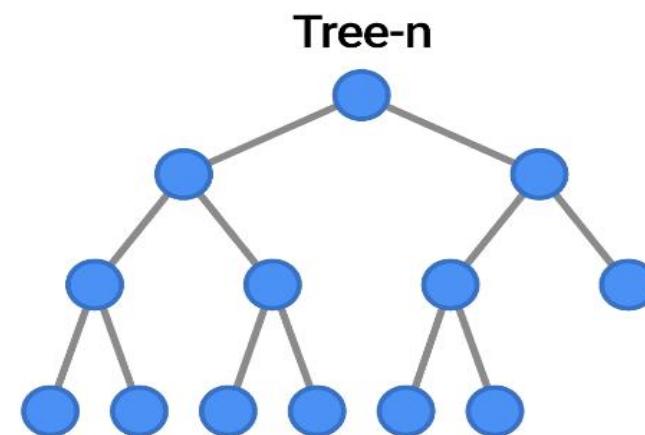


# Random Forests

- Random Forests are a popular type of decision forest model.
  - You can build random forests with the help of bagging in tandem (cascaded) with the random attribute selection.
  - Here, you can see a forest of trees classifying an example by voting on the outcome.



• • •



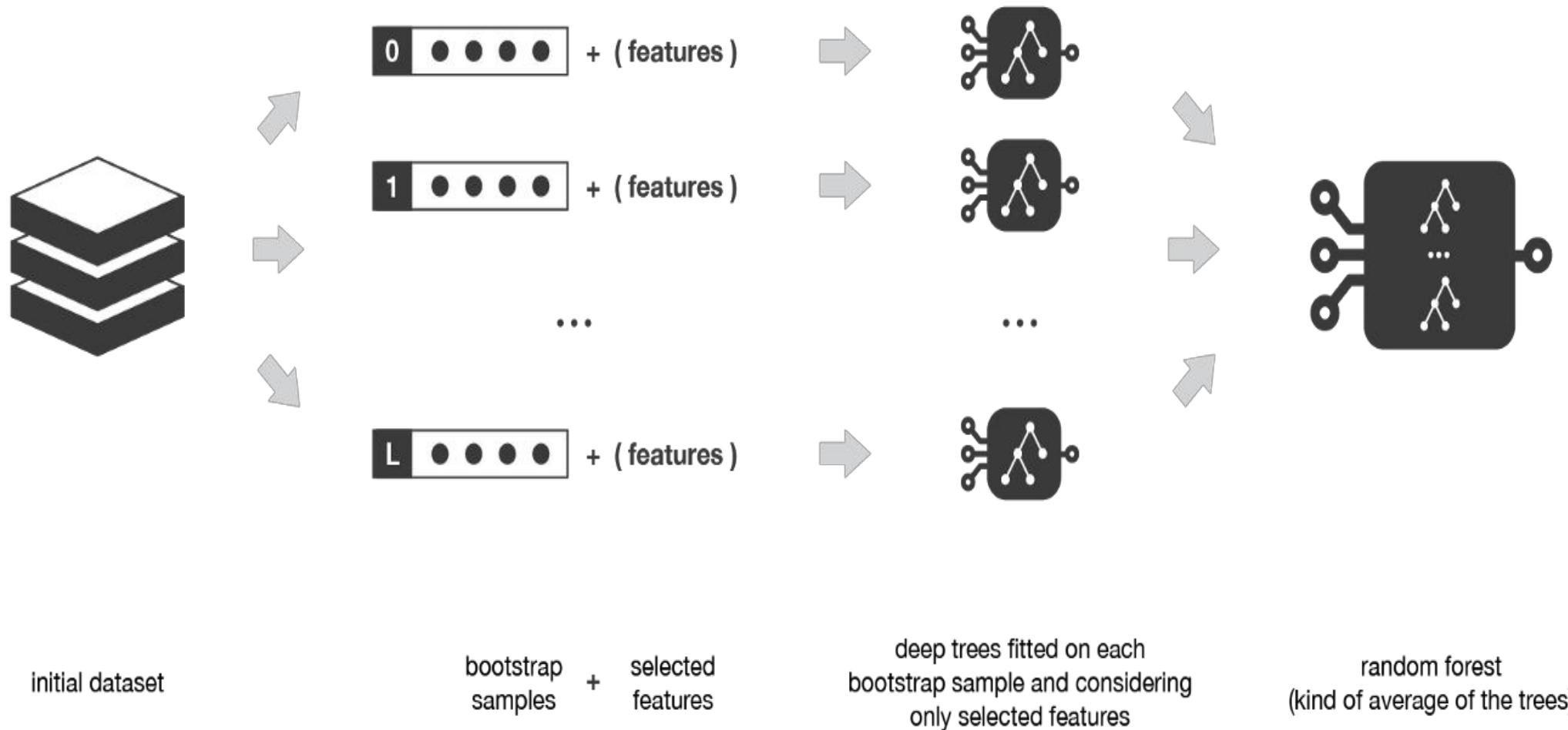
# Random Forests

- Trees that compose a forest can be chosen to be either shallow (few depths) or deep
  - lot of depths, if not fully grown
- Shallow trees have less variance but higher bias.
- Deep trees, on the other side, have low bias but high variance and, so, are relevant choices for bagging method that is mainly focused at reducing variance.
- When training, each tree in a random forest learns from a random sample of the data points.
- The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree.

# Random Forests

- Random forests also use another trick to make the multiple fitted trees a bit less correlated with each others:
- when growing each tree, instead of only sampling over the observations in the dataset to generate a bootstrap sample, we also sample over features and keep only a random subset of them to build the tree.
- Sampling over features has indeed the effect that all trees do not look at the exact same information to make their decisions and, so, it reduces the correlation between the different returned outputs.
- Another advantage of sampling over the features is that it makes the decision making process more robust to missing data:

# Random Forests



# Adaboost



train a weak model  
and aggregate it to  
the ensemble model



update the weights of  
observations misclassified by  
the current ensemble model

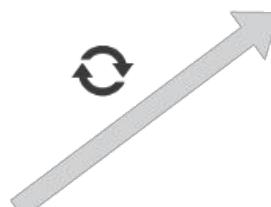
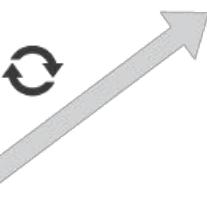
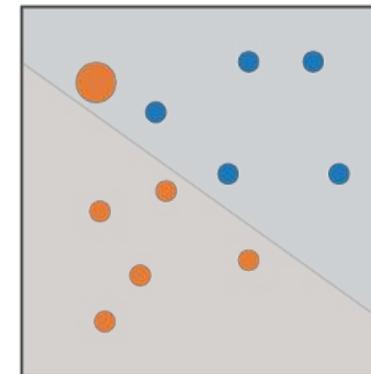
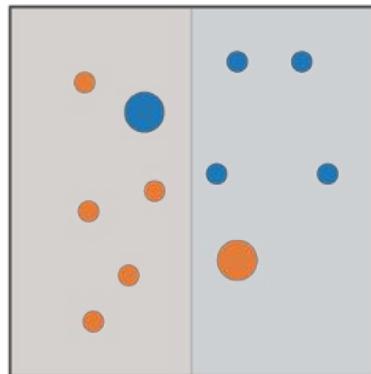
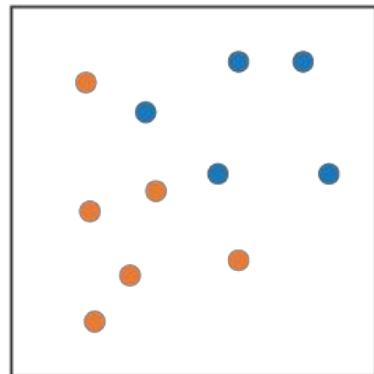


current ensemble model  
predicts “orange” class



current ensemble model  
predicts “blue” class

initial setting:  
all the  
observations  
have the  
same weight



...



# Bagging; Boosting; Random Forests; Adaboost

- Bagging consists in fitting several base models on different bootstrap samples and build an ensemble model that “average” the results of these weak learners.
- Boosting: Weak learners can be combined to get a model with better performances.
- Random forest method is a bagging method with trees as weak learners. Each tree is fitted on a bootstrap sample considering only a subset of variables randomly chosen.
- Adaboost updates weights of the observations at each iteration. Weights of well classified observations decrease relatively to weights of misclassified observations. Models that perform better have higher weights in the final ensemble model.

# PCA NAIVE BAYES

Dr. Srikanth Allamsetty

# Principal Component Analysis

# Introduction

- Principal Components Analysis (PCA) is abundantly used for reducing number of variables (features/attributes) in the dataset that are correlated.
- The reduced set of variables are weighted linear combinations of the original variables that (approximately) retain the information content of the original dataset.
- The linear combinations of the original variables lose physical meaning (if any) attached to the original variables.
- The goal of PCA is to **transform** data in the best possible way.
- The potential problems with the data are **noise** and **redundancy**

# Dealing with redundancy

- A simple way to quantify the redundancy between individual attributes is through the covariance matrix  $\Sigma$ .
- Covariance matrix describes all relationships between pairs of attributes in our dataset.
- If our goal is to reduce redundancy, we would like each variable to co-vary as little as possible, preferably zero covariance, with other variables.
- Evidently, in an ‘optimized’ matrix, all off-diagonal terms in  $\Sigma$  are zero.
- Therefore, removing redundancy **diagonalizes  $\Sigma$** .
- There are many methods for diagonalizing  $\Sigma$ ; PCA selects the easiest method based on linear algebra.

# Dealing with noise

- There exists no **absolute** scale for noise but rather all noise is measured relative to the signal.
- A common measure is signal-to-noise ratio (SNR), which is the ratio of variances:

$$\text{SNR} = \frac{\sigma_{\text{signal}}^2}{\sigma_{\text{noise}}^2}$$

- A high SNR ( $>>1$ ) indicates high precision data, while a low SNR indicates noise contaminated data.
- The data with high SNR will have attributes with **larger associated variances**, representing interesting dynamics.
- The data with low SNR will have attributes with lower variances, representing noise.

# linear algebra solution using PCA

- The solution is based on transformation of the covariance matrix to a diagonal form with **larger associated variances**.
- PCA is an unsupervised method as it does not use the output information.
- We are interested in finding a mapping from the inputs in the original n-dimensional space to a new ( $k < n$ )-dimensional space with the minimum loss of information.
- The **final** step in PCA is to derive the new dataset—transformation of the original dataset

$$\hat{\mathbf{Z}}_{(k \times N)} = \mathbf{W}_{(k \times n)} \hat{\mathbf{X}}_{(n \times N)}$$

How? 

# Dimensionality Reduction



# Basics of linear algebra

The *mean (average)* of the attribute  $x_j$  is given by,

$$\mu_j = \frac{\sum_{i=1}^N x_j^{(i)}}{N}; j = 1, \dots, n$$

Variance (a measure of the spread of the data) is given by,

$$var(x_j) = \sigma_j^2 = \frac{\sum_{i=1}^N (x_j^{(i)} - \mu_j)(x_j^{(i)} - \mu_j)}{N}$$

$$= \frac{\sum_{i=1}^N (x_j^{(i)} - \mu_j)^2}{N}; j = 1, \dots, n$$

Covariance (a measure to find out how attributes vary *with respect to each other*)

$$cov(x_j, x_l) = \sigma_{jl} = \frac{\sum_{i=1}^N (x_j^{(i)} - \mu_j)(x_l^{(i)} - \mu_l)}{N}; j, l = 1, \dots, n$$

For the attribute vector  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$  with mean  $\boldsymbol{\mu} = [\mu_1 \ \mu_2 \ \dots \ \mu_n]^T$ , the *covariance matrix*

$$\Sigma = \frac{\sum_{i=1}^N (\mathbf{x}^{(i)} - \boldsymbol{\mu})(\mathbf{x}^{(i)} - \boldsymbol{\mu})^T}{N}$$

$$= \begin{bmatrix} \sigma_1^2 & \sigma_{12} & \cdots & \sigma_{1n} \\ \sigma_{21} & \sigma_2^2 & \cdots & \sigma_{2n} \\ \vdots & \vdots & & \vdots \\ \sigma_{n1} & \sigma_{n2} & \cdots & \sigma_n^2 \end{bmatrix} \quad (7.52e)$$

Note that  $\Sigma$  is an  $n \times n$  square matrix and also symmetric matrix.

- When we subtract the mean from each of the data dimensions, we get the transformed data in **mean-deviation form**.

# Covariance

- We assume that the given dataset is a zero-mean dataset.

$$\Sigma_{\mathbf{X}} = \frac{\sum_{i=1}^N \mathbf{x}^{(i)} (\mathbf{x}^{(i)})^T}{N} = \frac{1}{N} \mathbf{X}^T \mathbf{X}$$

here,  $\mathbf{X}$  is  $N \times n$  data matrix

- Working with an  $n \times N$  data matrix is more convenient; thus a new data matrix is defined as follows  $\hat{\mathbf{X}} = \mathbf{X}^T$

In terms of  $\hat{\mathbf{X}}$ , the covariance is given by,

$$\Sigma_{\hat{\mathbf{X}}} = \frac{1}{N} \hat{\mathbf{X}} \hat{\mathbf{X}}^T$$

# For reducing redundancy

- To reduce redundancy, we would like each variable (feature) to co-vary as little as possible with other variables.
- More precisely, we would like the covariance between features to be zero.
- That is, we look for a mapping,

$$\hat{\mathbf{Z}}_{(n \times N)} = \mathbf{W}_{(n \times n)} \hat{\mathbf{X}}_{(n \times N)}$$

such that,

$$\begin{aligned}\Sigma_{\hat{\mathbf{Z}}} &= \frac{1}{N} \hat{\mathbf{Z}} \hat{\mathbf{Z}}^T \\ &= \frac{1}{N} (\mathbf{W} \hat{\mathbf{X}}) (\mathbf{W} \hat{\mathbf{X}})^T \\ &= \frac{1}{N} \mathbf{W} \hat{\mathbf{X}} \hat{\mathbf{X}}^T \mathbf{W}^T \\ &= \mathbf{W} \Sigma_{\hat{\mathbf{X}}} \mathbf{W}^T \quad \text{is diagonal}\end{aligned}$$

# For reducing redundancy

- The symmetric covariance matrix  $\Sigma_{\hat{X}}$  can be diagonalized by selecting the transformation matrix  $W$  to be a matrix whose each row is an eigenvector of  $\Sigma_{\hat{X}}$ .
- By this selection  $W \equiv E^T$  (**This was the first goal for PCA**)
- where,  $E = [e_1 \ e_2 \ \dots \ e_n]$ ; The columns  $e_1, e_2, \dots, e_n$  of  $E$  are eigenvectors of  $\Sigma_{\hat{X}}$  associated with its eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_n$ , respectively.
- As  $\Sigma_{\hat{X}}$  is symmetric, its eigenvectors  $e_1, e_2, \dots, e_n$  are orthonormal  $\rightarrow E^T = E^{-1}$

$$E^{-1} \Sigma_{\hat{X}} E = \Lambda$$

$$= \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & \lambda_n \end{bmatrix}$$

A **diagonal** matrix

# For reducing noise

- We have  $\Sigma_{\hat{Z}} = \mathbf{E}^T \Sigma_{\hat{X}} \mathbf{E} = \Lambda$

The variances in the transformed domain are, therefore, given by,

$$\begin{bmatrix} \sigma_{1\hat{z}}^2 & 0 & \cdots & 0 \\ 0 & \sigma_{2\hat{z}}^2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \sigma_{n\hat{z}}^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

or

$$\sigma_{j\hat{z}}^2 = \lambda_j ; j = 1, \dots, n$$

- The largest eigenvalue, thus, corresponds to maximum variance.

# What are the **principal components**?

- The **first** principal component is the eigenvector of the covariance matrix  $\Sigma_{\widehat{X}}$  associated with the **largest** (represents significance) eigenvalue.
- The **second** principal component should also maximize the variance, which implies that the eigenvector associated with the **second largest** eigenvalue is the second principal component.
- It follows that other principal components are given by eigenvectors associated with eigenvalues with decreasing magnitude.
- This gives us eigenvalue-eigenvector pairs in order of significance.
- The  $k$  eigenvectors are called **principle components**, arranged **in order** of significance.

# Steps Involved in PCA

- First eigenvectors and eigenvalues are to be found from the covariance matrix.
- The next step is to order the eigenvalue-eigenvector pairs by the magnitude of the eigenvalues: from the largest to the smallest.
- As we have,

$$\sum_{j=1}^n \lambda_j = \sum_{j=1}^n \sigma_{jz}^2$$

We should understand that some eigenvalues have little contribution to the variance and they may be discarded.

- We take the leading  $k$  components that explain more than, for example, 90% of variance.

# Deriving the new dataset

- The **proportion of variance** explained by the  $k$  principal components is,

$$\frac{\lambda_1 + \lambda_2 + \dots + \lambda_k}{\lambda_1 + \lambda_2 + \dots + \lambda_n}$$

- If the features are **highly correlated**, there will be small number of eigenvectors with large eigenvalues and  $k$  will be much smaller than  $n$ , and a **large reduction of dimensionality** may be attained.
- If the features are not correlated,  $k$  will be as large as  $n$ , and there is no gain through PCA.
- The final step in PCA is to derive the new dataset—transformation of the original dataset

$$\hat{\mathbf{Z}}_{(k \times N)} = \mathbf{W}_{(k \times n)} \hat{\mathbf{X}}_{(n \times N)}$$

# Steps Involved in PCA (Summary)

- Standardize the data. (with mean =0 and variance = 1)
- Compute the Covariance matrix of dimensions.
- Obtain the Eigenvectors and Eigenvalues from the covariance matrix .
- Sort eigen values in descending order and choose the top k Eigenvectors that correspond to the k largest eigenvalues (k will become the number of dimensions of the new feature subspace  $k \leq n$ , n is the number of original dimensions).
- Construct the projection matrix W from the selected k Eigenvectors.
- Transform the original data set X via W to obtain the new k-dimensional feature subspace Z.

---

**Example 7.4**

---

Table 7.5 gives a set of normalized data (mean subtracted), and Fig. 7.7 shows a plot of this data for the toy example. The zero-mean dataset has 10 samples ( $N = 10$ ), and it has two attributes/features ( $n = 2$ ).

The dataset may be expressed as a  $2 \times 10$  data matrix

$$\hat{\mathbf{X}} = \begin{bmatrix} 0.69 & -1.31 & 0.39 & 0.09 & 1.29 & 0.49 & 0.19 & -0.81 & -0.31 & -0.71 \\ 0.49 & -1.21 & 0.99 & 0.29 & 1.09 & 0.79 & -0.31 & -0.81 & -0.31 & -1.01 \end{bmatrix}$$

**Table 7.5** A zero-mean dataset

$s^{(i)}$	$x_1$	$x_2$
$s^{(1)}$	0.69	0.49
$s^{(2)}$	-1.31	-1.21
$s^{(3)}$	0.39	0.99
$s^{(4)}$	0.09	0.29
$s^{(5)}$	1.29	1.09
$s^{(6)}$	0.49	0.79
$s^{(7)}$	0.19	-0.31
$s^{(8)}$	-0.81	-0.81
$s^{(9)}$	-0.31	-0.31
$s^{(10)}$	-0.71	-1.01

The covariance matrix

$$\Sigma_{\hat{\mathbf{X}}} = \frac{1}{N-1} \hat{\mathbf{X}} \hat{\mathbf{X}}^T$$

$$= \begin{bmatrix} 0.617 & 0.615 \\ 0.615 & 0.717 \end{bmatrix}$$

For the square symmetric covariance matrix  $\Sigma_{\hat{\mathbf{X}}}$ , the eigenvalues and eigenvectors are:

$$|\Sigma_{\hat{\mathbf{X}}} - \lambda I| = 0 \quad \lambda_1 = 0.049; \lambda_2 = 1.284$$

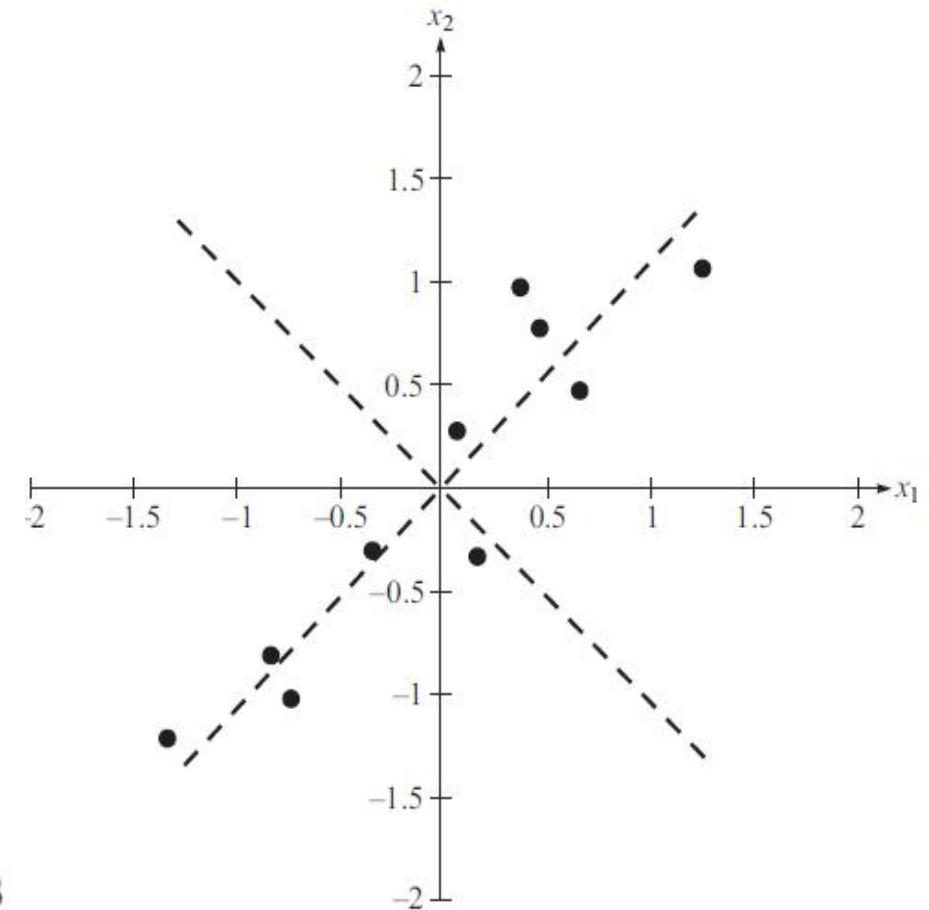
$$[\Sigma_{\hat{\mathbf{X}}} - \lambda I] e \quad \mathbf{e}_1 = \begin{bmatrix} -0.735 \\ 0.678 \end{bmatrix}; \mathbf{e}_2 = \begin{bmatrix} -0.678 \\ -0.735 \end{bmatrix}$$

Note that these are orthonormal eigenvectors; shown as dotted lines in Fig. 7.7. The eigenvectors provide us information about the patterns in the data.

Ordering eigenvalues from highest to lowest gives us the principle components in order of significance. In our example, principle components are:

$$\mathbf{e}_2^T = [-0.678 \quad -0.735]$$

$$\mathbf{e}_1^T = [-0.735 \quad -0.678]$$



A plot of the dataset of Table 7.5 with the eigenvectors of the covariance matrix

- Given our example set of data, and the fact that we have two components, we have two choices.
- We can either form a transformation matrix with both of the components:

$$\mathbf{W} = \begin{bmatrix} -0.678 & -0.735 \\ -0.735 & 0.678 \end{bmatrix}$$

or we can choose to leave out one corresponding to smaller eigenvalue—the less significant component:

$$\mathbf{W} = [-0.678 \quad -0.735]$$

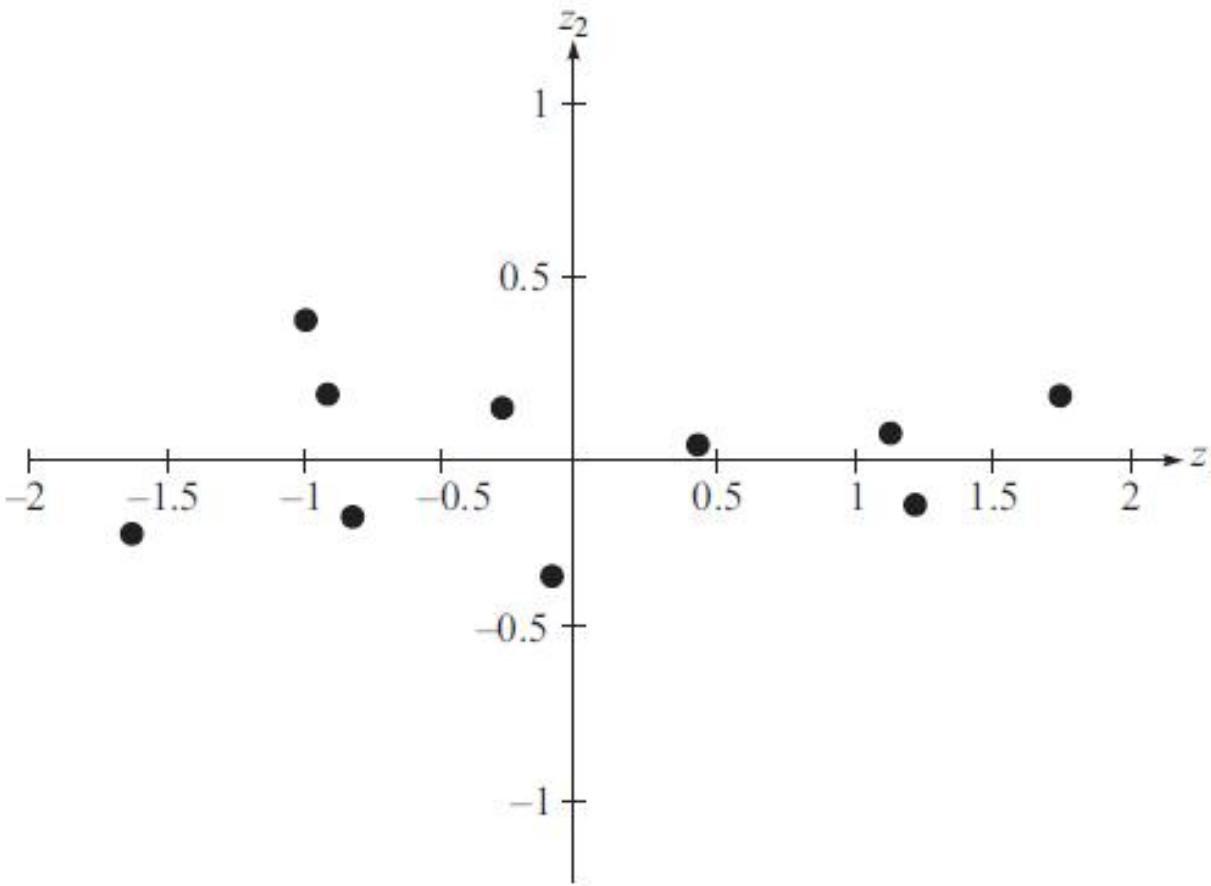
What needs to be done now is to generate the transformed data matrix:

$$\hat{\mathbf{Z}} = \mathbf{W} \hat{\mathbf{X}}$$

For the choice of  $k = n = 2$ , we use the transformation matrix given by Eqn (7.55a). The transformed dataset

$$\hat{\mathbf{Z}} = \begin{bmatrix} -0.828 & 1.778 & -0.992 & -0.274 & -1.676 & -0.913 & 0.099 & 1.145 & 0.438 & 1.224 \\ -0.175 & 0.143 & 0.384 & 0.130 & -0.209 & 0.175 & -0.350 & 0.046 & 0.018 & -0.163 \end{bmatrix}$$

The transformed data  $\hat{\mathbf{Z}}$  has been plotted in Fig. 7.8. The original data  $\hat{\mathbf{X}}$  was in terms of axes  $x_1$  and  $x_2$ . Through transformation matrix  $\mathbf{W}$ , we have changed the *basis vectors* of our data; from the axes  $x_1$  and  $x_2$  to the directions of principle components.



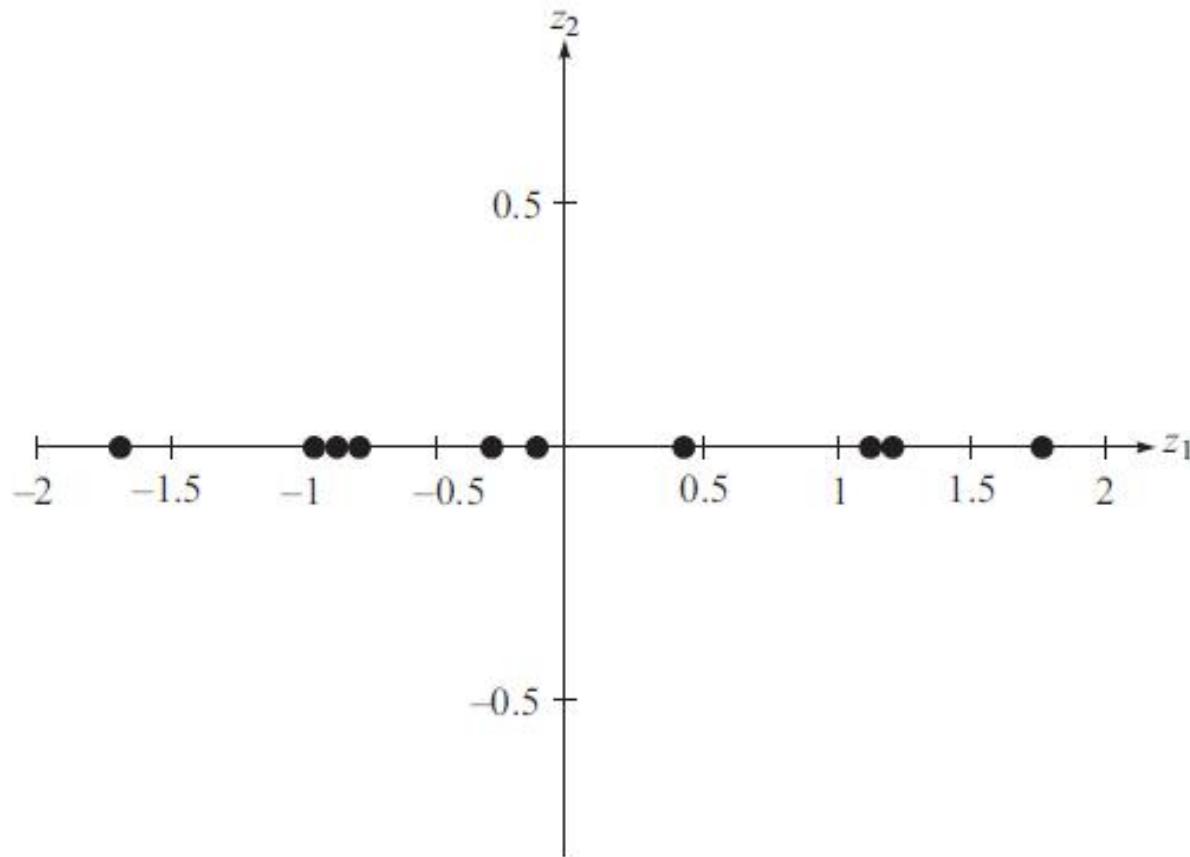
**Figure 7.8** A plot of data obtained by applying PCA using both eigenvectors.

In the case when the transformed data matrix has reduced dimensionality ( $k = 1 < n$ ), we use the transformation matrix given by Eqn (7.62).

$$\hat{\mathbf{Z}} = [-0.828 \quad 1.778 \quad -0.992 \quad -0.274 \quad -1.676 \quad -0.913 \quad 0.099 \quad 1.145 \quad 0.438 \quad 1.224]$$

# After applying PCA

As expected, it has only single dimension (Fig. 7.9).



**Figure 7.9** A plot of data obtained by applying PCA using one eigenvector.

# Advantages of PCA

- **Removes correlated features.** PCA will help you remove all the features that are correlated, a phenomenon known as *multicollinearity*. Finding features that are correlated is time consuming, especially if the number of features is large.
- **Improves machine learning algorithm performance.** With the number of features reduced with PCA, the time taken to train your model is now significantly reduced.
- **Reduce overfitting.** By removing the unnecessary features in your dataset, PCA helps to overcome overfitting.

# Disadvantages of PCA

- **Independent variables are now less interpretable.** PCA reduces your features into smaller number of components. Each component is now a linear combination of your original features, which makes it less readable and interpretable.
- **Information loss.** Data loss may occur if you do not exercise care in choosing the right number of components.
- **Feature scaling.** Because PCA is a *variance maximizing* exercise, PCA requires features to be scaled prior to processing.

# Naïve Bayes Classifier

# Introduction

- The naive Bayes classifier is probably among the most effective algorithms for learning tasks to classify text documents.
- The naive Bayes technique is extremely helpful in case of huge datasets.
- For example, Google employs naive Bayes classifier to correct the spelling mistakes in the text typed in by users.
- it gives a meaningful perspective to the comprehension of various learning algorithms that do not explicitly manipulate probabilities.
- Bayes theorem is the cornerstone of Bayesian learning methods.

# Bayes theorem

- Bayes theorem offers a method of calculating the probability of a hypothesis on the basis of its prior probability, the probabilities of observing different data given the hypothesis, and the observed data itself.
- The distribution of all possible values of discrete random variable  $y$  is expressed as probability distribution.

$$P(y) = \langle P(y_1), \dots, P(y_M) \rangle$$

$$P(y_1) + \dots + P(y_M) = 1$$

- We assume that there is some a priori probability (or simply prior)  $P(y_q)$  that the next feature vector belongs to the class q.

# Bayes theorem

- The continuous attributes are binned and converted to categorical variables.
- Therefore, each attribute  $x_j$  is assumed to have value set that are countable.
- Bayes theorem provides a way to calculate posterior  $P(y_k | x)$ ;  $k \in \{1, \dots, M\}$  from the known priors  $P(y_q)$ , together with known conditional probabilities  $P(x|yq)$ ;  $q = 1, \dots, M$ .

$$P(y_k | \mathbf{x}) = \frac{P(y_k) P(\mathbf{x}|y_k)}{P(\mathbf{x})}$$



$$P(\mathbf{x}) = \sum_{q=1}^M P(\mathbf{x}|y_q) P(y_q)$$

- $P(x)$  expresses variability of the observed data, independent of the class.

# Naive Rule

- As per this rule, the record gets classified as a member of the majority class.
- Assume that there are six attributes in the data table
  - $x_1$ : Day of Week,  $x_2$ : Departure Time,  $x_3$ : Origin,  $x_4$ : Destination,
  - $x_5$ : Carrier,  $x_6$ : Weather
  - and output  $y$  gives class labels (Delayed, On Time).
- Say 82% of the entries in  $y$  column record ‘On Time’.
- A naive rule for classifying a flight into two classes, ignoring information on  $x_1$ ,  $x_2$ , ...,  $x_6$  is to classify all flights as being ‘On Time’.
- The **naive rule** is used as a baseline for evaluating the performance of more complicated classifiers.
- Clearly, a classifier that uses attribute information should outperform the naive rule.

# Naive Bayes Classifier

- Takes into account the features as equally important and independent of each other, considering the class.
  - Not the scenario in real-life data.
- Each of the  $P(y_q)$  may be estimated simply by counting the frequency with which class  $y_q$  occurs in the training data:

$$P(y_q) = \frac{\text{Number of data with class } y_q}{\text{Total number } (N) \text{ of data}}$$

- If the decision must be made with **so little information**, it seems logical to use the following rule:  
**(Just like Naive rule)**

Decide  $y_k$  if  $P(y_k) > P(y_l); k \neq l$     **For balanced data, it will not work**

**Very much greater**  **decision will be right**

# Naive Bayes Classifier

- In most other circumstances, we need to estimate class-conditional probabilities  $P(x|y_q)$  as well

$$P(\mathbf{x}|y_q) = \frac{\text{Number of times pattern } \mathbf{x} \text{ appears with } y_q \text{ class}}{\text{Number of times } y_q \text{ appears in the data}}$$

- According to the assumption (attribute values are conditionally independent, given the class), given the class of the pattern, the probability of observing the conjunction  $x_1, x_2, \dots, x_n$  is just the product of the probabilities for the individual attributes:

$$P(x_1, x_2, \dots, x_n | y_q) = \prod_i P(x_j | y_q)$$

# Naive Bayes Classifier

$$y_{\text{NB}} = \arg \max_q P(y_q) \prod_j P(x_j | y_q)$$

- where  $y_{NB}$  denotes the class output by the naive Bayes classifier.
- The number of distinct  $P(x_j | y_q)$  terms that must be estimated from the training data is just the number of distinct attributes ( $n$ ) times the number of distinct classes ( $M$ ).

# Summary

The Bayes Naive classifier selects the most likely classification  $V_{nb}$  given the attribute values  $a_1, a_2, \dots, a_n$ . This results in:

$$V_{nb} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod P(a_i | v_j) \quad (1)$$

We generally estimate  $P(a_i | v_j)$  using m-estimates:

$$P(a_i | v_j) = \frac{n_c + mp}{n + m} \quad (2)$$

where:

$n$  = the number of training examples for which  $v = v_j$

$n_c$  = number of examples for which  $v = v_j$  and  $a = a_i$

$p$  = a priori estimate for  $P(a_i | v_j)$

$m$  = the equivalent sample size

**Table 3.1** Dataset for Example 3.1

# Example:

*y for  $x : \{M, 1.95\text{ m}\}$  ?*

- $y_1$  corresponds to the class ‘short’,
- $y_2$  corresponds to the class ‘medium’, and
- $y_3$  corresponds to the class ‘tall’.

	Gender $x_1$	Height $x_2$	Class	$y$
$s^{(1)}$	F	1.6 m	Short	$y_1$
$s^{(2)}$	M	2 m	Tall	$y_3$
$s^{(3)}$	F	1.9 m	Medium	$y_2$
$s^{(4)}$	F	1.88 m	Medium	$y_2$
$s^{(5)}$	F	1.7 m	Short	$y_1$
$s^{(6)}$	M	1.85 m	Medium	$y_2$
$s^{(7)}$	F	1.6 m	Short	$y_1$
$s^{(8)}$	M	1.7 m	Short	$y_1$
$s^{(9)}$	M	2.2 m	Tall	$y_3$
$s^{(10)}$	M	2.1 m	Tall	$y_3$
$s^{(11)}$	F	1.8 m	Medium	$y_2$
$s^{(12)}$	M	1.95 m	Medium	$y_2$
$s^{(13)}$	F	1.9 m	Medium	$y_2$
$s^{(14)}$	F	1.8 m	Medium	$y_2$
$s^{(15)}$	F	1.75 m	Medium	$y_2$

**Table 3.1** Dataset for Example 3.1

# Example:

*y for x : {M, 1.95 m} ?*

**N<sub>1</sub>= no. of y<sub>1</sub>=4; N<sub>2</sub>= no. of y<sub>2</sub>=8; N<sub>3</sub>= no. of y<sub>3</sub>=3;**  
 M=3, N=15.

$$P(y_1) = \frac{N_1}{N} = \frac{4}{15} = 0.267; \quad P(y_2) = \frac{N_2}{N} = \frac{8}{15} = 0.533$$

$$P(y_3) = \frac{N_3}{N} = \frac{3}{15} = 0.2$$

$$V_{x_1} : \{M, F\} = \{v_{1x_1}, v_{2x_1}\}; d_1 = 2$$

$$V_{x_2} = \{v_{1x_2}, v_{2x_2}, v_{3x_2}, v_{4x_2}, v_{5x_2}, v_{6x_2}\}; d_2 = 6$$

$$= \text{bins } \{(0, 1.6], (1.6, 1.7], (1.7, 1.8], (1.8, 1.9], (1.9, 2.0], (2.0, \infty)\}$$

	Gender <i>x<sub>1</sub></i>	Height <i>x<sub>2</sub></i>	Class	<i>y</i>
<i>s<sup>(1)</sup></i>	F	1.6 m	Short	<i>y<sub>1</sub></i>
<i>s<sup>(2)</sup></i>	M	2 m	Tall	<i>y<sub>3</sub></i>
<i>s<sup>(3)</sup></i>	F	1.9 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(4)</sup></i>	F	1.88 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(5)</sup></i>	F	1.7 m	Short	<i>y<sub>1</sub></i>
<i>s<sup>(6)</sup></i>	M	1.85 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(7)</sup></i>	F	1.6 m	Short	<i>y<sub>1</sub></i>
<i>s<sup>(8)</sup></i>	M	1.7 m	Short	<i>y<sub>1</sub></i>
<i>s<sup>(9)</sup></i>	M	2.2 m	Tall	<i>y<sub>3</sub></i>
<i>s<sup>(10)</sup></i>	M	2.1 m	Tall	<i>y<sub>3</sub></i>
<i>s<sup>(11)</sup></i>	F	1.8 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(12)</sup></i>	M	1.95 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(13)</sup></i>	F	1.9 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(14)</sup></i>	F	1.8 m	Medium	<i>y<sub>2</sub></i>
<i>s<sup>(15)</sup></i>	F	1.75 m	Medium	<i>y<sub>2</sub></i>

# Example:

*y for x : {M, 1.95 m} ?*

$N_1 = \text{no. of } y_1 = 4; N_2 = \text{no. of } y_2 = 8; N_3 = \text{no. of } y_3 = 3;$   
 $M = 3, N = 15.$

$$P(y_1) = \frac{N_1}{N} = \frac{4}{15} = 0.267; \quad P(y_2) = \frac{N_2}{N} = \frac{8}{15} = 0.533$$

$$P(y_3) = \frac{N_3}{N} = \frac{3}{15} = 0.2$$

$$V_{x_1} : \{M, F\} = \{v_{1x_1}, v_{2x_1}\}; d_1 = 2$$

$$V_{x_2} = \{v_{1x_2}, v_{2x_2}, v_{3x_2}, v_{4x_2}, v_{5x_2}, v_{6x_2}\}; d_2 = 6$$

$$= \text{bins } \{(0, 1.6], (1.6, 1.7], (1.7, 1.8], (1.8, 1.9], (1.9, 2.0], (2.0, \infty)\}$$

*Sorted w.r.t x2:*

Gender x1	Height x2 (m)	Class	y
F	1.6	Short	y1
F	1.6	Short	y1
F	1.7	Short	y1
M	1.7	Short	y1
F	1.75	Medium	y2
F	1.8	Medium	y2
F	1.8	Medium	y2
M	1.85	Medium	y2
F	1.88	Medium	y2
F	1.9	Medium	y2
F	1.9	Medium	y2
M	1.95	Medium	y2
M	2	Tall	y3
M	2.1	Tall	y3
M	2.2	Tall	y3

# Example:

*y for x : {M, 1.95 m} ?*

**N<sub>1</sub>**= no. of y<sub>1</sub>=4; **N<sub>2</sub>**= no. of y<sub>2</sub>=8; **N<sub>3</sub>**= no. of y<sub>3</sub>=3;  
M=3, N=15.

$$P(y_1) = \frac{N_1}{N} = \frac{4}{15} = 0.267; \quad P(y_2) = \frac{N_2}{N} = \frac{8}{15} = 0.533$$

$$P(y_3) = \frac{N_3}{N} = \frac{3}{15} = 0.2$$

$$V_{x_1}: \{M, F\} = \{v_{1x_1}, v_{2x_1}\}; d_1 = 2$$

$$V_{x_2} = \{v_{1x_2}, v_{2x_2}, v_{3x_2}, v_{4x_2}, v_{5x_2}, v_{6x_2}\}; d_2 = 6$$

$$= \text{bins } \{(0, 1.6], (1.6, 1.7], (1.7, 1.8], (1.8, 1.9], (1.9, 2.0], (2.0, \infty)\}$$

The count table generated from data is given in Table 3.2.

**Table 3.2** Number of training samples,  $N_{qv_{lx_j}}$ , of class  $q$  having value  $v_{lx_j}$

Value $v_{lx_j}$	Count $N_{qv_{lx_j}}$		
	Short $q = 1$	Medium $q = 2$	Tall $q = 3$
$v_{1x_1}: M$	1	2	3
$v_{2x_1}: F$	3	6	0
$v_{1x_2}: (0, 1.6] \text{ bin}$	2	0	0
$v_{2x_2}: (1.6, 1.7] \text{ bin}$	2	0	0
$v_{3x_2}: (1.7, 1.8] \text{ bin}$	0	3	0
$v_{4x_2}: (1.8, 1.9] \text{ bin}$	0	4	0
$v_{5x_2}: (1.9, 2.0] \text{ bin}$	0	1	1
$v_{6x_2}: (2.0, \infty] \text{ bin}$	0	0	2

# Example:

*y for x : {M, 1.95 m} ?*

$$N_1 = \text{no. of } y_1 = 4; N_2 = \text{no. of } y_2 = 8; N_3 = \text{no. of } y_3 = 3;$$

In the discretized domain, 'M' corresponds to  $v_{1x_1}$  and '1.95 m' corresponds to  $v_{5x_2}$ .

$$P(x_1|y_1) = N_{1v_{1x_1}}/N_1 = 1/4$$

$$P(x_1|y_2) = N_{2v_{1x_1}}/N_2 = 2/8$$

$$P(x_1|y_3) = N_{3v_{1x_1}}/N_3 = 3/3$$

$$P(x_2|y_1) = N_{1v_{5x_2}}/N_1 = 0/4$$

$$P(x_2|y_2) = N_{2v_{5x_2}}/N_2 = 1/8$$

$$P(x_2|y_3) = N_{3v_{5x_2}}/N_3 = 1/3$$

$$P(\mathbf{x}|y_1) = P(x_1|y_1) \times P(x_2|y_1) = \frac{1}{4} \times 0 = 0$$

The count table generated from data is given in Table 3.2.

**Table 3.2** Number of training samples,  $N_{qv_{lx_j}}$ , of class  $q$  having value  $v_{lx_j}$

Value $v_{lx_j}$	Count $N_{qv_{lx_j}}$		
	Short $q = 1$	Medium $q = 2$	Tall $q = 3$
$v_{1x_1}: M$	1	2	3
$v_{2x_1}: F$	3	6	0
$v_{1x_2}: (0, 1.6] \text{ bin}$	2	0	0
$v_{2x_2}: (1.6, 1.7] \text{ bin}$	2	0	0
$v_{3x_2}: (1.7, 1.8] \text{ bin}$	0	3	0
$v_{4x_2}: (1.8, 1.9] \text{ bin}$	0	4	0
$v_{5x_2}: (1.9, 2.0] \text{ bin}$	0	1	1
$v_{6x_2}: (2.0, \infty] \text{ bin}$	0	0	2

$$P(\mathbf{x}|y_2) = P(x_1|y_2) \times P(x_2|y_2) = \frac{2}{8} \times \frac{1}{8} = \frac{1}{32}$$

$$P(\mathbf{x}|y_3) = P(x_1|y_3) \times P(x_2|y_3) = \frac{3}{3} \times \frac{1}{3} = \frac{1}{3}$$

$$P(\mathbf{x}|y_1) P(y_1) = 0 \times 0.267 = 0$$

$$P(\mathbf{x}|y_2) P(y_2) = \frac{1}{32} \times 0.533 = 0.0166$$

$$P(\mathbf{x}|y_3) P(y_3) = \frac{1}{3} \times 0.2 = 0.066$$

$$y_{\text{NB}} = \arg \max_q P(\mathbf{x}|y_q) \times P(y_q)$$

- This gives  $q = 3$ .
- Therefore, for the pattern  $\mathbf{x} = \{\text{M } 1.95\text{m}\}$ , the predicted class is ‘tall’.

$$P(x_1|y_1) = N_{1v_{1x_1}}/N_1 = 1/4$$

$$P(x_1|y_2) = N_{2v_{1x_1}}/N_2 = 2/8$$

$$P(x_1|y_3) = N_{3v_{1x_1}}/N_3 = 3/3$$

$$P(x_2|y_1) = N_{1v_{5x_2}}/N_1 = 0/4$$

$$P(x_2|y_2) = N_{2v_{5x_2}}/N_2 = 1/8$$

$$P(x_2|y_3) = N_{3v_{5x_2}}/N_3 = 1/3$$

$$P(\mathbf{x}|y_1) = P(x_1|y_1) \times P(x_2|y_1) = \frac{1}{4} \times 0 = 0$$

- The true class in the data table is ‘medium’.
- Use of naive Bayes algorithm on real-life datasets will bring out the power of naive Bayes classifier when  $N$  is large.

## Example 2:

Let us say, we want to classify a Red Domestic SUV, as stolen or not

Attributes are Color , Type , Origin, and the subject, stolen can be either yes or no.

### data set

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

# Example 2:

Let us say, we want to classify a Red Domestic SUV, as stolen or not

Attributes are Color , Type , Origin, and the subject, stolen can be either yes or no.

## data set

Yes:

Red:

n = 5

n\_c = 3

p = .5

m = 3

SUV:

n = 5

n\_c = 1

p = .5

m = 3

Domestic:

n = 5

n\_c = 2

p = .5

m = 3

No:

Red:

n = 5

n\_c = 2

p = .5

m = 3

SUV:

n = 5

n\_c = 3

p = .5

m = 3

Domestic:

n = 5

n\_c = 3

p = .5

m = 3

Example No.	Color	Type	Origin	Stolen?
1	Red	Sports	Domestic	Yes
2	Red	Sports	Domestic	No
3	Red	Sports	Domestic	Yes
4	Yellow	Sports	Domestic	No
5	Yellow	Sports	Imported	Yes
6	Yellow	SUV	Imported	No
7	Yellow	SUV	Imported	Yes
8	Yellow	SUV	Domestic	No
9	Red	SUV	Imported	No
10	Red	Sports	Imported	Yes

## Example 2:

- We need to calculate the probabilities  $P(\text{Red}|\text{Yes})$ ,  $P(\text{SUV}|\text{Yes})$ ,  $P(\text{Domestic}|\text{Yes})$ ,  $P(\text{Red}|\text{No})$ ,  $P(\text{SUV}|\text{No})$ , and  $P(\text{Domestic}|\text{No})$
- and multiply them by  $P(\text{Yes})$  and  $P(\text{No})$  respectively.
- Then we can estimate these values using equation for  $Y_{NB}$
- Looking at  $P(\text{Red}|Y \text{ es})$ , we have 5 cases where  $v_j = \text{Yes}$ , and in 3 of those cases  $a_i = \text{Red}$ .
- So for  $P(\text{RedjY es})$ ,  $n = 5$  and  $n_c = 3$ .
- Note that all attribute are binary (two possible values).
- We are assuming no other information so,  $p = 1 / (\text{number-of-attribute-values}) = 0.5$  for all of our attributes.
- Our  $m$  value is arbitrary, (We will use  $m = 3$ ) but consistent for all attributes.
- Now we simply apply equation (3) using the precomputed values of  $n$ ,  $n_c$ ,  $p$ , and  $m$ .

## Example 2:

$$P(\text{Red}|\text{Yes}) = \frac{3 + 3 * .5}{5 + 3} = .56$$

$$P(\text{SUV}|\text{Yes}) = \frac{1 + 3 * .5}{5 + 3} = .31$$

$$P(\text{Domestic}|\text{Yes}) = \frac{2 + 3 * .5}{5 + 3} = .43$$

$$P(\text{Red}|\text{No}) = \frac{2 + 3 * .5}{5 + 3} = .43$$

$$P(\text{SUV}|\text{No}) = \frac{3 + 3 * .5}{5 + 3} = .56$$

$$P(\text{Domestic}|\text{No}) = \frac{3 + 3 * .5}{5 + 3} = .56$$

We have  $P(\text{Yes}) = .5$  and  $P(\text{No}) = .5$ , so we can apply equation (2). For  $v = \text{Yes}$ , we have

$$\begin{aligned} & P(\text{Yes}) * P(\text{Red} \mid \text{Yes}) * P(\text{SUV} \mid \text{Yes}) * P(\text{Domestic} \mid \text{Yes}) \\ &= .5 * .56 * .31 * .43 = .037 \end{aligned}$$

and for  $v = \text{No}$ , we have

$$\begin{aligned} & P(\text{No}) * P(\text{Red} \mid \text{No}) * P(\text{SUV} \mid \text{No}) * P(\text{Domestic} \mid \text{No}) \\ &= .5 * .43 * .56 * .56 = .069 \end{aligned}$$

Since  $0.069 > 0.037$ , our example gets classified as 'NO'

# SUPPORT VECTOR MACHINE

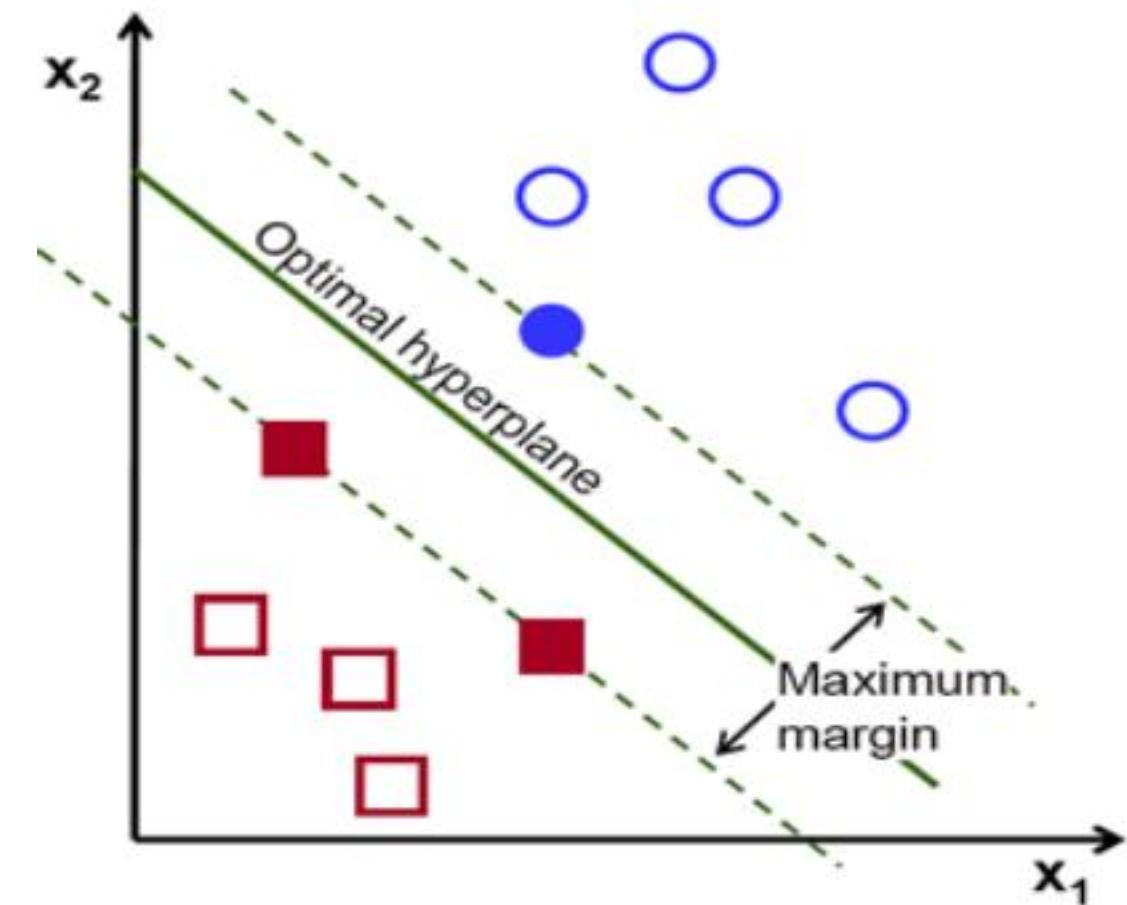
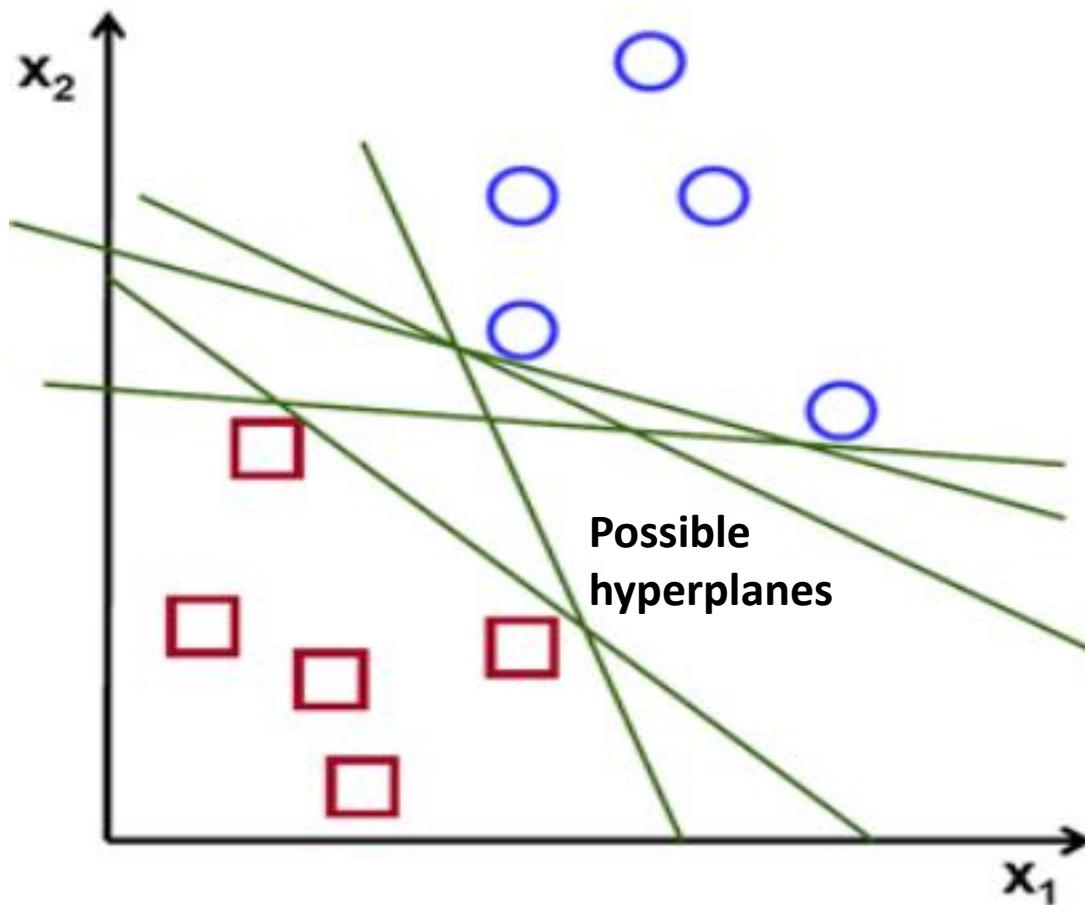
Dr. Srikanth Allamsetty

The idea of support vectors and its importance

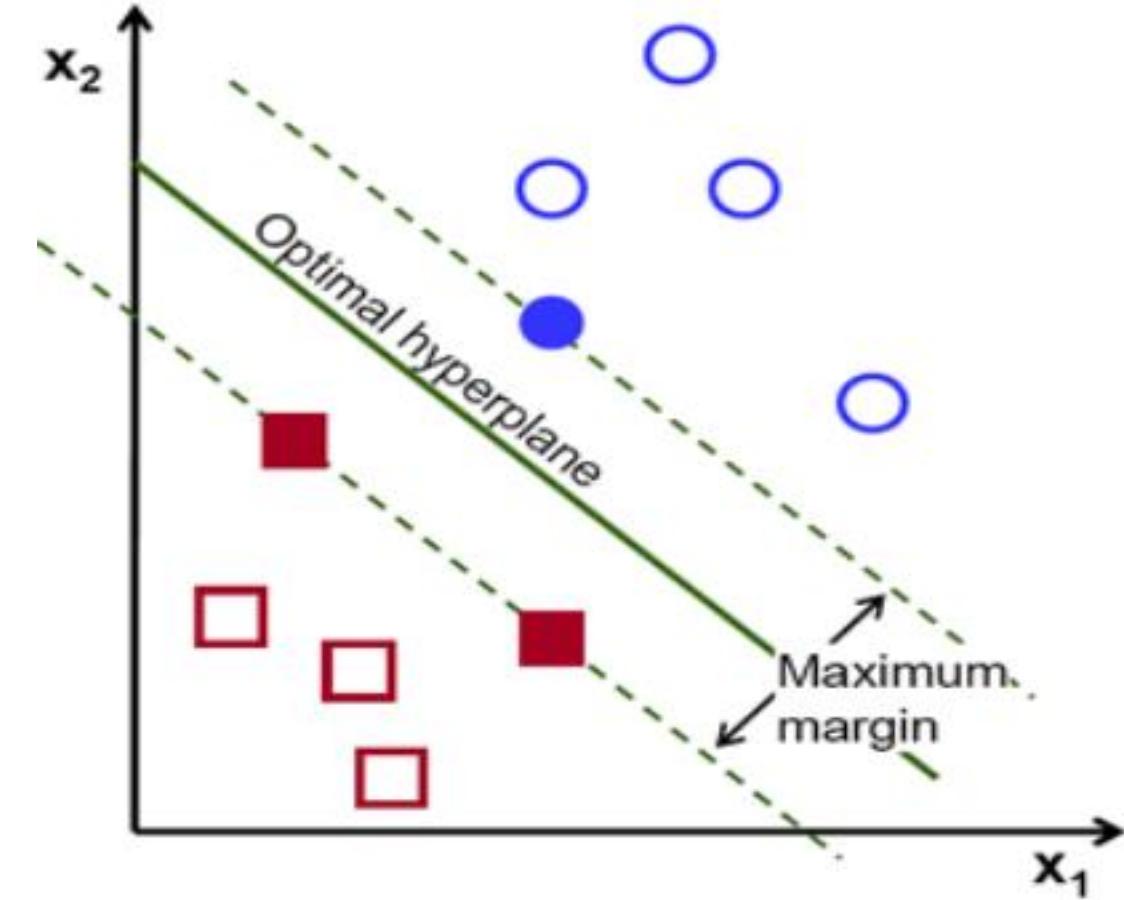
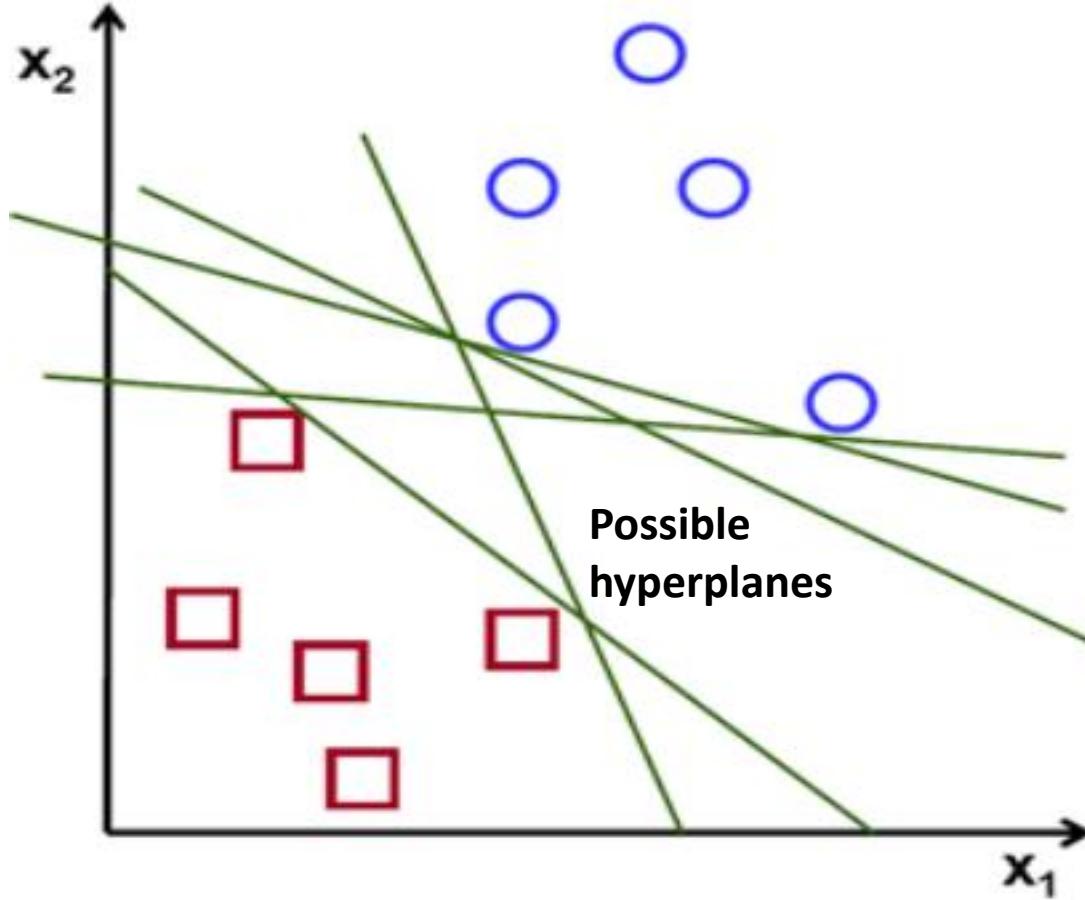
# Introduction

- The support vector machine is currently considered to be the best off-the-shelf learning algorithm and has been applied successfully in various domains.
- Support vector machines were originally designed for binary classification.
- Then, it is extended to solve multi-class and regression problems.
- But, it is widely used in classification objectives.
- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space (N – the number of features) that distinctly classifies the data points.

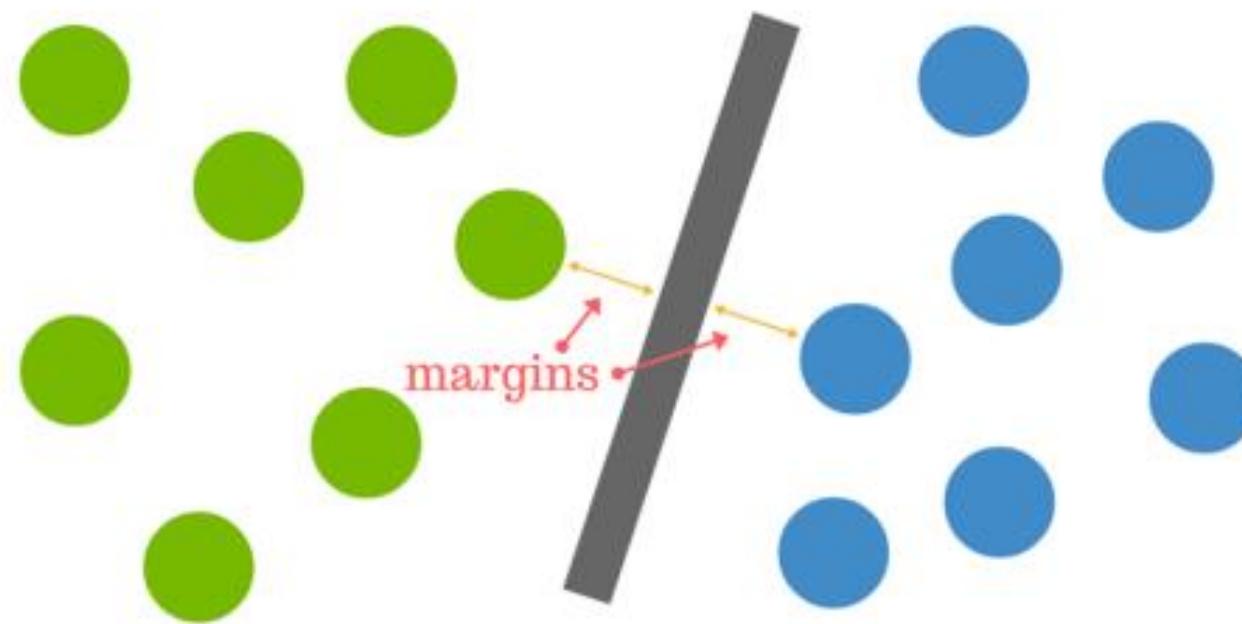
- To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes.



- Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

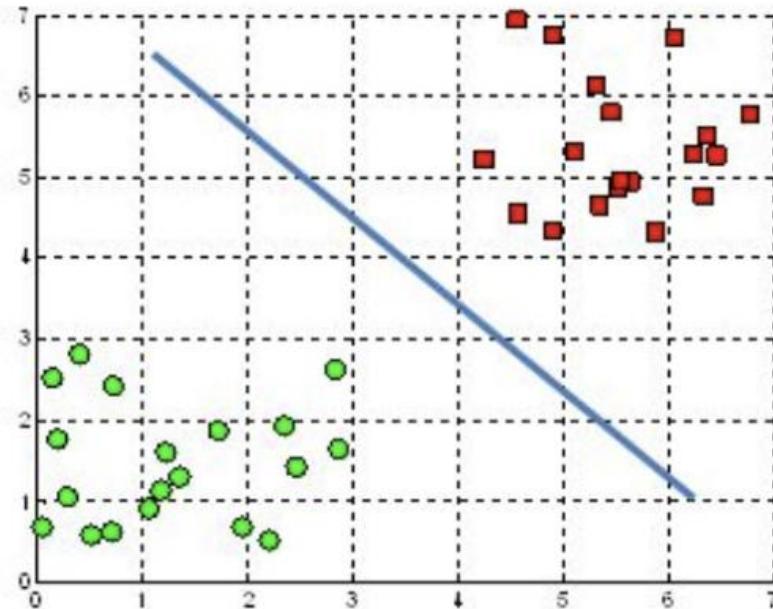


- Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.
- The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly.

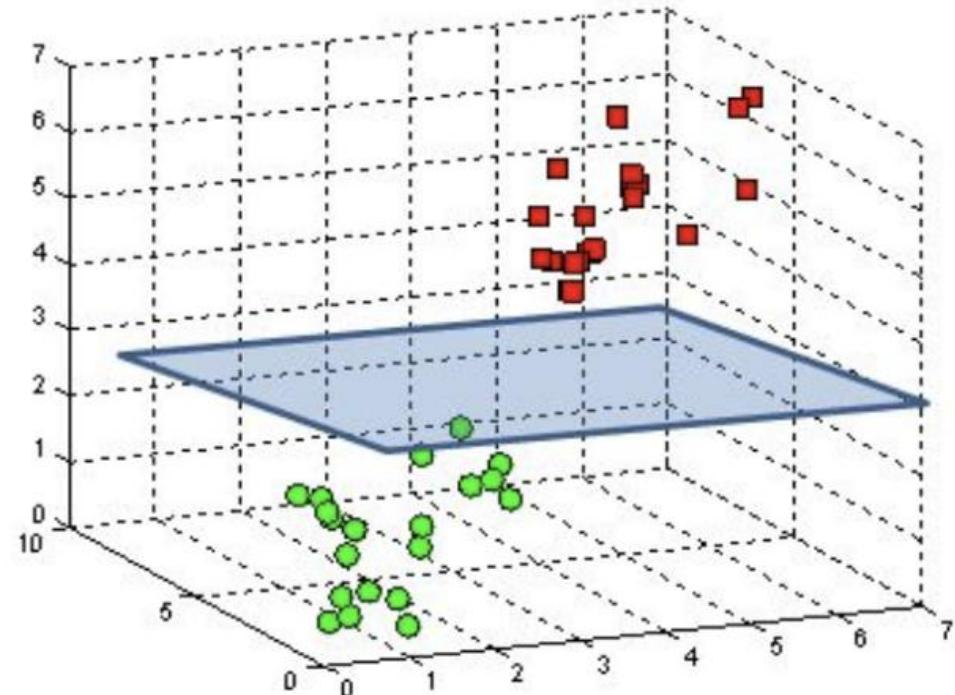


- Hyperplanes are decision boundaries that help classify the data points.
- Data points falling on either side of the hyperplane can be attributed to different classes.
- Also, the dimension of the hyperplane depends upon the number of features.
- It becomes difficult to imagine when the number of features exceeds 3.

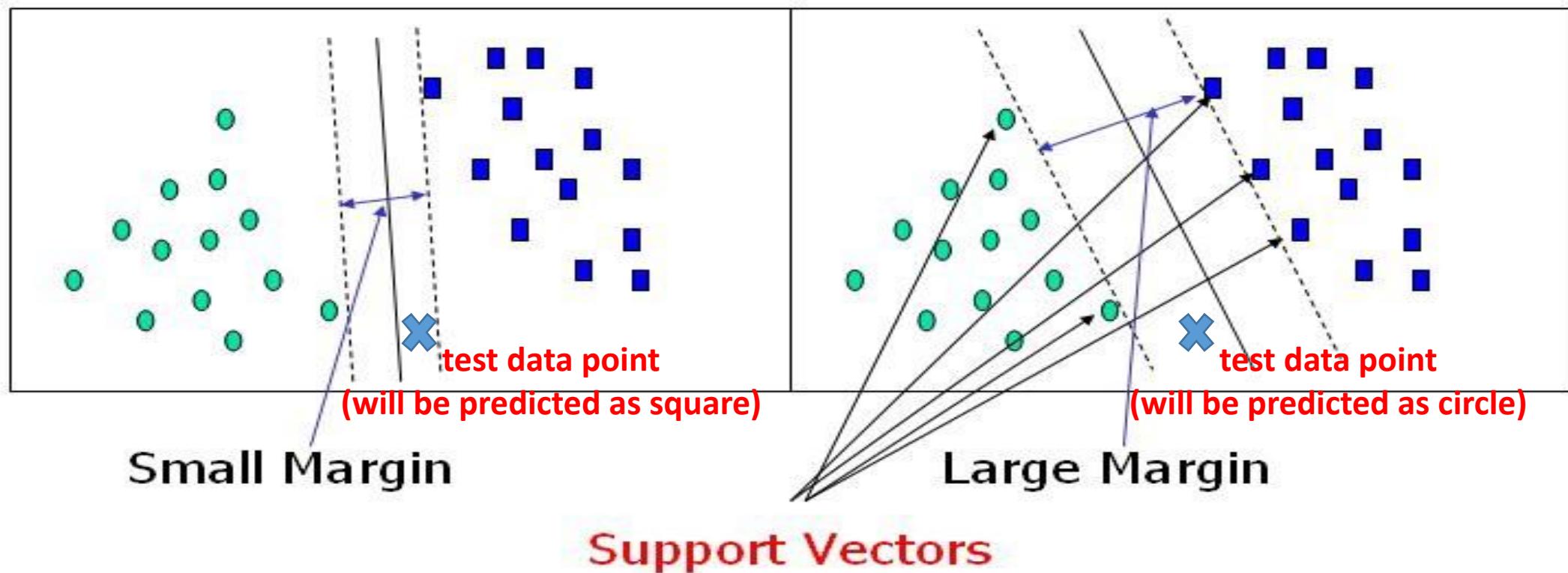
A hyperplane in  $\mathbb{R}^2$  is a line



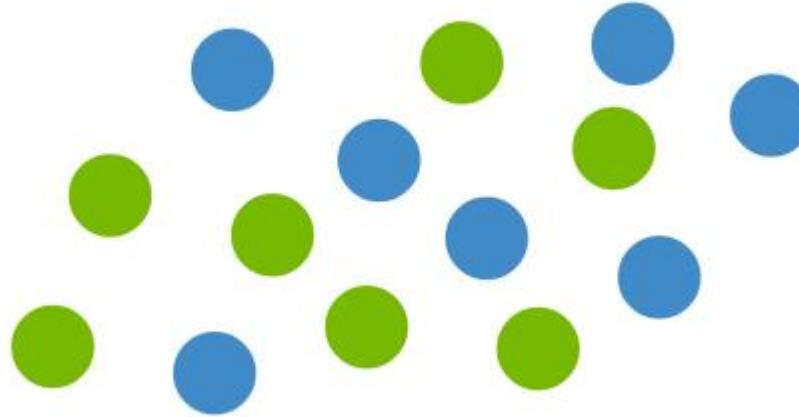
A hyperplane in  $\mathbb{R}^3$  is a plane



- Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane.
- Using these support vectors, we maximize the margin of the classifier.
- Deleting the support vectors will change the position of the hyperplane.
- These are the points that help us build our SVM (that works for a new/test data).

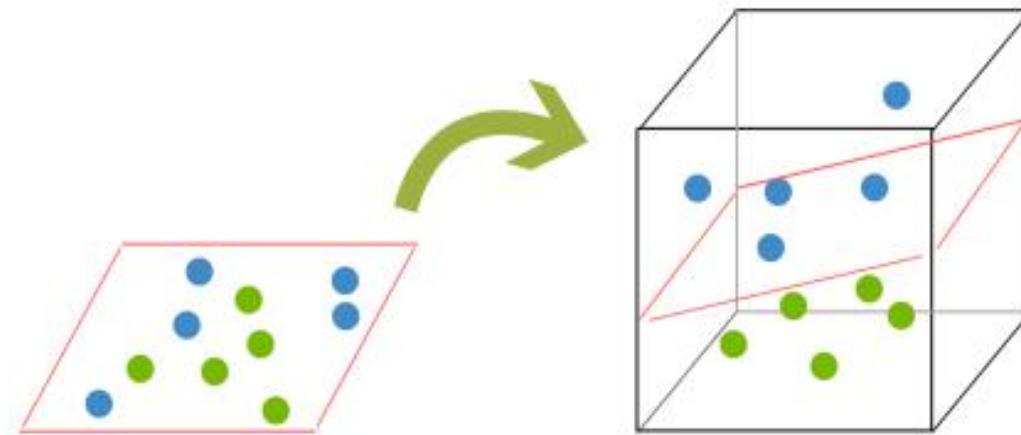


- But what happens when there is no clear hyperplane?
- A dataset will often look more like the jumbled balls below which represent a linearly non separable dataset.



- In order to classify a dataset like the one above it's necessary to move away from a 2d view of the data to a 3d view.

- Explaining this is easiest with another simplified example.
- Imagine that our two sets of colored balls above are sitting on a sheet and this sheet is lifted suddenly, launching the balls into the air.
- While the balls are up in the air, you use the sheet to separate them.
- This ‘lifting’ of the balls represents the mapping of data into a higher dimension.



- This is known as kernelling.

# Pros & Cons of Support Vector Machines

## Pros

- Accuracy
- Works well on smaller cleaner datasets
- It can be more efficient because it uses a subset of training points

## Cons

- Isn't suited to larger datasets as the training time with SVMs can be high
- Less effective on noisier datasets with overlapping classes

# Applications

- SVM is used for text classification tasks such as category assignment, detecting spam and sentiment analysis.
- It is also commonly used for image recognition challenges, performing particularly well in aspect-based recognition and color-based classification.
- SVM also plays a vital role in many areas of handwritten digit recognition, such as postal automation services.

# Derivation of Support Vector Equation

# Comparison with logistic regression

Training set:  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

m examples  
 $x \in \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$   
 $x_0 = 1, y \in \{0, 1\}$

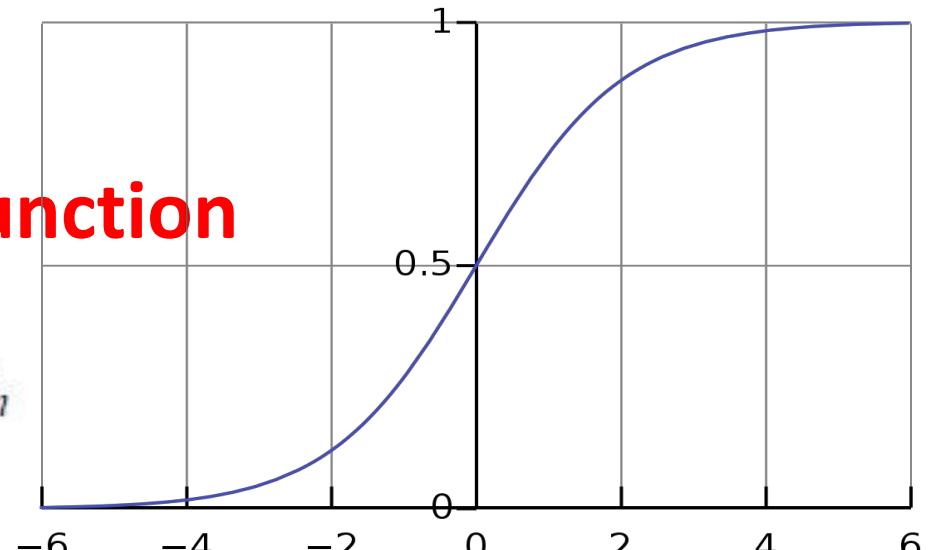
$$\log(\text{odds}) = \log \frac{P(\text{Class 1}|\mathbf{x})}{1 - P(\text{Class 1}|\mathbf{x})} = w_0 + w_1 x_1 + \dots + w_n x_n$$

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters  $\theta$ ?

Max Likelihood Estimation (**already discussed**)

Sigmoid function



Threshold classifier output  $h_{\theta}(x)$  at 0.5:

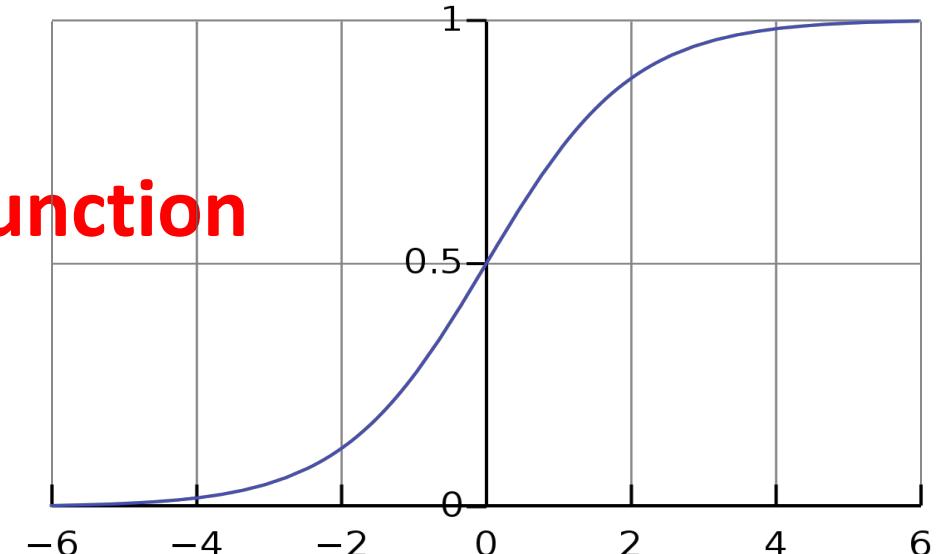
If  $h_{\theta}(x) \geq 0.5$  , predict "y = 1"

If  $h_{\theta}(x) < 0.5$  , predict "y = 0"

# Comparison with logistic regression

- In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is less than -1, we identify it with another class.
- Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values ( $[-1,1]$ ) which acts as margin.

Sigmoid function



Threshold classifier output  $h_\theta(x)$  at 0.5:

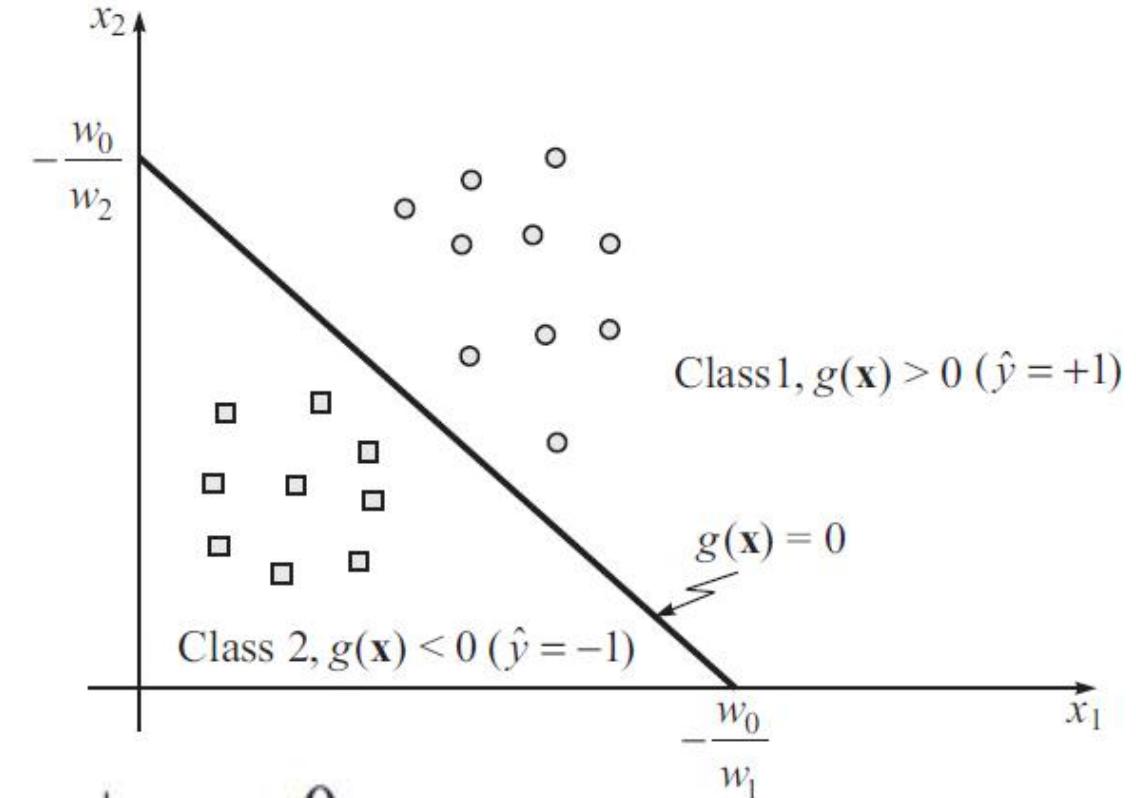
If  $h_\theta(x) \geq 0.5$  , predict “y = 1”

If  $h_\theta(x) < 0.5$  , predict “y = 0”

# Comparison with logistic regression

- In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is less than -1, we identify it with another class.
- Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values ( $[-1,1]$ ) which acts as margin.

$$g(\mathbf{x}) = w_1 x_1 + w_2 x_2 + w_0 = 0$$



- $g(x)$  is a linear discriminant function that divides (categorizes)  $\Re^2$  into two decision regions.

- The generalization of the linear discriminant function for an n-dimensional feature space in  $\mathbb{R}^n$  is straight forward:

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$$

$\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$  is the feature vector

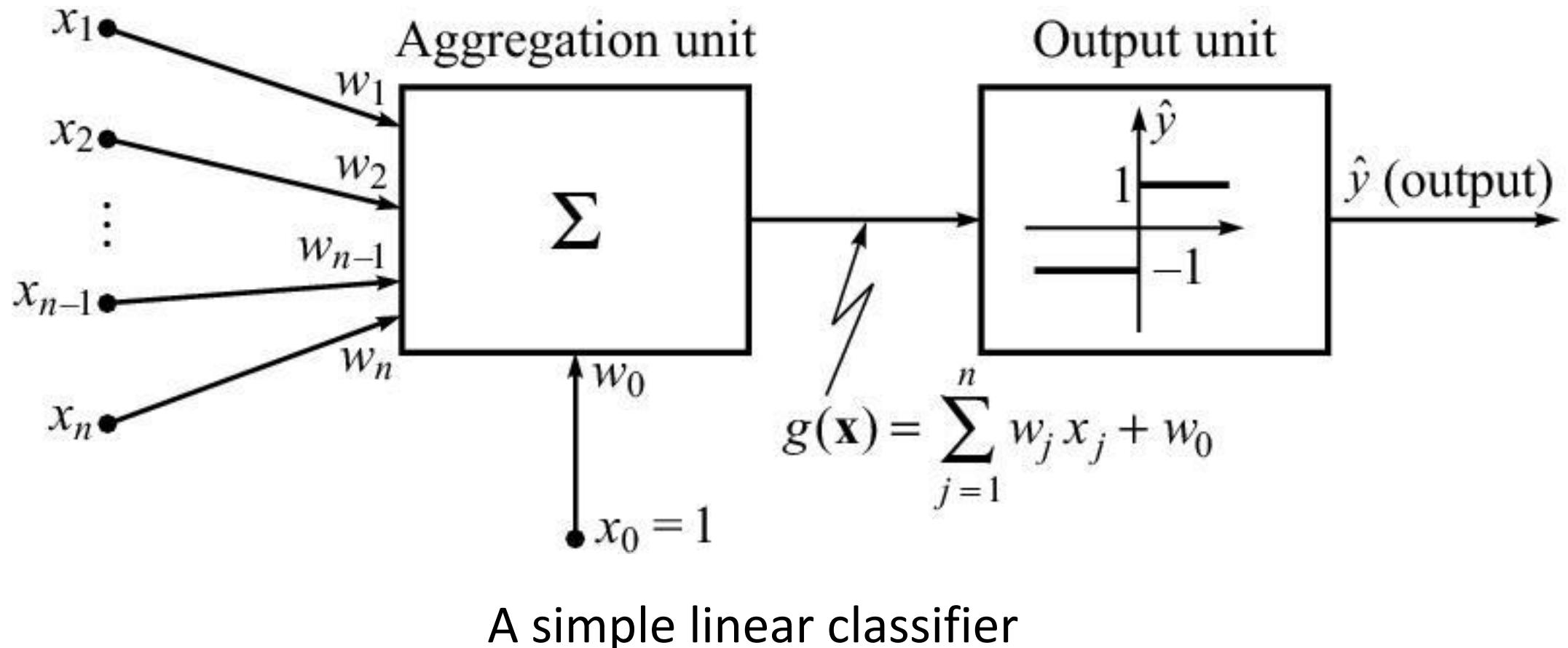
$\mathbf{w} = [w_1 \ w_2 \ \dots \ w_n]^T$  is a *weight vector*

$w_0$  = *bias* parameter

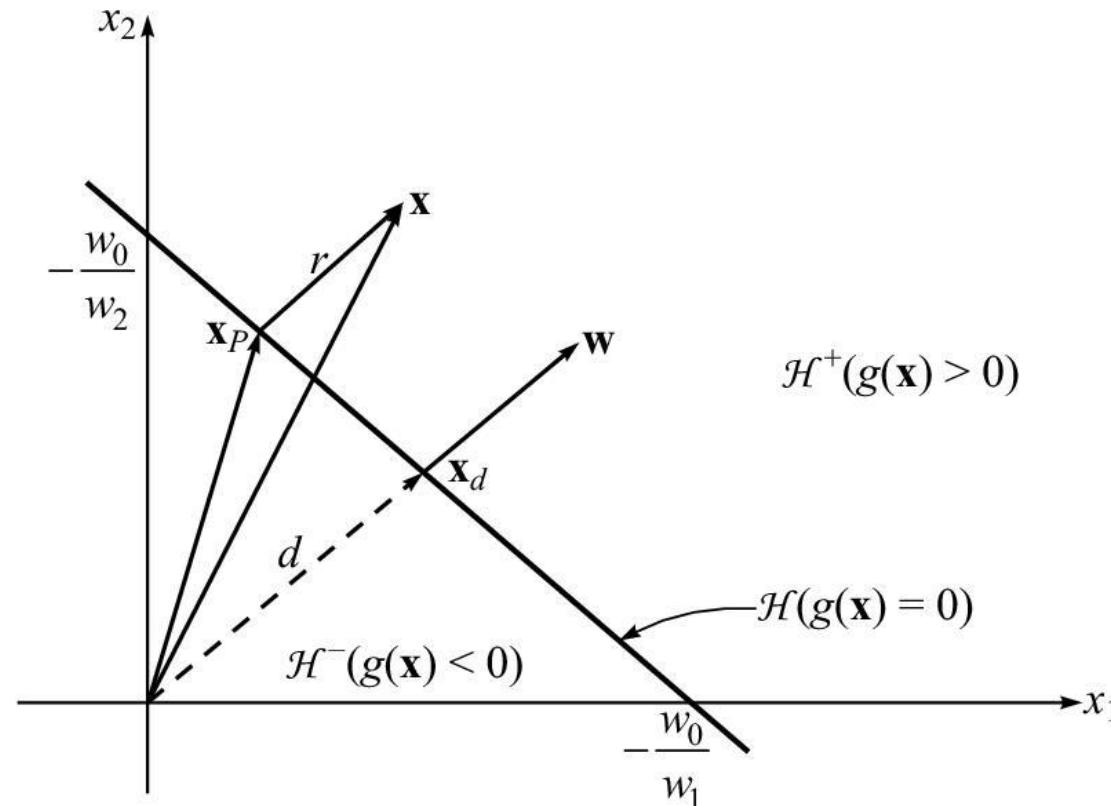
- The discriminant function is now a linear n-dimensional surface, called a hyperplane; symbolized as  $\mathcal{H}$
- A two-category classifier implements the following decision rule:

Decide Class 1 if  $g(x) > 0$  and Class 2 if  $g(x) < 0$
- Thus,  $x$  is assigned to Class 1 if the inner product  $\mathbf{w}^T \mathbf{x}$  exceeds the threshold (bias)  $-w_0$ , and to Class 2 otherwise.

- Figure shows the architecture of a typical implementation of the linear classifier.
- It consists of two computational units: an aggregation unit and an output unit.



- Geometry for  $n = 2$  with  $w_1 > 0$ ,  $w_2 > 0$  and  $w_0 < 0$  is shown in Figure below.
- The origin is on the negative side of  $\mathcal{H}$  if  $w_0 < 0$ , and if  $w_0 > 0$ , the origin is on the positive side of  $\mathcal{H}$ .
- If  $w_0 = 0$ , the hyperplane passes through the origin.



Linear decision boundary between two classes

Location of any point  $\mathbf{x}$  may be considered relative to  $\mathcal{H}$ .

Defining  $\mathbf{x}_p$  as the normal projection of  $\mathbf{x}$  onto  $\mathcal{H}$ ,

$$\mathbf{x} = \mathbf{x}_P + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

where  $\|\mathbf{w}\|$  is the Euclidean norm of  $\mathbf{w}$  and  $\frac{\mathbf{w}}{\|\mathbf{w}\|}$  is a unit vector.

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + w_0 = \mathbf{w}^T \left( \mathbf{x}_P + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) + w_0 \\ &= \mathbf{w}^T \mathbf{x}_P + w_0 + \frac{\mathbf{w}^T r \mathbf{w}}{\|\mathbf{w}\|} \\ &= r \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} = r \frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} = r \|\mathbf{w}\| \end{aligned}$$

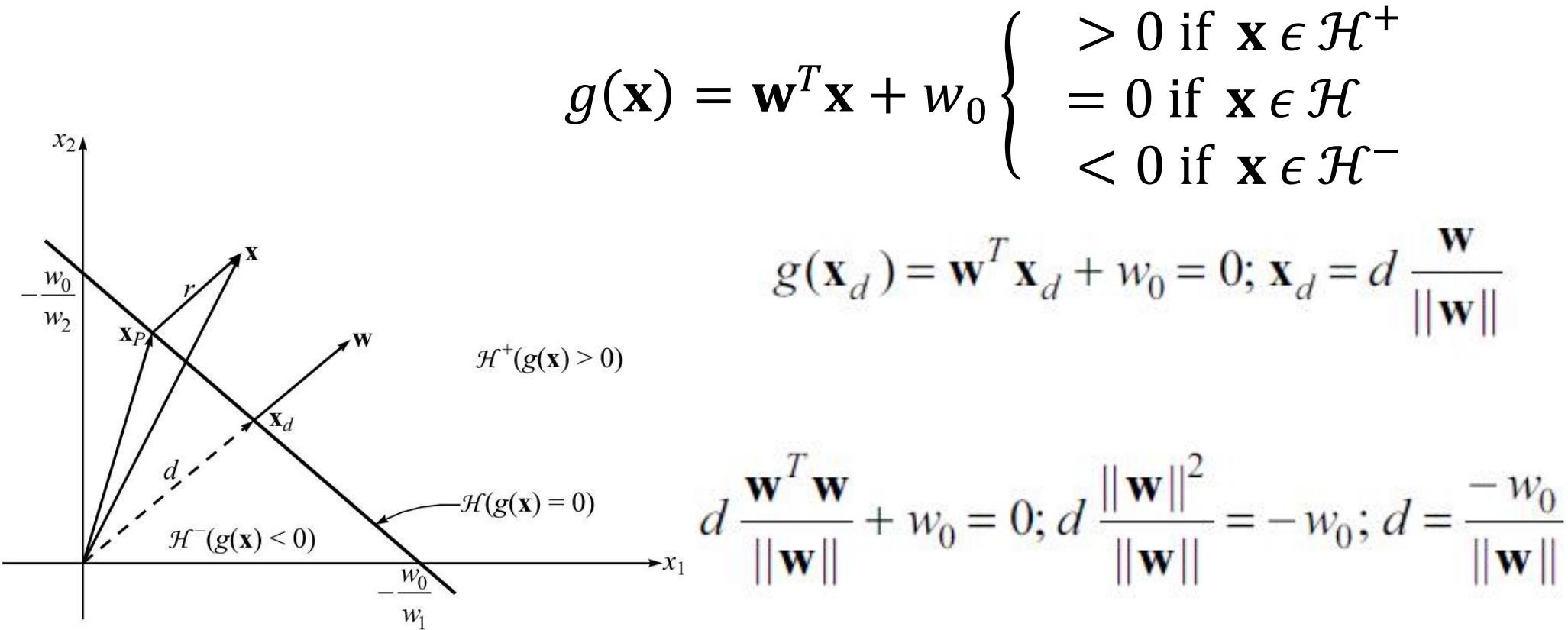
Algebraic measure of the distance  
from  $\mathbf{x}$  to the hyperplane

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$

As

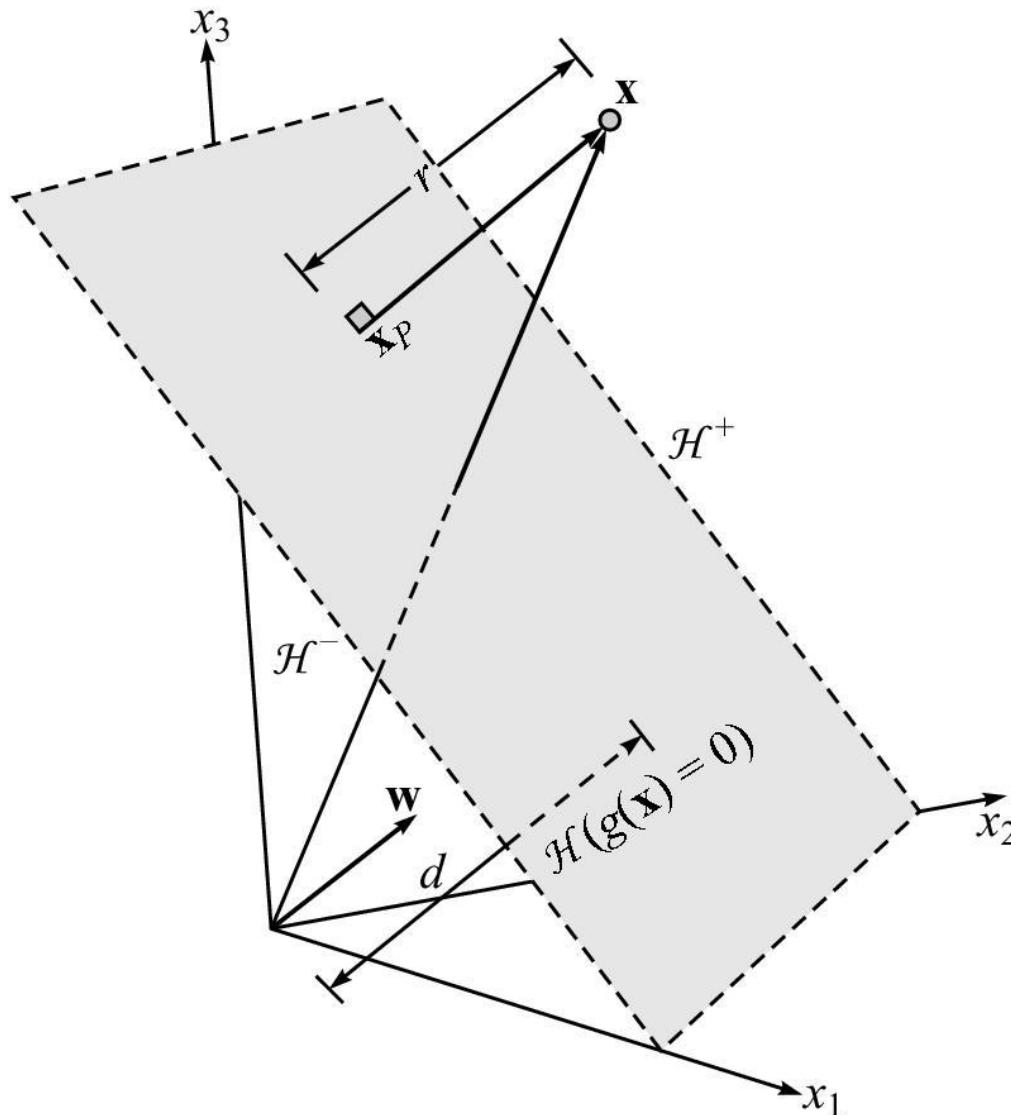
$$g(\mathbf{x}) = \mathbf{r} \cdot \|\mathbf{w}\|$$

$|g(\mathbf{x})|$  is a measure of the Euclidean distance of the point  $\mathbf{x}$  from the decision hyperplane  $\mathcal{H}$ .



Perpendicular distance  $d$  from coordinate origin to  $\mathcal{H} = w_0/\|\mathbf{w}\|$

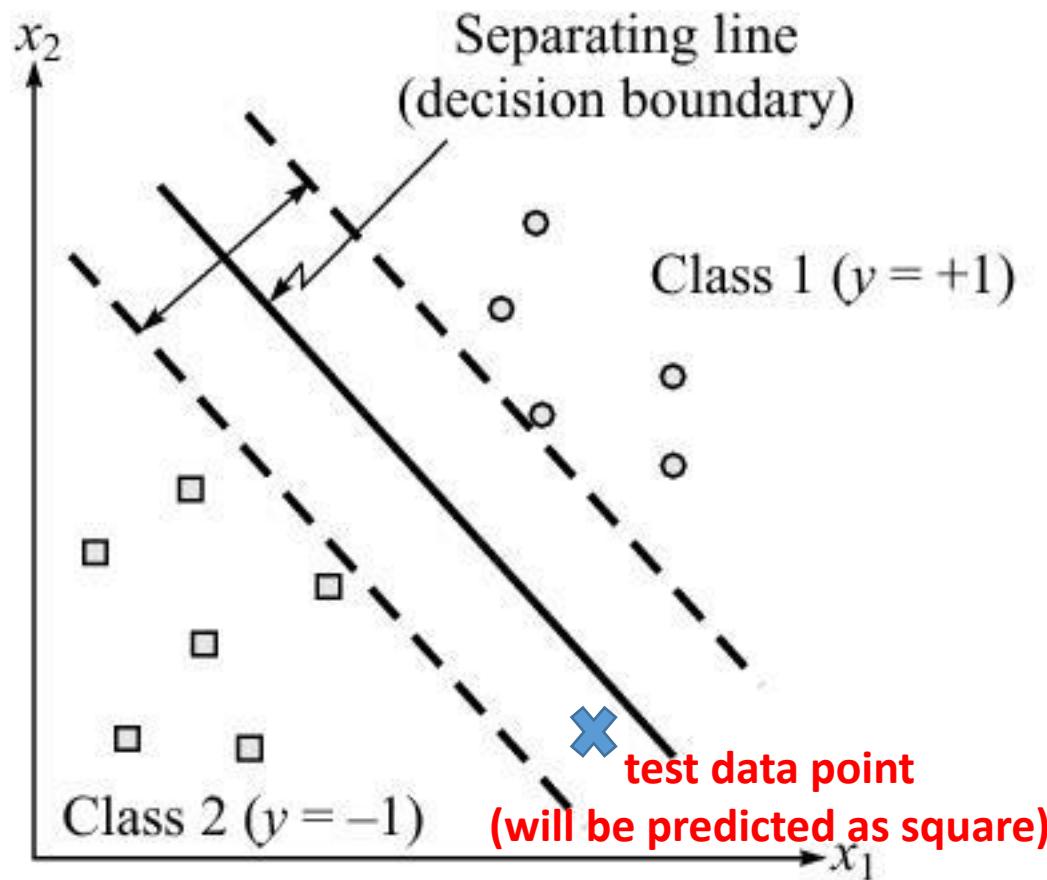
## Geometry for 3-dimensions ( $n=3$ )



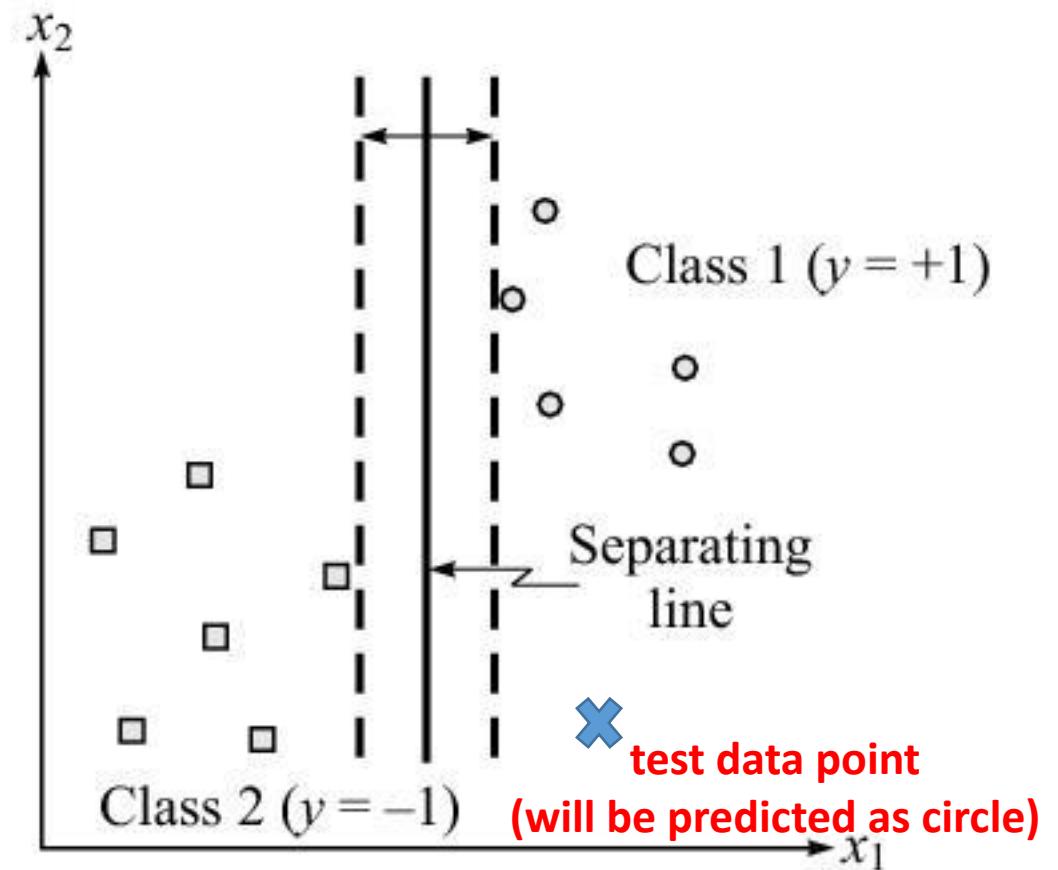
Hyperplane  $\mathcal{H}$  separates the feature space into two half space  $\mathcal{H}^+$  and  $\mathcal{H}^-$

# Linear Maximal Margin Classifier for Linearly Separable Data

- For linearly separable, many hyperplanes exist to perform separation.
- SVM framework tells which hyperplane is best.
- Hyperplane with the largest *margin* which minimizes training error.
- Select the decision boundary that is far away from both the classes.
- Large margin separation is expected to yield good generalization.
- in  $w^T x + w_0 = 0$ ,  $w$  defines a direction perpendicular to the hyperplane.
- $w$  is called the normal vector (or simply normal) of the hyperplane.
- Without changing the normal vector  $w$ , varying  $w_0$  moves the hyperplane parallel to itself.



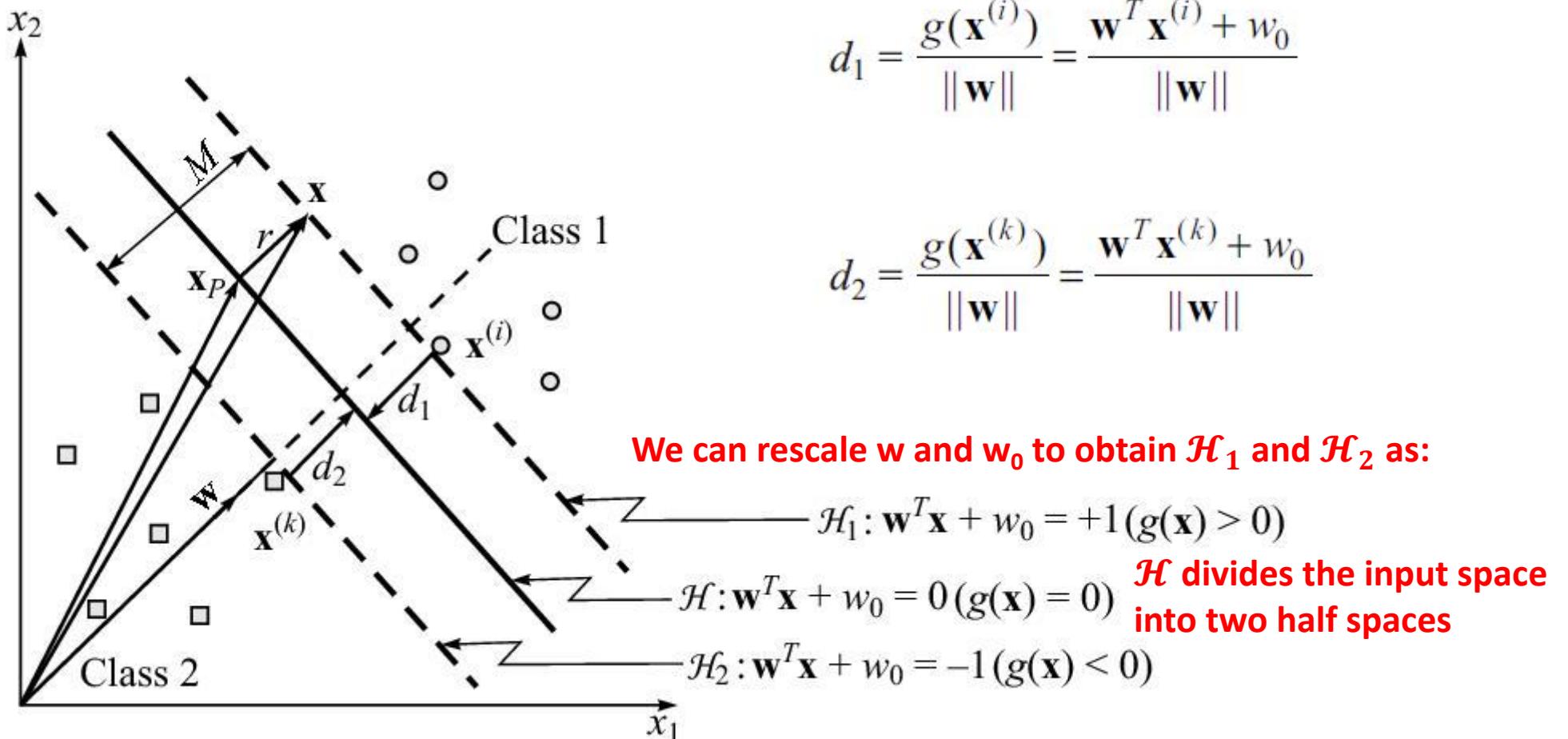
(a) Large margin separation



(b) Small margin separation

Large margin and small margin separation

Two parallel hyperplanes  $\mathcal{H}_1$  and  $\mathcal{H}_2$  that pass through  $\mathbf{x}^{(i)}$  and  $\mathbf{x}^{(k)}$  respectively.



Geometric interpretation of algebraic distances of points to a hyperplane for two-dimensional case

$\mathcal{H}_1$  and  $\mathcal{H}_2$  are parallel to the hyperplane  $\mathbf{w}^T \mathbf{x} + w_0 = 0$ .

$$\mathcal{H}_1: \mathbf{w}^T \mathbf{x} + w_0 = +1$$

$$\mathcal{H}_2: \mathbf{w}^T \mathbf{x} + w_0 = -1$$

such that

$$\mathbf{w}^T \mathbf{x}^{(i)} + w_0 \geq 1 \text{ if } y^{(i)} = +1$$

$$\mathbf{w}^T \mathbf{x}^{(i)} + w_0 \leq -1 \text{ if } y^{(i)} = -1$$

or equivalently,

$$y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1$$

distance between the two hyperplanes = margin  $M$

$$d_1 = \frac{1}{\|\mathbf{w}\|}; d_2 = \frac{-1}{\|\mathbf{w}\|} \rightarrow M = \frac{2}{\|\mathbf{w}\|}$$

This equation states that maximizing the margin of separation between classes is equivalent to minimizing the Euclidean norm of the weight vector  $w$ .

# KKT Condition

# Learning problem in SVM

- Linearly separable training examples,

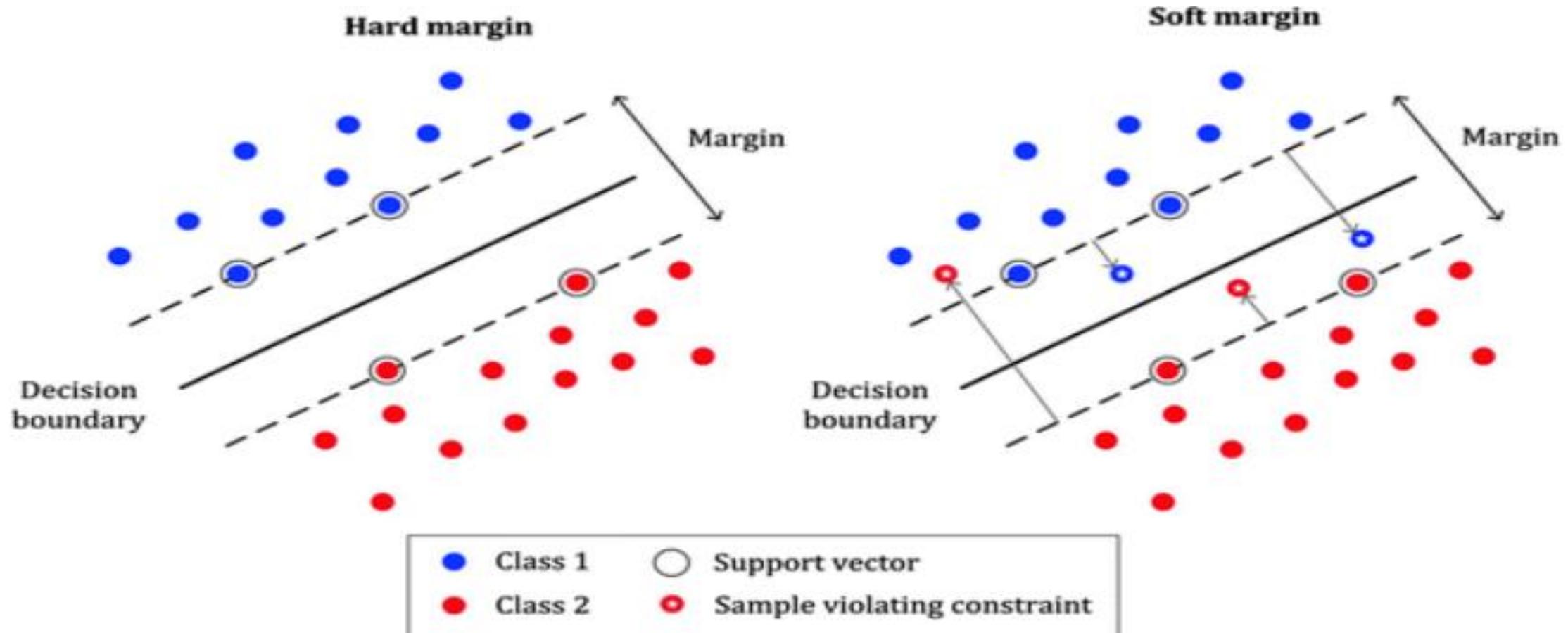
$$\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$$

**Problem:** Solve the following constrained minimization problem:

$$\begin{aligned} & \text{minimize } f(\mathbf{w}) = \frac{1}{2} \mathbf{w}^T \mathbf{w} \\ & \text{subject to } y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1; i = 1, \dots, N \end{aligned}$$

This is the formulation of ***hard-margin*** SVM.

# Hard margin svm Vs Soft margin svm



## Dual formulation of constrained optimization problem:

Lagrangian is constructed:

$$L(\mathbf{w}, w_0, \lambda) = \frac{1}{2} \mathbf{w}^T \mathbf{w} - \sum_{i=1}^N \lambda_i [y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1]$$

The Karush-Kuhn-Tucker (**KKT**) conditions are as follows:

- i.  $\frac{\partial L}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^N \lambda_i y^{(i)} \mathbf{x}^{(i)}$   
 $\frac{\partial L}{\partial w_0} = 0 \Rightarrow \sum_{i=1}^N \lambda_i y^{(i)} = 0$
- ii.  $y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 \geq 0; i = 1, \dots, N$
- iii.  $\lambda_i \geq 0; i = 1, \dots, N$
- iv.  $\lambda_i (y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1) = 0; i = 1, \dots, N$

After solving the dual problem numerically, the resulting optimum  $\lambda_i$  values are used to compute  $\mathbf{w}$  and  $w_0$  using the KKT conditions.

- $\mathbf{w}$  is computed using condition (i) of KKT conditions

$$\mathbf{w} = \sum_{i=1}^N \lambda_i y^{(i)} \mathbf{x}^{(i)}$$

- and  $w_0$  is computed using condition (iv) of KKT conditions

$$\lambda_i [y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1] = 0; i = 1, \dots, N$$

- from, condition (iii), it can be said that Very small percentage have  $\lambda_i > 0$ 
  - most among N, vanish with  $\lambda_i = 0$ .
  - $\mathbf{x}^{(i)}$  whose  $\lambda_i > 0$  are the *support vectors* and they lie on the margin.

- $\mathbf{w}$  is the weighted sum of these training instances that are selected as the support vectors:

$$\mathbf{w} = \sum_{i \in svindex} \lambda_i y^{(i)} \mathbf{x}^{(i)}$$

- where  $svindex$  is the set of indices of support vectors
- All support vectors are used to compute  $w_0$ , and then their average is taken for the final value of  $w_0$

$$w_0 = \frac{1}{|svindex|} \sum_{i \in svindex} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})$$

- where  $|svindex|$  is the total number of indices in  $svindex$ , i.e., total number of support vectors.

The majority of  $\lambda_i$  are 0, for which  $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) > 1$ . These are the  $\mathbf{x}^{(i)}$  points that exist more than adequately away from the discriminant, and have zero effect on the hyperplane. The instances that are not support vectors have no information; the same solution will be obtained on removing any subset from them. From this viewpoint, the SVM algorithm can be said to be similar to the  $k$ -NN algorithm (Section 3.4) which stores only the instances neighboring the class discriminant.

During testing, we do not enforce a margin. We calculate

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$$

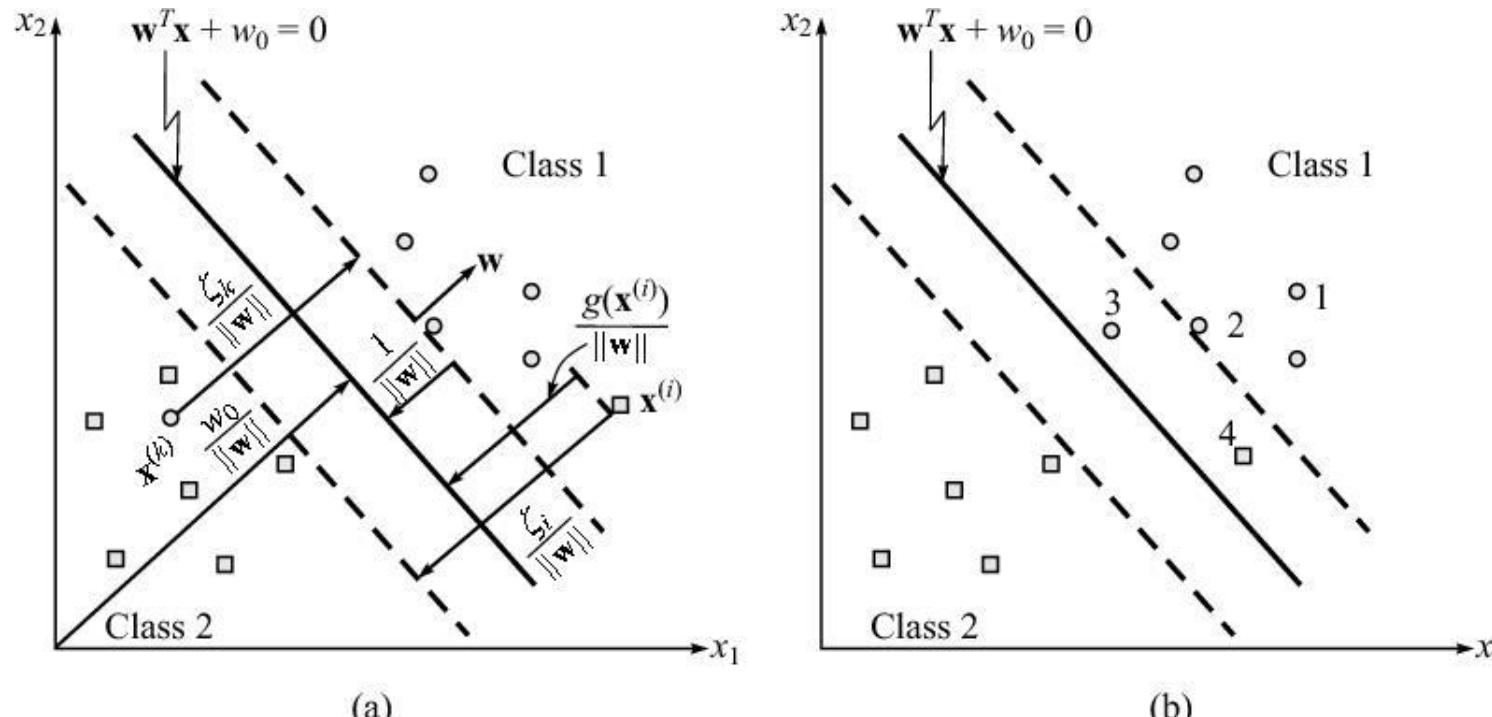
and choose the class according to the sign of  $g(\mathbf{x})$ :  $sgn(g(\mathbf{x}))$  which we call the *indicator function*  $i_F$ ,

$$i_F = \hat{y} = sgn(\mathbf{w}^T \mathbf{x} + w_0)$$

Choose Class 1 ( $\hat{y} = +1$ ) if  $\mathbf{w}^T \mathbf{x} + w_0 > 0$ , and Class 2 ( $\hat{y} = -1$ ) otherwise.

# Linear Soft Margin Classifier for Overlapping Classes

- To generalize SVM, allow noise in the training data.
- Hard margin linear SVM algorithm will not work.



Soft decision boundary

- To allow error in data, relax margin constraints by invoking *slack* variables  $\zeta_i (\geq 0)$ :

$$\begin{aligned}\mathbf{w}^T \mathbf{x}^{(i)} + w_0 &\geq 1 - \zeta_i \text{ for } y^{(i)} = +1 \\ \mathbf{w}^T \mathbf{x}^{(i)} + w_0 &\leq -1 + \zeta_i \text{ for } y^{(i)} = -1\end{aligned}$$

Thus, new constraints:

$$\begin{aligned}y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) &\geq 1 - \zeta_i; i = 1, \dots, N \\ \zeta_i &\geq 0\end{aligned}$$

Penalize the errors by assigning extra cost and change the objective function to

$$\frac{1}{2} \mathbf{w}^T \mathbf{w} + C \left( \sum_{i=1}^N \zeta_i \right); C \geq 0$$

where  $C$  is penalty parameter which is a trade-off parameter between margin and mistake

- Hence it all boils down to optimization problem

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \zeta_i \\ \text{subject to} \quad & y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \geq 1 - \zeta_i; i = 1, \dots, N \\ & \zeta_i \geq 0; i = 1, \dots, N \end{aligned}$$

This formulation is the *soft margin* SVM.

Lagrangian

$$\begin{aligned} & L(\mathbf{w}, w_0, \zeta, \lambda, \mu) \\ &= \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \zeta_i - \sum_{i=1}^N \lambda_i [y^{(i)} (\mathbf{w}^T \mathbf{x}^{(i)} + w_0) - 1 + \zeta_i] - \sum_{i=1}^N \mu_i \zeta_i \end{aligned}$$

where  $\mu_i, \lambda_i \geq 0$  are the dual variables.

Using KKT conditions, the dual formulation of the *soft-margin SVM* is reduced to

$$\text{maximize } L_*(\lambda) = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^N \lambda_i \lambda_k y^{(i)} y^{(k)} \mathbf{x}^{(i)T} \mathbf{x}^{(k)}$$

$$\begin{aligned} & \text{subject to } \sum_{i=1}^N \lambda_i y^{(i)} = 0 \\ & 0 \leq \lambda_i \leq C; i = 1, \dots, N \end{aligned}$$

- $\zeta_i$  and  $\mu_i$ - not in the dual objective function.
- The objective function is identical to that for separable case.
- Only difference - constraint  $\lambda_i \leq C$
- Can be solved numerically and  $\lambda_i$  values are used to compute  $\mathbf{w}$  and  $w_0$ .

$\lambda_i$  can have values in the interval  $0 \leq \lambda_i \leq C$ . thus three cases are there:

*Case 1:  $\lambda_i = 0$*

Don't contribute to the optimum value of  $\mathbf{w}$ .

*Case 2:  $0 < \lambda_i < C$*

Corresponding patterns are on the margin.

*Case 3:  $\lambda_i = C$*

Corresponding pattern is misclassified or lies inside the margin.

- support vectors define  $\mathbf{w}$  ( $\lambda_i > 0$ );  $\mathbf{w} = \sum_{i \in svindex} \lambda_i y^{(i)} \mathbf{x}^{(i)}$   
 $svindex$  is the set of indices of support vectors

- To compute  $w_0$ ,  $w_0 = \frac{1}{|svindex|} \sum_{i \in svindex} (y^{(i)} - \mathbf{w}^T \mathbf{x}^{(i)})$

$|svindex|$  is set of support vectors that fall on margin

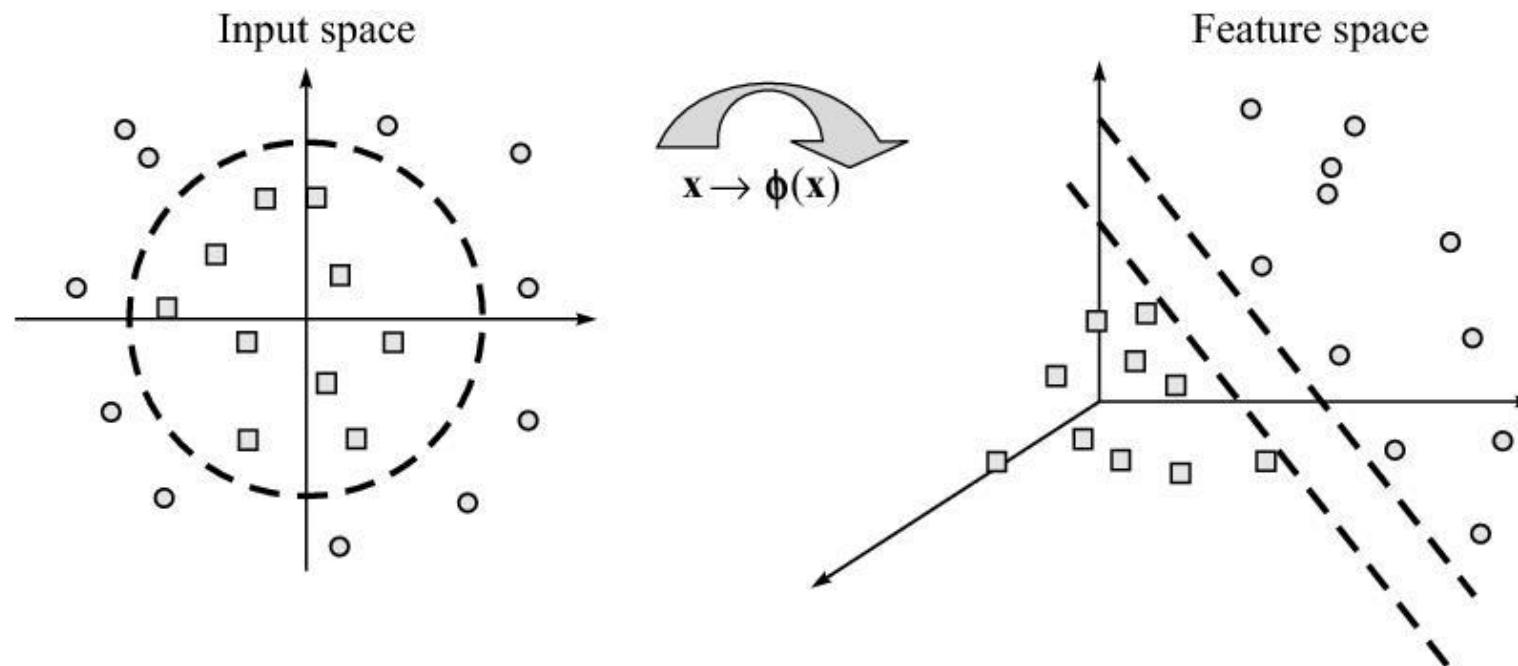
# Kernel Function: Dealing with non linearity

# Non-linear classifiers

- for several real-life datasets, the decision boundaries are nonlinear.
- To deal with nonlinear case, the formulation and solution methods employed for the linear case are still applicable.
- Only input data is transformed from its original space into another space (higher dimensional space) so that a linear decision boundary can separate Class 1 examples from Class 2.
- The transformed space is called the feature space.
- The original data space is known as the input space.

# Non-linear classifiers

- For training examples which cannot be linearly separated.
- In the feature space, they can be separated linearly with some transformations.



Transformation from input space to feature space

The new optimization problem becomes

$$\begin{aligned} & \text{minimize} \quad \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^N \zeta_i \\ \text{subject to} \quad & y^{(i)} (\mathbf{w}^T \Phi(\mathbf{x}^{(i)}) + w_0) \geq 1 - \zeta_i; i = 1, \dots, N \\ & \zeta_i \geq 0; i = 1, \dots, N \end{aligned}$$

The corresponding dual is

$$\text{minimize } L_*(\lambda) = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i=1}^N \sum_{k=1}^N \lambda_i \lambda_k y^{(i)} [\Phi(\mathbf{x}^{(i)})]^T \Phi(\mathbf{x}^{(k)})$$

$$\begin{aligned} \text{subject to} \quad & \sum_{i=1}^N \lambda_i y^{(i)} = 0 \\ & 0 \leq \lambda_i \leq C; i = 1, \dots, N \end{aligned}$$

The decision boundary becomes:

$$\sum_{i=1}^N \lambda_i y^{(i)} [\Phi(\mathbf{x}^{(i)})]^T \Phi(\mathbf{x}) + w_0 = 0$$

- Is there a need to know the mapping of  $\Phi$ ? No.

In SVM, this is done through the use of *kernel function*, denoted by  $K$ .

$$K(\mathbf{x}^{(i)}, \mathbf{x}) = [\Phi(\mathbf{x}^{(i)})]^T \Phi(\mathbf{x})$$

There is no explicit need to know what  $\Phi$  is.

## Constructing Kernels:

- Does any kernel work? No, only valid kernel functions work. Identification of  $\Phi$  is not needed if it can be shown whether the function is a kernel or not without the need of mapping.
- Function satisfying Mercer's theorem can work as kernel function.
- Mercer's theorem, which provides a test whether a function  $K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$  constitutes a valid kernel without having to construct the function  $\Phi(\mathbf{x})$ .

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = [\Phi(\mathbf{x}^{(i)})]^T \Phi(\mathbf{x}^{(k)})$$

# Mercer's theorem

$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)})$  is a kernel function if and only if the matrix  $\mathbf{K}$  is positive semidefinite.

$$\mathbf{K} = \begin{bmatrix} K(\mathbf{x}^{(1)}, \mathbf{x}^{(1)}) & K(\mathbf{x}^{(1)}, \mathbf{x}^{(2)}) & \dots & K(\mathbf{x}^{(1)}, \mathbf{x}^{(N)}) \\ \vdots & \vdots & K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) & \vdots \\ K(\mathbf{x}^{(N)}, \mathbf{x}^{(1)}) & K(\mathbf{x}^{(N)}, \mathbf{x}^{(2)}) & \dots & K(\mathbf{x}^{(N)}, \mathbf{x}^{(N)}) \end{bmatrix}$$

A positive semidefinite matrix is a Hermitian matrix (a complex square matrix that is equal to its own conjugate transpose) all of whose eigenvalues are nonnegative.

# Polynomial and Radial Basis Kernel

Common kernel functions used:

*Polynomial kernel of degree d*

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = (\mathbf{x}^{(i)T} \mathbf{x}^{(k)} + c)^d; c > 0, d \geq 2$$

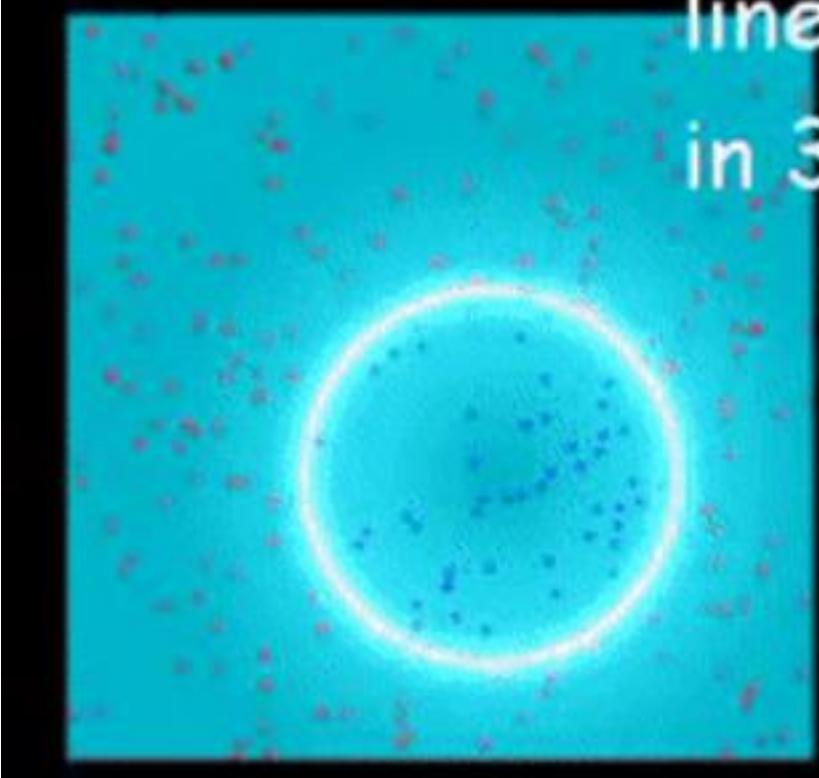
*Gaussian radial basis function kernel (RBF)*

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(k)}\|^2}{2\sigma^2}\right); \sigma > 0$$

Each of these results in a different nonlinear classifier in (the original) input space.

# Polynomial Kernel

Using a polynomial kernel they can be linearly separated in 3D space



# Polynomial Kernel

- The polynomial kernel represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models.
- It looks not only at the given features of input samples to determine their similarity, but also combinations of these (interaction features).
- Quite popular in natural language processing (NLP).
- The most common degree is  $d = 2$  (quadratic), since larger degrees tend to overfit on NLP problems.
- One problem with the polynomial kernel is that it may suffer from numerical instability: (result ranges from 0 to infinity)

# Radial Basis Kernel

- RBF kernels are the most generalized form of kernelization.
- It is one of the most widely used kernels due to its similarity to the Gaussian distribution.
- The RBF kernel function for two points  $X_1$  and  $X_2$  computes the similarity or how close they are to each other.

$$K(\mathbf{x}^{(i)}, \mathbf{x}^{(k)}) = \exp\left(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(k)}\|^2}{2\sigma^2}\right); \sigma > 0$$

where,

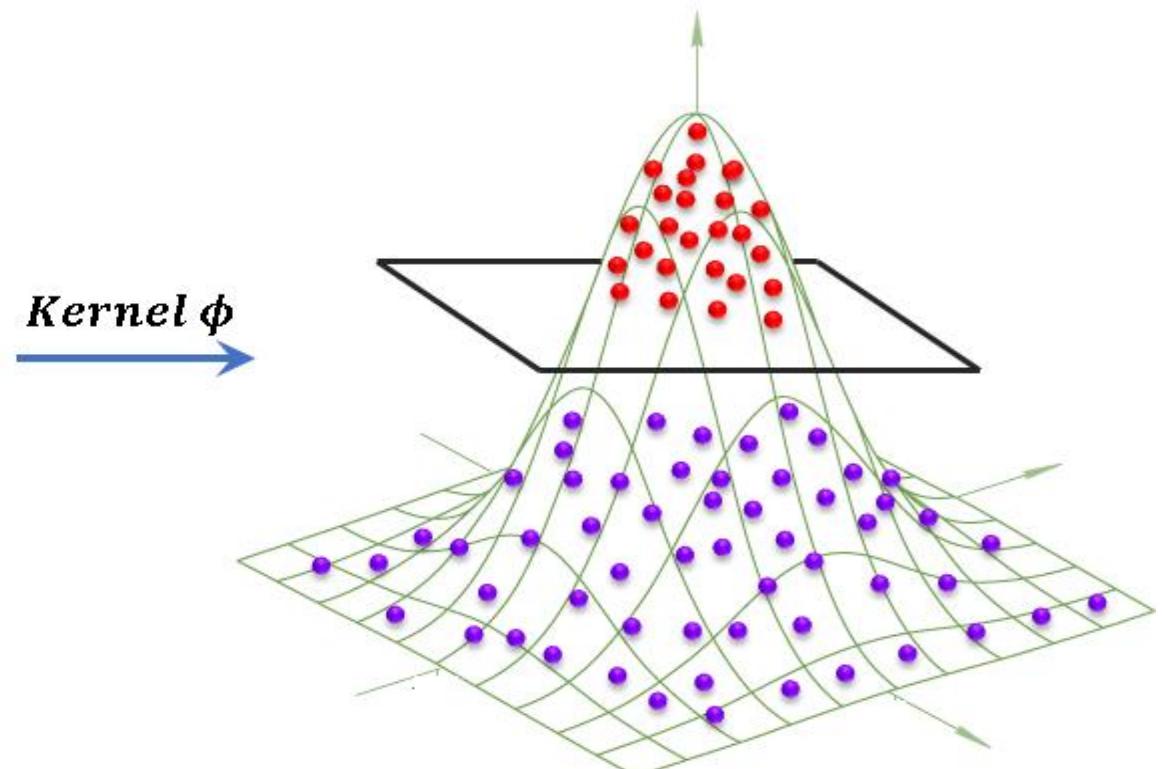
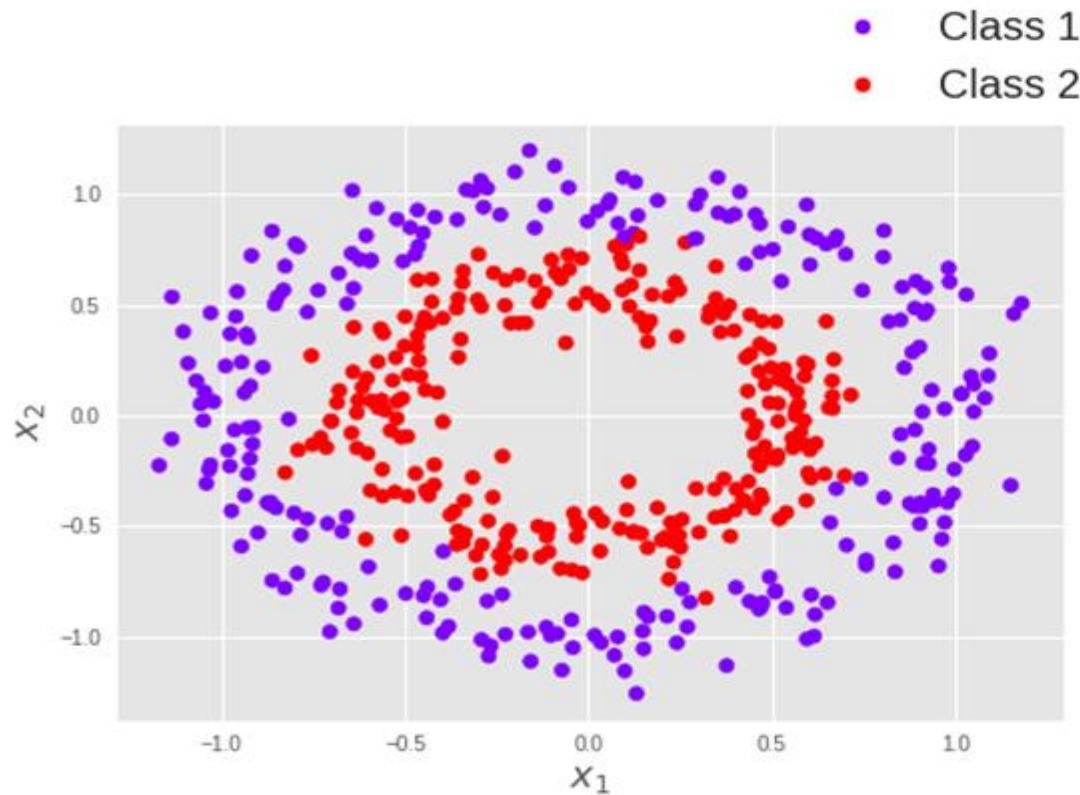
‘ $\sigma$ ’ is the variance and our hyperparameter

$\|\mathbf{X}_1 - \mathbf{X}_2\|$  is the Euclidean ( $L_2$ -norm) Distance between two points  $X_1$  and  $X_2$

# Radial Basis Kernel

- The maximum value that the RBF kernel can get is 1 and occurs when  $d_{12}$  is 0 which is when the points are the same, i.e.  $X_1 = X_2$ .
- When the points are the same, there is no distance between them and therefore they are extremely similar.
- When the points are separated by a large distance, then the kernel value is less than 1 and close to 0 which would mean that the points are dissimilar.

- There are no golden rules for determining which admissible kernel will result in the most accurate SVM.
- In practice, the kernel chosen does not generally make a large difference in resulting accuracy.
- SVM training always finds a global solution, unlike neural networks (to be discussed in the next chapter) where many local minima usually exist.



# NEURAL NETWORK

Dr. Srikanth Allamsetty

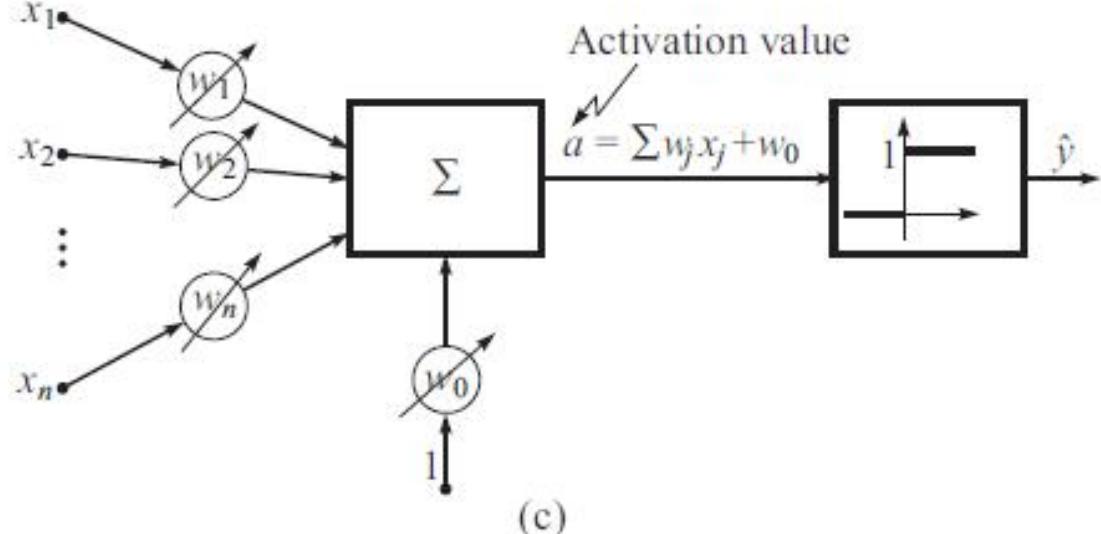
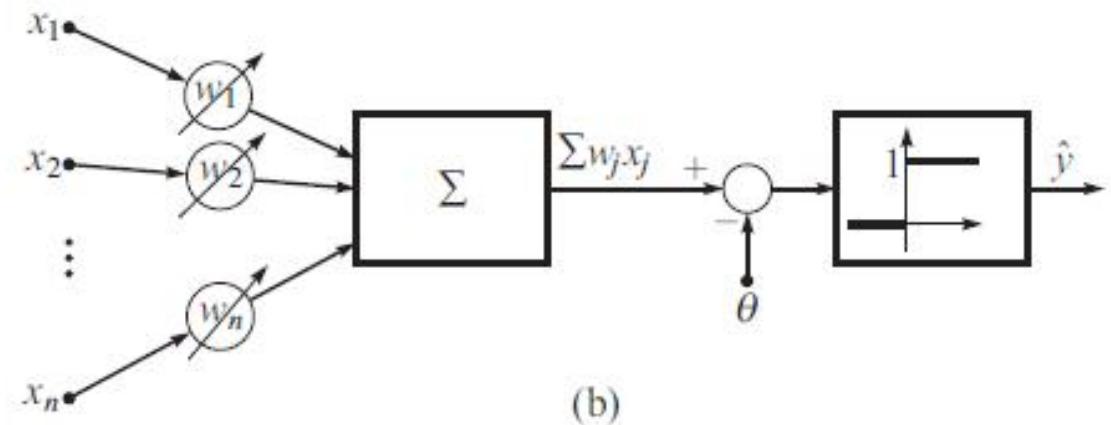
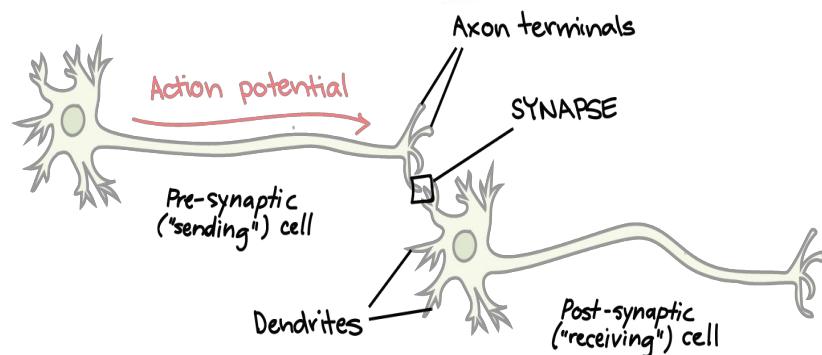
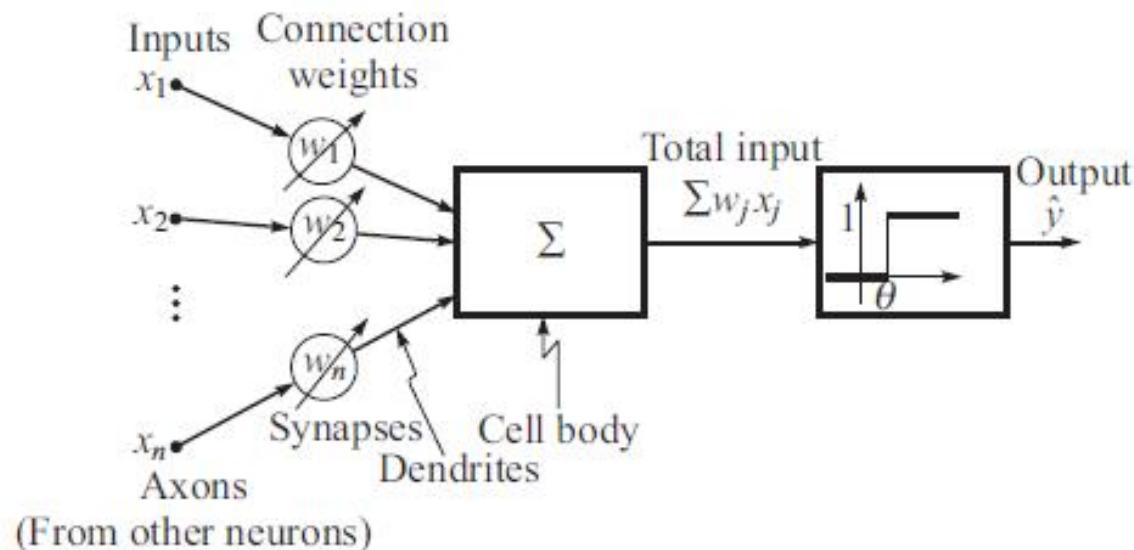
# McCulloch-Pitts Neuron Model

# Artificial Neuron?

- Evaluates the input signals, determining the strength of each one;
- Calculates a total for the combined input signals and compares that total to some threshold level;
- Determines what the output should be.
- Parts:
  - Input and Outputs;
  - Weighting Factors;
  - Activation Functions

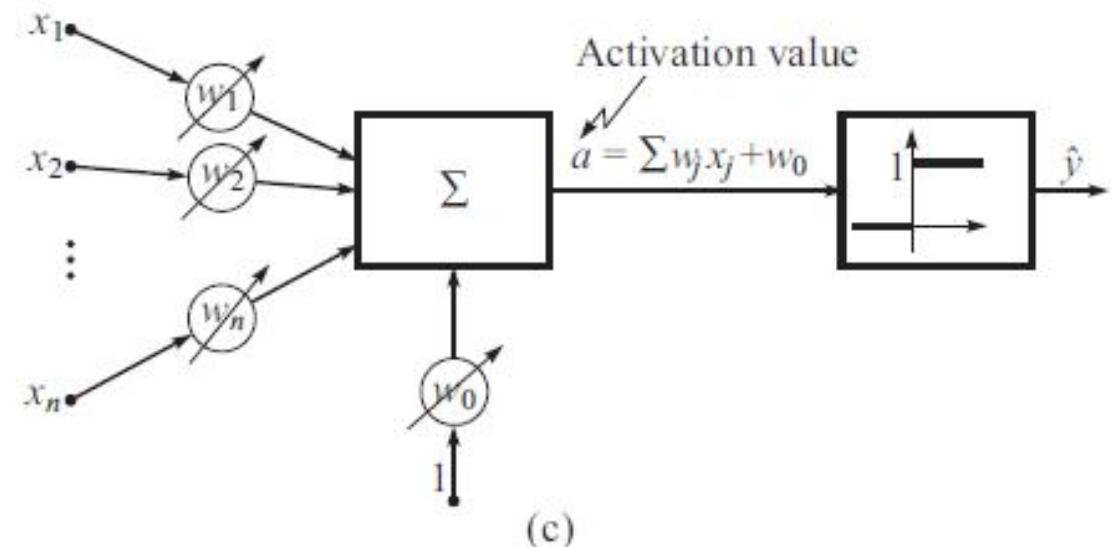
# McCulloch-Pitts Neuron Model

- The first formal definition of a synthetic neuron model was formulated by McCulloch and Pitts in 1943.

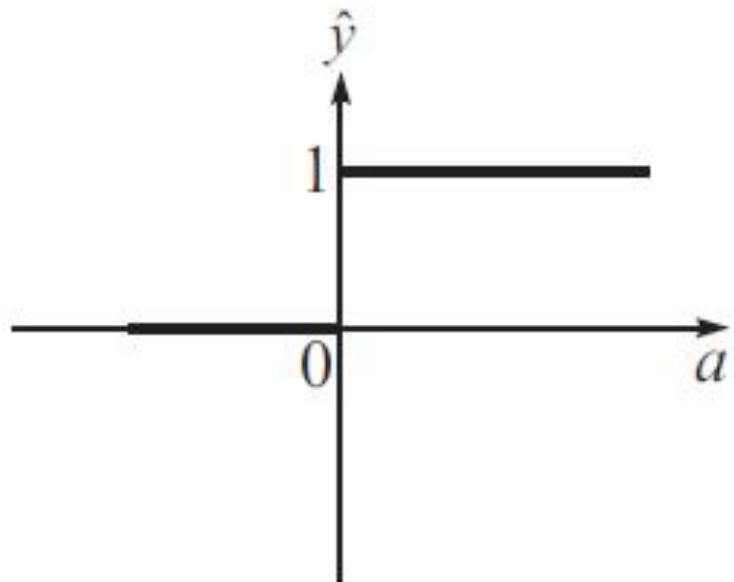


# Two important processes

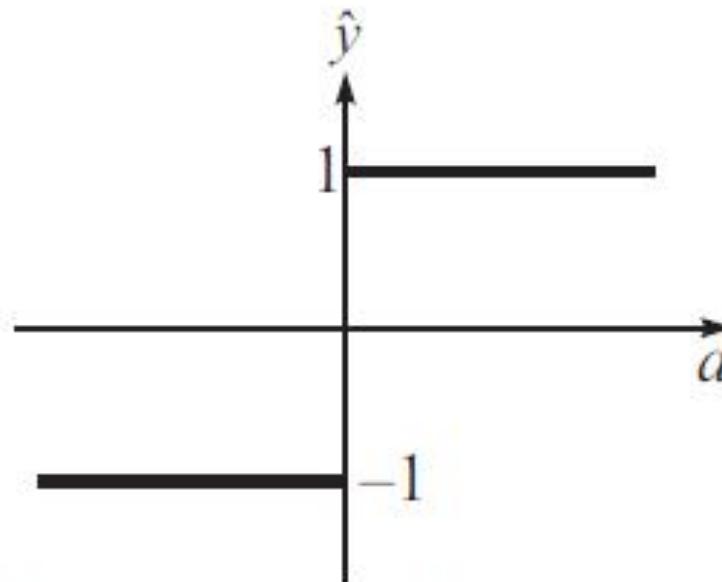
- Forming net activation by combining inputs. The input values are combined by a weighted additive process to achieve the neuron activation value  $a$  (refer to Fig. c).
- Mapping this activation value  $a$  into the neuron output  $\hat{y}$ . This mapping from activation to output may be characterized by an ‘activation’ or ‘squashing’ function.



# Squashing functions



(a) Unipolar squashing function



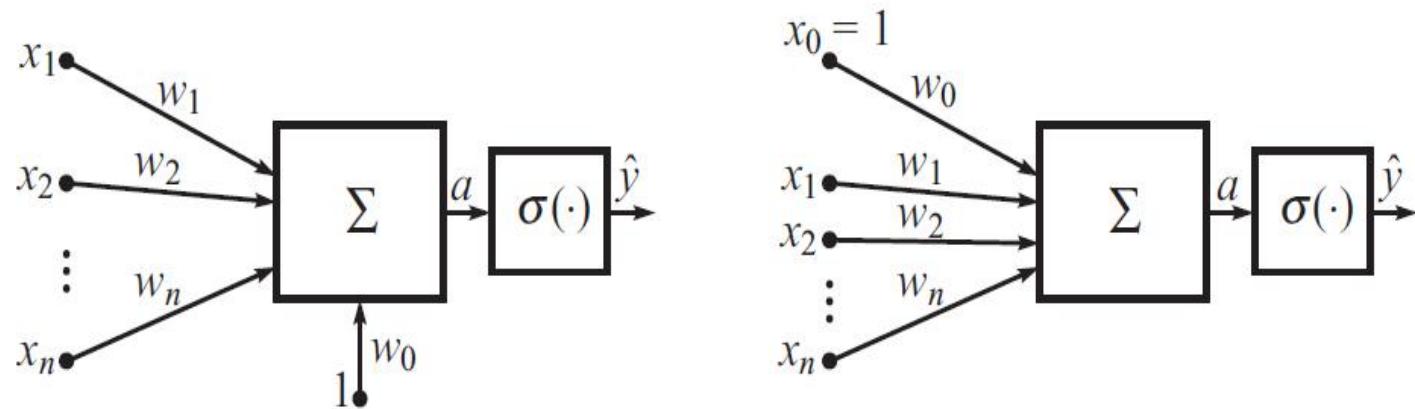
(b) Bipolar squashing function

The activation functions that implement input-to-output compression

# Perceptron Learning

# Artificial Neuron

- A mathematical equation for calculating an output value from a set of input values.



Mathematical model of a neuron (perceptron)

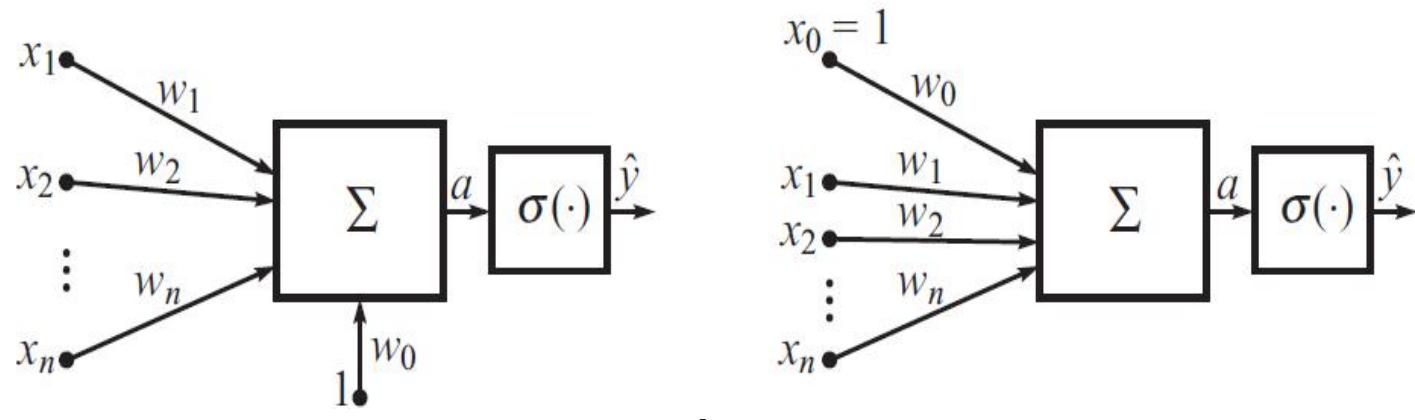
- Referred as a perceptron also.

- Input vector

$$\mathbf{x} = [x_1, x_2 \dots x_n]^T;$$

- Connection weight vector

$$\mathbf{w}^T = [w_1, w_2 \dots w_n];$$



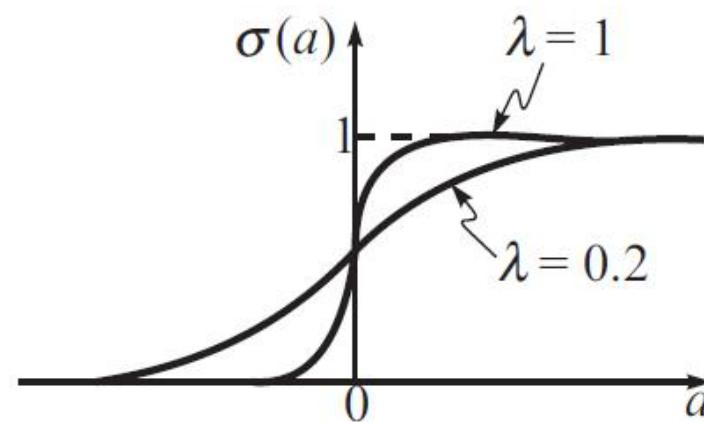
- Relation of input  $\mathbf{x}$  and the output  $\hat{y}$ :

$$\hat{y} = \sigma(\mathbf{w}^T \mathbf{x} + w_0) = \sigma\left(\sum_{j=1}^n w_j x_j + w_0\right)$$

- Where  $\sigma(\cdot)$  is the activation function (transfer function) of the neuron.
- The weights are always adaptive.
- The first hidden layer may have the log-sigmoid or tan-sigmoid activation function:

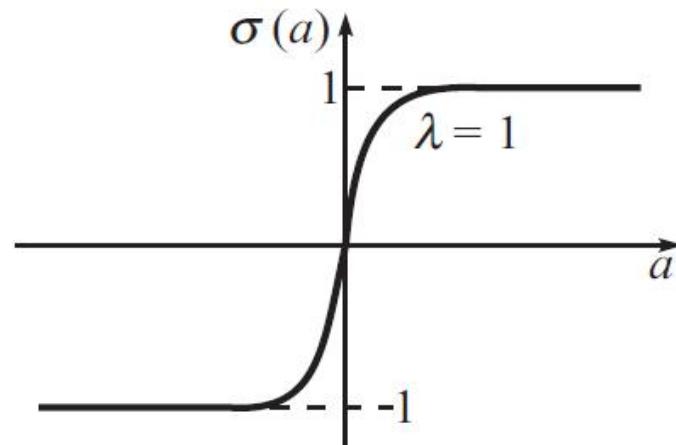
### Log-sigmoid activation function

$$\sigma(a) = \frac{1}{1+e^{-a}}$$



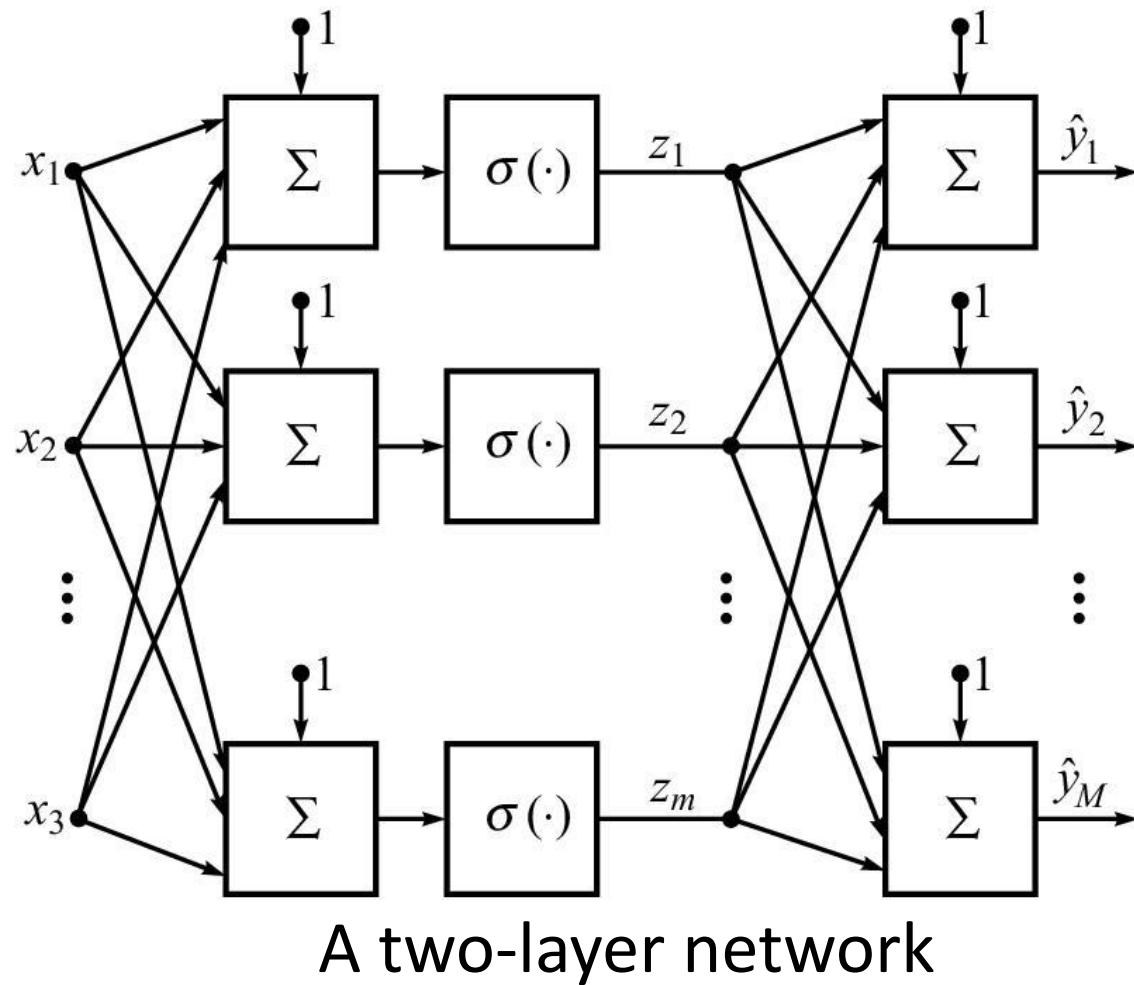
Or tan-sigmoid activation function:

$$\sigma(a) = \frac{1-e^{-a}}{1+e^{-a}}$$



- The second (output) layer having linear activation function  
 $\sigma(a) = a$

# Feedforward Networks



# Feedforward Networks

- The most commonly used network architecture, as it works quite well in many practical applications.
- The input terminals are defined as:

$$x_j; j = 1, \dots, n$$

- The hidden-layer outputs as  $z_l$ , maybe expressed as:

$$\begin{aligned} z_l &= \sigma \left( \sum_{j=1}^n w_{ij} x_j + w_{l0} \right); l = 1, 2, \dots, m \\ &= \sigma(\mathbf{w}_l^T \mathbf{x} + w_{l0}) \end{aligned}$$

- weights connecting input terminals to hidden layer

$$\mathbf{w}_l^T \triangleq [w_{l1} w_{l2} \dots w_{ln}]$$

- Output-layer nodes as  $\hat{y}_q$ , we may write:

$$\begin{aligned}\hat{y}_q &= \left( \sum_{l=1}^m v_{ql} z_l + v_{q0} \right); q=1, \dots, M \\ &= \mathbf{v}_q^T \mathbf{z} + v_{q0}\end{aligned}$$

- weights connecting hidden layer to output layer

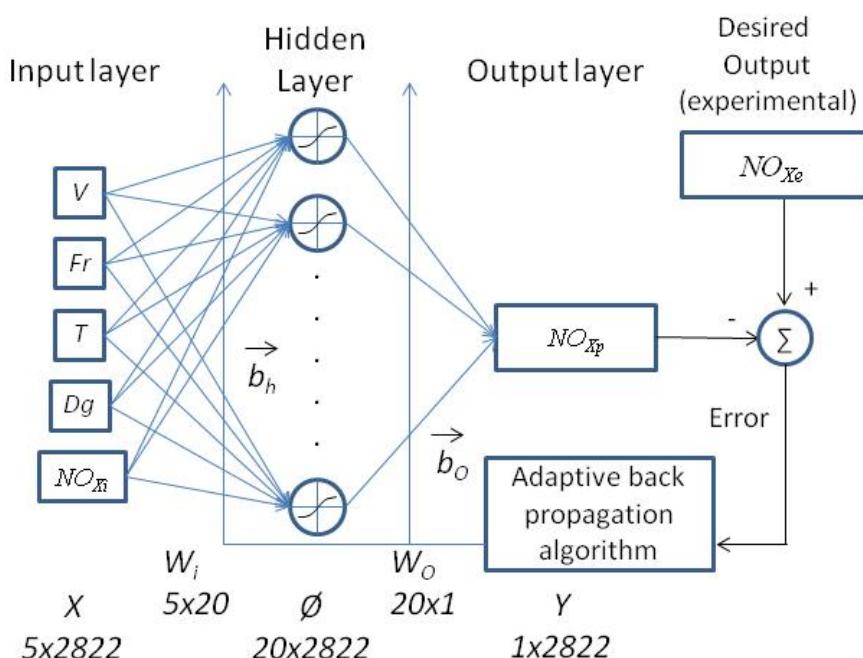
$$\mathbf{v}_q^T \triangleq [v_{q1} v_{q2} \dots v_{qm}]$$

- For the multiclass discrimination problems, we may use two-layer feedforward neural networks with sigmoidal/hyperbolic tangent hidden units and sigmoidal output units.
- The NN output of this multilayer structure may be written as:

$$\hat{y}_q = \sigma \left( \sum_{l=1}^m v_{ql} z_l + v_{q0} \right); q = 1, \dots, M$$

# ANN based models in my Research work

Observe the sizes of matrices

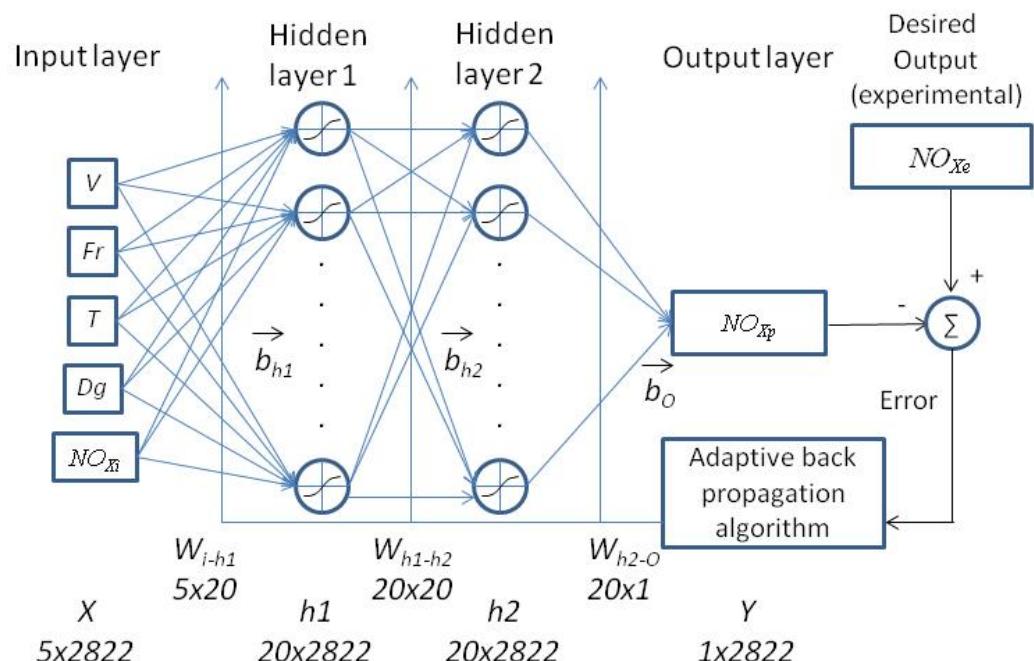


$$NO_{xp} = \sum_{i=1}^{20} [W_o(i)' \times \emptyset(i) + b_o(i)]$$

where  $\emptyset(i) = \emptyset(\sum_{i=1}^5 [W_i(i)' \times X(i) + b_h(i)])$ ;

$$\text{and } \emptyset(x) = \frac{1}{(1+e^{-x})}$$

ANN



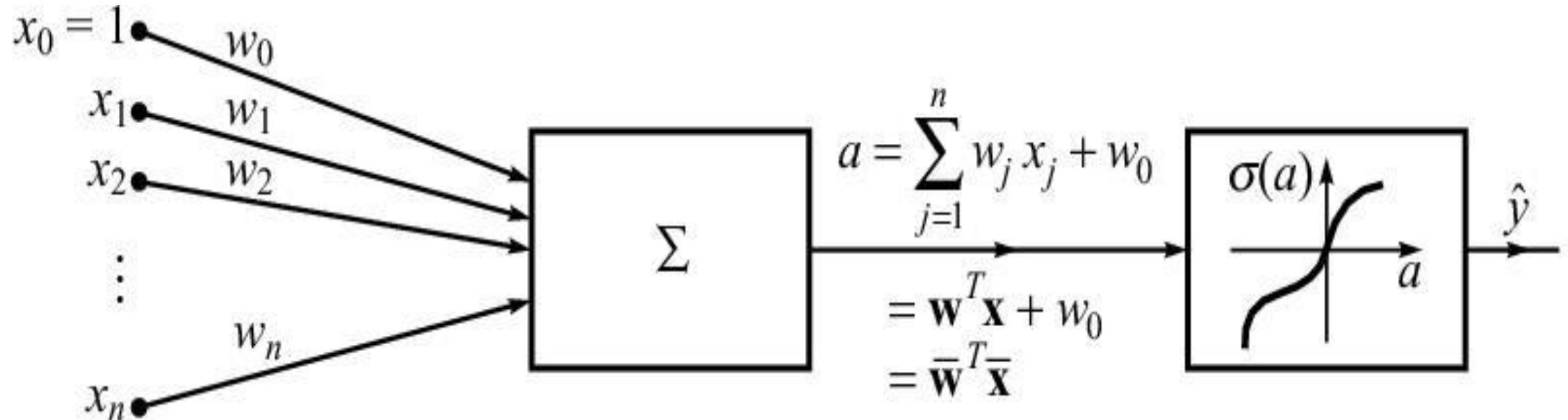
$$NO_{xp} = \sum_{i=1}^{20} [W_{h2-o}(i)' \times \emptyset_{h2}(i) + b_o(i)]$$

$$\emptyset_{h2}(i) = \emptyset(\sum_{i=1}^{20} [W_{h1-h2}(i)' \times \emptyset_{h1}(i) + b_{h2}(i)]) ;$$

$$\text{and } \emptyset_{h1}(i) = \emptyset(\sum_{i=1}^5 [W_{i-h2}(i)' \times X(i) + b_{h1}(i)]) .$$

MLP

# The error correction delta rule



Neural unit with any differentiable activation function

- A neural unit has a differentiable function  $\sigma(a)$ .
- First a linear combination of its inputs (activation value  $a$ ) are computed;
- Then a nonlinear activation function  $\sigma(a)$  is applied to the result.
- The output  $\hat{y}$  of nonlinear unit is a continuous function of its input  $a$ .
- The problem is to find the expression for the learning rule for adapting weights using a training set of pairs of input and output patterns.

- The error function  $E(k)$  maybe defined as:

$$E(k) = \frac{1}{2} (y^{(i)} - \hat{y}(k))^2 = \frac{1}{2} [e(k)]^2;$$

$$e(k) = y^{(i)} - \hat{y}(k)$$

$$\hat{y}(k) = \sigma \left( \sum_{j=1}^n w_j(k) x_j^{(i)} + w_0(k) \right); \quad a = \sum_{j=1}^n w_j x_j + w_0$$

- For each training example  $i$ , weights  $w_j; j = 1, 2, \dots, n$  (and bias  $w_0$ ) are updated by adding to it  $\Delta w_j$  (and  $\Delta w_0$ ).

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

$$w_j(k+1) = w_j(k) - \eta \frac{\partial E(k)}{\partial w_j(k)}$$

$$w_0(k+1) = w_0(k) - \eta \frac{\partial E(k)}{\partial w_0(k)}$$

**Applying the chain rule:**

$$\frac{\partial E(k)}{\partial w_j(k)} = \frac{\partial E(k)}{\partial e(k)} \frac{\partial e(k)}{\partial \hat{y}(k)} \frac{\partial \hat{y}(k)}{\partial a(k)} \frac{\partial a(k)}{\partial w_j(k)}$$

$$= e(k) [-1] \frac{\partial \sigma(a(k))}{\partial a(k)} x_j^{(i)}$$

$$= -e(k) \sigma'(a(k)) x_j^{(i)}$$

- The learning rule can be written as:

$$w_j(k+1) = w_j(k) + \eta e(k) \sigma'(a(k)) x_j^{(i)}$$

$$w_0(k+1) = w_0(k) + \eta e(k) \sigma'(a(k)) x_0^{(i)}$$

- Also known as delta learning rule with delta defined as:

$$\delta(k) = e(k) \sigma'(a(k))$$

- In terms of  $\delta(k)$ , the weights-update equations become

$$w_j(k+1) = w_j(k) + \eta \delta(k) x_j^{(i)}$$

$$w_0(k+1) = w_0(k) + \eta \delta(k)$$

- Interestingly, for a linear activation function

$$\sigma(a(k)) = a(k)$$

Therefore,

$$\sigma'(a(k)) = 1$$

And

$$\delta(k) = e(k) \sigma'(a(k)) = e(k)$$

For log-sigmoid

$$\sigma(a) = \frac{1}{1+e^{-a}} \text{ and } \sigma'(a(k)) = \hat{y}(1 - \hat{y})$$

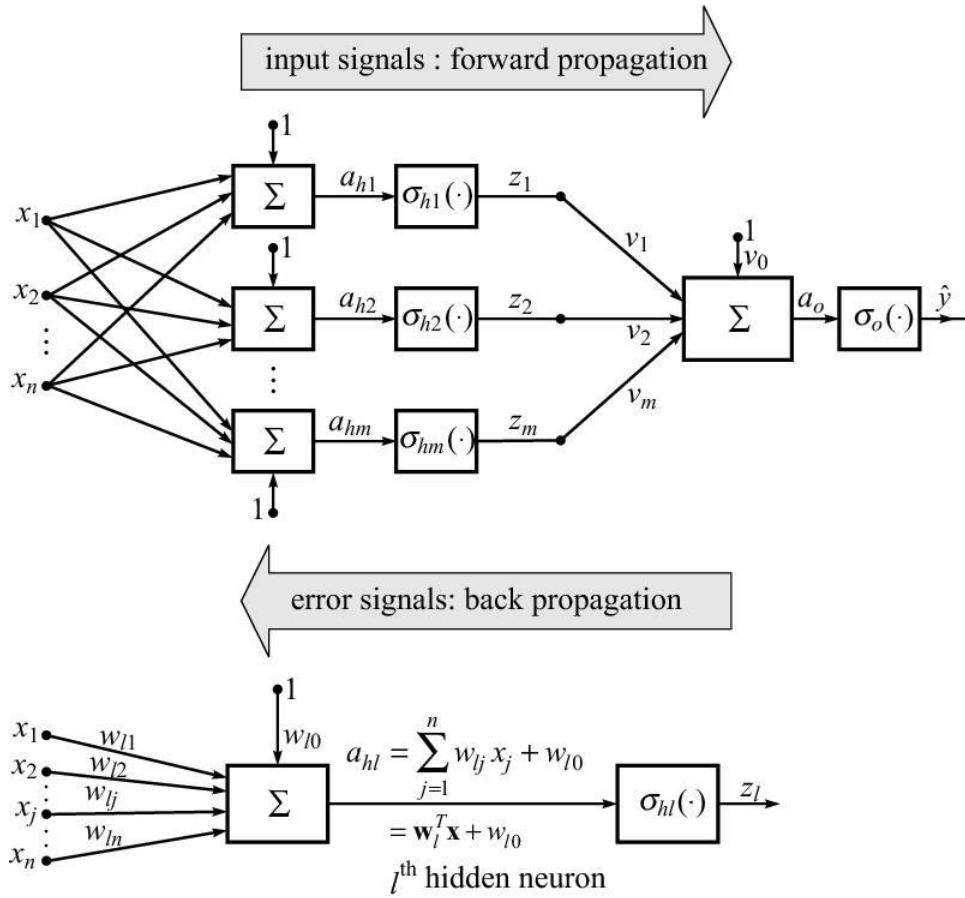
Where  $\hat{y} = \sigma(a)$

## Training Protocols

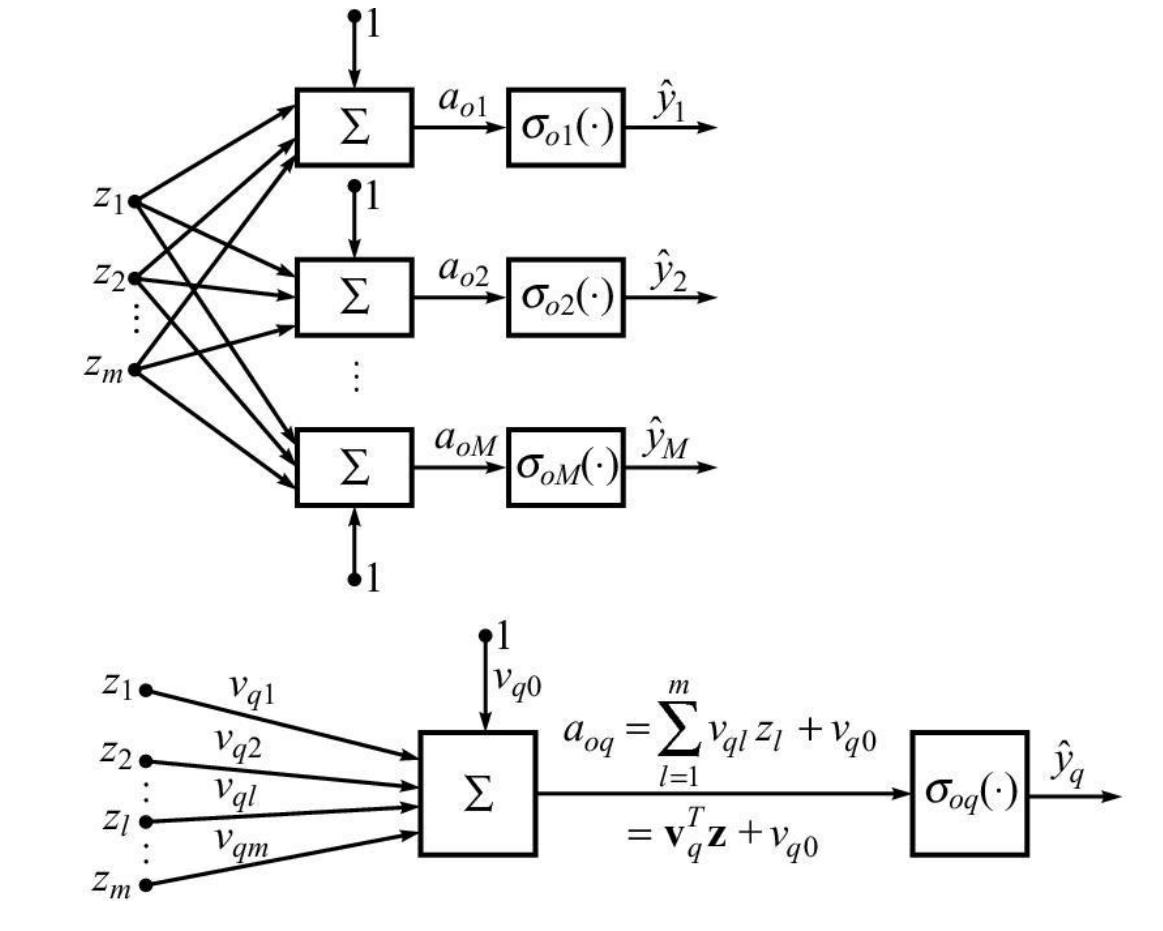
- The *batch training* rule computes the weight updates after summing error over all the training examples (*batch of training data*).
- Incremental training rule is to update weights incrementally, following the calculation of error for each individual example.

# Back-propagation

# Error-Backpropagation algorithm

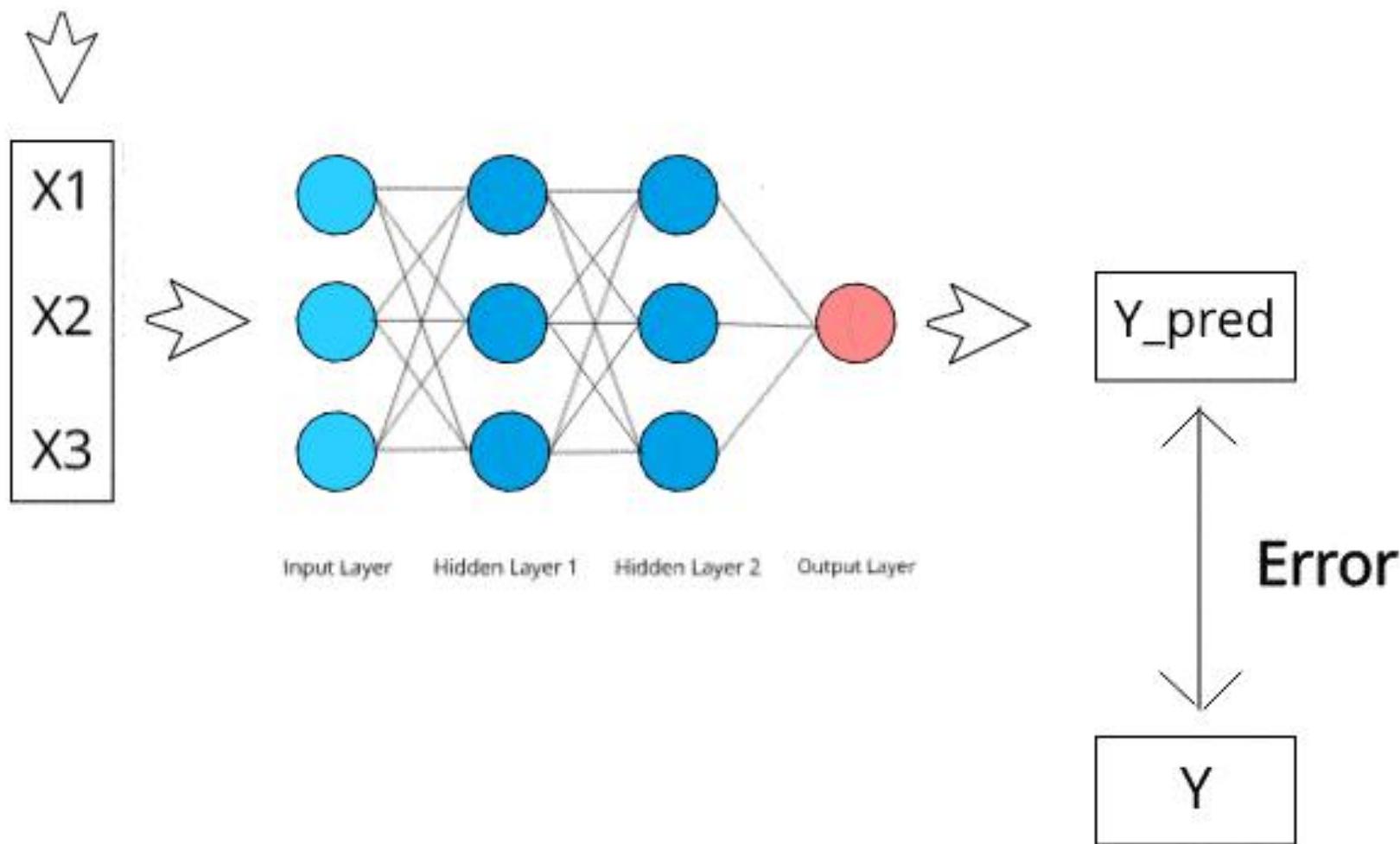


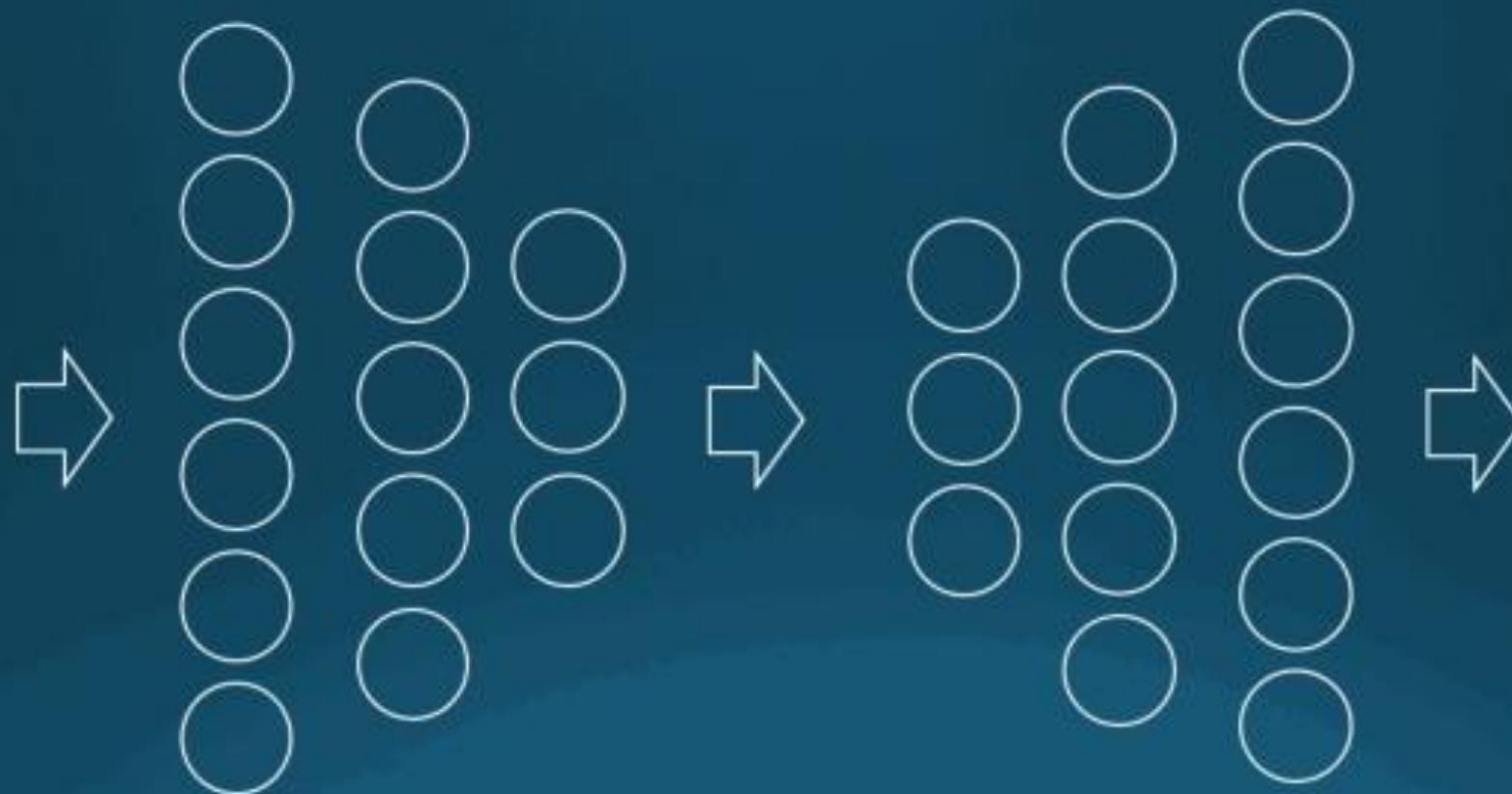
Scalar-output MLP network



Output layer of vector-output MLP network

Feed new data





# Neural networks training in a nutshell

For a number of iterations\*:

1. Go through the data  $y^{(i)}$  and forward it through the network
2. Make predictions  $\hat{y}^{(i)}$
3. Compute the loss of the network between  $y^{(i)}$  and  $\hat{y}^{(i)}$
4. Compute the gradients of the loss w.r.t weights of the network
5. Apply gradient descent to update the weights

\* : iterations through all the data are also called epochs

# Multi Layer Perceptron

Weights update equations employing incremental training for the multi-output structure (log-sigmoid output units)

- Forward recursion to compute MLP output
- Present the input  $x^{(i)}$  to the MLP network, and compute the output using

$$z_l(k) = \sigma_{hl} \left( \sum_{j=1}^n w_{ij}(k)x_j^{(i)} + w_{l0}(k) \right); l = 1, \dots, m$$

$$\hat{y}_q(k) = \sigma_{oq} \left( \sum_{l=1}^m (\nu_{ql}(k)z_l(k)) + \nu_{q0}(k) \right); q = 1, \dots, M$$

- With initial weights  $w_{ij}$ ,  $w_{l0}$ ,  $w_{ql}$ ,  $w_{q0}$ , randomly chosen.
- Backward recursion for backpropagated errors

$$\delta_{oq}(k) = \left[ y_q^{(i)} - \hat{y}_q(k) \right] \hat{y}_q(k) [1 - \hat{y}_q(k)]$$

$$\delta_{hl}(k) = z_l(k) [1 - z_l(k)] \sum_{q=1}^M \delta_{oq}(k) v_{ql}(k)$$

- Computation of MLP weights and bias updates

$$v_{ql}(k+1) = v_{ql}(k) + \eta \delta_{oq}(k) z_l(k)$$

$$v_{q0}(k+1) = v_{q0}(k) + \eta \delta_{oq}(k)$$

$$w_{ij}(k+1) = w_{ij}(k) + \eta \delta_{hl}(k) x_j^{(i)}$$

$$w_{l0}(k+1) = w_{l0}(k) + \eta \delta_{hl}(k)$$

## Weights-update equations employing batch training for the multi-output structure (log-sigmoid output units)

- The initial weights are randomly chosen, and iteration index is set 0.
- All the training data pairs,  $i=1,\dots,N$ , are presented to the network before the algorithm moves to iteration index  $k+1$  to update the weights.
- Weights-update is done using the following equations:

$$z_l^{(i)} = \sigma_{hl} \left( \sum_{l=1}^m w_{lj}(k) x_j^{(i)} + w_{l0}(k) \right)$$

$$\hat{y}_q^{(i)} = \sigma_{oq} \left( \sum_{l=1}^m (v_{ql}(k) z_l^{(i)}) + v_{q0}(k) \right)$$

$$\delta_{oq}^{(i)} = (y_q^{(i)} - \hat{y}_q^{(i)})\hat{y}_q^{(i)}(1-\hat{y}_q^{(i)})$$

$$\Delta v_{ql}(k) = \eta \sum_{i=1}^M \delta_{oq}^{(i)} z_l^{(i)}$$

$$\delta_{hl}^{(i)} = z_l^{(i)}(1-z_l^{(i)}) \sum_{q=1} \delta_{oq}^{(i)} v_{ql}(k)$$

$$\Delta w_{lj}(k) = \eta \sum_{i=1}^N \delta_{hl}^{(i)} x_j^{(i)}$$

$$v_{ql}(k+1)=v_{ql}(\overline{k})+\Delta v_{ql}(k)\\w_{lj}(k+1)=w_{li}(k)+\Delta w_{lj}(k)$$

# Multi-class discrimination with MLP networks

- Multi-layer perceptron (MLP) networks can be used to realize nonlinear classifier.
- Most real-world problems require nonlinear classification.
- A realistic approach is 1-of-M encoding.
- A separate node is used to represent each possible class and the target vectors consist of 0s everywhere except for in the element that corresponds to the correct class.
- For a four class ( $M=4$ ) discrimination problem, the target vector

$$y^{(i)} = [y_1^{(i)} \ y_2^{(i)} \ y_3^{(i)} \ y_4^{(i)}]$$

- To encode four possible classes, we use

$\mathbf{y}^{(i)} = [1 \ 0 \ 0 \ 0]^T$  to encode class 1,

$\mathbf{y}^{(i)} = [0 \ 1 \ 0 \ 0]^T$  to encode class 2,

$\mathbf{y}^{(i)} = [0 \ 0 \ 1 \ 0]^T$  to encode class 3,

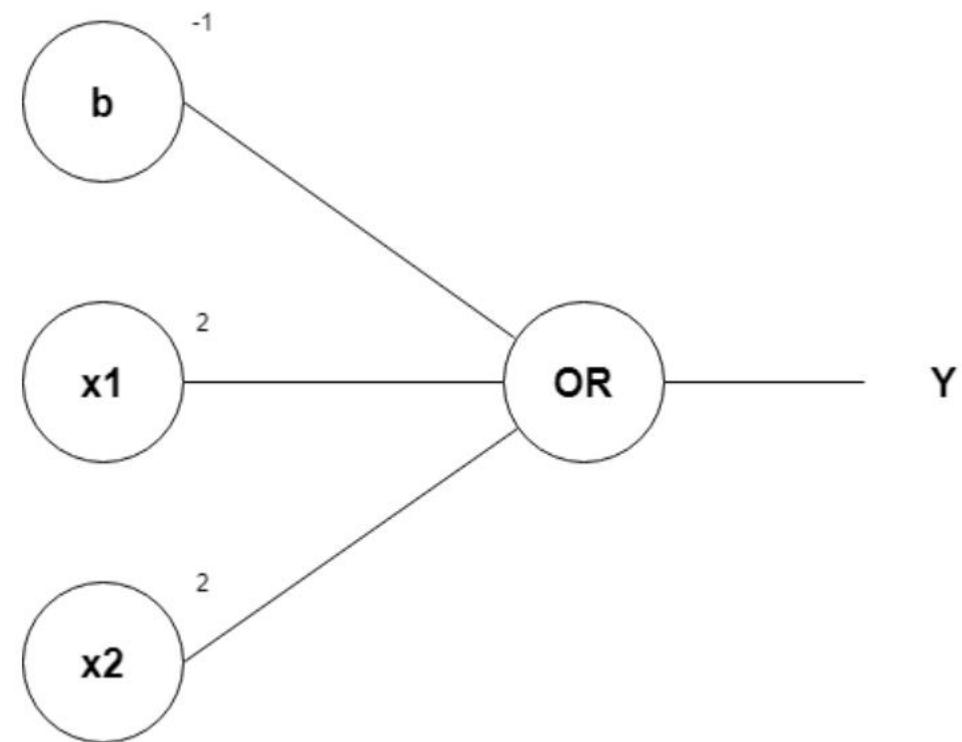
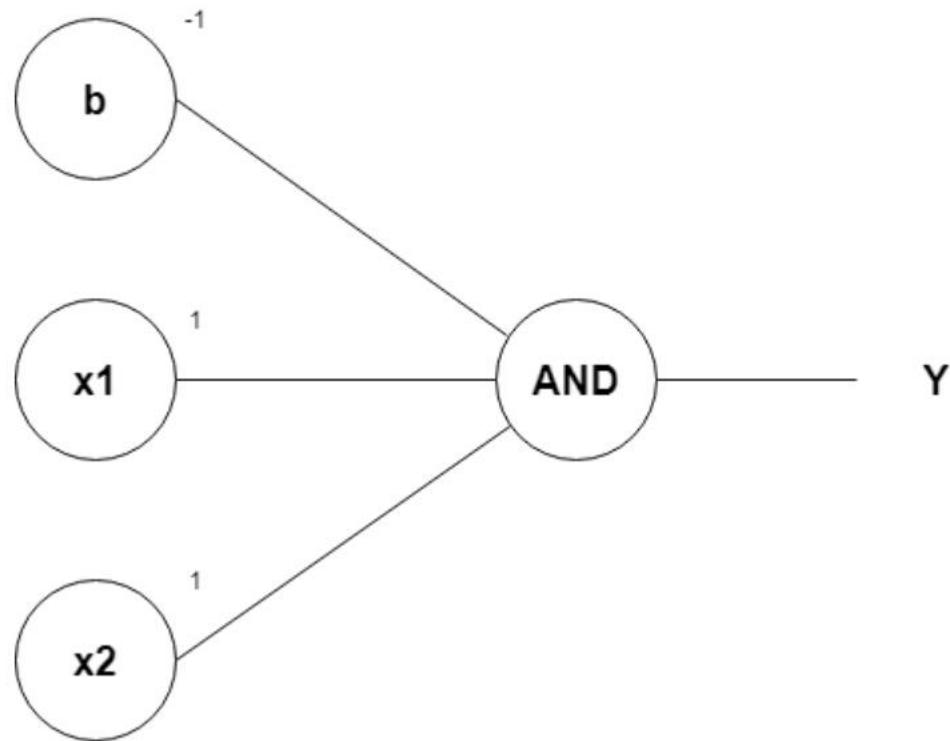
$\mathbf{y}^{(i)} = [0 \ 0 \ 0 \ 1]^T$  to encode class 4.

- Four log-sigmoid nodes are used in the output layer of the MLP network.
- Binary output values are used.

- Once the network has been trained, performing the classification is easy.
- Simply choose the element  $\hat{y}_k$  of the output vector that is the largest element of  $\mathbf{y}$ , i.e.,  $\hat{y}_k$  for which
$$\hat{y}_l > \hat{y}_q \quad \forall q \neq k; q = 1, \dots, M.$$
- This generates an unambiguous decision since it is very unlikely that two output neurons will have identical output values.

# Non-linear Problem Solving

# AND and OR gates using ANN



# Examples

- <https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1>
- <https://www.javatpoint.com/pytorch-backpropagation-process-in-deep-neural-network>

or (same)
- <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>
- <https://www.anotsorandomwalk.com/backpropagation-example-with-numbers-step-by-step/>
- <https://hmkcode.com/ai/backpropagation-step-by-step/>