



Introduction to ML

UNIT – 1

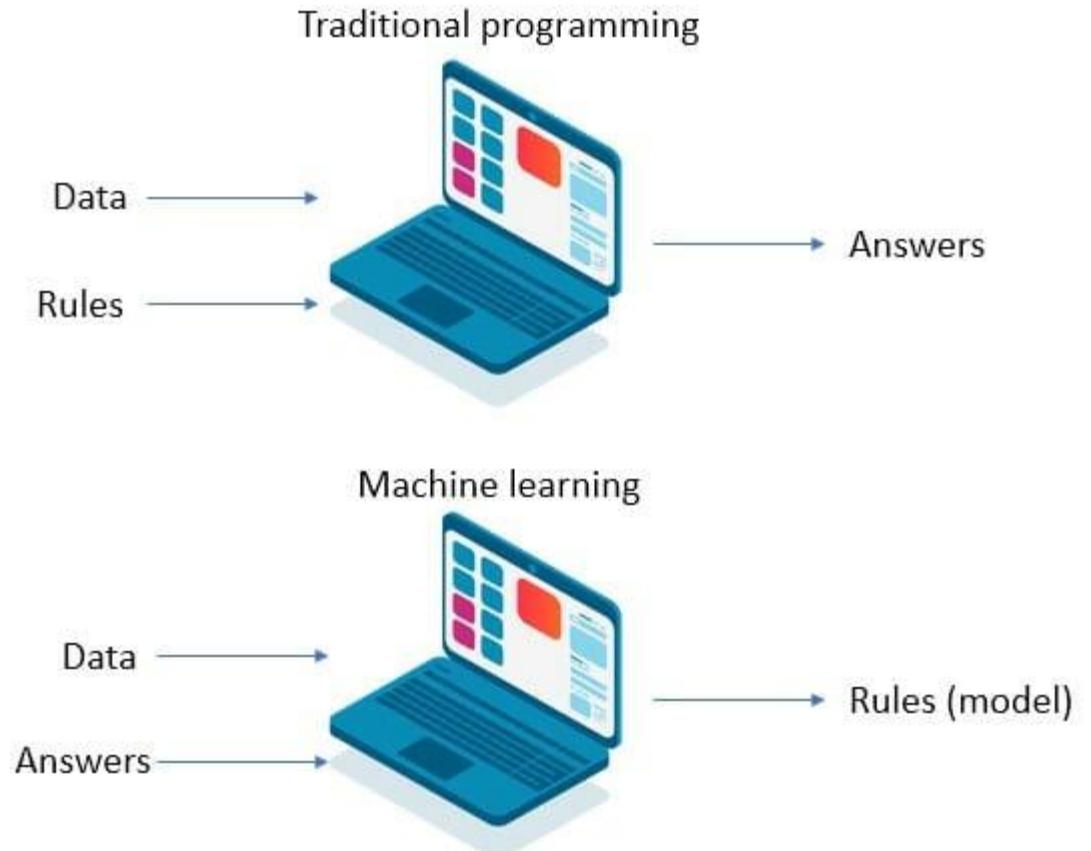
PART - 1

Topics Covered

- ▶ Applications of Machine Learning
- ▶ Well-Posed Learning Problems
- ▶ Designing a Learning System
- ▶ Issues in Machine Learning
- ▶ Types of Machine Learning

What is Machine Learning?

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy

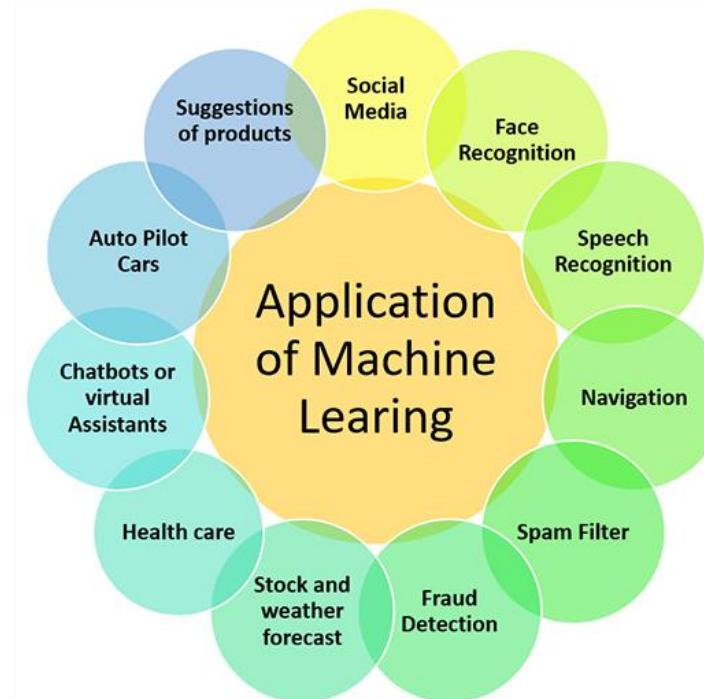


What is Machine Learning

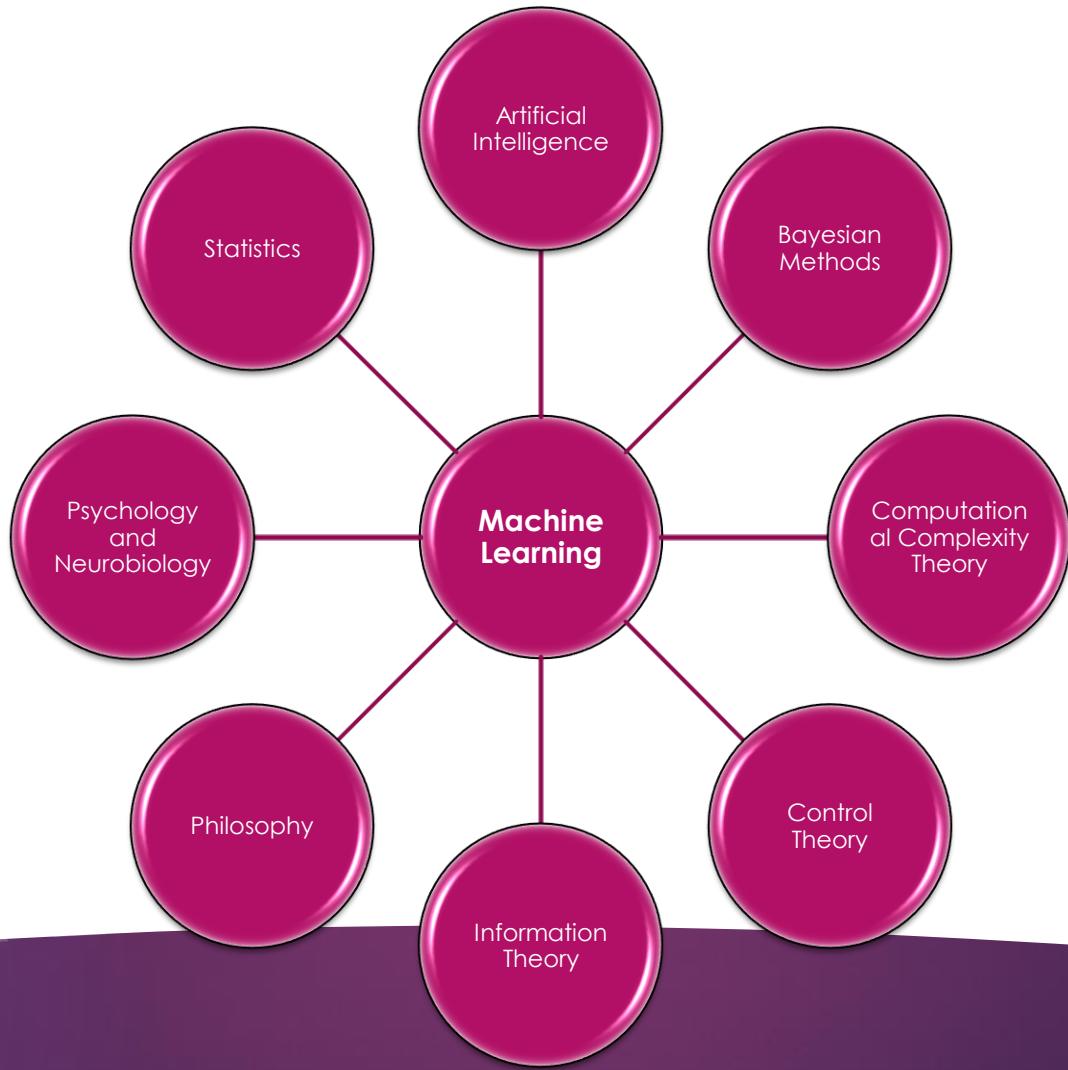
Machine learning is a “*Field of study that gives computers the ability to learn without being explicitly programmed*”
- *Arthur Samuel (1959)*

Applications of Machine Learning

- ▶ Learning to recognize spoken words
- ▶ Learning to drive an autonomous vehicle
- ▶ Learning to classify new astronomical structures
- ▶ Learning to play world-class backgammon
- ▶ Predict recovery rates of Pneumonia patients
- ▶ Detect Fraudulent use of credit cards



ML will play an increasingly central role in Computer Science and Technology



Machine Learning is a Multidisciplinary Field

ML draws results from AI, P&S, Computational complexity theory, control theory, information theory, philosophy, psychology, neurobiology and many other fields.

Key ideas and influence of some disciplines on ML

► Artificial Intelligence

- ▶ Learning symbolic representations of concepts
- ▶ ML as a search problem
- ▶ Learning as an approach to improving problem solving

► Bayesian Methods

- ▶ Bayes Theorem – basis for calculating probabilities of hypothesis

► Computational Complexity Theory

- ▶ Theoretical bounds on the complexity of learning tasks...computational effort, no. of training examples, no. of mistakes, required to learn

► Control Theory

- ▶ Procedures to optimize predefined objectives.

► Information Theory

- ▶ Measures of entropy and information content

► Philosophy

- ▶ Occam's razor – simplest hypothesis is the best

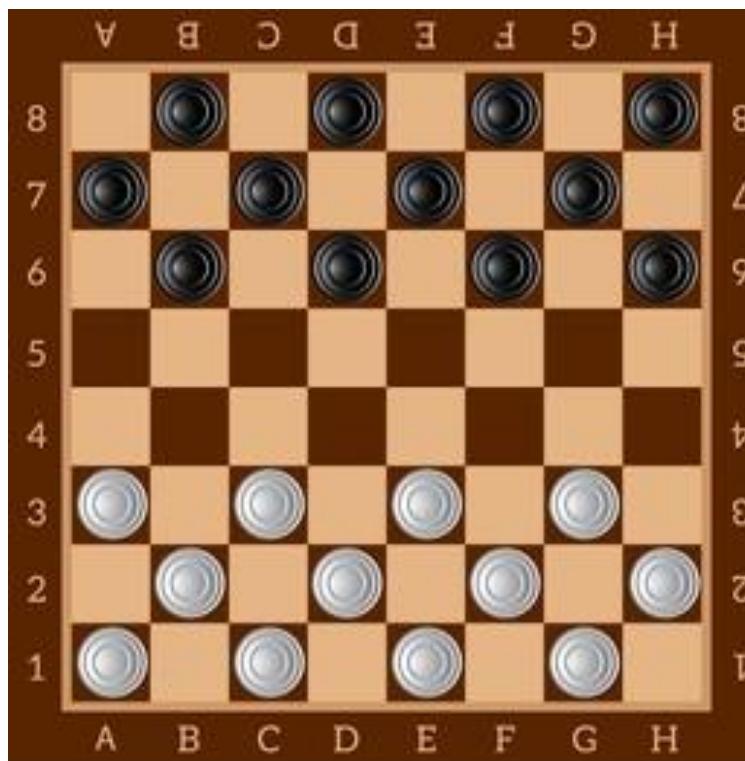
► Psychology and Neurobiology

- ▶ The power law of practice – people's response time increases with practice according to a power law.
- ▶ Motivation for neural network models

► Statistics

- ▶ Characterization of errors that occur when estimating the accuracy

Checkers Game



Well-Posed Learning Problems

Definition

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**.

Well-Posed Learning Problem

- ▶ To have a well defined learning problem, we must identify 3 features
 - ▶ The class of tasks
 - ▶ The measure of performance to be improved
 - ▶ The source of experience

Task “I want to learn Italian”



Training Experience

A performance measure

“Bionji”... “ Buonyo”



“Buongiorno”



Well Posed Learning Problems

Example : A Checkers Learning Problem

Task T	Performance Measure P	Training Experience E
Playing Checkers	Percent of games won against opponents	Playing Practice Games against itself

Well Posed Learning Problems

Example : A hand writing recognition
Learning Problem

Task **T**

Recognising and classifying handwritten words within images

Performance Measure **P**

Percentage of words correctly classified

Training Experience **E**

A database of handwritten words with given classifications

Well Posed Learning Problems

Example : A Robot Driving Learning Problem

Task **T**

Driving on public four-lane highways using vision sensors

Performance Measure **P**

Average distance travelled before an error (as judged by human observer)

Training Experience **E**

A sequence of images and steering commands recorded while observing a human driver

Well Posed Learning Problems

Example : Spam E-mail detection problem

Task T

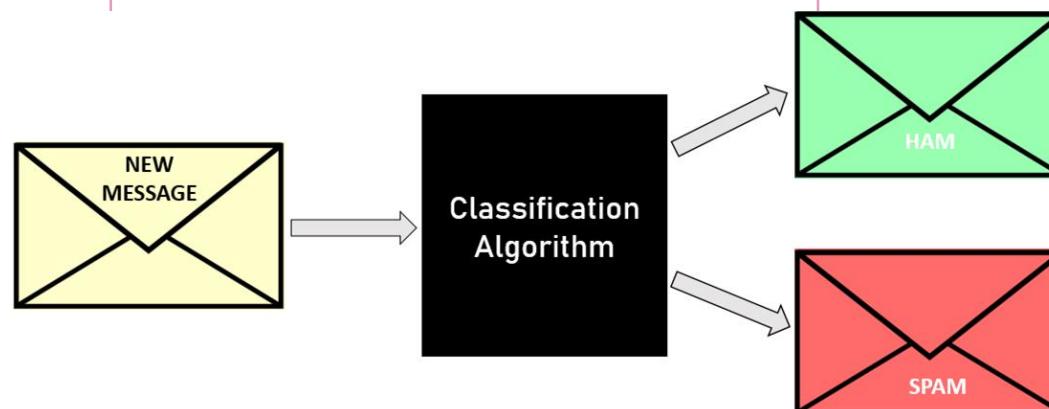
To classify mails into Spam or Not-Spam

Performance Measure P

Total percent of mails being correctly classified as being "Spam" or "Not Spam"

Training Experience E

Set of labelled mails



Designing a Learning System

- ✓ Consider designing a program – **to learn to play checkers**
- ✓ Goal – World checker tournament
- ✓ Adopted Performance measure – The percentage of games it wins in this tournament

Steps for Designing Learning Systems

Choosing the
Training
Experience

Choosing the
Target
Function

Choosing
Representation
of Target
Function

Choosing
Function
approximation

Final Design

Choosing the Training Experience

- ▶ First Design Choice
- ▶ Type of Training Experience can have significant impact on success or failure of the learner
- ▶ Attributes which impact the success or failure
 - ▶ The training experience will be able to provide direct or indirect feedback regarding choices
 - ▶ The degree to which the learner will control the sequence of training examples.
 - ▶ How well it represents the distribution of examples over which the final system performance P must be measured

Well Posed Learning Problems

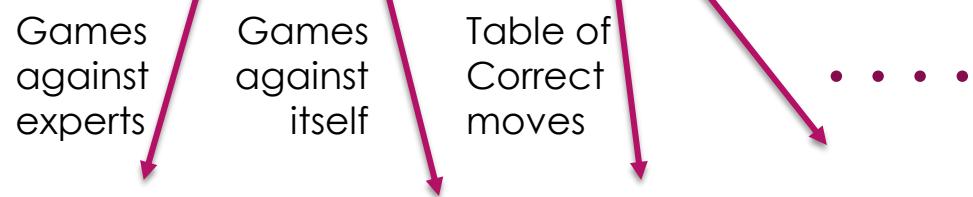
Example : A Checkers Learning Problem

Task T	Performance Measure P	Training Experience E
Playing Checkers	Percent of games won against opponents	Playing Practice Games against itself

In order to complete the design of the learning system, we must now choose

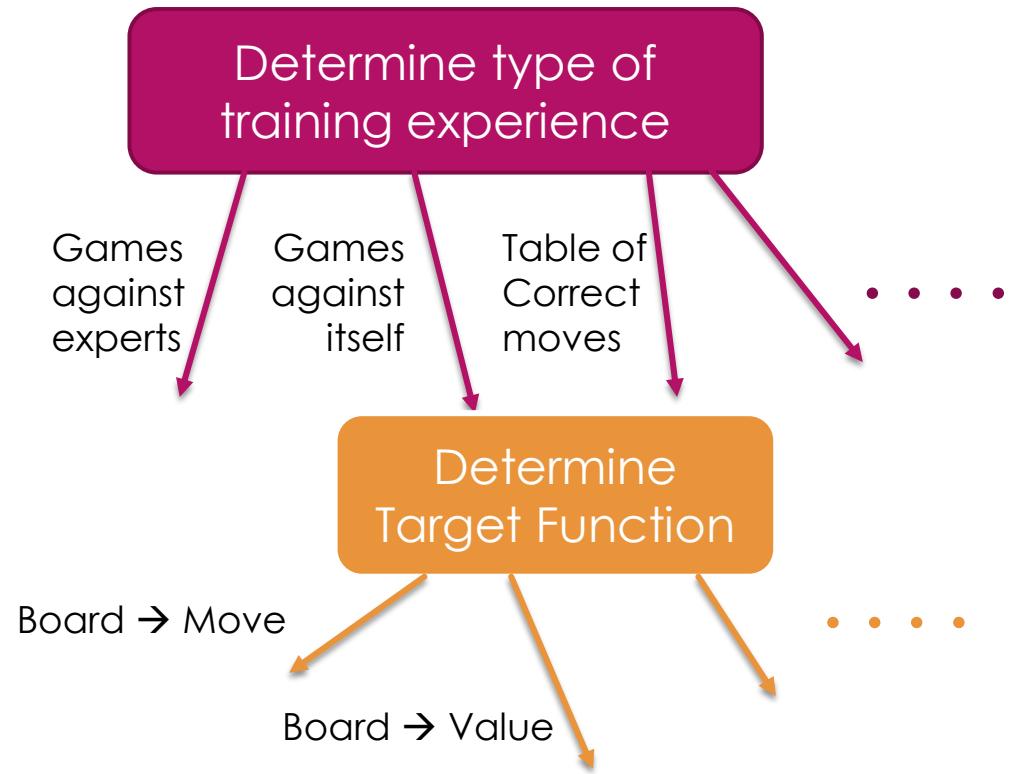
- The exact type of knowledge to be learned
- A representation for this target variable
- A Learning mechanism

Determine type of training experience



Choosing the Target Function

- ▶ It is the **key** design choice – Determining exactly what type of knowledge will be learned and how this will be used by the performance program.
 - ▶ Eg: Generating legal moves from any board state – learning how to choose best from the legal moves.
- ▶ Fn: **ChooseMove : $B \rightarrow M$**
 - ▶ This function accepts input from the set of board states B and produces some move as output from the set of legal moves M
- ▶ Fn: **$V : B \rightarrow \mathbb{R}$**
 - ▶ Assigns a numeric score to any given board state
- ▶ If a system successfully learn a target function V , then it can be easily used to select the best move from any current board position.
- ▶ The process of learning the target function is often called **Function Approximation**



Choosing Representation for the Target Function

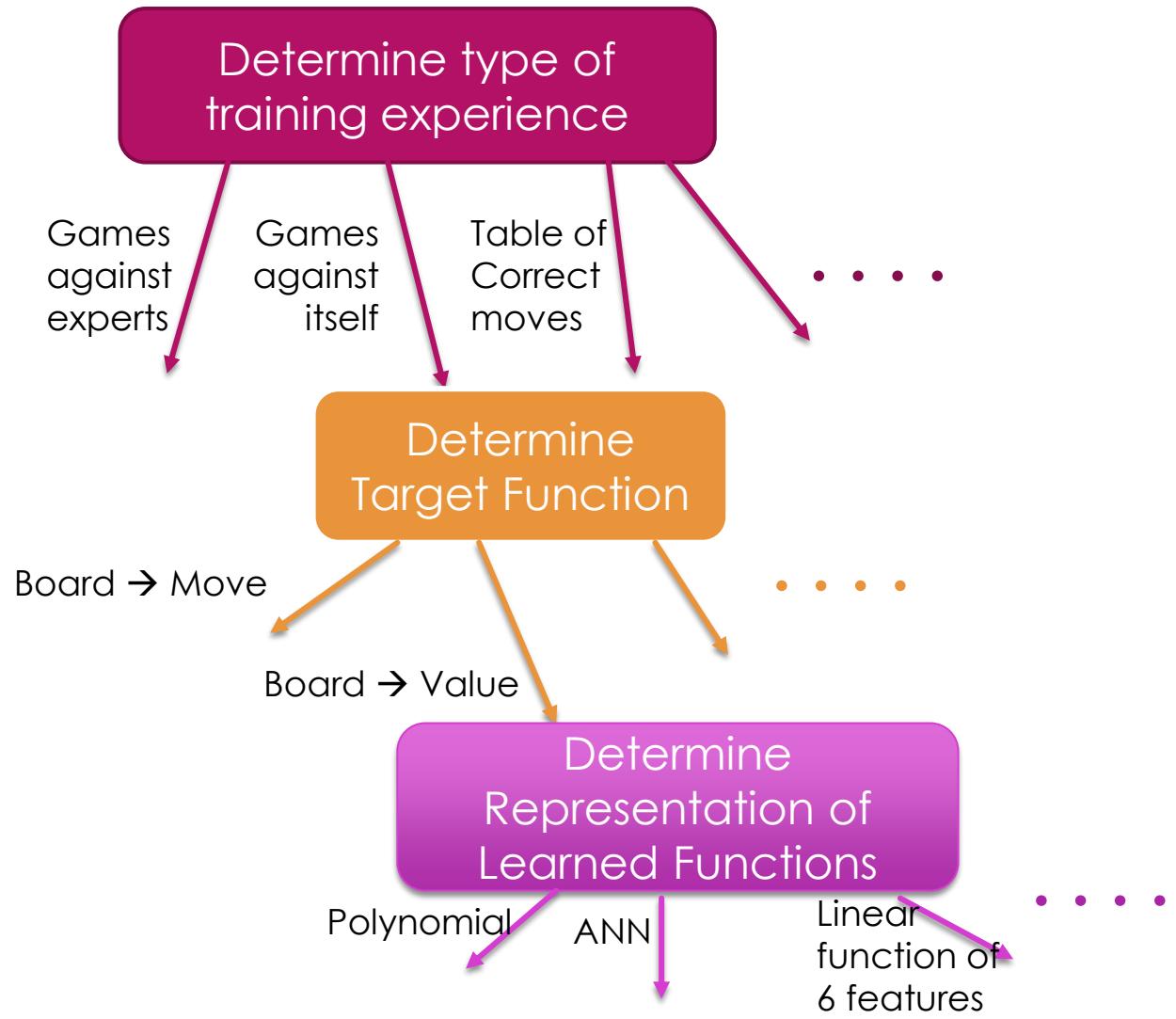
- ▶ After specifying the ideal target function, the next step is choosing a representation function
- ▶ Many options are available to choose the target function \hat{V} . \hat{V} can be represented using
 - ▶ a large table with a distinct entry specifying the value for each board state
 - ▶ Or a collection of rules that match against features of the board state
 - ▶ Or a quadratic polynomial function of predefined board features
 - ▶ Or an Artificial Neural Network
- ▶ A good expressive representation needs more training data.

Choosing Representation for the Target Function – Example Representation

- ▶ \hat{V} calculated as linear combination of
 - ▶ x_1 – Number of **black** pieces on the board
 - ▶ x_2 – Number of **red** pieces on the board
 - ▶ x_3 – Number of **black** kings on the board
 - ▶ x_4 – Number of **red** kings on the board
 - ▶ x_5 – Number of **black** pieces threatened by **red**
 - ▶ x_6 – Number of **red** pieces threatened by **black**
- $$\hat{V}(b) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$$
- ▶ w_1 to w_6
 - ▶ Numerical coefficients
 - ▶ Determine the relative importance of various board features in determining value of the board
- ▶ w_0 – additive constant to the board values

Partial Design of Checkers Learning Program

- ▶ **Task T**
 - ▶ Playing Checkers
- ▶ **Performance measure P**
 - ▶ Percent of games won in the world tournament
- ▶ **Training Experience E**
 - ▶ Games played against itself
- ▶ **Target Function**
 - ▶ $v: B \rightarrow \mathbb{R}$
- ▶ **Target Function Representation**
 - ▶ $\hat{V}(\mathbf{b}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + w_5 x_5 + w_6 x_6$



Choosing a Function Approximation Algorithm

- ▶ A set of training examples are required to learn the target function
- ▶ Each training example describe a specific board state \mathbf{b} and the training value $V_{\text{train}}(\mathbf{b})$ for \mathbf{b}

$$\langle \mathbf{b}, V_{\text{train}}(\mathbf{b}) \rangle$$

- ▶ How to derive these training examples from the indirect training experience available to the learner, adjust the weights w_i to best fit the training examples?
- ▶ **Estimating training values**
 - ▶ Assign $V_{\text{train}}(b) \rightarrow \hat{V}(\text{successor}(b))$ for any intermediate board state b , where
 - ▶ \hat{V} is the learner's current approximation to V and
 - ▶ Successor(b) denotes the next board state following b

Choosing a Function Approximation Algorithm

- ▶ How to specify the learning algorithm for choosing the weights w_i to best fit the set of training examples $\{ \langle b, V_{train}(b) \rangle \}$.
 - ▶ **Adjusting the values**
 - ▶ Best fit means – finding the set of weights that minimizes the squared error between the training values and the values picked by \hat{V}
- $$E \equiv \sum_{\langle b, V_{train}(b) \rangle \in \text{training examples}} [V_{train}(b) - \hat{V}(b)]^2$$
- ▶ We seek weights that minimize E for the observed training examples.
 - ▶ An algorithm is required that will incrementally refine the weights as new training examples become available. **Least Mean Squares** or **LMS** training rule is one such algorithm.
 - ▶ The LMS rule adjusts the weights a small amount in the direction that reduces the error for each observed training example.

Choosing a Function Approximation Algorithm

- ▶ Adjusting the values contd..

LMS weight update rule.

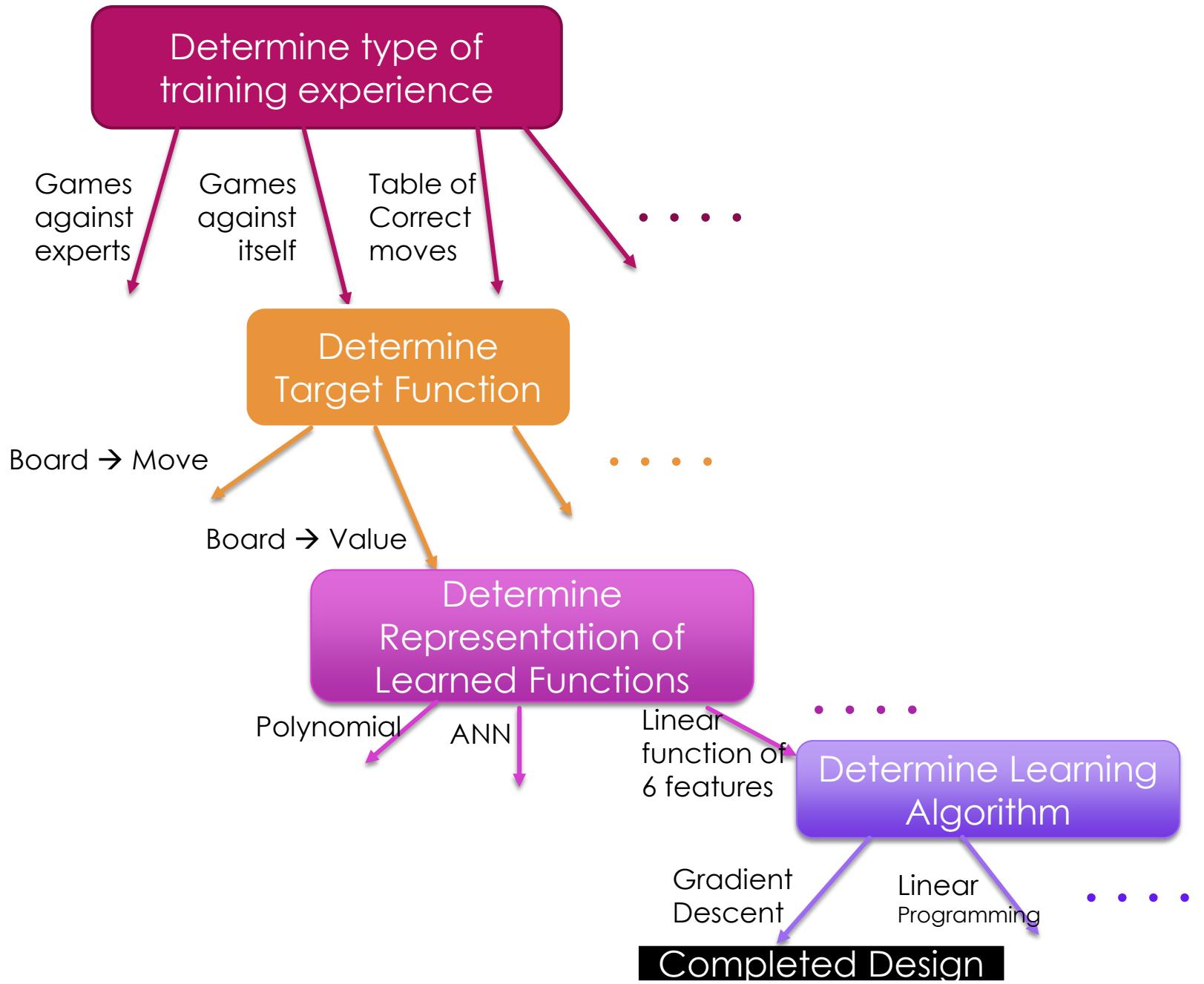
For each training example $\langle b, V_{train}(b) \rangle$

- Use the current weights to calculate $\hat{V}(b)$
- For each weight w_i , update it as

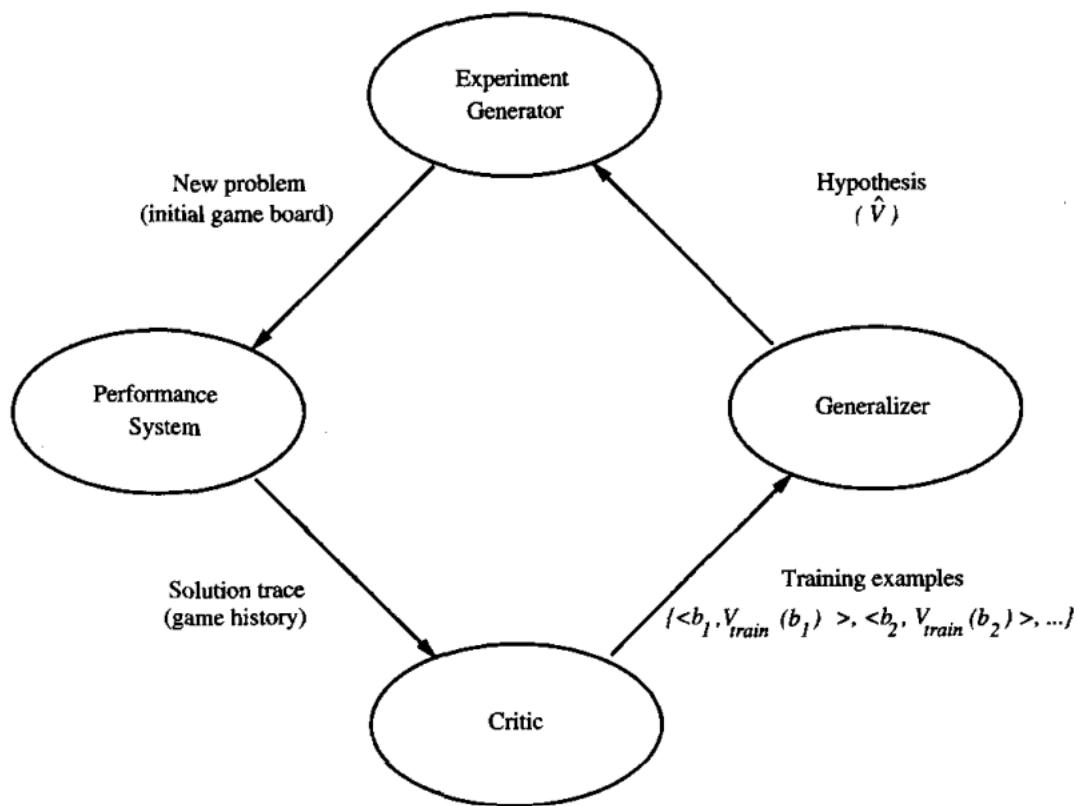
$$w_i \leftarrow w_i + \eta (V_{train}(b) - \hat{V}(b)) x_i$$

Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

- ▶ When $[V_{train}(b) - \hat{V}(b)] = 0$ no weights are changed
- ▶ When $[V_{train}(b) - \hat{V}(b)]$ is positive, then each weight is increased in proportion to the value of its corresponding feature



Final Design



Issues in ML

1

What algorithms exist for learning general target functions from specific training examples?

In what settings will particular algorithms converge to the desired function, given sufficient training data?

Which algorithm perform best for which types of problems and representations?

2

How much training data is sufficient?

What general bounds can be found to relate the confidence learned to the amount of training experience?

3

When and how can prior knowledge held by the learner guide the process of generalizing from examples?

Can prior knowledge be helpful even when it is only approximately correct?

Issues in ML

4

What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?

5

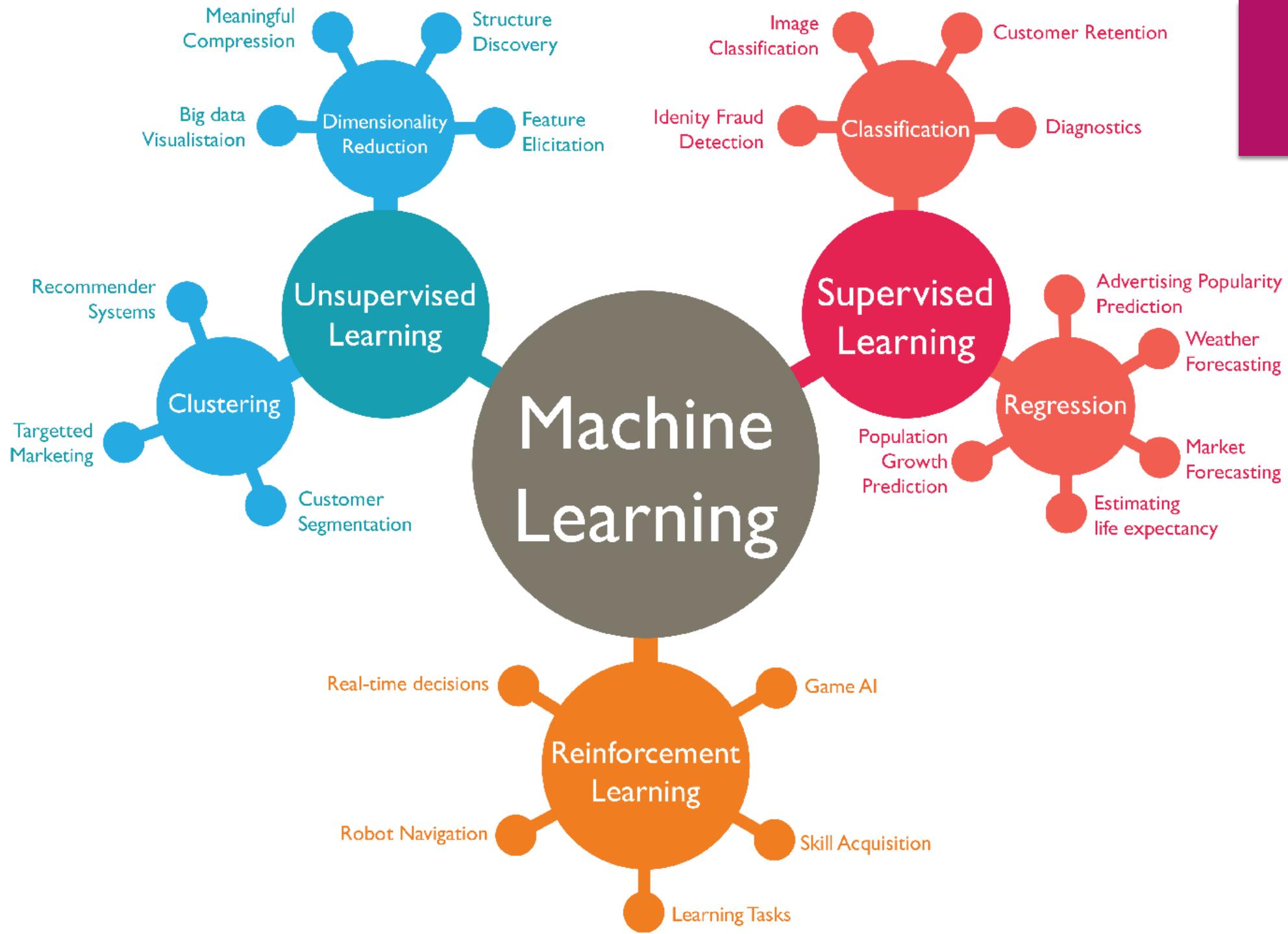
What is the best way to reduce the learning task to one or more function approximation problems?

What specific functions should the system attempt to learn?

Can this process itself be automated?

6

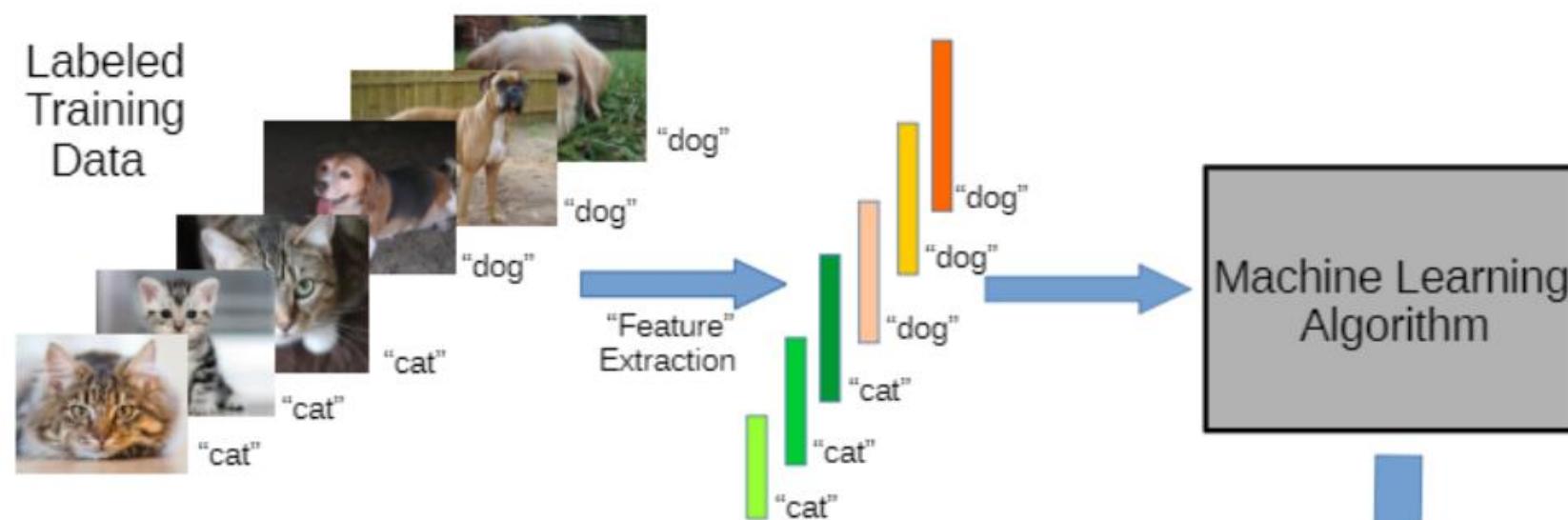
How can the learner automatically alter its representation to improve its ability to represent and learn the target function?



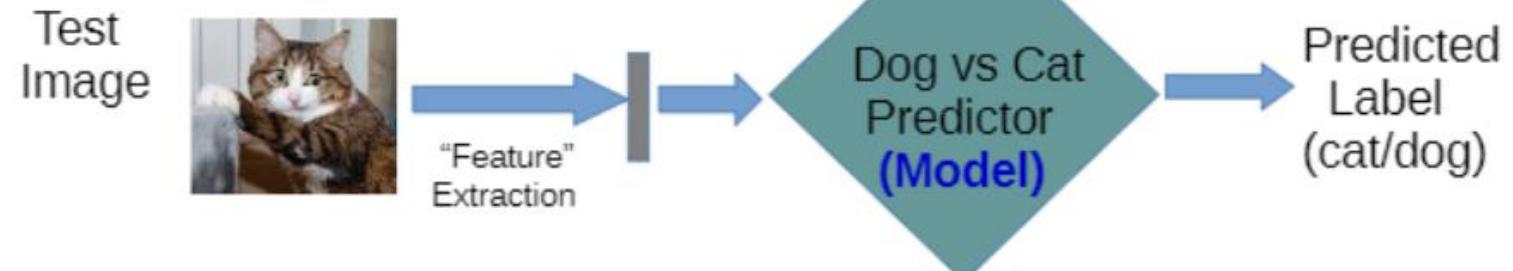
Types of Machine Learning

- ▶ Supervised Learning
- ▶ Unsupervised Learning
- ▶ Reinforcement Learning

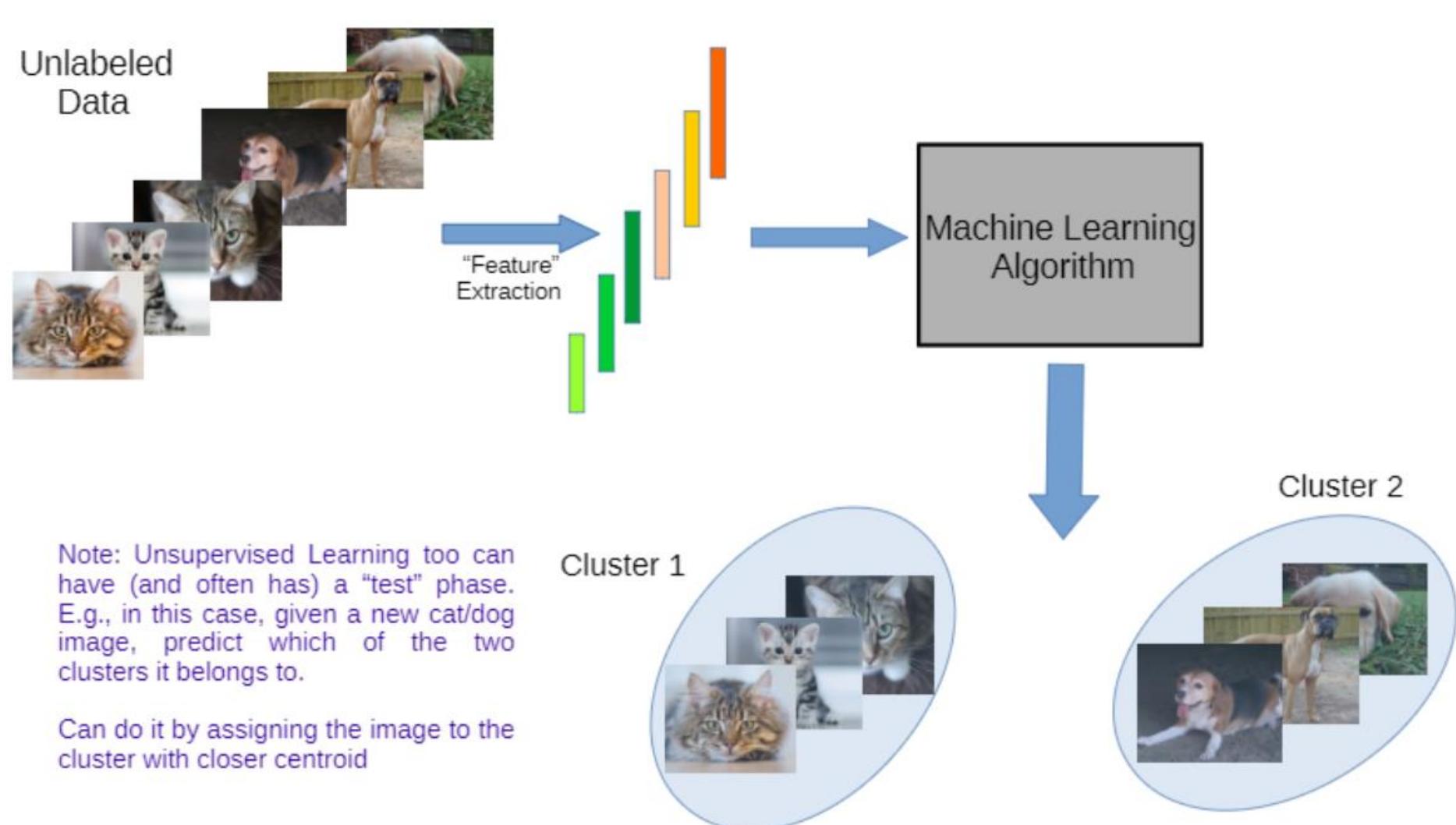
A Typical Supervised Learning Workflow (for classification)

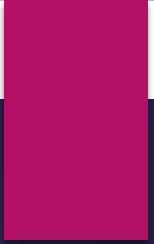
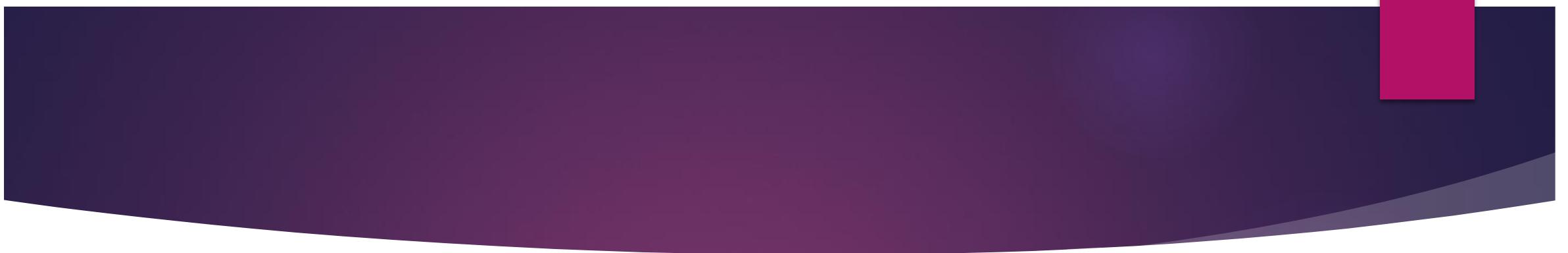


Note: The **feature extraction** phase may be part of the machine learning algorithm itself
(referred to as "feature learning" or "representation learning")
Modern "**deep learning**" algos do precisely that!



A Typical Unsupervised Learning Workflow (for Clustering)





Introduction

Motivation, Applications of Machine Learning - Well-Posed Learning Problems - Designing a Learning System - Issues in Machine Learning - Types of Machine Learning

Supervised Learning - Regression Techniques

Basic concepts and applications of Regression - Simple Linear & Multiple Regression - Gradient Descent - Evaluation Measures for Regression Techniques - overfitting - underfitting - Regularization - Train-test-split, k-fold cross validation - Hyperparameter tuning.

Refer Lecture slides for Introduction (part 1)

Supervised Learning – Regression Techniques

Basic Concepts and applications of Regression

- Regression is a predictive modelling technique where the target variable to be estimated is continuous. Following are few of the regression tasks.
 - Predicting stock market index using other economic indicators.
 - Forecasting the amount of prediction in a region based on the characteristics of the jet stream.
 - Projecting the total sales of a company based on the amount spend for advertising.
- Let D denote the dataset that has m observations.
$$D = \{(x_i, y_i) \mid i = 1, 2, \dots, m\}$$

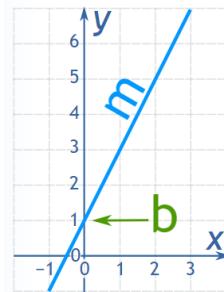
$x_i \rightarrow$ corresponds to the set of attributes of the i^{th} observation. It can be either continuous or discrete

$y_i \rightarrow$ corresponds to the target variable

- **Def :** Regression is the task of learning a target function f that maps each attribute set \mathbf{X} into the continuous valued space \mathbf{y} .
- Based on the number of input features in the training example, the regression models are classified into
 - Simple Linear Regression (Univariate)
 - Multiple Linear Regression (Multivariate)
- Simple regression model: This is the most basic regression model in which predictions are formed from a single, univariate feature of the data.
- Multiple regression model: As name implies, in this regression model the predictions are formed from multiple features of the data.
- The goal of regression is to find a target function that can fit the input data with minimum error.
- To find the target function in regression, a best fit straight line is fit on the training points.
- The equation of the straight line is given below.

$$y = mx + b$$

Slope or Gradient y value when $x=0$
 (see [Y Intercept](#))



- **m** value is the **slope** or **gradient** which tells how steep the line is. Slope is given by the unit change in y axis for the unit change in x-axis.
- **b** is value of y when x=0 and it is referred as **intercept**.
- One can draw multiple lines in 2D space. Best fit line is one which gives the minimal error.
- For which values of m and c, you are getting minimum error are called as **optimal parameters**.
- The error function for a regression task can be expressed in terms of the sum of absolute or squared errors.

$$\text{Absolute Error} = \sum_i |y_i - f(x_i)|$$

$$\text{Squared Error} = \sum_i (y_i - f(x_i))^2$$

The **applications** of ML regression algorithms are as follows:

- **Forecasting or Predictive analysis:** One of the important uses of regression is forecasting or predictive analysis. For example, we can forecast GDP, oil prices or in simple words the quantitative data that changes with the passage of time.
- **Optimization:** We can optimize business processes with the help of regression. For example, a store manager can create a statistical model to understand the peak time of coming of customers.
- **Error correction:** In business, taking correct decision is equally important as optimizing the business process. Regression can help us to take correct decision as well in correcting the already implemented decision.
- **Economics:** It is the most used tool in economics. We can use regression to predict supply, demand, consumption, inventory investment etc.
- **Finance:** A financial company is always interested in minimizing the risk portfolio and want to know the factors that affects the customers. All these can be predicted with the help of regression model.

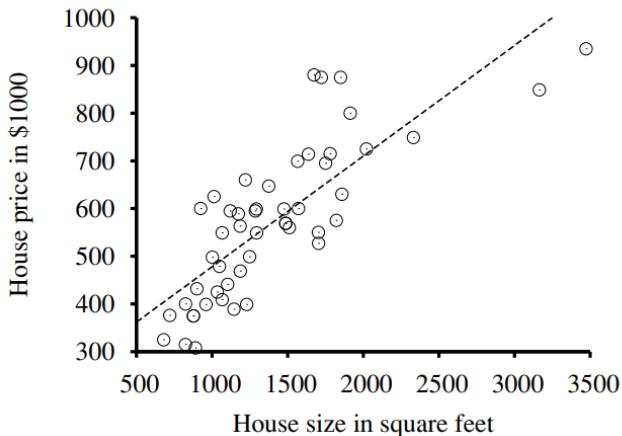
Univariate Linear Regression (Simple Linear Regression)

- A univariate linear regression (a straight line) with input x and output y has the form
- $$y = w_0 + w_1 x$$
- Where w_0 and w_1 are real-valued coefficients to be learned. The letter w is used as they are referred as **weights**.

Unit – 1 Fundamentals of Machine Learning

- The value of y is changed by changing the relative weight of one term or another.
- Let w be vector $[w_0, w_1]$ and define the target function

$$\hat{y} = f(x) = h_w(x) = w_0 + w_1 x$$

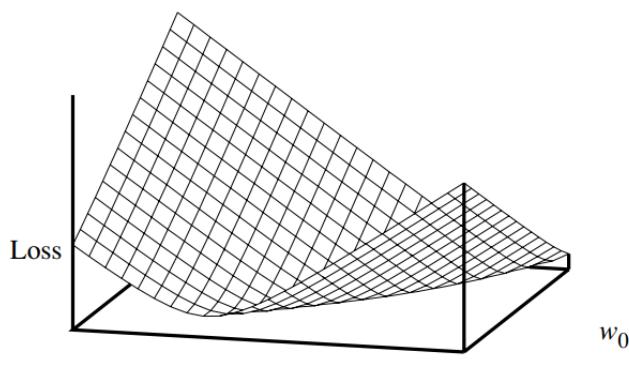


The above figure shows the data points of price verses floor space of houses along with linear function that minimizes squared error loss : $y=0.232x+246$

- The task of finding the h_w that best fits these data is called **linear regression**
- To fit a line to the data, we have to find the values of weights $[w_0, w_1]$ that minimizes the empirical loss.
- To find the loss, squared loss function (L_2) summed over all the training examples is used.

$$\begin{aligned} Loss(h_w) &= \sum_{j=1}^N L_2(y_j, h_w(x_j)) \\ &= \sum_{j=1}^N (y_j - h_w(x_j))^2 \\ &= \sum_{j=1}^N (y_j - (w_0 + w_1 x_j))^2 \end{aligned}$$

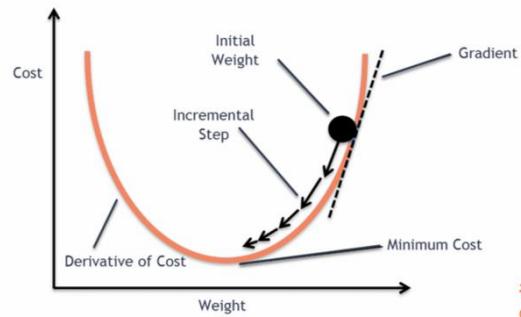
- In linear regression the goal is to find $w^* = \underset{w}{\operatorname{argmin}} \ Loss(h_w)$
- The following figure shows the weight space. The **weight space** is the space defined by all possible settings of the weights. The loss function of w_0 and w_1 is plotted here as a 3D plot.



- The loss function is convex for every linear regression problem with an L_2 loss function

Gradient Descent

- The gradient descent loss is used to minimize the loss.
- Gradient Descent is a first-order optimization algorithm. It involves taking steps in the opposite direction of the gradient in order to find the global minimum (or local minimum in non-convex functions) of the objective function. The image below provides a great illustration of how Gradient Descent takes steps towards the global minimum of a convex function.
- Gradient Descent starts with a point (w_0, w_1) in the weight space, and then moves to the neighbouring point that is downhill. This is repeated until we converge on minimum possible loss.



$w \leftarrow$ any point in the parameter space

loop until convergence do

for each w_i in w do

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$$

- α is called the **learning rate**. It can be a fixed constant or it can decay over time as the learning process proceeds.
- The loss function of univariate regression is a quadratic function. Therefore the partial derivative are a linear function.
- Consider a simple training example (\mathbf{x}, \mathbf{y}) and find out the slopes.

$$\begin{aligned} \frac{\partial}{\partial w_i} Loss(w) &= \frac{\partial}{\partial w_i} (y - \hat{y})^2 \\ &= 2(y - \hat{y}) \frac{\partial}{\partial w_i} (y - \hat{y}) \\ &= 2(y - \hat{y}) \frac{\partial}{\partial w_i} (y - (w_0 + w_1 x)) \end{aligned}$$

- Applying this to both w_0 and w_1 , we get

$$\begin{aligned} \frac{\partial}{\partial w_0} Loss(w) &= -2(y - \hat{y}) \\ \frac{\partial}{\partial w_1} Loss(w) &= -2(y - \hat{y}) x \end{aligned}$$

- When you use these values in the weight updation function, they will become

$$\begin{aligned} w_0 &\leftarrow w_0 + \alpha (y - \hat{y}) \\ w_1 &\leftarrow w_1 + \alpha (y - \hat{y}) x \end{aligned}$$

- If $\hat{y} > y$,

- Reduce w_0 a bit
- Reduce w_1 if x is positive input or increase w_1 if x is a negative input.

Unit – 1 Fundamentals of Machine Learning

- The above weight updation equations are for one training example. For N training examples, we want to minimize the sum of the individual losses for each example.
- The derivative of sum is the sum of the derivatives.

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - \hat{y}_j)$$

$$w_1 \leftarrow w_1 + \alpha \sum_j (y_j - \hat{y}_j) x_j$$

- Different variants of Gradient Descent are

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini-Batch Gradient Descent
<ul style="list-style-type: none"> Entire dataset for updation Cost function reduces smoothly Computation cost is very high 	<ul style="list-style-type: none"> Single observation for updation Lot of variations in cost function Computation time is more 	<ul style="list-style-type: none"> Subset of data for updation Smoother cost function as compared to SGD Computation time is lesser than SGD Computation cost is lesser than Batch Gradient Descent

Multivariate Linear Regression

- In multivariate linear regression problems, every example x_j is an n-element vector, and a target y .
- The hypothesis space for Multiple Linear Regression is the set of functions of the form

$$\begin{aligned} h_{sw}(x_j) &= w_0 + w_1 x_{j1} + w_2 x_{j2} + \dots + w_n x_{jn} \\ &= w_0 + \sum_i w_i x_{ji} \end{aligned}$$

- w_0 is the intercept term and w_1 to w_n are regression coefficients for the respective inputs.
- By taking a dummy variable $x_{j0} = 1$, we can rewrite the above equation as

$$\begin{aligned} \hat{y}_j &= h_{sw}(x_j) = \sum_i w_i x_{ji} \\ &= w \cdot x_j \\ &= w^T x_j \end{aligned}$$

- The best vector of weights w^* is the one which minimizes the squared error loss over the examples.

$$\begin{aligned} w^* &= \operatorname{argmin}_w \sum_j L_2(y_j, \hat{y}_j) \\ &= \operatorname{argmin}_w \sum_j L_2(y_j, w \cdot x_j) \end{aligned}$$

- The gradient descent will reach the minimum of the loss function. The update equation for each weight w_i is

$$w_i \leftarrow w_i + \alpha \sum_j x_{ji} (y_j - \hat{y}_j)$$

Evaluation measures for regression techniques

- Evaluation metrics are used **to measure the quality of the statistical or machine learning model.**
- Evaluating machine learning models or algorithms is essential for any project.
- It is necessary to obtain good accuracy on the training data. But it is also important to get a genuine and approximate result on unseen data otherwise the trained model is of no use.
- It is required to evaluate the model on different metrics to build a generalized model. It also helps to better optimize the performance, fine-tune it and obtain a better result.
- As a single evaluation metric is not suitable for all kinds of datasets, we have different types of evaluation metrics.
- There are many different types of evaluation metrics available to test a regression model.

1. Mean Absolute Error (MAE)

- MAE is a very simple metric. It is calculated by taking the absolute difference between the actual and the predicted.

$$MAE = \frac{1}{N} \sum_i |y_i - \hat{y}_i|$$

```
from sklearn.metrics import mean_absolute_error

y = [1, 2, 3, 4]
ypred = [1, 1, 2, 2]

mae = mean_absolute_error(y, ypred)
print(mae)
```

1.0

- Advantages :
 - The MAE value is of same unit as the output variable.
 - It is most robust to outliers.
- Disadvantages:
 - The graph of MAE is not differentiable, so we have to apply various optimizers like Gradient Descent which can be differentiable.

2. Mean Squared Error (MSE)

- It is the most widely used and very simple metric.
- It is the squared difference between the actual and the predicted value.

$$MSE = \frac{1}{N} \sum_i (y_i - \hat{y}_i)^2$$

```
from sklearn.metrics import mean_squared_error

y = [1, 2, 3, 4]
ypred = [1, 1, 2, 2]

mse = mean_squared_error(y, ypred)
print(mse)
```

1.5

- Advantages:
 - MSE is differentiable and you can use it as a loss function.
- Disadvantages :
 - It is not robust to outliers. MSE penalizes the outliers most and the calculated MSE is bigger.
 - The MSE unit is not the same as the target. For eg., if target variable is in meters, then MSE is meter squared.

3. Root Mean Squared Error (RMSE)

- It is the square root of the mean squared error.

$$RMSE = \sqrt{\frac{1}{N} \sum_i (y_i - \hat{y}_i)^2}$$

```
from sklearn.metrics import mean_squared_error
import numpy as np

y = [1, 2, 3, 4]
ypred = [1, 1, 2, 2]

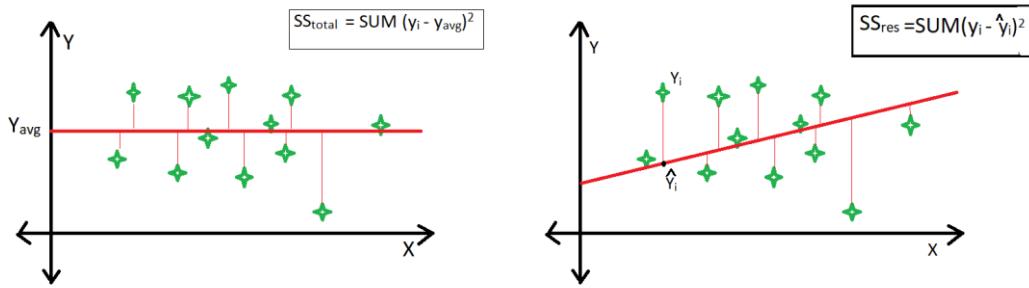
rmse = np.sqrt(mean_squared_error(y, ypred))
print(rmse)
```

1.224744871391589

- Advantages:
 - The RMSE value and the target is of same unit. It makes interpretation of loss easy.
- Disadvantage:
 - It is not that robust to outliers when compared to MAE.

4. R Squared (R2 Score)

- R-squared is a statistical measure that represents the goodness of fit of a regression model. The ideal value for r-square is 1. The closer the value of r-square to 1, the better is the model fitted.
- R-square is a comparison of residual sum of squares (SSres) with total sum of squares(SStot).
- Total sum of squares is calculated by summation of squares of perpendicular distance between data points and the average line.
- Residual sum of squares is calculated by the summation of squares of perpendicular distance between data points and the best fitted line.



- R square is calculated by using the following formula :

$$R^2 = 1 - \frac{SS_{res}}{SS_{total}}$$

```
from sklearn.metrics import r2_score

y = [1, 2, 3, 4]
ypred = [1, 1, 2, 3]

r2score = r2_score(y, ypred)
print(r2score)

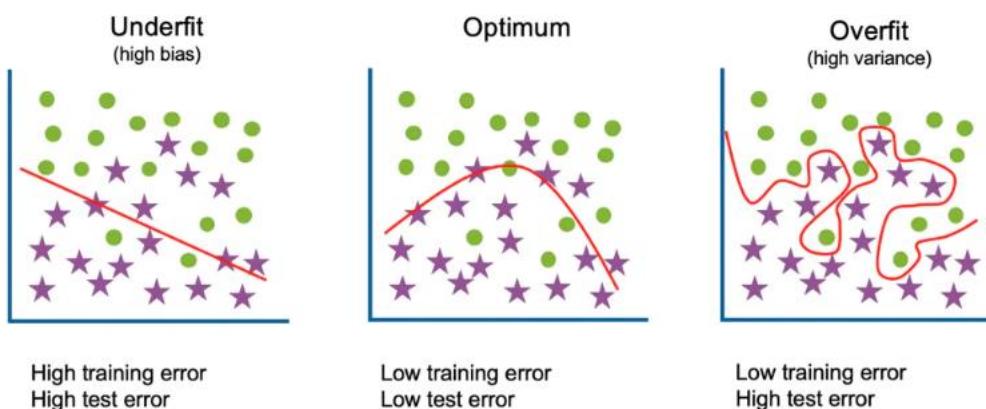
print(r2_score([1,2,3], [1,2,3]))
```

0.4

1.0

Overfitting and underfitting

- Generalization** is the model's ability to perform well on unseen data.
- Overfitting is a phenomenon that occurs when a machine learning model is tailored to a particular dataset and is unable to generalize to other datasets.
- Overfitting occurs when a model fits exactly against its training data and is unable to capture the relationship between the input and the output variables accurately. When a model is overfit, the algorithm cannot perform accurately against unseen data.



- There is no problem of overfitting with univariate linear regression.
- But with Multivariate Linear Regression in high dimensional spaces, it is possible that some dimension that is actually irrelevant appears by chance, to be useful and results in overfitting.

Regularization

- Regularization is a technique commonly used in multivariate linear functions to avoid overfitting.
- With regularization, the total cost of the hypothesis is minimized by counting both empirical loss and the complexity of the hypothesis.

$$Cost(h) = Empirical\ Loss(h) + \lambda Complexity(h)$$

- For a linear function, the complexity can be specified as a function of weights.

$$Complexity(h) = L_q(w) = \sum_i |w_i|^q$$

- Regularization basically adds the penalty as model complexity increases. Regularization term (lambda) penalizes all the parameters except intercept so that model generalizes the data and won't overfit.
- penalizing the parameters fulfills two main objectives.
 - i. The order of the curve that is to be fitted got simpler
 - ii. Overfitting reduced.
- The new cost function controls the trade-off between two goals.
- 1. Fitting the training set well, which is ensured by the first term (Empirical loss) of the cost function.
- 2. Keeping the parameters small, which is ensured by regularization term. lambda is the regularization parameter.

L1 regularization:

L1 regularization, also known as L1 norm or **Lasso** combats overfitting by shrinking the parameters towards 0. This makes some features obsolete.

The q value is 1 in L1 regularization

L1 regularization minimizes the sum of absolute values and penalizes the absolute value of the weights.

$$Cost(h) = \sum_{j=1}^N (y_j - h_w(x_j))^2 + \sum_i |w_i|$$

It is a form of feature selection. Lasso shrinks the coefficient of irrelevant or less important features to zero and eventually helps in feature selection.

L1 regularization return a sparse solution.

L2 Regularization:

- L2 regularization, or the L2 norm, or Ridge combats overfitting by forcing weights to be small, but not making them exactly zero.
- When L2 regularization is used, the less significant features also have some influence over the final prediction.

$$Cost(h) = \sum_{j=1}^N (y_j - h_w(x_j))^2 + \sum_i w_i^2$$

- L2 regularization return a non-sparse solution since the weights will be non-zero.

The differences between L1 and L2 regularization:

- L1 regularization penalizes the sum of absolute values of the weights, whereas L2 regularization penalizes the sum of squares of the weights.
- The L1 regularization solution is sparse. The L2 regularization solution is non-sparse.

Unit – 1 Fundamentals of Machine Learning

- L2 regularization doesn't perform feature selection, since weights are only reduced to values near 0 instead of 0. L1 regularization has built-in feature selection.
- L1 regularization is robust to outliers, L2 regularization is not.

Cross Validation

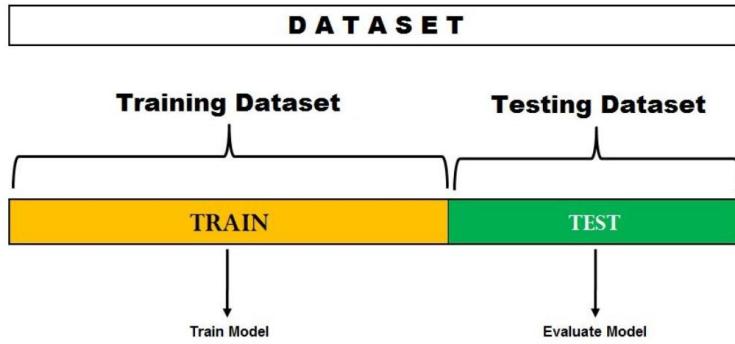
- When any Machine Learning model is built, it is trained with initial data. And then we feed some unknown data to understand how well the model performs and generalizes over unseen data.
- If the model performs well on the unseen data, it's consistent, and is able to predict with good accuracy on a wide range of input data; then this model is stable.
- But this is not the case always! Machine learning models are not always stable and we have to evaluate the stability of the machine learning model. That is where Cross Validation comes into the picture.
- Cross-Validation is a technique used to assess how well the Machine learning models perform on unseen data.
- **Def :** Cross-Validation is the process of assessing how the results of a statistical analysis will generalize to an independent data set.
- There are many ways to perform Cross-Validation. They are
 - Hold Out method
 - Leave One Out Cross-Validation
 - K-Fold Cross Validation
 - Startified K-Fold Cross Validation

Need of cross-validation :

- Suppose you build a machine learning model to solve a problem, and you have trained the model on a given dataset. When you check the accuracy of the model on the training data, it is close to 95%. Does this mean that your model has trained very well, and it is the best model because of the high accuracy?
- No, it's not! Because your model is trained on the given data, it knows the data well, captured even the minute variations(noise), and has generalized very well over the given data. If you expose the model to completely new, unseen data, it might not predict with the same accuracy and it might fail to generalize over the new data. This problem is called over-fitting.
- Sometimes the model doesn't train well on the training set as it's not able to find patterns. In this case, it wouldn't perform well on the test set as well. This problem is called Under-fitting.

Hold-Out Method

- This is the simplest and most widely used evaluation method in Machine Learning.
- In this method, the entire dataset is divided into 2 sets.
 - The Train set and
 - The Test set
- The data can be divided into 70-30, 80-20 or 50-50 depending on the use case.
- The proportion of training data has to be larger than the test data.



- The drawbacks of hold-out method are
 - In the Hold out method, the test error rates are highly variable and it totally depends on which training examples end up in the training and test data sets.
 - Only a part of the data is used to train the model. This leads problems when the data is not huge.
- Advantages:
 - Computationally inexpensive when compared to other cross-validation techniques.

The following example divides a toy dataset with 10 samples in training and test sets with 70-30 percent split.

```

from sklearn.model_selection import train_test_split

X = [10,20,30,40,50,60,70,80,90,100]
y = [0, 0, 1, 1, 0, 0, 1, 1, 0, 1]

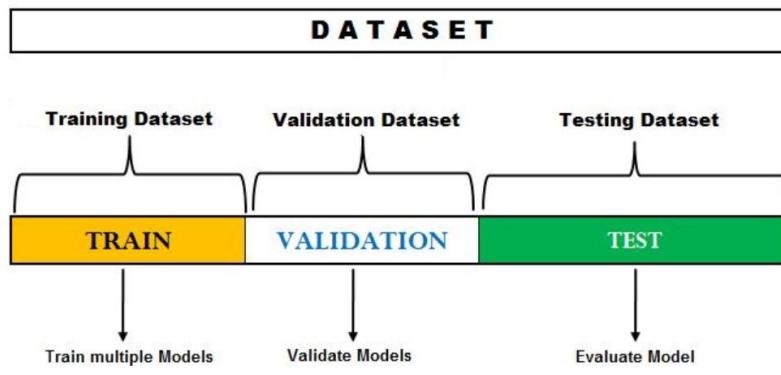
X_train, X_test, y_train, y_test= train_test_split(X,y,test_size=0.3, random_state=1)
print(f'Train-X = {X_train} \nTrain-y = {y_train}')
print(f'Test-X = {X_test} \nTest-y = {y_test}')

Train-X = [50, 10, 40, 20, 80, 90, 60]
Train-y = [0, 0, 1, 0, 1, 0, 0]
Test-X = [30, 100, 70]
Test-y = [1, 1, 1]
  
```

In the above code, `random_state` is the seed for reproducibility.

`train_test_split()` method in the above code snippet split arrays or matrices into random train and test subsets. `test_size` represents the proportion of the dataset to include in the test split, and `random_state` is the seed used by the random number generator.

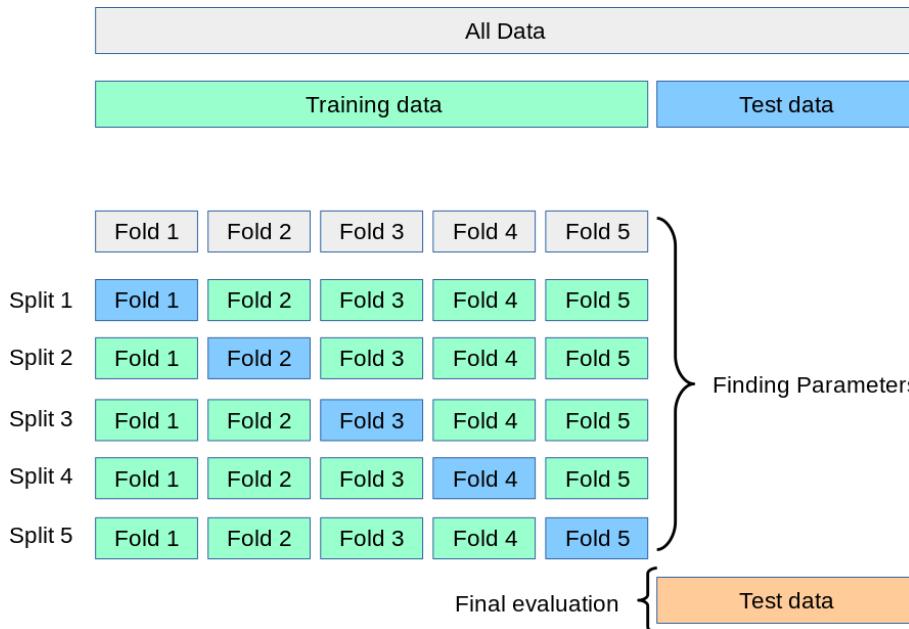
- The limitation of the above method is that the error found in the test dataset can highly depend on the observations included in the train data. This method is not effective to tune the hyperparameters.
- To come out of the above limitations, we have another popular form of hold-out method which includes the splitting of data into 3 separate sets.
 - Training Data
 - Validation Data
 - Test Data



- The validation set, which is a hold-out set from the training set is used to optimize the hyper-parameters of the model to evaluate the model.

K-Fold Cross Validation

- In K-Fold Cross Validation technique the training data is divided into K sets of almost equal sizes.
- The first set is selected as validation set and the model is trained on the remaining ($K-1$) sets and then error is calculated.
- In the second iteration, the 2nd set selected as validation set and the remaining ($K-1$) sets are used to train the model and then error is calculated.
- This process continues for all the K sets.



- The following code snippet divides a small dataset into 5-Folds.

```

from sklearn.model_selection import KFold

X = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

kf = KFold(n_splits=5, shuffle=False, random_state=None)

for train, test in kf.split(X):
    print(f'Train : {train} -- Test : {test}')

```

Train : [2 3 4 5 6 7 8 9] -- Test : [0 1]
 Train : [0 1 4 5 6 7 8 9] -- Test : [2 3]
 Train : [0 1 2 3 6 7 8 9] -- Test : [4 5]
 Train : [0 1 2 3 4 5 8 9] -- Test : [6 7]
 Train : [0 1 2 3 4 5 6 7] -- Test : [8 9]

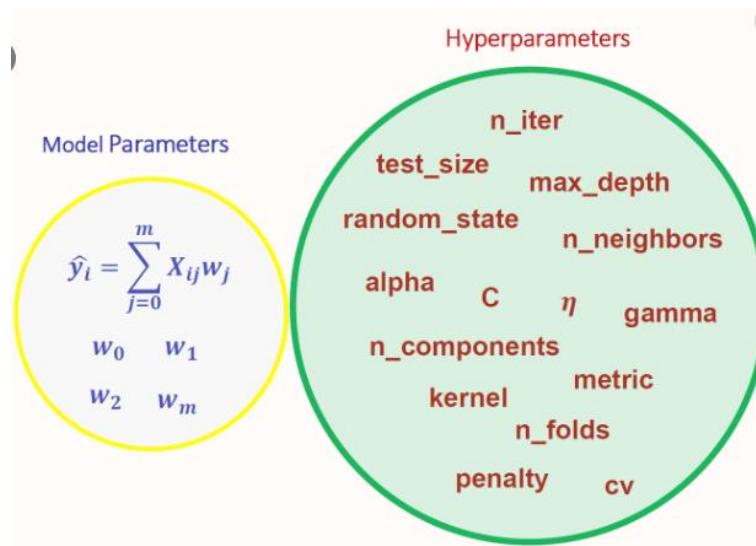
- The mean of errors from all the iterations is calculated as the CV error estimate

$$CV_{(k)} = \frac{1}{k} \sum_{i=1}^k MSE_i$$

- In K-Fold CV, the number of folds K is less than the number of observations in the data (K<N)
- The best part about this method is each data point gets to be in validation set exactly once and gets to be part of the training set (K-1) times.
- The major disadvantage of this method is that the model has to be run from scratch K-times and is computationally expensive than the Hold Out method.

Hyper-parameter Tuning

- Defining model complexity is one of the key in ML algorithm performance. If the model is too complex, it will overfit i.e performs well on train data but poor on unseen data/test data (production like data) whereas if the complexity is too low the model underfit and won't capture all the information in the data.
- So, we need to define Parameters and Hyperparameters. The model parameters tell how to transform input data into desired output whereas, the hyperparameters are used to determine the structure of the model in use. Hyperparameters must be determined before training starts.



- Hyperparameter tuning is choosing a set of optimal hyperparameters for a learning algorithm.
- A hyperparameter is a model argument whose value is set before the learning process begins.
- Hyper parameter tuning is the key to improve the performance of machine learning algorithms.
- Examples of hyperparameters
 - K in KNN
 - Regularization constant, kernel type
 - Number of layers,
 - number of units per layer.
- The popular techniques to tune hyperparameters are Grid Search Parameter Tuning, and Random Search Parameter Training.

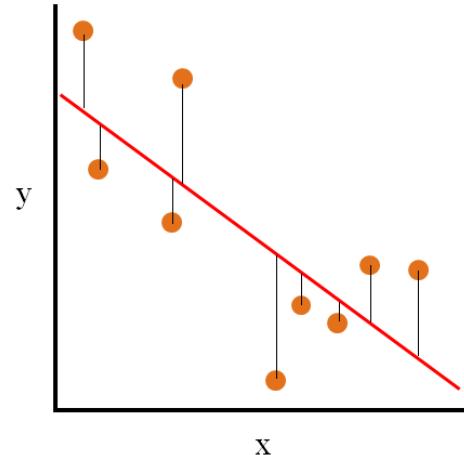
Grid Search is a basic optimal hyperparameter tuning technique. It builds a model for each permutation of all of the given hyperparameter values. For every combination, cross validation is used and average score is calculated.

While tuning hyper parameters, the data should have been split into three parts — Training, validation and testing.

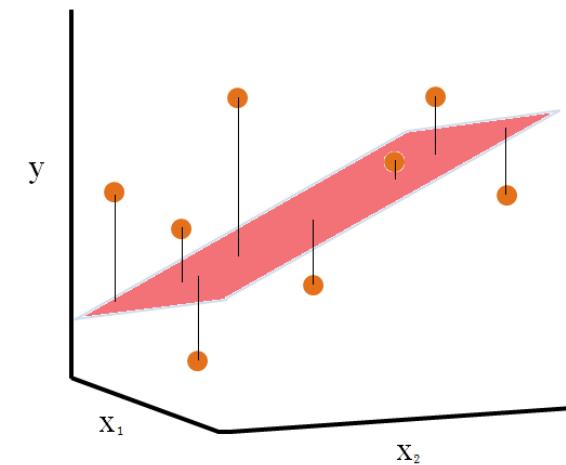


REGRESSION

Simple Linear Regression

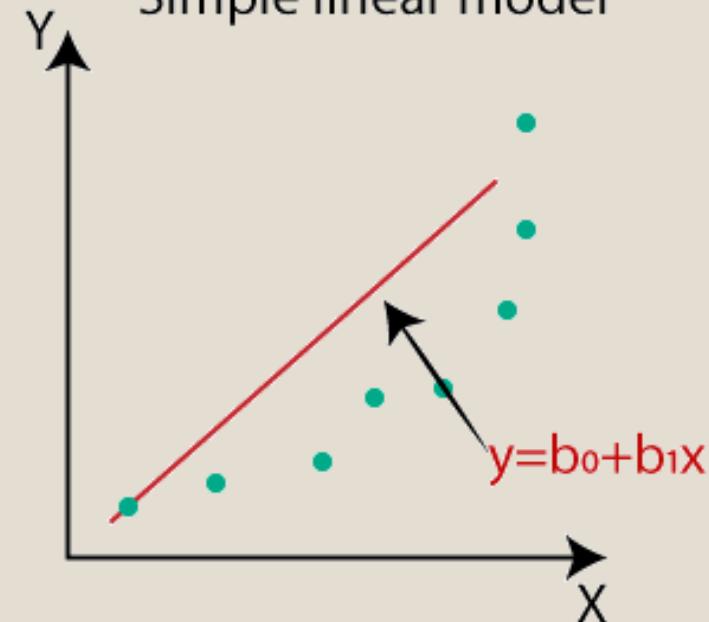


Multiple Linear Regression (2 Independent Variables (x_1, x_2))

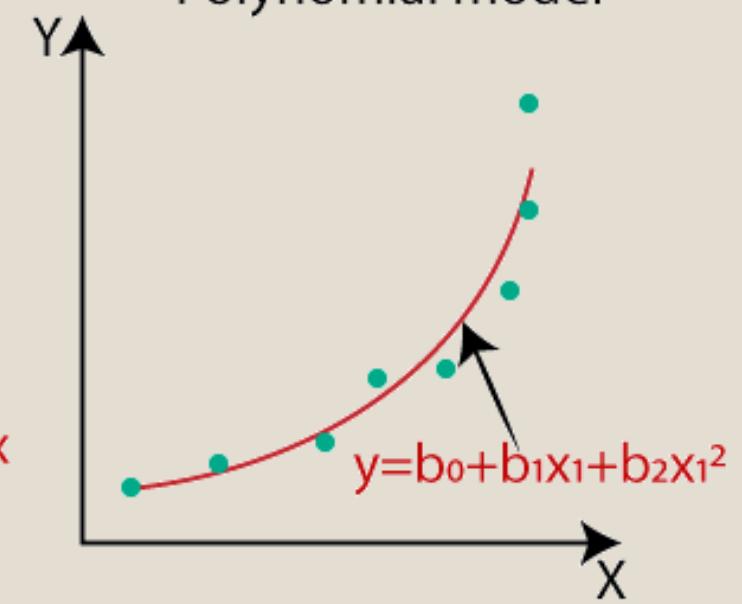


How the curves of various regression models look like?

Simple linear model

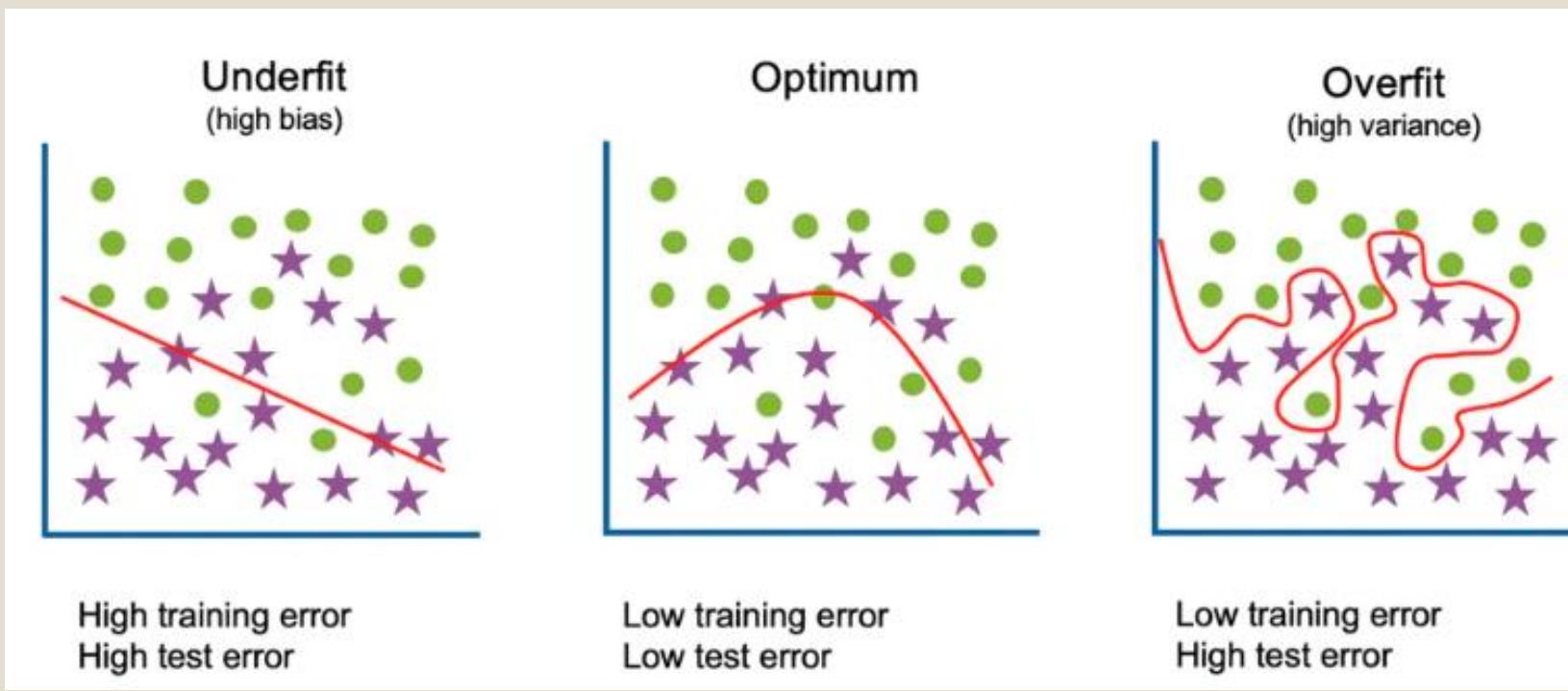
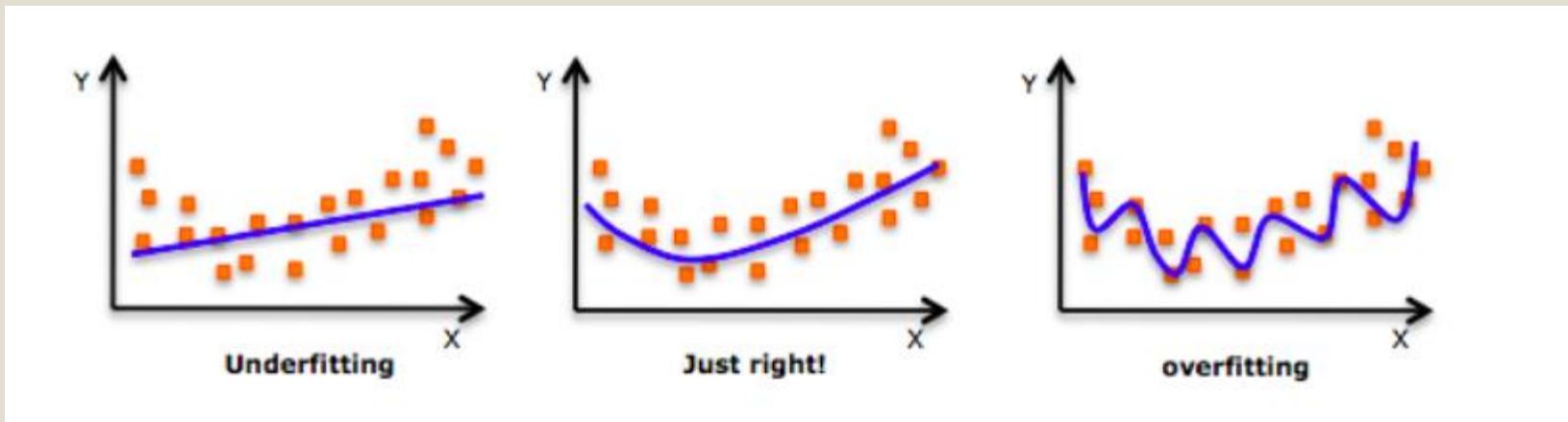


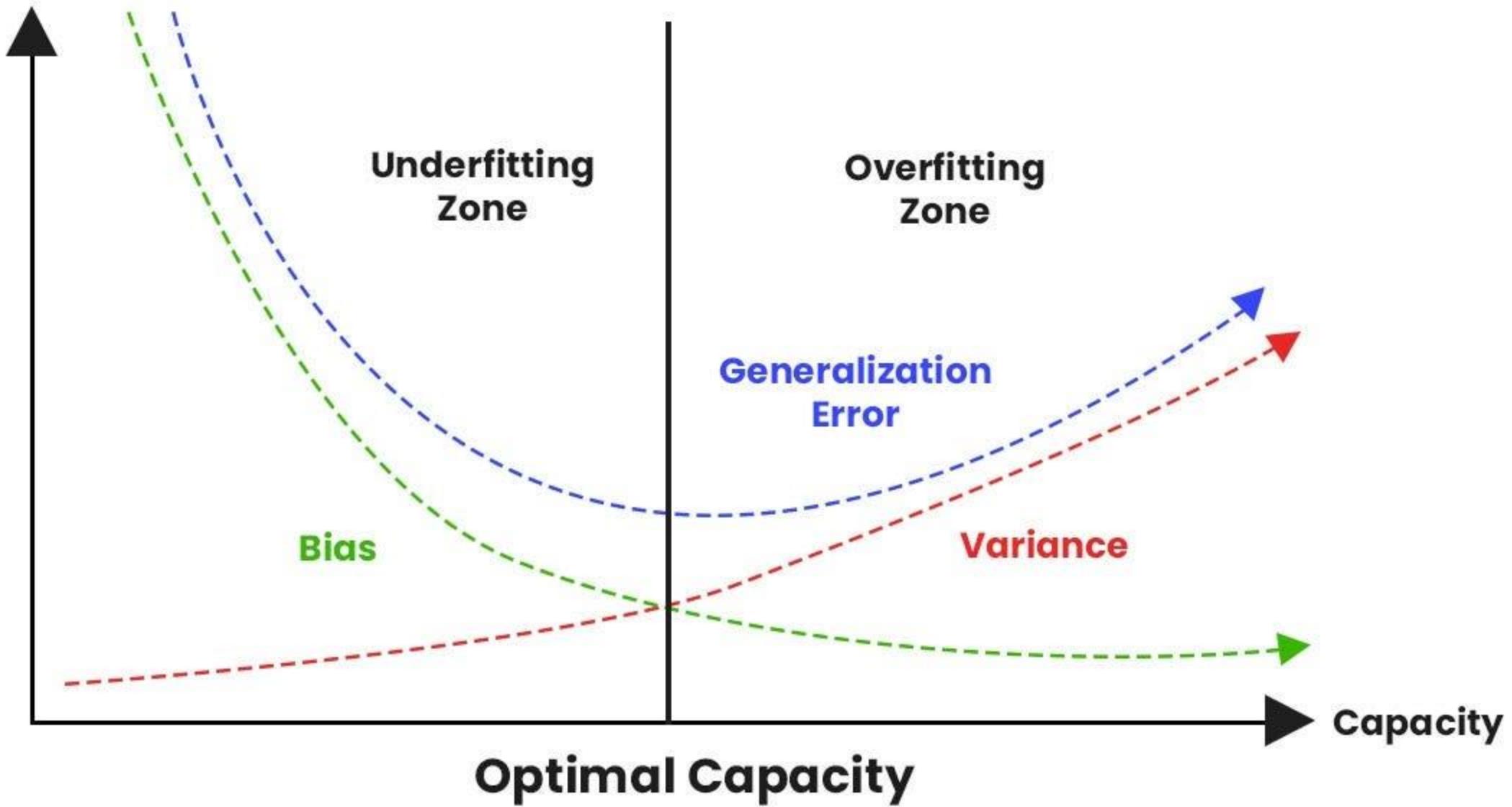
Polynomial model



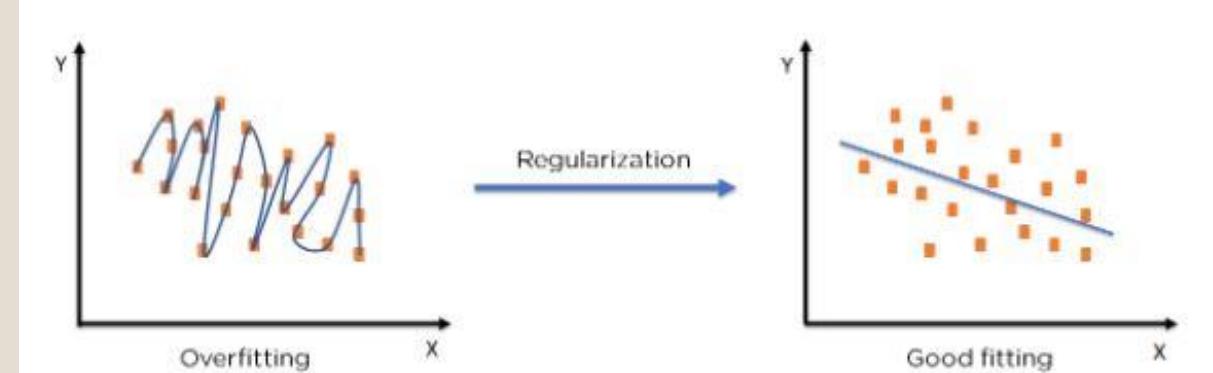
Overfitting and underfitting

- **Generalization** is the model's ability to perform well on unseen data.
- **Underfitting**
 - a data model is unable to capture the relationship between the input and output variables accurately
 - generates a high error rate on both the training set and unseen data.
- **Overfitting**
 - is a phenomenon that occurs when a machine learning model is tailored to a particular dataset and is unable to generalize to other datasets.
 - occurs when a model fits exactly against its training data and is unable to capture the relationship between the input and the output variables accurately.
 - When a model is overfit, the algorithm cannot perform accurately against unseen data.
 - no problem of overfitting with univariate linear regression
 - Multivariate linear regression - ?





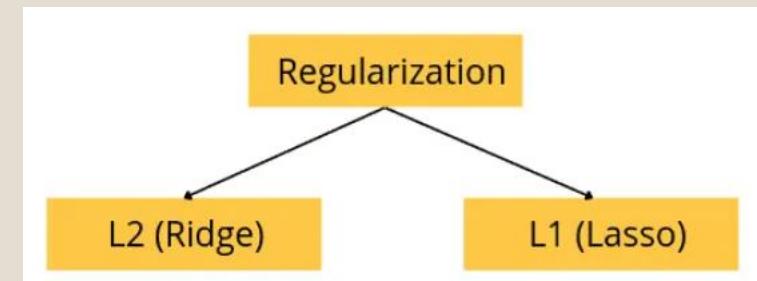
Regularization



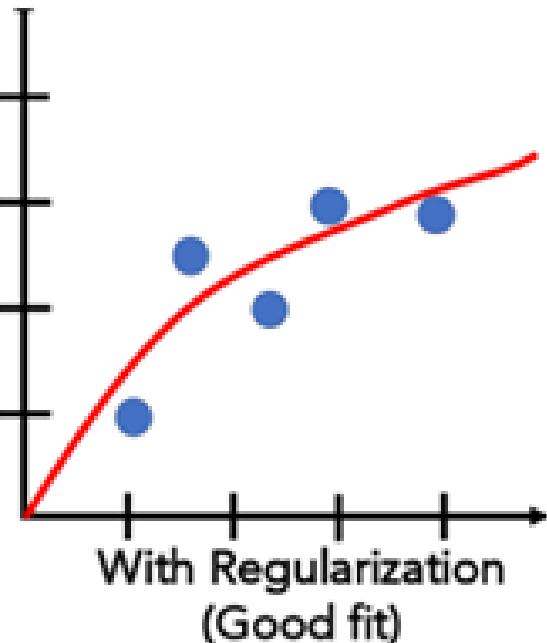
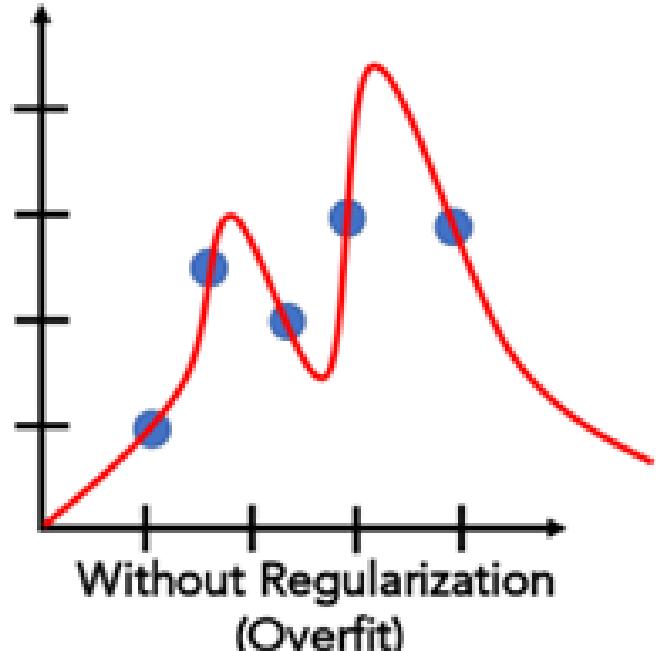
- Regularization is a technique commonly used to avoid overfitting
- With regularization, the total cost of the hypothesis is minimized by counting both empirical loss and the complexity of the hypothesis.
 - $Cost(h) = Empirical\ Loss(h) + \lambda Complexity(h)$
- For a linear function, the complexity can be specified as a function of weights.
 - $Complexity(h) = L_q(w) = \sum_i |w_i|^q$

Regularization(contd...)

- adds the penalty as model complexity increases.
- Regularization term (**lambda**) penalizes all the parameters except intercept so that model generalizes the data and won't overfit.
- penalizing the parameters fulfils two main objectives.
 - i. The order of the curve that is to be fitted got simpler
 - ii. Overfitting reduced.
- The new cost function controls the trade-off between two goals.
 1. Fitting the training set well, which is ensured by the first term (Empirical loss) of the cost function.
 2. Keeping the parameters small, which is ensured by regularization term. lambda is the regularization parameter.



Impact of Regularization



L1 regularization

- also known as **L1-norm** or **Lasso** combats overfitting by shrinking the parameters towards 0.
- makes some features obsolete.
- The **q** value is **1** in L1 regularization
- minimizes the sum of absolute values and penalizes the absolute value of the weights.

$$Cost(\mathbf{h}) = \sum_{j=1}^N (y_j - h_w(x_j))^2 + \lambda \sum_i |w_i|$$

- It is a form of feature selection. Lasso shrinks the coefficient of irrelevant or less important features to zero and eventually helps in feature selection.
- L1 regularization return a sparse solution.

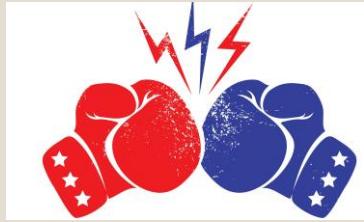
L2 Regularization

- Also known as the **L2-norm**, or **Ridge** combats overfitting by forcing weights to be small, but not making them exactly zero.
- When L2 regularization is used, the less significant features also have some influence over the final prediction.

$$Cost(h) = \sum_{j=1}^N (y_j - h_w(x_j))^2 + \lambda \sum_i w_i^2$$

- L2 regularization return a non-sparse solution since the weights will be non-zero.

L1

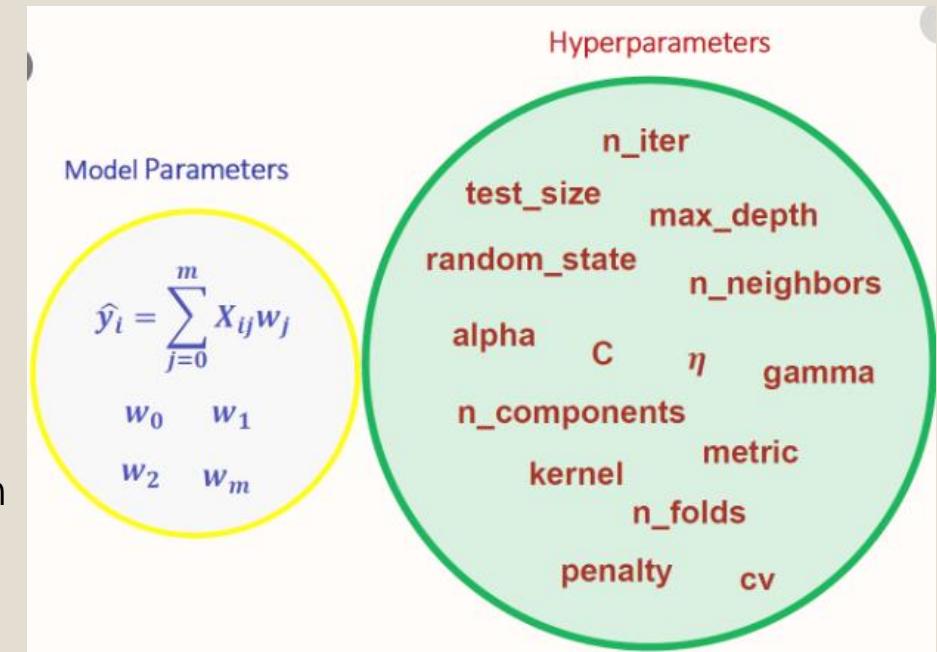
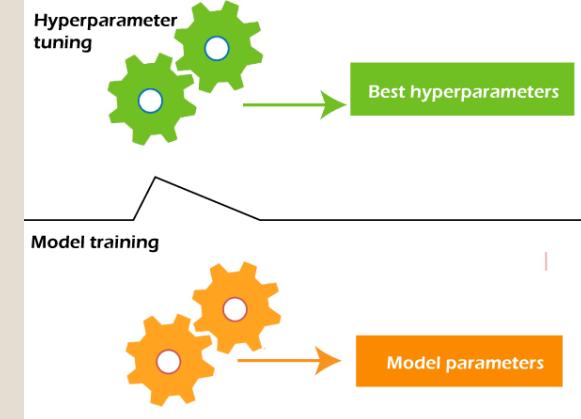


L2

- L1 regularization penalizes the sum of absolute values of the weights, whereas L2 regularization penalizes the sum of squares of the weights.
- The L1 regularization solution is sparse. The L2 regularization solution is non-sparse.
- L2 regularization doesn't perform feature selection, since weights are only reduced to values near 0 instead of 0. L1 regularization has built-in feature selection.
- L1 regularization is robust to outliers, L2 regularization is not.

Hyperparameter Tuning

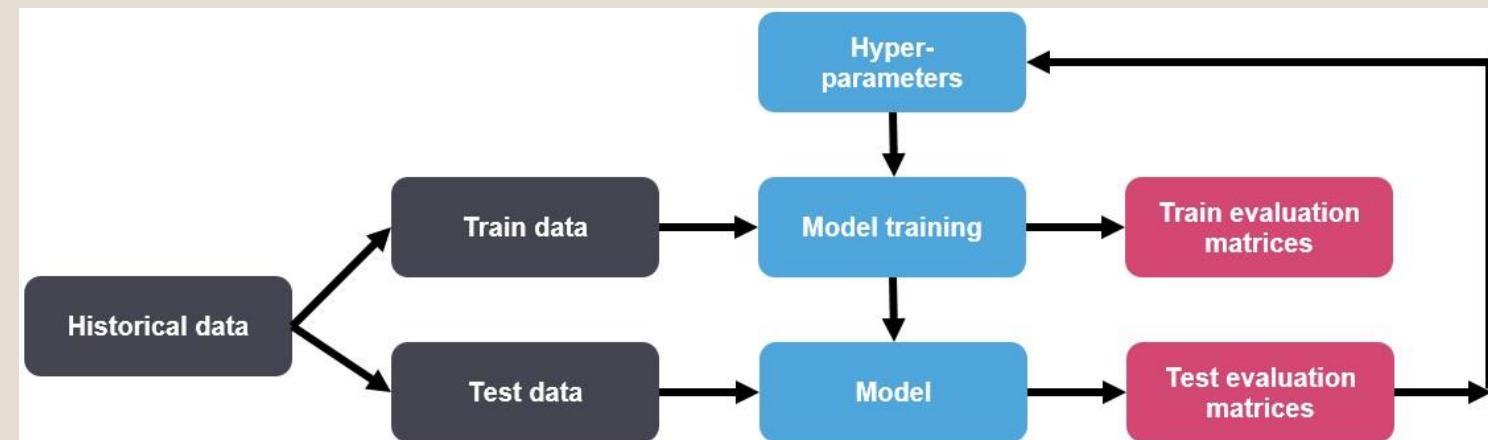
- A hyperparameter is a model argument whose value is set before the learning process begins.
- Key to Machine Learning Algorithms
- choosing a set of optimal hyperparameters for a learning algorithm
- Some hyperparameters
 - K in K-NN
 - Regularization constant, kernel type, and constants in SVMs
 - Number of layers, number of units per layer, regularization in neural network



Hyperparameter Tuning

- Optimal Hyperparameters: Hyperparameters control the over-fitting and under-fitting of the model. Optimal hyperparameters often differ for different datasets. To get the best hyperparameters the following steps are followed:
 - For each proposed hyperparameter setting the model is evaluated
 - The hyperparameters that give the best model are selected.

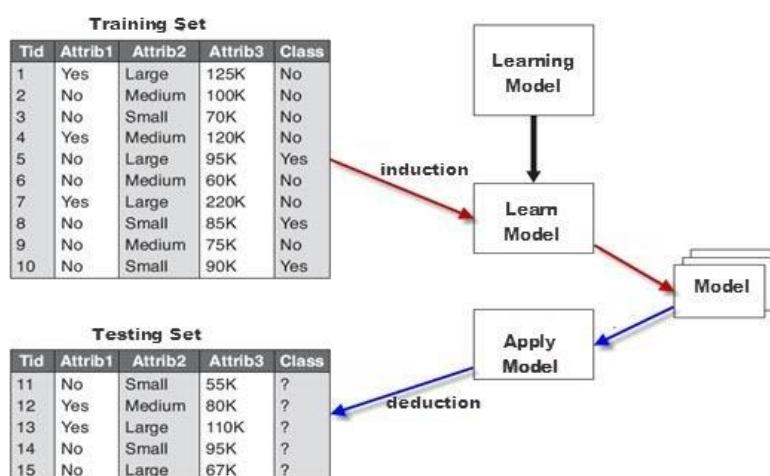
- Popular Techniques
 - Grid Search
 - Random Search



Supervised Learning - Classification Techniques

Basic concepts and applications of classification - Naïve Bayes Classification, Logistic Regression, K-Nearest Neighbors, Classification Trees, Support Vector Machines, Evaluation Measures for Classification Techniques

- Classification is the task of assigning objects to one of several predefined categories.
- Def :** Classification is the task of learning a target function f that maps each attribute set X to one of the predefined class labels y .
- The target function is known as classification model.
- The input data for classification task is a collection of records. Each record is a tuple (X, y) , where X is the attribute set and y is the label.
- The class label y must be discrete. The attribute set can contain both discrete and continuous.
- A classification technique is a systematic approach to build classification model from an input dataset. Classification algorithms are most suited for predicting or describing datasets with binary or nominal categories. Some of the classification algorithms are
 - Logistic Regression
 - Naïve Bayes
 - Decision Tree Classifier
 - KNN classifier
 - Neural Networks
 - Support Vector Machines
- Each technique listed above employs a learning algorithm to identify a model that best fits the relationship between the attribute set X and the class label y of the input data.
- The key objective of the learning algorithm is to build models with good generalization capability
- The following figure shows the general approach for solving a classification problem.



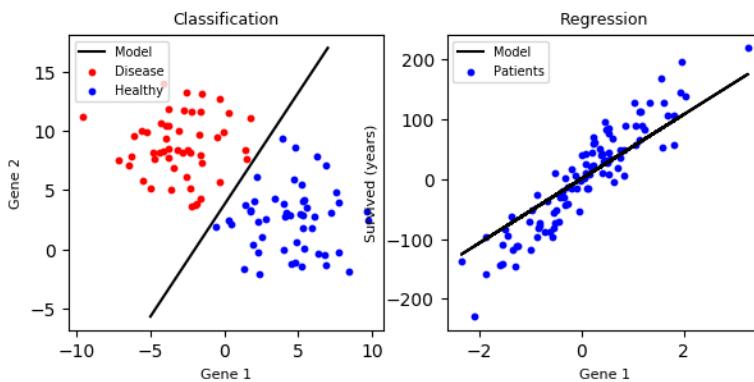
- A classification framework generally involves a 2-step process
 - An inductive step – for constructing a classification model from data.
 - A deductive step – for applying the model to test examples.

Unit – II Fundamentals of Machine Learning

- Classification can be used with many diverse applications
 - Detecting spam email messages based on header and content.
 - Categorizing cells as malignant or benign based on MRI scans.
 - Classifying galaxies based on their shapes.
 - Biometric identification
 - Drug Classification
 - Speech Recognition

Linear Models

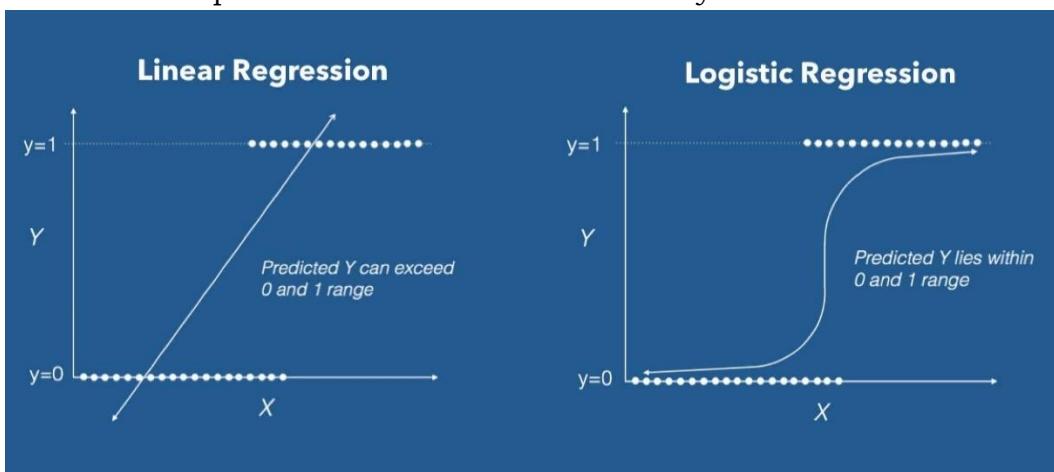
- Linear models are the class of models that make prediction using a linear function of the input features.
- For linear regression, the general prediction formula is
$$\hat{y} = w_0 + w_1x_1 + w_2x_2 + \dots + w_px_p$$
- Linear regression finds the parameters which minimizes the mean squared error between y and \hat{y} . The discrepancy between the performance of the training set and the test set is the clear sign of overfitting.
- These linear models can also be used for classification by following
$$\hat{y} = wx + b > 0$$
- Instead of just returning the linear sum, we threshold the predicted value at 0
 - If $wx + b < 0$ then the predicted class is -1.
 - If $wx + b > 0$ then the predicted class is 1.
- The binary linear classifier is a classifier that separates two classes using a line, a plane, or a hyperplane (decision boundary).
- In linear regression, the model comes up with the best fit line and in linear classifier the model learns a decision boundary.



- The two most common linear classification algorithms are
 - Logistic Regression
 - Support Vector Machines.

Logistic Regression

- Logistic regression is a statistical method for predicting binary classes. The outcome or target variable is dichotomous in nature. Dichotomous means there are only two possible classes. For example, it can be used for cancer detection problems. It computes the probability of an event occurrence.
- It is a special case of linear regression where the target variable is categorical in nature. Logistic Regression predicts the probability of occurrence of a binary event utilizing a logit function.
- Passing the output of a linear function through the threshold function creates a classifier. But the hard nature of the threshold causes few problems
 - Outliers results in misclassification problem
 - This classifier announces completely a confident prediction of 1 or 0, even to the examples that are close to the boundary.



- These issues can be resolved by softening the threshold function. The threshold function is approximated using a continuous, differentiable logistic function (sigmoid)

$$\text{Logistic}(z) = \frac{1}{1 + e^{-z}}$$

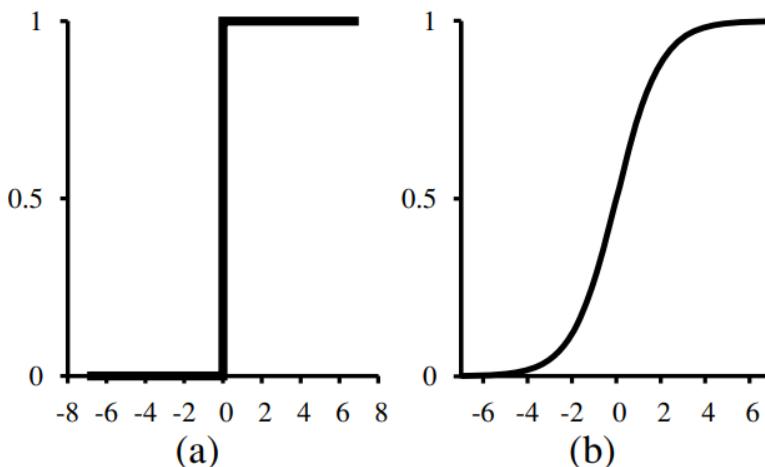


Figure (a) shows the threshold with 0/1 output. It is non-differentiable at $z=0$

Figure (b) shows the logistic function (also called sigmoid)

Unit – II Fundamentals of Machine Learning

- Sigmoid has most convenient mathematical properties.

$$h_w(x) = \text{Logistic}(wx) = \frac{1}{1 + e^{-wx}}$$

- The output is a number between 0 and 1. It can be interpreted as a probability of a belonging to a class labelled 1.
- The hypothesis with sigmoid gives a probability of 0.5 for any input at the centre of the boundary region, and approaches 0 or 1 as we move away from the boundary.
- The process of fitting the weights of the model to minimize loss on a dataset is called Logistic Regression.
- Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable. This justifies the name "Logistic Regression".
- We can use Gradient Descent computation to find optimal values of w. Either L2 loss or Cross-Entropy loss can be used in classification.
- Notation followed is
 - g for sigmoid
 - g^1 for derivative of sigmoid
- For a single example (x,y)

$$\begin{aligned} \frac{\partial g(f(x))}{\partial x} &= g^1(f(x)) \cdot \frac{\partial f(x)}{\partial x} \\ \frac{\partial}{\partial w_i} \text{Loss}(W) &= \frac{\partial}{\partial w_i} (y - \hat{y})^2 \\ &= 2(y - \hat{y}) \frac{\partial}{\partial w_i} (y - \hat{y}) \\ &= -2(y - \hat{y}) g^1(w \cdot x) \frac{\partial}{\partial w_i} w \cdot x \\ &= -2(y - \hat{y}) g^1(w \cdot x) x_i \end{aligned}$$

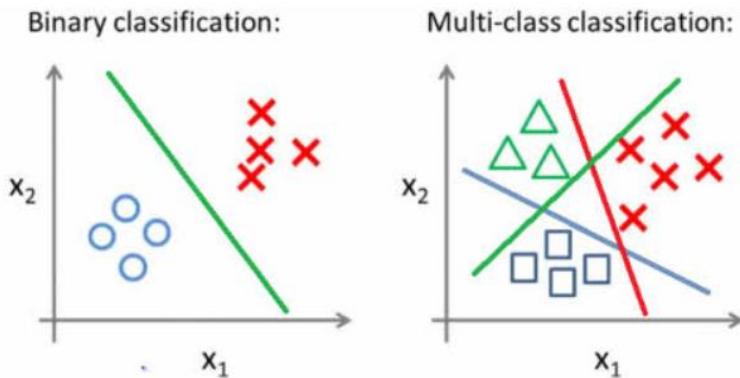
- The derivative of the logistic function is

$$\begin{aligned} g^1(z) &= g(z) \cdot (1 - g(z)) \\ g^1(w \cdot x) &= g(w \cdot x) \cdot (1 - g(w \cdot x)) = \hat{y} \cdot (1 - \hat{y}) \end{aligned}$$

- By using the above derivative, the weight update for minimizing the loss becomes

$$\begin{aligned} w_i &\leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(W) \\ w_i &\leftarrow w_i - \alpha (y - \hat{y}) \hat{y} (1 - \hat{y}) x_i \end{aligned}$$

Binary classification vs. Multi-class classification



Binary Classification

- Only two class instances are present in the dataset.
- It requires only one classifier model.
- Confusion Matrix is easy to derive and understand.
- Example:- Check email is spam or not, predicting gender based on height and weight.

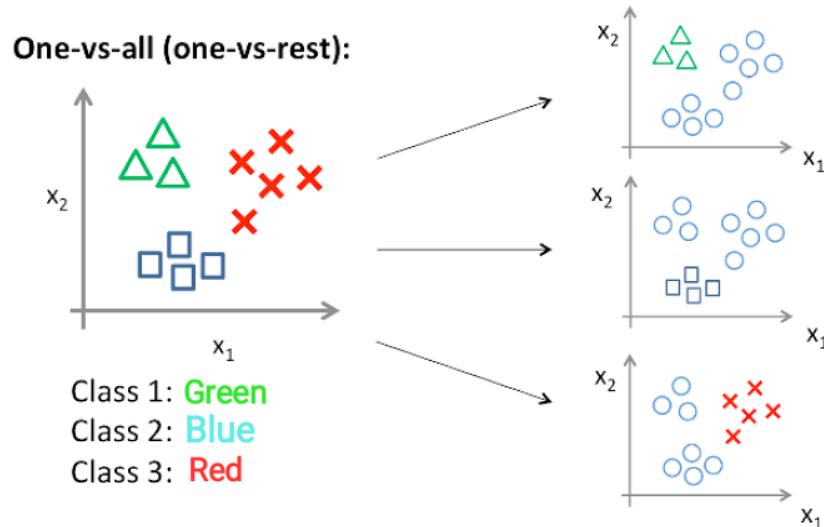
Multi-class Classification

- Multiple class labels are present in the dataset.
- The number of classifier models depends on the classification technique we are applying to.
- One vs. All:- N-class instances then N binary classifier models
- One vs. One:- N-class instances then $N \times (N-1)/2$ binary classifier models
- The Confusion matrix is easy to derive but complex to understand.

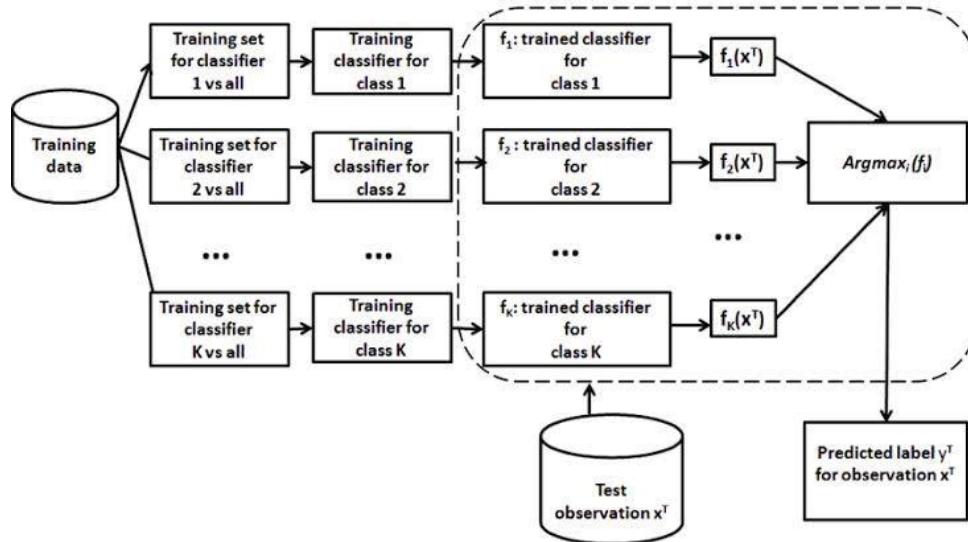
Example:- Check whether the fruit is apple, banana, or orange.

Multiclass classification using logistic regression:

- To perform multiclass classification using logistic regression, one vs rest approach can be used. In one vs. rest (one vs. all) approach a binary model is learned for each class that tries to separate that class from all the other classes.
- In one-vs-All classification, for the N-class instances dataset, we have to generate the N-binary classifier models. The number of class labels present in the dataset and the number of generated binary classifiers must be the same.



- Three classifiers need to be created for the above data.
 - Classifier 1:- [Green] vs [Red, Blue]
 - Classifier 2:- [Blue] vs [Green, Red]
 - Classifier 3:- [Red] vs [Blue, Green]
- To make a prediction, all the binary classifiers are run on a test point.
- The classifier that has the highest score on its single class wins and this class label is returned as the prediction. this entire process of prediction is explained in the below figure.



Advantages of Logistic Regression:

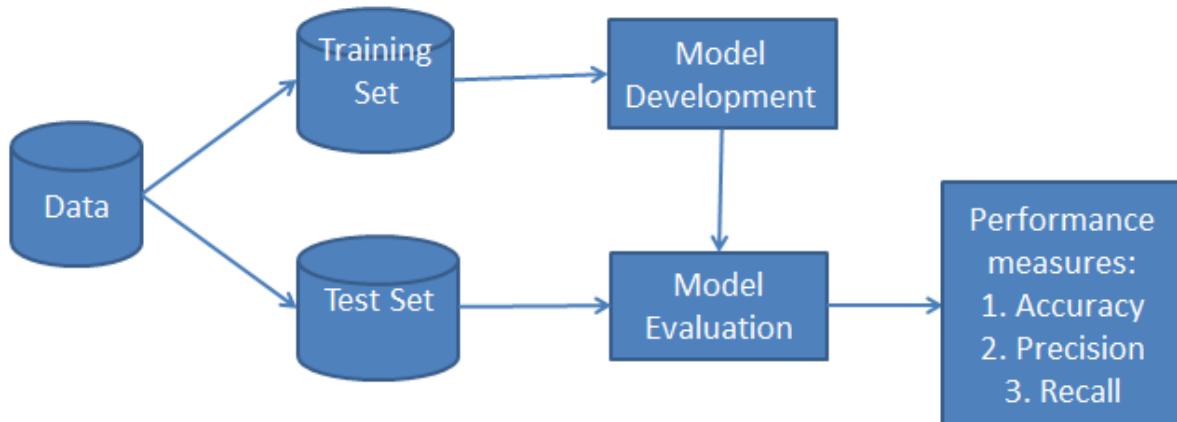
- Logistic regression is easier to implement, interpret, and very efficient to train.
- It makes no assumptions about distributions of classes in feature space.
- It can easily extend to multiple classes
- It is very fast at classifying unknown records.
- Good accuracy for many simple data sets and it performs well when the dataset is linearly separable
- It can interpret model coefficients as indicators of feature importance.
- Logistic regression is less inclined to over-fitting but it can overfit in high dimensional datasets. One may consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.

Disadvantages of Logistic Regression:

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting.
- It constructs linear boundaries.
- The major limitation is the assumption of linearity between the dependent variable and the independent variables.
- Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

Evaluation measures for classification

- The classification has two phases, a learning phase, and the evaluation phase. In the learning phase, classifier trains its model on a given dataset and in the evaluation phase, it tests the classifier performance. Performance is evaluated on the basis of various parameters such as accuracy, error, precision, and recall.



- Evaluation of the performance of a classification model is based on the counts of the test records correctly or incorrectly predicted by the model. These counts are tabulated in a table called as Confusion matrix.
- Consider the spam detection task
 - It is a binary decision task.
 - The goal of this task is to label every text as being in the spam category (positive) or not in the spam category (negative).
 - For each email document, we need to know whether it is spam or not.
 - We should also know whether the email is actually spam or not. These are human defined labels for each document and are referred as the gold labels.
 - We need a metric to know how well the spam detector is doing.
- Building a confusion matrix will help us evaluate any system.
- Def: **A confusion matrix** is a table for visualizing how an algorithm performs with respect to the human gold labels, using two dimensions (system output and gold labels), and each cell labelling a set of possible outcomes.
- For example, in the spam detection case example,
 - true positives are documents that are indeed spam that our system correctly said were spam.
 - False negatives are documents that are indeed spam but our system incorrectly labelled as non-spam
- In simple words
 - A true positive is an outcome where the model correctly predicts the positive class. Similarly, a true negative is an outcome where the model correctly predicts the negative class.
 - A false positive is an outcome where the model incorrectly predicts the positive class. And a false negative is an outcome where the model incorrectly predicts the negative class.

Unit – II Fundamentals of Machine Learning

The following figure shows a confusion matrix for visualizing how well a binary classification system performs against gold standard labels.

		gold standard labels		
		gold positive	gold negative	
system output labels	system positive	true positive	false positive	precision = $\frac{tp}{tp+fp}$
	system negative	false negative	true negative	
		recall = $\frac{tp}{tp+fn}$		accuracy = $\frac{tp+tn}{tp+fp+tn+fn}$

Accuracy:

- accuracy tells what percentage of all the observations our system labelled correctly
- accuracy is not used for text classification tasks as it does not work well when the classes are unbalanced.
- Accuracy is defined as

$$\text{accuracy} = \frac{\text{true positives} + \text{true negatives}}{\text{true positives} + \text{false positives} + \text{true negatives} + \text{false negatives}}$$

Precision:

- Precision measures the percentage of the items that the system detected (i.e., the system labeled as positive) that are in fact positive (i.e., are positive according to the human gold labels).
- Precision is defined as

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

Recall:

- Recall measures the percentage of items actually present in the input that were correctly identified by the system.
- Recall is defined as

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

- Unlike accuracy, precision and recall emphasize true positives: finding the things that we are supposed to be looking for

F-measure:

- F-measure is a single metric that incorporates aspects of both Precision and Recall. It is defined as

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

- The β parameter differentially weights the importance of recall and precision, based on the needs of an application.
- Values of $\beta > 1$ favour recall, while values of $\beta < 1$ favour precision.
- When $\beta = 1$, precision and recall are equally balanced; this is the most frequently used metric, and is called $F_{\beta=1}$ or just F

$$F_1 = \frac{2PR}{P+R}$$

Evaluating with more than two classes

- Many of the classification tasks in language processing have more than two classes.
For eg. Sentiment Analysis can have 3 classes (positive, negative, neutral)

- The figure below shows a confusion matrix for a three-class categorization task

		gold labels			
		urgent	normal	spam	
system output	urgent	8	10	1	$\text{precision}_u = \frac{8}{8+10+1}$
	normal	5	60	50	$\text{precision}_n = \frac{60}{5+60+50}$
	spam	3	30	200	$\text{precision}_s = \frac{200}{3+30+200}$
		$\text{recall}_u = \frac{8}{8+5+3}$	$\text{recall}_n = \frac{60}{10+60+30}$	$\text{recall}_s = \frac{200}{1+50+200}$	

Exercise :

n=165	Predicted:	
	NO	YES
Actual: NO	50	10
Actual: YES	5	100

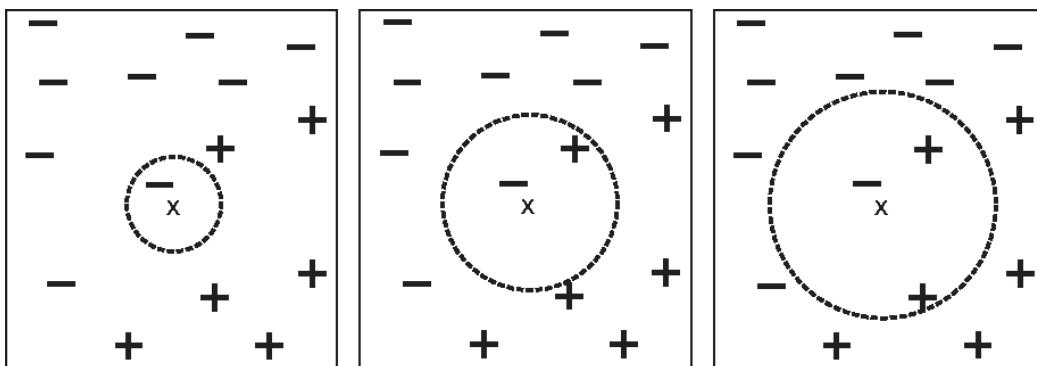
For the above confusion matrix, calculate and fill the following.

True positives	
False positives	
False negatives	
True negatives	
Precision	
Recall	
F1-Score	
accuracy	

K-Nearest Neighbors Classification

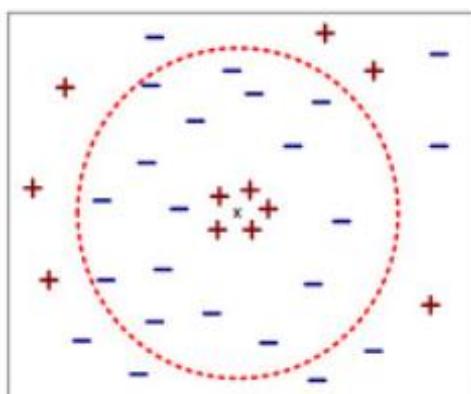
"If it walks like a duck, quacks like a duck, and looks like a duck, then its probably a duck."

- KNN is a very simple and widely used algorithm to solve classification problems. KNN stands for K-Nearest Neighbors. K is the number of Neighbors in KNN.
- In KNN, a new data point is classified based on similarity in the specific group of neighboring points. Given a test point x_T , find the k examples that are nearest to x_T . This is called K-Nearest Neighbors algorithm.
- A Nearest Neighbor Classifier represents each example as a data point in a d-dimensional space, where d is the number of attributes.
- Given a test example, we compute its proximity with rest of the data points in training data. The K-nearest Neighbors of a given example z refer to the k points that are closest to z.



The above figure shows the 1-, 2-, 3- nearest neighbors of data point located at the center of each circle. The data point is classified based on the majority class labels of its neighbors.

- Choosing the right value of k is very important.
 - If k is too small, then the Nearest Neighbors classifier may be susceptible to overfitting, because of the noise in the training data.
 - If k is too large, the Nearest Neighbors classifier may misclassify the test instance because of its list of nearest Neighbors may include data points that are located far away from its Neighbors as shown in below figure.



Algorithm:

A high-level summary of the KNN classification method is present in the following algorithm.

1. Let k be the number of Nearest Neighbors and D be the set of training examples.
 2. For each training example do
 - a. Compute the $d(x^1, x)$ distance between z and every example $(x, y) \in D$.
 - b. Select $D_z \subseteq D$, the set of k closest training examples to z
 - c. $y^1 = \operatorname{argmax}_v \sum_{(x_i, y_i) \in D_z} I(v = y_i)$
 3. end for
- The algorithm computes the distance between each test example $z = (x^1, y^1)$ and all training examples $(x, y) \in D$ to determine its nearest neighbor list D_z .
 - The test example is classified based on the majority class of its nearest neighbors.

$$\text{Majority voting } y^1 = \operatorname{argmax}_v \sum_{(x_i, y_i) \in D_z} I(v = y_i)$$

Where v , is the class label, y_i is the class label for one the nearest neighbors. $I(\cdot)$ is a function, that returns the value 1 if its argument is true, and 0 otherwise.

Distance calculation:

- In Nearest Neighbors, the word “nearest” implies to a distance metric. Typically, the distances are measured using Minkowski distance or L^P norm.

$$L^P(x, y) = \left(\sum_i |x_i - y_i|^P \right)^{1/P}$$

- With $P=2$, this is called Euclidean distance or L2-norm

$$L^2(x, y) = \sqrt{\sum_i (x_i - y_i)^2}$$

- With $P=1$, this is called Manhattan distance or L1-norm

$$L^1(x, y) = \sum_i |x_i - y_i|$$

- $P=2$ is used if the dimensions are measuring similar properties, such as width, height and depth of parts on a conveyer belt.
- $P=1$ is used if the dimensions are dissimilar such as age, weight, gender of a patient.

Normalization:

- If we use raw numbers from each dimension then the total distance will be affected by a change in scale in any dimension.
- For eg: if we change dimension I from measurements in centimeters to miles while keeping the other dimensions the same, we will get different nearest neighbors.
- To avoid this problem, normalization is applied to measurements in each dimension.
- To do normalization, compute mean (μ_i) and standard deviation (σ_i) of the values in each dimension, and rescale them so that x_{ji} becomes

$$x_{ji} = \frac{x_{ji} - \mu_i}{\sigma_i}$$

Advantages of using KNN

1. **No Training Period:** KNN is called **Lazy Learner** (Instance based learning). It does not learn anything in the training period. It does not derive any discriminative function from the training data. In other words, there is no training period for it. It stores the training dataset and learns from it only at the time of making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g. SVM, Linear Regression etc.
2. Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly which will not impact the accuracy of the algorithm.
3. **KNN is very easy to implement.** There are only two parameters required to implement KNN i.e. the value of K and the distance metric (e.g. Euclidean or Manhattan etc.)

Disadvantages of KNN

1. **Does not work well with large dataset:** In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.
2. **Does not work well with high dimensions:** The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.
3. **Need feature scaling:** We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.
4. sensitive to outliers.

Naïve Bayes Classifier

- Naive Bayes is the most straightforward and fast classification algorithm, which is suitable for a large chunk of data. Naive Bayes classifier is successfully used in various applications such as spam filtering, text classification, sentiment analysis, and recommender systems. It uses Bayes theorem of probability for prediction of unknown class.

Bayes Theorem

- Let X, Y be a pair of random variables.
- The **joint probability** $P(X=x, Y=y)$ refers to the probability that variable X will take on the value x and variable Y will take on the value y.
- The **conditional probability** $P(Y=y | X=x)$ refers to probability that the variable Y will take on the value y, given that the variable X is observed to have the value x.

$$P(X, Y) = P(Y|X) P(X)$$
$$P(Y, X) = P(X|Y) P(Y)$$

- Rearranging the above two, we get

$$P(Y|X) P(X) = P(X|Y) P(Y)$$

Unit – II Fundamentals of Machine Learning

- From this we get bayes theorem.

$$P(Y|X) = \frac{P(X|Y) P(Y)}{P(X)}$$

- This Bayes theorem can be used to solve prediction problem.
- In many applications, the relationship between the attribute set (X) and the class variable (y) is **non-deterministic**. Bayes theorem can be used in these scenarios to do classification.
- If the class variable has a non-deterministic relationship with the attributes, then we can treat X and y as random variables and capture their relationship probability using $P(y|X)$
- The conditional probability $P(y|X)$ is also known as **posterior probability** for y as opposed to its **prior probability** $P(y)$
- During the training phase, we need to learn the posterior probabilities $P(y|X)$ for every combination of X and y, based on the information gathered from the training data.
- By knowing these probabilities, a test record \hat{X} can be classified as by finding the class \hat{y} that maximizes the posterior probability $P(\hat{y}|\hat{X})$

Consider an example task of prediction whether a loan borrower will default on their payments. Loan borrowers who defaulted on their payments are classified as yes, while those who repaid their loans are classified as no.

Consider a test record,

$X = (\text{Home owner} = \text{No}, \text{Marital Status} = \text{married}, \text{Annual income} = 5\text{Lakhs})$.

To classify the above record, we need compute $P(\text{Yes} | X)$ and $P(\text{No} | X)$ based on the training data.

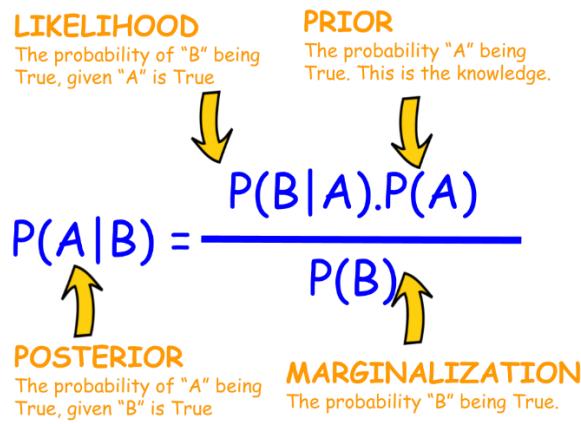
If $P(\text{Yes} | X) > P(\text{No} | X)$ then

The test record is classified as **yes**.

Otherwise

The test record is classified as **no**.

Bayes theorem can be employed to find these probabilities



$$P(Y|X) = \frac{P(X|Y) P(Y)}{P(X)}$$

As $P(X)$ is constant, it can be ignored. $P(Y)$ can be easily estimated from the training set.

To compute $P(X|Y)$, we follow Naïve Bayes classification method

Naïve Bayes Classifier:

- A naïve bayes classifier estimates the class-conditional probability by assuming that the attributes are conditionally independent, given the class label y.
- If X is set of d attributes $X = \{x_1, x_2, \dots, x_d\}$, then the conditional independence can be stated as

$$P(X | Y = y) = P(x_1|y).P(x_2|y) \dots .P(x_d|y)$$

$$P(X | Y = y) = \prod_{i=1}^d P(x_i|y)$$

- To classify a test record, the Naïve Bayes classifier computes the posterior probability for each class y.

$$P(y | X) = \frac{P(y) \prod_{i=1}^d P(x_i|y)}{P(X)}$$

- Since $P(X)$ is fixed for every y, it is sufficient to choose the class that maximizes the numerator term

$$P(y) \prod_{i=1}^d P(x_i|y)$$

- For a categorical attribute x_i , the conditional probability $P(X=x_i | Y=y)$ is estimated according to the fraction of training instances in class y that takes on a particular attribute value x_i .
- For eg: 3 out of 7 people who repaid their loans also own a home.

Example 1:

Classify whether players play or not based on weather conditions.

Below we have a training data set of weather and corresponding target variable 'Play' (suggesting possibilities of playing). Now, we need to classify whether players will play or not based on weather condition. Let's follow the below steps to perform it.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

Step 1: Create a frequency table from dataset

Frequency Table		
Weather	No	Yes
Overcast		4
Rainy	3	2
Sunny	2	3
Grand Total	5	9

Step 2 : Create a likelihood table

Likelihood table				
Weather	No	Yes		
Overcast		4	=4/14	0.29
Rainy	3	2	=5/14	0.36
Sunny	2	3	=5/14	0.36
All	5	9		
	=5/14	=9/14		
	0.36	0.64		

Step 3: Now, use [Naive Bayesian](#) equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

- **Problem:** Players will play if weather is sunny. Is this statement is correct?
- $P(\text{Yes} | \text{Sunny}) = P(\text{ Sunny} | \text{ Yes}) * P(\text{Yes}) / P(\text{Sunny})$
- Here we have
 - $P(\text{Sunny} | \text{Yes}) = 3/9 = 0.33$,
 - $P(\text{Sunny}) = 5/14 = 0.36$,
 - $P(\text{ Yes})= 9/14 = 0.64$
- $P(\text{Yes} | \text{Sunny}) = 0.33 * 0.64 / 0.36 = 0.60$, which has higher probability.

Example 2 :

Predicting whether a player play golf or not based on 4 weather conditions – Outlook, temperature, humidity, windy.

Attributes				Classes
Outlook	Temperature	Humidity	Windy	Play Golf
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Overcast	Hot	High	FALSE	Yes
Sunny	Mild	High	FALSE	Yes
Sunny	Cool	Normal	FALSE	Yes
Sunny	Cool	Normal	TRUE	No
Overcast	Cool	Normal	TRUE	Yes
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Sunny	Mild	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes
Overcast	Mild	High	TRUE	Yes
Overcast	Hot	Normal	FALSE	Yes
Sunny	Mild	High	TRUE	No

Prior probabilities:

Play		P(Yes)/P(No)
Yes	9	9/14
No	5	5/14
Total	14	100%

Frequency tables

Outlook					Temperature				
	Yes	No	P(yes)	P(no)		Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5	Hot	2	2	2/9	2/5
Overcast	4	0	4/9	0/5	Mild	4	2	4/9	2/5
Rainy	3	2	3/9	2/5	Cool	3	1	3/9	1/5
Total	9	5	100%	100%	Total	9	5	100%	100%

Humidity					Wind				
	Yes	No	P(yes)	P(no)		Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5	False	6	2	6/9	2/5
Normal	6	1	6/9	1/5	True	3	3	3/9	3/5
Total	9	5	100%	100%	Total	9	5	100%	100%

Consider a test record, today = (**Sunny, Hot, Normal, False**) and predict play golf

The probability of playing golf is given by,

$$P(Yes|today) =$$

$$\frac{P(Outlook = Sunny|Yes) P(Temperature = Hot|Yes) P(Humidity = Normal|Yes) P(Wind = False|Yes) P(Yes)}{P(today)}$$

The probability to not play golf is given by

$$P(No|today) =$$

$$\frac{P(Outlook = Sunny|No) P(Temperature = Hot|No) P(Humidity = Normal|No) P(Wind = False|No) P(No)}{P(today)}$$

Since, $P(today)$ is common in both probabilities, we can ignore $P(today)$ and find proportional probabilities as:

$$P(Yes|today) \propto \frac{2}{9} \cdot \frac{2}{9} \cdot \frac{6}{9} \cdot \frac{9}{14} \approx 0.0141$$

and

$$P(No|today) \propto \frac{3}{5} \cdot \frac{2}{5} \cdot \frac{1}{5} \cdot \frac{2}{5} \cdot \frac{5}{14} \approx 0.0068$$

Now, since

$$P(Yes|today) + P(No|today) = 1$$

These numbers can be converted into a probability by making the sum equal to 1 (normalization):

$$P(Yes|today) = \frac{0.0141}{0.0141+0.0068} = 0.67$$

and

$$P(No|today) = \frac{0.0068}{0.0141+0.0068} = 0.33$$

Since

$$P(Yes|today) > P(No|today)$$

So, prediction that golf would be played is 'Yes'.

Advantages of Naïve Bayes Classification

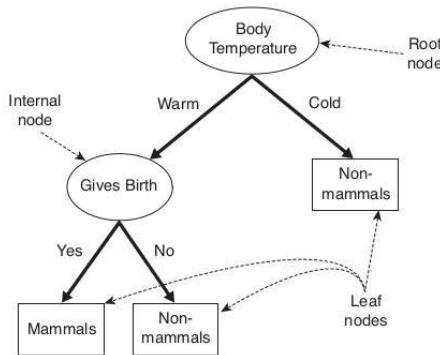
- It is not only a simple approach but also a fast and accurate method for prediction.
- Naive Bayes has very low computation cost.
- It can efficiently work on a large dataset.
- It performs well in case of discrete response variable compared to the continuous variable.
- It can be used with multiple class prediction problems.
- It also performs well in the case of text analytics problems.
- When the assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression.

Disadvantages of Naïve Bayes Classification

- The assumption of independent features. In practice, it is almost impossible that model will get a set of predictors which are entirely independent.
- If there is no training tuple of a particular class, this causes zero posterior probability. In this case, the model is unable to make predictions. This problem is known as Zero Probability/Frequency Problem.

Decision Tree Classification

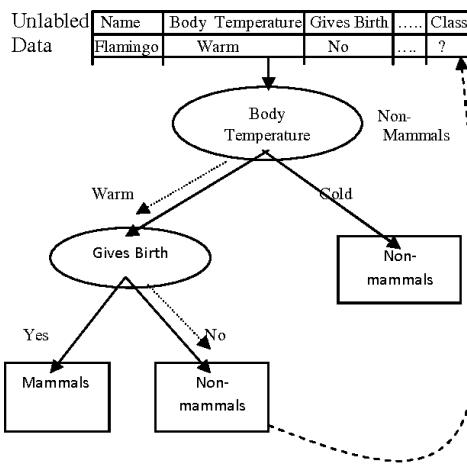
- Decision Tree classification is a simple and widely used classification technique.
- Consider a simple vertebrate classification problem (mammal / non-mammal) to understand decision tree. Decision Tree approach works by passing a series of questions about the characteristics of the species. The series of questions and possible answers can be organized in the form of a decision tree as shown below.



- The tree has 3 types of nodes.
 - **A root node** – that has no incoming edges and 0 or more outgoing edges.
 - **Internal node** – Each of which has exactly one incoming edge and two or more outgoing edges.
 - **Leaf / Terminal nodes** – each of which has exactly one incoming edge, and no outgoing edge.
- In a Decision Tree, each leaf node is assigned a class label.
- The root and internal nodes contain attribute test conditions to separate records that have different characteristics.
- Classifying a test record becomes straight forward once a Decision Tree has been constructed. The tree is traversed starting from root node to leaf node. The class label associated with the leaf node will be assigned to the test record.

Unit – II Fundamentals of Machine Learning

- Following unlabeled test record can be classified using Decision Tree as shown below.



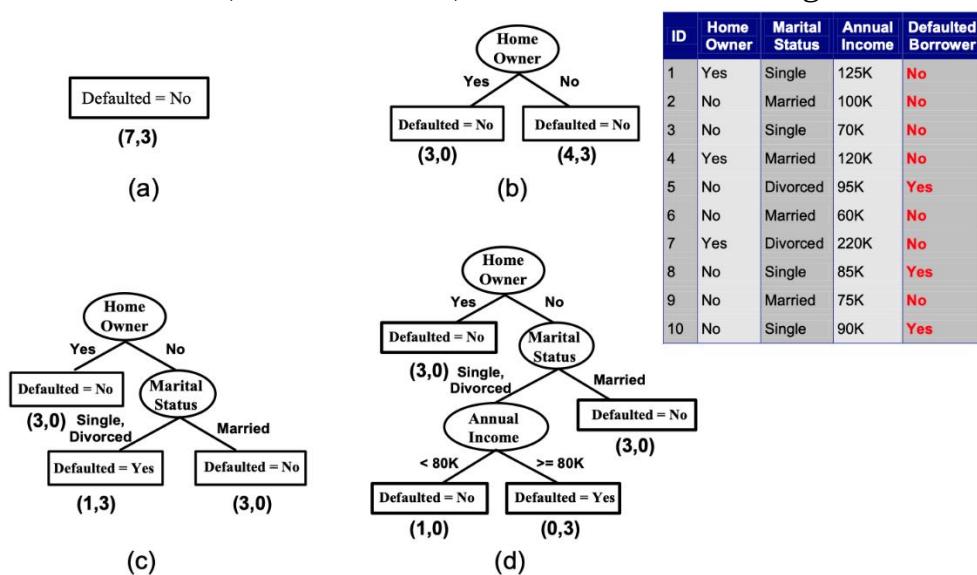
Building a Decision Tree:

- Hunt's algorithm is the basis of existing Decision Tree Induction algorithms (ID3, C4.5, CART etc.)
- In Hunt's algorithm a Decision Tree is grown in a recursive fashion by partitioning the training records into purer subsets.
- Let D_t be the set of training records that are associated with node t and $y = \{y_1, y_2, \dots, y_c\}$ be the class labels.

The Recursive Definition of Hunt's algorithm is

- Step 1 :** If all the records in D_t belong to the same class y_t , then t is a leaf node labelled as y_t .
- Step 2 :** If D_t contains records that belong to more than one class, an attribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

For eg: Consider the problem of predicting whether a loan applicant will repay her loan or not. From the decision tree we need to find the class label for unlabeled data. The feature columns are Home Owner, Marital Status, Annual Income and target has Yes or No.



Unit – II Fundamentals of Machine Learning

- Initial Tree has **Defaulted = No** as most of the borrowers successfully repaid the loan.
- The tree needs to be refined as the root node contains records from both the classes.
- The records are divided into smaller subsets based on outcomes of **Home Owner** test condition as shown in figure b. (Why Homeowner is chosen as attribute condition?)
- All borrowers who are Home Owners successfully repaid their loans. The left child of the root node is therefore a leaf node labelled **Defaulted=No**.
- For the right child apply the recursive step of Hunt's algorithm until all the records belong to the same class. The resulting trees are shown in figures C and D.

Design Issues of Decision Tree Induction

A learning algorithm for inducing decision trees must address the following two issues.

- How should the training records be split?
- How should the splitting procedure stop?

Measures for selecting the best split:

- Many measures can be used to determine the best way to split the records. These measures are defined in terms of the class distribution of the records before and after the split.
- Let $P(i|t)$ denote the fraction of records belonging to class i at node t .
- In a two-class problem, the class distributions at any node can be written as $(P(0|t), P(1|t))$, where $P(1|t) = 1 - P(0|t)$.
- The measures developed for selecting the best split are often based on the degree of impurity of the child nodes.
- A node with class distribution $(0, 1)$ has zero impurity, whereas a node with uniform class distribution $(0.5, 0.5)$ has the highest impurity.
- Examples are impurity measures are

$$\begin{aligned} \text{Entropy}(t) &= - \sum_{i=0}^{C-1} P(i|t) \log_2 P(i|t) \\ \text{Gini}(t) &= 1 - \sum_{i=0}^{C-1} [P(i|t)]^2 \\ \text{classification error}(t) &= 1 - \max_i [P(i|t)] \end{aligned}$$

Where C is the number of classes and $0 \log_2 0 = 0$ in entropy calculations.

- To determine how well a test condition performs, we need to compare the degree of impurity of parent node (before splitting) with the degree of impurity of the child nodes (after splitting). The larger the difference, the better the test condition.
- The gain, Δ , is a criterion that can be used to determine the goodness of split.

$$\Delta = I(\text{parent}) - \sum_{j=1}^k \frac{N(v_j)}{N} \cdot I(v_j)$$

- Where,
 - $I(\cdot)$ is the impurity measure of a given node,
 - N is the total no.of records at the parent node,
 - k is the number of attribute values, and
 - $N(v_j)$ is the number of records associated with the child node, v_j .

Decision tree induction algorithms often choose a test condition that maximizes the gain.

Example: Construct a Decision tree for the given dataset

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

- Let us consider S is the given dataset, and it has 14 examples, i.e., $|S|=14$
- Play Tennis is the target variable and it has two different values – (Yes, No)
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

$$Entropy(S) = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

- Where $p(+)$, is the proportion of positive examples in S and $p(-)$, is the proportion of negative examples in S. In all calculations involving entropy we define $0 \log 0$ to be 0. Entropy is 0 if all members of S belong to the same class.
- S is a collection of 14 examples of a boolean concept, including 9 positive and 5 negative examples [9+, 5].
- Then the entropy of S relative to this Boolean classification is:

$$Entropy(S) = -\frac{9}{14} \log_2 \frac{9}{14} - \frac{5}{14} \log_2 \frac{5}{14} = 0.94$$

- Gain (S, A) of an attribute A relative to a collection of examples S is defined as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

- Values(A) is the set of positive values for attribute A. S_v is the subset of S for which attribute A has value v.

Let us find out Gain of all the attributes to decide the root node.

Gain of attribute wind:

$$Values(wind) = [Weak, Strong]$$

$$S = [9 \text{ Yes}, 5 \text{ No}], \quad S_{weak} = [6 - \text{Yes}, 2 - \text{No}], \quad S_{strong} = [3 - \text{Yes}, 3 - \text{No}]$$

$$\begin{aligned} Gain(S, wind) &= Entropy(S) - \sum_{v \in [weak, strong]} \frac{|S_v|}{|S|} Entropy(S_v) \\ &= Entropy(S) - \frac{8}{14} Entropy(S_{weak}) - \frac{6}{14} Entropy(S_{strong}) \end{aligned}$$

$$Entropy(S_{weak}) = -\frac{6}{8} \log_2 \frac{6}{8} - \frac{2}{8} \log_2 \frac{2}{8} = 0.811$$

$$Entropy(S_{strong}) = -\frac{3}{6} \log_2 \frac{3}{6} - \frac{3}{6} \log_2 \frac{3}{6} = 1$$

Substitute in Gain(S, wind)

$$Gain(S, wind) = 0.94 - \frac{8}{14} (0.811) - \frac{6}{14} (1) = \mathbf{0.048}$$

Gain of attribute outlook:

$$Values(outlook) = [Sunny, overcast, rainy]$$

$$S = [9 \text{ Yes}, 5 \text{ No}], \quad S_{Sunny} = [2 - Y, 3 - N], \quad S_{overcast} = [4 - Y, 0 - N], \quad S_{rainy} = [3 - Y, 2 - N]$$

$$\begin{aligned} Gain(S, outlook) &= Entropy(S) - \sum_{v \in [sunny, overcast, rainy]} \frac{|S_v|}{|S|} Entropy(S_v) \\ Entropy(S_{sunny}) &= -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.97 \\ Entropy(S_{overcast}) &= -\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} = 0 \\ Entropy(S_{rainy}) &= -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.97 \end{aligned}$$

Substitute in Gain(S, outlook)

$$Gain(S, outlook) = 0.94 - \frac{5}{14} (0.97) - \frac{4}{14} (0) - \frac{5}{14} (0.97) = \mathbf{0.246}$$

Gain of attribute Humidity:

$$Values(humidity) = [High, Normal]$$

$$S = [9 \text{ Yes}, 5 \text{ No}], \quad S_{High} = [3 - Y, 4 - N], \quad S_{Normal} = [6 - Y, 1 - N]$$

$$\begin{aligned} Gain(S, Humidity) &= Entropy(S) - \sum_{v \in [high, normal]} \frac{|S_v|}{|S|} Entropy(S_v) \\ Entropy(S_{high}) &= -\frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0.98 \\ Entropy(S_{Normal}) &= -\frac{6}{7} \log_2 \frac{6}{7} - \frac{1}{7} \log_2 \frac{1}{7} = 0.59 \end{aligned}$$

Substitute in Gain(S, Humidity)

$$Gain(S, Humidity) = 0.94 - \frac{7}{14} (0.98) - \frac{7}{14} (0.59) = \mathbf{0.048}$$

Gain of attribute Temperature:

$$Values(\text{temperature}) = [\text{Hot}, \text{Mild}, \text{Cool}]$$

$$S = [9 \text{ Yes}, 5 \text{ No}], S_{\text{Hot}} = [2 - Y, 2 - N], S_{\text{Mild}} = [4 - Y, 2 - N], S_{\text{Cool}} = [3 - Y, 1 - N]$$

$$Gain(S, \text{Temperature}) = Entropy(S) - \sum_{v \in \{\text{Hot}, \text{Mild}, \text{Cool}\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Entropy(S_{\text{Hot}}) = -\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4} = 1$$

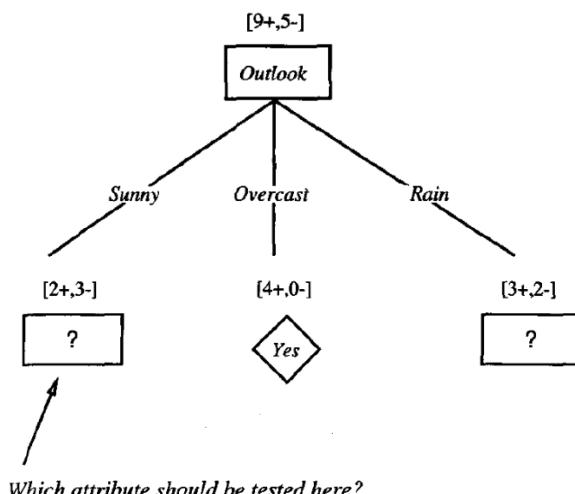
$$Entropy(S_{\text{Mild}}) = -\frac{4}{6} \log_2 \frac{4}{6} - \frac{2}{6} \log_2 \frac{2}{6} = 0.918$$

$$Entropy(S_{\text{Cool}}) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} = 0.811$$

Substitute in Gain(S, Temperature)

$$Gain(S, \text{Humidity}) = 0.94 - \frac{4}{14} (1) - \frac{6}{14} (0.918) - \frac{4}{14} (0.811) = \mathbf{0.029}$$

Outlook is selected as the decision attribute for the root node as its gain is more.



- As all records of outlook=overcast belongs to yes, there won't be any further splitting.

Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

Split the records of overcast = sunny now.

Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

Unit – II Fundamentals of Machine Learning

S_{sunny} is a collection of 5 examples - 2 positive and 3 negative examples [2+, 3].

$$\text{Entropy}(S_{\text{sunny}}) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.97$$

- Now an internal node has to be selected from temperature, humidity and wind.

Gain of attribute Temperature:

$$\text{Values(temperature)} = [\text{Hot}, \text{Mild}, \text{Cool}]$$

$$S = [2 \text{ Yes}, 3 \text{ No}], S_{\text{Hot}} = [0 - Y, 2 - N], S_{\text{Mild}} = [1 - Y, 1 - N], S_{\text{Cool}} = [1 - Y, 0 - N]$$

$$\text{Entropy}(S_{\text{Hot}}) = 0$$

$$\text{Entropy}(S_{\text{Mild}}) = 1$$

$$\text{Entropy}(S_{\text{Cool}}) = 0$$

$$\text{Gain}(S, \text{Humidity}) = 0.97 - \frac{2}{5} (1) = \mathbf{0.57}$$

Gain of attribute Humidity:

$$\text{Values(humidity)} = [\text{High}, \text{Normal}]$$

$$S = [2 \text{ Yes}, 3 \text{ No}], S_{\text{High}} = [0 - Y, 3 - N], S_{\text{Normal}} = [2 - Y, 0 - N]$$

$$\text{Entropy}(S_{\text{High}}) = 0$$

$$\text{Entropy}(S_{\text{Normal}}) = 0$$

$$\text{Gain}(S, \text{Humidity}) = 0.97 - 0 = \mathbf{0.97}$$

Gain of attribute wind:

$$\text{Values(wind)} = [\text{Weak}, \text{Strong}]$$

$$S = [2 \text{ Yes}, 3 \text{ No}], S_{\text{weak}} = [1 - Yes, 2 - No], S_{\text{strong}} = [1 - Yes, 1 - No]$$

$$\text{Entropy}(S_{\text{weak}}) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} = 0.918$$

$$\text{Entropy}(S_{\text{strong}}) = 1$$

$$\text{Gain}(S, \text{wind}) = 0.97 - \frac{3}{5} (0.918) - \frac{2}{5} (1) = \mathbf{0.0192}$$

Therefore, humidity is selected as the internal node.

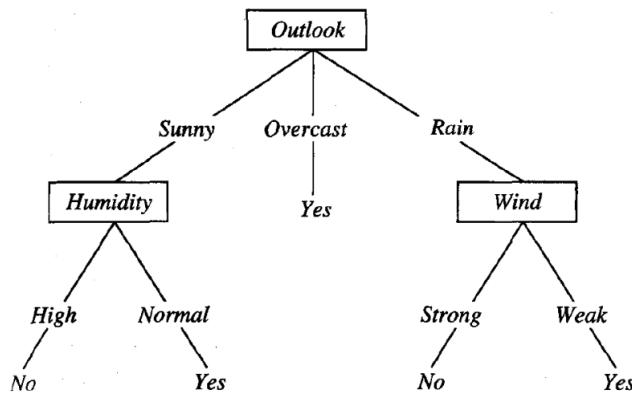
Split the records of overcast = rainy now

Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

From the dataset itself, it is clear that when wind=weak, the play attribute is Yes and when wind is Strong, play attribute is No.

Therefore, the internal nodes is Wind.

The final decision tree is



Question Bank

1. Explain the concept of classification and list down its applications.
2. Write about Logistic function along with its most convenient mathematical properties.
3. Explain how logistic regression can be used for the prediction of binary classes.
4. Explain the one-vs-rest approach of multi-class classification in Logistic Regression
5. Summarize the pros and cons of Logistic Regression
6. Differentiate between Binary and multi-class classification
7. Summarize the evaluation measures used for classification.
8. For confusion matrix shown below, find the evaluation measures accuracy, precision, recall and F1 measure.

n=165	Predicted:		60
	NO	YES	
Actual: NO	TN = 50	FP = 10	
Actual: YES	FN = 5	TP = 100	105
		55	110

9. Suppose 10000 patients get tested for flu; out of them, 9000 are actually healthy and 1000 are actually sick. For the sick people, a test was positive for 620 and negative for 380. For the healthy people, the same test was positive for 180 and negative for 8820. Construct a confusion matrix for the data and compute accuracy, precision, recall and F1 measure for the data.
10. Explain K-Nearest Neighbor learning algorithm
11. Write a short note on the following
 - a. Distance measures used in K-Nearest Neighbors algorithm
 - b. Normalization of data.
12. Summarize the pros and cons of K-Nearest Neighbors algorithm
13. Discuss the Naive Bayes Classifier.
14. Consider the training data in the following table where Play is a class attribute. In the table, the Humidity attribute has values "L" (for low) or "H" (for high), Sunny has values "Y" (for yes) or "N" (for no), Wind has values "S" (for strong) or "W" (for weak), and Play has values "Yes" or "No".

Unit – II Fundamentals of Machine Learning

Humidity	Sunny	Wind	Play
L	N	S	No
H	N	W	Yes
H	Y	S	Yes
H	N	W	Yes
L	Y	S	No

What is class label for the following day (**Humidity=L, Sunny=N, Wind=W**), according to naïve Bayesian classification?

15. The following data set contains factors that determine whether tennis is played or not. Using Naive Bayes classifier, find the play prediction for the day **<Sunny, Cool, High, Strong>**

DAY	OUTLOOK	TEMP	HUMIDITY	WIND	PLAY
Day 1	Sunny	Hot	High	Weak	NO
Day 2	Sunny	Hot	High	Strong	NO
Day 3	Overcast	Hot	High	Weak	YES
Day 4	Rain	Mild	High	Weak	YES
Day 5	Rain	Cool	Normal	Weak	YES
Day 6	Rain	Cool	Normal	Strong	NO
Day 7	Overcast	Cool	Normal	Strong	YES
Day 8	Sunny	Mild	High	Weak	NO
Day 9	Sunny	Cool	Normal	Weak	YES
Day 10	Rain	Mild	Normal	Weak	YES
Day 11	Sunny	Mild	Normal	Strong	YES
Day 12	Overcast	Mild	High	Strong	YES
Day 13	Overcast	Hot	Normal	Weak	YES
Day 14	Rain	Mild	High	Strong	NO

16. Summarize the pros and cons of Naïve Bayes Classification
17. Explain the basic decision tree learning algorithm.
18. What do you mean by gain and entropy? How it is used to build the decision tree?
19. For the following set of training samples, find which attribute can be chosen as the root for decision tree classification.

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

20. Consider the following set of training examples:

Unit – II Fundamentals of Machine Learning

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

- What is the entropy of this collection of training examples with respect to the target function classification?

- What is the information gain of a2 relative to these training examples?

21. Identify the first splitting attribute for Decision Tree using ID3 algorithm with the following dataset.

Major	Experience	Tie	Hired?
CS	programming	pretty	NO
CS	programming	pretty	NO
CS	management	pretty	YES
CS	management	ugly	YES
business	programming	pretty	YES
business	programming	ugly	YES
business	management	pretty	NO
business	management	pretty	NO

22. Explain how Support Vector Machine can be used for classification of linearly separable data.

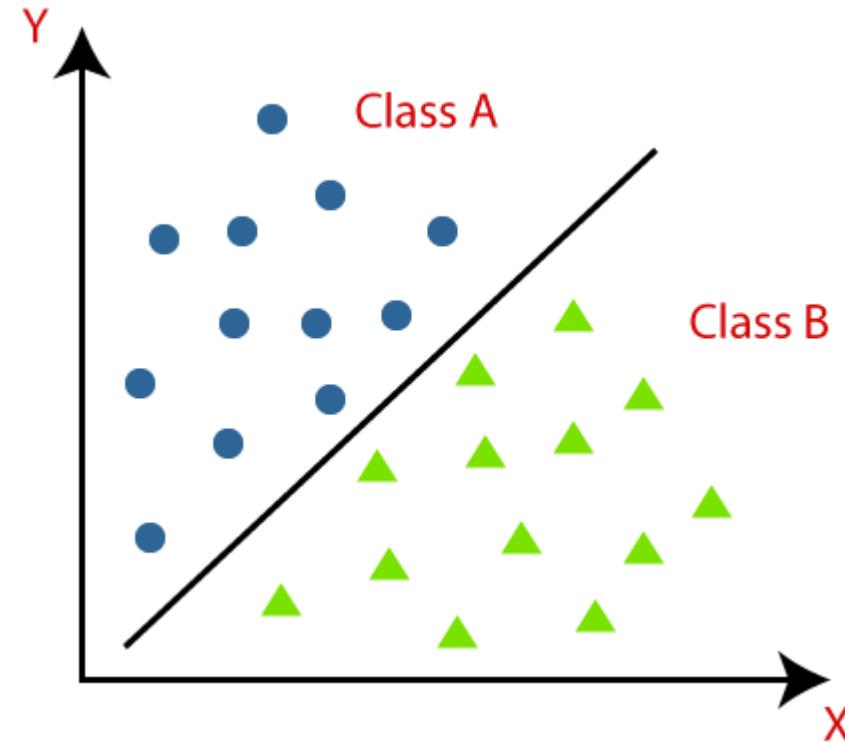


Supervised Learning - Classification

UNIT - II

Topics

- ✓ Basic concepts and applications of classification
- ✓ Naïve Bayes Classification
- ✓ Logistic Regression
- ✓ K-Nearest Neighbors
- ✓ Classification Trees
- ✓ Support Vector Machines
- ✓ Evaluation Measures for Classification Techniques



What is Classification?

- ❖ classification refers to a predictive modeling problem where a class label is predicted for a given example of input data.
- ❖ Task of assigning objects to one of several predefined categories
- ❖ **Def :** Classification is the task of learning a target function f that maps each attribute set X to one of the predefined class labels y .
 - ❖ The target function is known as **classification model**.
- ❖ Examples
 - ❖ Classification of email as spam or ham
 - ❖ Classification of handwritten digits

Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Learning Model

induction

Learn Model

Model

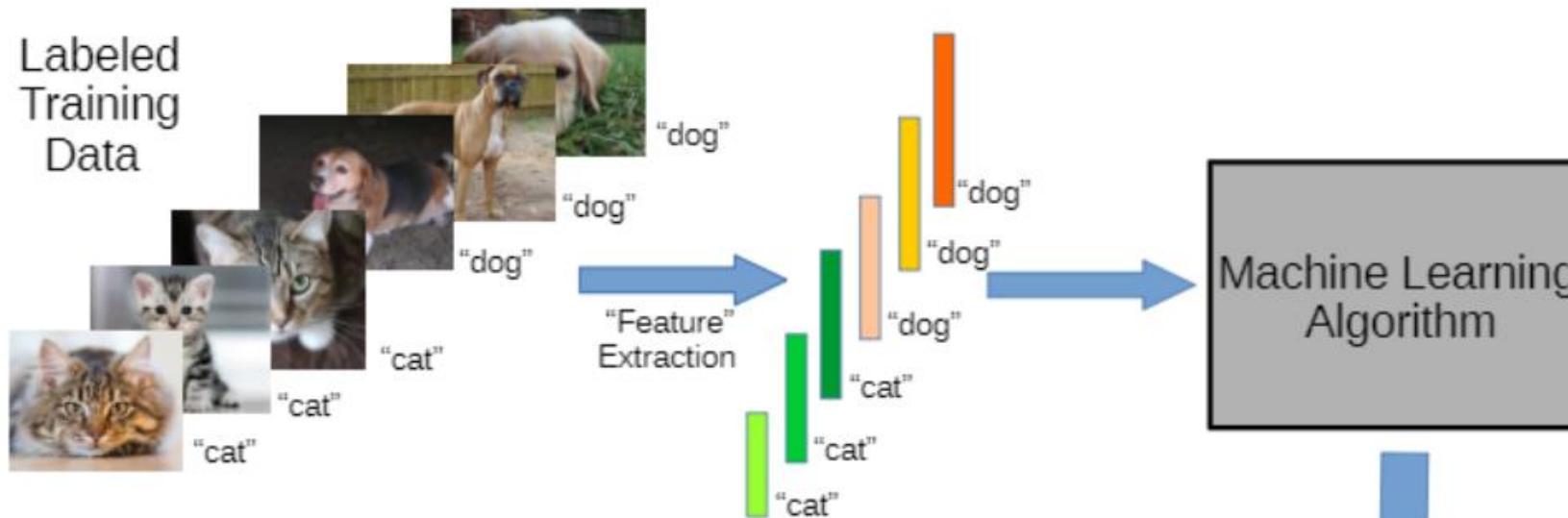
Testing Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

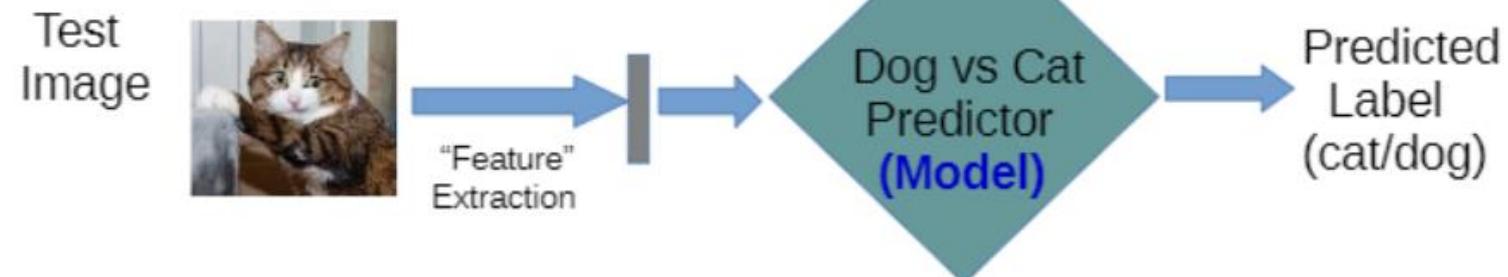
Apply Model

deduction

Typical Classification Problem



Note: The **feature extraction** phase may be part of the machine learning algorithm itself
(referred to "feature learning" or "representation learning")
Modern "**deep learning**" algs do precisely that!



More about classification

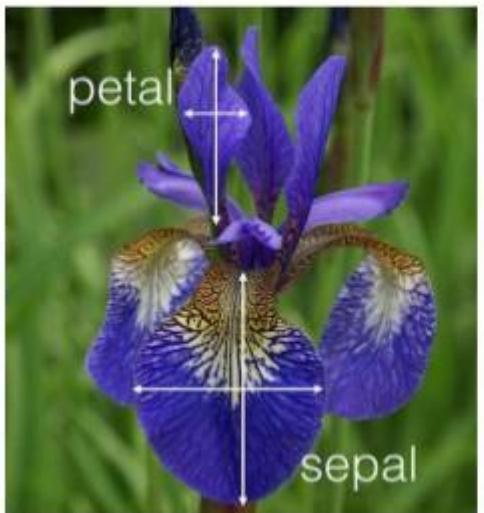
- The input data for classification task is a collection of records.
 - Each record is a tuple (X, y)
 - X is the attribute set, y is the label
 - Class label y must be **discrete**
 - The attribute set can contain both **discrete and continuous**.
- Classification algorithms are most suited for predicting or describing datasets with binary or nominal categories.

Classification Algorithms

- Various algorithms present to do classification are
 - Decision Tree Classifier
 - KNN
 - Neural Networks
 - SVM
 - Logistic Regression
 - Naïve Bayes
- Each technique employs a learning algorithm
- Key objective of these algorithms is **to build models with good generalization capability.**

Sample Datasets

Supervised learning **classification** problem
(using the [Iris flower data set](#))



Training / test data
Features Labels

Sepal length	Sepal width	Petal length	Petal width	Species
5.1	3.5	1.4	0.2	Iris setosa
4.9	3.0	1.4	0.2	Iris setosa
7.0	3.2	4.7	1.4	Iris versicolor
6.4	3.2	4.5	1.5	Iris versicolor
6.3	3.3	6.0	2.5	Iris virginica
5.8	3.3	6.0	2.5	Iris virginica

	Gender	Height	Weight	Index	Status
0	Male	174	96	4	Obesity
1	Male	189	87	2	Normal
2	Female	185	110	4	Obesity
3	Female	195	104	3	Overweight
4	Male	149	61	3	Overweight
5	Male	189	104	3	Overweight
6	Male	147	92	5	Extreme Obesity
7	Male	154	111	5	Extreme Obesity
8	Male	174	90	3	Overweight
9	Female	169	103	4	Obesity

Sample Datasets

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

Sample Datasets

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes
31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Applications of Classification

- ❑ Sentiment Analysis
- ❑ Email Spam Classification
- ❑ Categorizing cells as malignant or benign based on MRI scans
- ❑ Document classification
- ❑ Image classification
- ❑ Image and speech recognition
- ❑ Language Modelling
- ❑ Machine Translation

Linear Models

- ✓ Makes predictions using a linear function of the input features..

- ✓ **What is the general prediction formula for linear regression?**

- ✓ Linear regression finds the parameters which minimizes the mean squared error between y and \hat{y} .

- ✓ **Can we use a linear model for classification?**

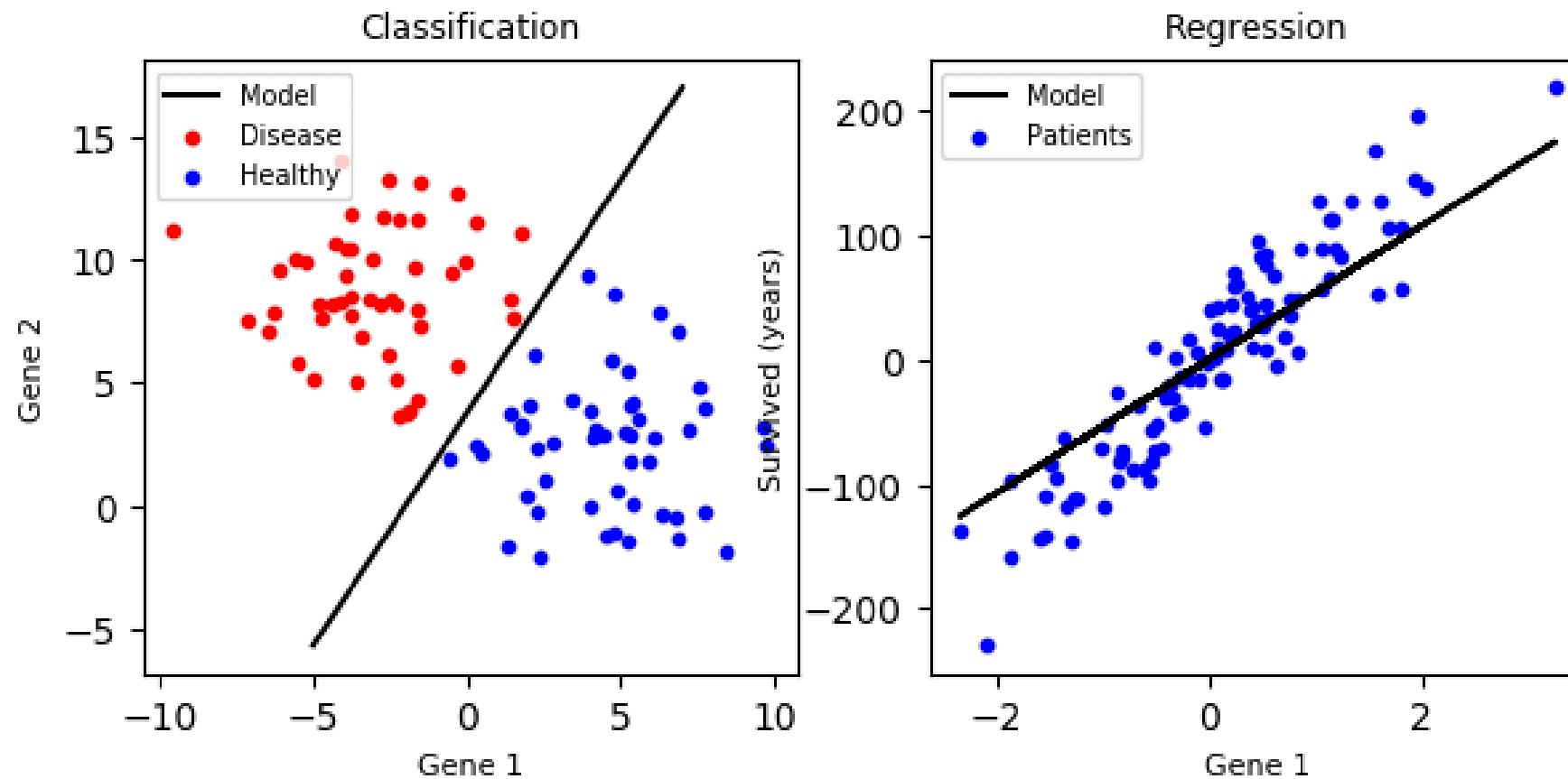
- ✓ linear models can also be used for classification by following

$$\hat{y} = wx + b > 0$$

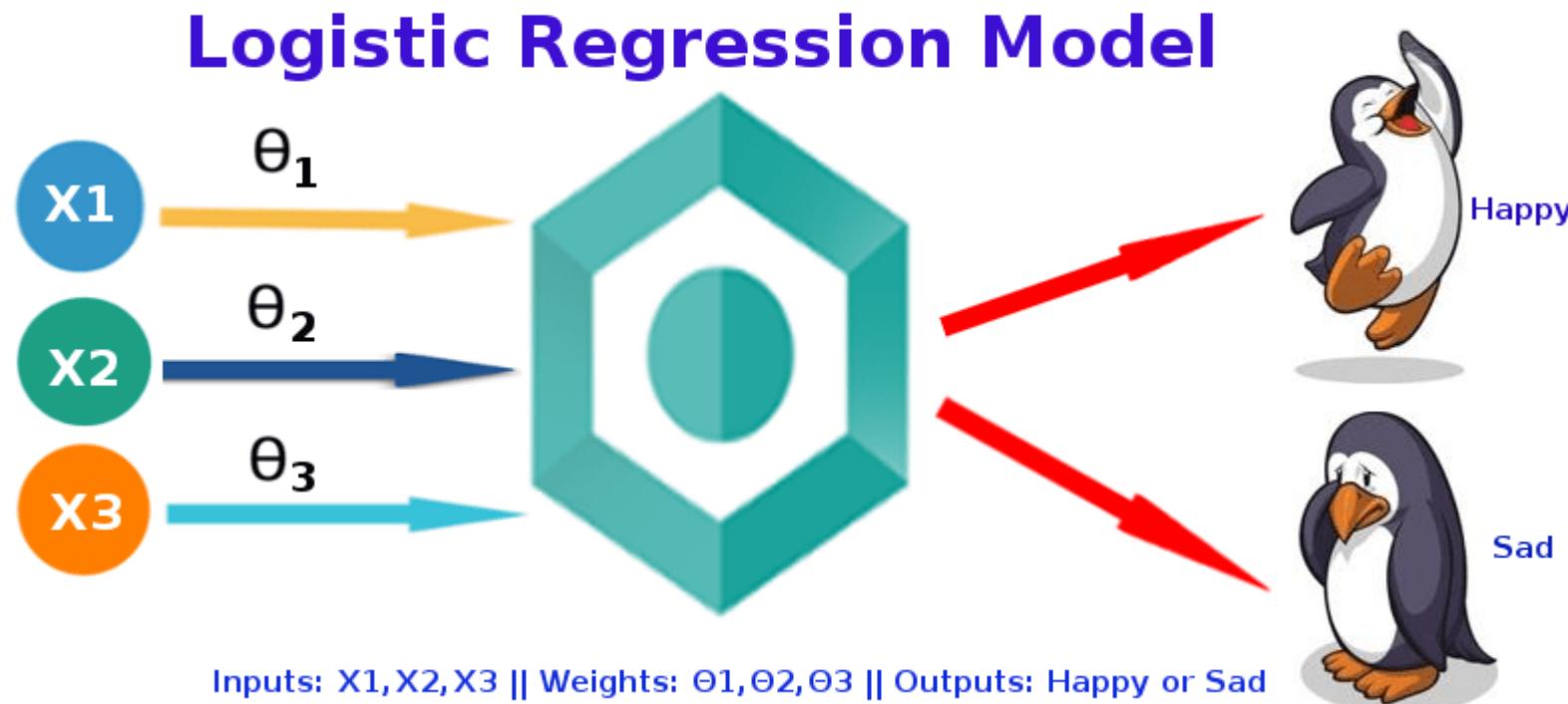
- ✓ Instead of just returning the linear sum, we threshold the predicted value at 0
 - ✓ If $wx + b < 0$ then the predicted class is -1.
 - ✓ If $wx + b > 0$ then the predicted class is 1.

- ✓ The two most common linear classification algorithms are

- ✓ Logistic Regression
 - ✓ Support Vector Machines.

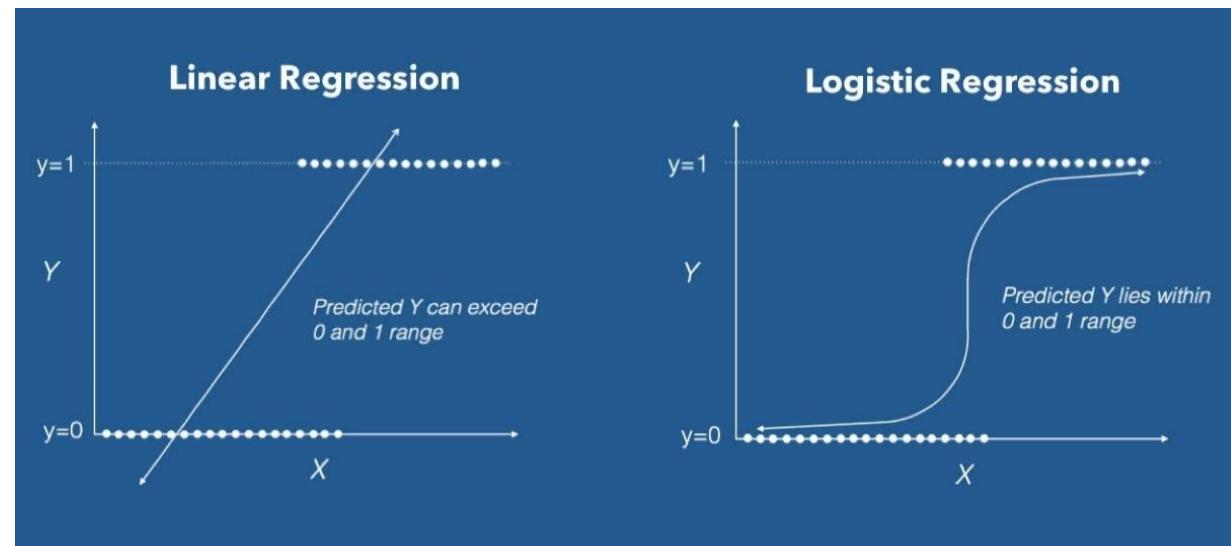


Logistic Regression



Logistic Regression

- A statistical method for predicting binary classes. predicts the probability of occurrence of a binary event utilizing a logit function.
- The target variable is dichotomous in nature.
 - Any examples ...?
- In general, a linear function with threshold creates a classifier. but it causes few problems
 - Outliers leads to misclassifications
 - This classifier announces completely a confident prediction of 1 or 0, even to the examples that are close to the boundary



Logistic Regression

Sigmoid function

To resolve the issues - soften the threshold function

The threshold function is approximated using a continuous, differentiable logistic function (sigmoid)

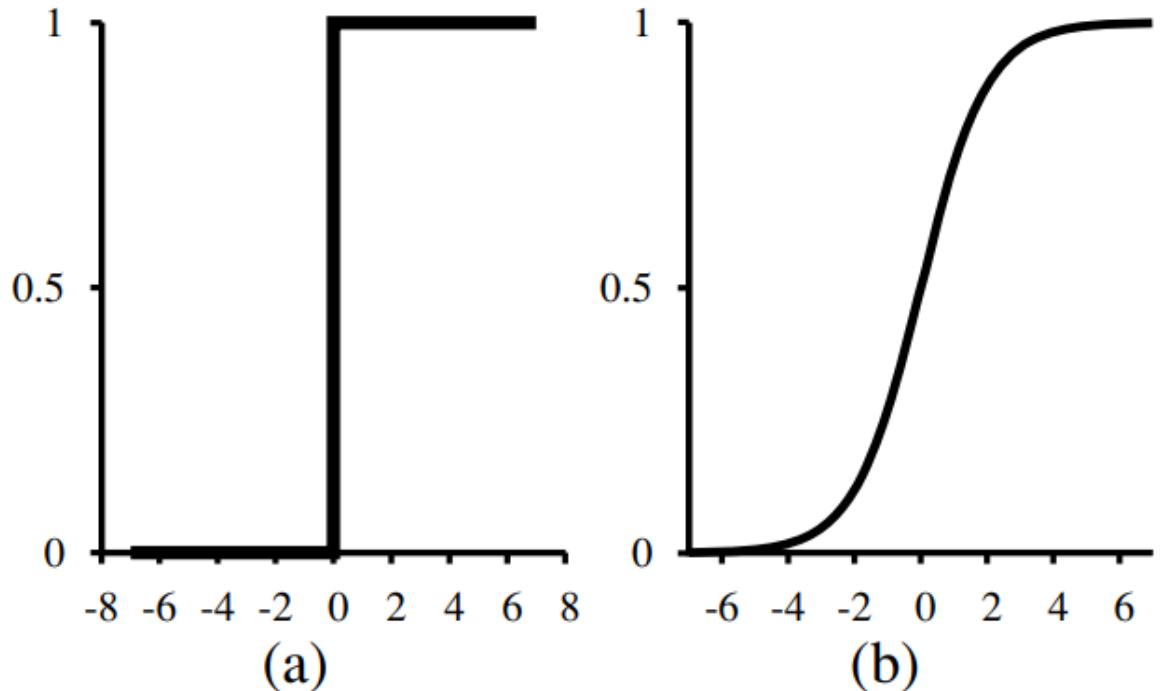


Figure (a) shows the threshold with 0/1 output. It is non-differentiable at $z=0$

Figure (b) shows the logistic function (also called sigmoid)

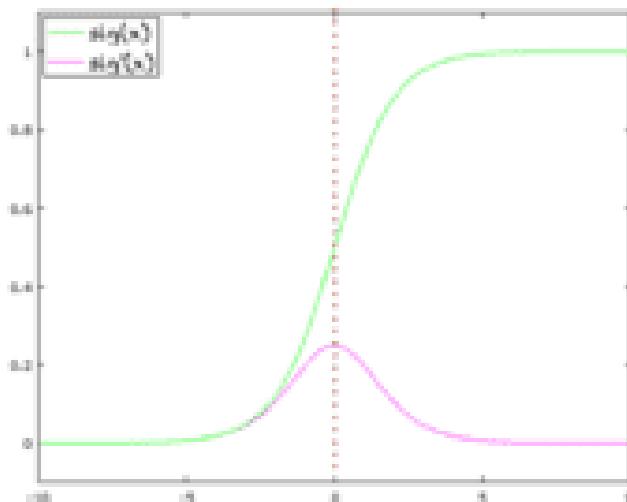
Logistic Regression

Sigmoid

Sigmoid has most convenient mathematical properties

The output is a number between 0 and 1.

It can be interpreted as a probability of a belonging to a class labelled 1.



Plot of $\sigma(x)$ and its derivate $\sigma'(x)$

$$\text{Logistic}(wx) = \frac{1}{1 + e^{-wx}}$$

Domain: $(-\infty, +\infty)$

Range: $(0, +1)$

$$\sigma(0) = 0.5$$

Other properties

$$\sigma(x) = 1 - \sigma(-x)$$

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

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Logistic Regression

The process of fitting the weights of the model to minimize loss on a dataset is called Logistic Regression

Why the name Logistic Regression?

Data is fit into linear regression model, which then be acted upon by a logistic function predicting the target categorical dependent variable.

Logistic Regression

Finding optimal values of W...

For a Single Example (X, y)

$$\begin{aligned}\frac{\partial g(f(x))}{\partial x} &= g^1(f(x)) \cdot \frac{\partial f(x)}{\partial x} \\ \frac{\partial}{\partial w_i} \text{Loss}(W) &= \frac{\partial}{\partial w_i} (y - \hat{y})^2 \\ &= 2(y - \hat{y}) \frac{\partial}{\partial w_i} (y - \hat{y}) \\ &= -2(y - \hat{y}) g^1(w \cdot x) \frac{\partial}{\partial w_i} w \cdot x \\ &= -2(y - \hat{y}) g^1(w \cdot x) x_i\end{aligned}$$

Derivative of the logistic function is

$$\begin{aligned}g^1(z) &= g(z) \cdot (1 - g(z)) \\ g^1(w \cdot x) &= g(w \cdot x) \cdot (1 - g(w \cdot x)) = \hat{y} \cdot (1 - \hat{y})\end{aligned}$$

Weight updates for minimizing the loss

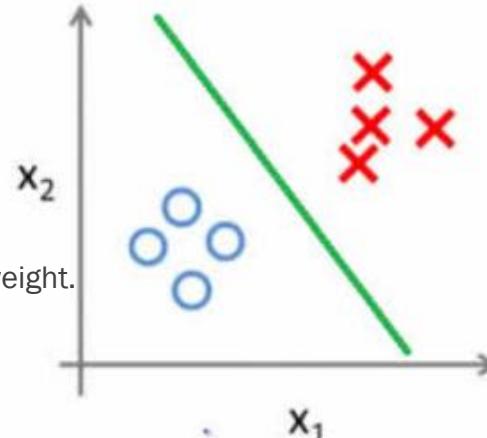
$$\begin{aligned}w_i &\leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \text{Loss}(W) \\ w_i &\leftarrow w_i - \alpha (y - \hat{y}) \hat{y} (1 - \hat{y}) x_i\end{aligned}$$

Binary vs. Multi-class classification

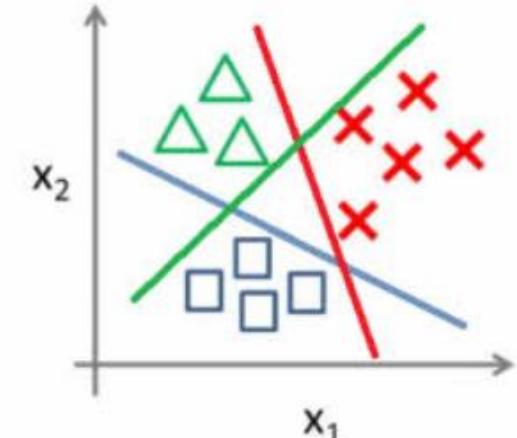
Binary Classification

- Only two class instances are present in the dataset.
- It requires only one classifier model.
- Confusion Matrix is easy to derive and understand.
 - Example:- Check email is spam or not, predicting gender based on height and weight.

Binary classification:



Multi-class classification:



Multi-class Classification

- Multiple class labels are present in the dataset.
- The number of classifier models depends on the classification technique we are applying to.
- One vs. All**: N-class instances then N binary classifier models
- One vs. One**: N-class instances then $N * (N-1)/2$ binary classifier models
- The Confusion matrix is easy to derive but complex to understand.
 - Example:- Check whether the fruit is apple, banana, or orange.

Logistic Regression Multi-class Classification - Training

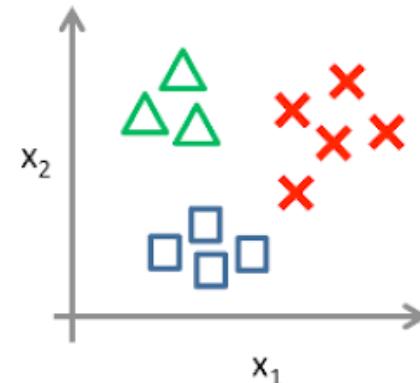
one vs. rest (one vs. all) approach –

a binary model is learned for each class that tries to separate that class from all the other classes.

for the N-class instances dataset, it generates the N-binary classifier models

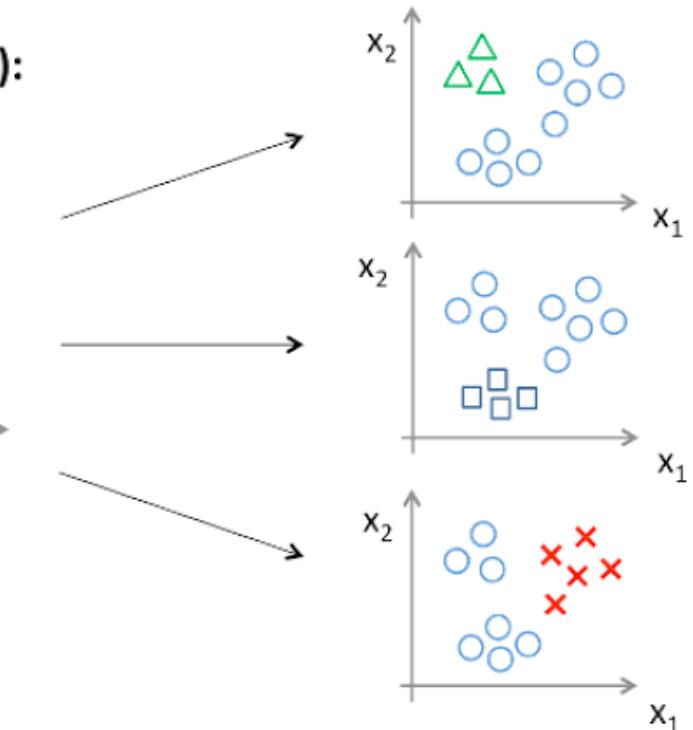
The number of class labels present in the dataset and the number of generated binary classifiers must be the _____

One-vs-all (one-vs-rest):

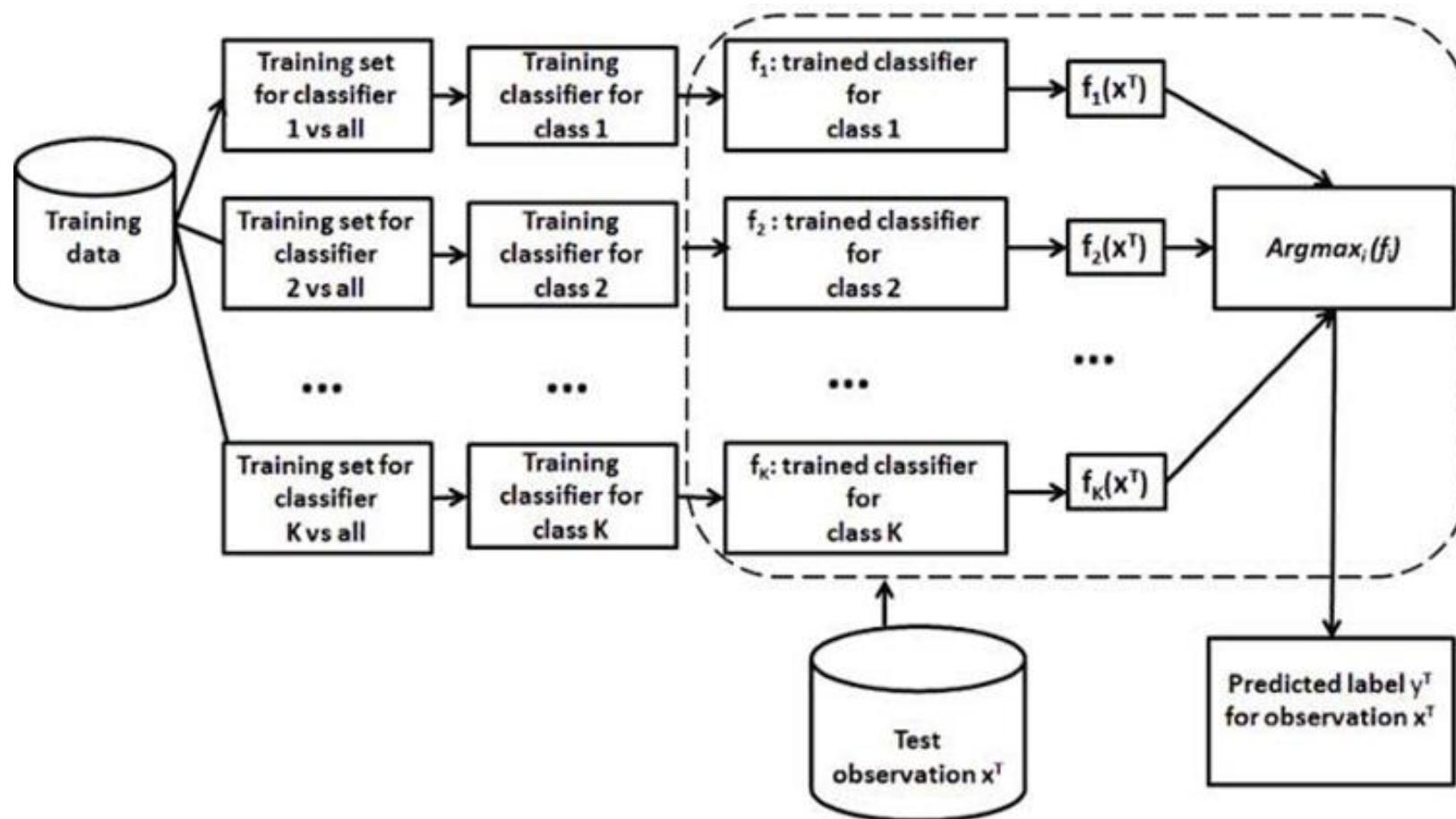


Class 1: Green
Class 2: Blue
Class 3: Red

Classifier 1: [Green] vs [Red, Blue]
Classifier 2: [Blue] vs [Green, Red]
Classifier 3: [Red] vs [Blue, Green]



Logistic Regression Multi-class Classification - Predictions



Logistic Regression Advantages

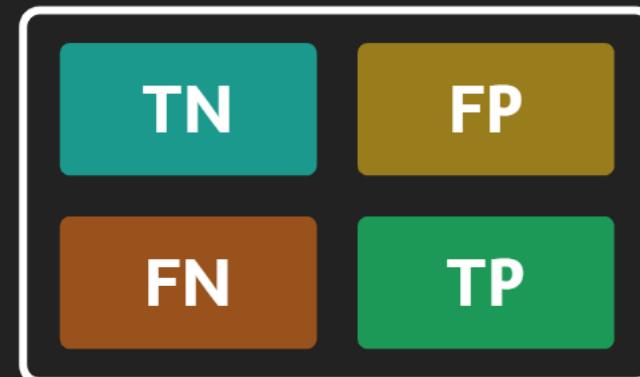
- Logistic regression is easier to implement, interpret, and very efficient to train.
- It makes no assumptions about distributions of classes in feature space.
- It can easily extend to multiple classes
- It is very fast at classifying unknown records.
- Good accuracy for many simple data sets and it performs well when the dataset is linearly separable
- It can interpret model coefficients as indicators of feature importance.
- Logistic regression is less inclined to over-fitting but it can overfit in high dimensional datasets. One may consider Regularization (L1 and L2) techniques to avoid over-fitting in these scenarios.

Logistic Regression Disadvantages

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to overfitting.
- It constructs linear boundaries.
- The major limitation is the assumption of linearity between the dependent variable and the independent variables.
- Non-linear problems can't be solved with logistic regression because it has a linear decision surface. Linearly separable data is rarely found in real-world scenarios.

Confusion Matrix

and Classification Evaluation Metrics



Evaluation measures for classification

What is the need of evaluating a classifier?

How you will evaluate a email spam classifier?

What are positive and negative classes in the above task?

What is a confusion matrix?

- A confusion matrix is a table for visualizing how an algorithm performs with respect to the human gold labels, using two dimensions (system output and gold labels), and each cell labelling a set of possible outcomes.

Ready for few more new words....

- True Positive, True Negative, False Positive, False Negative

Confusion matrix

		Predicted Value	
		Negative	Positive
Actual Value	Negative	TN	FP
	Positive	FN	TP

Guess	→	Fact
This car is NOT red	→	This car is NOT red
This car is red	→	This car is red
This car is NOT red	→	This car is red
This car is red	→	This car is NOT red

TN TP FN FP

TP, TN, FP, FN

In simple words

- A **true positive** is an outcome where the model correctly predicts the positive class. Similarly, a **true negative** is an outcome where the model correctly predicts the negative class.
- A **false positive** is an outcome where the model incorrectly predicts the positive class. And a **false negative** is an outcome where the model incorrectly predicts the negative class.

For example, in the spam detection case example,

- true positives are documents that are indeed spam that our system correctly said were spam.
- False negatives are documents that are indeed spam but our system incorrectly labelled as non-spam

		<i>gold standard labels</i>	
		gold positive	gold negative
<i>system output labels</i>	system positive	true positive	false positive
	system negative	false negative	true negative
		$\text{precision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$	
		$\text{recall} = \frac{\text{tp}}{\text{tp} + \text{fn}}$	$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{fp} + \text{tn} + \text{fn}}$

Accuracy

$$\frac{\text{TP} + \text{TN}}{\text{FP} + \text{TP} + \text{TN} + \text{FN}}$$

Positive Predictive Value
Precision

$$\frac{\text{TP}}{\text{TP} + \text{FP}}$$

Sensitivity or True Positive Rate
Recall

$$\frac{\text{TP}}{\text{TP} + \text{FN}}$$

F1-Score

$$2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Accuracy

- tells what percentage of all the observations our system labelled correctly
 - Not suitable for unbalanced datasets
-

Precision

- measures the percentage of the items that the system detected (i.e., the system labeled as positive)

Recall

- measures the percentage of items actually present in the input that were correctly identified by the system (i.e., that are in fact positive (i.e., are positive according to the human gold labels))

F-measure

- a single metric that incorporates aspects of both Precision and Recall.

$$F_{\beta} = \frac{(\beta^2 + 1)PR}{\beta^2P + R}$$

The β parameter differentially weights the importance of recall and precision, based on the needs of an application.

Exercise

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

For the above confusion matrix, calculate and fill the following.

True positives	
False positives	
False negatives	
True negatives	
Precision	
Recall	
F1-Score	
accuracy	

Exercise

TN	FP
9	1
FN	TP
1000	9000

TN	FP
9000	1000
FN	TP
1	9

Evaluating with more than two classes

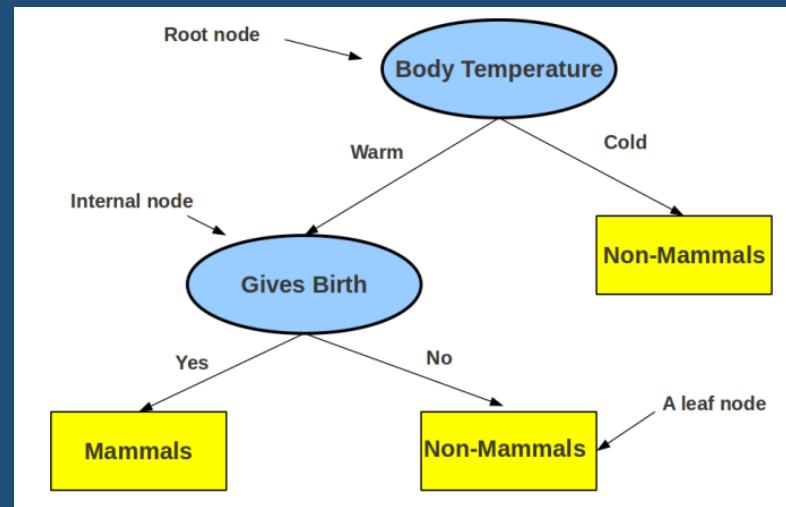
		gold labels		
		urgent	normal	spam
system output	urgent	8	10	1
	normal	5	60	50
	spam	3	30	200
		precision_u = $\frac{8}{8+10+1}$	precision_n = $\frac{60}{5+60+50}$	precision_s = $\frac{200}{3+30+200}$
		recall_u = $\frac{8}{8+5+3}$	recall_n = $\frac{60}{10+60+30}$	recall_s = $\frac{200}{1+50+200}$

Decision Tree Classification

Simple and widely used classification technique

Decision Tree

- Defines a tree-structured hierarchy of rules
- Consists of a root node, internal nodes, and leaf nodes
- Root and internal nodes contain the rules
- Leaf nodes define the predictions
- Decision Tree (DT) learning is about learning such a tree from labeled training data

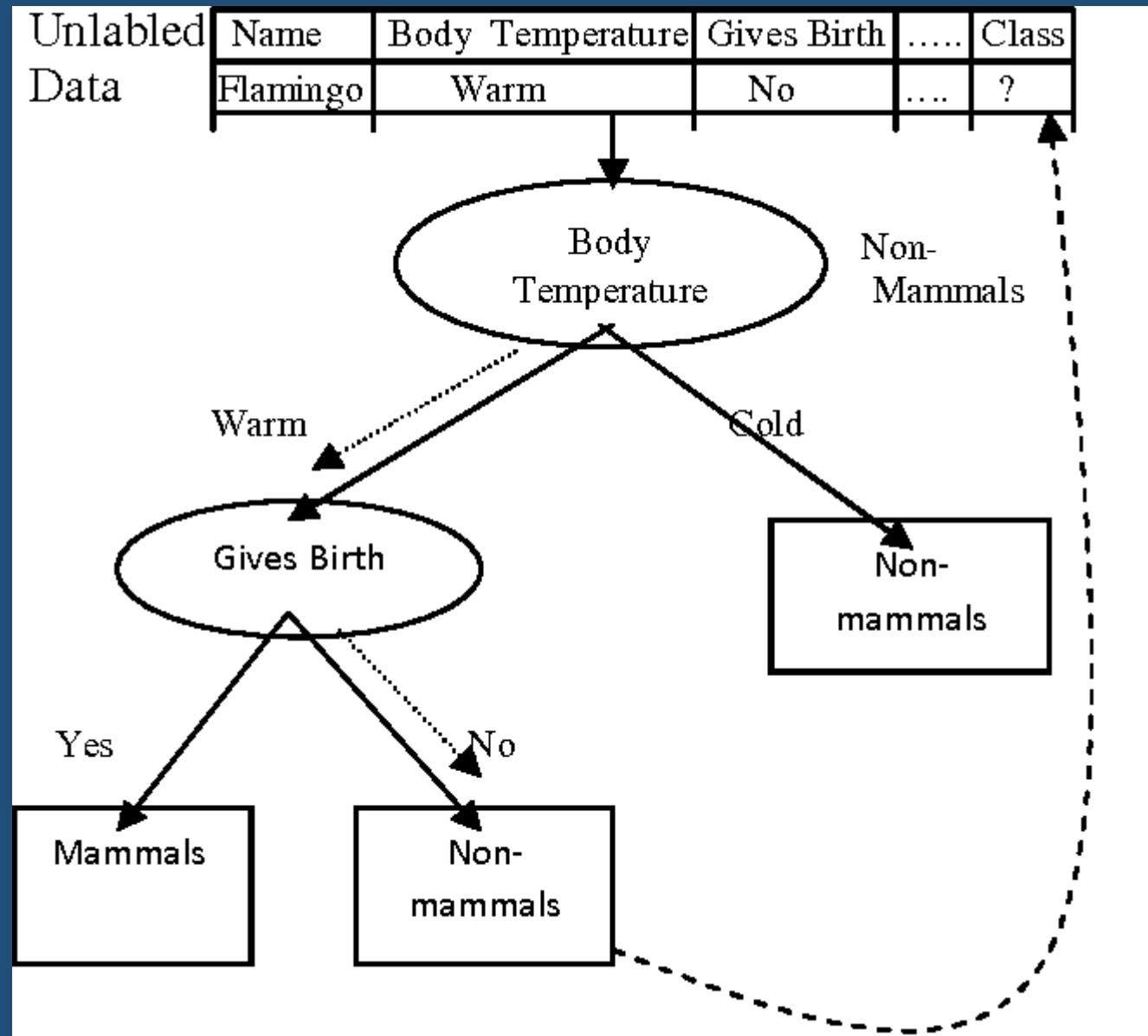


Decision Tree

- The tree has 3 types of nodes
- Root Node
 - Has no incoming edges and 0 or more outgoing edges
- Internal Node
 - Each of which has exactly one incoming edge and 2 or more outgoing edges
- Leaf / Terminal nodes
 - Each of which has exactly one incoming edge, and no outgoing edge.
- Each leaf node is assigned a class label.
- The root node and internal nodes contain attribute test conditions to separate records.
- Classifying a test record becomes simple once a Decision Tree has been constructed.

Decision Tree

Classifying
unlabelled data



Building a Decision Tree

- Hunt's algorithm is the basis for existing Decision Tree algorithms
- Tree grows in a recursive fashion by partitioning the data into purer subsets.
- D_t set of training records associated with node t , and $y = \{y_1, y_2, \dots, y_c\}$ be the class labels.
- Recursive def. of Hunt's algorithm
 - Step 1: If all the records in D_t belong to the same class y_t , then t is a leaf node labelled as y_t .
 - Step 2: If D_t contains records that belong to more than one class, an attribute test condition is selected to partition the records into smaller subsets. A child node is created for each outcome of the test condition and the records in D_t are distributed to the children based on the outcomes. The algorithm is then recursively applied to each child node.

Predicting whether a loan applicant will repay her loan or not.

Defaulted = No

(7,3)

(a)

Home
Owner

No

Defaulted = No

(3,0) Single,
Divorced

Marital
Status

Married

Defaulted = Yes

(1,3)

Defaulted = No

(3,0)

(c)

Home
Owner

Yes

No

Defaulted = No

(3,0)

Defaulted = No

(4,3)

(b)

Home
Owner

No

Defaulted = No

(3,0) Single,
Divorced

Marital
Status

Married

Annual
Income

< 80K

>= 80K

Defaulted = No

(3,0)

Defaulted = No

(1,0)

Defaulted = Yes

(0,3)

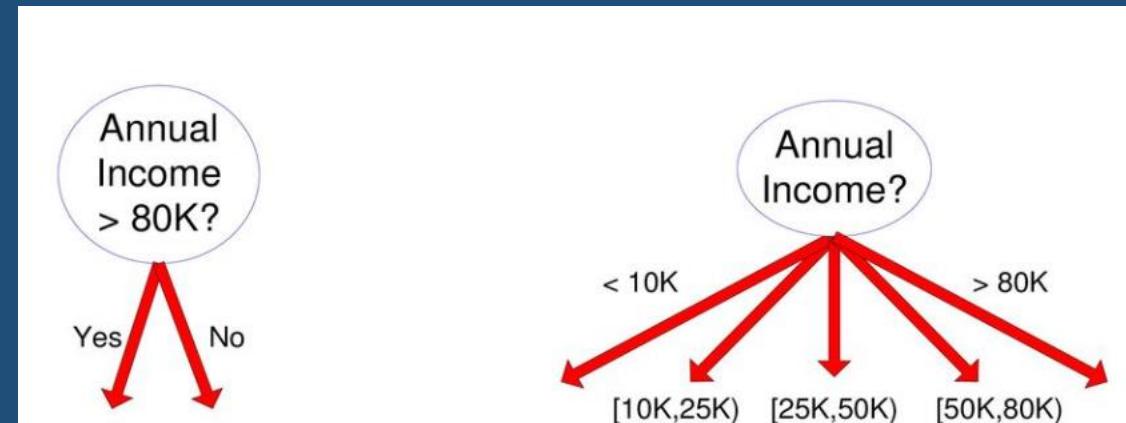
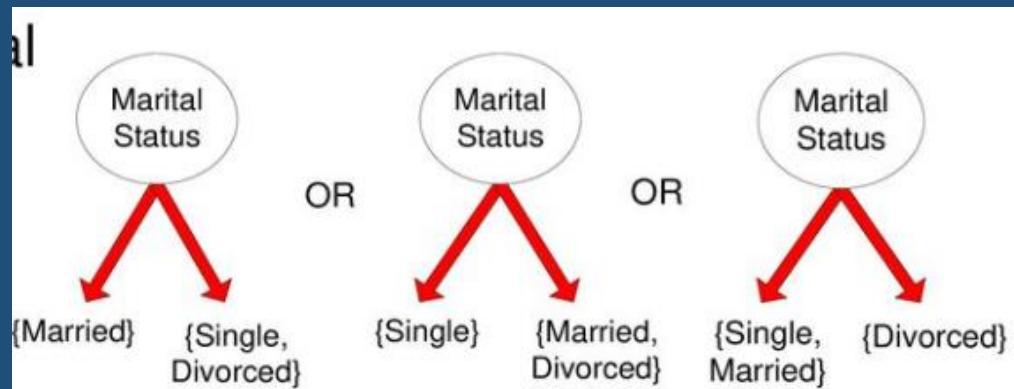
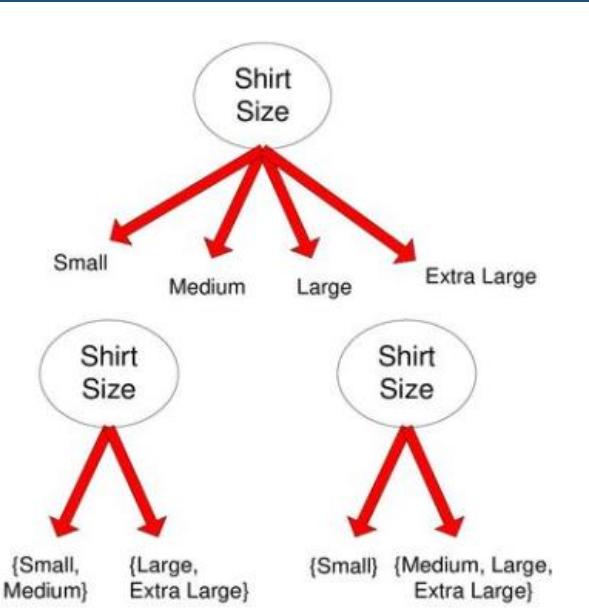
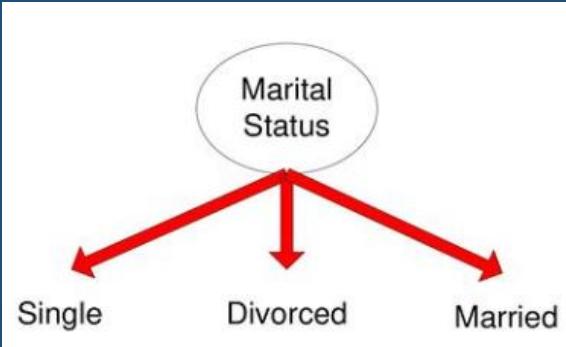
(d)

ID	Home Owner	Marital Status	Annual Income	Defaulted Borrower
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Design issues of Decision Tree induction

- A learning algorithm for inducing decision trees must address the following two issues
 - How should the training records be split?
 - How should the splitting procedure stop?

Methods for expressing attribute test conditions



Measures for selecting the best split

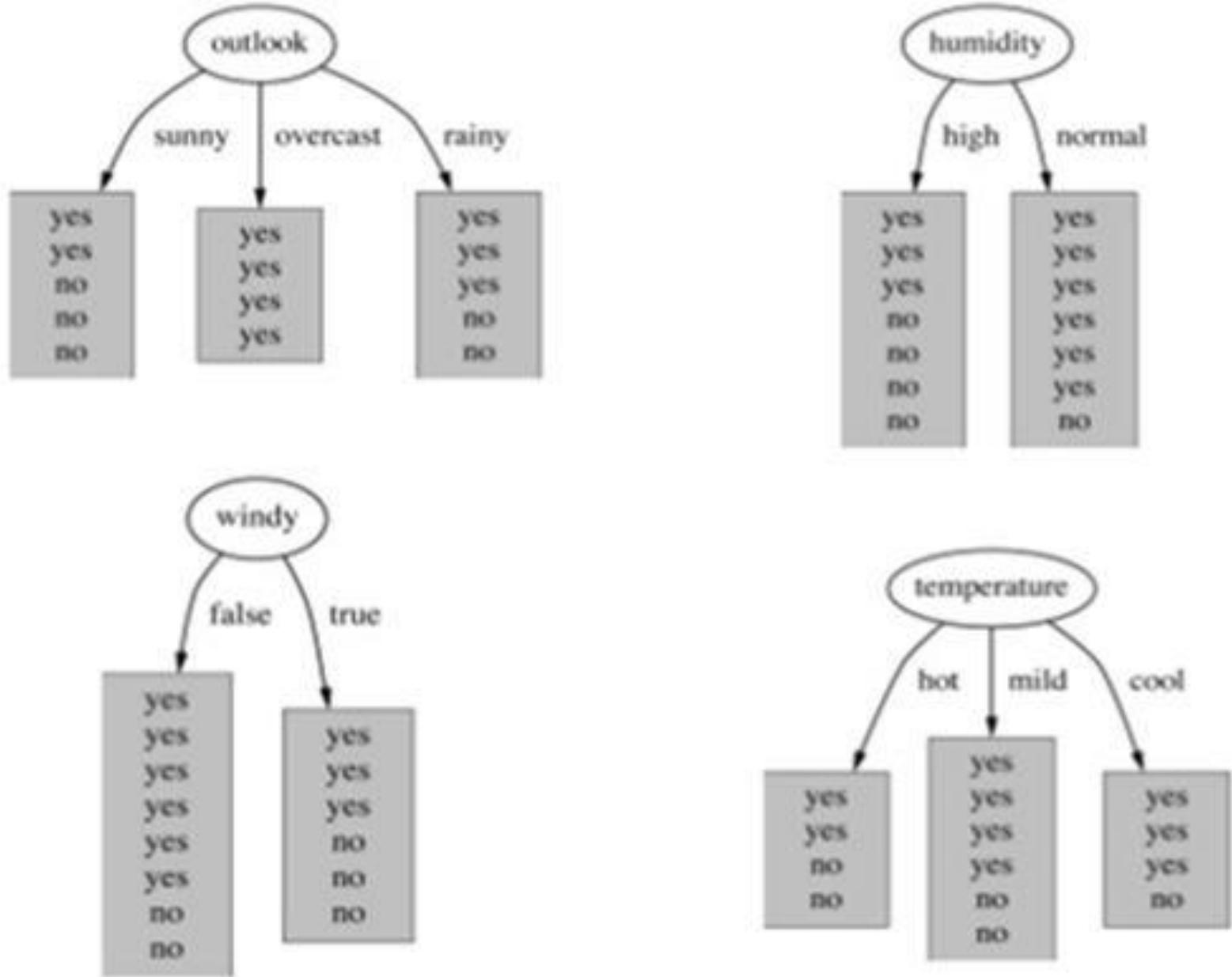
- Let $P(i|t)$ denote the fraction of records belonging to class i at node t .
- The measures developed for selecting the best split are often based on the degree of impurity of the child nodes.
- A node with class distribution $(0,1)$ has zero impurity. Whereas a node with uniform class distribution $(0.5, 0.5)$ has the highest impurity.
- Examples of impurity measures are

$$Entropy(t) = - \sum_{i=0}^{c-1} P(i|t) \log_2 P(i|t)$$

$$Gini(t) = 1 - \sum_{i=0}^{c-1} [P(i|t)]^2$$

$$classification\ error(t) = 1 - \max_i[P(i|t)]$$

- Where c is the no.of classes and $0 \log 20 = 0$ in entropy calculations.



Information Gain

- To determine how well a test condition performs, we need to compare the degree of impurity of the parent node (before splitting) with the impurity of the child nodes (after splitting)
- The larger the difference the better the test condition.
 - The gain, Δ , is a criterion that can be used to determine the goodness of split.

$$\Delta = I(\text{parent}) - \sum_{j=1}^k \frac{N(v_j)}{N} \cdot I(v_j)$$

- Where,
 - $I(\cdot)$ is the impurity measure of a given node,
 - N is the total no. of records at the parent node,
 - k is the number of attribute values, and
 - $N(v_j)$ is the number of records associated with the child node, v_j .

Decision tree induction algorithms often choose a test condition that maximizes the gain.

Example : Construct a Decision tree for the given dataset

Outlook	Temperature	Humidity	Wind	Play Tennis
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

dataset

Find the entropy of the class variable?

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Overcast	Hot	High	Weak	Yes
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Overcast	Cool	Normal	Strong	Yes
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Rain	Mild	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes
Rain	Mild	High	Strong	No

$$Entropy(t) = - \sum_{i=0}^{C-1} P(i|t) \log_2 P(i|t)$$

$$Entropy(S) = -P_+ \log_2 P_+ - P_- \log_2 P_-$$

$$E(S) = -[(9/14)\log(9/14) + (5/14)\log(5/14)] = 0.94$$

Gain (S, A) of an attribute A relative to a collection of examples S is defined as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Let us find out Gain of all the attributes to decide the root node.

Gain of attribute of wind

$$Values(wind) = [Weak, Strong]$$
$$S = [9 \text{ Yes}, 5 \text{ No}], \quad S_{weak} = [6 - \text{Yes}, 2 - \text{No}], \quad S_{strong} = [3 - \text{Yes}, 3 - \text{No}]$$

$$\begin{aligned} Gain(S, wind) &= Entropy(S) - \sum_{v \in [weak, strong]} \frac{|S_v|}{|S|} Entropy(S_v) \\ &= Entropy(S) - \frac{8}{14} Entropy(S_{weak}) - \frac{6}{14} Entropy(S_{strong}) \end{aligned}$$

$$Entropy(S_{weak}) = -\frac{6}{8} \log_2 \frac{6}{8} - \frac{2}{8} \log_2 \frac{2}{8} = 0.811$$

$$Entropy(S_{strong}) = -\frac{3}{6} \log_2 \frac{3}{6} - \frac{3}{6} \log_2 \frac{3}{6} = 1$$

Substitute in Gain(S, wind)

$$Gain(S, wind) = 0.94 - \frac{8}{14} (0.811) - \frac{6}{14} (1) = \mathbf{0.048}$$

Gain of attribute outlook?

$$Values(outlook) = [Sunny, overcast, rainy]$$

$$S = [9 \text{ Yes}, 5 \text{ No}], \quad S_{Sunny} = [2 - Y, 3 - N], \quad S_{overcast} = [4 - Y, 0 - N], \quad S_{rainy} = [3 - Y, 2 - N]$$

$$Gain(S, outlook) = Entropy(S) - \sum_{v \in [sunny, overcast, rainy]} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Entropy(S_{sunny}) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.97$$

$$Entropy(S_{overcast}) = -\frac{4}{4} \log_2 \frac{4}{4} - \frac{0}{4} \log_2 \frac{0}{4} = 0$$

$$Entropy(S_{rainy}) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.97$$

Substitute in Gain(S, outlook)

$$Gain(S, outlook) = 0.94 - \frac{5}{14} (0.97) - \frac{4}{14} (0) - \frac{5}{14} (0.97) = \mathbf{0.246}$$

Gain of attribute Humidity?

$$\begin{aligned} \text{Values(humidity)} &= [\text{High}, \text{Normal}] \\ S &= [9 \text{ Yes}, 5 \text{ No}], \quad S_{\text{High}} = [3 - Y, 4 - N], \quad S_{\text{Normal}} = [6 - Y, 1 - N] \end{aligned}$$

$$Gain(S, \text{Humidity}) = Entropy(S) - \sum_{v \in [\text{high}, \text{normal}]} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Entropy(S_{\text{high}}) = - \frac{3}{7} \log_2 \frac{3}{7} - \frac{4}{7} \log_2 \frac{4}{7} = 0.98$$

$$Entropy(S_{\text{Normal}}) = - \frac{6}{7} \log_2 \frac{6}{7} - \frac{1}{7} \log_2 \frac{1}{7} = 0.59$$

Substitute in Gain(S, Humidity)

$$Gain(S, \text{Humidity}) = 0.94 - \frac{7}{14} (0.98) - \frac{7}{14} (0.59) = \mathbf{0.048}$$

Gain of attribute temperature?

$$Values(temperature) = [Hot, Mild, Cool]$$

$$S = [9 \text{ Yes}, 5 \text{ No}], \quad S_{Hot} = [2 - Y, 2 - N], \quad S_{Mild} = [4 - Y, 2 - N], \quad S_{Cool} = [3 - Y, 1 - N]$$

$$Gain(S, Temperature) = Entropy(S) - \sum_{v \in [Hot, Mild, Cool]} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Entropy(S_{Hot}) = -\frac{2}{4} \log_2 \frac{2}{4} - \frac{2}{4} \log_2 \frac{2}{4} = 1$$

$$Entropy(S_{Mild}) = -\frac{4}{6} \log_2 \frac{4}{6} - \frac{2}{6} \log_2 \frac{2}{6} = 0.918$$

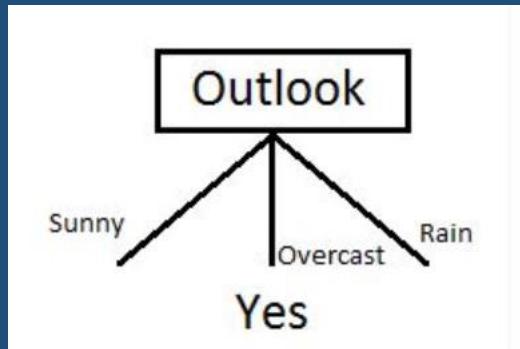
$$Entropy(S_{Cool}) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} = 0.811$$

Substitute in Gain(S, Temperature)

$$Gain(S, Humidity) = 0.94 - \frac{4}{14} (1) - \frac{6}{14} (0.918) - \frac{4}{14} (0.811) = \mathbf{0.029}$$

Which is selected as root node?

The data now...



Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

As all records of outlook=overcast belongs to yes, there won't be any further splitting.

Overcast	Hot	High	Weak	Yes
Overcast	Cool	Normal	Strong	Yes
Overcast	Mild	High	Strong	Yes
Overcast	Hot	Normal	Weak	Yes

Split the records of overcast = sunny now.

Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

S_{sunny} is a collection of 5 examples - 2 positive and 3 negative examples [2+, 3].

$$\text{Entropy}(S_{sunny}) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.97$$

- Now an internal node has to be selected from temperature, humidity and wind.

Gain of attribute Temperature:

$$\text{Values(temperature)} = [\text{Hot}, \text{Mild}, \text{Cool}]$$

$$S = [2 \text{ Yes}, 3 \text{ No}], S_{\text{Hot}} = [0 - Y, 2 - N], S_{\text{Mild}} = [1 - Y, 1 - N], S_{\text{Cool}} = [1 - Y, 0 - N]$$

$$\text{Entropy}(S_{\text{Hot}}) = 0$$

$$\text{Entropy}(S_{\text{Mild}}) = 1$$

$$\text{Entropy}(S_{\text{Cool}}) = 0$$

$$\text{Gain}(S, \text{Humidity}) = 0.97 - \frac{2}{5} (1) = \mathbf{0.57}$$

Gain of attribute Humidity:

$$\begin{aligned}Values(\text{humidity}) &= [\text{High}, \text{Normal}] \\S &= [2 \text{ Yes}, 3 \text{ No}], \quad S_{\text{High}} = [0 - Y, 3 - N], \quad S_{\text{Normal}} = [2 - Y, 0 - N] \\Entropy(S_{\text{High}}) &= 0 \\Entropy(S_{\text{Normal}}) &= 0 \\Gain(S, \text{Humidity}) &= 0.97 - 0 = \mathbf{0.97}\end{aligned}$$

Gain of attribute wind:

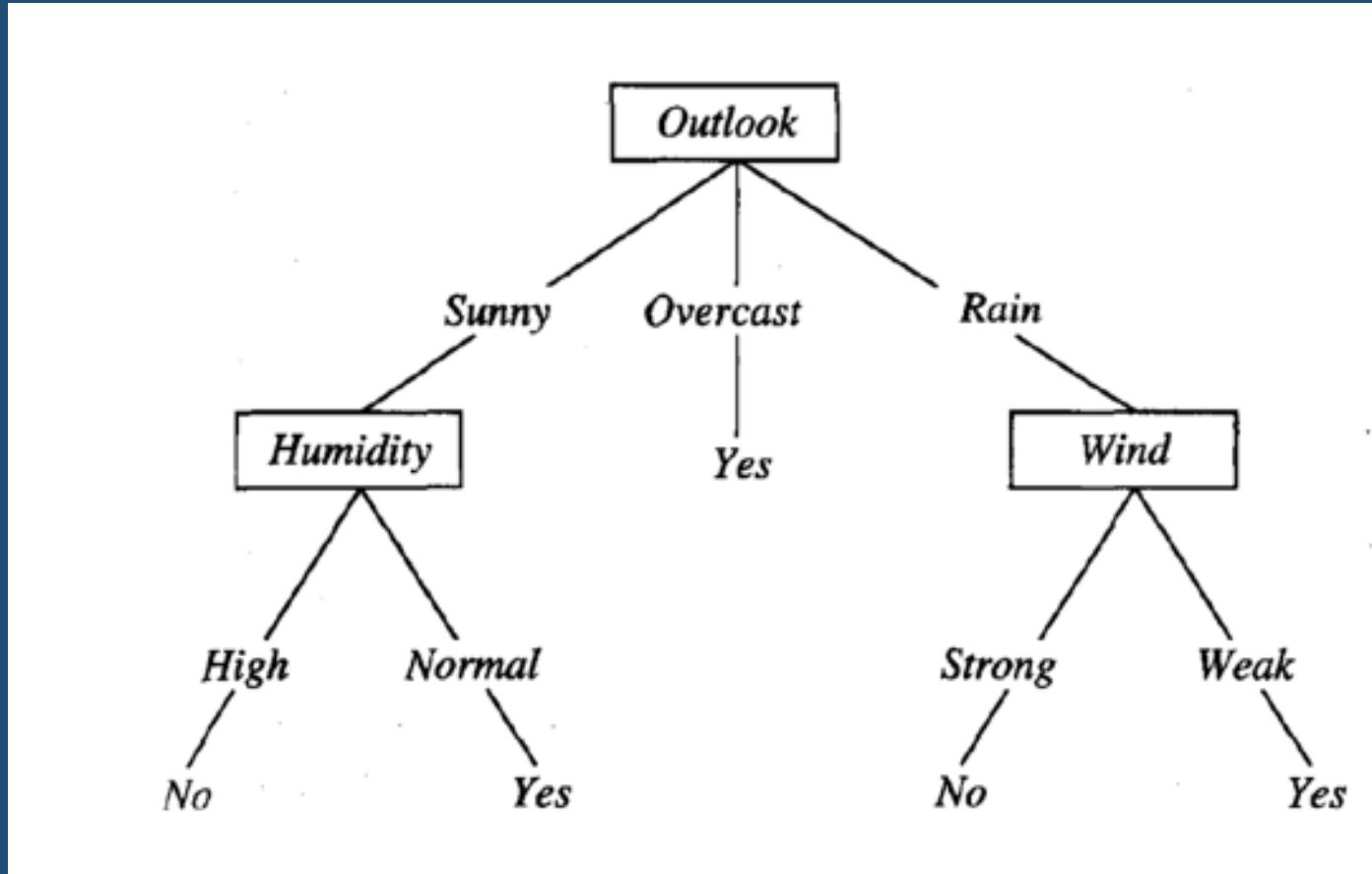
$$\begin{aligned}Values(\text{wind}) &= [\text{Weak}, \text{Strong}] \\S &= [2 \text{ Yes}, 3 \text{ No}], \quad S_{\text{weak}} = [1 - \text{Yes}, 2 - \text{No}], \quad S_{\text{strong}} = [1 - \text{Yes}, 1 - \text{No}] \\Entropy(S_{\text{weak}}) &= -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3} = 0.918 \\Entropy(S_{\text{strong}}) &= 1 \\Gain(S, \text{wind}) &= 0.97 - \frac{3}{5} (0.918) - \frac{2}{5} (1) = \mathbf{0.0192}\end{aligned}$$

Therefore, humidity is selected as the internal node.

Split the records of overcast = rainy now

Rain	Mild	High	Weak	Yes
Rain	Cool	Normal	Weak	Yes
Rain	Cool	Normal	Strong	No
Rain	Mild	Normal	Weak	Yes
Rain	Mild	High	Strong	No

Final decision tree



For the following set of training samples, find which attribute can be chosen as the root for decision tree classification.

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

- a. What is the entropy of this collection of training examples with respect to the target function classification?
- b. What is the information gain of a2 relative to these training examples?

Finding next node under sunny

- We have to determine which of the following Temperature, Humidity or Wind has higher information gain.

$$E(\text{sunny, Temperature}) = (2/5)*E(0,2) + (2/5)*E(1,1) + (1/5)*E(1,0)=2/5=0.4$$

Now calculate information gain.

$$IG(\text{sunny, Temperature}) = 0.971 - 0.4 = 0.571$$

Calculate parent entropy $E(\text{sunny})$

$$E(\text{sunny}) = -(3/5)\log(3/5)-(2/5)\log(2/5)) = 0.971.$$

Similarly we get

$$IG(\text{sunny, Humidity}) = 0.971$$

$$IG(\text{sunny, Windy}) = 0.020$$

		play		
		yes	no	total
Temperature	hot	0	2	2
	cool	1	1	2
	mild	1	0	1
				5

Outlook	Temperature	Humidity	Wind	Played football(yes/no)
Sunny	Hot	High	Weak	No
Sunny	Hot	High	Strong	No
Sunny	Mild	High	Weak	No
Sunny	Cool	Normal	Weak	Yes
Sunny	Mild	Normal	Strong	Yes

K-MEANS

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible.

Algorithm for Basic K-Means algorithm

Select **K** points as initial centroids

Repeat

 Form K clusters by assigning each point to its closest centroid.

 Recompute the centroid of each cluster

Until Centroids do not change

Example:

Cluster the following eight points (with (x, y) representing locations) into three clusters (let us say, we need 3 clusters):

A1(2, 10),

A2(2, 5),

A3(8, 4),

A4(5, 8),

A5(7, 5),

A6(6, 4),

A7(1, 2),

A8(4, 9)

Let, Initial cluster centers are:

A1(2, 10),

A4(5, 8) and

A7(1, 2).

The distance function between two points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is defined as-

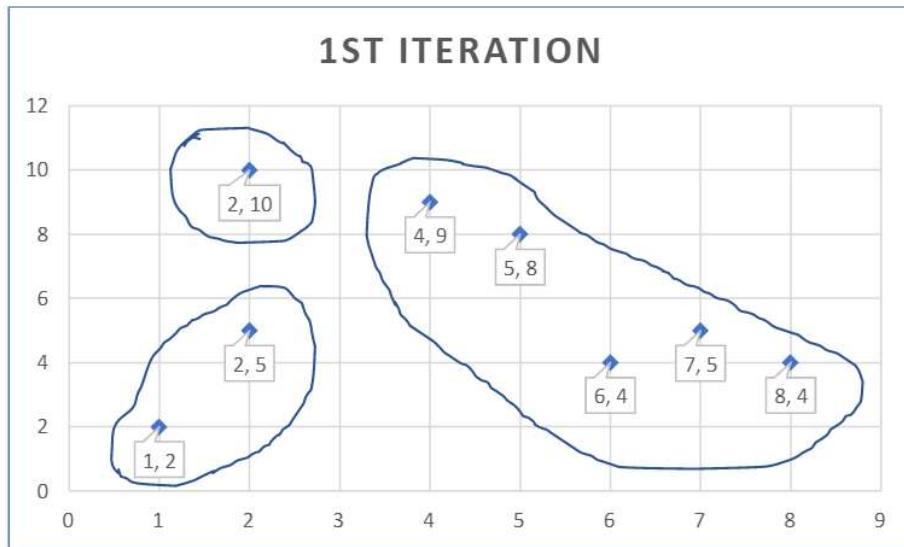
$$P(a, b) = |x_2 - x_1| + |y_2 - y_1|$$

Note: You can use any other distance formula like Euclidean distance too.

Now, calculate the distance from each point to the randomly picked centroids. And make the cluster to which the points are **nearer**.

1st Iteration

Given Points	Distance from centre (2, 10) of Cluster-01	Distance from centre (5, 8) of Cluster-02	Distance from centre (1, 2) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	5	9	C1
A2(2, 5)	5	6	4	C3
A3(8, 4)	12	7	9	C2
A4(5, 8)	5	0	10	C2
A5(7, 5)	10	5	9	C2
A6(6, 4)	10	5	7	C2
A7(1, 2)	9	10	0	C3
A8(4, 9)	3	2	10	C2



For Cluster-01:

We have only one point A1(**2, 10**) in Cluster-01.
So, cluster center remains the same.

For Cluster-02:

Centre of Cluster-02
 $= ((8 + 5 + 7 + 6 + 4)/5, (4 + 8 + 5 + 4 + 9)/5)$
 $= (\mathbf{6, 6})$

For Cluster-03:

Centre of Cluster-03
 $= ((2 + 1)/2, (5 + 2)/2)$
 $= (\mathbf{1.5, 3.5})$

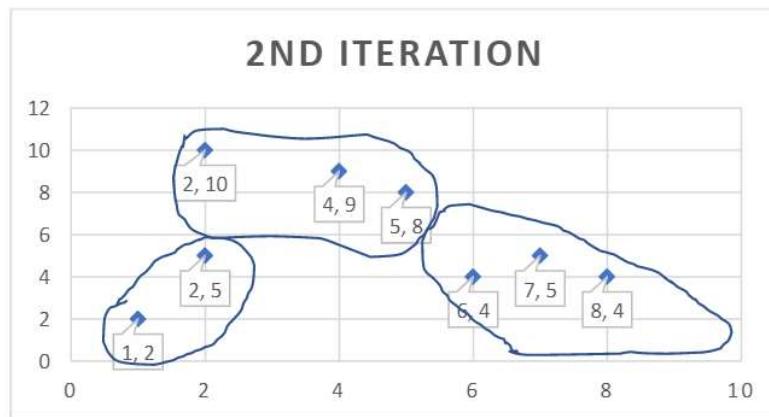
This is completion of Iteration-01.

2nd Iteration

- We calculate the distance of each point from each of the center of the three clusters **(2, 10), (6, 6), (1.5, 3.5)**
- The distance is calculated by using the given distance function.
-

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (6, 6) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	8	7	C1
A2(2, 5)	5	5	2	C3
A3(8, 4)	12	4	7	C2
A4(5, 8)	5	3	8	C2
A5(7, 5)	10	2	7	C2
A6(6, 4)	10	2	5	C2
A7(1, 2)	9	9	2	C3
A8(4, 9)	3	5	8	C1

Note that A8(4,9) has jumped from cluster c2 to cluster c1.
So, we should do another iteration.



For Cluster-01:

Center of Cluster-01

$$= ((2 + 4)/2, (10 + 9)/2) = (3, 9.5)$$

For Cluster-02:

Center of Cluster-02

$$= ((8 + 5 + 7 + 6)/4, (4 + 8 + 5 + 4)/4) = (6.5, 5.25)$$

For Cluster-03:

Center of Cluster-03

$$= ((2 + 1)/2, (5 + 2)/2) = (1.5, 3.5)$$

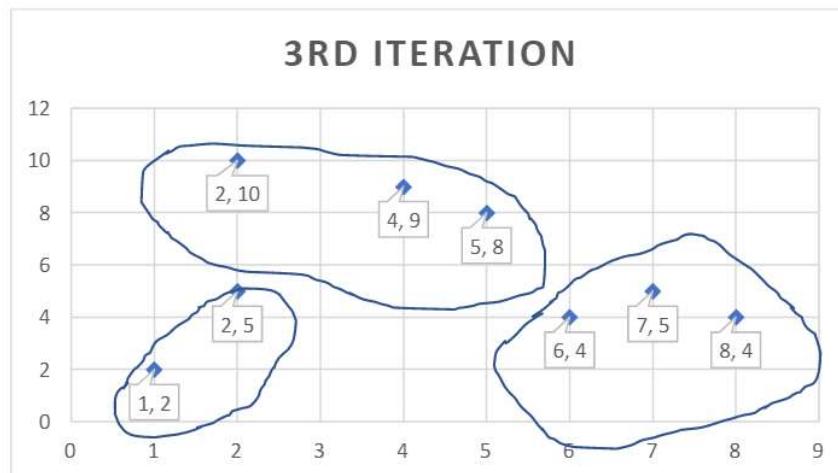
This is completion of Iteration-02.

3rd iteration,

- We calculate the distance of each point from each of the centre of the three clusters **(3, 9.5), (6.5, 5.25), (1.5, 3.5)**
- The distance is calculated by using the given distance function.

Given Points	Distance from center (3, 9.5) of Cluster-01	Distance from center (6.5, 5.25) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	1.5	9.25	7	C1
A2(2, 5)	5.5	4.75	2	C3
A3(8, 4)	10.5	2.75	7	C2
A4(5, 8)	3.5	4.25	8	C1
A5(7, 5)	8.5	0.75	7	C2
A6(6, 4)	8.5	1.75	5	C2
A7(1, 2)	9.5	8.75	2	C3
A8(4, 9)	1.5	6.25	8	C1

Note that cluster A4(5, 8) has jumped from C2 to C1. So, we should do another iteration.



Repeat the process until no element jumps from one cluster to another cluster,

Strengths, Weaknesses, Time and space complexity of K-Means

Strengths

1. It is simple and useful for all varieties of data types.

2. Even though multiple iterations are performed, it is quite efficient.
3. Bisecting k- means is also efficient.

Weakness

1. It cannot handle non-globular clusters, clusters of different sizes and densities even though it can find pure sub-clusters.
2. Outliers cannot be clustered.
3. K-means is restricted for the notion of centroid.

Time complexity

$$O(I^*K^*m^*n)$$

Where,

- I- Number of Iteration for making perfect clusters
K- Number of clusters
m- Number of attributes
n- Number of elements

Space complexity

$$O((m+K)n)$$

Hierarchical clustering

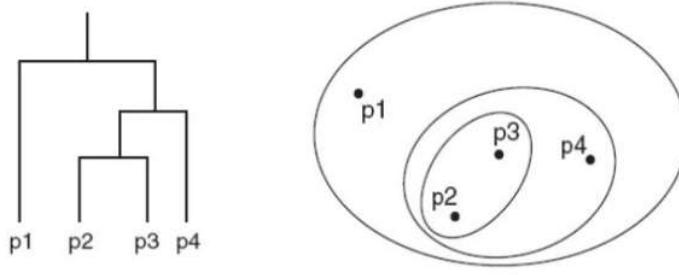
In this type of clustering, the set of clusters are nested clusters that are organized in the form of a tree known as dendrogram. Each node in the tree is the union of its children and root of the tree is the cluster containing all the objects.

There are two types of hierarchical clustering. They are:

1. Agglomerative hierarchical clustering
2. Divisive hierarchical clustering

Agglomerative hierarchical clustering

Start with the points as individual clusters and at each step, merge the closest pair of clusters.



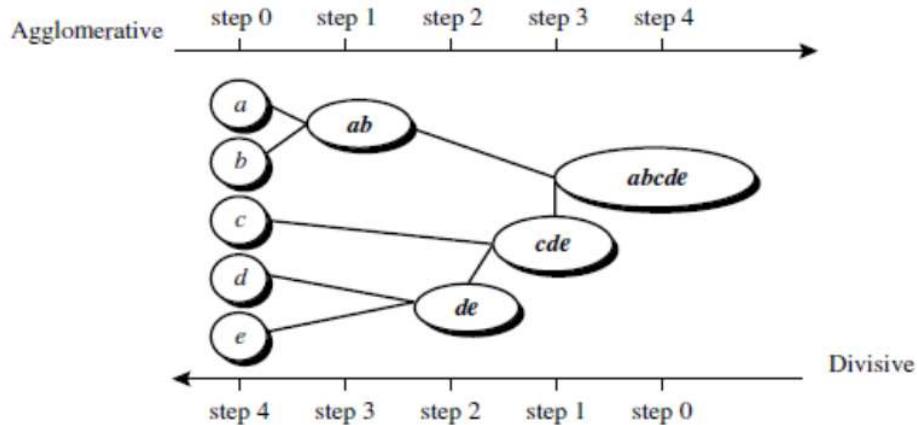
(a) Dendrogram.

(b) Nested cluster diagram.

A hierarchical clustering of four points shown as a dendrogram and as nested clusters.

Divisive hierarchical clustering

Start with one i.e.; group all data objects into a single cluster, at each step, split a cluster until only single cluster of individual points remains.



Basic agglomerative hierarchical clustering algorithm

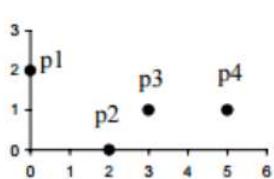
Starting with initial points as clusters, successively merge the two closest clusters until only one cluster remains.

Algorithm 8.3 Basic agglomerative hierarchical clustering algorithm.

- 1: Compute the proximity matrix, if necessary.
 - 2: **repeat**
 - 3: Merge the closest two clusters.
 - 4: Update the proximity matrix to reflect the proximity between the new cluster and the original clusters.
 - 5: **until** Only one cluster remains.
-

cluster proximity

Cluster proximity is defined as similarity or dissimilarity between elements or clusters. They are generally defined by proximity matrix.



point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

	p1	p2	p3	p4
p1	0.000	2.828	3.162	5.099
p2	2.828	0.000	1.414	3.162
p3	3.162	1.414	0.000	2.000
p4	5.099	3.162	2.000	0.000

Four points and their corresponding data and proximity (distance) matrices.

Proximity matrix is a matrix containing distance between elements.

Methods for defining the proximity between the clusters

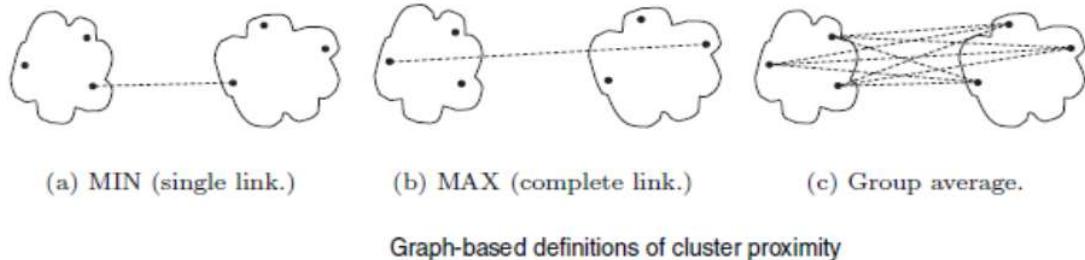
Proximity between clusters can be defined in three ways. They are:

1. Max
2. Min
3. Group average

Min defines cluster proximity between the closest two points that are in different clusters. This type of technique is also known as **single link technique**.

Max defines cluster proximity between the farthest two points that are in different clusters. This type of technique is also known as **complete link technique**.

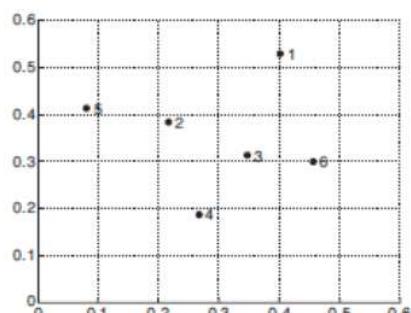
Group average defines cluster proximity to be the average proximities of all pairs of points from different clusters.



Proximity between two clusters using MIN technique (Or)

Single- Link hierarchical clustering

Let us consider a data set with 6 data points.



Set of 6 two-dimensional points.

Point	x Coordinate	y Coordinate
p1	0.40	0.53
p2	0.22	0.38
p3	0.35	0.32
p4	0.26	0.19
p5	0.08	0.41
p6	0.45	0.30

xy coordinates of 6 points.

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Euclidean distance matrix for 6 points.

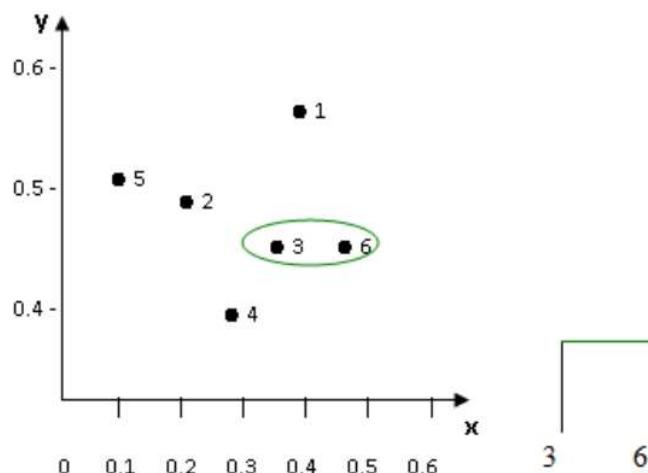
Min or single link technique

The proximity of two clusters is defined as the minimum of the distance between any two points in the two different clusters. The single link technique is good at handling non-elliptical shapes, but is sensitive to noise and outliers.

	P1	P2	P3	P4	P5	P6
P1	0	0.24	0.22	0.37	0.34	0.23
P2	0.24	0	0.15	0.20	0.14	0.25
P3	0.22	0.15	0	0.15	0.28	0.11
P4	0.37	0.20	0.15	0	0.29	0.22
P5	0.34	0.14	0.28	0.29	0	0.39
P6	0.23	0.25	0.11	0.22	0.39	0

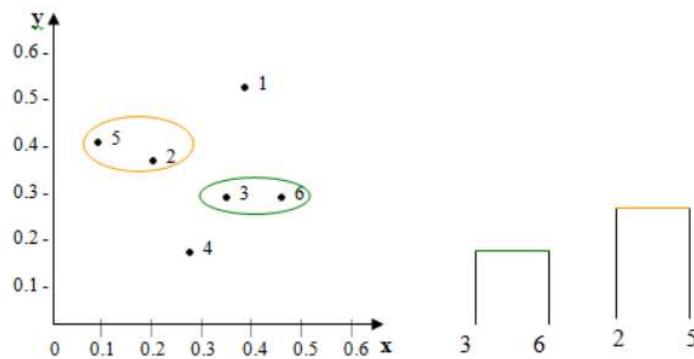
In the above table, p3, p6 is having the lowest distance, so merge the data points into a single cluster.

	P1	P2	P3P6	P4	P5
P1	0	0.24	0.22	0.37	0.34
P2	0.24	0	0.15	0.20	0.14
P3P6	0.22	0.15	0	0.15	0.28
P4	0.37	0.20	0.15	0	0.29
P5	0.34	0.14	0.28	0.29	0



The next lowest distance is for P2P5, so merge those two data points into a single cluster. The matrix obtains as follows:

	P1	P2P5	P3P6	P4
P1	0	0.24	0.22	0.37
P2P5	0.24	0	0.15	0.20
P3P6	0.22	0.15	0	0.15
P4	0.37	0.20	0.15	0



The distance between (p3, p6) and (p2, p5) would be calculated as follows:

$$dist((p3, p6), (p2, p5)) = \min(dist(p3, p2), dist(p6, p2), dist(p3, p5),$$

$$dist(p6, p5))$$

$$= \min(0.15, 0.25, 0.28, 0.39)$$

$$= 0.15$$

$$dist((p3, p6), (p1)) = \min(dist(p3, p1), dist(p6, p1))$$

$$= \min(0.22, 0.23)$$

$$= 0.22$$

$$dist((p3, p6), (p4)) = \min(dist(p3, p4), dist(p6, p4))$$

$$= \min(0.15, 0.22)$$

$$= 0.15$$

$$dist((p2, p5), (p1)) = \min(dist(p2, p1), dist(p5, p1))$$

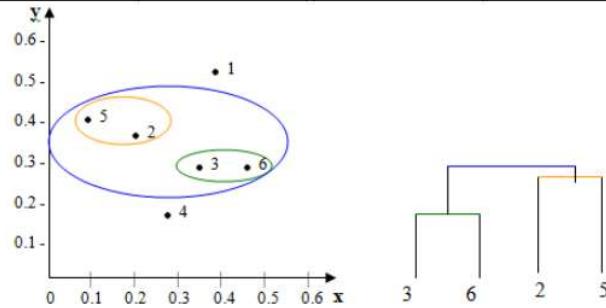
$$= \min(0.24, 0.34)$$

$$=0.24$$

$$\begin{aligned} \text{dist((p2, p5), (p4))} &= \text{MIN} (\text{dist}(p2, p4) , \text{dist}(p5, p4)) \\ &= \text{MIN} (0.20, 0.29) \\ &= 0.20 \end{aligned}$$

So, looking at the last distance matrix above, we see that (p2, p5) and (p3, p6) have the smallest distance from all - 0.15. We also notice that p4 and (p3, p6) have the same distance - 0.15. In that case, we can pick either one. We choose (p2, p5) and (p3, p6). So, we merge those two in a single cluster, and re-compute the distance matrix.

	P1	P2P5P3P6	P4
P1	0	0.22	0.37
P2P5P3P6	0.22	0	0.15
P4	0.37	0.20	0



The distance between (P2, P5, P3, P6) and P1 would be calculated as follows:

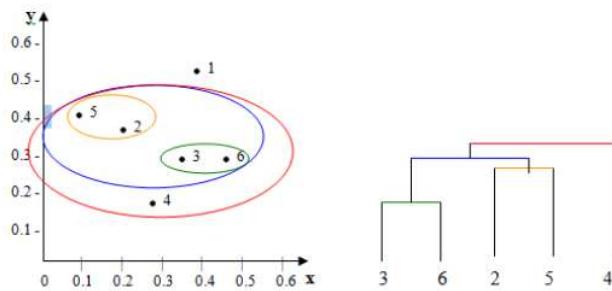
$$\begin{aligned} \text{dist((P2, P5, P3, P6), (p1))} &= \text{MIN} (\text{dist}(p2, p1) , \text{dist}(p5, p1), \text{dist}(p3, p1) \\ &\quad , \text{dist}(p6, p1)) \\ &= \text{MIN} (0.24, 0.34, 0.22, 0.23) \\ &= 0.22 \end{aligned}$$

The distance between (P2, P5, P3, P6) and P4 would be calculated as follows:

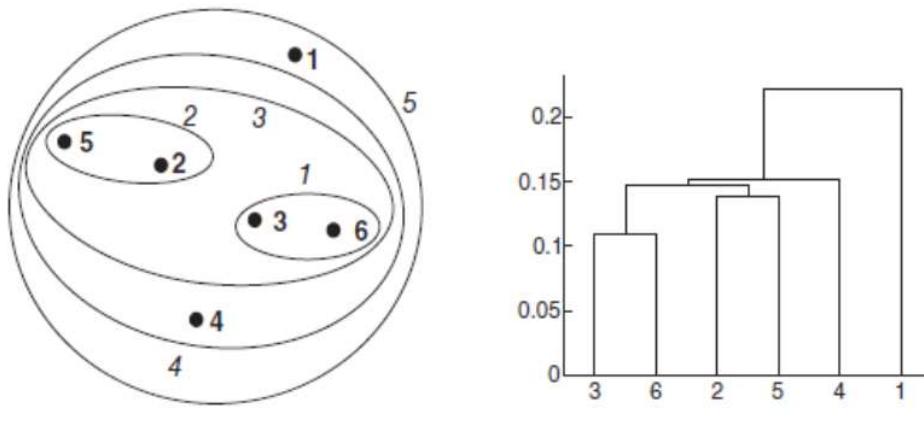
$$\begin{aligned} \text{dist((P2, P5, P3, P6), (p4))} &= \text{MIN} (\text{dist}(p2, p4) , \text{dist}(p5, p4), \text{dist}(p3, p4) \\ &\quad , \text{dist}(p6, p4)) \\ &= \text{MIN} (0.20, 0.29, 0.15, 0.22) \\ &= 0.15 \end{aligned}$$

So, looking at the last distance matrix above, we see that (p2, p5, p3, p6) and p4 have the smallest distance from all - 0.15. So, we merge those two in a single cluster, and re-compute the distance matrix.

	P1	P2P5P3P6P4
P1	0	0.22
P2P5P3P6P4	0.22	0



Finally merge clusters (P2, P5, P3, P6, P4) and P1. The clusters and dendrogram are formed as follows:



(a) Single link clustering.

(b) Single link dendrogram.

Fig: single link clustering and the dendrogram of the 6 data points

Proximity between two clusters using MAX technique (Or)

Complete Link hierarchical clustering (Or)

Clique technique for clustering.

The proximity of two clusters is defined as the maximum of the distance between any two points in the two different clusters. Complete link is less susceptible to noise and outliers, but it can break large clusters and it favors globular shapes.

The procedure is same as Min but the difference is max distance is considered while combining to closest clusters.

For Example,

After the clusters (P2, P5) and (P3, P6) are formed, the distances are calculated as follows:

$$\begin{aligned} \text{dist((p3, p6), (p2, p5))} &= \text{MAX (} \text{dist(p3, p2) , dist(p6, p2), dist(p3, p5),} \\ &\quad \text{dist(p6, p5))} \\ &= \text{MAX (0.15, 0.25, 0.28, 0.39)} \\ &= 0.39 \\ \text{dist((p3, p6), (p1))} &= \text{MAX (} \text{dist(p3, p1) , dist(p6, p1)} \\ &= \text{MAX (0.22, 0.23)} \\ &= 0.23 \\ \text{dist((p3, p6), (p4))} &= \text{MAX (} \text{dist(p3, p4) , dist(p6, p4)} \\ &= \text{MAX (0.15, 0.22)} \\ &= 0.22 \\ \text{dist((p2, p5), (p1))} &= \text{MAX (} \text{dist(p2, p1) , dist(p5, p1)} \\ &= \text{MAX (0.24, 0.34)} \\ &= 0.34 \\ \text{dist((p2, p5), (p4))} &= \text{MAX (} \text{dist(p2, p4) , dist(p5, p4)} \\ &= \text{MAX (0.20, 0.29)} \\ &= 0.29 \end{aligned}$$

Here the proximity is defined as maximum of distance but minimum of similarity. The lowest among all the distances is 0.22. So, merge clusters (P3, P6) and P4.

The distance between (P3, P6, P4) and remaining points are calculated as follows:

$$\begin{aligned} \text{dist((p3, p6, p4), (p2, p5))} &= \text{MAX} (\text{dist}(p3, p2), \text{dist}(p6, p2), \text{dist}(p4, p2), \\ &\quad \text{dist}(p3, p5), \text{dist}(p6, p5), \text{dist}(p4, p5)) \end{aligned}$$

$$= \text{MAX} (0.22, 0.23, 0.37, 0.28, 0.39, 0.29)$$

$$= 0.39$$

$$\begin{aligned} \text{dist((p3, p6, p4), (p1))} &= \text{MAX} (\text{dist}(p3, p1), \text{dist}(p6, p1), \text{dist}(p4, p1)) \end{aligned}$$

$$= \text{MAX} (0.22, 0.23, 0.37)$$

$$= 0.37$$

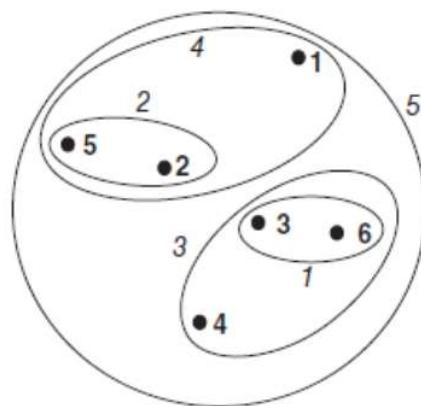
$$\begin{aligned} \text{dist((p2, p5), (p1))} &= \text{MAX} (\text{dist}(p2, p1), \text{dist}(p5, p1)) \end{aligned}$$

$$= \text{MAX} (0.24, 0.34)$$

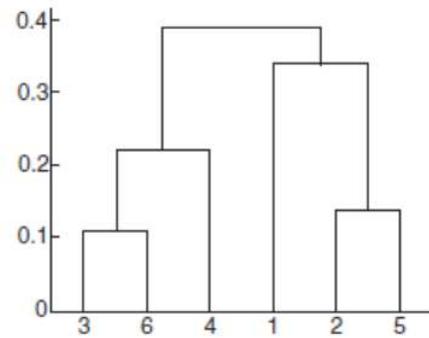
$$= 0.34$$

The lowest among all the distances is 0.34. So, merge clusters (P2, P5) and P1.

Now merge clusters (P2, P5, P1) and (P3, P6, P4). The final clusters and dendrogram are formed as follows:



(a) Complete link clustering.



(b) Complete link dendrogram.

Fig: Complete link clustering and the dendrogram of the 6 data points



UNSUPERVISED LEARNING

Unit - III

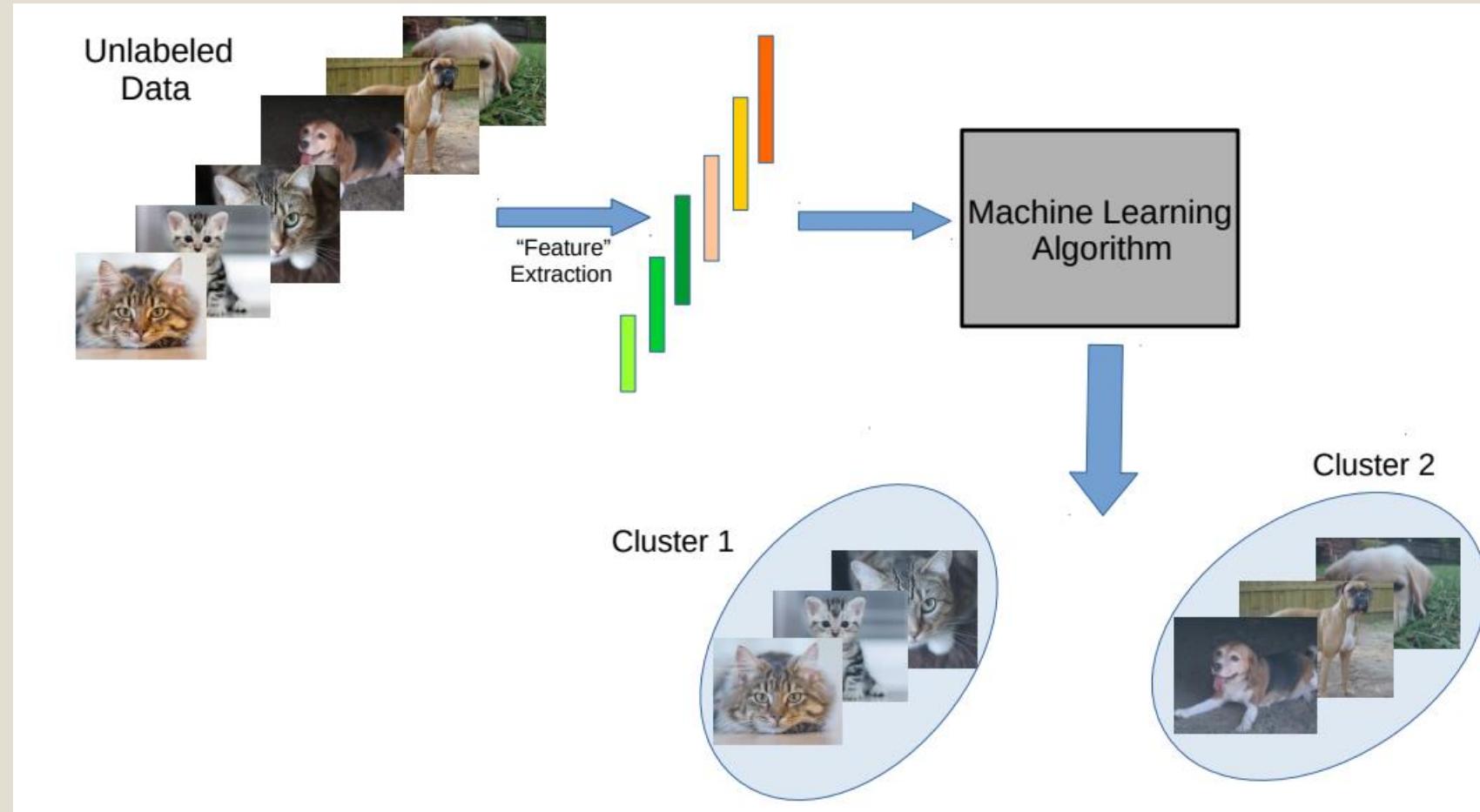
Topics

- Definition,
- K-Means,
- Hierarchical clustering techniques.
- Dimensionality reduction using PCA.
- Feature Engineering –selection, factor analysis.
- Time series modeling (time series data types, stationarity and ARIMA modeling)

Unsupervised Learning

- No labels are given to the learning algorithm, leaving it on its own to find structure in its input.
- it is about learning interesting structures in the data (unsupervisedly!)
- There is no supervision (no labels/responses), only inputs x_1, \dots, x_d
- Some examples of unsupervised learning
 - **Clustering**: Grouping similar inputs together (and dissimilar ones far apart)
 - **Dimensionality Reduction**: Reducing the data dimensionality
 - **Estimating the probability density of data** (which distribution “generated” the data)

A Typical unsupervised learning workflow for clustering



Issues and need of unsupervised learning

- ***Issues with Unsupervised Learning:***

- Unsupervised Learning is harder as compared to Supervised Learning tasks..
- How do we know if results are meaningful since no answer labels are available?
- Let the expert look at the results (external evaluation)
- Define an objective function on clustering (internal evaluation)

- ***Why Unsupervised Learning is needed despite of these issues?***

- Annotating large datasets is very costly and hence we can label only a few examples manually. Example: Speech Recognition
- There may be cases where we don't know how many/what classes the data is divided into. Example: Data Mining
- We may want to use clustering to gain some insight into the structure of the data before designing a classifier.



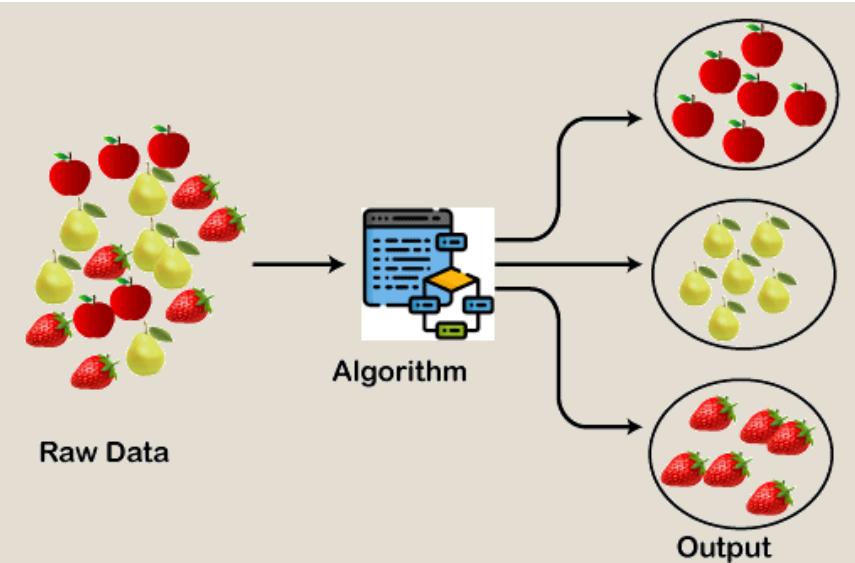
CLUSTERING

Cluster Analysis

- Cluster analysis groups the data objects based only on information found in the data that describes the objects and their relationships.
- The goal is that the objects within a group be similar (or related) to one another and different from objects in another group.
- The greater the similarity within a group and the greater the difference between groups the better or more distinct the clustering.

Clustering

- Given:
 - N unlabeled examples $\{x_1, \dots, x_d\}$;
 - no. of desired partitions K
- Goal:
 - Group the examples into K “homogeneous” partitions
 - Loosely speaking, it is classification without ground truth labels
 - A good clustering is one that achieves:
 - High within-cluster similarity
 - Low inter-cluster similarity



Few examples of clustering

- Document/Image/Webpage Clustering
- Image Segmentation (clustering pixels)

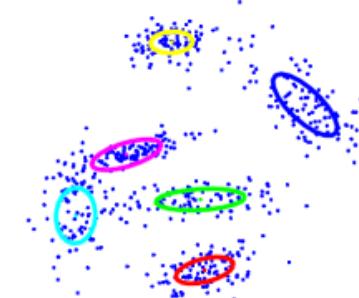


- Clustering web-search results
- Clustering (people) nodes in (social) networks/graphs
- .. and many more..

Types of Clustering

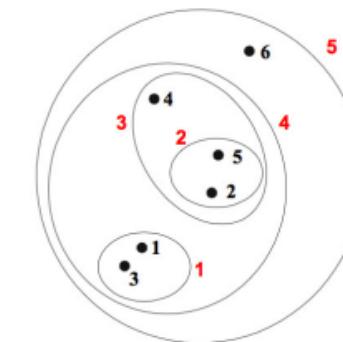
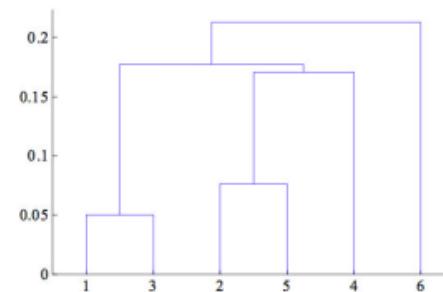
① Flat or Partitional clustering

- Partitions are independent of each other



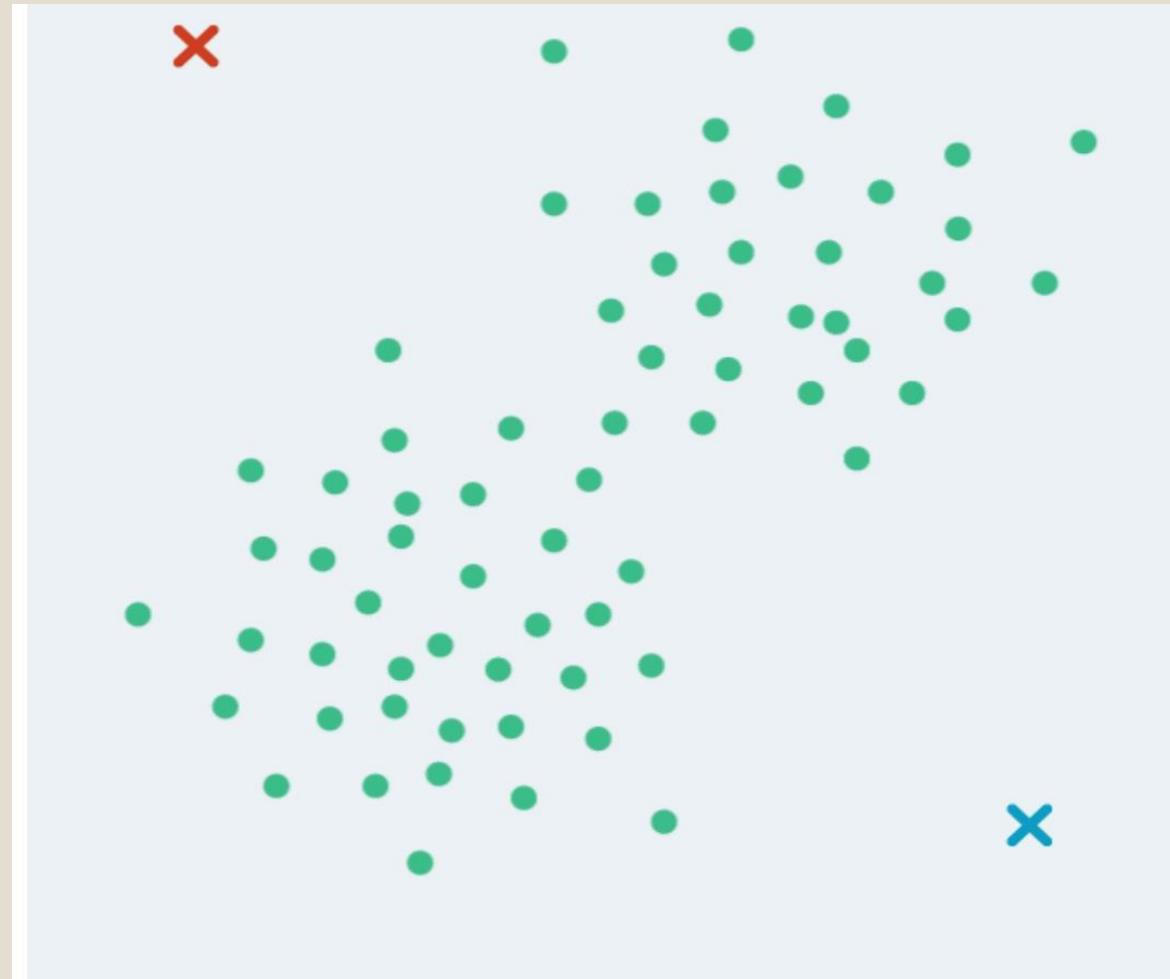
② Hierarchical clustering

- Partitions can be visualized using a tree structure (a dendrogram)



- Possible to view partitions at different levels of granularities by “cutting” the tree at some level

K-means

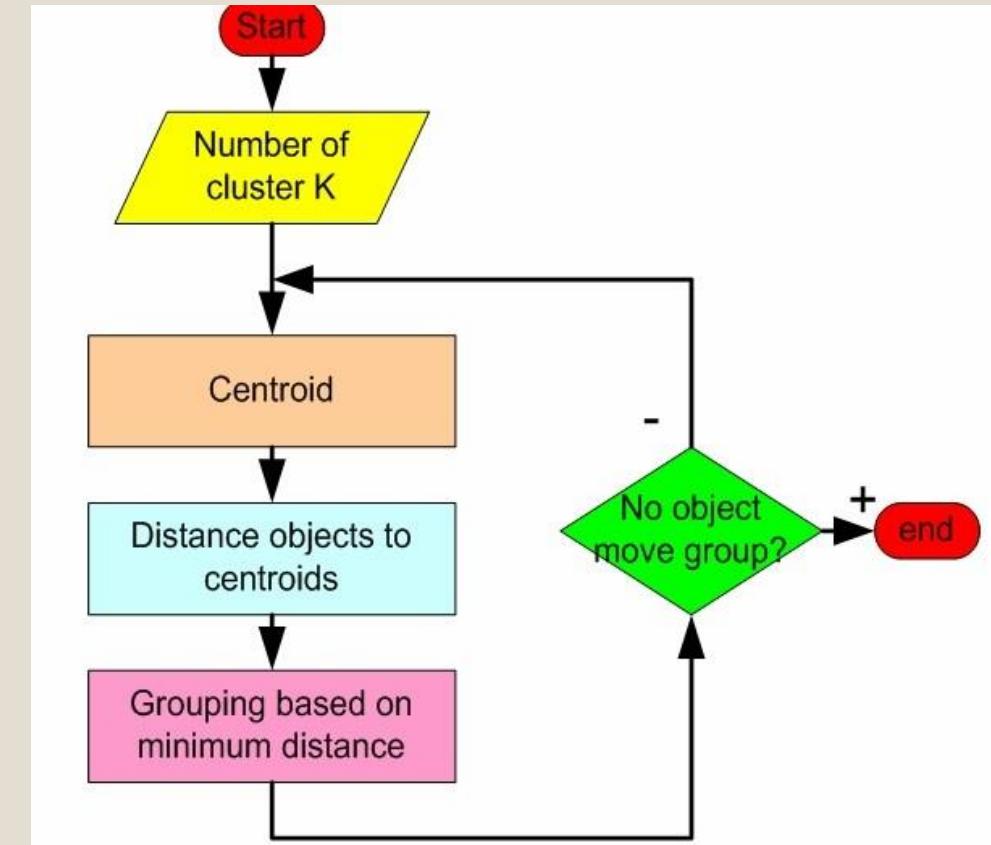


K-means

- A simple Prototype-based clustering technique
- Defines a prototype in terms of centroid (mean of group of points)
- This is typically applied to objects in continuous n-dimensional space.
- K-means algorithm is an iterative algorithm that tries to partition the dataset into K distinct non-overlapping subgroups(clusters) where each data point belongs to only one group.
- It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different as possible.

Basic K-means algorithm

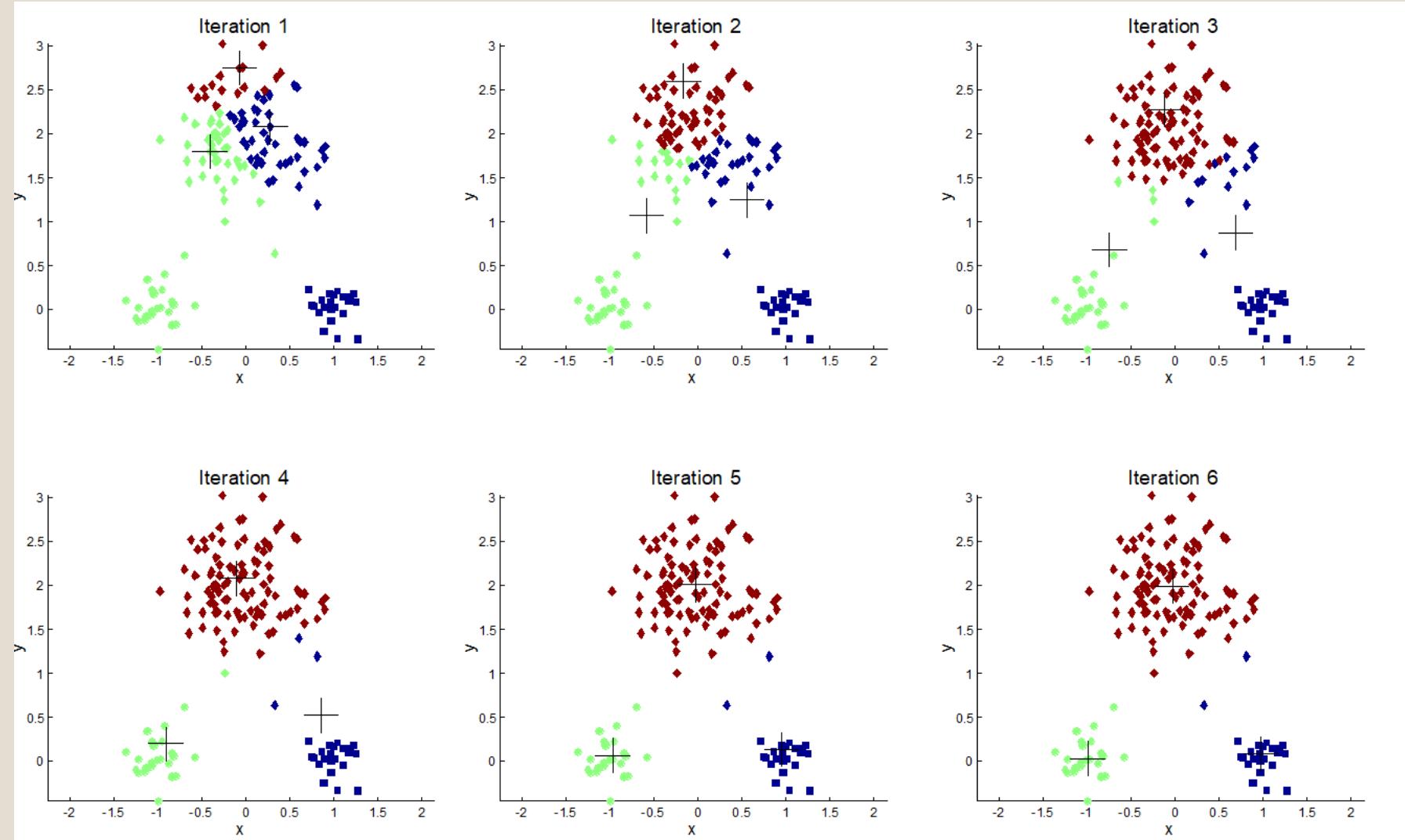
- First choose K initial centroids, where K is the user-specified parameter (number of desired clusters)
- Each point is then assigned to the closest centroid, and each collection of points assigned to a centroid is a cluster.
- The centroid of each cluster is then updated based on the points assigned to the cluster.
- We repeat the assignment and update steps until no point changes clusters, or equivalently, until the centroid remain the same.



Basic K-means algorithm

- Select K points as *initial centroids*
- **Repeat**
 - From K clusters by assigning each point to its closest centroid
 - Recompute the centroid of each cluster
- **Until** centroid do not change

Using K-means to find 3 clusters in sample data.



Example

- cluster the following eight points into three clusters.

**A1(2, 10),
A2(2, 5),
A3(8, 4),
A4(5, 8),
A5(7, 5),
A6(6, 4),
A7(1, 2),
A8(4, 9)**

Let, Initial cluster centers are:

**A1(2, 10),
A4(5, 8) and
A7(1, 2).**

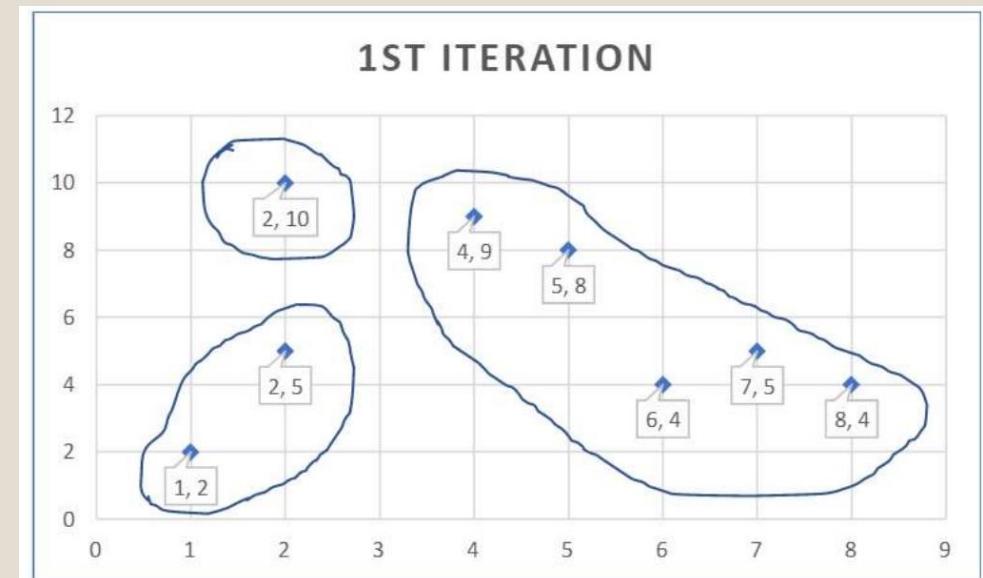
The distance function between two points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is defined as-

$$P(a, b) = |x_2 - x_1| + |y_2 - y_1|$$

Example (contd...)

Iteration – 1

Given Points	Distance from centre (2, 10) of Cluster-01	Distance from centre (5, 8) of Cluster-02	Distance from centre (1, 2) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	5	9	C1
A2(2, 5)	5	6	4	C3
A3(8, 4)	12	7	9	C2
A4(5, 8)	5	0	10	C2
A5(7, 5)	10	5	9	C2
A6(6, 4)	10	5	7	C2
A7(1, 2)	9	10	0	C3
A8(4, 9)	3	2	10	C2



Example (contd...)

Iteration – 1



For Cluster-01:

We have only one point A1(**2, 10**) in Cluster-01.
So, cluster center remains the same.

For Cluster-02:

Centre of Cluster-02

$$\begin{aligned} &= ((8 + 5 + 7 + 6 + 4)/5, (4 + 8 + 5 + 4 + 9)/5) \\ &= (\mathbf{6}, \mathbf{6}) \end{aligned}$$

For Cluster-03:

Centre of Cluster-03

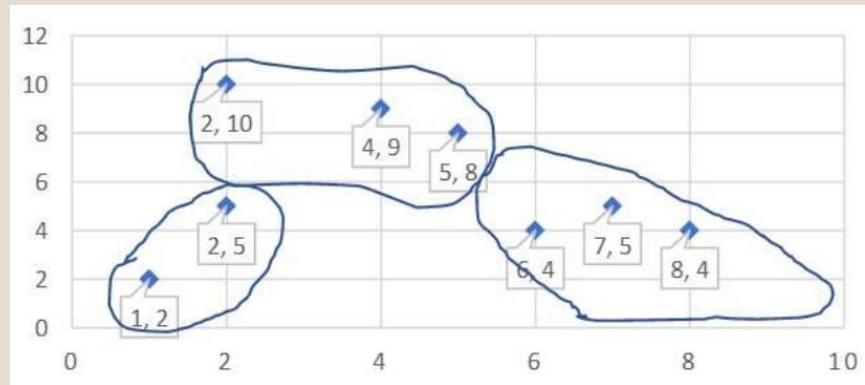
$$\begin{aligned} &= ((2 + 1)/2, (5 + 2)/2) \\ &= (\mathbf{1.5}, \mathbf{3.5}) \end{aligned}$$

This is completion of Iteration-01.

Example (contd...)

Iteration – 2

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (6, 6) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	8	7	C1
A2(2, 5)	5	5	2	C3
A3(8, 4)	12	4	7	C2
A4(5, 8)	5	3	8	C2
A5(7, 5)	10	2	7	C2
A6(6, 4)	10	2	5	C2
A7(1, 2)	9	9	2	C3
A8(4, 9)	3	5	8	C1



For Cluster-01:

Center of Cluster-01

$$= ((2 + 4)/2, (10 + 9)/2) = (3, 9.5)$$

For Cluster-02:

Center of Cluster-02

$$= ((8 + 5 + 7 + 6)/4, (4 + 8 + 5 + 4)/4) = (6.5, 5.25)$$

For Cluster-03:

Center of Cluster-03

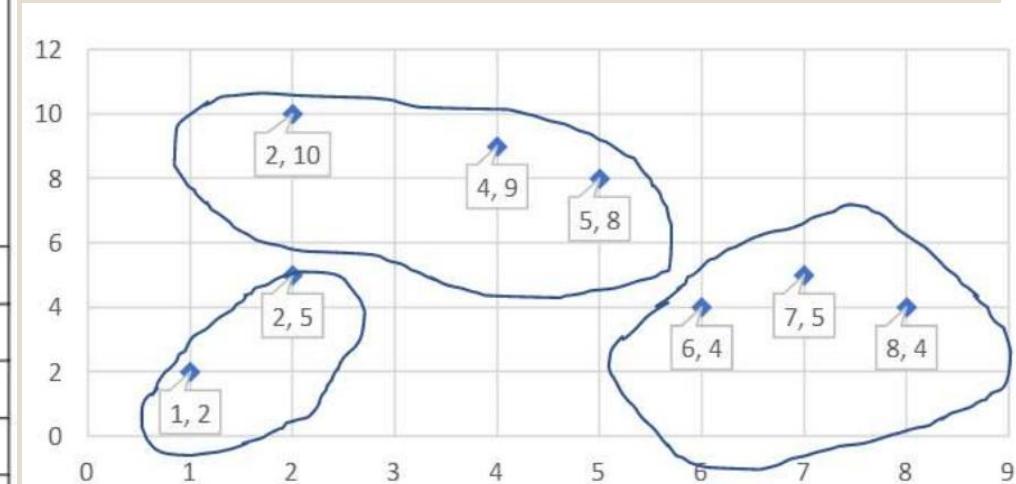
$$= ((2 + 1)/2, (5 + 2)/2) = (1.5, 3.5)$$

This is completion of Iteration-02.

Example (contd...)

Iteration – 3

Given Points	Distance from center (3, 9.5) of Cluster-01	Distance from center (6.5, 5.25) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	1.5	9.25	7	C1
A2(2, 5)	5.5	4.75	2	C3
A3(8, 4)	10.5	2.75	7	C2
A4(5, 8)	3.5	4.25	8	C1
A5(7, 5)	8.5	0.75	7	C2
A6(6, 4)	8.5	1.75	5	C2
A7(1, 2)	9.5	8.75	2	C3
A8(4, 9)	1.5	6.25	8	C1



K-means clustering details

- Initial centroids are often chosen randomly.
 - Clusters produced vary from one run to another.
- The centroid is (typically) the mean of the points in the cluster.
- ‘Closeness’ is measured by Euclidean distance, cosine similarity, correlation, etc.
- K-means will converge for common similarity measures mentioned above.
- Most of the convergence happens in the first few iterations.
 - Often the stopping condition is changed to ‘Until relatively few points change clusters’
- Complexity is $O(n * K * I * d)$
 - n = number of points, K = number of clusters,
 I = number of iterations, d = number of attributes

Strengths and weaknesses

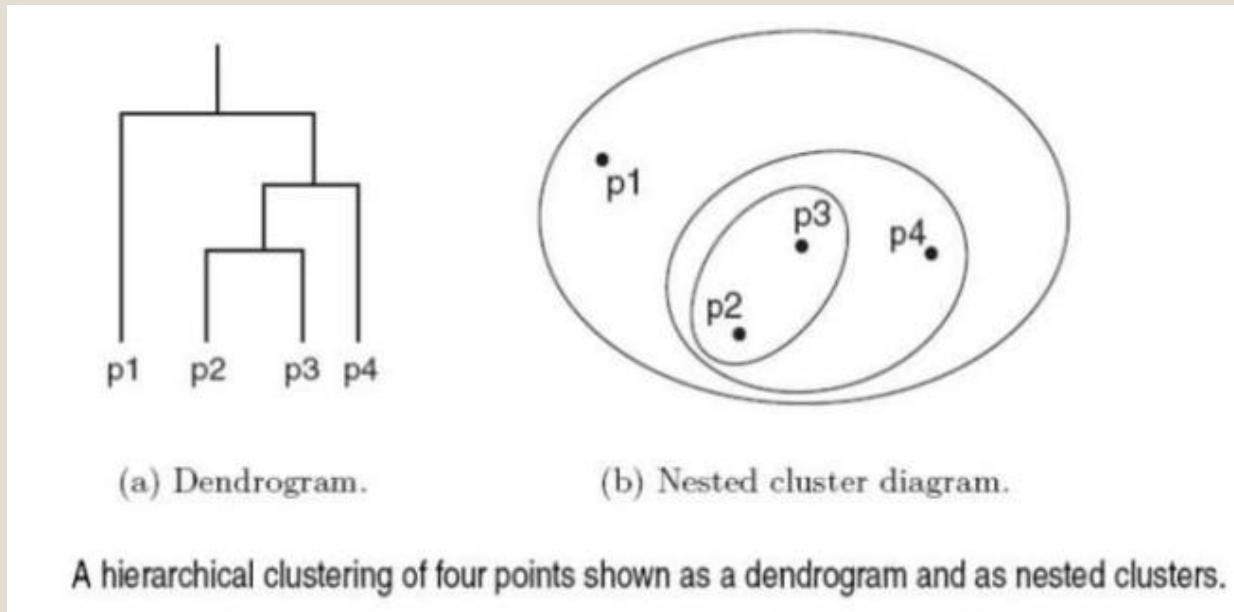
- Strengths
 - It is simple and useful for all varieties of datatypes
 - It is quiet efficient, even though multiple iterations are required.
- Weakness
 - it cannot handle non-globular clusters, clusters of different sizes and densities even though it can find pure sub-clusters.
 - Outliers cannot be clustered.
 - K-means is restricted for the notion of centroid.

Hierarchical Clustering

- In hierarchical clustering, the set of clusters organized as nested clusters in the form of a tree known as dendrogram.
- There are 2 types of hierarchical clustering.
- **Agglomerative** hierarchical clustering
 - Start with points as individual clusters.
 - at each step, merge the closest pair of clusters.
- **Divisive** hierarchical clustering.
 - start with one, all-inclusive cluster
 - at each step, split a cluster until only singleton clusters of individual points remain.

Hierarchical Clustering - Dendrogram

- **Dendrogram** is a tree-like diagram to display a hierarchical clustering graphically
- It displays both the cluster, sub-cluster relationships and the order in which the clusters were merged.
- A **Nested Cluster Diagram** can also be used for sets of 2D points



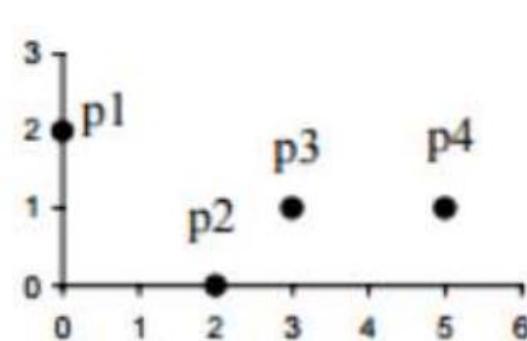
Basic Agglomerative Hierarchical Clustering Algorithm

- Starts with initial points as clusters, successively merge the two closest clusters until only one cluster remains.

```
1: Compute the proximity matrix, if necessary.  
2: repeat  
3:   Merge the closest two clusters.  
4:   Update the proximity matrix to reflect the proximity between the new  
      cluster and the original clusters.  
5: until Only one cluster remains.
```

Cluster proximity

- cluster proximity is defined as similarity or dissimilarity between elements or clusters. They are generally defined by proximity matrix.
- Proximity matrix is a matrix containing distance between clusters.
- Four points and their corresponding data and proximity(distance) matrices are shown below.

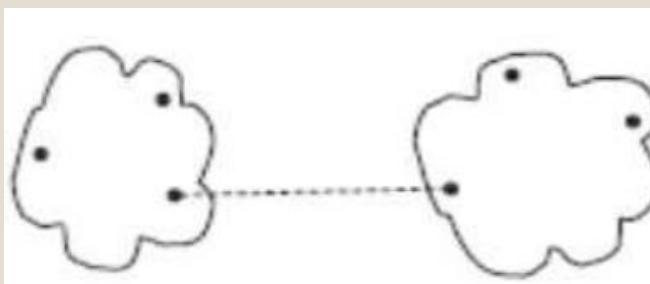


point	x	y
p1	0	2
p2	2	0
p3	3	1
p4	5	1

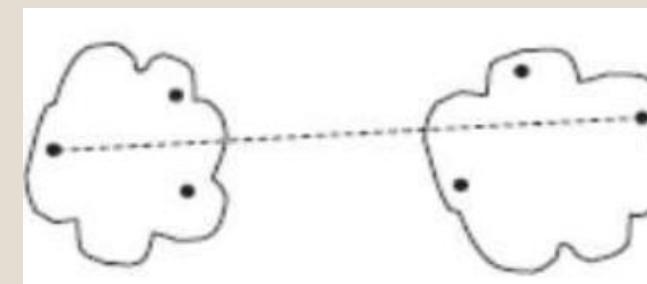
	p1	p2	p3	p4
p1	0.000	2.828	3.162	5.099
p2	2.828	0.000	1.414	3.162
p3	3.162	1.414	0.000	2.000
p4	5.099	3.162	2.000	0.000

Graph based Methods for defining proximity between the clusters

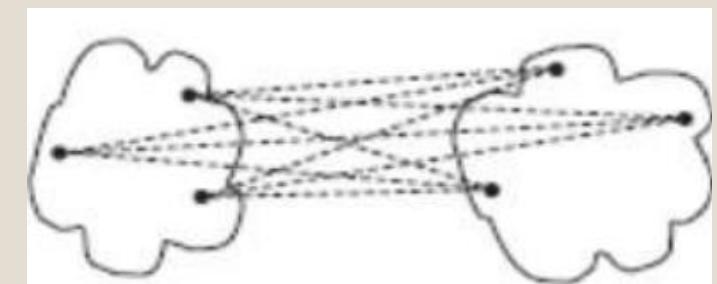
1. **Min** : Min defines the cluster proximity between the closest two points that are in different clusters. This is also called **Single Link Technique**.



2. **Max** : Max defines cluster proximity between the farthest two points that are in different clusters. This is also called as **Complete Link Technique**.



3. **Group Average** defines cluster proximity to be the average proximities of all pairs of points from different clusters.

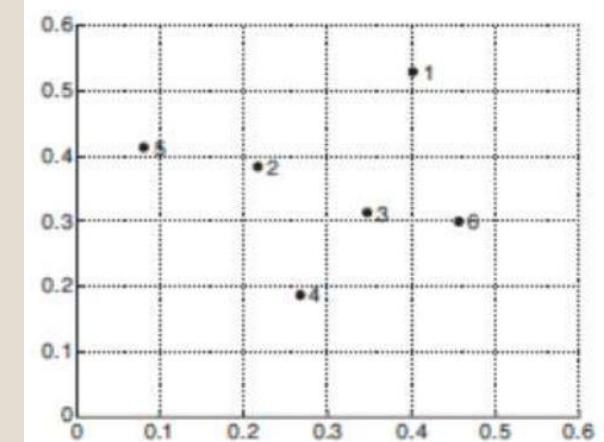


Single Link Hierarchical Clustering

Let us consider a dataset with 6 points

Point	x Coordinate	y Coordinate
p1	0.40	0.53
p2	0.22	0.38
p3	0.35	0.32
p4	0.26	0.19
p5	0.08	0.41
p6	0.45	0.30

xy coordinates of 6 points.



Set of 6 two-dimensional points.

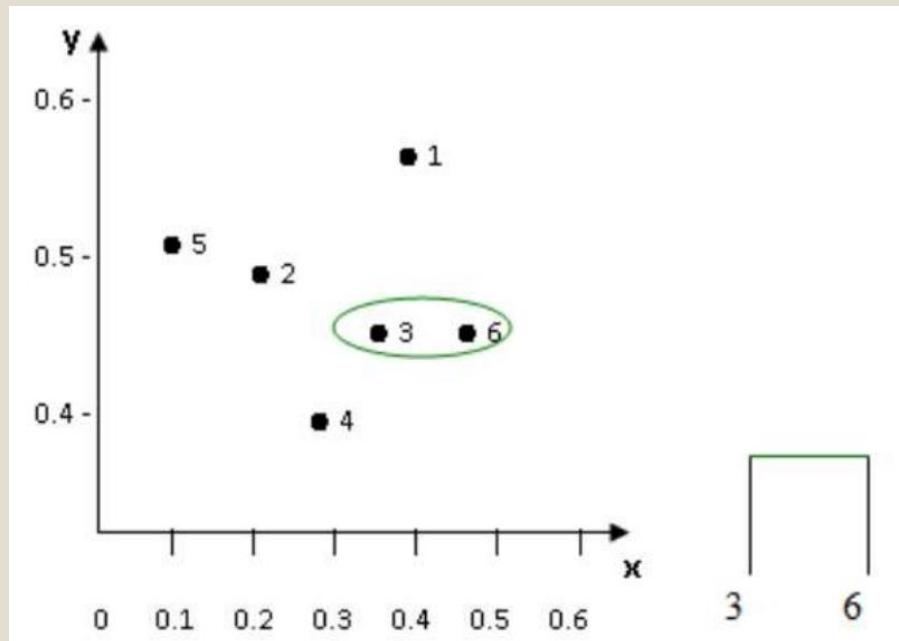
	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

Euclidean distance matrix for 6 points.

◦ Min or Single Link Technique

- The proximity of two clusters - the minimum of the distance between any two points in two different clusters.
- P3, P6 is having the lowest distance, so merge the data points into a single cluster.

	P1	P2	P3	P4	P5	P6
P1	0	0.24	0.22	0.37	0.34	0.23
P2	0.24	0	0.15	0.20	0.14	0.25
P3	0.22	0.15	0	0.15	0.28	0.11
P4	0.37	0.20	0.15	0	0.29	0.22
P5	0.34	0.14	0.28	0.29	0	0.39
P6	0.23	0.25	0.11	0.22	0.39	0

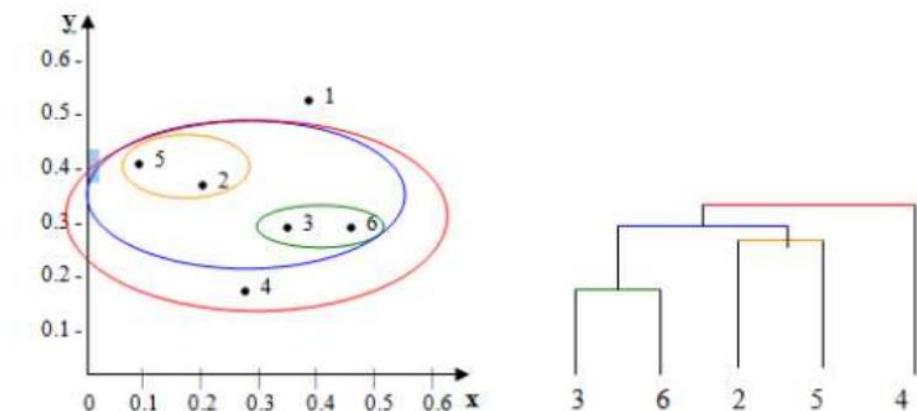


	P1	P2	P3P6	P4	P5
P1	0	0.24	0.22	0.37	0.34
P2	0.24	0	0.15	0.20	0.14
P3P6	0.22	0.15	0	0.15	0.28
P4	0.37	0.20	0.15	0	0.29
P5	0.34	0.14	0.28	0.29	0

The next lowest distance is for P2P5, so merge those two data points into a single cluster. the matrix obtained is shown below.

	P1	P2P5	P3P6	P4
P1	0	0.24	0.22	0.37
P2P5	0.24	0	0.15	0.20
P3P6	0.22	0.15	0	0.15
P4	0.37	0.20	0.15	0

	P1	P2P5P3P6P4
P1	0	0.22
P2P5P3P6P4	0.22	0



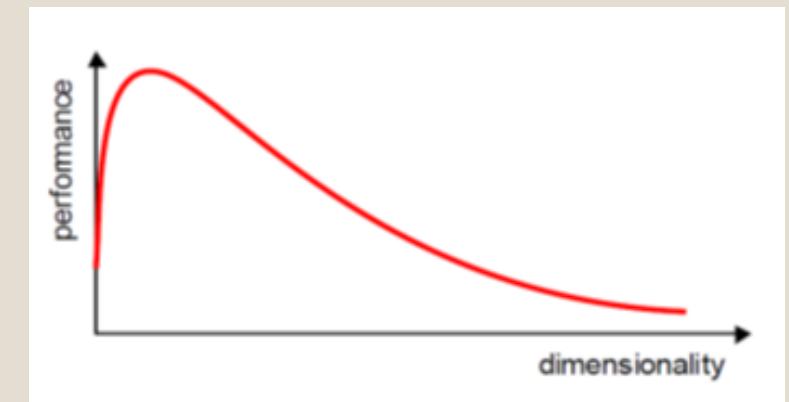


DIMENSIONALITY REDUCTION

PCA

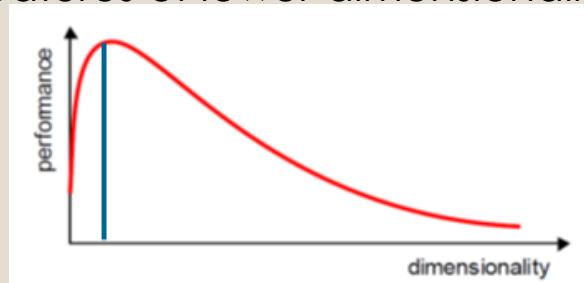
Curse of Dimensionality

- Increasing the number of features will not always improve classification accuracy.
- In practice, the inclusion of more features might actually lead to **worse** performance.
- The number of training examples required increases **exponentially** with dimensionality \mathbf{d} (i.e., k^d).
- More features leads to model overfitting.



Dimensionality Reduction

- What is the objective?
 - Choose an optimum set of features of lower dimensionality to **improve** classification accuracy.



- What happens if we get rid of non-important features?
 - Faster training and inference
 - Data Visualization becomes easier.
- Different methods can be used to reduce dimensionality:
 - Feature extraction
 - Feature selection

Dimensionality Reduction (contd...)

- **Feature extraction:** finds a set of **new** features (i.e., through some mapping $f()$) from the **existing** features.
- **Feature selection:** chooses a subset of the **original** features.

The mapping $f()$ could be **linear** or **non-linear**

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

$K \ll N$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \rightarrow \mathbf{y} = \begin{bmatrix} x_{i_1} \\ x_{i_2} \\ \cdot \\ \cdot \\ \cdot \\ x_{i_K} \end{bmatrix}$$

$K \ll N$

Feature Extraction

- Linear combinations are particularly attractive because they are simpler to compute and analytically tractable.
- Given $\mathbf{x} \in \mathbb{R}^N$, find an $K \times N$ matrix \mathbf{T} such that:

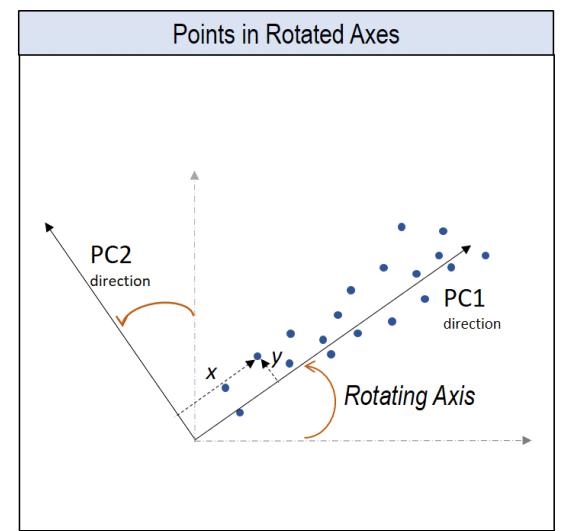
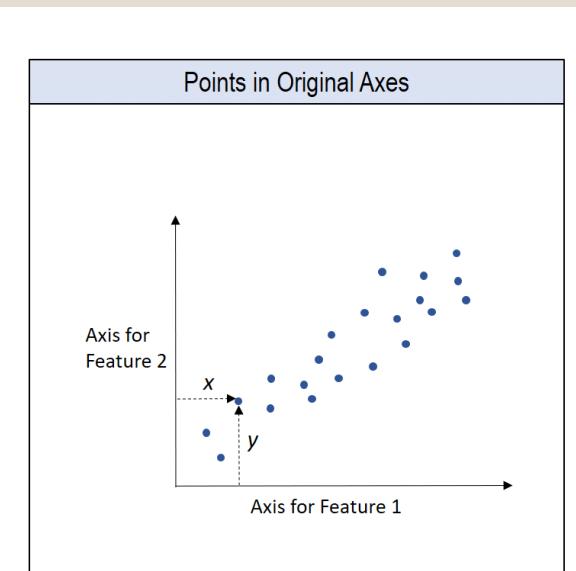
$$\mathbf{y} = \mathbf{T}\mathbf{x} \in \mathbb{R}^K \text{ where } K << N$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \\ x_N \end{bmatrix} \xrightarrow{\mathbf{T}, f(\mathbf{x})} \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \cdot \\ \cdot \\ \cdot \\ y_K \end{bmatrix}$$

This is a projection from the N -dimensional space to a K -dimensional space.

Principal Component Analysis (PCA)

- Is a process of figuring out most important features or Principal Components that has the most impact on the target variable.
- The main idea of PCA
 - Reduce the dimensionality of the dataset consisting of many variables correlated with each other.
 - Retaining the variation present in the dataset upto the maximum extent. – This is done by transforming the variables to a new set of variables called principal components.
 - Principal components are orthogonal and ordered.



PCA

- The new variables are dimensions
 - Are linear combination of the original ones
 - Are uncorrelated with one another
 - Orthogonal in original dimension space
 - Capture as much of the original variance as possible
 - Are called principal components.
- First principal component is the direction of the greatest variability (covariance) in the data.
- Second is the next orthogonal (uncorrelated) direction of greatest variability.

PCA

- The “optimal” set of basis vectors $\langle u_1, u_2, \dots, u_K \rangle$ can be found as follows (we will see why):

(1) Find the **eigenvectors** u_i of the **covariance** matrix of the (training) data Σ_x

$$\Sigma_x u_i = \lambda_i u_i$$

(2) Choose the K “**largest**” eigenvectors u_i (i.e., corresponding to the K “**largest**” eigenvalues λ_i)

$\langle u_1, u_2, \dots, u_K \rangle$ correspond to the “optimal” basis!

We refer to the “**largest**” eigenvectors u_i as **principal components**.

Some terms to move forward...

- Dimensionality : No.of variables in the dataset
- Correlation : How strongly two variables are related. Ranges from -1 to 1
- Eigen vector : An eigen vector of a matrix is one which is scalar multiple.

$$\Sigma_x u_i = \lambda_i u_i$$

- Covariance matrix : Consists of variances between the pairs of variables.

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

$$Cov(x, y) = \frac{\sum_{i=1}^n (x_i - \bar{x}) * (y_i - \bar{y})}{N}$$

$$Var(xi) = \frac{\sum_{i=1}^n (x_i - \bar{x})^2}{N}$$

Steps to perform PCA

1. Normalize the data
2. Calculate the covariance matrix
3. Calculate the eigen values and eigen vectors
4. Choosing the components and forming a feature vector
5. Recasting data along PC axes

Example

- Consider the following dataset

f1	f2	f3	f4
1	2	3	4
5	5	6	7
1	4	2	3
5	3	2	1
8	1	2	2

Step 1 : Normalize the data

- PCA works well with normalized data? Why?

	f1	f2	f3	f4
Mean	4	3	3	3.4
Std. Deviation	3	1.58	1.73	2.3

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

$$Standard\ deviation = \sqrt{\frac{\sum_{i=1}^n (x_i - \bar{x})^2}{N}}$$

$$x_{new} = \frac{x - \mu}{\sigma}$$

- The dataset after normalization

f1	f2	f3	f4
-1	-0.632	0	0.260
0.333	1.264	1.732	1.563
-1	0.632	-0.577	-0.173
0.333	0	-0.577	-1.042
1.333	-1.264	-0.577	-0.608

Step 2: Calculate the covariance matrix

	f1	f2	f3	f4
f1	Var(f1)	Cov (f1, f2)	Cov (f1, f3)	Cov (f1, f4)
f2	Cov (f2, f1)	Var(f2)	Cov (f2, f3)	Cov (f2, f4)
f3	Cov (f3, f1)	Cov (f3, f2)	Var(f3)	Cov (f3, f4)
f4	Cov (f4, f1)	Cov (f4, f2)	Cov (f4, f3)	Var (f4)

	f1	f2	f3	f4
f1	0.8	-0.252	0.338	-0.144
f2	-0.252	0.8	0.511	0.494
f3	0.003	0.511	0.8	0.752
f4	-0.144	0.494	0.752	0.8

Step 3: Calculate the eigen values and eigen vectors for the covariance matrix

$$\det(A - \lambda I) = 0$$
$$\det \begin{bmatrix} 0.8 & -0.252 & 0.338 & -0.144 \\ -0.252 & 0.8 & 0.511 & 0.494 \\ 0.003 & 0.511 & 0.8 & 0.752 \\ -0.144 & 0.494 & 0.752 & 0.8 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = 0$$

$$\det \begin{bmatrix} 0.8 - \lambda & -0.252 & 0.338 & -0.144 \\ -0.252 & 0.8 - \lambda & 0.511 & 0.494 \\ 0.003 & 0.511 & 0.8 - \lambda & 0.752 \\ -0.144 & 0.494 & 0.752 & 0.8 - \lambda \end{bmatrix} = 0$$

Solving the above equation =0

$$\lambda = 2.51579324, 1.0652885, 0.39388704, 0.02503121$$

Step 3: Calculate the eigen values and eigen vectors for the covariance matrix

Eigenvectors:

Solving the $(A - \lambda I)v = 0$ equation for v vector with different λ values:

$$\begin{bmatrix} 0.8 - \lambda & -0.252 & 0.338 & -0.144 \\ -0.252 & 0.8 - \lambda & 0.511 & 0.494 \\ 0.003 & 0.511 & 0.8 - \lambda & 0.752 \\ -0.144 & 0.494 & 0.752 & 0.8 - \lambda \end{bmatrix} * \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \end{bmatrix} = 0$$

For $\lambda = 2.51579324$, solving the above equation, the values for v vector are.

$$v_1 = 0.16195986$$

$$v_2 = -0.52404813$$

$$v_3 = -0.58589647$$

$$v_4 = -0.59654663$$

Step 3: Calculate the eigen values and eigen vectors for the covariance matrix

Going by the same approach, we can calculate the eigen vectors for the other eigen value $\lambda = 1.0652885$. We can form a matrix using the eigen vectors.

v1 = -0.9170

v2 = 0.2069

v3 = -0.320

v4 = -0.115

Step 4 : Choosing components and forming a feature vector

- Order the eigen values from largest to smallest to get the components in order of significance.
- Dimensionality reduction part :
 - With n variables we get n eigen values and vectors.
 - Eigen vector corresponding to highest eigen value is the principal component of the dataset.
 - Choose the required first p vectors and ignore the rest.
- Do we lose out some information in this process?
- Stick to top 2 values if we want 2-dimensions.

$$\begin{bmatrix} 0.161 & -0.9170 \\ -0.524 & 0.2069 \\ -0.585 & -0.320 \\ -0.596 & -0.1159 \end{bmatrix}$$

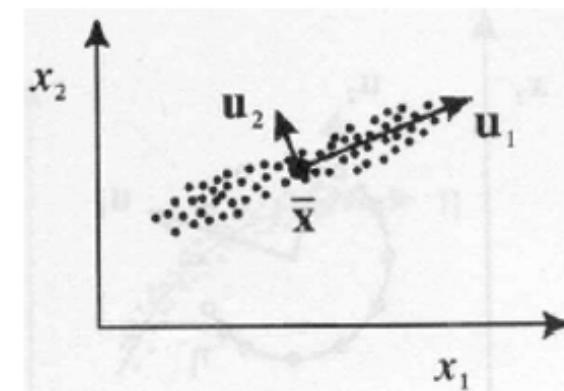
Step 5: Recasting data along principal component axes

- transform the original samples onto the new subspace by re-orienting data from original axes to the ones that are now represented by principal components.

$$\begin{bmatrix} -1 & -0.632 & 0 & 0.260 \\ 0.333 & 1.264 & 1.732 & 1.563 \\ -1 & 0.632 & -0.577 & -0.173 \\ 0.333 & 0 & -0.577 & -1.042 \\ 1.333 & -1.264 & -0.577 & -0.608 \end{bmatrix} \times \begin{bmatrix} 0.161 & -0.9170 \\ -0.524 & 0.2069 \\ -0.585 & -0.320 \\ -0.596 & -0.1159 \end{bmatrix} = \begin{bmatrix} 0.014 & 0.755 \\ -2.556 & -0.780 \\ -0.051 & 1.253 \\ 1.0141 & 0.0002 \\ 1.579 & -1.2289 \end{bmatrix}$$

Interpretation of PCA

- PCA chooses the **eigenvectors** of the covariance matrix corresponding to the **largest** eigenvalues.
- The **eigenvalues** correspond to the **variance** of the data along the eigenvector directions.
- Therefore, PCA projects the data along the directions where the data varies **most**.
- PCA preserves as much **information** in the data by preserving as much **variance** in the data.



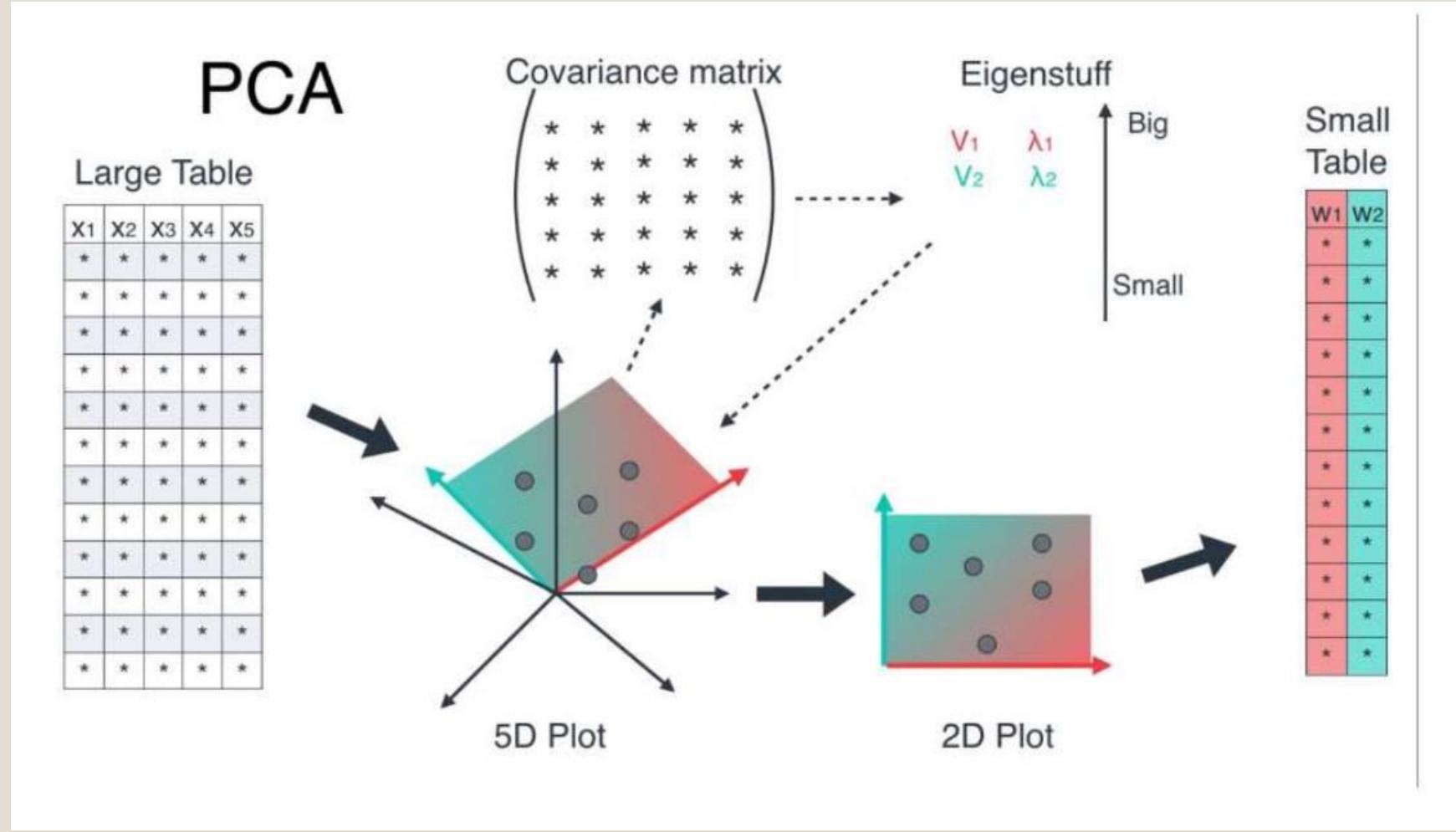
u_1 : direction of **max** variance

u_2 : orthogonal to u_1

Applications of PCA

- PCA is used as a dimensionality reduction technique in domains like
 - Facial Recognition
 - Computer Vision and
 - Image Compression.
- It is also used to find patterns in high dimensional data in the fields of finance, data mining, bioinformatics, psychology etc.

The whole picture of PCA



Application of PCA to images

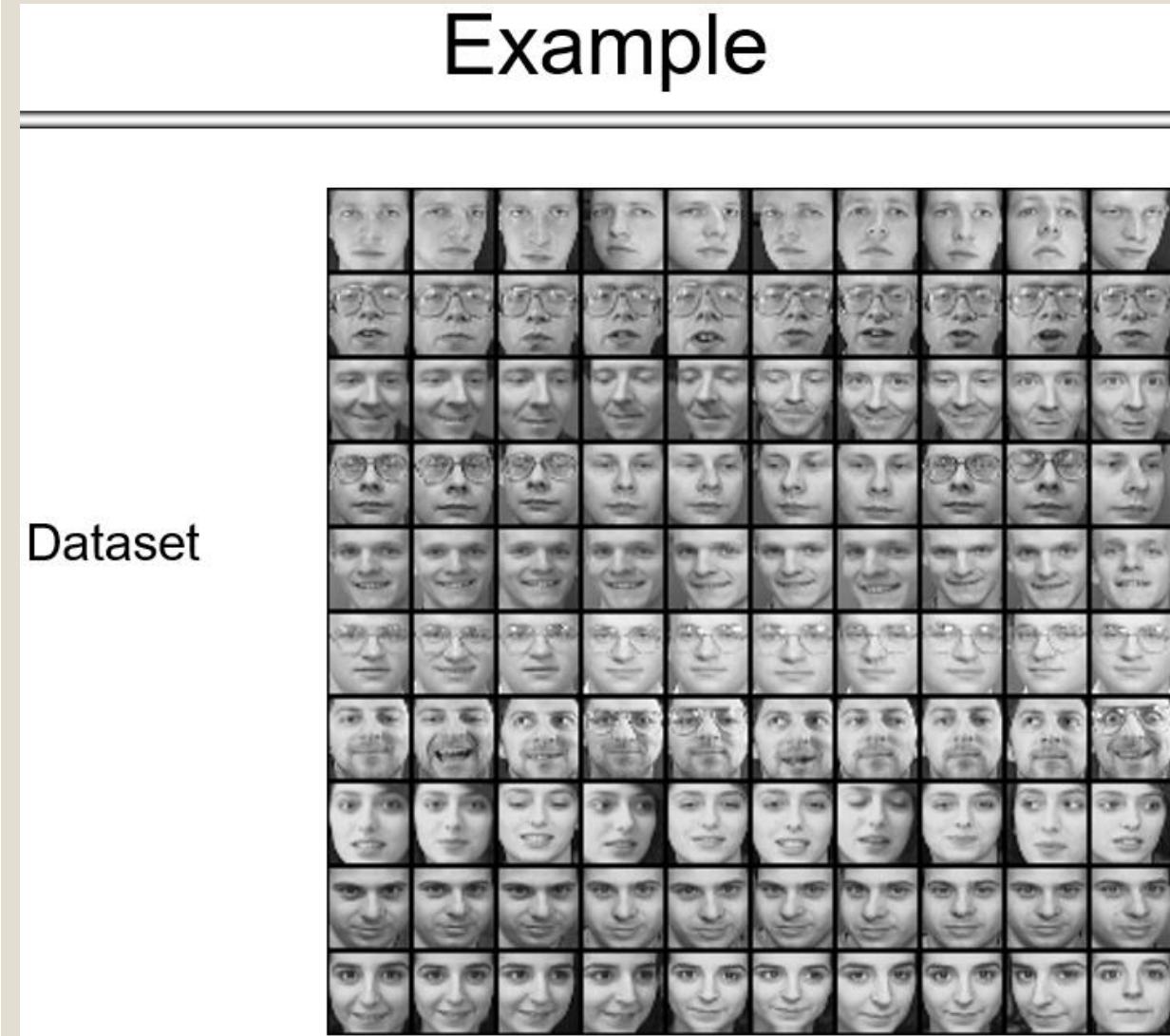
- The goal is to represent images in a space of lower dimensionality using PCA.
 - Useful for various applications, e.g., face recognition, image compression, etc.
- Given **M** images of size **N x N**, first represent each image as a 1D vector (i.e., by stacking the rows together).
 - Note that for **face recognition**, faces must be **centered** and of the same **size**.



Application of PCA to images

- Next compute the co-variance matrix.
 - It is a challenging job as it is very large ($N^2 \times N^2$)
- Compute the eigen values and eigen vectors for the covariance matrix.
 - Computationally expensive
- Do the dimensionality reduction step and approximate the first K eigen vectors.

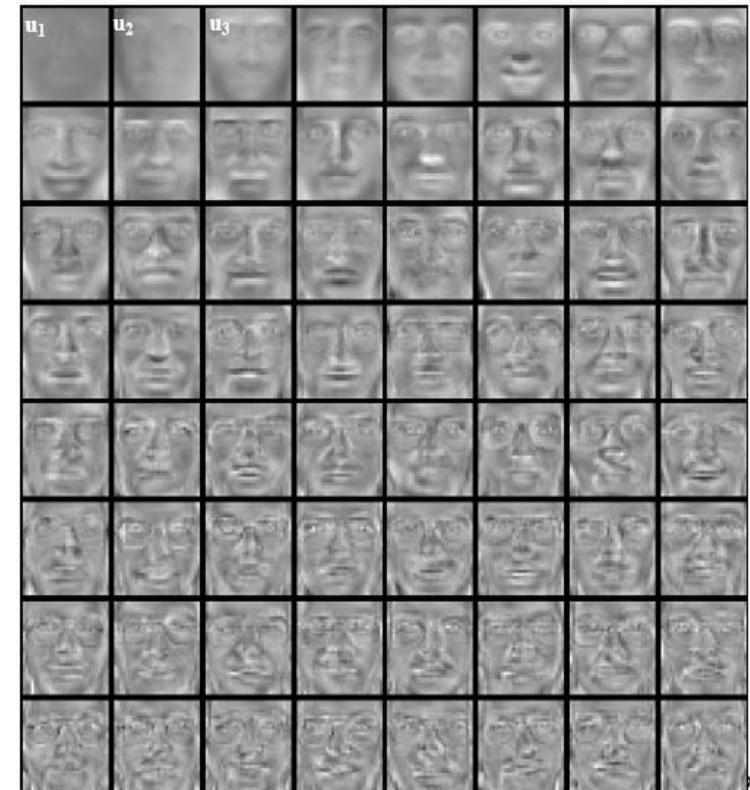
Dataset



Mean face: \bar{x}



Top eigenvectors: u_1, \dots, u_k
(visualized as an image - eigenfaces)



K-MEANS

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K distinct non-overlapping subgroups (clusters) where each data point belongs to **only one group**. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible.

Algorithm for Basic K-Means algorithm

Select **K** points as initial centroids

Repeat

 Form K clusters by assigning each point to its closest centroid.

 Recompute the centroid of each cluster

Until Centroids do not change

Example:

Cluster the following eight points (with (x, y) representing locations) into three clusters (let us say, we need 3 clusters):

A1(2, 10),

A2(2, 5),

A3(8, 4),

A4(5, 8),

A5(7, 5),

A6(6, 4),

A7(1, 2),

A8(4, 9)

Let, Initial cluster centers are:

A1(2, 10),

A4(5, 8) and

A7(1, 2).

The distance function between two points $a = (x_1, y_1)$ and $b = (x_2, y_2)$ is defined as-

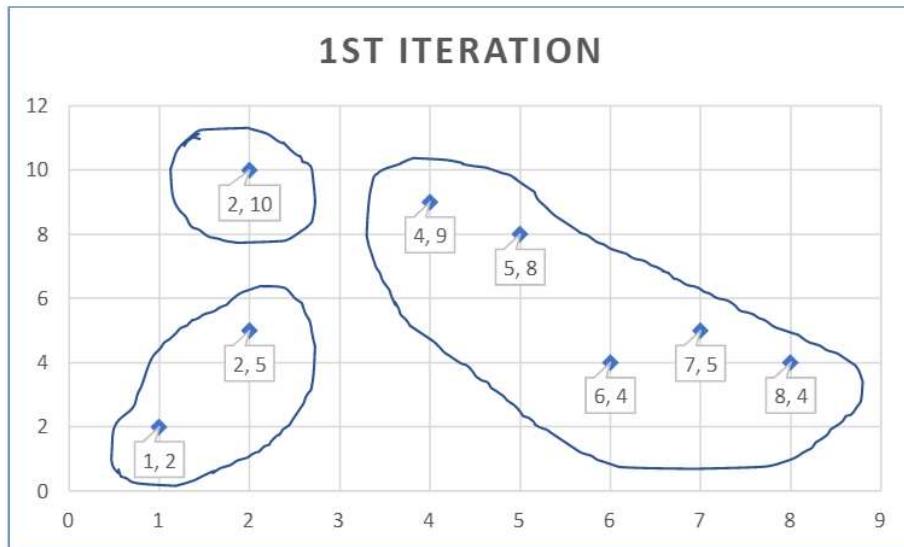
$$P(a, b) = |x_2 - x_1| + |y_2 - y_1|$$

Note: You can use any other distance formula like Euclidean distance too.

Now, calculate the distance from each point to the randomly picked centroids. And make the cluster to which the points are **nearer**.

1st Iteration

Given Points	Distance from centre (2, 10) of Cluster-01	Distance from centre (5, 8) of Cluster-02	Distance from centre (1, 2) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	5	9	C1
A2(2, 5)	5	6	4	C3
A3(8, 4)	12	7	9	C2
A4(5, 8)	5	0	10	C2
A5(7, 5)	10	5	9	C2
A6(6, 4)	10	5	7	C2
A7(1, 2)	9	10	0	C3
A8(4, 9)	3	2	10	C2



For Cluster-01:

We have only one point A1(**2, 10**) in Cluster-01.
So, cluster center remains the same.

For Cluster-02:

Centre of Cluster-02
 $= ((8 + 5 + 7 + 6 + 4)/5, (4 + 8 + 5 + 4 + 9)/5)$
 $= (\mathbf{6, 6})$

For Cluster-03:

Centre of Cluster-03
 $= ((2 + 1)/2, (5 + 2)/2)$
 $= (\mathbf{1.5, 3.5})$

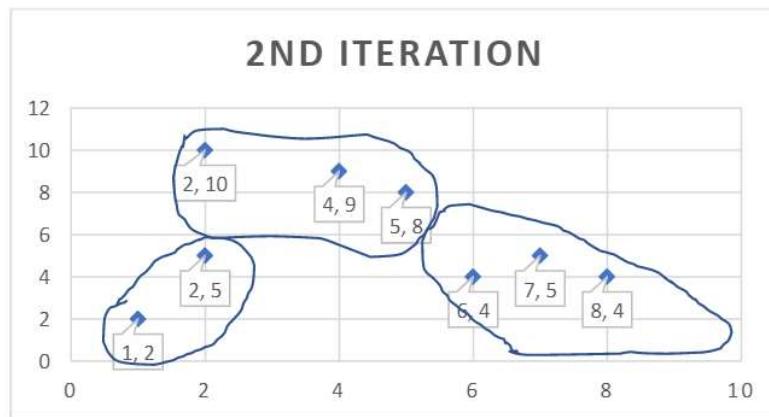
This is completion of Iteration-01.

2nd Iteration

- We calculate the distance of each point from each of the center of the three clusters **(2, 10), (6, 6), (1.5, 3.5)**
- The distance is calculated by using the given distance function.
-

Given Points	Distance from center (2, 10) of Cluster-01	Distance from center (6, 6) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	0	8	7	C1
A2(2, 5)	5	5	2	C3
A3(8, 4)	12	4	7	C2
A4(5, 8)	5	3	8	C2
A5(7, 5)	10	2	7	C2
A6(6, 4)	10	2	5	C2
A7(1, 2)	9	9	2	C3
A8(4, 9)	3	5	8	C1

Note that A8(4,9) has jumped from cluster c2 to cluster c1.
So, we should do another iteration.



For Cluster-01:

Center of Cluster-01

$$= ((2 + 4)/2, (10 + 9)/2) = (3, 9.5)$$

For Cluster-02:

Center of Cluster-02

$$= ((8 + 5 + 7 + 6)/4, (4 + 8 + 5 + 4)/4) = (6.5, 5.25)$$

For Cluster-03:

Center of Cluster-03

$$= ((2 + 1)/2, (5 + 2)/2) = (1.5, 3.5)$$

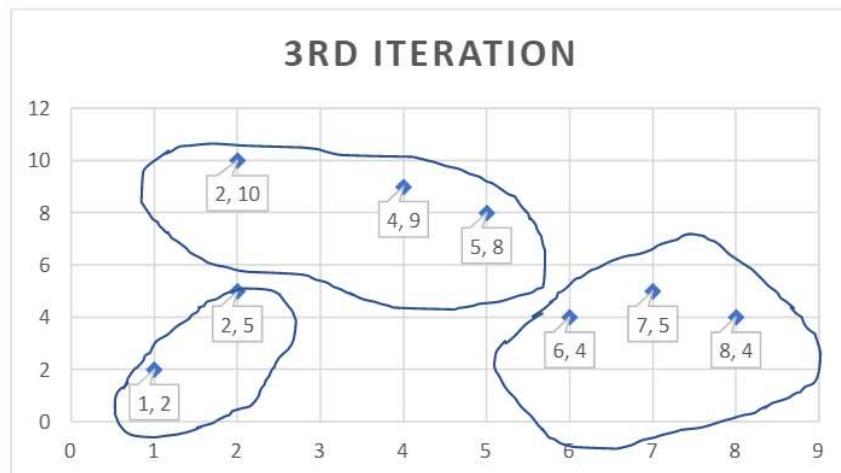
This is completion of Iteration-02.

3rd iteration,

- We calculate the distance of each point from each of the centre of the three clusters **(3, 9.5), (6.5, 5.25), (1.5, 3.5)**
- The distance is calculated by using the given distance function.

Given Points	Distance from center (3, 9.5) of Cluster-01	Distance from center (6.5, 5.25) of Cluster-02	Distance from center (1.5, 3.5) of Cluster-03	Point belongs to Cluster
A1(2, 10)	1.5	9.25	7	C1
A2(2, 5)	5.5	4.75	2	C3
A3(8, 4)	10.5	2.75	7	C2
A4(5, 8)	3.5	4.25	8	C1
A5(7, 5)	8.5	0.75	7	C2
A6(6, 4)	8.5	1.75	5	C2
A7(1, 2)	9.5	8.75	2	C3
A8(4, 9)	1.5	6.25	8	C1

Note that cluster A4(5, 8) has jumped from C2 to C1. So, we should do another iteration.



Repeat the process until no element jumps from one cluster to another cluster,

Strengths, Weaknesses, Time and space complexity of K-Means

Strengths

1. It is simple and useful for all varieties of data types.

2. Even though multiple iterations are performed, it is quite efficient.
3. Bisecting k- means is also efficient.

Weakness

1. It cannot handle non-globular clusters, clusters of different sizes and densities even though it can find pure sub-clusters.
2. Outliers cannot be clustered.
3. K-means is restricted for the notion of centroid.

Time complexity

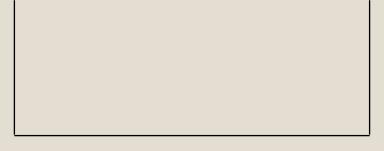
$$O(I^*K^*m^*n)$$

Where,

- I- Number of Iteration for making perfect clusters
- K- Number of clusters
- m- Number of attributes
- n- Number of elements

Space complexity

$$O((m+K)n)$$



FEATURE SELECTION

UNIT – III

Need for feature selection

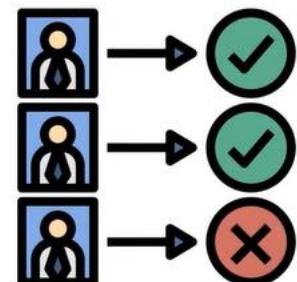
Students Dataset

Name	Marks	Address	Race	Religion	Attendance



60%

All the features the model does not give
good accuracy



80%

With relevant and best features the model
performs well

What is feature selection?

- Selecting Optimal and best features for Machine Learning Model
- Removing irrelevant or partially relevant and redundant features from the data.
- Important features of feature selection
 - Reduces the curse of dimensionality
 - Minimizes the cost of computation
 - Helps in learning the model
 - Helps in achieving good accuracy.

Feature Subset Selection

- Redundant and irrelevant information removal causes no information loss.
- Redundant features
 - Duplicate much or all of the information contained in one or more other attributes.
 - Eg: The purchase price of a product and the amount of sales tax paid
- Irrelevant features
 - Contain no useful information for the machine learning task.
 - Eg: student id number in the prediction of student cgpa
- Redundant and irrelevant features can reduce classification accuracy and the quality of clusters.

Feature selection

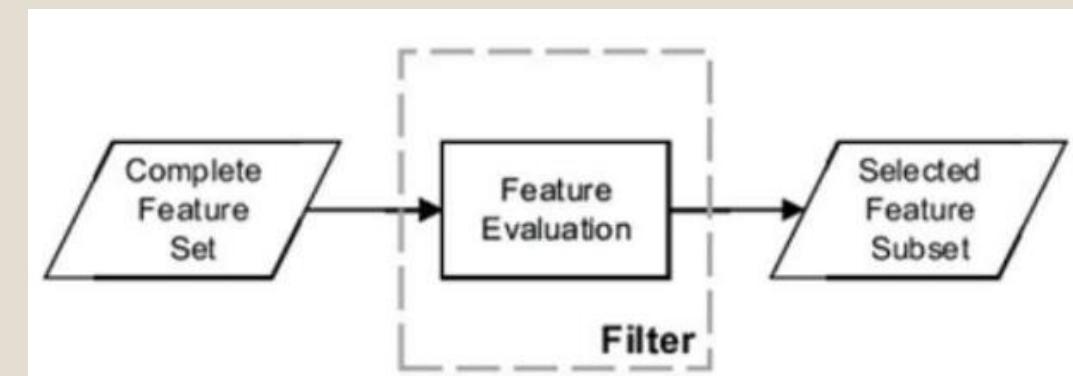
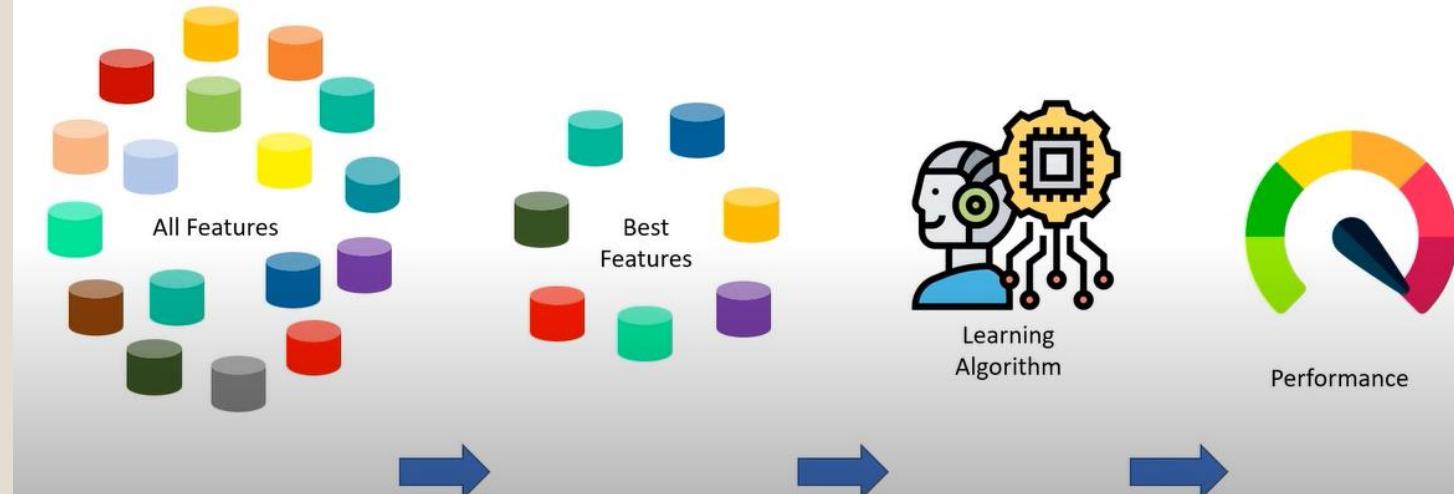
- Can be done with domain knowledge,
- to select the best set features a systematic approach is needed
- The ideal approach
 - Try all possible subsets of features as input based on the algorithm
 - Disadvantage: the number of subsets involving n attributes is 2^n .
 - Therefore this approach is impractical and alternate strategies are required.

Feature Selection Standard Approaches

- **Embedded approaches**
 - Feature selection occurs naturally as part of machine learning algorithm
- **Filter approaches**
 - Features are selected before the machine learning algorithm is run using some approach
 - Eg: attributes whose pair wise correlation is as low as possible.
- **Wrapper approaches**
 - Uses ML algorithm as a black box to find the best subset of attributes
- We will focus on only filter and wrapper approaches as embedded approach is algorithm specific

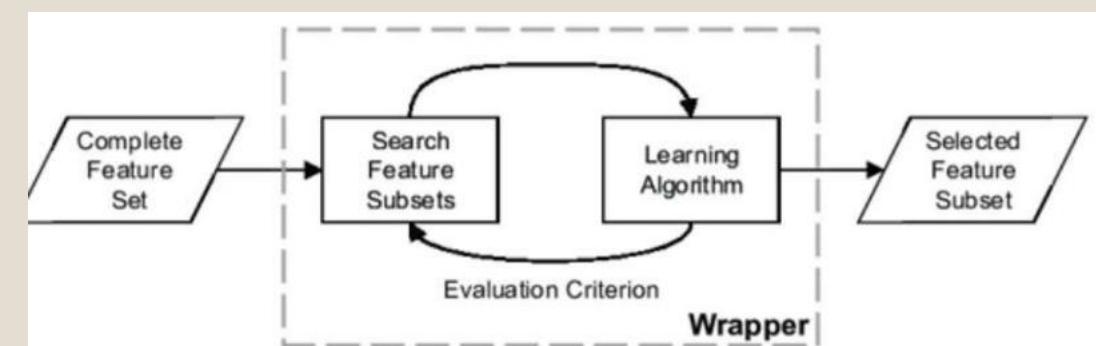
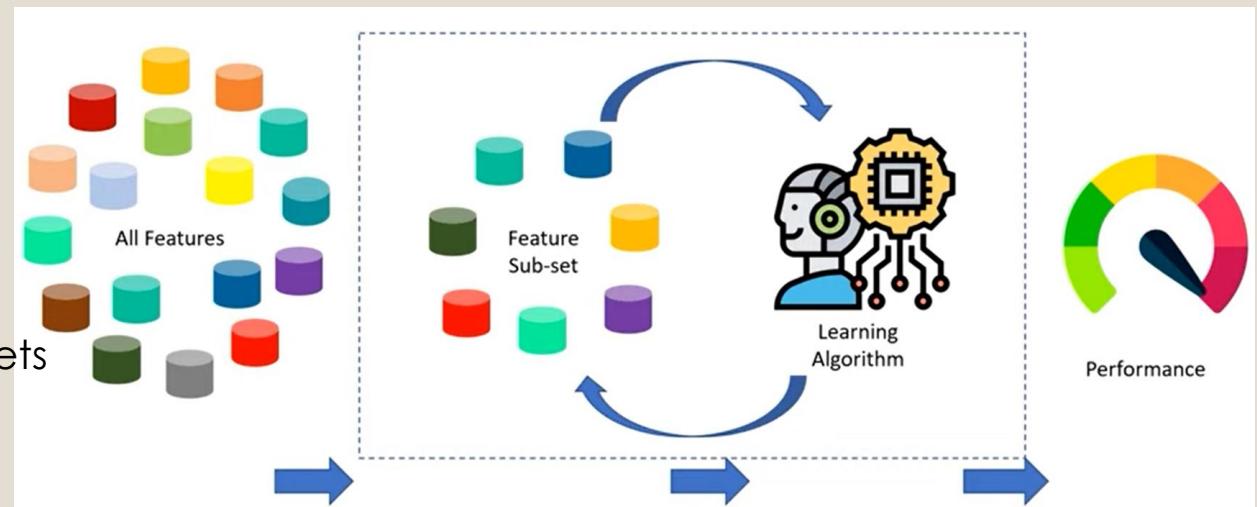
Filter method

- Filter Techniques
 - Domain Knowledge
 - Variance
 - Finding correlation between features
 - Entropy and information gain
 - Attributes with high information gain can be retained for building the model
 - Correlation of a feature with target variable.
 - Other techniques – F-Score, Chi-Square, ANOVA



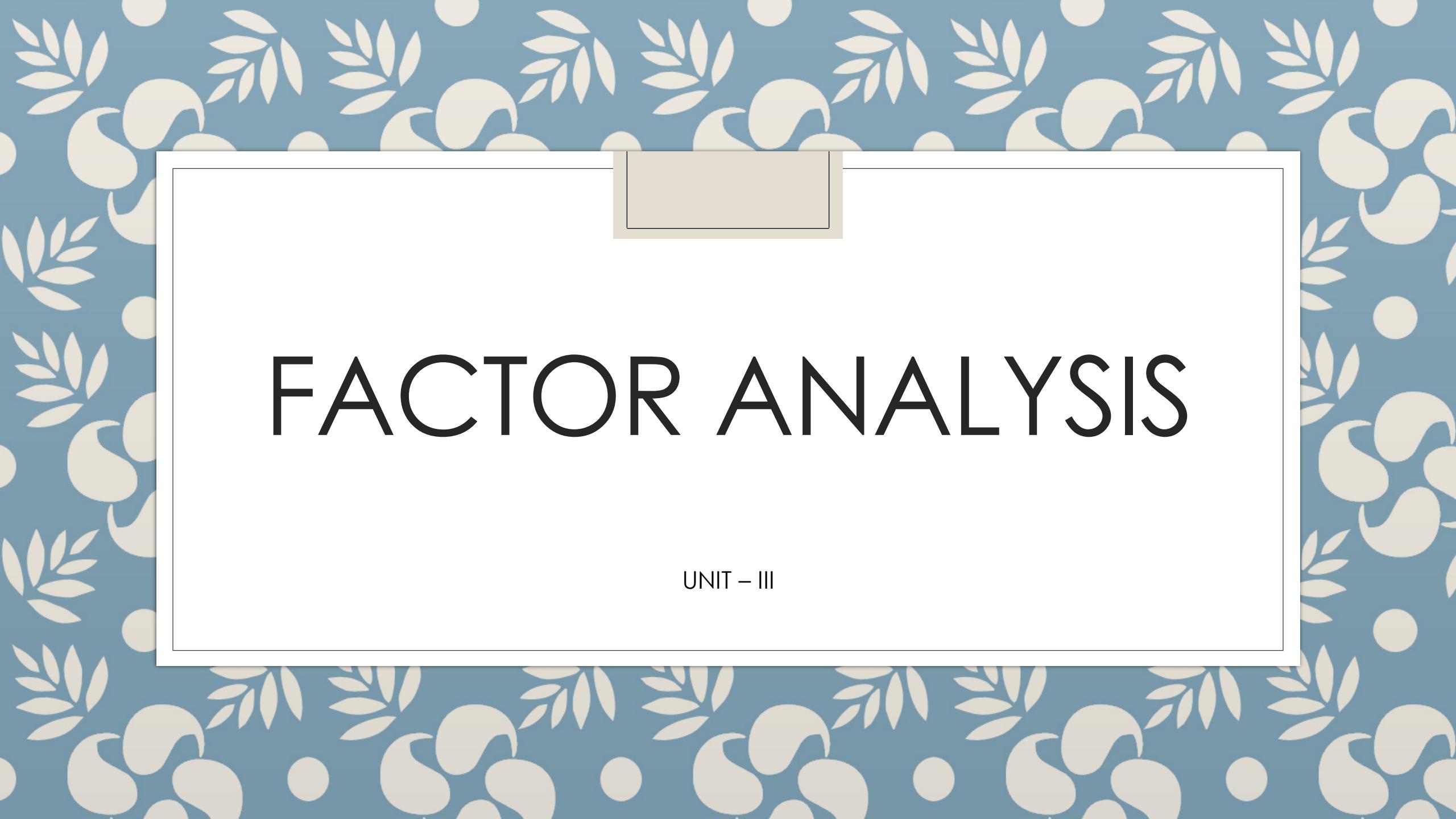
Wrapper Method

- train the model on some subsets and retain the subsets which give maximum model accuracy
- the wrapper techniques
- **Stepwise Forward Selection**
 - starts with an empty set
 - best of the original feature set are added to the reduced set
 - At every iteration, best of the remaining features is added to the set.
- **Stepwise Backward Elimination**
 - Starts with full set of variables
 - At each step, it removes the worst attribute and continues.
- **Recursive Feature Elimination**
 - Computes the weights and removes features with smallest weights
 - Recompute the weights on reduced data
 - Repeats the above process until performance improves.



Advantages of Feature selection

- Filter Method
 - Better Computational Complexity
 - Independent of Classifier
 - Fast and Scalable
- Wrapper Method
 - Interacts with the classifier
 - Model is feature dependent
 - Minimizes computational cost

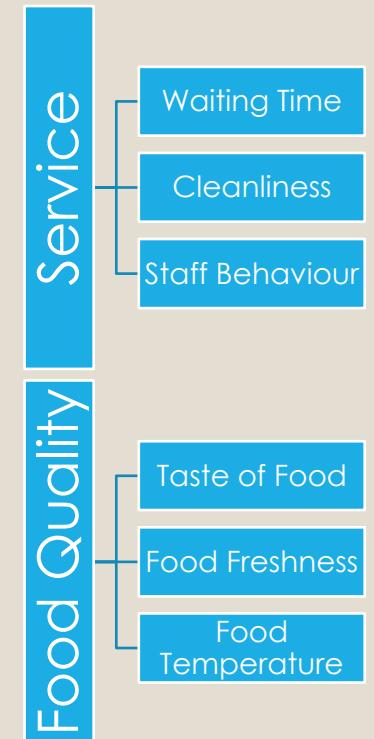
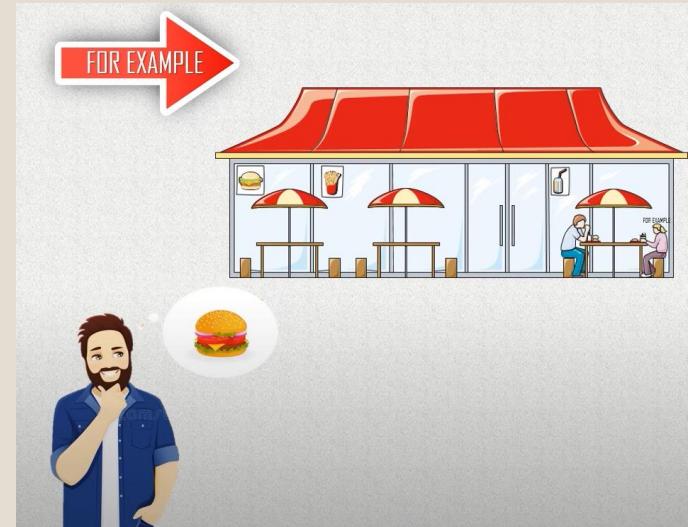


FACTOR ANALYSIS

UNIT – III

What is Factor Analysis?

- Factor analysis is a technique that is used to reduce a large number of variables into fewer number of factors.
- It is a way to condense the data in many variables into just a few variables.
- Factor analysis is also called Dimension Reduction.
- It is an example of Latent variable model.
- Example :
 - You are foodie and you want to go to a restaurant. the restaurant data is available in the form of Waiting Time, Cleanliness, Staff Behavior, Taste of Food, Food Freshness, Food Temperature.
 - Too many variables.. Two factors you care about restaurant are Service and Food Quality
 - The variables in the reviews can be grouped into two latent variables as shown below.



Latent Variables

- Latent Variables are variables that are not directly observed but are inferred from other variables
- Mathematical models that aim to explain observed variables in terms of latent variables are called latent variable models.
- Factor analysis is a latent variable model.
- Examples of latent variable: quality of life

More about factor analysis

- Assumptions in Factor Analysis
 - There are no outliers in the data
 - Sample size is supposed to be greater than the factor
 - Variables must be interrelated
 - Numeric variables are expected.
- Purpose of factor analysis
 - Data Reduction
 - Latent Variable Discovery
 - Simplification of items into subsets of concepts

Types of Factor Analysis

- Exploratory Factor Analysis (EFA)
 - used to discover underlying structure in the data.
 - uses correlation matrix
 - gets insights of the data.
- Confirmatory Factor Analysis
 - It is based on the insights derived in EFA
- Issues with factor analysis

Issues with factor analysis

1. use PCA or FA
2. How to interpret the results?
3. How many factors?

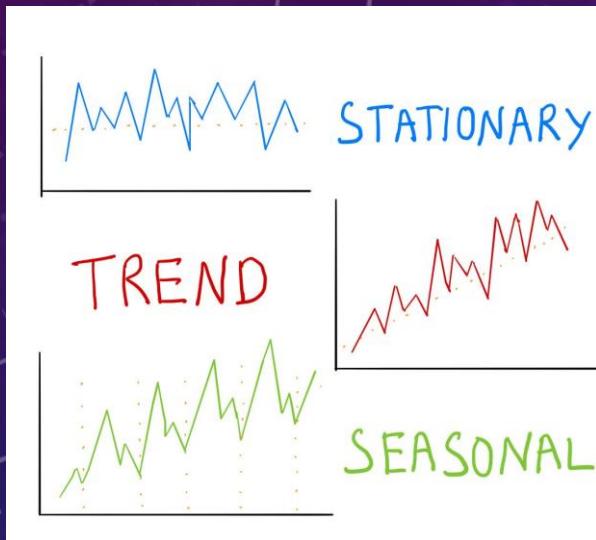
PCA	FA
seeks to identify variables that are composites of the observed variables.	FA assumes the existence of latent factors underlying the observed data.
Total variance is taken into account to derive the factors	Only common variance is taken into consideration

- **How to interpret?**
 - with the help of factor loading
 - Factor loading is the correlation coefficient for the variable and factor

- **How many factors?**
 - Latent Root Criterion : Eigen Values > 1

TIME SERIES ANALYSIS

UNIT - III



WHY TIME SERIES?

- In time series analysis, we have only one variable ... **TIME**
- We can analyse this time series data in order to extract meaningful statistics and other characteristics.
 - Eg: coffee sales in next month.
- The simplest form of data is a long-ish series of continuous measurements at equally spaced time points
- that is
 - observations are made at distinct points in time, these time points being equally spaced
 - and, the observations may take values from a continuous distribution.

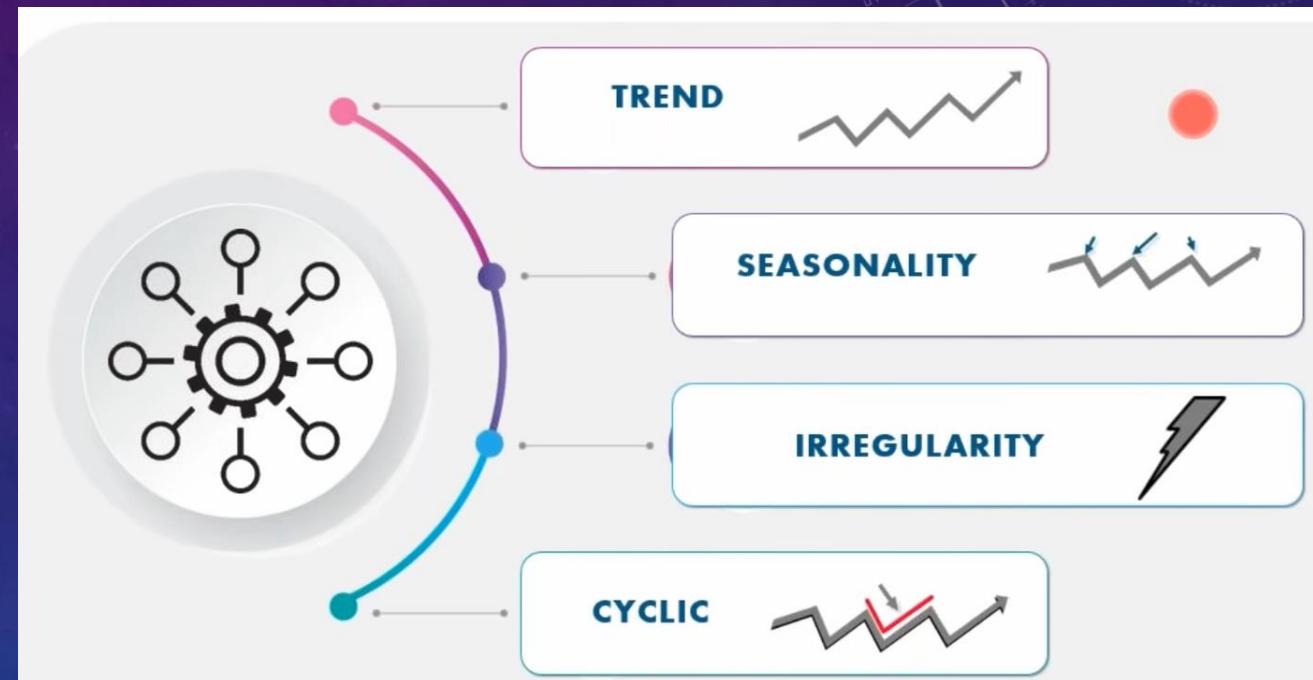
WHAT IS TIME SERIES?

- A Time Series (TS) is a set of observations taken at specified times usually at equal intervals.
- It is used to predict the future values based on the previous observed values.
- Importance of Time Series
 - Business forecasting
 - Understand past behaviour
 - Plan future
 - Evaluate current accomplishments
- Examples of time series
 - Economics and Finance, Environmental Modelling, Meteorology and Hydrology
 - Demographics, Medicine, Quality Control



COMPONENTS OF TIME SERIES

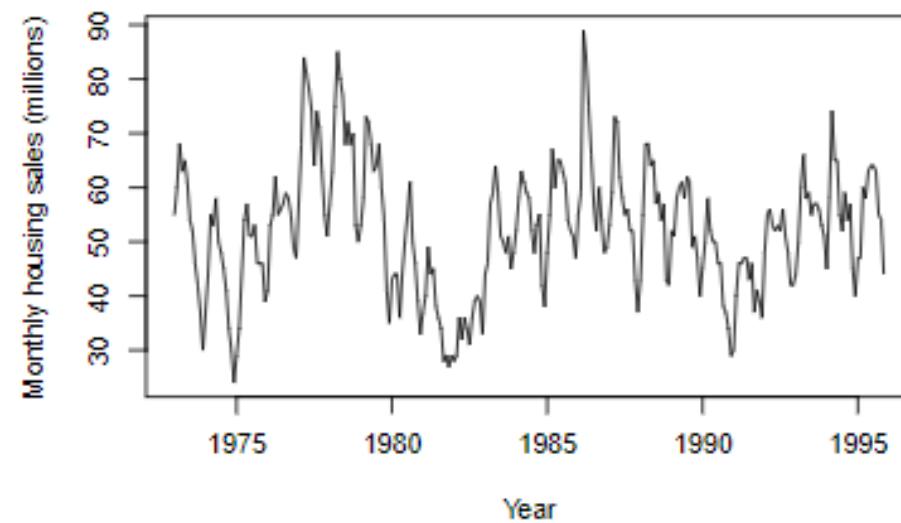
- Real world data often governed by a **trend** and they might have **cyclical** or **seasonal** components in addition to the **irregular**/remainder component.



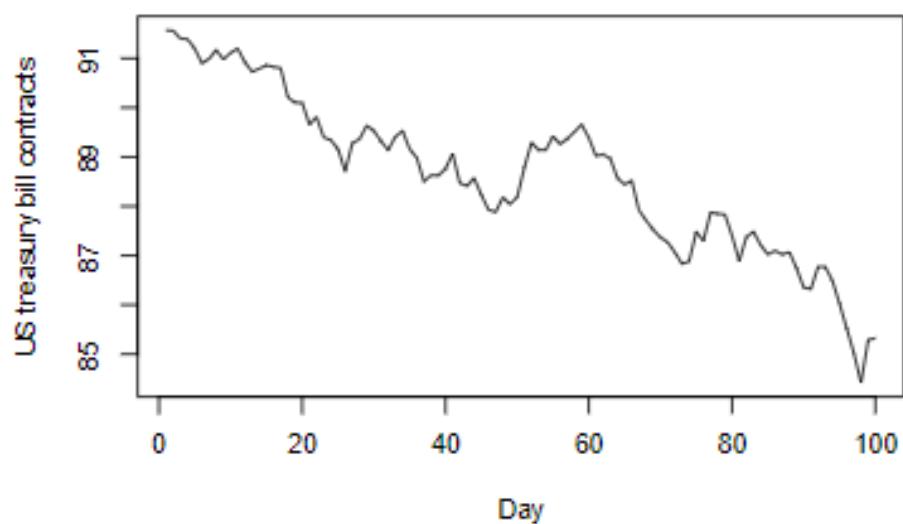
COMPONENTS OF TIME SERIES

- ▶ **Trend component** - a long-term increase or decrease in the data which might not be linear. Sometimes the trend might change direction as time increases.
- ▶ **Cyclical component** - exists when data exhibit rises and falls that are not of fixed period. The average length of cycles is longer than the length of a seasonal pattern. In practice, the trend component is assumed to include also the cyclical component. Sometimes the trend and cyclical components together are called as trend-cycle.
- ▶ **Seasonal component** - exists when a series exhibits regular fluctuations based on the season (e.g. every month/quarter/year). Seasonality is always of a fixed and known period.
- ▶ **Irregular component** - a stationary process.

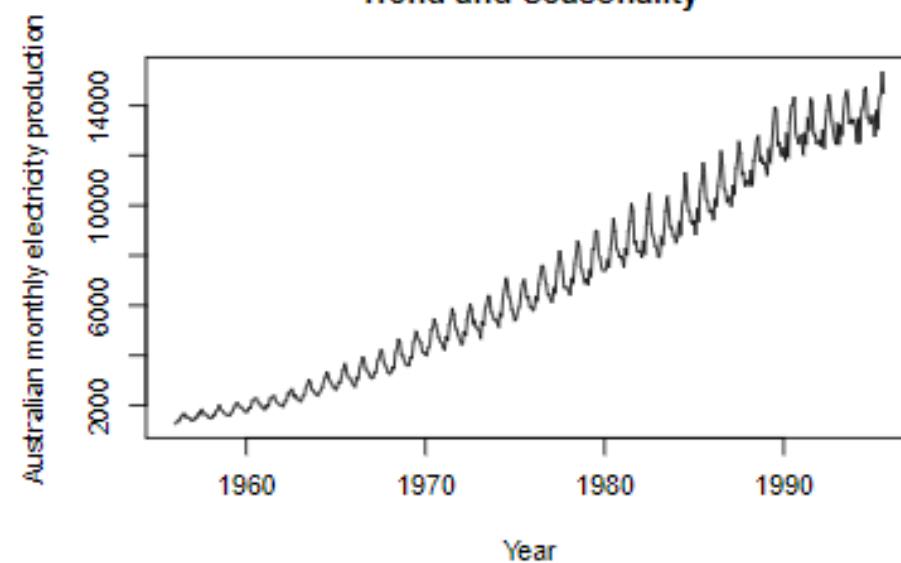
Seasonality and Cyclical



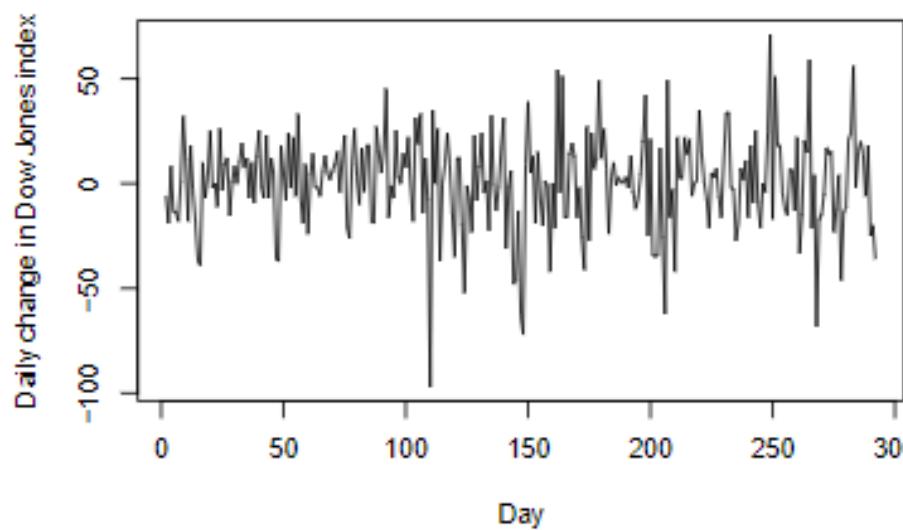
Trend



Trend and Seasonality



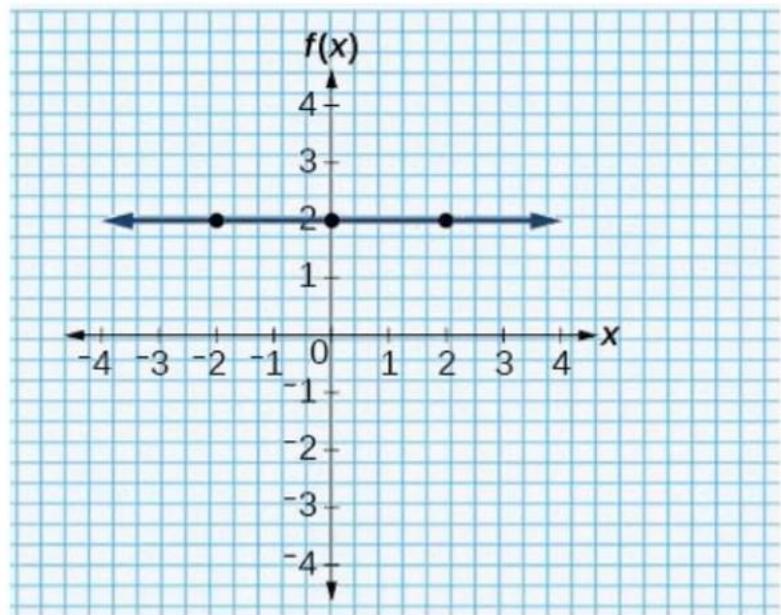
No Deterministic Components



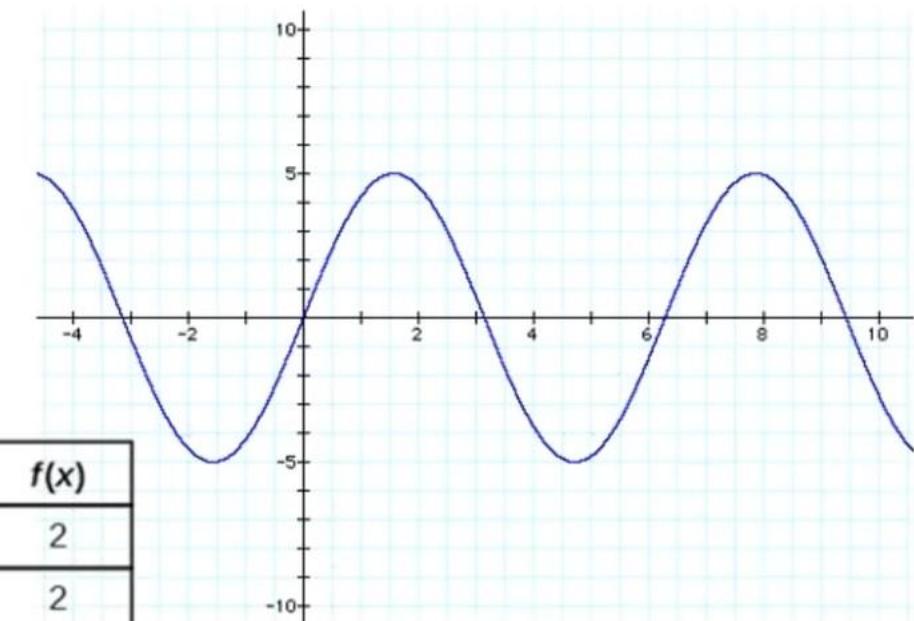
WHEN NOT TO USE TIME SERIES ANALYSIS



Values are constant



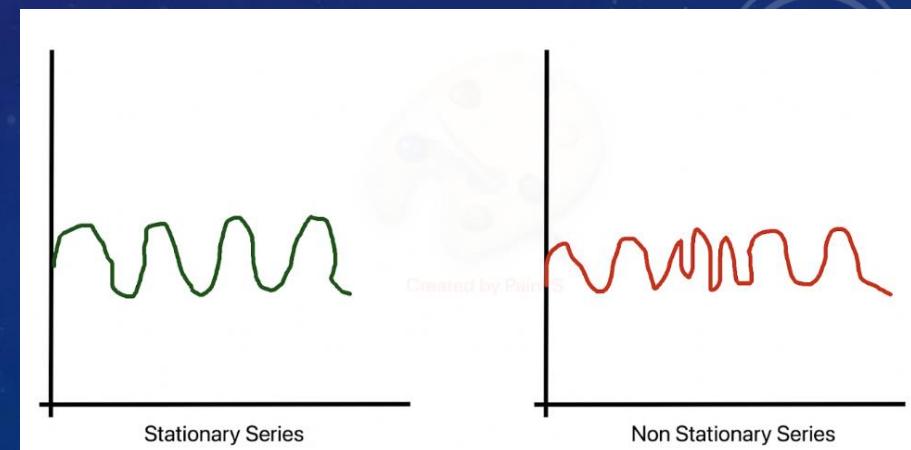
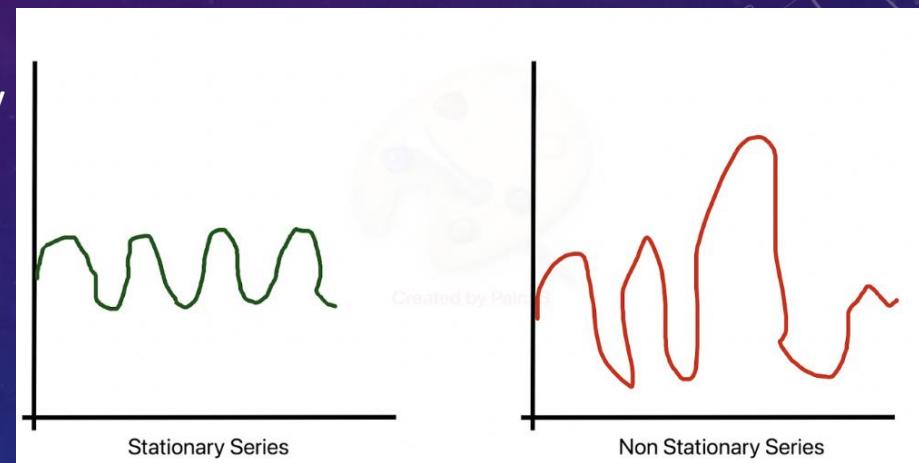
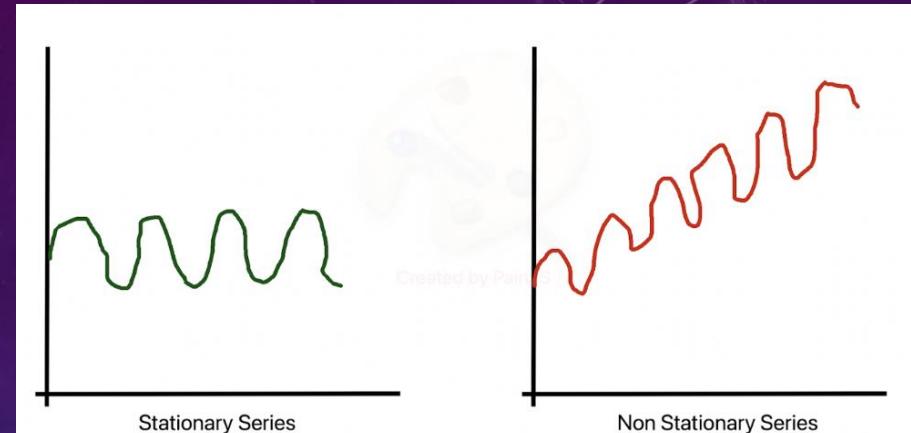
Values in the form of functions



x	$f(x)$
-2	2
0	2
2	2

STATIONARITY

- Time series requires the data to be stationary.
- A necessary condition to take advantage of models for any time series
- For a time series to be stationary, it should satisfy 3 conditions
 - Mean (μ) is constant
 - Standard deviation (σ) is constant
 - Seasonality does not exist (autocovariance does not depend on time)
- In most cases, these conditions can be visually analysed by studying the plot against time



CHECKING STATIONARITY IN PYTHON

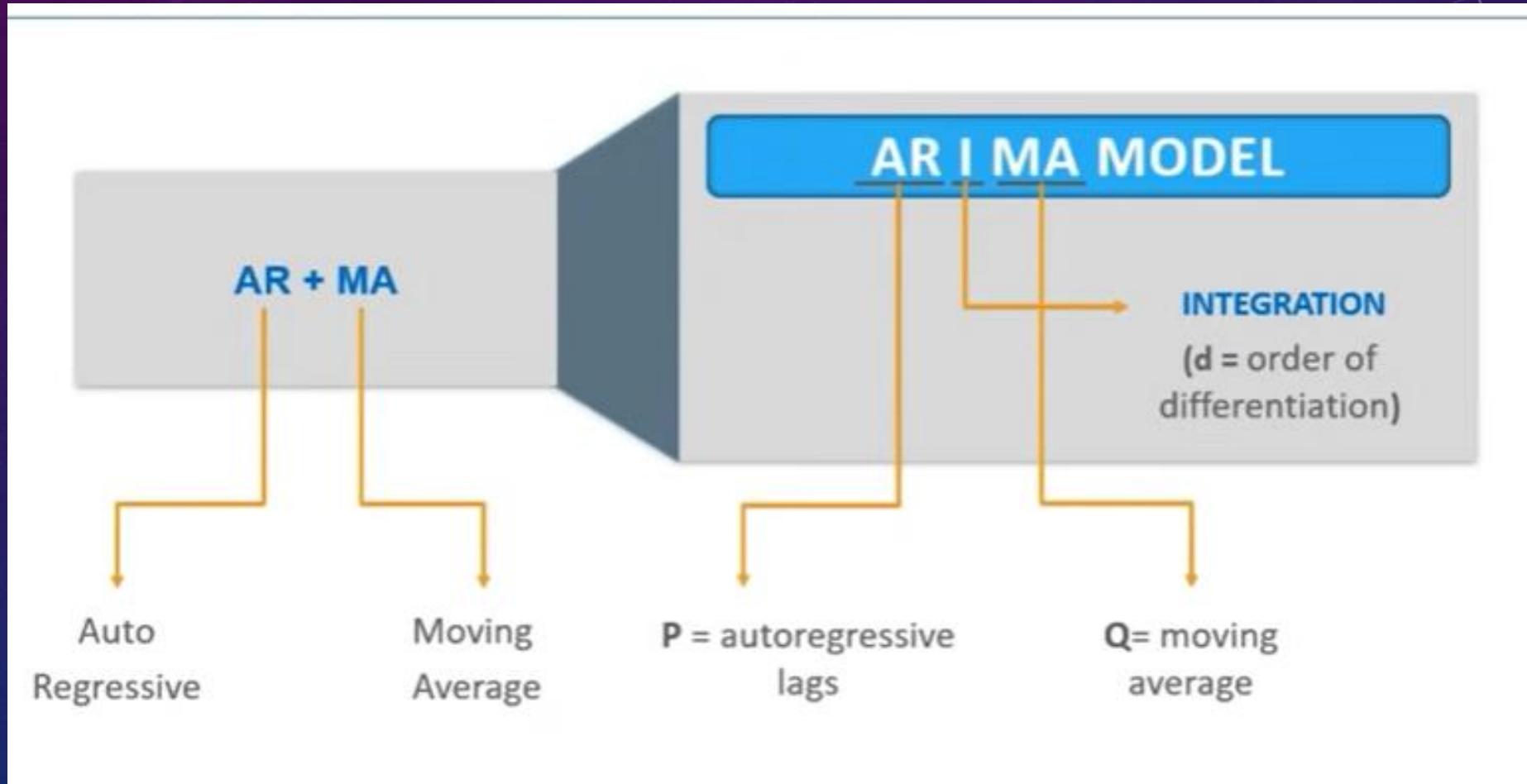
1. Rolling Statistics:

- A visual technique
- Plot the moving average or the moving variable and check whether it varies with time

2. ADCTF test:

- Null hypothesis is that the TS is non-stationary
- The test results comprise of a test statistic and some critical values

ARIMA MODEL



AUTOREGRESSIVE MODELS (AR)

- Assume that a current value of the series is linearly dependent upon its previous value, with some error. Then we could have the linear relationship

$$X_t = \alpha X_{t-1} + \epsilon_t$$

where ϵ_t is a white noise time series.

- This model is called an autoregressive (AR) model, since X is regressed on itself. Here the lag of autoregression is 1.
- More generally we could have an autoregressive model of order p, an AR(p) model, defined by

$$X_t = \sum_{i=1}^p \alpha_i X_{t-i} + \epsilon_t.$$

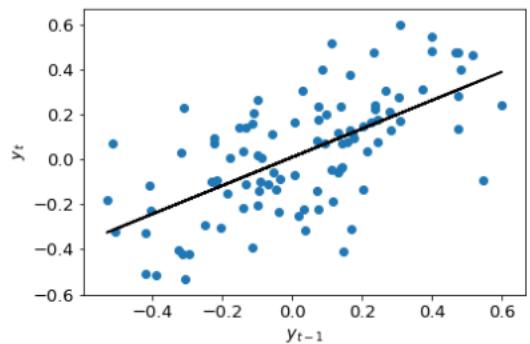
At first sight, the AR(1) process

$$X_t = \alpha X_{t-1} + \epsilon_t$$

AR MODELS

AR(1) model :

$$y_t = a_1 y_{t-1} + \epsilon_t$$



AR(1) model :

$$y_t = a_1 y_{t-1} + \epsilon_t$$

AR(2) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \epsilon_t$$

AR(p) model :

$$y_t = a_1 y_{t-1} + a_2 y_{t-2} + \dots + a_p y_{t-p} + \epsilon_t$$

MOVING AVERAGE

- Another possibility is to assume that the current value of the series is a weighted sum of past white noise terms, so for example that

$$X_t = \epsilon_t + \beta \epsilon_{t-1}$$

- such a model is called a moving average (MA) model, since X is expressed as a weighted average of past values of the white noise series.
- Here the lag of the moving average is q.
- We can think of the white noise series as being innovations or shocks: new stochastically uncorrelated information which appears at each time step, which is combined with other innovations to provide the observable series X.
- More generally we could have a moving average model of order q, an MA(q) model, defined by

$$X_t = \epsilon_t + \sum_{j=1}^q \beta_j \epsilon_{t-j}.$$

MA models

Moving average (MA) model

MA(1) model :

$$y_t = m_1 \epsilon_{t-1} + \epsilon_t$$

MA(2) model :

$$y_t = m_1 \epsilon_{t-1} + m_2 \epsilon_{t-2} + \epsilon_t$$

MA(q) model :

$$y_t = m_1 \epsilon_{t-1} + m_2 \epsilon_{t-2} + \dots + m_q \epsilon_{t-q} + \epsilon_t$$

An autoregressive moving average process ARMA(p,q) is defined by

$$X_t = \sum_{i=1}^p \alpha_i X_{t-i} + \sum_{j=0}^q \beta_j \epsilon_{t-j}$$

where $\beta_0 = 1$.

ARMA models

Autoregressive moving-average (ARMA) model

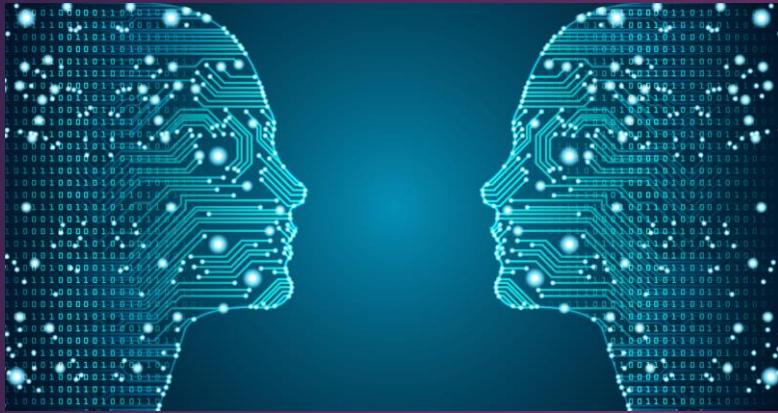
- ARMA = AR + MA

ARMA(1,1) model :

$$y_t = a_1 y_{t-1} + m_1 \epsilon_{t-1} + \epsilon_t$$

ARMA(p, q)

- p is order of AR part
- q is order of MA part



Natural Language Processing

UNIT – IV

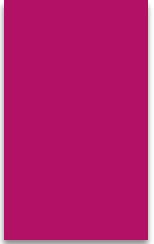
Contents

- ▶ **Natural Language Processing / Text mining**
 - ▶ Introduction.
 - ▶ Applications.
 - ▶ Chatbots, virtual agents (Alexa, Google Assistant, Siri). Importance, Applications,
 - ▶ NLP Subproblems.
 - ▶ Components of Natural Language.
 - ▶ Steps to get text data into workable format.
 - ▶ Terms Frequency, Inverse Document Frequency,
 - ▶ Bag of Words,
 - ▶ ngram,
 - ▶ One hot encoding.
 - ▶ Notion of corpus. Intro to NLTK

NLP

- ▶ is among the hottest topic in the field of data science.
- ▶ Companies are putting tons of money into research in this field.
- ▶ Everyone is trying to understand NLP and its applications to make a career around it.
- ▶ Every business out there wants to integrate it into their business somehow.

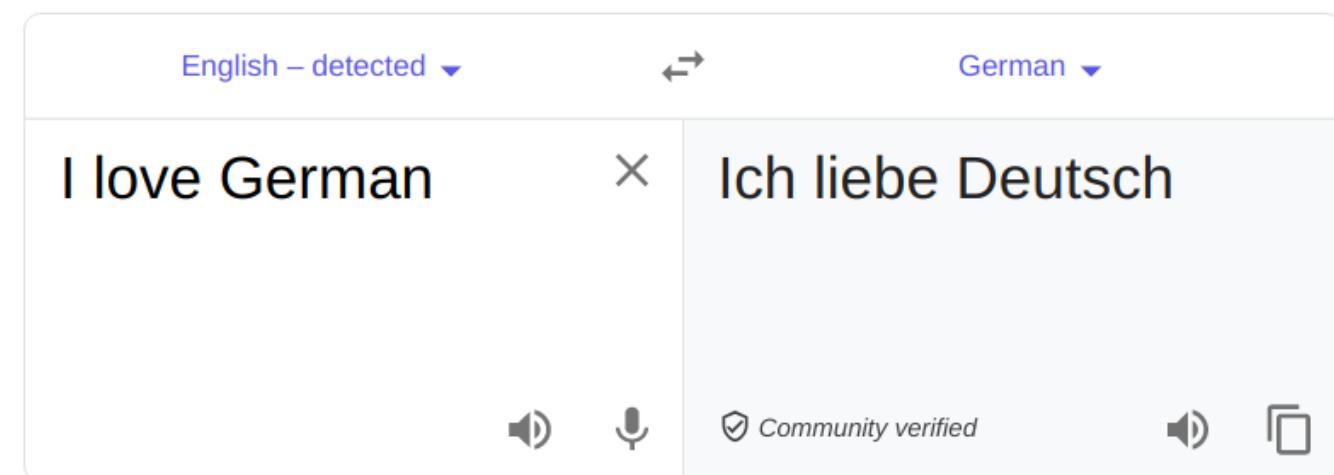
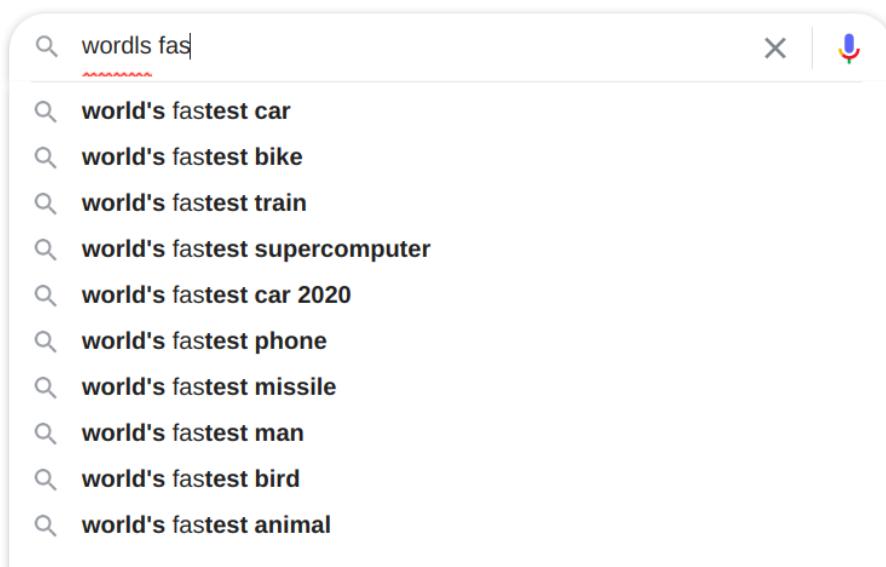
Why?



Are you using NLP these days?

Search Autocorrect and Autocomplete – Language Translator

Google

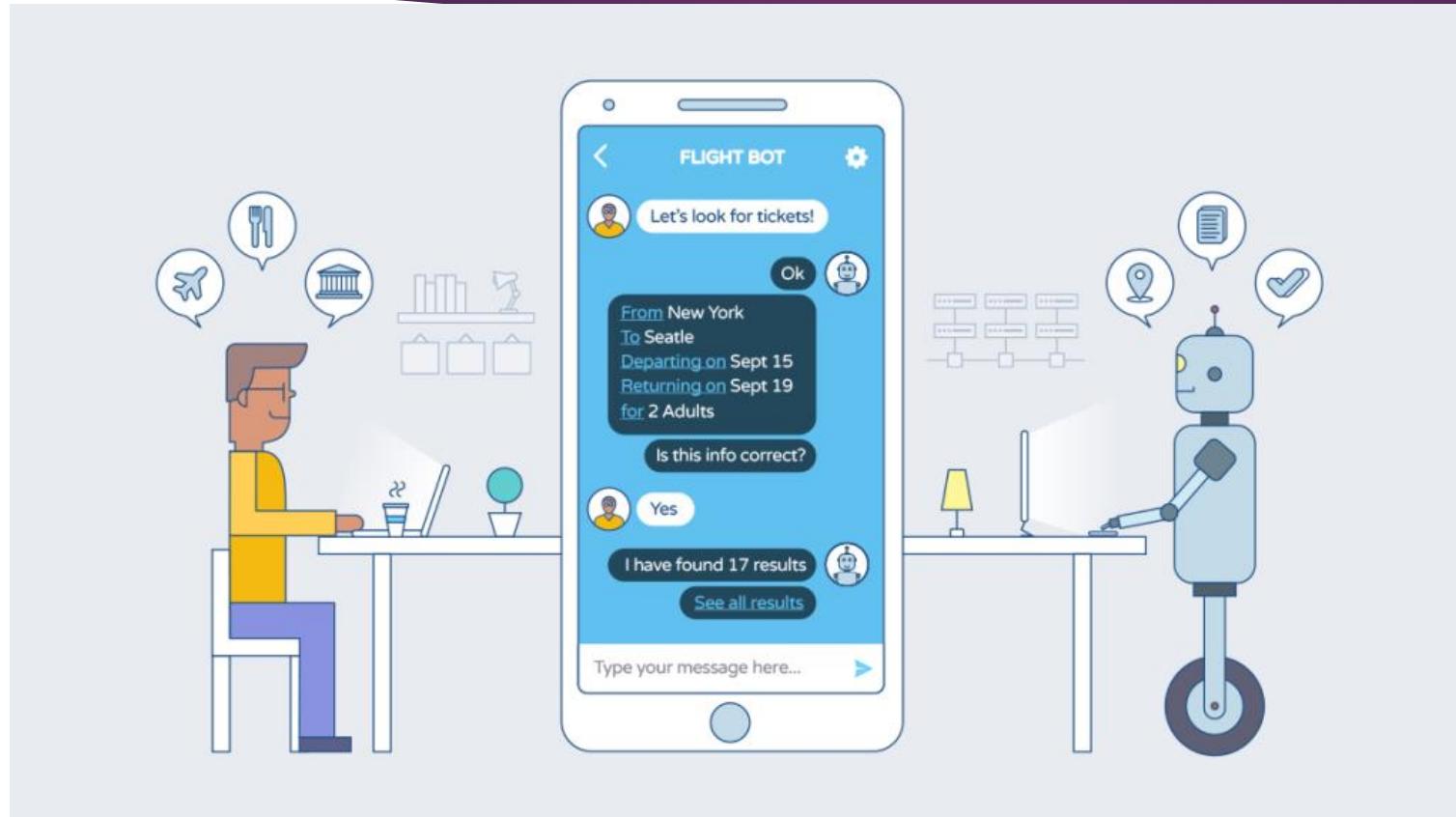


Social media monitoring



- ▶ More people these days have started using social media for posting their thoughts about a particular product, policy, or matter.
- ▶ These could contain some useful information about an individual's likes and dislikes.
- ▶ Analyzing this unstructured data can help in generating valuable insights. NLP comes to rescue here too.
- ▶ Various NLP techniques are used by companies to analyze social media posts and know what customers think about their products.
- ▶ Companies are also using social media monitoring to understand the issues and problems that their customers are facing by using their products.

Chatbots



Modern Conversational Agents can

- Answer questions
- Book flights
- Find Restaurants
- functions for which they rely on a much more sophisticated understanding of the user's intent

Survey Analysis

- ▶ Surveys are an important way of evaluating a company's performance.
 - ▶ to get customer's feedback on various products.
 - ▶ useful in understanding the flaws and help companies improve their products.
- ▶ NLP is used to analyze the surveys and generate insights from them, like knowing the sentiments of users analyzing product reviews to understand the pros and cons



Targeted Advertising – Hiring and Recruitment

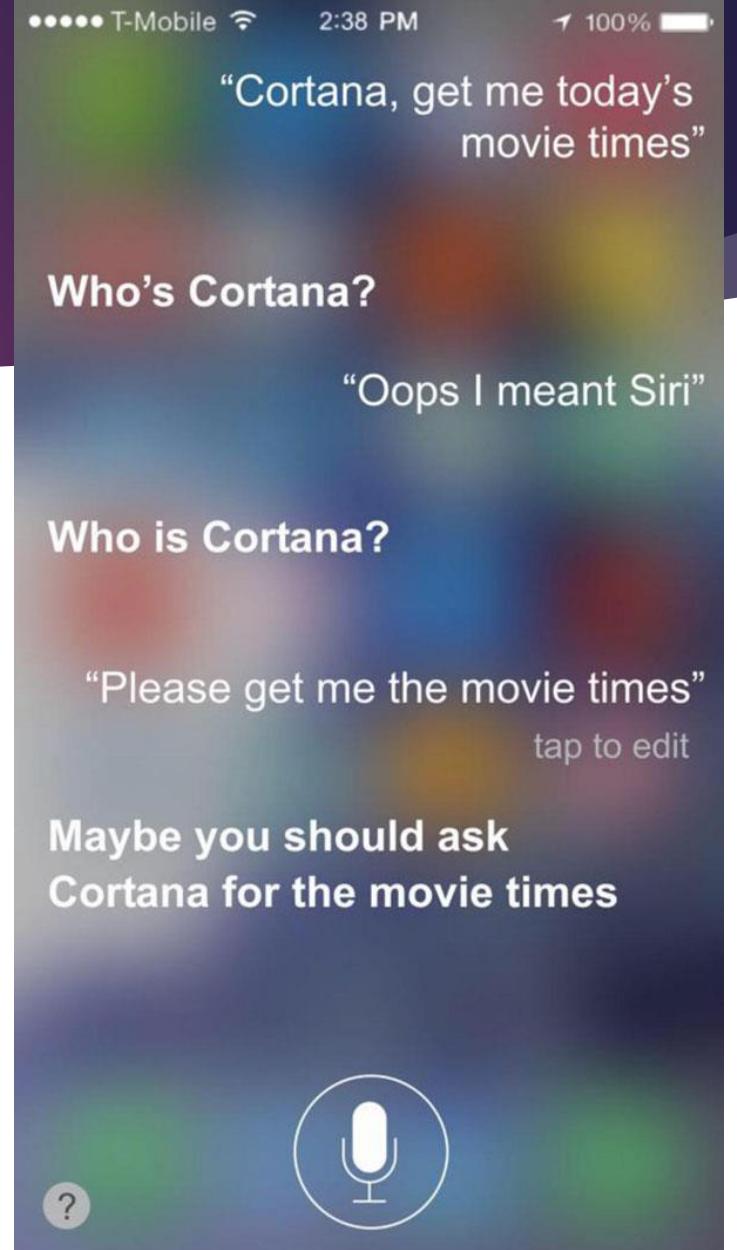
Targeted advertising is a type of online advertising where ads are shown to the user based on their online activity.



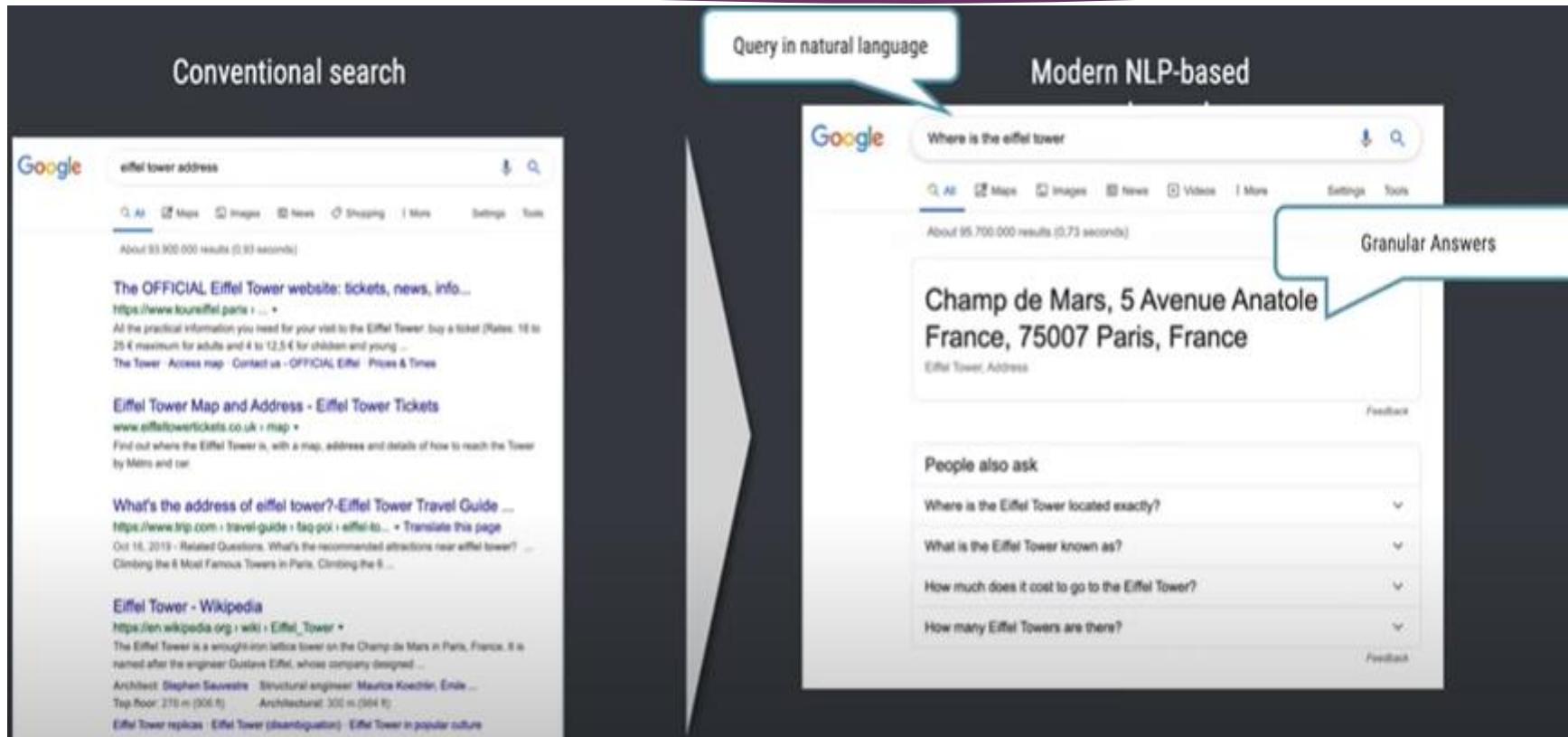
it saves companies a lot of money
relevant ads are shown only to the potential customers.



Voice Assistants

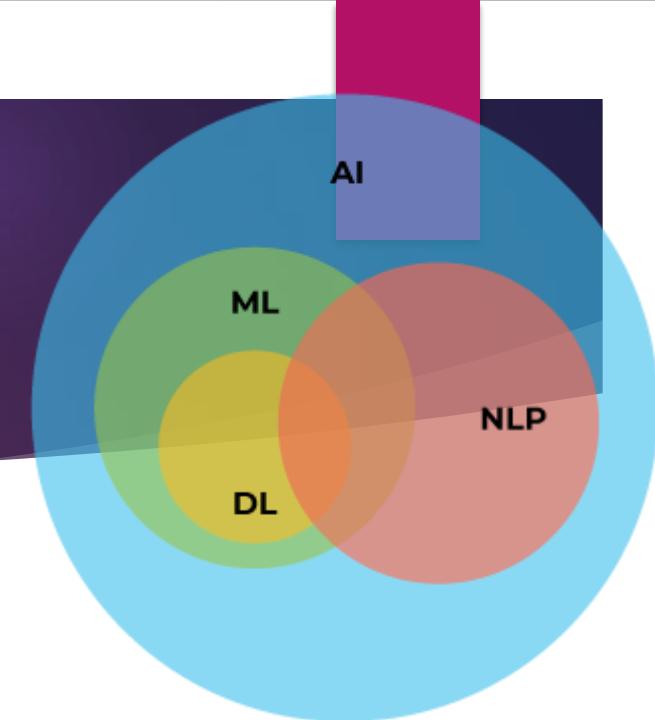


Conventional vs. NLP-based search



What is NLP?

- ▶ Natural language processing is a sub-field of linguistics, computer science and AI concerned with the interactions between computers and human language
- ▶ NLP makes computers understand complex language structure and retrieve meaningful pieces of information from it
- ▶ Modern challenges in NLP involve
 - ▶ speech recognition,
 - ▶ natural language understanding and
 - ▶ natural language generation



Applications of NLP

- ▶ Text Classification
- ▶ Language Modelling
- ▶ Information Extraction
- ▶ Information Retrieval
- ▶ Conversational Agents
- ▶ Text Summarization
- ▶ Question Answering
- ▶ Machine Translation
- ▶ Topic Modelling
- ▶ Speech Recognition

Chatbots and Virtual Assistants

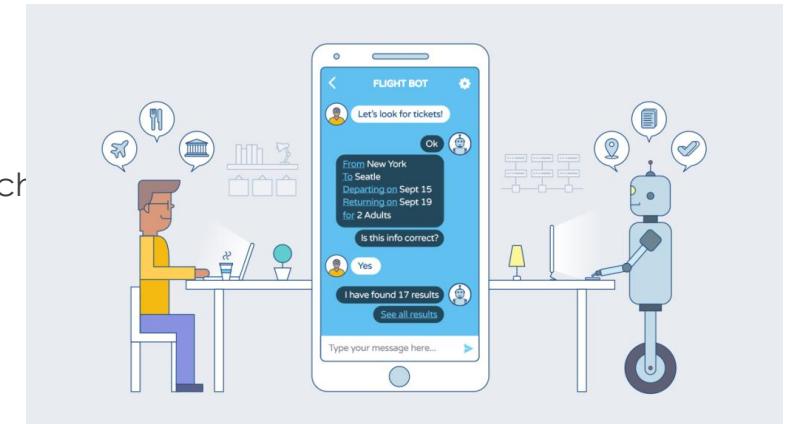


► What is a chatbot?

- ▶ It's all about the conversation.
- ▶ A chatbot is a piece of software, usually powered by artificial intelligence, with which interact.
- ▶ They are sometimes called virtual assistants, chatbox, or even chatterbox.
- ▶ Chatbots can converse with us humans through both text and voice

► Who is using this technology?

- ▶ Though chatbots are relatively old technology, but recently businesses have started putting them to effective use.
- ▶ Companies like Disney use chatbots as a brand and marketing play.
- ▶ Companies like Google use chatbots to innovate in their field.
- ▶ Companies like Burberry, Staples use chatbots as a customer service tool.
- ▶ And many more!



Chatbots and Virtual Assistants

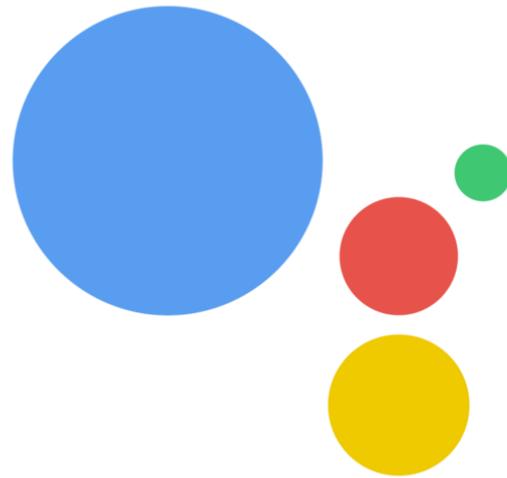
- ▶ **Artificial intelligence and chatbots**
- ▶ Most chatbots are powered by artificial intelligence, or AI.
- ▶ AI chatbots tend to be more useful, purely because they are smarter and can learn over time. This is, of course, valuable to businesses.
- ▶ Artificial intelligence in chatbots comes in many forms. The most common are natural language processing (NLP) which powers the language side of the chatbot, to machine learning (ML) which powers data and algorithms.

Some Virtual Assistants

Alexa



Google Assistant



Siri



Benefits/importance/applications of Chatbots and Virtual Assistants

- ▶ Businesses use Chatbots extensively to optimize internal business processes, boost productivity, raise revenue, and improve customer experience.
- ▶ Website chatbots increase engagement by giving quick and personalized responses. Customers who would otherwise have to wait longer to respond via traditional telephone channels benefit significantly from this.
- ▶ Because of their ease of use, chatbots have a very high adoption rate. Chatbot adoption opens up a floodgate of possibilities for businesses to better their client interaction process and operational efficiency, lowering customer service costs.
- ▶ Virtual Assistants make organizing and carrying out our daily chores easier.
- ▶ Virtual Assistants come in helpful in supporting our tasks, whether it's setting reminders and alarms, adding commissions to the calendar, making calls, or retrieving information from the internet.
- ▶ Virtual assistants can now control home intelligent gadgets such as lighting, thermostats, and music devices.

Words – What counts as a word?

- ▶ **corpus (plural corpora):** a computer-readable corpora collection of text or speech
 - ▶ For example the Brown corpus is a million-word collection of samples from 500 written English texts from different genres (newspaper, fiction, non-fiction, academic, etc.)

How many words are in the following Brown sentence?

Sentence : *He stepped out into the hall, was delighted to encounter a water brother.*

- ▶ This sentence has **13** words if we don't count punctuation marks as words,
- ▶ **15** if we count punctuation.
- ▶ Are capitalized tokens like **They** and uncapitalized tokens like **they** the same word?
- ▶ How about inflected forms like cats versus cat?
 - ▶ These two words have the same lemma cat but are different wordforms.
- ▶ A lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.
- ▶ The wordform is the full inflected or derived form of the word.

Notion of Corpus: Words – Types and Tokens

- ▶ **Types** are the number of distinct words in a corpus; if the set of words in the vocabulary is V , the number of types is the word token vocabulary size $|V|$.
- ▶ **Tokens** are the total number N of running words.
- ▶ ignore punctuation and find the number of tokens and types in the following sentence

They picnicked by the pool, then lay back on the grass and looked at the stars

16
tokens
14 types

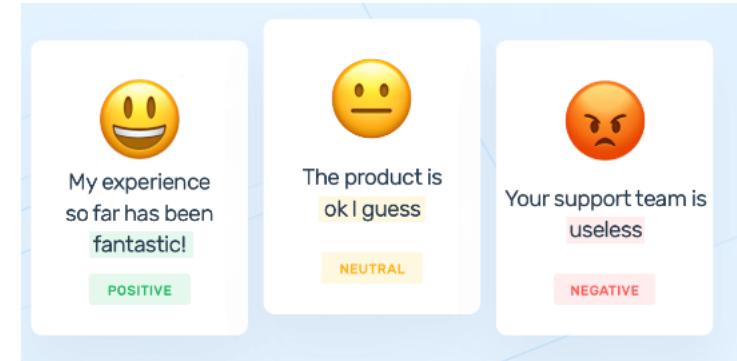
Notion of Corpus: Corpora

- ▶ Any particular piece of text that we study is produced by
 - ▶ one or more specific speakers or writers,
 - ▶ in a specific dialect of a specific language,
 - ▶ at a specific time,
 - ▶ in a specific place,
 - ▶ for a specific function.
- ▶ The most important dimension of variation is the language.
 - ▶ NLP algorithms are most useful when they apply across many languages. The world has 7097 languages.
 - ▶ It is important to test algorithms on more than one language, and particularly on languages with different properties; by contrast there is an unfortunate current tendency for NLP algorithms to be developed or tested just on English
 - ▶ Code Switching : A phenomenon which uses multiple languages in a single communicative act
- ▶ Another variations are Genre, demographic characteristics of the writer, time.

NLP Subproblems

▶ **Text Categorization**

- ▶ The goal of classification is
 - ▶ o to take a single observation,
 - ▶ o extract some useful features, and
 - ▶ o thereby classify the observation into one of a set of discrete classes.
- ▶ Examples:
 - ▶ Sentiment Analysis
 - ▶ Spam Detection
 - ▶ Authorship attribution



▶ **Machine Translation**

- ▶ translates one neural language into another language automatically.
- ▶ Can be done using rules, statistics, or neural language models
- ▶ Encoder-Decoder models are commonly employed to solve Machine Translation problems.



▶ **Text Summarization**

- ▶ aims to transform lengthy documents into shortened versions

NLP Subproblems

► Entity Recognition

- ▶ automatically scan entire articles and pull out some fundamental entities in a text and classify them into predefined categories.
- ▶ Named Entity Recognition (NER) is the process of detecting the named entities such as person names, location names, company names, etc from the text.
- ▶ It is also known as entity identification or entity extraction or entity chunking.
- ▶ With the help of NER, we can extract key information to understand the text, or merely use it to extract important information to store in a database.
- ▶ The applicability of entity detection can be seen in many applications such as

- ▶ Automated Chat!
- ▶ Content Analyzer:

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for \$37.5 million

- ▶ Consumer Insights

[organization]

[person]

[location]

[monetary value]

► Text Generation

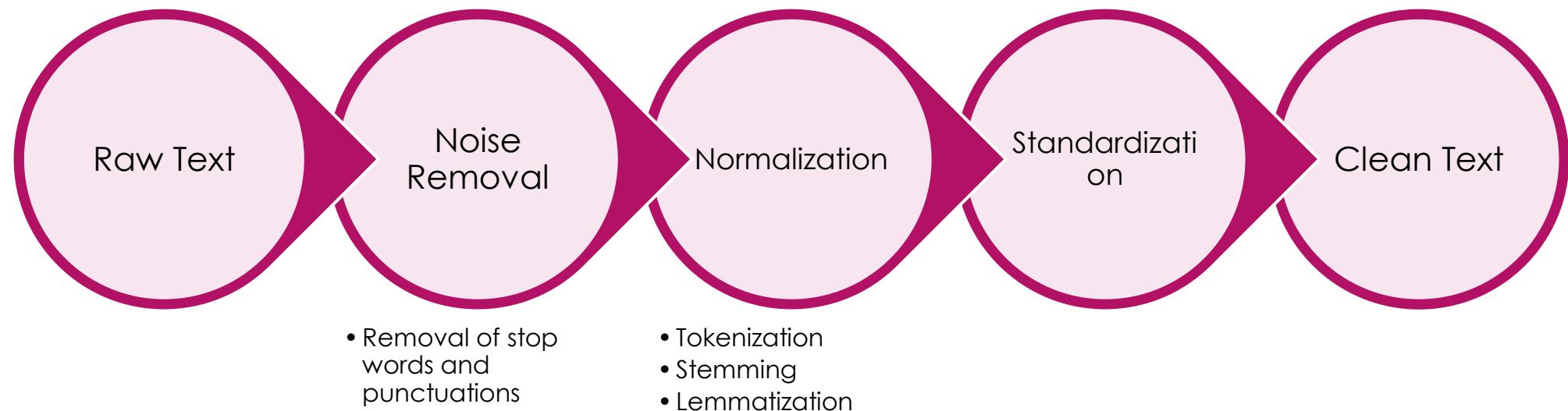
- ▶ is a subfield of NLP. It leverages knowledge in computational linguistics and AI to automatically generate natural language texts, which can satisfy certain communicative requirements.

NLP Subproblems

- ▶ Natural Language Interface
- ▶ Speech Recognition
- ▶ Text to speech

Getting text to workable format

Approximate order of steps for preprocessing text data



Noise Removal

- ▶ **Noise** : Any piece of text which is not relevant to the context of the data.
- ▶ Generally, the noisy entities are
 - ▶ Stop words,
 - ▶ punctuation marks.
- ▶ Stop words Removal
 - ▶ It is a process of removing common language articles, Pronouns and propositions such as “and”, “the” or “to” in English.
 - ▶ These words provide little or no value to NLP.
 - ▶ Stop words are removed /filtered from text for better efficiency

Stop words using NLTK

```
import nltk  
from nltk.corpus import stopwords  
print(stopwords.words('english'))  
  
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", "your", 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]
```

Removing stop words from a sentence using NLTK

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

example_sent = """We are learning Natural Language Processing as part of
Fundamentals of Machine Learning in our second year B.Tech."""

stop_words = set(stopwords.words('english'))

word_tokens = word_tokenize(example_sent)

filtered_sentence = [w for w in word_tokens if not w.lower() in stop_words]

filtered_sentence = []
for w in word_tokens:
    if w not in stop_words:
        filtered_sentence.append(w)
print(word_tokens)
print(filtered_sentence)
```

['We', 'are', 'learning', 'Natural', 'Language', 'Processing', 'as', 'part', 'of', 'Fundamentals', 'of', 'Machine', 'Learning', 'in', 'our', 'second', 'year', 'B.Tech', '.']

['We', 'learning', 'Natural', 'Language', 'Processing', 'part', 'Fundamentals', 'Machine', 'Learning', 'second', 'year', 'B.Tech', '.']

Write a simple script to remove
punctuations.

Text Normalization

- Before almost any natural language processing of a text, the text has to be normalized.
- At least three tasks are commonly applied as part of any normalization process:
 1. Tokenizing (segmenting) words
 2. Normalizing word formats
 3. Segmenting sentences

Tokenization

- ▶ Common task in NLP
- ▶ Foremost step
- ▶ It is a way of separating a piece of text into smaller units called Tokens.
- ▶ Tokens are the building blocks of Natural Language
 - ▶ Tokens can be words, characters or subwords
 - ▶ Divided into 3 types
 - ▶ Word
 - ▶ Character
 - ▶ subword
- ▶ Goal – the creation of Vocabulary
 - ▶ Vocabulary – set of unique tokens in the corpus

Example

- ▶ word tokenization for the sentence: "Never give up"
 - ▶ ["Never", "give", "up"]
- ▶ Character tokenization for "smarter" is
 - ▶ ['s','m','a','r','t','e','r']
- ▶ Subword tokenization for "smarter" is
 - ▶ ["smart", "er"]

Word Tokenization

- ▶ Most commonly used tokenization
- ▶ Splits a piece of text into individual words based on a certain delimiter
- ▶ Methods to perform tokenization
 - ▶ Using python's split() function
 - ▶ Using regular expressions
 - ▶ Using NLTK

Using Python's split() function



```
text = '''Once there lived a greedy man in a small town. He was very rich, and he loved gold and all things fancy. But he loved his daughter more than anything. One day, he chanced upon a fairy. The fairy's hair was caught in a few tree branches. He helped her out, but as his greediness took over, he realised that he had an opportunity to become richer by asking for a wish in return (by helping her out). The fairy granted him a wish. He said, "All that I touch should turn to gold." And his wish was granted by the grateful fairy.'''

#word tokenization
tokens = text.split()
print(tokens)
print("No.of tokens : ", len(tokens))

#sentence tokenization
sentences = text.split('.')
print(sentences)
print("No.of sentences : ", len(sentences))
```

Tokenization using Regular Expressions

```
● ● ●

import re

#word tokenization
tokens = re.findall("[\w']+", text)
print(tokens)
print("No.of tokens : ", len(tokens))

#sentence tokenization
sentences = re.compile('[.?!] ').split(text)
print(sentences)
print("No.of sentences : ", len(sentences))
```

Tokenization using NLTK

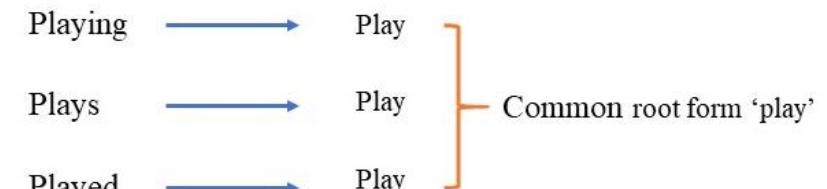


```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
print(tokens)
print("No.of tokens : ", len(tokens))

from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
print(sentences)
print("No.of sentences : ", len(sentences))
```

Word Normalization, Stemming and Lemmatization

- ▶ used to prepare text, words, and documents for further processing
- ▶ Stemming and Lemmatization helps us to achieve the root forms of inflected words



am, are, is → be

Car cars, car's, cars' → car

Using above mapping a sentence could be normalized as follows:

the boy's cars are different colors → the boy car be differ color

Stemming

- helps us to achieve the root forms of inflected words.
- Stem (root) is the part of the word to which you add inflectional (changing/deriving) affixes such as (-ed,-ize, -s,-de,mis).
- stemming a word or sentence may result in words that are not actual words. Stems are created by removing the suffixes or prefixes used with a word.
- A computer program that stems word is called a stemming program, or stemmer
- PorterStemmer is stemming algorithm present in NLTK which uses Suffix Stripping
- It does not follow linguistics rather a set of 5 rules for different cases that are applied in phases to generate stems.

```
from nltk.stem import PorterStemmer
```

```
#create an object of class PorterStemmer
porter = PorterStemmer()
print(porter.stem("cats"))
print(porter.stem("troubles"))
```

```
cat
troubl
```

create a function which takes a sentence and returns the stemmed sentence.



```
from nltk.stem import PorterStemmer
porter = PorterStemmer()

sentence="Pythoners are very intelligent and work very pythonly and now they are pythoning their way to success."

from nltk.tokenize import sent_tokenize, word_tokenize
def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    print(token_words)
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")
    return "".join(stem_sentence)

x=stemSentence(sentence)
print("Sentence after stemming :", x)
```

```
['Pythoners', 'are', 'very', 'intelligent', 'and', 'work', 'very', 'pythonly', 'and', 'now', 'they', 'are', 'pythoning', 'their', 'way', 'to', 'success', '.']
Sentence after stemming : python are veri intellig and work veri pythonli and now they are python their way to success .
```

Lemmatization

- Lemmatization reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called Lemma
- For example, **runs**, **running**, **ran** are all forms of the word **run**, therefore run is the **lemma** of all these words.
- As lemmatization returns an actual word of the language, it is used where it is necessary to get valid words.
- Python NLTK provides **WordNetLemmatizer** that uses the **WordNet** Database to lookup lemmas of words.

```
from nltk.stem import WordNetLemmatizer  
wordnet_lemmatizer = WordNetLemmatizer()  
  
print(wordnet_lemmatizer.lemmatize("cats"))  
print(wordnet_lemmatizer.lemmatize("troubles"))
```

cat
trouble

create a function which takes a sentence and returns the lemmatized sentence.

Sentence Segmentation

- ▶ Sentence segmentation is another important step in text processing.
- ▶ The most useful cues for segmenting a text into sentences are punctuation, like periods, question marks, and exclamation points.

Standardization of Data

The common operations performed to standardize the data are

- ▶ Removal of duplicate whitespaces and punctuation.
- ▶ Accent removal
- ▶ Capital letter removal
- ▶ Removal or substitution of special characters/emojis (e.g.: remove hashtags).
- ▶ Substitution of contractions (very common in English; e.g.: 'I'm'→'I am').
- ▶ Transform word numerals into numbers (eg.: 'twenty three'→'23').
- ▶ Substitution of values for their type (e.g.: '\$50'→'MONEY').
- ▶ Acronym normalization (e.g.: 'US'→'United States'/'U.S.A') and abbreviation normalization (e.g.: 'btw'→'by the way').
- ▶ Normalize date formats, social security numbers
- ▶ Spell correction — this is very important if you're dealing with open user inputs, such as tweets, IMs and emails.
- ▶ Removal of gender/time/grade variation with Stemming or Lemmatization.
- ▶ Substitution of rare words for more common synonyms.
- ▶ Stop word removal (more a dimensionality reduction technique than a normalization technique).

Natural Language Processing / Text mining Introduction. Applications. Chatbots, virtual agents (Alexa, Google Assistant, Siri). Importance, Applications, NLP Subproblems. Components of Natural Language. Steps to get text data into workable format. Terms Frequency, Inverse Document Frequency, Bag of Words, n-gram, One hot encoding. Notion of corpus. Intro to NLTK

Natural Language Processing / Text mining Introduction

Definition : Natural language processing is a sub-field of linguistics, computer science and AI concerned with the interactions between computers and human language.

The goal of NLP : Computers and machines are great at working with tabular, structured data. Much of the information humans generate has complex structure and definitely not tabular, making it very difficult for computers to interpret. NLP makes computers understand complex language structure and retrieve meaningful pieces of information from it.

- Modern challenges in NLP frequently involve
 - speech recognition,
 - natural language understanding and
 - natural language generation.

Refer slides to understand where we are using NLP these days....

Applications of NLP

- **Text Classification:** This is the task of bucketing the text into a known set of categories based on its content. It is the most popular task in NLP and is used in a variety of tools, from email spam identification to sentiment analysis.
- **Language Modelling:** This is the task of predicting what the next word in a sentence will be based on the history of previous words. The goal of this task is to learn the probability of a sequence of words appearing in a given language. Language modelling is useful for building solutions for a wide variety of problems, such as speech recognition, optical character recognition, handwriting recognition, machine translation, and spelling correction.
- **Information Extraction:** As the name indicates, this is the task of extracting relevant information from text, such as calendar events from emails or the names of people mentioned in a social media post. Some of its common sub-tasks are Named Entity Recognition, Co-reference Resolution.
- **Information Retrieval:** This is the task of finding documents relevant to a user query from a large collection. Applications like Google Search are well-known use cases of information retrieval.
- **Conversational Agent:** This is the task of building dialogue systems that can converse in human languages. Alexa, Siri, etc., are some common applications of this task.
- **Text Summarization:** This task aims to create summaries of longer documents while retaining the core content and preserving the overall meaning of the text.
- **Question Answering:** This is the task of building a system that can automatically answer questions posed in natural language.
- **Machine Translation:** This is the task of converting a piece of text from one language to another. Tools like Google Translate are common applications of this task.
- **Topic Modelling:** This is the task of uncovering the topical structure of a large collection of documents. Topic modelling is a common text-mining tool and is used in a wide range of domains.

- **Speech Recognition:** This is the task of building technologies that enables the recognition and translation of spoken language into text. Applications like Voice user interface, search keywords, speech-to-text processing and direct voice input in modern military aircraft are well-known use cases of this task.

Chatbots, virtual agents (Alexa, Google Assistant, Siri). Importance, Applications

Refer slides

NLP Subproblems

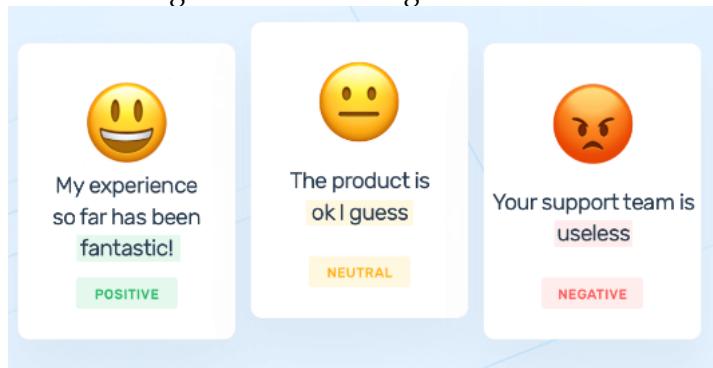
- Text Categorization
- Machine Translation
- Text Summarization
- Entity Recognition
- Temporal even recognition
- Text Generation
- Natural Language Interface
- Speech Recognition
- Text to speech

Text Categorization

1. **Sentiment Analysis** – It is the extraction of the sentiment, the positive or negative orientation that a writer expresses toward some object. A review of a movie, book, or a product on the web expresses the author's sentiment toward the product. Extracting sentiment is relevant for fields from marketing to politics. To perform this task the words in the review provide excellent cues. For example, words like *great, richly, awesome, pathetic, awful and ridiculously* are informative cues

Examples :

- i. “ I really like the new design of your website! ” – Positive
- ii. “ The new design is awful ” – Negative.



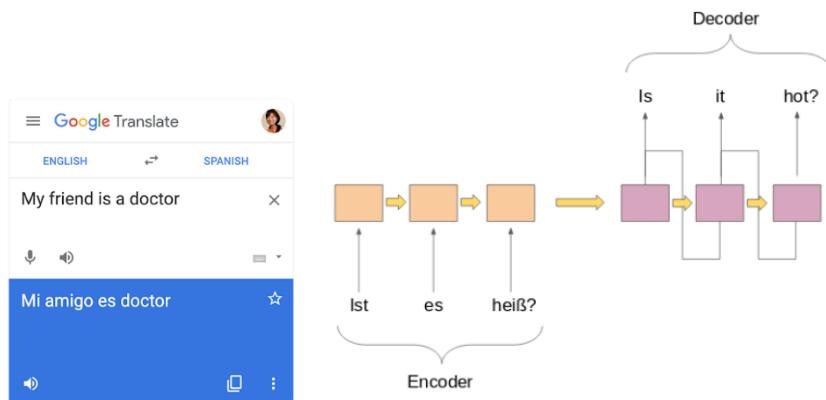
2. **Spam Detection** – It is another important commercial application. It is a binary classification task of assigning an email to one of the two classes *spam* or *not-spam*. Many lexical and other features can be used to perform this classification. For example, you might quite reasonably be suspicious of an email containing phrases like “online pharmaceutical” or “WITHOUT ANY COST” or “Dear Winner”
 3. **Authorship Attribution** - Authorship attribution is the task of identifying the author of a given document.
- The **goal** of classification is
 - to take a single observation,
 - extract some useful features, and

Unit – IV Natural Language Processing

- thereby classify the observation into one of a set of discrete classes.

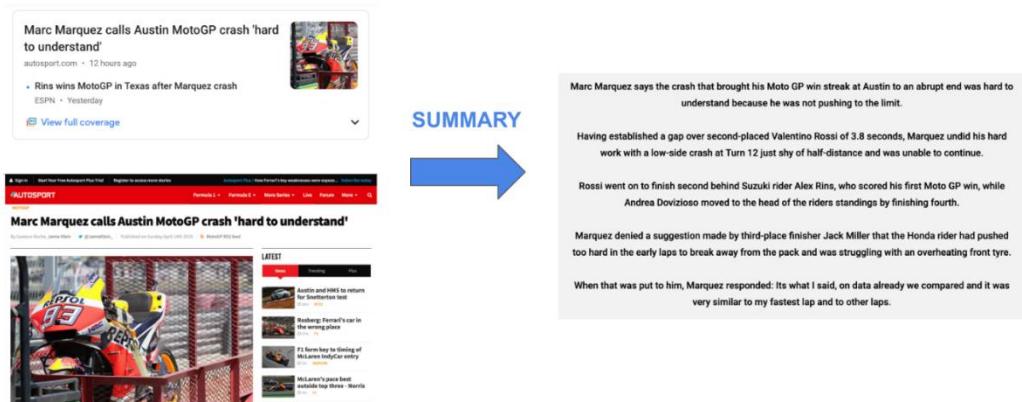
Machine Translation

- Machine Translation(MT) translates one neural language into another language automatically. Encoder-Decoder models are commonly employed to solve Machine Translation problems.
- Machine translation can be done using Statistics or based on rules.
- Neural Machine Translation relies upon neural network models to build statistical models
- MT can translate huge amounts of text rapidly.



Text Summarization

- Automatic text summarization aims to transform lengthy documents into shortened versions, something which could be difficult and costly to undertake if done manually. Machine learning algorithms can be trained to comprehend documents and identify the sections that convey important facts and information before producing the required summarized texts. For example, the image below is of [this news article](#) that has been fed into a machine learning algorithm to generate a summary.



Entity Recognition

- Named Entity Recognition is one of the key entity detection methods in NLP.
- Named entity recognition is a natural language processing technique that can automatically scan entire articles and pull out some fundamental entities in a text and classify them into predefined categories. Entities may be,
 - Organizations,

- Quantities,
 - Monetary values,
 - Percentages, and more.
 - People's names
 - Company names
 - Geographic locations (Both physical and political)
 - Product names
 - Dates and times
 - Amounts of money
 - Names of events
- In simple words, Named Entity Recognition is the process of detecting the named entities such as person names, location names, company names, etc from the text.
 - It is also known as entity identification or entity extraction or entity chunking.

For Example,

Ousted WeWork founder Adam Neumann lists his Manhattan penthouse for \$37.5 million

[organization]

[person]

[location]

[monetary value]

- With the help of named entity recognition, we can extract key information to understand the text, or merely use it to extract important information to store in a database.
- The applicability of entity detection can be seen in many applications such as
 - Automated Chatbots,
 - Content Analyzers,
 - Consumer Insights, etc.

Text Generation

- Text generation is a subfield of **natural language processing (NLP)**. It leverages knowledge in computational linguistics and artificial intelligence to *automatically generate natural language texts*, which can *satisfy certain communicative requirements*.

Natural Language Interface

- A *natural language interface* is a user interface in which the user and the system communicate via a natural (human) language. The user provides input via speech or some other method, and the system generates responses in the form of utterances delivered by speech, text or another suitable modality.

Speech Recognition

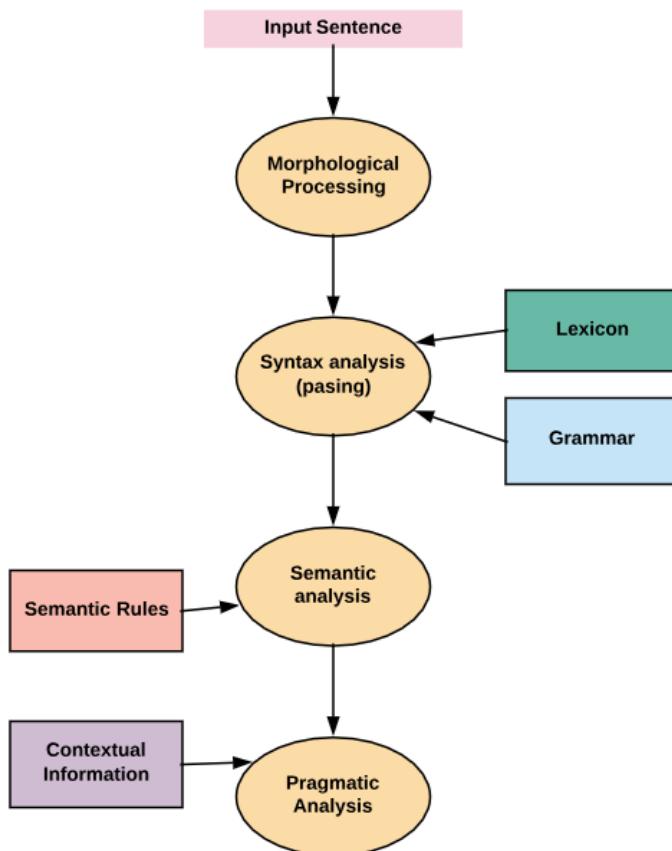
- Speech recognition, also known as automatic speech recognition (ASR), computer speech recognition, or speech-to-text, is a capability which enables a program to process human speech into a written format.
- Many speech recognition applications and devices are available, but the more advanced solutions use AI and machine learning. They integrate grammar, syntax, structure, and composition of audio and voice signals to understand and process human speech. Ideally, they learn as they go – evolving responses with each interaction.



Components of NLP

Five main Component of Natural Language processing in AI are:

- Morphological and Lexical Analysis
- Syntactic Analysis
- Semantic Analysis
- Discourse Integration
- Pragmatic Analysis



Morphological and Lexical Analysis

- Lexical analysis is a vocabulary that includes its words and expressions. It depicts analyzing, identifying and description of the structure of words. It includes dividing a text into paragraphs, words and the sentences
- Individual words are analyzed into their components, and nonword tokens such as punctuations are separated from the words.

Semantic Analysis

- Semantic Analysis is a structure created by the syntactic analyzer which assigns meanings. This component transfers linear sequences of words into structures. It shows how the words are associated with each other.
- Semantics focuses only on the literal meaning of words, phrases, and sentences. This only abstracts the dictionary meaning or the real meaning from the given context. The structures assigned by the syntactic analyzer always have assigned meaning
- E.g., “colorless green idea.” This would be rejected by the Symantec analysis as colorless Here; green doesn’t make any sense.

Pragmatic Analysis

- Pragmatic Analysis deals with the overall communicative and social content and its effect on interpretation. It means abstracting or deriving the meaningful use of language in situations. In this analysis, the main focus always on what was said in reinterpreted on what is meant.
- Pragmatic analysis helps users to discover this intended effect by applying a set of rules that characterize cooperative dialogues.
- E.g., “close the window?” should be interpreted as a request instead of an order.

Syntax analysis

- The words are commonly accepted as being the smallest units of syntax. The syntax refers to the principles and rules that govern the sentence structure of any individual languages.
- Syntax focus about the proper ordering of words which can affect its meaning. This involves analysis of the words in a sentence by following the grammatical structure of the sentence. The words are transformed into the structure to show how the word are related to each other.

Discourse Integration

- It means a sense of the context. The meaning of any single sentence which depends upon that sentences. It also considers the meaning of the following sentence.
- For example, the word “that” in the sentence “He wanted that” depends upon the prior discourse context.

Notion of Corpus

Words

- What is a corpus? - a computer-readable corpora collection of text or speech
 - For example, the Brown corpus is a million-word collection of samples from 500 written English texts from different genres (newspaper, fiction, non-fiction, academic, etc.)

How many words are in the following Brown sentence?

Sentence: *He stepped out into the hall, was delighted to encounter a water brother.*

- This sentence has 13 words if we don't count punctuation marks as words,

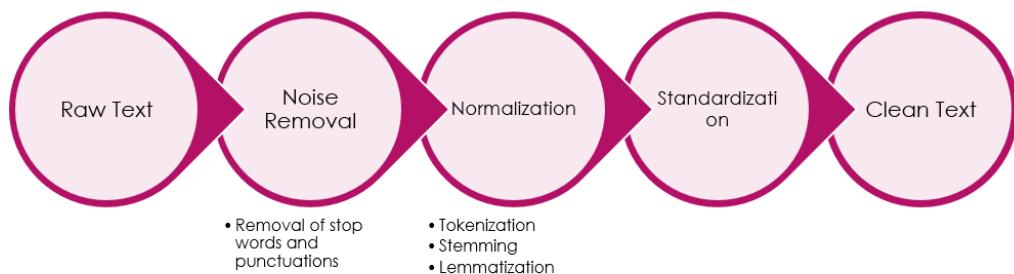
- 15 if we count punctuation.
- Are capitalized tokens like **They** and uncapitalized tokens like **they** the same word?
 - These are lumped together in some tasks (speech recognition)
 - for part-of-speech or named-entity tagging, capitalization is a useful feature and is retained.
- How about inflected forms like cats versus cat?
 - These two words have the same lemma cat but are different wordforms.
- A lemma is a set of lexical forms having the same stem, the same major part-of-speech, and the same word sense.
- The wordform form is the full inflected or derived form of the word
- **Word Types** are the number of distinct words in a corpus; if the set of words in the vocabulary is V , the number of types is the word token vocabulary size $|V|$.
- **Word Tokens** are the total number N of running words.

Corpora

- Words don't appear out of nowhere. Any particular piece of text that we study is produced by one or more specific speakers or writers, in a specific dialect of a specific language, at a specific time, in a specific place, for a specific function.
- The most important dimension of variation is the language. Other variations are genre, demographic characters of writer.
- Language:
- NLP algorithms are most useful when they apply across many languages. The world has 7097 languages.
- It is important to test algorithms on more than one language, and particularly on languages with different properties; by contrast there is an unfortunate current tendency for NLP algorithms to be developed or tested just on English
- Code Switching is a phenomenon which uses multiple languages in a single communicative act.
 - Eg: What is your name andi? (Combining English and Telegu)
- Genre:
 - The text that our algorithms must process might come from newswire, fiction or non-fiction books, scientific articles, Wikipedia, or religious texts.
 - It might come from spoken genres like telephone conversations, business meetings, police body-worn cameras, medical interviews or transcripts of television shows or movies.
 - It might come from work situations like doctors' notes, legal text, or parliamentary proceedings.
- demographic characteristics of the writer (or speaker):
 - their age, gender, race, socioeconomic class can all influence the linguistic properties of the text we are processing. And finally, time matters too.
- Time:
 - Language changes over time, and for some languages we have good corpora of texts from different historical periods.

Steps to get text into workable format

Approximate order for pre-processing text data is shown below.



Noisy Entities removal:

- Any piece of text which is not relevant to the context of the data can be specified as the noise.
- Generally, the noisy entities are
 - Stop words,
 - punctuation marks.

Stop Words removal -

- It is a process of removing common language articles, Pronouns and propositions such as "and", "the" or "to" in English.
- These words provide little or no value to NLP.
- Stop words are removed /filtered from text for better efficiency

Some common stop words in English are:

```

import nltk
from nltk.corpus import stopwords
print(stopwords.words('english'))

```

```

['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
 "you'd", "your", 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she',
 'she's', 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these',
 'those', 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because',
 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into',
 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in',
 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there',
 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other',
 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
 's', 't',
 'can', 'will', 'just', 'don', 'don't', 'should', 'should've', 'now', 'd', 'll', 'm', 'o', 'r',
 'e', 've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn',
 "doe sn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma',
 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

```

Text Normalization

- Before almost any natural language processing of a text, the text has to be normalized.
- At least three tasks are commonly applied as part of any normalization process:
 1. Tokenizing (segmenting) words
 2. Normalizing word formats
 3. Segmenting sentences

1. Tokenization :

- Tokenization is a common task in Natural Language Processing (NLP). It is the foremost step while modelling the text data.
- It is a way of separating a piece of text into smaller units called tokens.
- **Tokens** are the building blocks of Natural Language. The most common way of processing the raw text happens at the token level.
- The ultimate goal of Tokenization is the creation of vocabulary - Tokenization is performed on the corpus to obtain tokens. The following tokens are then used to prepare a vocabulary. **Vocabulary** refers to the set of unique tokens in the corpus.
- The tokens can be words, characters, or subwords.
- Hence, tokenization can be broadly classified into 3 types.
 - word, character and subword tokenization.
- Example:
 - word tokenization for the sentence: "Never give up" - ["Never", "give", "up"]
 - Character tokenization for "smarter" is ['s', 'm', 'a', 'r', 't', 'e', 'r']
 - Subword tokenization for "smarter" is ["smart", "er"]

Word Tokenization : Word Tokenization is the most commonly used tokenization algorithm. It splits a piece of text into individual words based on a certain delimiter. Depending upon delimiters, different word-level tokens are formed.

Methods to perform tokenization:

1. Tokenization using Python's split() function :

- Tokenization using Python **split()** method is the most basic one. It returns a list of strings after breaking the given string by the specified separator. By default, **split()** breaks a string at each space. We can change the separator to anything.
- The drawbacks of using Python's **split()** method are
 - we can use only one separator at a time.
 - **split()** did not consider punctuation as a separate token.

```
● ● ●

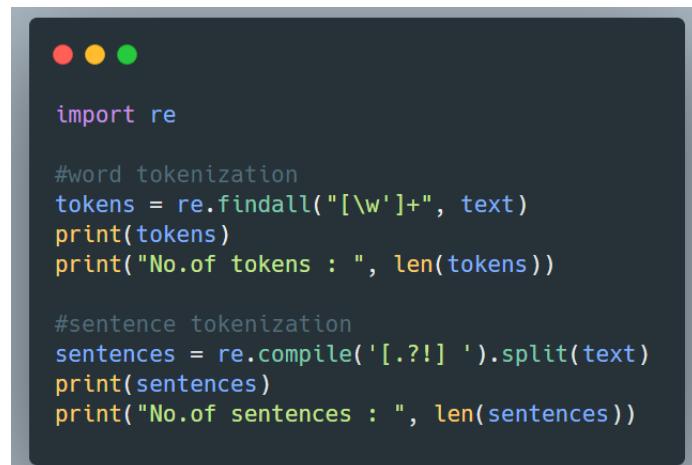
text = '''Once there lived a greedy man in a small town. He was very rich, and he loved gold and all
things fancy. But he loved his daughter more than anything. One day, he chanced upon a fairy. The
fairy's hair was caught in a few tree branches. He helped her out, but as his greediness took over, he
realised that he had an opportunity to become richer by asking for a wish in return (by helping her
out). The fairy granted him a wish. He said, "All that I touch should turn to gold." And his wish was
granted by the grateful fairy.'''
print(text)

#word tokenization
tokens = text.split()
print(tokens)
print("No.of tokens : ", len(tokens))

#sentence tokenization
sentences = text.split('.')
print(sentences)
print("No.of sentences : ", len(sentences))
```

2. Tokenization using Regular Expressions (RegEx)

- The `re.findall()` function finds all the words that match the pattern passed on it and stores it in the list.
- `[\w']+` signals that the code should find all the alphanumeric characters until any other character is encountered.
 - The “\w” represents “any word character” which usually means alphanumeric (letters, numbers) and underscore (_).
 - ‘+’ means any number of times.



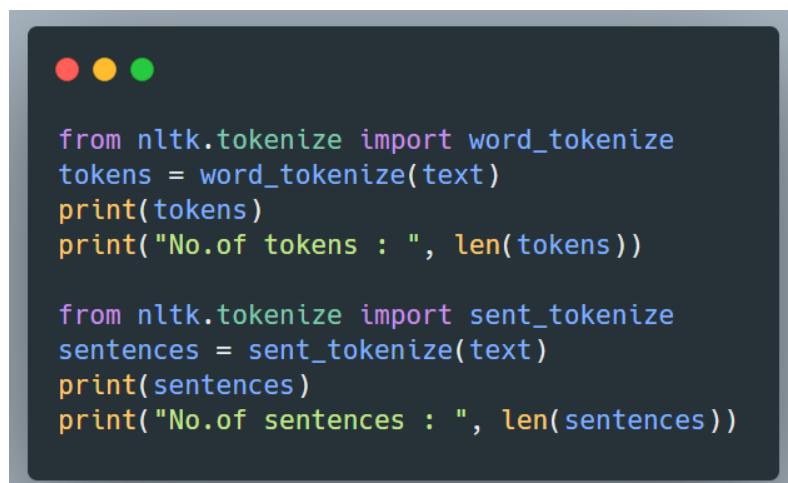
```
import re

#word tokenization
tokens = re.findall("[\w']+", text)
print(tokens)
print("No.of tokens : ", len(tokens))

#sentence tokenization
sentences = re.compile('[.?!] ').split(text)
print(sentences)
print("No.of sentences : ", len(sentences))
```

3. Tokenization using NLTK

- NLTK, short for Natural Language Tool Kit, is a library written in Python for NLP.
- NLTK contains a module called `tokenize()` which further classifies into two sub-categories:
- Word tokenize:
 - We use the `word_tokenize()` method to split a sentence into tokens or words
- Sentence tokenize:
 - We use the `sent_tokenize()` method to split a document or paragraph into sentences



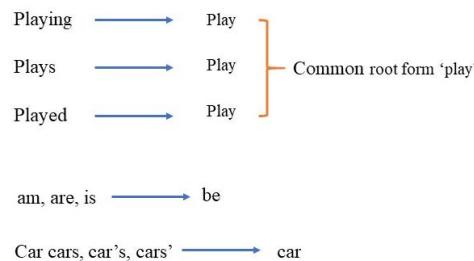
```
from nltk.tokenize import word_tokenize
tokens = word_tokenize(text)
print(tokens)
print("No.of tokens : ", len(tokens))

from nltk.tokenize import sent_tokenize
sentences = sent_tokenize(text)
print(sentences)
print("No.of sentences : ", len(sentences))
```

2. Word Normalization, Stemming and Lemmatization

- Stemming and Lemmatization are Text Normalization techniques in the field of Natural Language Processing that are used to prepare text, words, and documents for further processing.

- Languages we speak and write are made up of several words often derived from one another. When a language contains words that are derived from another word as their use in the speech changes is called Inflected Language. Inflected words will have common root form. Examples are shown below.



Using above mapping a sentence could be normalized as follows:

the boy's cars are different colors —————→ the boy car be differ color

- Stemming and Lemmatization helps us to achieve the root forms of inflected words. Stemming is different to Lemmatization in the approach it uses to produce root forms of words and the word produced.
- **PorterStemmer** is stemming algorithm present in NLTK which uses Suffix Stripping to produces stems.
- It does not follow linguistics rather a set of 5 rules for different cases that are applied in phases (step by step) to generate stems.

Stemming with Python NLTK package

- Stem (root) is the part of the word to which you add inflectional (changing/ deriving) affixes such as (-ed,-ize, -s,-de,mis). So, stemming a word or sentence may result in words that are not actual words. Stems are created by removing the suffixes or prefixes used with a word.
- A computer program that stems word is called a stemming program, or **stemmer**.
- PorterStemmer is available in Python NLTK to stem words. It uses **SuffixStripping** to produce stems.
- The following code snippet takes a sentence and produces stems for each token in the sentence:

```

from nltk.stem import PorterStemmer
porter = PorterStemmer()

sentence="Pythoners are very intelligent and work very pythonly and now they are pythoning their way to success."

from nltk.tokenize import sent_tokenize, word_tokenize
def stemSentence(sentence):
    token_words=word_tokenize(sentence)
    print(token_words)
    stem_sentence=[]
    for word in token_words:
        stem_sentence.append(porter.stem(word))
        stem_sentence.append(" ")
    return "".join(stem_sentence)

x=stemSentence(sentence)
print("Sentence after stemming :", x)
  
```

Output :

```
['Pythoners', 'are', 'very', 'intelligent', 'and', 'work', 'very', 'pythononly', 'and', 'now', 'they', 'are', 'pythoning', 'their', 'way', 'to', 'success', '.']
```

Sentence after stemming : python are veri intellig and work veri pythonli and now they are python their way to success .

Lemmatization with Python nltk package

- Lemmatization, unlike Stemming, reduces the inflected words properly ensuring that the root word belongs to the language. In Lemmatization root word is called **Lemma**
- For example, **runs**, **running**, **ran** are all forms of the word **run**, therefore run is the lemma of all these words.
- Because lemmatization returns an actual word of the language, it is used where it is necessary to get valid words.
- Python NLTK provides WordNet Lemmatizer that uses the WordNet Database to lookup lemmas of words.
- The following code snippet takes a sentence and produces lemmas for each token in the sentence:

```
● ● ●

import nltk
from nltk.stem import WordNetLemmatizer
wordnet_lemmatizer = WordNetLemmatizer()

sentence = "He was running and eating at same time. He has bad habit of swimming after playing long
hours in the Sun."
punctuations="?:!.,;"
token_words = nltk.word_tokenize(sentence)
print(token_words)

lemma_sentence=[]
for word in token_words:
    lemma_sentence.append(wordnet_lemmatizer.lemmatize(word))
    lemma_sentence.append(" ")

print("lemmas of tokens: ", ''.join(lemma_sentence))
```

Output :

```
['He', 'was', 'running', 'and', 'eating', 'at', 'same', 'time', '.', 'He', 'has', 'bad', 'habit', 'of', 'swimming', 'after',
'playing', 'long', 'hours', 'in', 'the', 'Sun', '.']
```

lemmas of tokens: He wa running and eating at same time . He ha bad habit of swimming after playing long hour in the Sun .

To summarize,

- Stemming and Lemmatization both generate the root form of the inflected words. The difference is that stem might not be an actual word whereas, lemma is an actual language word.
- Stemming follows an algorithm with steps to perform on the words which makes it faster. Whereas, in lemmatization, you used WordNet corpus and a corpus for stop words as well to produce lemma which makes it slower than stemming. You also had to define a parts-of-speech to obtain the correct lemma.

3. Sentence Segmentation

- Sentence segmentation is another important step in text processing.
- The most useful cues for segmenting a text into sentences are punctuation, like periods, question marks, and exclamation points.
- Question marks and exclamation points are relatively unambiguous markers of sentence boundaries. Periods, on the other hand, are more ambiguous.
- The period character “.” is ambiguous between a sentence boundary marker and a marker of abbreviations like Mr. or Inc.

3. Standardization of Data

The common operations performed to standardize the data are

- Removal of duplicate whitespaces and punctuation.
- Accent removal (if your data includes diacritical marks from ‘foreign’ languages – this helps to reduce errors related to encoding type).
- Capital letter removal (often, working with lowercase words deliver better results. In some cases, however, capital letters are very important to extract information, like names and locations).
- Removal or substitution of special characters/emojis (e.g.: remove hashtags).
- Substitution of contractions (very common in English; e.g.: ‘I’m’→‘I am’).
- Transform word numerals into numbers (eg.: ‘twenty three’→‘23’).
- Substitution of values for their type (e.g.: ‘\$50’→‘MONEY’).
- Acronym normalization (e.g.: ‘US’→‘United States’/‘U.S.A’) and abbreviation normalization (e.g.: ‘btw’→‘by the way’).
- Normalize date formats, social security numbers or other data that have a standard format.
- Spell correction (one could say that a word can be misspelled infinite ways, so spell corrections reduce the vocabulary variation by “correcting”) – this is very important if you’re dealing with open user inputs, such as tweets, IMs and emails.
- Removal of gender/time/grade variation with Stemming or Lemmatization.
- Substitution of rare words for more common synonyms.
- Stop word removal (more a dimensionality reduction technique than a normalization technique, but let us leave it here for the sake of mentioning it).

Feature Extraction

Pre-processing is the first phase before applying any classification model on text data. But the cleaned text isn’t enough to be passed directly to the classification model. The features need to be numeric, not strings. There are many state-of-art approaches to extract features from the text data.

One-hot vectors

- In one hot encoding, every word (even symbols) which are part of the given text data are written in the form of vectors, constituting only of 1 and 0 .
- So one hot vector is a vector whose elements are only 1 and 0. Each word is written or encoded as one hot vector, with each one hot vector being unique.

- This allows the word to be identified uniquely by its one hot vector and vice versa, that is no two words will have same one hot vector representation.

For example see the below image which shows one hot encoding of words for the given sentence.

The cat sat on the mat

```
The: [0 1 0 0 0 0]
cat: [0 0 1 0 0 0]
sat: [0 0 0 1 0 0]
on: [0 0 0 0 1 0]
the: [0 0 0 0 0 1]
mat: [0 0 0 0 0 1]
```

- Notice that in the image to the left the words ‘The’ and ‘the’ have different encoding implying they are different. Thus we are representing every word and symbols in the text data as a unique one hot vector which contains numerical data (1 and 0) as its constituent elements.
- One word is represented as a vector therefore the list of words in the sentence can be represented as an array of vectors or a matrix.

Implementation of one-hot encoding from scratch

```
import numpy as np
# Sample set for our example
samples = {'Jupiter has 79 known moons .', 'Neptune has 14 confirmed moons !'}
# Create an empty dictionary
token_index = {}
#Create a counter for counting the number of key-value pairs in the token_length
counter = 0

# Select the elements of the samples which are the two sentences
for sample in samples:
    for considered_word in sample.split():
        if considered_word not in token_index:
            token_index.update({considered_word : counter + 1})
            counter = counter + 1

print(token_index)

# Set max_length to 6
max_length = 6
# Create a tensor of dimension 3 named results whose every elements are initialized to 0
results = np.zeros(shape = (len(samples),
                            max_length,
                            max(token_index.values()) + 1))

# Now create a one-hot vector corresponding to the word
```

```
# iterate over enumerate(samples) enumerate object
for i, sample in enumerate(samples):

# Convert enumerate object to list and iterate over resultant list
for j, considered_word in list(enumerate(sample.split())):

    # set the value of index variable equal to the value of considered_word in token_index
    index = token_index.get(considered_word)

    # In the previous zero tensor: results, set the value of elements with their positional index as [i, j, index] = 1.
    results[i, j, index] = 1.

print(results)
```

Output:

```
{'Neptune': 1, 'has': 2, '14': 3, 'confirmed': 4, 'moons': 5, '!': 6, 'Jupiter': 7, '79': 8, 'known': 9, '!': 10}
```

```
[[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]]
```

```
[[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]]]
```

drawbacks of one-hot encoding:

- The result of a one-hot encoding process on a corpus is a sparse matrix. Consider if you had a corpus with 20,000 unique words: a single short document in that corpus of, perhaps, 40 words would be represented by a matrix with 20,000 rows (one for each unique word) with a maximum of 40 non-zero matrix elements. This leaves a lot of zeroes, and can end up taking a large amount of memory to house these sparse representations.
- Beyond potential memory capacity issues, a major drawback of one-hot encoding is the lack of meaning representation. While we capture the presence and absence of words in a particular text well with this approach, we can't easily determine any meaning from simple presence/absence of these words.

- The concept of word similarity is difficult to extract as well, since word vectors are statistically orthogonal. Take, for example, the word pairs "dog" and "dogs," or "car" and "auto." Clearly these word pair are similar in different ways, respectively.

Bag-of-Words (BoW):

- The most simple and known method is the Bag-Of-Words representation. It's an algorithm that transforms the text into fixed-length vectors. This is possible by counting the number of times the word is present in a document. The word occurrences allow to compare different documents and evaluate their similarities for applications, such as search, document classification, and topic modelling.
- The reason for its name, "Bag-Of-Words", is due to the fact that it represents the sentence as a bag of terms. It doesn't take into account the order and the structure of the words, but it only checks if the words appear in the document.

Example :

Review 1: Game of Thrones is an amazing tv series!
 Review 2: Game of Thrones is the best tv series!
 Review 3: Game of Thrones is so great

In the table, I show all the calculations to obtain the Bag-Of-Words approach:

	amazing	an	best	game	great	is	of	series	so	the	thrones	tv
0	1	1	0	1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	1	1	0	1	1	1
2	0	0	0	1	1	1	1	1	0	1	0	1

Each row corresponds to a different review, while the rows are the unique words, contained in the three documents.

Obtaining Bag-of-words vectors using scikit-learn

```

import pandas as pd
import numpy as np

doc1 = 'Game of Thrones is an amazing tv series!'
doc2 = 'Game of Thrones is the best tv series!'
doc3 = 'Game of Thrones is so great'

from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
X = vectorizer.fit_transform([doc1, doc2, doc3])
print(vectorizer.get_feature_names())

['amazing', 'an', 'best', 'game', 'great', 'is', 'of', 'series', 'so', 'the', 'thrones', 'tv']

```

```

df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
print(df_bow_sklearn.head())

   amazing  an  best  game  great  is  of  series  so  the  thrones  tv
0        1    1     0     1      0    1    1       1    0     0       1    1
1        0    0     1     1      0    1    1       1    0     1       1    1
2        0    0     0     1      1    1    1       0    1     0       1    0

vectorizer = CountVectorizer(stop_words='english',ngram_range=(2,2))
X = vectorizer.fit_transform([doc1,doc2,doc3])
df_bow_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
df_bow_sklearn.head()

   amazing tv  best tv  game thrones  thrones amazing  thrones best  thrones great  tv series
0        1    0        1       1       1        0        0    1        0        1
1        0    1        1       1       0        1        1    0        1        0    1
2        0    0        0       1       0        0        0    0        0        1    0

```

Drawbacks of BoW

- The length of the vector can be high, and most of the values are zero.
- Computationally, it is not efficient if the vocabulary is very high.
- Uni-gram based BoW is unable to capture the context of the text. Bi-grams and tri-grams can do the job, but they are computationally expensive.

TF-IDF

- tf-idf weighting is usually used when the dimensions are documents.
- It is the product of two terms. first term is term frequency and other is used to give a higher weight to words that occur only in few documents.
- first term - **term frequency** : the frequency of the word t in the document d . we can just use the raw count as the term frequency

$$\text{tf}_{t,d} = \text{count}(t,d)$$

- commonly the raw frequencies are squashed a bit by using \log_{10} of the frequency instead.
- As we can't take the log of 0, we normally add 1 to count.

$$\text{tf}_{t,d} = \log_{10}(\text{count}(t,d) + 1)$$

- terms which occur 0 times in a document would have $tf = \log_{10}(1) = 0$,

10 times in a document $tf = \log_{10}(11) = 1.04$,

100 times $tf = \log_{10}(101) = 2.004$,

1000 times $tf = 3.00044$, and so on.

- second term - **inverse document frequency**: idf gives a higher weight to words that occur only in a few documents.
- Terms that are limited to a few documents are useful for discriminating those documents from the rest of the collection; terms that occur frequently across the entire collection are not as helpful.
- The document frequency df_t of a term t is the number of documents it occurs in. (note this is not collection frequency: total count across all documents)
- From the following table, we can observe that "Romeo" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

- discriminative words like Romeo are emphasized using the *inverse document frequency* or idf term weight. The idf is defined using the fraction $\frac{N}{df_t}$, where N is the total number of documents in the collection, and df_t is the number of documents in which term t occurs. The fewer documents in which a term occurs, the higher this weight. The lowest weight is assigned to terms that occur in all the documents.
- Because of the large number of documents in many collections, this measure too is usually squashed with a log function. The resulting definition for inverse document frequency (idf) is thus

$$idf_t = \log_{10} \left(\frac{N}{df_t} \right)$$

- Following table shows some idf values for some words in the Shakespeare corpus, ranging from extremely informative words which occur in only one play like *Romeo*, to those that occur in few like *salad*, to those which are very common like *fool*, or so common like *good* or *sweet*.

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

- the $tf - idf$ weighted value $w_{t,d}$ for word t in document d thus combines term frequency $tf_{t,d}$.

$$w_{t,d} = tf_{t,d} \times idf_t$$

- $tf - idf$ weighting applied to Shakespeare term-document matrix is shown below. The $tf - idf$ values corresponding to the word *good* have now all become 0; since this word appears in every document.

Raw counts:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

tf-idf:

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.074	0	0.22	0.28
good	0	0	0	0
fool	0.019	0.021	0.0036	0.0083
wit	0.049	0.044	0.018	0.022

- To conclude, the *tf - idf* weighting is the way for weighting co-occurrence matrices in information retrieval, but also plays a role in many other aspects of natural language processing. It's also a great baseline, and the simple thing to try first.

```
import pandas as pd
import numpy as np

doc1 = 'Game of Thrones is an amazing tv series!'
doc2 = 'Game of Thrones is the best tv series!'
doc3 = 'Game of Thrones is so great'

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform([doc1,doc2,doc3])
print(vectorizer.get_feature_names())

df_tfidf_sklearn = pd.DataFrame(X.toarray(),columns=vectorizer.get_feature_names())
print(df_tfidf_sklearn.head())
```

Output:

```
['amazing', 'an', 'best', 'game', 'great', 'is', 'of', 'series', 'so', 'the', 'thrones', 'tv']
      amazing      an      best      game      great      is      of  \
0  0.468699  0.468699  0.000000  0.276821  0.000000  0.276821  0.276821
1  0.000000  0.000000  0.468699  0.276821  0.000000  0.276821  0.276821
2  0.000000  0.000000  0.000000  0.320528  0.542701  0.320528  0.320528

      series      so      the     thrones      tv
0  0.356457  0.000000  0.000000  0.276821  0.356457
1  0.356457  0.000000  0.468699  0.276821  0.356457
2  0.000000  0.542701  0.000000  0.320528  0.000000
```

Pros of using TF-IDF

- The biggest advantages of TF-IDF come from how simple and easy to use it is. It is simple to calculate, it is computationally cheap, and it is a simple starting point for similarity calculations (via TF-IDF vectorization + cosine similarity).

Cons of using TF-IDF

- Something to be aware of is that TF-IDF cannot help carry semantic meaning. It considers the importance of the words due to how it weighs them, but it cannot necessarily derive the contexts of the words and understand importance that way.

N-Grams

- N-grams are continuous sequences of words or symbols or tokens in a document. In technical terms, they can be defined as the neighbouring sequences of items in a document. They come into play when we deal with text data in NLP tasks.
- n-grams are classified into the following types, depending on the value that 'n' takes. The following table also shows the generated N-grams for a sentence.

N	Term	Example: "I reside in Bengaluru"
1	Unigram	["I", "reside", "in", "Bengaluru"]
2	Bigram	["I reside", "reside in", "in Bengaluru"]
3	Trigram	["I reside in", "reside in Bengaluru"]
N	N-gram	

- From the table above, it's clear that unigram means taking only one word at a time, bigram means taking two words at a time and trigram means taking three words at a time.
- Different types of n-grams are suitable for different types of applications. You should try different n-grams on your data in order to confidently conclude which one works the best among all for your text analysis. For instance, research has substantiated that trigrams and 4-grams work the best in the case of spam filtering.

N-gram Language Model:

- An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. A good N-gram model can predict the next word in the sentence i.e the value of $p(w | h)$



- An N-gram language model predicts the probability of a given N-gram within any sequence of words in the language. If we have a good N-gram model, we can predict $p(w | h)$ - what is the probability of seeing the word w given a history of previous words h - where the history contains $n-1$ words.

We must estimate this probability to construct an N-gram model.

We compute this probability in two steps:

1. Apply the chain rule of probability
2. We then apply a very strong simplification assumption to allow us to compute $p(w_1 \dots w_n)$ in an easy manner

The chain rule of probability is:

$$p(w_1 \dots w_n) = p(w_1) \cdot p(w_2 | w_1) \cdot p(w_3 | w_1 w_2) \cdot p(w_4 | w_1 w_2 w_3) \dots p(w_n | w_1 \dots w_{n-1})$$

Chain rule tells the way to compute the joint probability of a sequence by using the conditional probability of a word given previous words.

But we do not have access to these conditional probabilities with complex conditions of up to $n-1$ words. So how do we proceed?

This is where we introduce a simplification assumption. We can assume for all conditions, that:

$$p(w_k \mid w_1 \dots w_{k-1}) = p(w_k \mid w_{k-1})$$

Here, we **approximate** the history (the context) of the word w_k by looking only at the last word of the context. This assumption is called the **Markov assumption**.

Building a basic language model:

Now that we understand what an N-gram is, let's build a basic language model using trigrams of the Reuters corpus. Reuters corpus is a collection of 10,788 news documents totalling 1.3 million words. We can build a language model in a few lines of code using the NLTK package:

```
from nltk.corpus import reuters
from nltk import bigrams, trigrams
from collections import Counter, defaultdict

# Create a placeholder for model
model = defaultdict(lambda: defaultdict(lambda: 0))

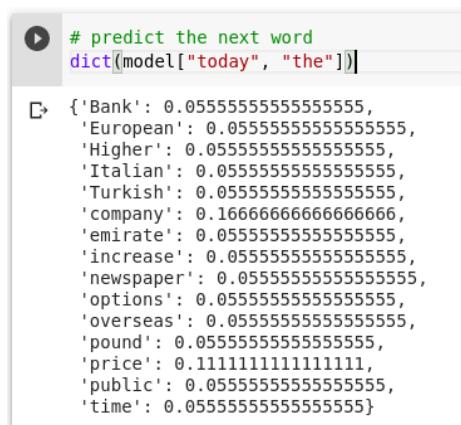
# Count frequency of co-occurrence
for sentence in reuters.sents():
    for w1, w2, w3 in trigrams(sentence, pad_right=True, pad_left=True):
        model[(w1, w2)][w3] += 1

# Let's transform the counts to probabilities
for w1_w2 in model:
    total_count = float(sum(model[w1_w2].values()))
    for w3 in model[w1_w2]:
        model[w1_w2][w3] /= total_count
```

We first split our text into trigrams with the help of NLTK and then calculate the frequency in which each combination of the trigrams occurs in the dataset.

We then use it to calculate probabilities of a word, given the previous two words. That's essentially what gives us our Language Model!

Let's make simple predictions with this language model. We will start with two simple words – "today the". We want our model to tell us what will be the next word:



```
# predict the next word
dict(model["today", "the"])

{'Bank': 0.0555555555555555,
'European': 0.0555555555555555,
'Higher': 0.0555555555555555,
'Italian': 0.0555555555555555,
'Turkish': 0.0555555555555555,
'company': 0.1666666666666666,
'emirate': 0.0555555555555555,
'increase': 0.0555555555555555,
'newspaper': 0.0555555555555555,
'options': 0.0555555555555555,
'overseas': 0.0555555555555555,
'pound': 0.0555555555555555,
'price': 0.1111111111111111,
'public': 0.0555555555555555,
'time': 0.0555555555555555}
```

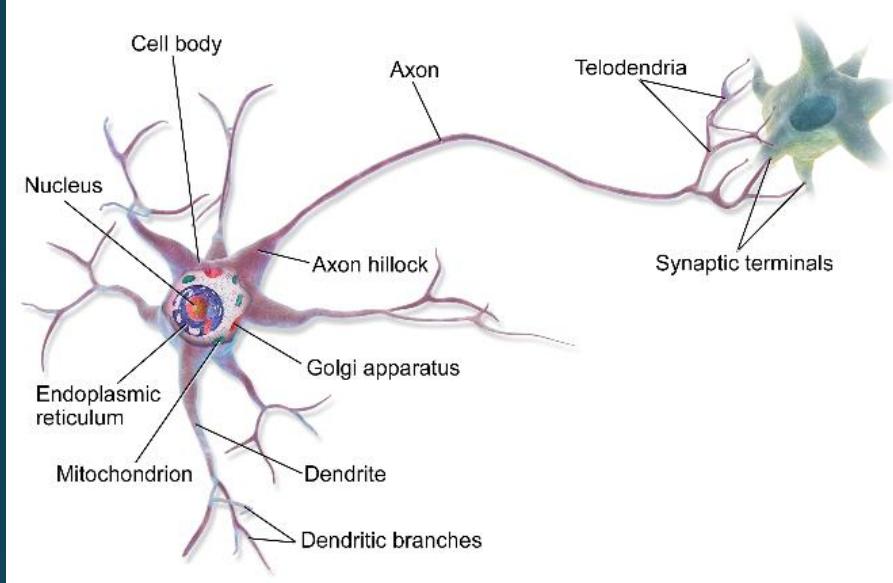
Unit - V

Neural Networks

contents

- Neural Networks
 - Biological Neurons and Biological Neural Networks
 - Perceptron Learning
 - Activation Functions
 - Multilayer Perceptron
 - Back-propagation Neural Networks
 - Convolution Neural Network

Biological Neuron



- Human Brain
 - Motivation behind neural network
 - The best processor even though it works slower than other computers.
 - contains billion of neurons which are connected to many other neurons to form a network so that if it sees any image, it recognizes the image and processes the output.
- Many researchers thought to make a machine that would work in the prospective of the human brain.

Dendrite receives signals from other neurons.

Cell body sums the incoming signals to generate input.

When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons.

The amount of signal transmitted depend upon the strength of the connections.

Connections can be inhibitory, i.e. decreasing strength or excitatory, i.e. increasing strength in nature.

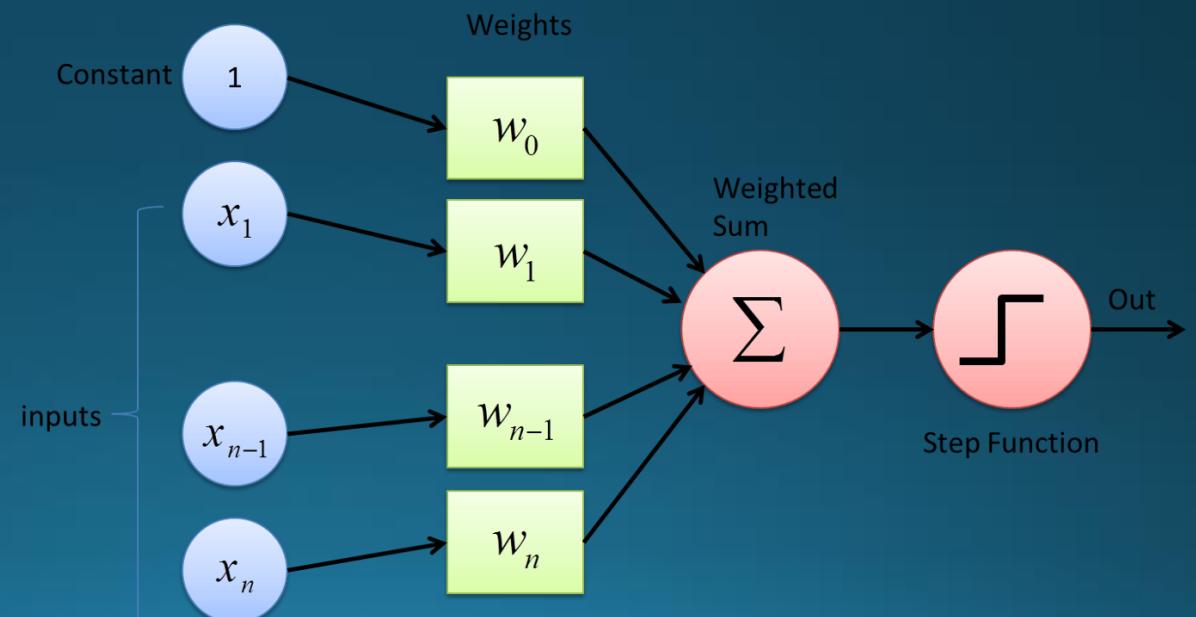
How a Biological Neuron is motivation for Neural Network?

Biological Neuron... (motivation)

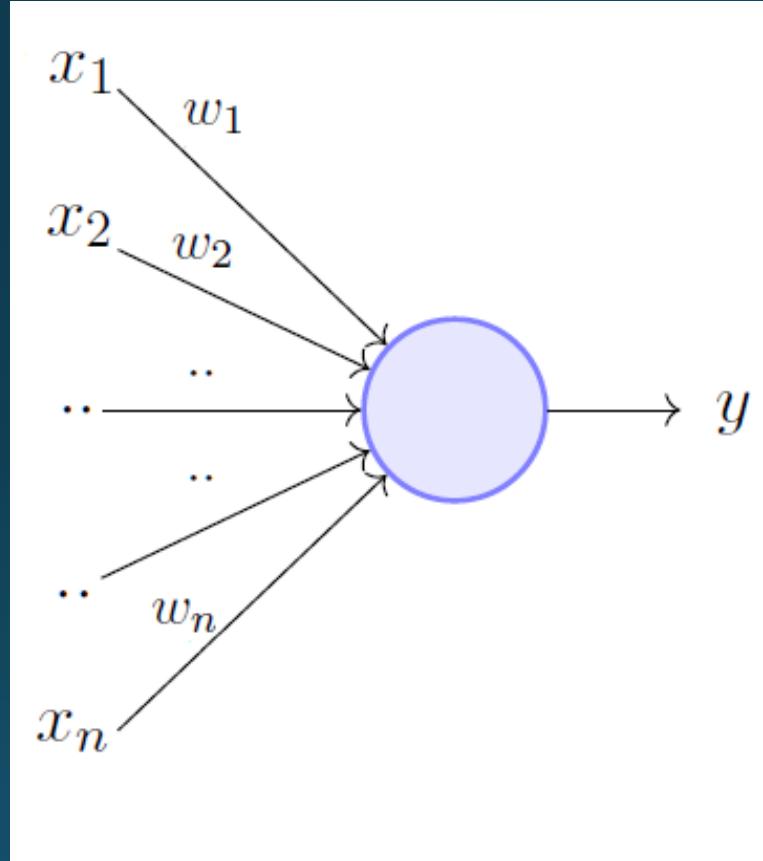
- a modern neural network is a network of small computing units, each of which takes a vector of input values and produces a single output value.
- Neural networks share much of the same mathematics as logistic regression. But neural networks are a more powerful classifier than logistic regression

Perceptron

- The perceptron model is a more general computational model.
- It takes
 - an input, → aggregates it (weighted sum) → and returns 1 only if the aggregated sum is more than some threshold else returns 0.
- How a step function works?



Perceptron Learning



$$y = 1 \quad if \sum_{i=1}^n w_i * x_i \geq \theta$$

$$= 0 \quad if \sum_{i=1}^n w_i * x_i < \theta$$

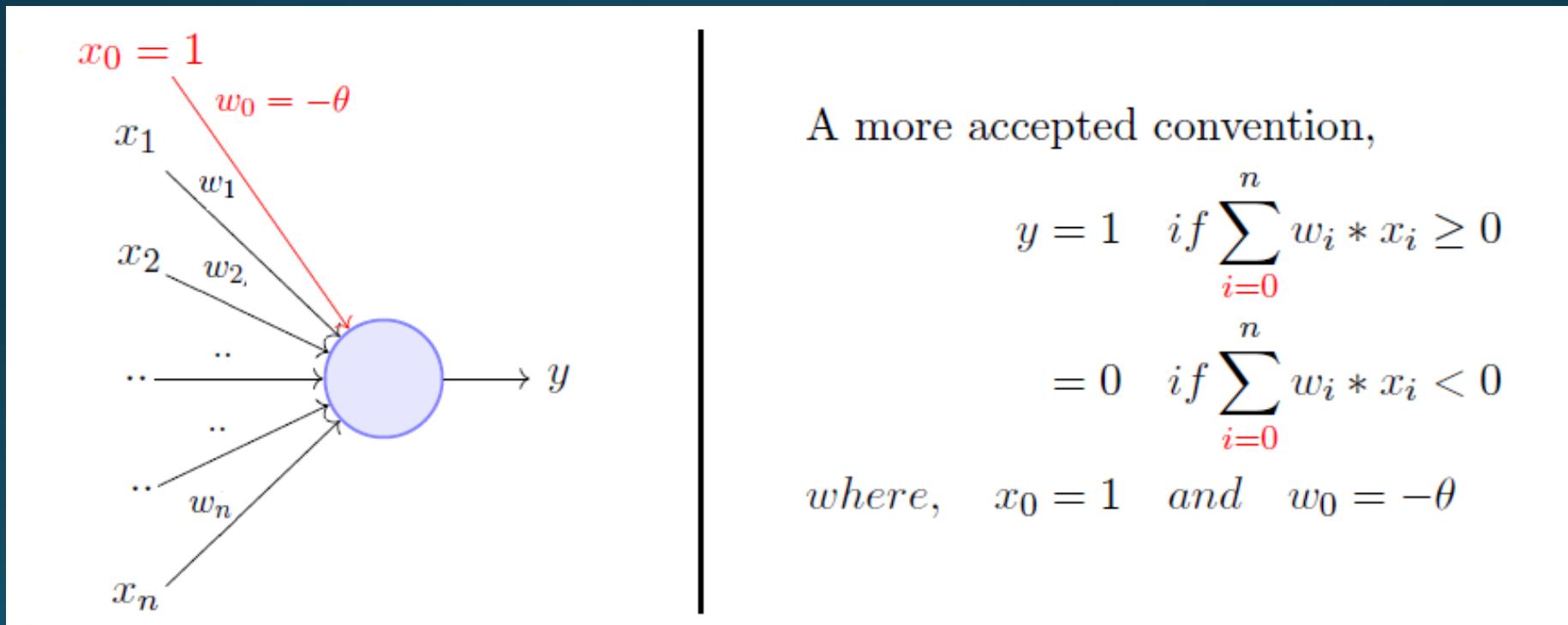
Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^n w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^n w_i * x_i - \theta < 0$$

Perceptron Learning

- Making the threshold a constant input with variable weight



OR Boolean function using perceptron

- A single perceptron can only be used to implement linearly separable functions.
- It takes both real and Boolean inputs and associates a set of weights to them, along with a bias
- We learn the weights, we get the function.
- Let's use a perceptron to learn an OR function.

x_1	x_2	OR	
0	0	0	$w_0 + \sum_{i=1}^2 w_i x_i < 0$
1	0	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
0	1	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
1	1	1	$w_0 + \sum_{i=1}^2 w_i x_i \geq 0$

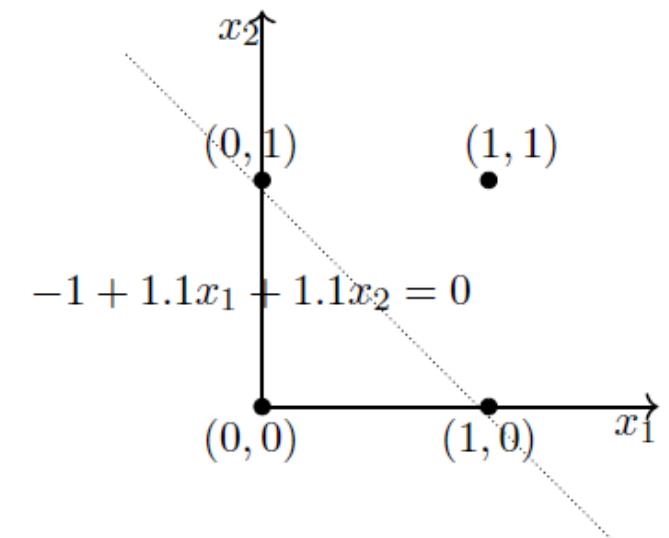
$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0$$

One possible solution is
 $w_0 = -1, w_1 = 1.1, w_2 = 1.1$



Neuron - unit

- The building block of a neural network is a single computational unit.
- A unit takes a set of real valued numbers as input, performs some computation on them, and produces an output.
- A neural unit takes a weighted sum of its inputs, with one additional term in the sum called a bias term.
- Given a set of inputs $x_1 \dots x_n$, a unit has a set of corresponding weights $w_1 \dots w_n$ and a bias b , so the weighted sum z can be represented as

$$z = b + \sum_i w_i x_i$$

- Representation of the above in vector notation..

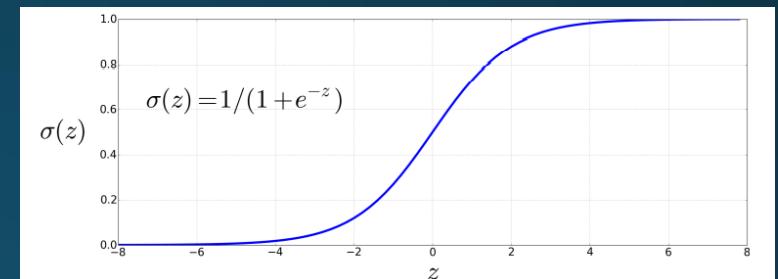
$$z = w \cdot x + b$$

Neuron - unit

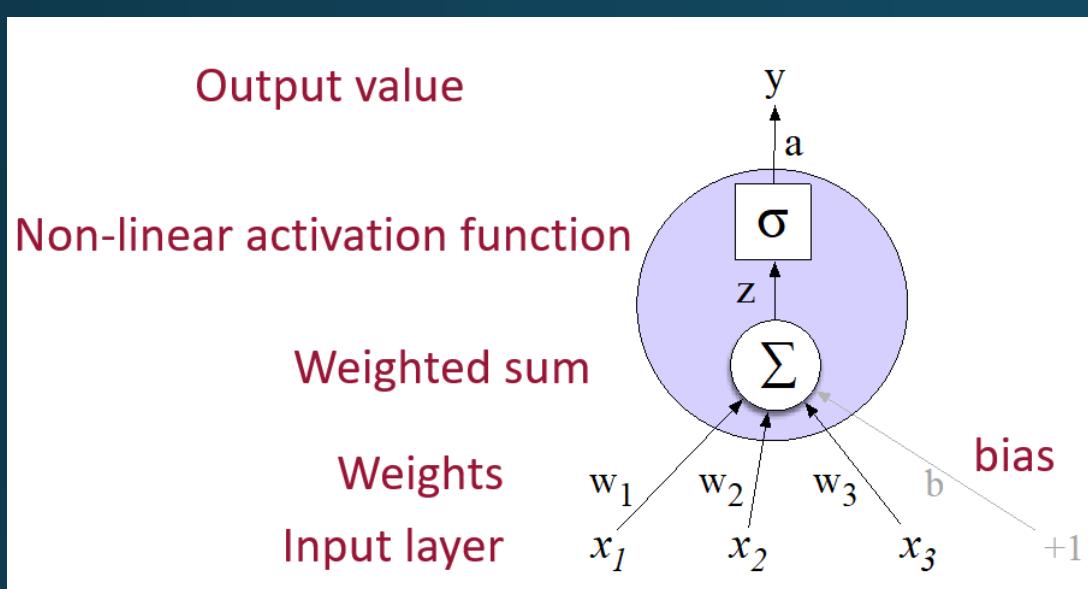
$$y = \sigma(z) = \frac{1}{1+e^{-z}}$$

- activation function : z is a linear function of x . Instead of using z as the output, neural units apply a non-linear function f to z .
- The output of the function is called activation value, which is the final output of the unit.
- Sigmoid is an activation function with lot of advantages.
 - It maps the output into the range $[0,1]$, which is useful in squashing outliers toward 0 or 1.
 - It's differentiable
- the output of the neural unit after sigmoid substitution is

$$y = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + \exp(-(\mathbf{w} \cdot \mathbf{x} + b))}$$



Neuron – unit (final schematic)



Let's suppose we have a unit with the following weight vector and bias:

$$\mathbf{w} = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

What would this unit do with the following input vector?

$$\mathbf{x} = [0.5, 0.6, 0.1]$$

$$\begin{aligned}y &= \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \\&= \frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70\end{aligned}$$

Activation Functions

- z is a linear function of x . Instead of using z as the output, neural units apply a non-linear function f to z . The output of this function is the activation value (a) for the unit.
- The activation value is the final output of the unit, and we generally refer it as y .

$$y = a = f(z)$$

- There are three popular non-linear functions.
 - the sigmoid,
 - the tanh,
 - the ReLU.

Activation Function - Sigmoid

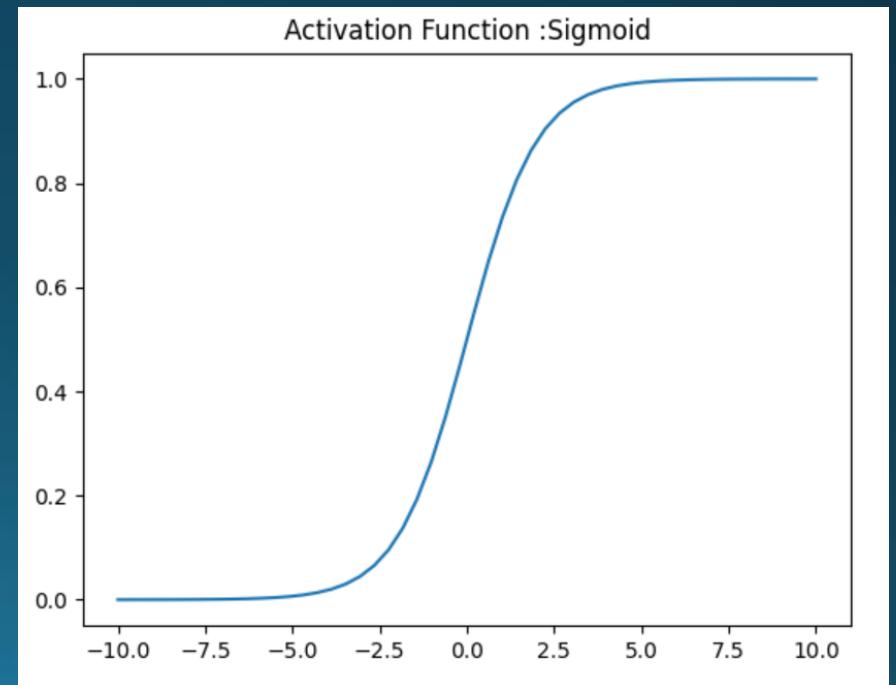
- What do you know about sigmoid?
- What is the derivative of sigmoid function?

$$\sigma'(x) = \frac{1}{(1+e^{-x})} \left(1 - \frac{1}{(1+e^{-x})} \right)$$
$$\sigma'(x) = \sigma(x) \cdot (1 - \sigma(x))$$

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1+np.exp(-x))

x = np.linspace(-10, 10)
plt.plot(x, sigmoid(x))
plt.axis('tight')
plt.title('Activation Function :Sigmoid')
plt.show()
```

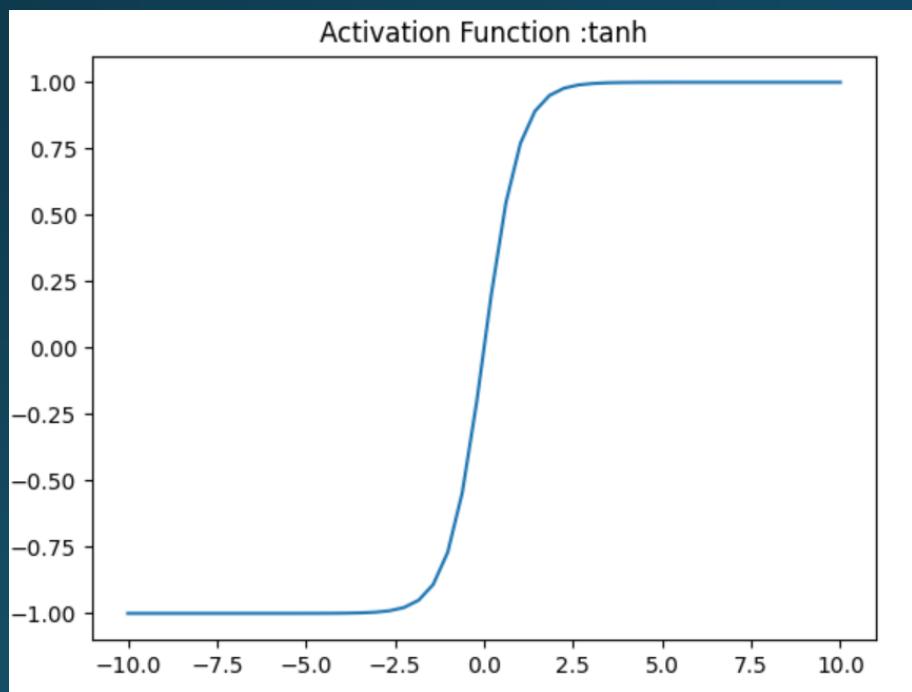


Activation function - tanh

- tanh is similar to sigmoid and almost always better than sigmoid.
- Its value ranges from -1 to +1.
- tanh is shown in below figure and it is calculated as.
- What is the derivative of tanh?

$$y = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$\frac{da}{dz} = 1 - a^2$$



```
def tanh(x):  
    return np.tanh(x)
```

Activation function - ReLU

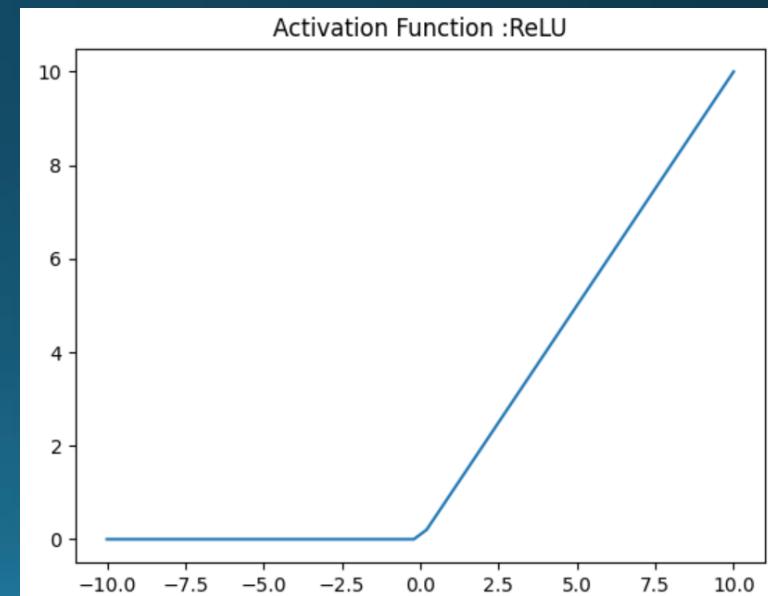
- ReLU stands for rectified linear unit.
- It is the simplest activation function, and perhaps the most commonly used.
- It's just the same as z when z is positive, and 0 otherwise.
- ReLU is shown in below figure and it is calculated using.

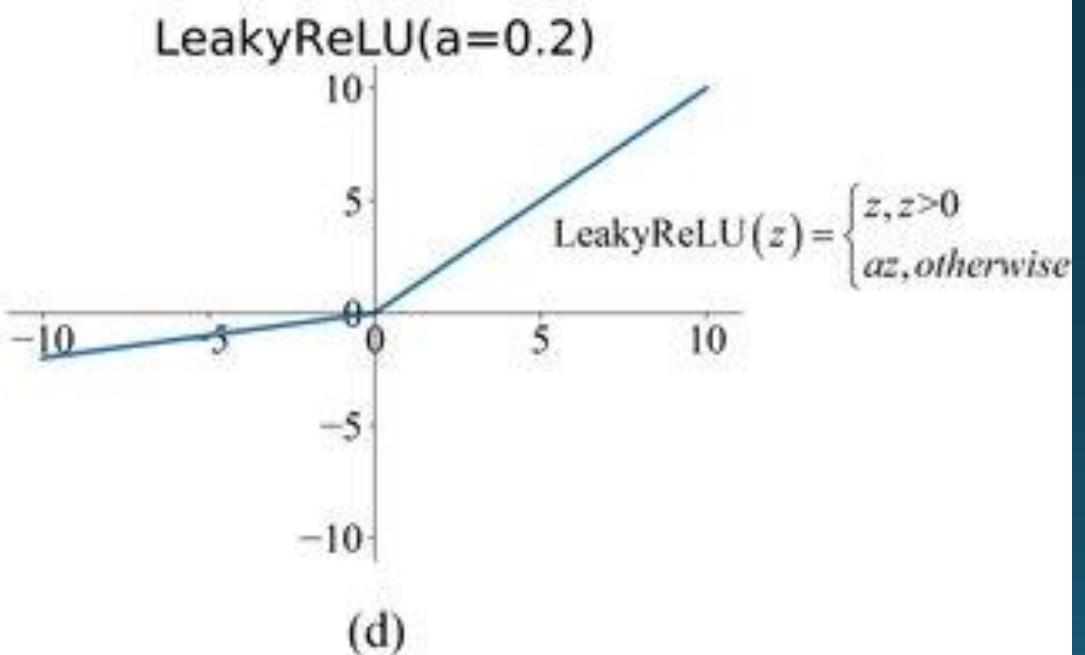
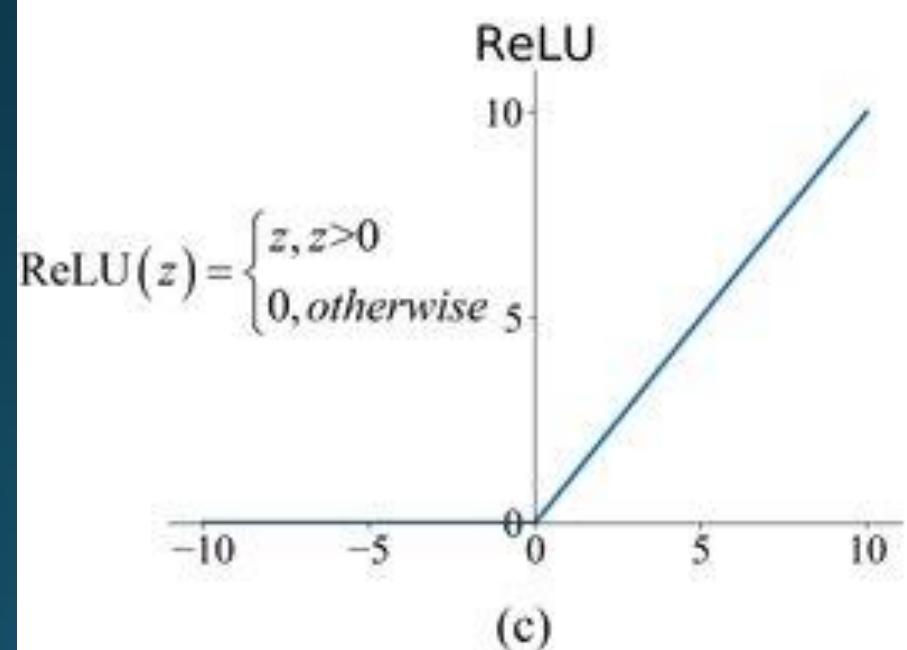
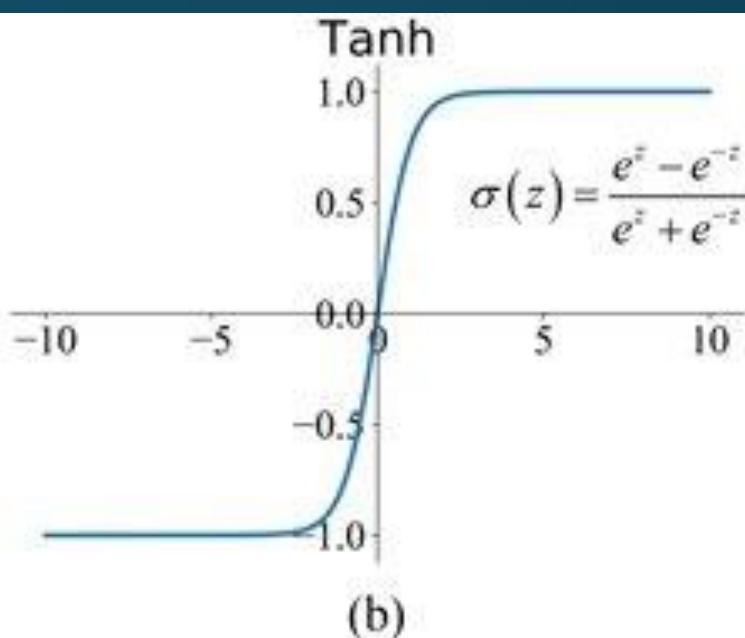
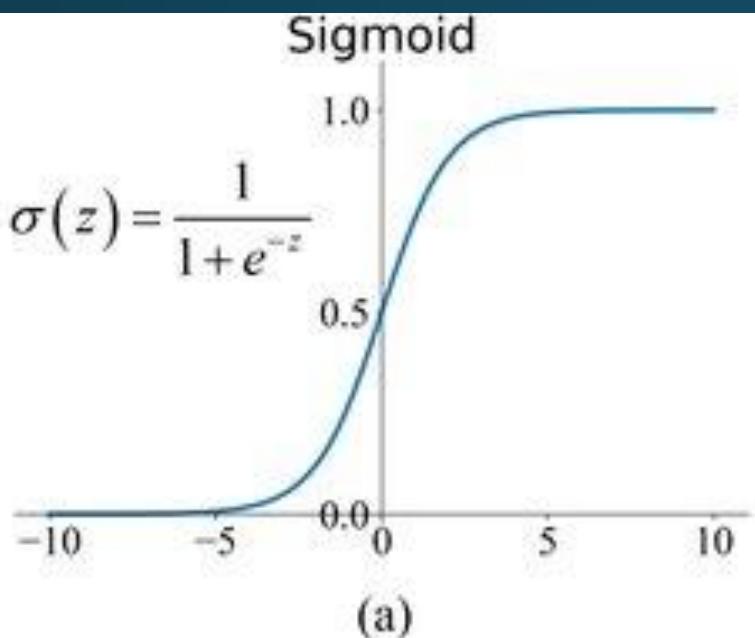
$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

$$y = \max(z, 0)$$

- What is the derivative of ReLU?

```
def RELU(x):  
    x1=[]  
    for i in x:  
        if i<0:  
            x1.append(0)  
        else:  
            x1.append(i)  
  
    return x1
```

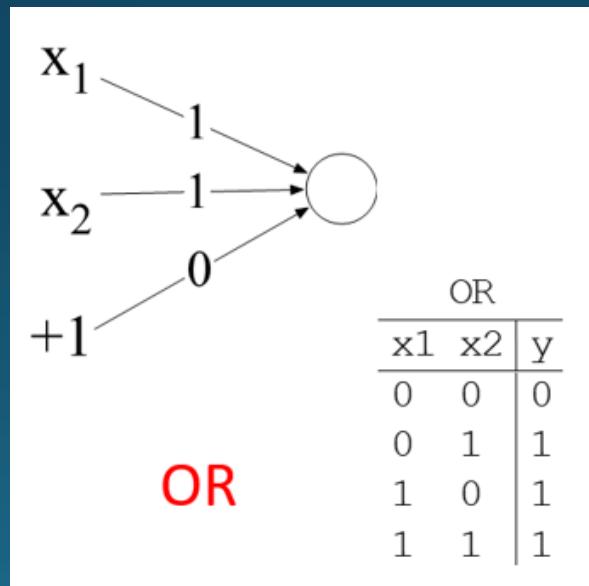
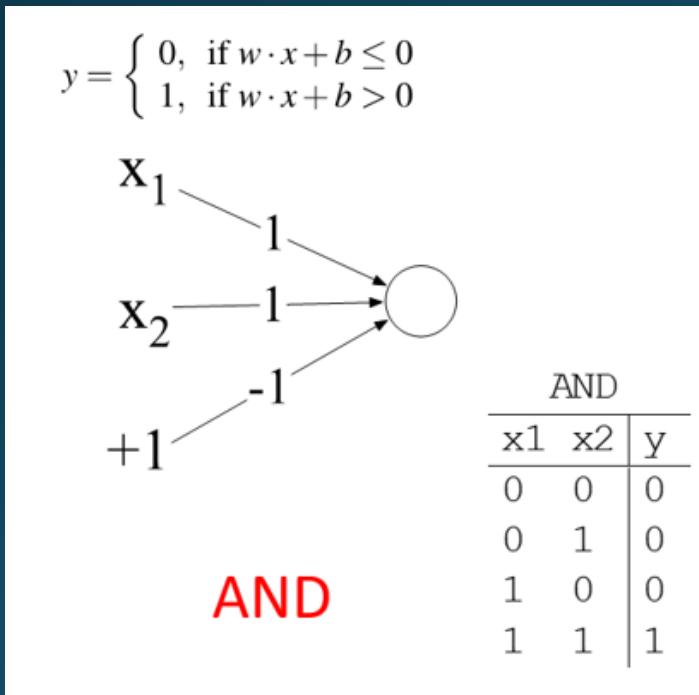




XOR Problem

- Do you remember Perceptron?
- Lets compute the elementary logical functions AND, OR, and XOR using perceptron?

$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

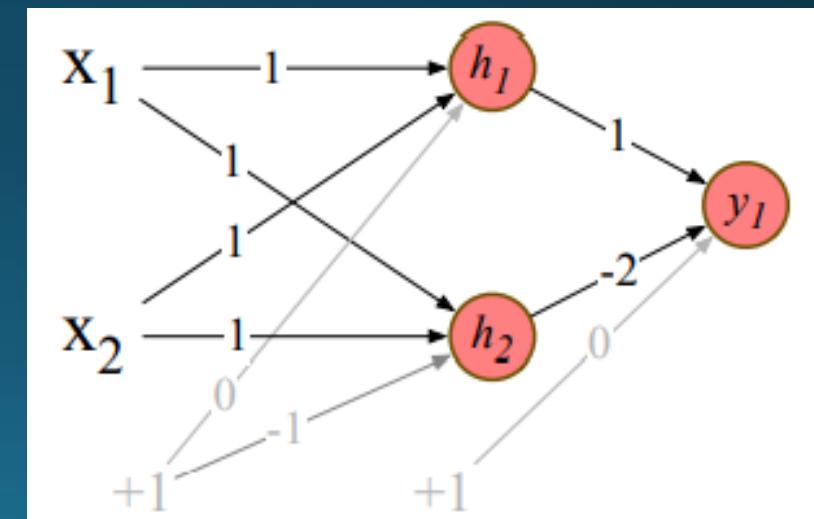
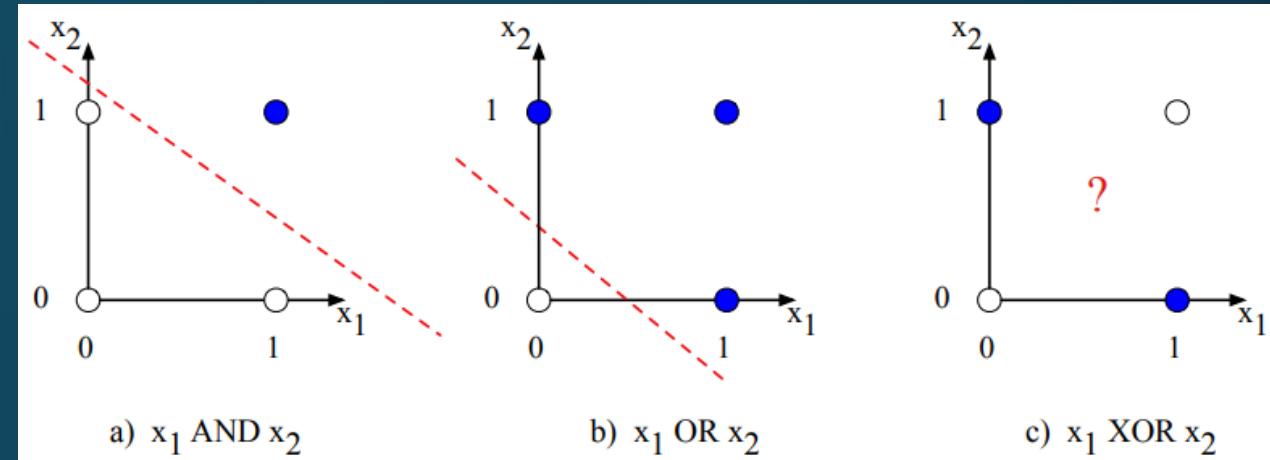


AND		OR		XOR	
x1	x2	y	x1	x2	y
0	0	0	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	0

Build a perceptron
to find XOR?

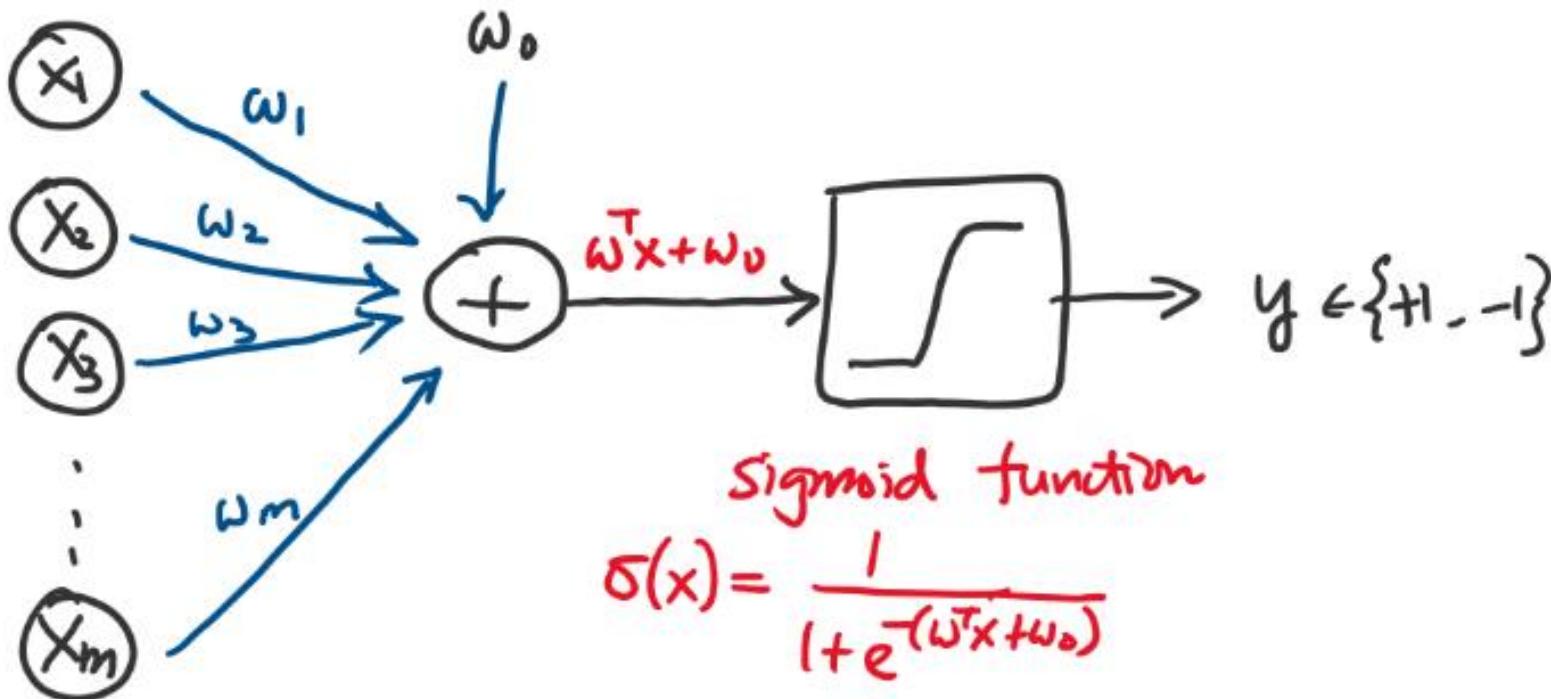
XOR problem

- Perceptron is a linear classifier
- XOR is not a linearly separable function.
- Can we draw a line that separates the positive cases of XOR (01 and 10) from the negative cases (00 and 11)?
- Solution
 - XOR can't be calculated by a single perceptron
 - XOR can be calculated by a layered network of units
- This is the basic for feed forward neural network.

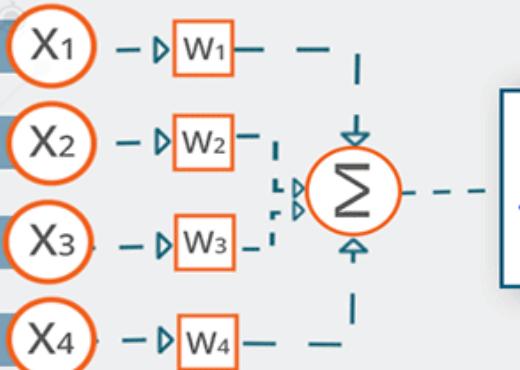


Multilayer Perceptron

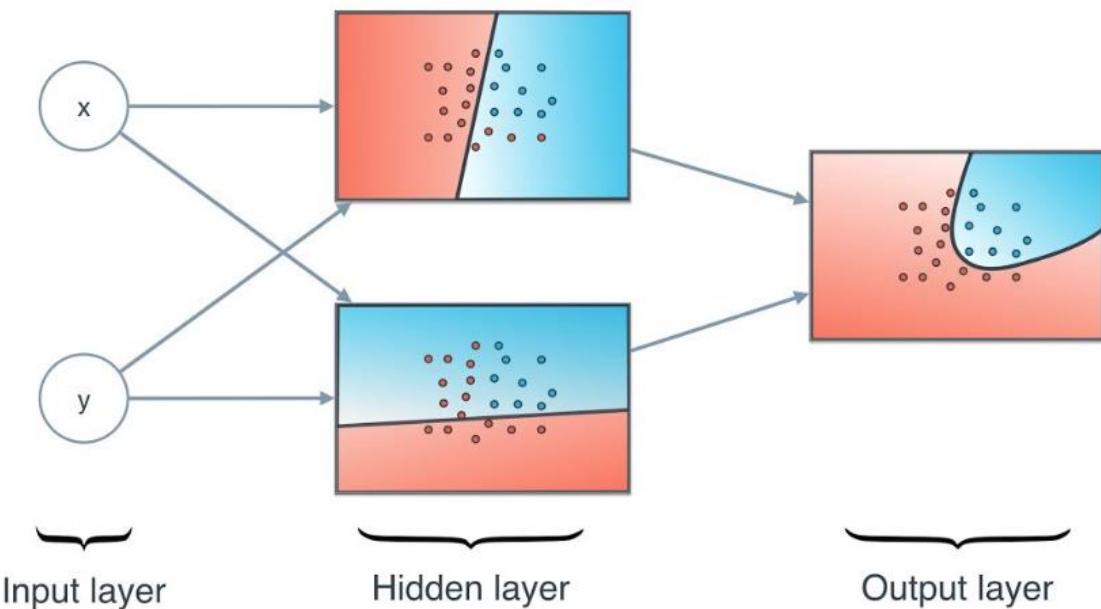
Single Layer Perceptron



- Input neurons x
- Weights w
- Predicted label $= \sigma(w^T x + w_0)$.

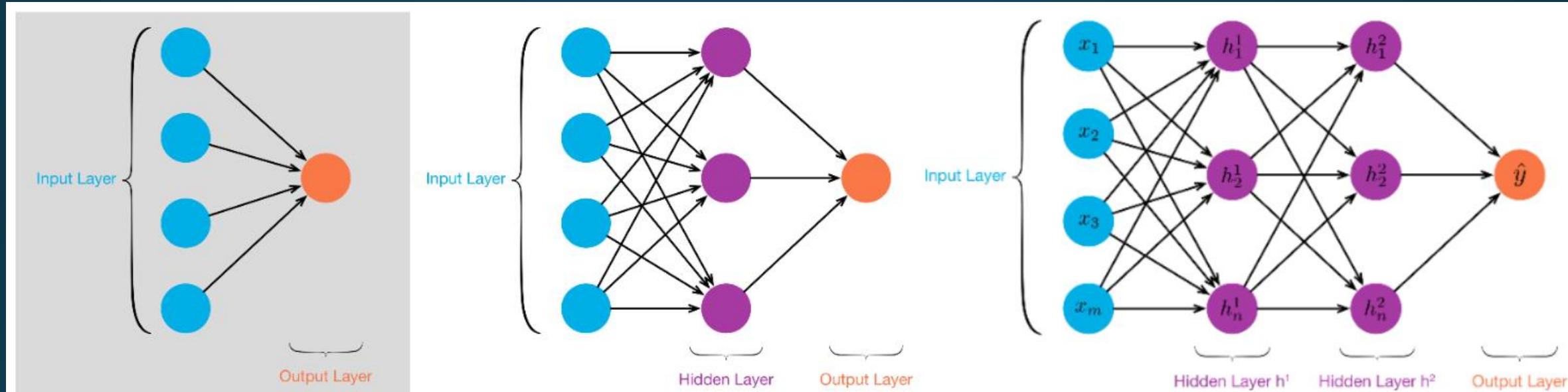


Perceptron Learning Algorithm



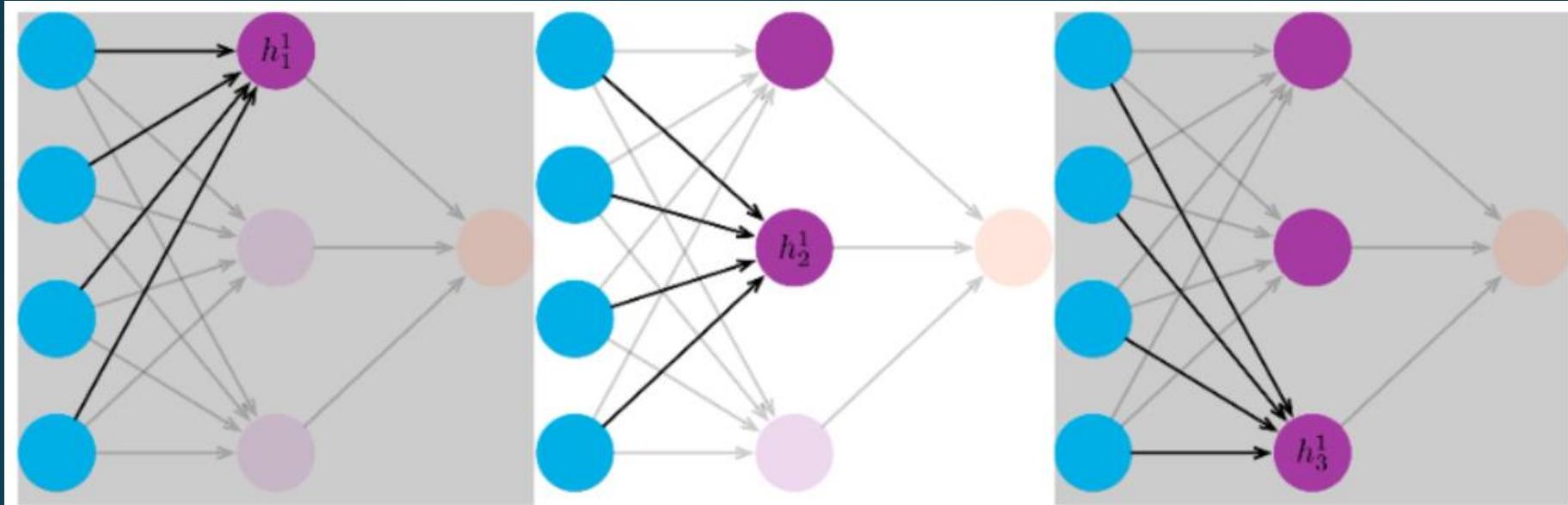
Combining Neurons allows Neural Networks to learn more complex, Nonlinear Decision Boundaries

Multi-Layer Network



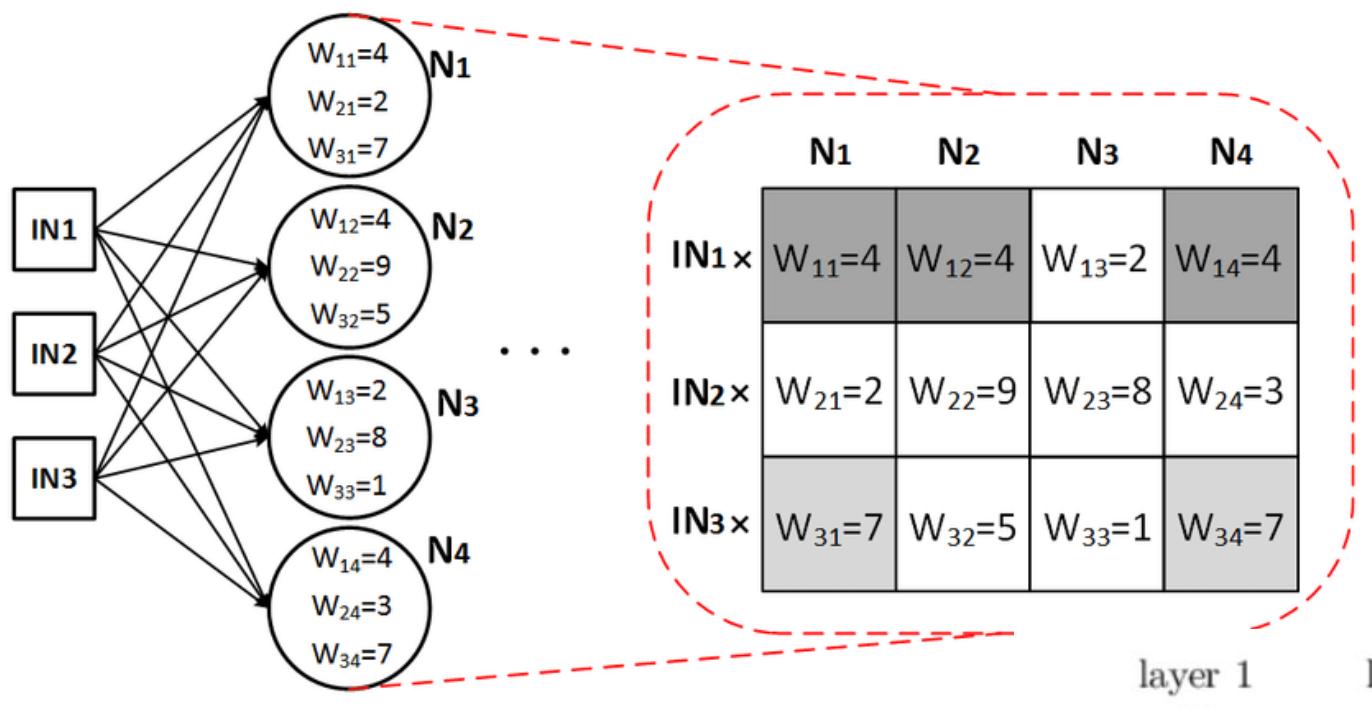
- Introduce a layer of hidden neurons
- So now you have two sets of weights: from input to hidden, and from hidden to output
- You can introduce as many hidden layers as you want.
- Every time you add a hidden layer, you add a set of weights.

Understanding the weights

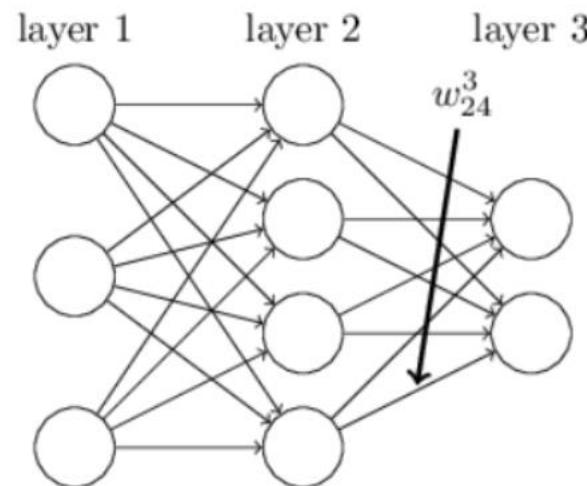


- Each hidden neuron is an **output** of a perceptron
- So you will have

$$\begin{bmatrix} h_1^1 \\ h_2^1 \\ \dots \\ h_n^1 \end{bmatrix} = \begin{bmatrix} w_{11}^1 & w_{12}^1 & \dots & w_{1n}^1 \\ w_{21}^1 & w_{22}^1 & \dots & w_{2n}^1 \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}^1 & w_{m2}^1 & \dots & w_{mn}^1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$



	N1	N2	N3	N4
IN1 ×	$W_{11}=4$	$W_{12}=4$	$W_{13}=2$	$W_{14}=4$
IN2 ×	$W_{21}=2$	$W_{22}=9$	$W_{23}=8$	$W_{24}=3$
IN3 ×	$W_{31}=7$	$W_{32}=5$	$W_{33}=1$	$W_{34}=7$



- w_{24}^3 : The 3rd layer
- w_{24}^3 : From 4-th neuron to 2-nd neuron

w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

Progression to Deep(Linear) Neural Networks

Single-layer:

$$h = \mathbf{w}^T \mathbf{x}$$

Hidden-layer:

$$\mathbf{h} = \mathbf{W}^T \mathbf{x}$$

Two Hidden Layers:

$$\mathbf{h} = \mathbf{W}_2^T \mathbf{W}_1^T \mathbf{x}$$

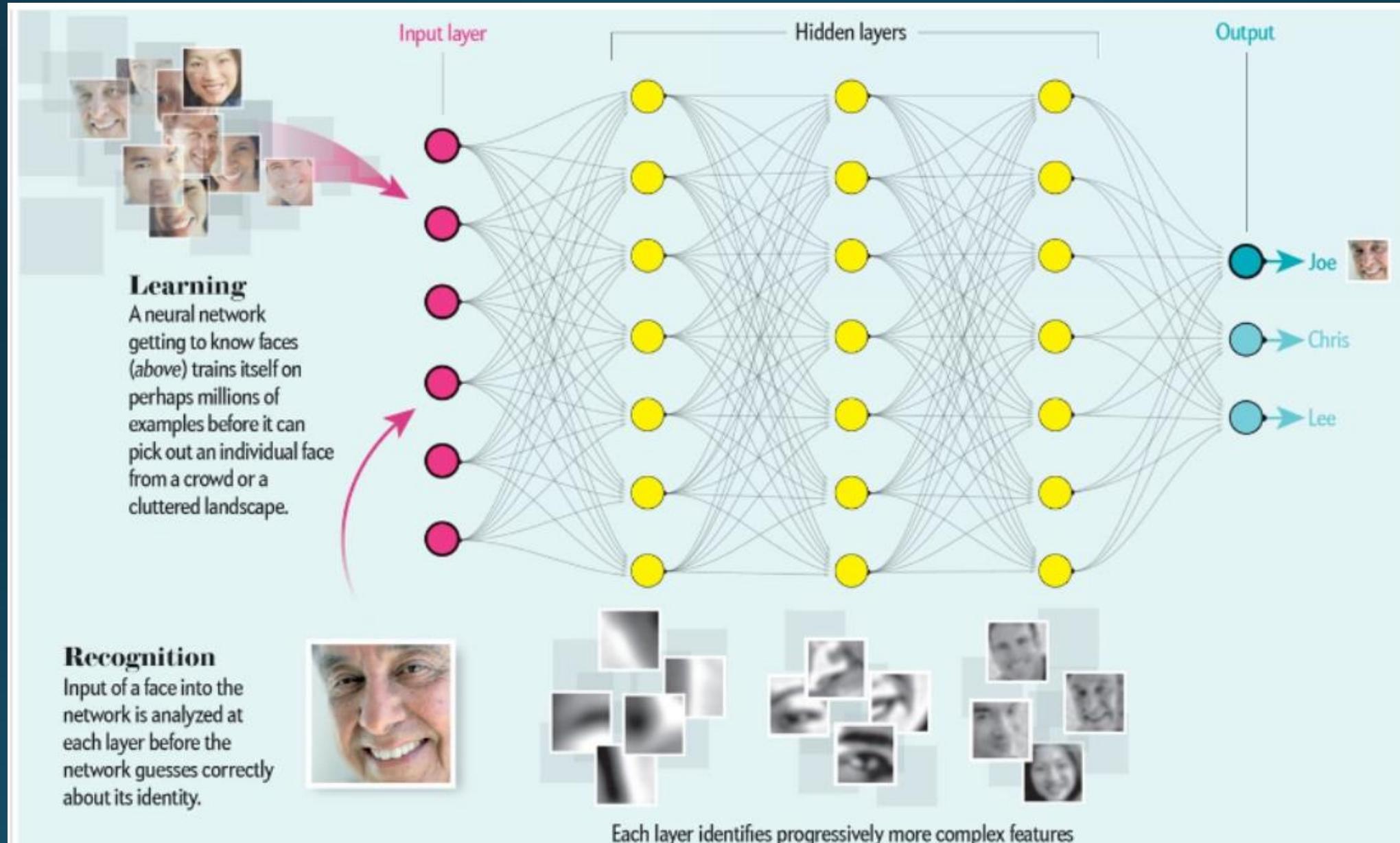
Three Hidden Layers:

$$\mathbf{h} = \mathbf{W}_3^T \mathbf{W}_2^T \mathbf{W}_1^T \mathbf{x}$$

A LOT of Hidden Layers:

$$\mathbf{h} = \mathbf{W}_N^T \dots \mathbf{W}_2^T \mathbf{W}_1^T \mathbf{x}$$

Interpreting the hidden layer

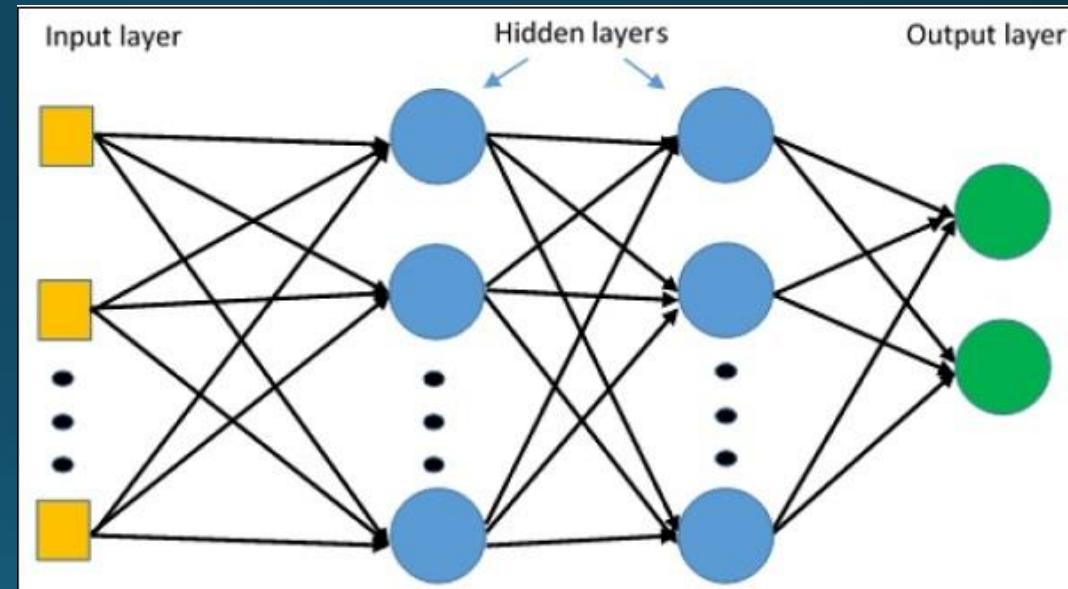


Multilayer Perceptron – Feedforward Neural Network

- A neural network where the mapping between inputs and outputs is non-linear.
- A Multilayer perceptron has input and output layers, and one or more hidden layers with many neurons stacked together.
- It comes under a feedforward algorithms - inputs are combined with initial weights - subjected to activation function - propagated to the next layer.
- The units are arranged into a set of layers.
- Each layer contains some identical units
- Every unit in one layer is connected to every unit in the next layer - This network is called fully connected neural network.

Multilayer Perceptron – Feedforward Neural Network

- **Input layer** - first layer - takes the values of the input features.
- **Output layer** - Last layer - has one unit for each value the network outputs
- How many units in output layer for
 - Regression
 - Binary Classification?
 - K-class classification
- **Hidden Layer** - The layer in between input and output layers
- The units in these layers are known as **input units, output units, and hidden units**.
- The number of layers is known as the **depth**, and the number of units in a layer is known as the **width**
- **Deep Learning** - refers to training neural nets with many layers



What is next in multilayer network

- How do we efficiently learn the weights?
 - Our ultimate goal is to minimize the loss

$$J(\mathbf{w}_1, \dots, \mathbf{w}_L) = \sum_{i=1}^N \|\sigma(\mathbf{w}_L^T \dots \sigma(\mathbf{w}_2^T \sigma(\mathbf{w}_1^T \mathbf{x}_i))) - \mathbf{y}_i\|^2$$

- Single Layer – Gradient Descent
- Multi Layer - ???
- Back propagation is very powerful book keeping and chain rule used in multi layer network training.

softmax

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^k \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^k \exp(z_i)}, \dots, \frac{\exp(z_k)}{\sum_{i=1}^k \exp(z_i)} \right]$$

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^k \exp(z_j)} \quad 1 \leq i \leq k$$

Example:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

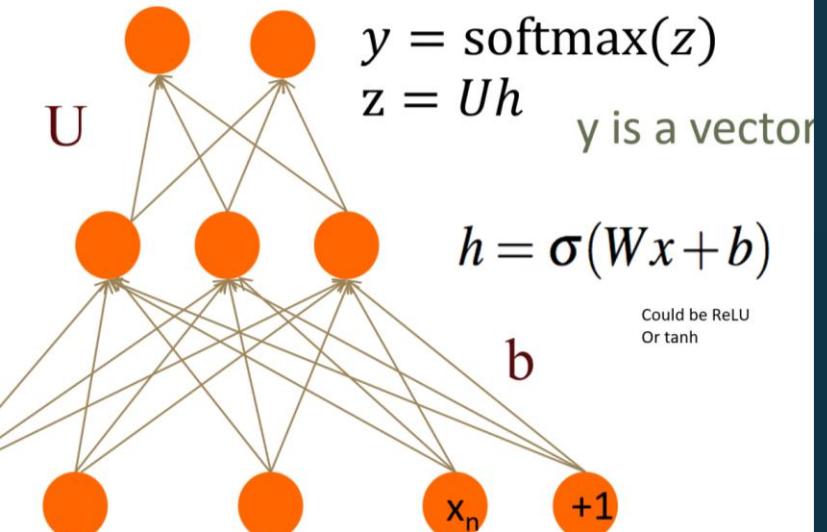
$$\text{softmax}(z) = [0.055, 0.090, 0.006, 0.099, 0.74, 0.010]$$

Two-Layer Network with softmax output

Output layer
(σ node)

hidden units
(σ node)

Input layer
(vector)

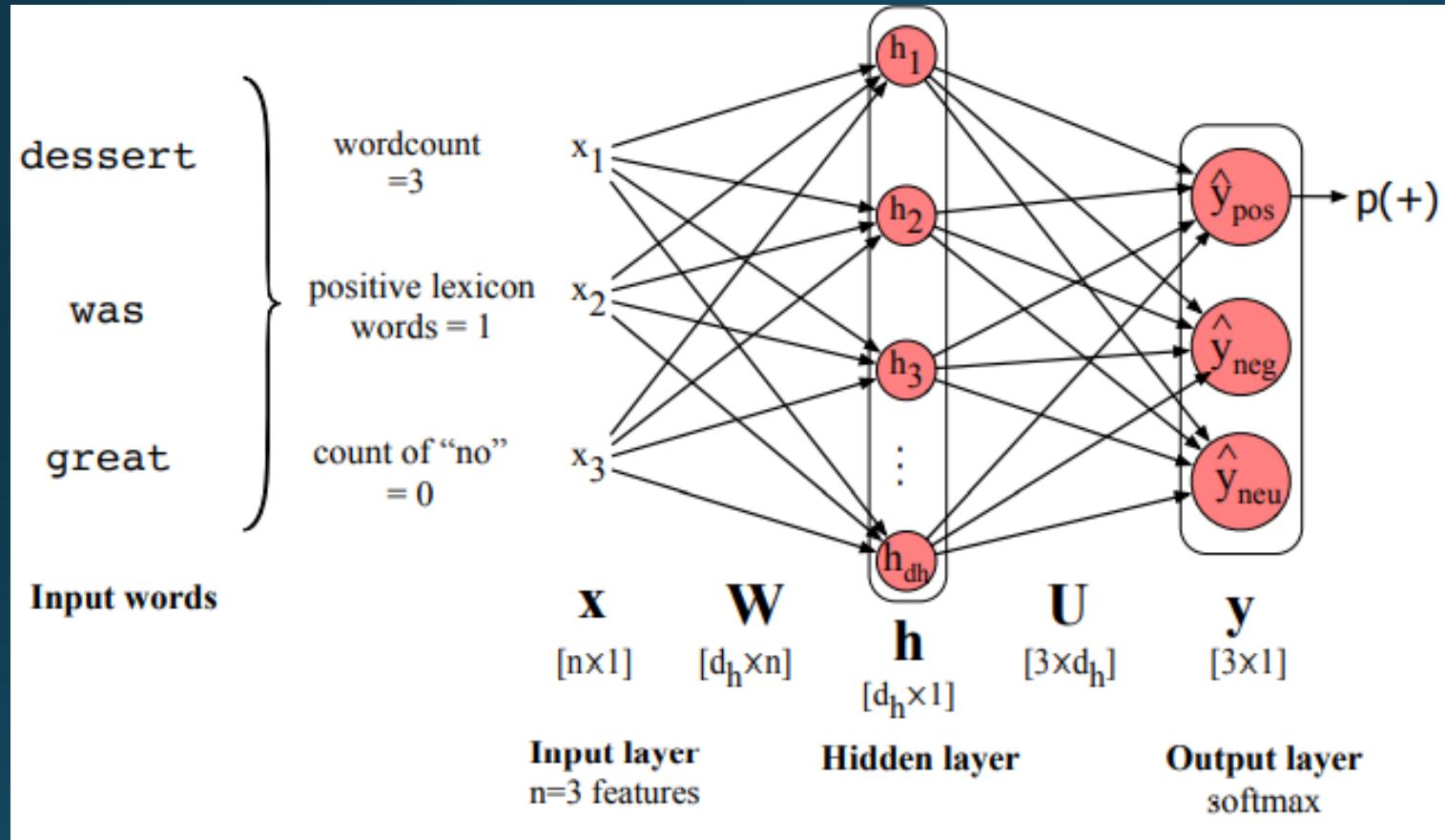


$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{z} = \mathbf{U}\mathbf{h}$$

$$\mathbf{y} = \text{softmax}(\mathbf{z})$$

An example network



x = vector of hand-designed features
h = $\sigma(\mathbf{Wx} + \mathbf{b})$
z = \mathbf{Uh}
y = softmax(**z**)

B

1. Input x : Set the corresponding activation a^1 for the input layer.

2. Feedforward: For each $l = 2, 3, \dots, L$ compute

$$z^l = w^l a^{l-1} + b^l \text{ and } a^l = \sigma(z^l).$$

3. Output error δ^L : Compute the vector $\delta^L = \nabla_a C \odot \sigma'(z^L)$.

4. Backpropagate the error: For each $l = L - 1, L - 2, \dots, 2$ compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$.

5. Output: The gradient of the cost function is given by

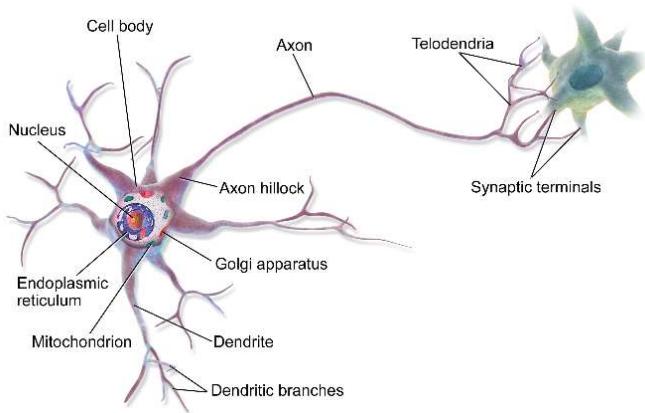
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \text{ and } \frac{\partial C}{\partial b_j^l} = \delta_j^l.$$

Biological Neuron

Motivation behind neural network is human brain.

Human brain is called as the best processor even though it works slower than other computers. Many researchers thought to make a machine that would work in the prospective of the human brain.

Human brain contains billion of neurons which are connected to many other neurons to form a network so that if it sees any image, it recognizes the image and processes the output.



- Dendrite receives signals from other neurons.
- Cell body sums the incoming signals to generate input.
- When the sum reaches a threshold value, neuron fires and the signal travels down the axon to the other neurons.
- The amount of signal transmitted depend upon the strength of the connections.
- Connections can be inhibitory, i.e. decreasing strength or excitatory, i.e. increasing strength in nature.

In the similar manner, it was thought to make artificial interconnected neurons like biological neurons making up an Artificial Neural Network(ANN).

Each biological neuron is capable of taking a number of inputs and produce output.

Neurons in human brain are capable of making very complex decisions, so this means they run many parallel processes for a particular task. One motivation for ANN is that to work for a particular task identification through many parallel processes.

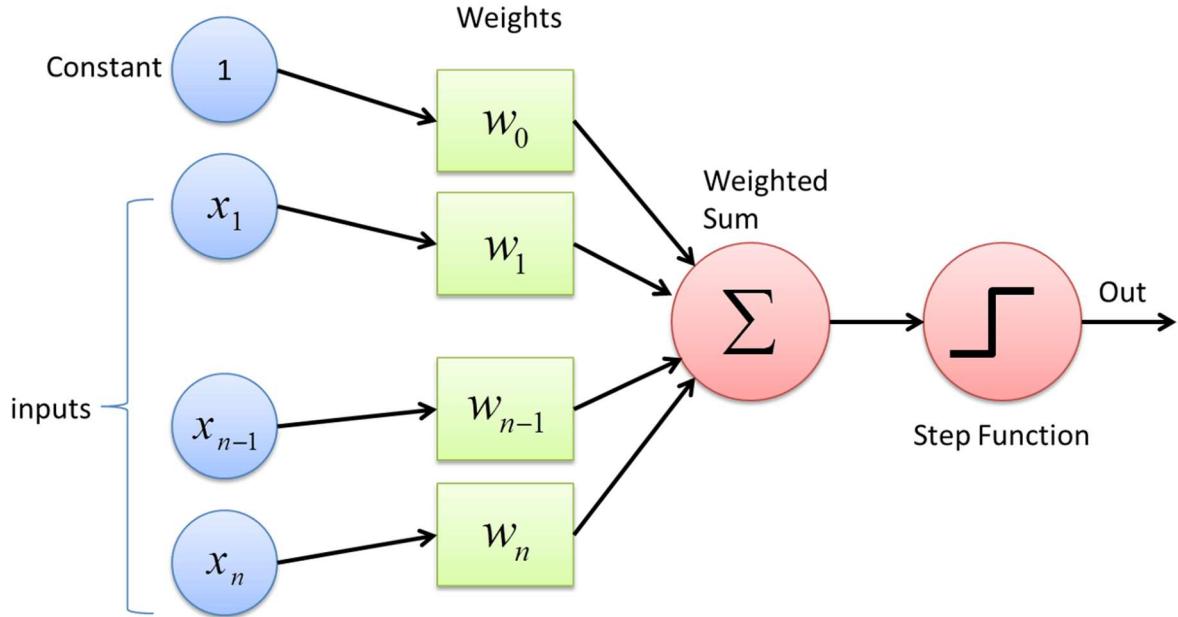
a modern neural network is a network of small computing units, each of which takes a vector of input values and produces a single output value.

Neural networks share much of the same mathematics as logistic regression. But neural networks are a more powerful classifier than logistic regression

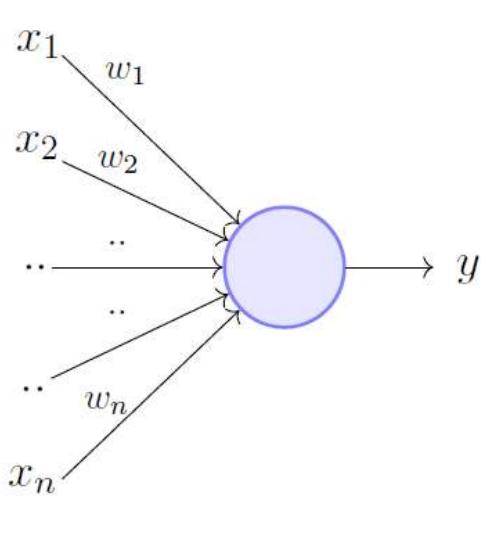
Perceptron Learning

The perceptron model is a more general computational model.

It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0.



A step function is **is used by perceptron**. The output of step function is a certain value, A_1 , if the input sum is above a certain threshold and A_0 if the input sum is below a certain threshold.

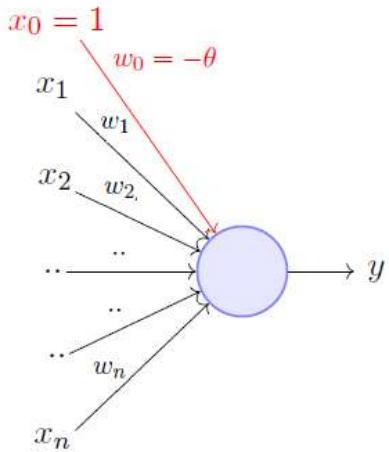


$$y = 1 \quad \text{if } \sum_{i=1}^n w_i * x_i \geq \theta \\ = 0 \quad \text{if } \sum_{i=1}^n w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad \text{if } \sum_{i=1}^n w_i * x_i - \theta \geq 0 \\ = 0 \quad \text{if } \sum_{i=1}^n w_i * x_i - \theta < 0$$

Rewriting the threshold as shown above and making it a constant input with a variable weight, we would end up with something like the following:



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^n w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^n w_i * x_i < 0$$

where, $x_0 = 1$ and $w_0 = -\theta$

A single perceptron can only be used to implement **linearly separable** functions. It takes both real and boolean inputs and associates a set of **weights** to them, along with a **bias** (the threshold mentioned above). We learn the weights, we get the function. Let's use a perceptron to learn an OR function.

OR function using a perceptron:

x_1	x_2	OR
0	0	0 $w_0 + \sum_{i=1}^2 w_i x_i < 0$
1	0	1 $w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
0	1	1 $w_0 + \sum_{i=1}^2 w_i x_i \geq 0$
1	1	1 $w_0 + \sum_{i=1}^2 w_i x_i \geq 0$

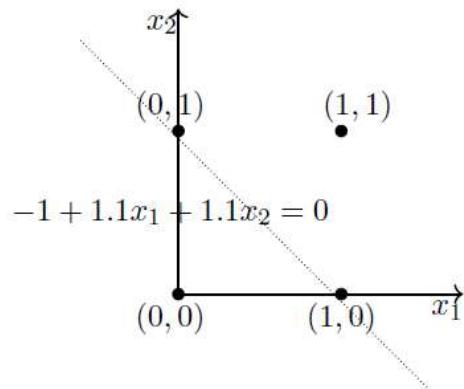
$$w_0 + w_1 \cdot 0 + w_2 \cdot 0 < 0 \implies w_0 < 0$$

$$w_0 + w_1 \cdot 0 + w_2 \cdot 1 \geq 0 \implies w_2 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 0 \geq 0 \implies w_1 \geq -w_0$$

$$w_0 + w_1 \cdot 1 + w_2 \cdot 1 \geq 0 \implies w_1 + w_2 \geq -w_0$$

One possible solution is
 $w_0 = -1, w_1 = 1.1, w_2 = 1.1$



Units

- The building block of a neural network is a single computational **unit**.
- A unit takes a set of real valued numbers as input, performs some computation on them, and produces an output.
- A neural unit takes a weighted sum of its inputs, with one additional term in the sum called a **bias term**.
- Given a set of inputs $x_1 \dots x_n$, a unit has a set of corresponding weights $w_1 \dots w_n$ and a bias b , so the weighted sum z can be represented as:

$$z = b + \sum_i w_i x_i$$

- It is more convenient to express this weighted sum using vector notation.

$$z = \mathbf{w} \cdot \mathbf{x} + b$$

here z is a real valued number.

- **activation function :** z is a linear function of X . Instead of using z as the output, neural units apply a non-linear function f to z . The output of this function is the activation value (a) for the unit.
- The activation value is the final output of the unit, and we generally refer it as y .

$$y = a = f(z)$$

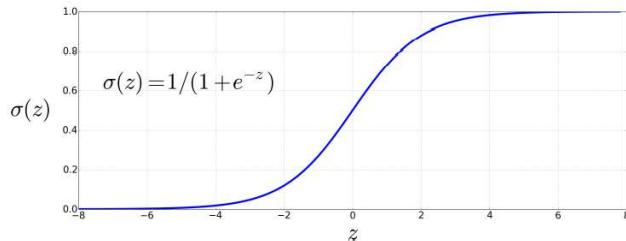
- There are three popular non-linear functions.
 - the sigmoid,
 - the tanh,
 - the ReLU.

Sigmoid :

- It is always convenient to start with the sigmoid function.

$$y = \sigma(z) = \frac{1}{1 + e^{-z}}$$

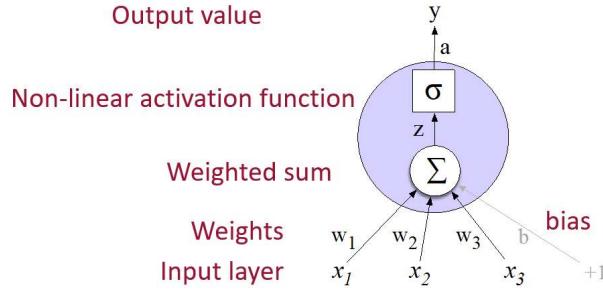
- The sigmoid shown in the below figure has a number of advantages;
 - it maps the output into the range $[0,1]$, which is useful in squashing outliers toward 0 or 1.
 - It's differentiable.



- the output of the neural unit after sigmoid substitution is

$$y = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + \exp(-(\mathbf{w} \cdot \mathbf{x} + b))}$$

- The final schematic of a basic neural unit is shown in the below figure.



- In this example the unit takes 3 input values x_1, x_2 , and x_3 , and computes a weighted sum, multiplying each value by a weight (w_1, w_2 , and w_3 , respectively), adds them to a bias term b , and then passes the resulting sum through a sigmoid function to result in a number between 0 and 1.

Example:

- Let's suppose we have a unit with the following weight vector and bias:

$$\mathbf{w} = [0.2, 0.3, 0.9]$$

$$b = 0.5$$

What would this unit do with the following input vector?

$$\mathbf{x} = [0.5, 0.6, 0.1]$$

The resulting output y would be:

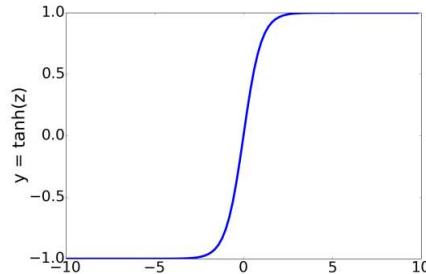
$$\begin{aligned}
 y &= \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \\
 &= \frac{1}{1 + e^{-(.5*.2+.6*.3+.1*.9+.5)}} = \frac{1}{1 + e^{-0.87}} = .70
 \end{aligned}$$

**Note : Calculate output y using $tanh$ and $ReLU$ also.

tanh:

- $tanh$ is similar to sigmoid and almost always better than sigmoid.
- Its value ranges from -1 to +1.
- $tanh$ is shown in below figure and it is calculated as.

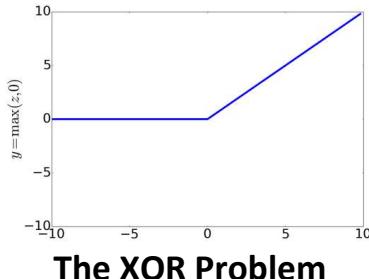
$$y = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$



ReLU:

- ReLU stands for rectified linear unit.
- It is the simplest activation function, and perhaps the most commonly used.
- It's just the same as z when z is positive, and 0 otherwise.
- ReLU is shown in below figure and it is calculated using.

$$y = \max(z, 0)$$



The XOR Problem

- **perceptron** : perceptron is a very simple neural unit that has a binary output and does not have a non-linear activation function. The output y of a perceptron is 0 or 1, and is computed as follows.

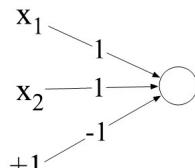
$$y = \begin{cases} 0, & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq 0 \\ 1, & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \end{cases}$$

- Consider the task of computing elementary logical functions of two inputs, like AND, OR, and XOR. The truth tables for those functions are shown below.

		AND		OR		XOR		
x_1	x_2	y	x_1	x_2	y	x_1	x_2	y
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

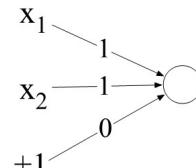
- It's very easy to build a perceptron that can compute the logical AND and OR functions of its binary inputs;

$$y = \begin{cases} 0, & \text{if } w_1x_1 + w_2x_2 + b \leq 0 \\ 1, & \text{if } w_1x_1 + w_2x_2 + b > 0 \end{cases}$$



AND

AND		
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

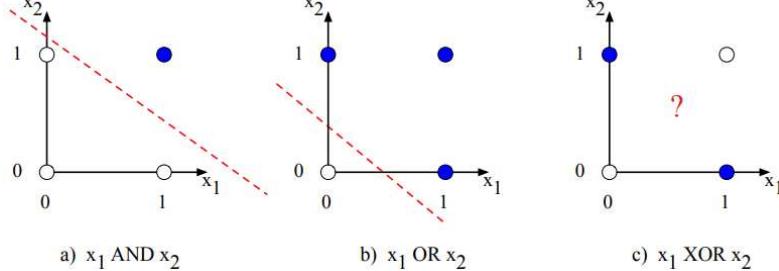


OR

OR		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

- But it's not possible to build a perceptron to compute logical XOR!
- The reason for this is "perceptron is a linear classifier". For a two-dimensional input x_1 and x_2 , the perception equation, $w_1x_1 + w_2x_2 + b = 0$ is the equation of a line.

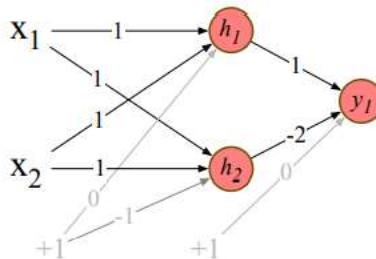
- This line acts as a decision boundary in two-dimensional space in which the output 0 is assigned to all inputs lying on one side of the line, and the output 1 to all input points lying on the other side of the line.
- Following figure shows the possible logical inputs (00, 01, 10, and 11) and the line drawn by one possible set of parameters for an AND and an OR classifier.



- Notice that there is simply no way to draw a line that separates the positive cases of XOR (01 and 10) from the negative cases (00 and 11). We say that XOR is not a linearly separable function.

The solution : neural network

- XOR can't be calculated by a single perceptron
- XOR can be calculated by a layered network of units.
- The below network shows a figure with the input being processed by two layers of neural units. The middle layer (called h) has two units, and the output layer (called y) has one unit. A set of weights and biases are shown for each ReLU that correctly computes the XOR function.



- Consider input $x = [0, 0]$. If we multiply each input value by the appropriate weight, sum, and then add the bias b , we get the vector $[0 -1]$, apply the ReLU and we get $[0 0]$, and the final value is 0.
- The solution to the XOR problem requires a network of units with nonlinear activation functions.

It is a neural network where the mapping between inputs and output is non-linear.

A Multilayer perceptron has input and output layers, and one or more hidden layers with many neurons stacked together.

Multilayer Perceptron falls under the category of feedforward algorithms, because inputs are combined with the initial weights in a weighted sum and subjected to the activation function, just like in the Perceptron. But the difference is that each linear combination is propagated to the next layer.

Each layer is feeding the next one with the result of their computation, their internal representation of the data. This goes all the way through the hidden layers to the output layer.

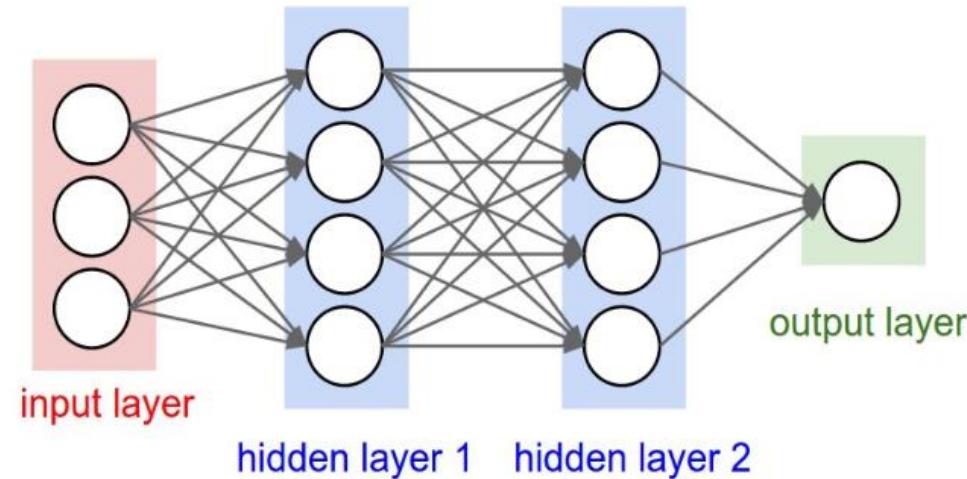
But it has more to it.

If the algorithm only computed the weighted sums in each neuron, propagated results to the output layer, and stopped there, it wouldn't be able to learn the weights that minimize the cost function. If the algorithm only computed one iteration, there would be no actual learning.

WHAT IS CNN?

CNN = Neural Network with a convolution operation instead of matrix multiplication in at least one of the layers

NEURAL NETWORKS



Input example : one image



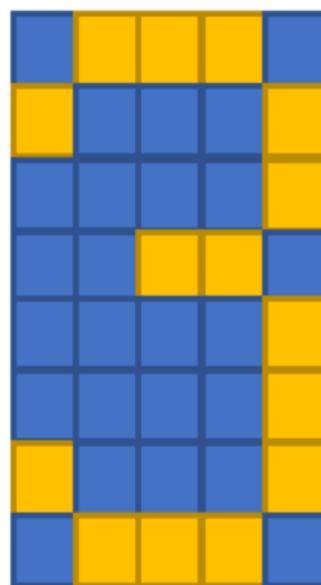
Output example : one class

airplane	dog
automobile	frog
bird	horse
cat	ship
deer	truck

What digit do you think this pattern represents?



1D matrix



2D matrix

	1	2	3	4
1				
2				
3				
4				



Input



KERNEL AND CONVOLUTION

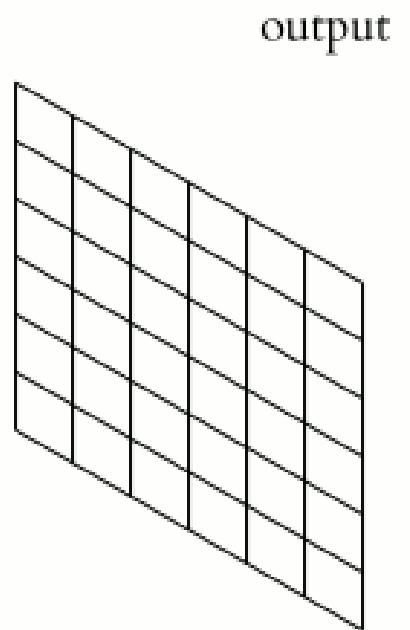
- *Convolution is using a ‘kernel’ to extract certain ‘features’ from an input image..*
- A kernel is a matrix, which is slid across the image and multiplied with the input such that the output is enhanced in a certain desirable manner.
- Convolution is
 - when our kernel is multiplied with our image,
 - when we apply one function to another function, but we can think of our base image as a function of color, and our kernel as a function of pixels.

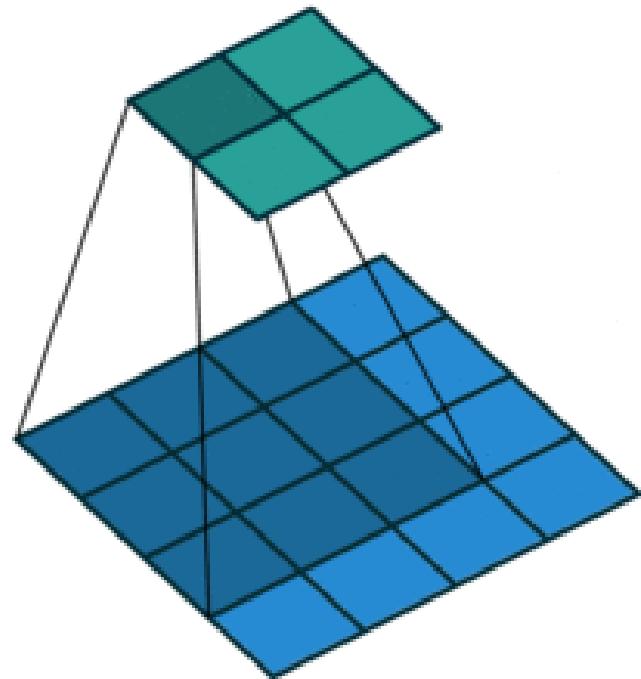
$$\begin{array}{c}
 \text{Kernel} \\
 \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}
 \end{array}
 \times
 \begin{array}{c}
 \text{Image} \\
 \begin{array}{|c|c|c|} \hline 2 & 2 & 2 \\ \hline 2 & 3 & 2 \\ \hline 2 & 2 & 2 \\ \hline \end{array}
 \end{array}
 \equiv
 \begin{array}{c}
 \text{Output} \\
 \boxed{7}
 \end{array}$$

$$\begin{array}{c}
 \text{Kernel} \\
 \begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}
 \end{array}
 \times
 \begin{array}{c}
 \text{Image} \\
 \begin{array}{|c|c|c|} \hline 2 & 2 & 2 \\ \hline 2 & 1 & 2 \\ \hline 2 & 2 & 2 \\ \hline \end{array}
 \end{array}
 \equiv
 \begin{array}{c}
 \text{Output} \\
 \boxed{-3}
 \end{array}$$

input

7	6	5	5	6	7
6	4	3	5	4	6
5	3	2	3	3	5
6	3	2	2	3	6
7	4	3	2	3	5
6	6	5	3	4	6
7	6	5	5	6	7





1 <small>x1</small>	1 <small>x0</small>	1 <small>x1</small>	0	0
0 <small>x0</small>	1 <small>x1</small>	1 <small>x0</small>	1	0
0 <small>x1</small>	0 <small>x0</small>	1 <small>x1</small>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Original Image



Kernel

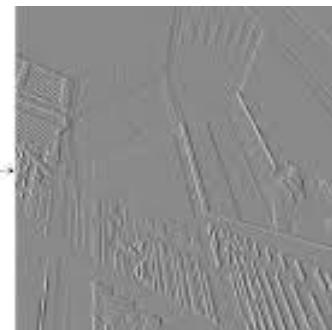
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Filtered Image



$$\begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

Horizontal Sobel kernel



*

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$



KERNELS AND CONVOLUTION

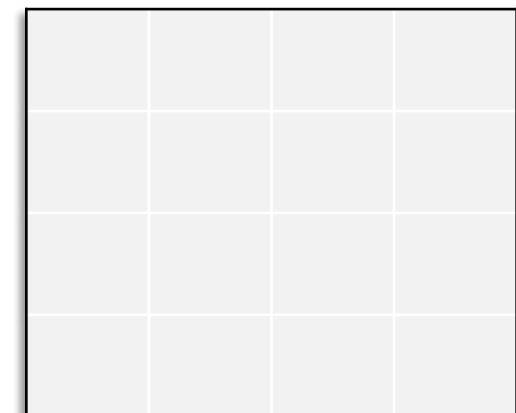
Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

Original Image

$$\begin{matrix} & \begin{matrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{matrix} \\ * & \end{matrix}$$

Convolved Image



KERNELS AND CONVOLUTION

Blur Kernel

$$\begin{matrix} .06 & .13 & .06 \\ .13 & .25 & .13 \\ .06 & .13 & .06 \end{matrix}$$

Original Image

$$\begin{matrix} * & \begin{matrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{matrix} & = & \begin{matrix} \square \end{matrix} \end{matrix}$$

The diagram illustrates the convolution process. A 3x3 'Blur Kernel' (blue box) is multiplied (*) by a 7x7 'Original Image' (blue box). The result is a 'Convolved Image' (green box). The kernel is applied to the original image in a sliding window fashion. The highlighted step shows the kernel being applied to the center of the original image, resulting in a value of 1.0 in the convolved image.

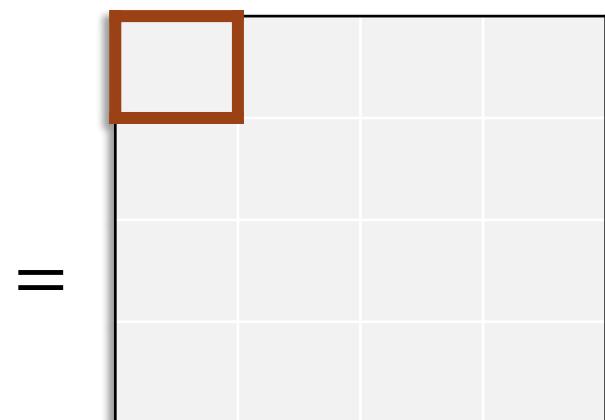
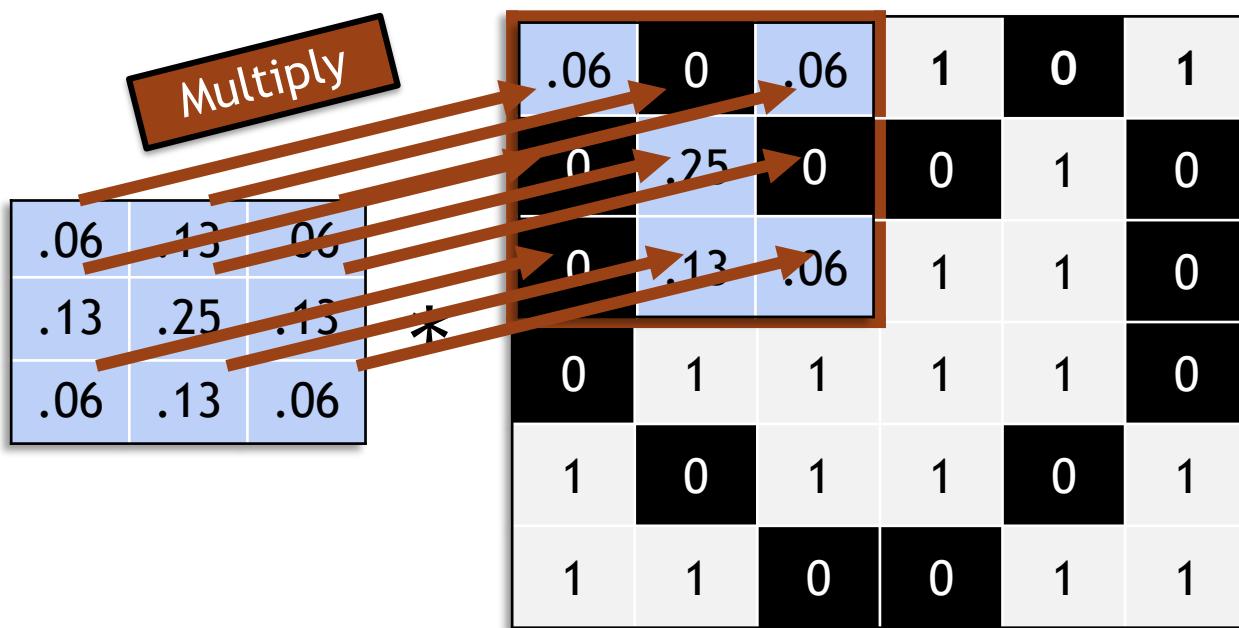
Convolved Image

KERNELS AND CONVOLUTION

Blur Kernel

Original Image

Convolved Image

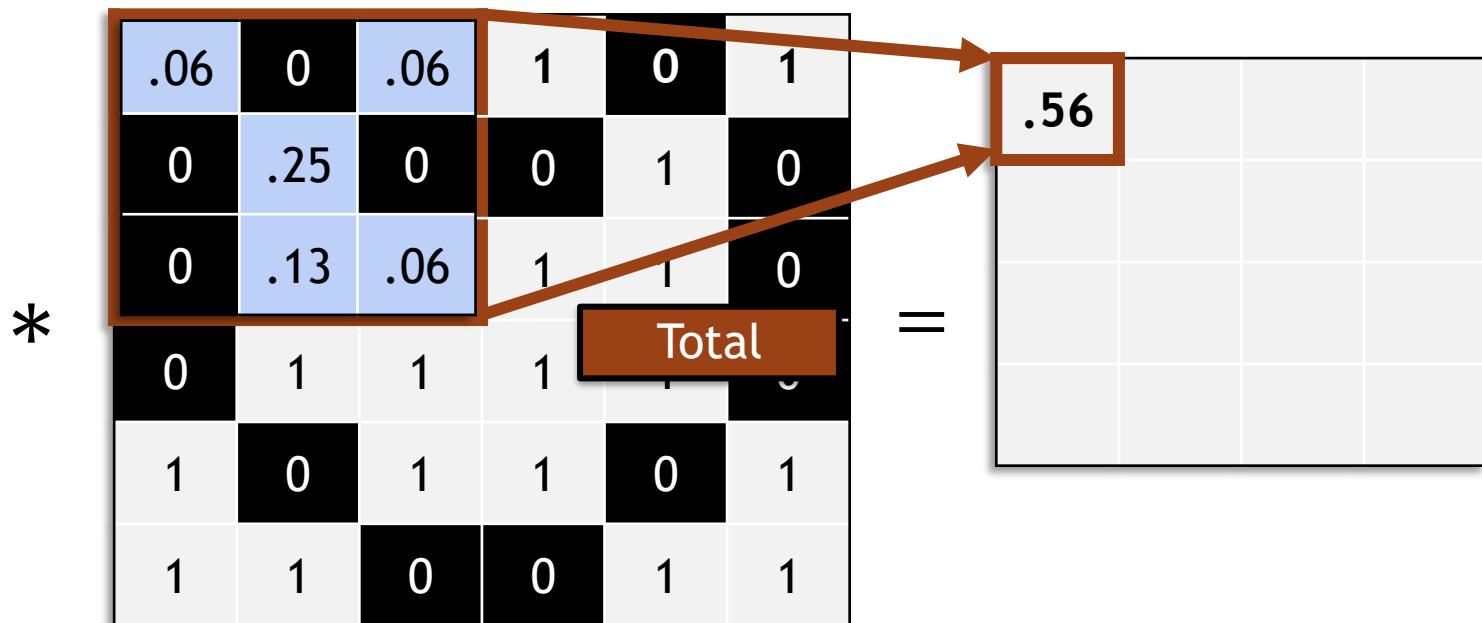


KERNELS AND CONVOLUTION

Blur Kernel

$$\begin{bmatrix} .06 & .13 & .06 \\ .13 & .25 & .13 \\ .06 & .13 & .06 \end{bmatrix}$$

Original Image



KERNELS AND CONVOLUTION

Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

Original Image

*

1	0	.13	.06	0	1
0	.13	0	0	1	0
0	.06	.13	.06	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

Convolved Image

=

.56	.57

KERNELS AND CONVOLUTION

Blur Kernel

.06	.13	.06
.13	.25	.13
.06	.13	.06

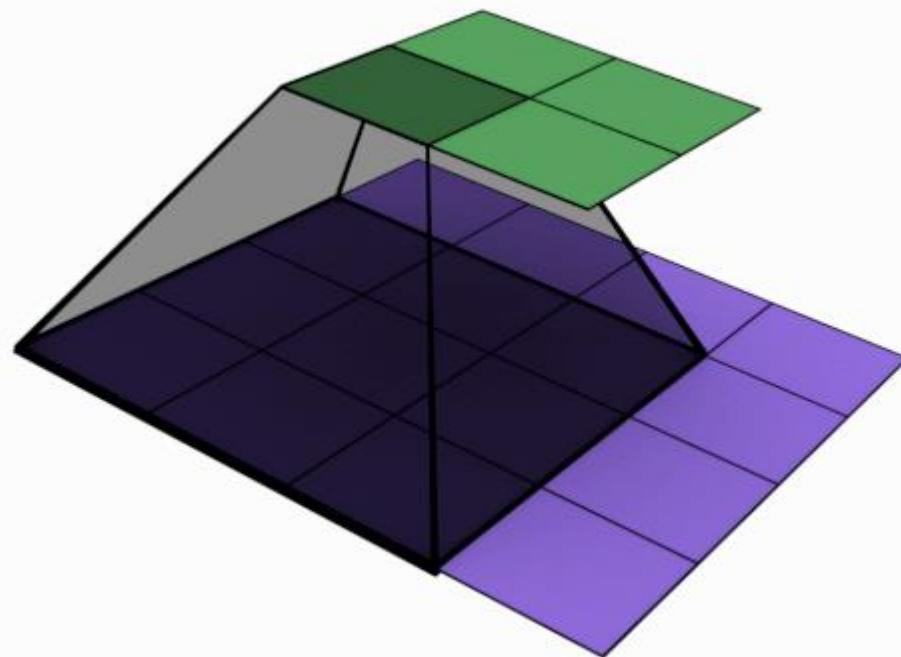
Original Image

$$\begin{matrix} & \begin{matrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{matrix} \\ * & \end{matrix}$$

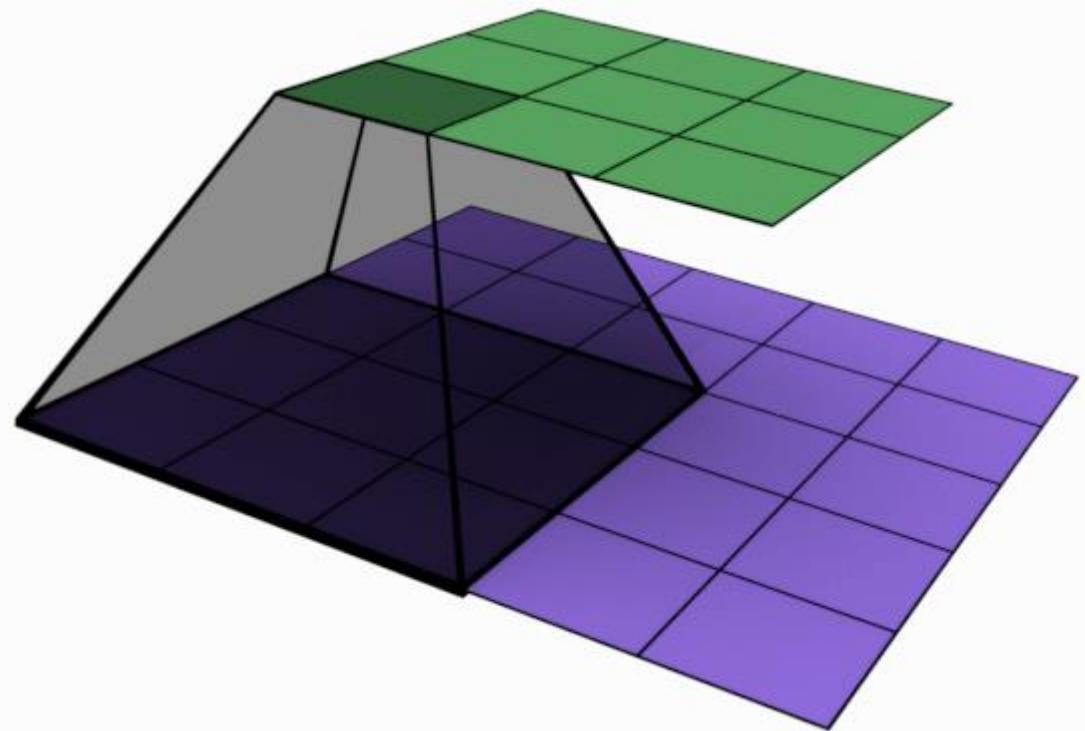
Convolved Image

.56	.57	.57	.56
.7	.82	.82	.7
.69	.95	.95	.69
.64	.69	.69	.64

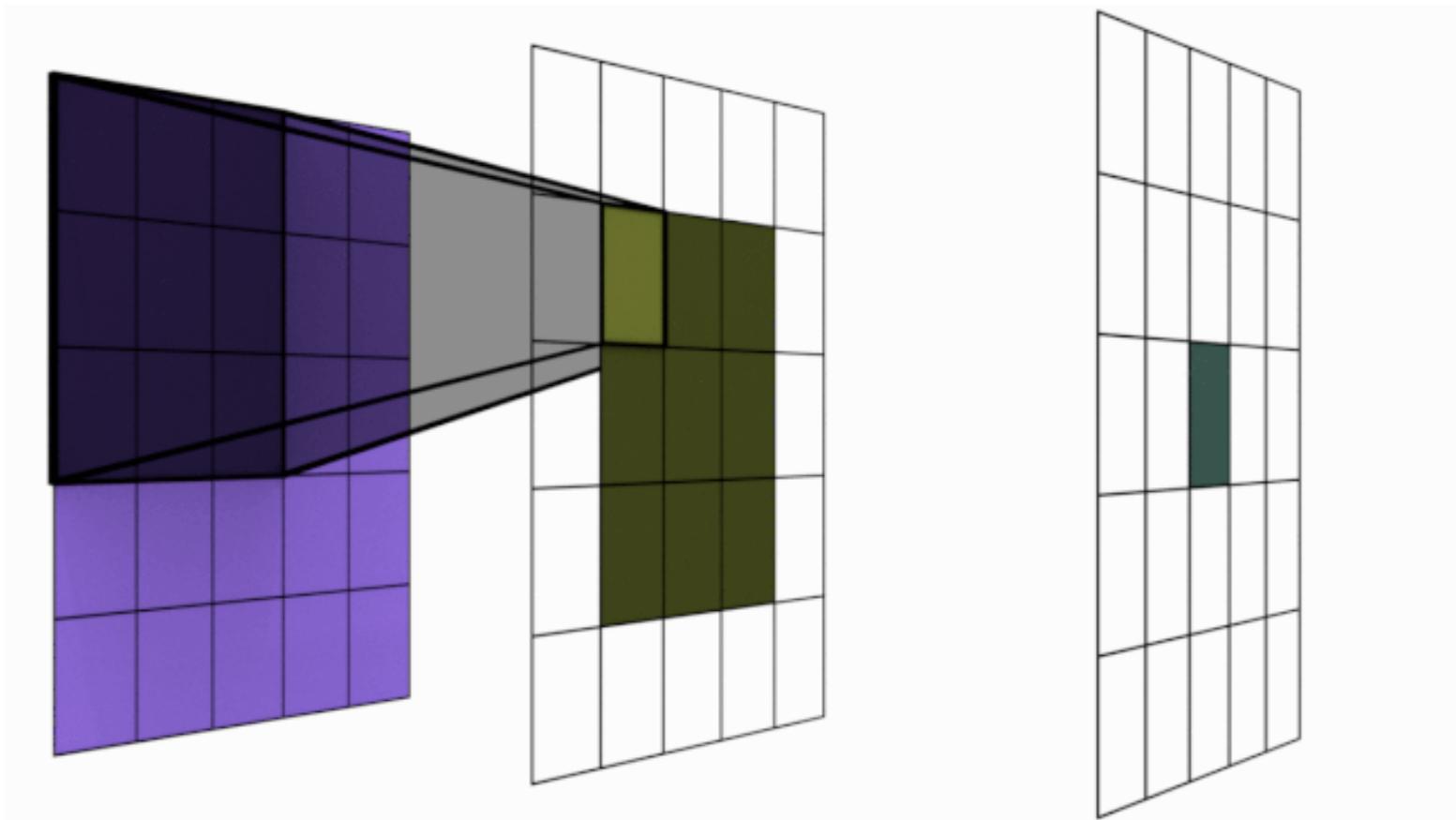
3X3 on a 4X4



3X3 on a 5X5



RECEPTIVE FIELD



How to obtain a 1X1
from 5X5 using
single convolution
operation?

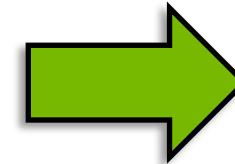
How many layers
would we need to
move from 400x400
image to 1x1?

STRIDE

how many cells should we move our window by when doing convolution

Stride 1

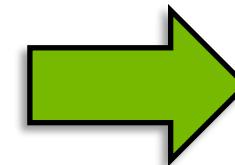
1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0



.56	.57	.57	.56
-----	-----	-----	-----

Stride 2

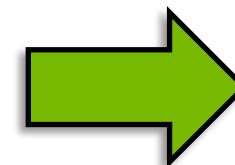
1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0



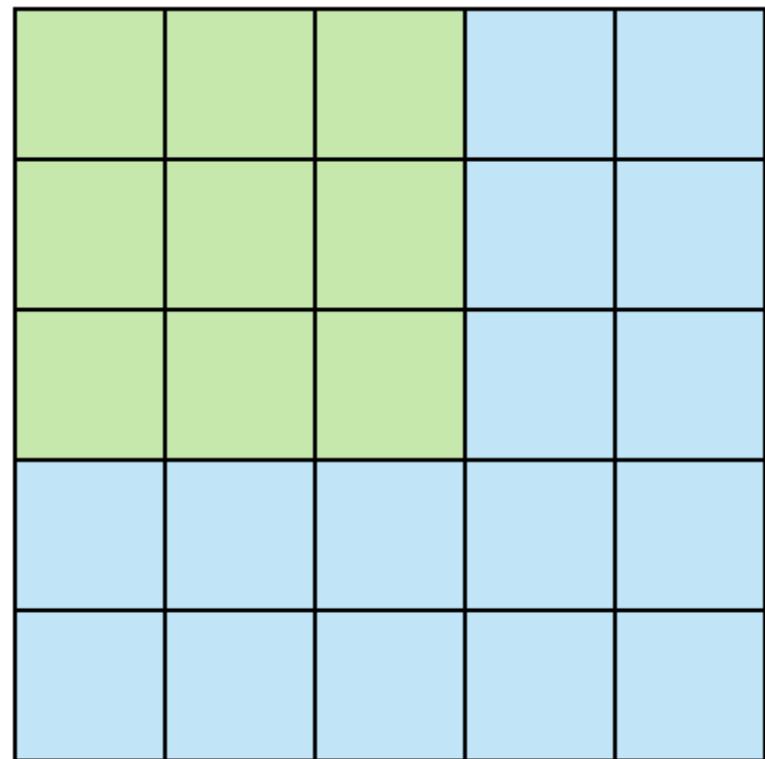
.56	.57
-----	-----

Stride 3

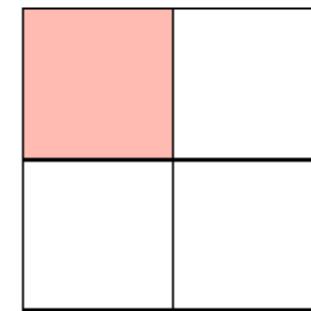
1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0



.56	.56
-----	-----



Stride 2



Feature Map

PADDING

Original Image

1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

Zero Padding

0	0	0	0	0	0	0	0
0	1	0	1	1	0	1	0
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
0	1	0	1	1	0	1	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0

PADDING

Original Image

1	0	1	1	0	1
0	1	0	0	1	0
0	1	1	1	1	0
0	1	1	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1

Same Padding

1	1	0	1	1	0	1	1
1	1	0	1	1	0	1	1
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	1	0	0
1	1	0	1	1	0	1	1
1	1	1	0	0	1	1	1
1	1	1	0	0	1	1	1

0 ₂	0 ₀	0 ₁	0	0	0	0	0
0 ₁	2 ₀	2 ₀	3	3	3	0	0
0 ₀	0 ₁	1 ₁	3	0	3	0	0
0	2	3	0	1	3	0	0
0	3	3	2	1	2	0	0
0	3	3	0	2	3	0	0
0	0	0	0	0	0	0	0

1	6	5
7	10	9
7	10	8

0	0	0	0	0	0	0
0	60	113	56	139	85	0
0	73	121	54	84	128	0
0	131	99	70	129	127	0
0	80	57	115	69	134	0
0	104	126	123	95	130	0
0	0	0	0	0	0	0

Kernel

0	-1	0
-1	5	-1
0	-1	0

114				

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

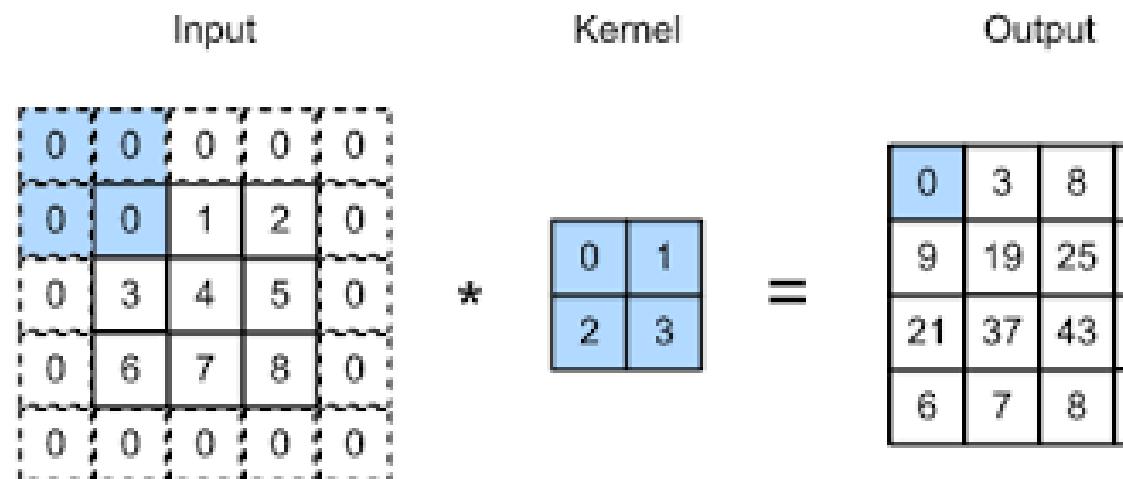
n_{in} : number of input features

n_{out} : number of output features

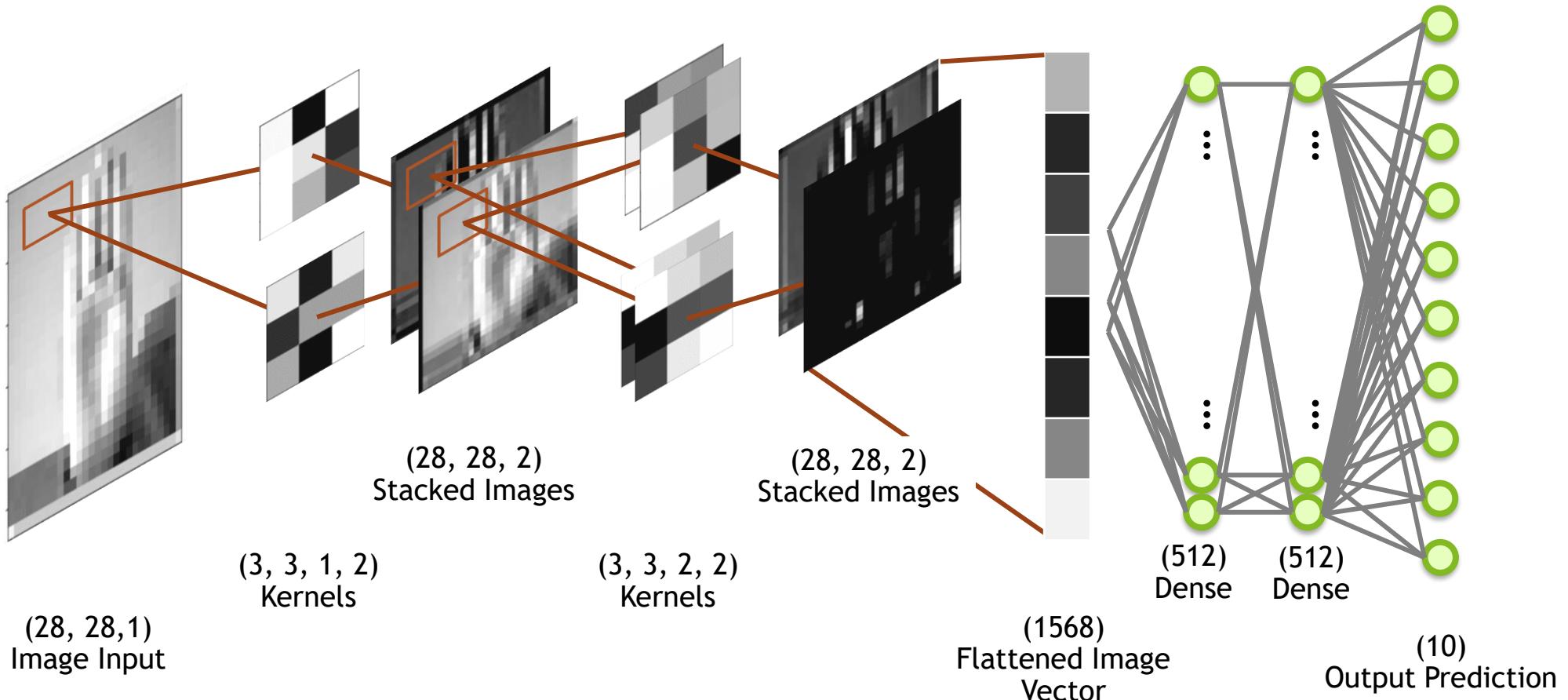
k : convolution kernel size

p : convolution padding size

s : convolution stride size

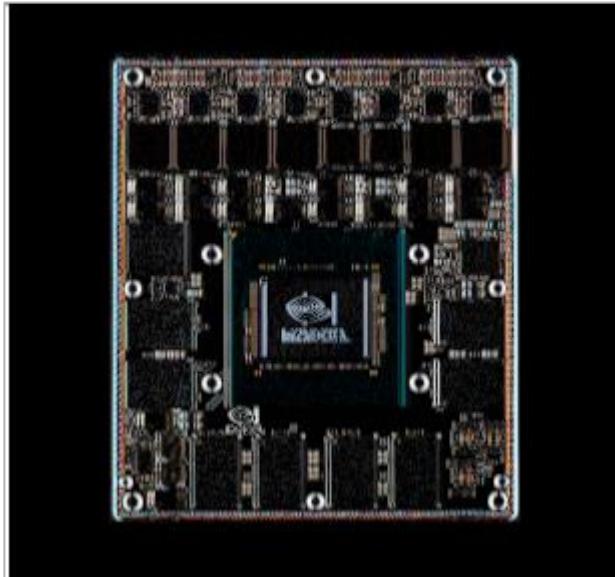


KERNELS AND NEURAL NETWORKS

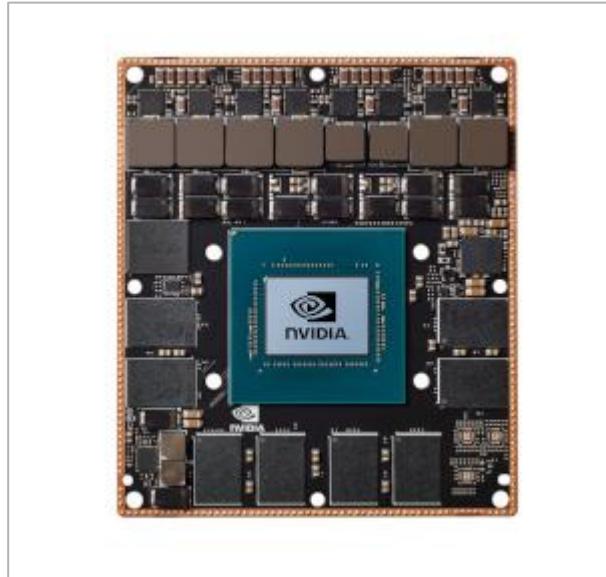


FINDING EDGES

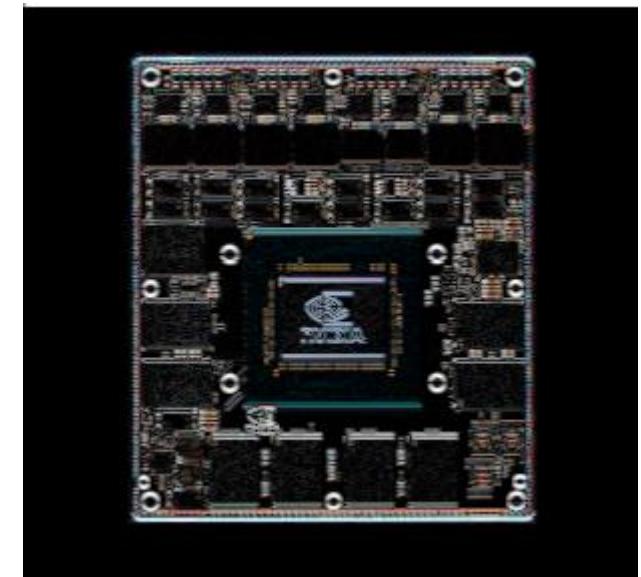
Vertical Edges



Original Image



Horizontal Edges

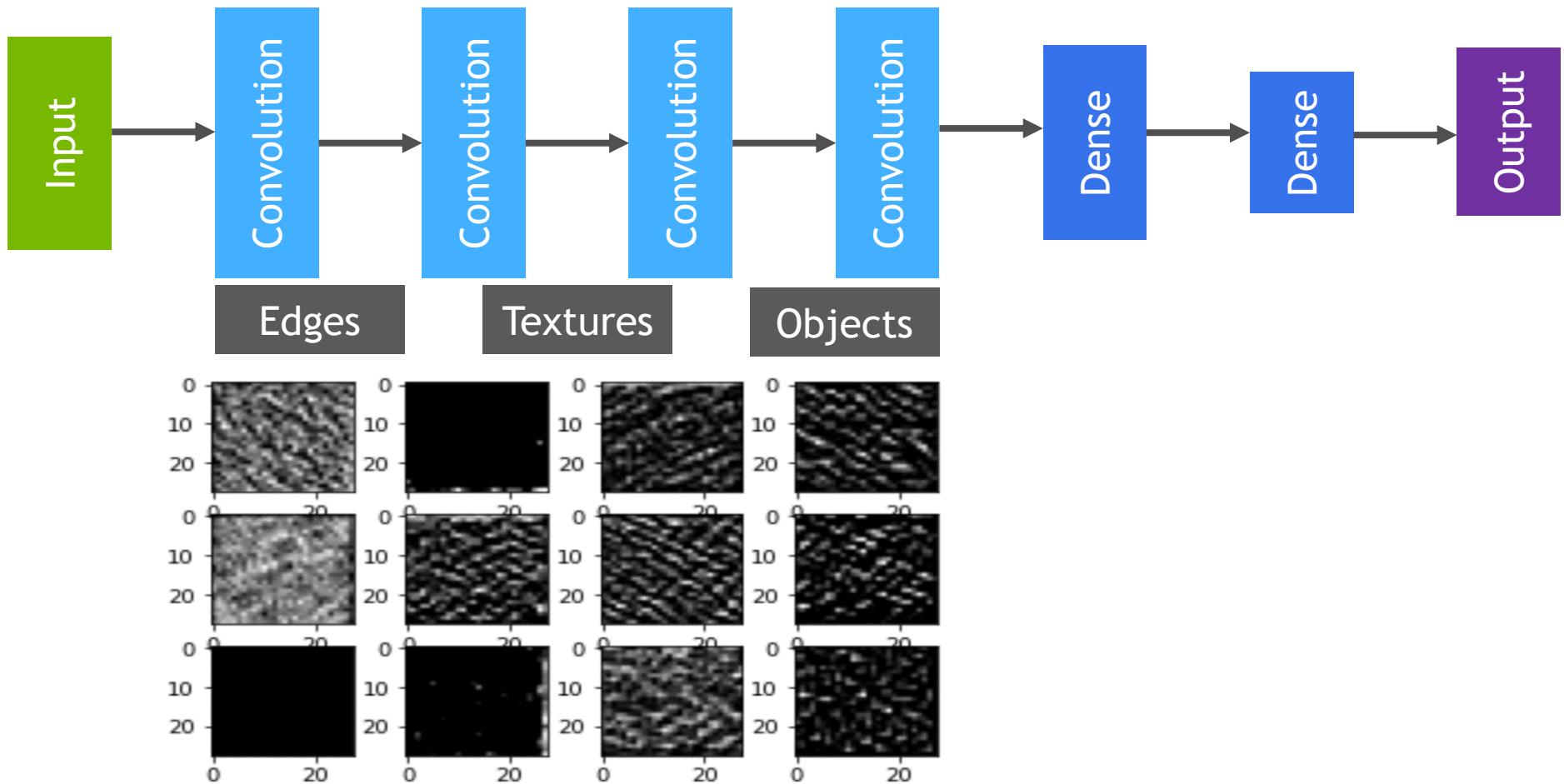


1	0	-1
2	0	-2
1	0	-1

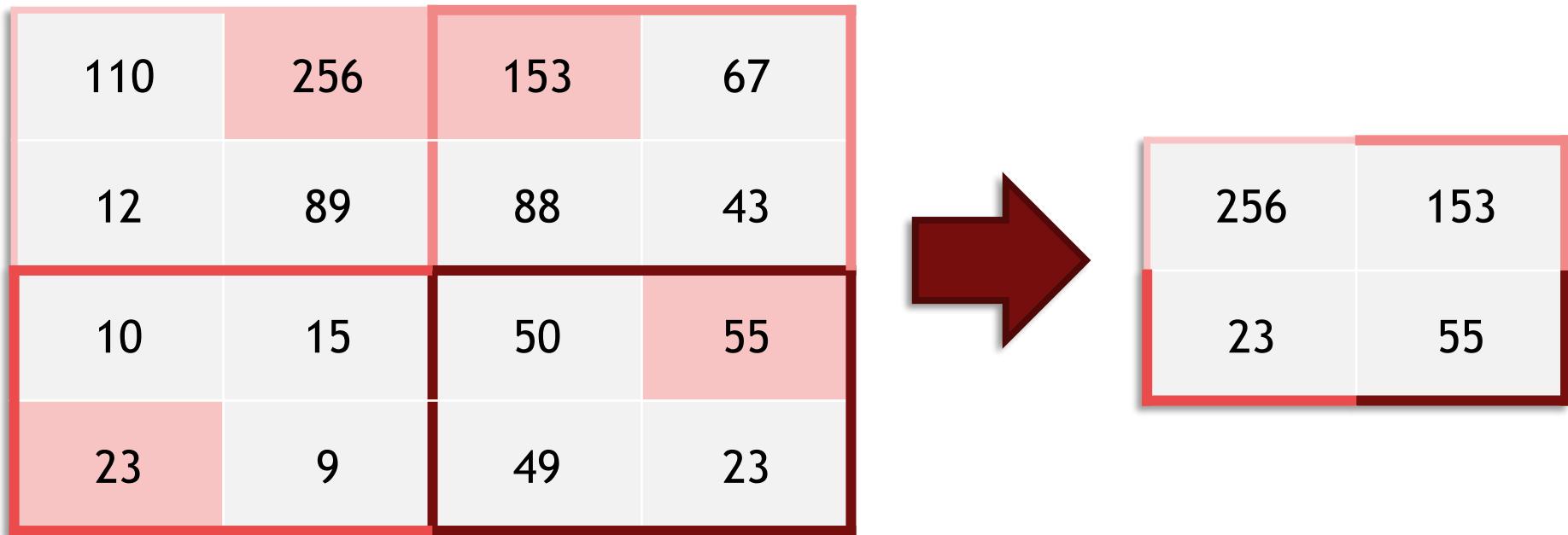
0	0	0
0	1	0
0	0	0

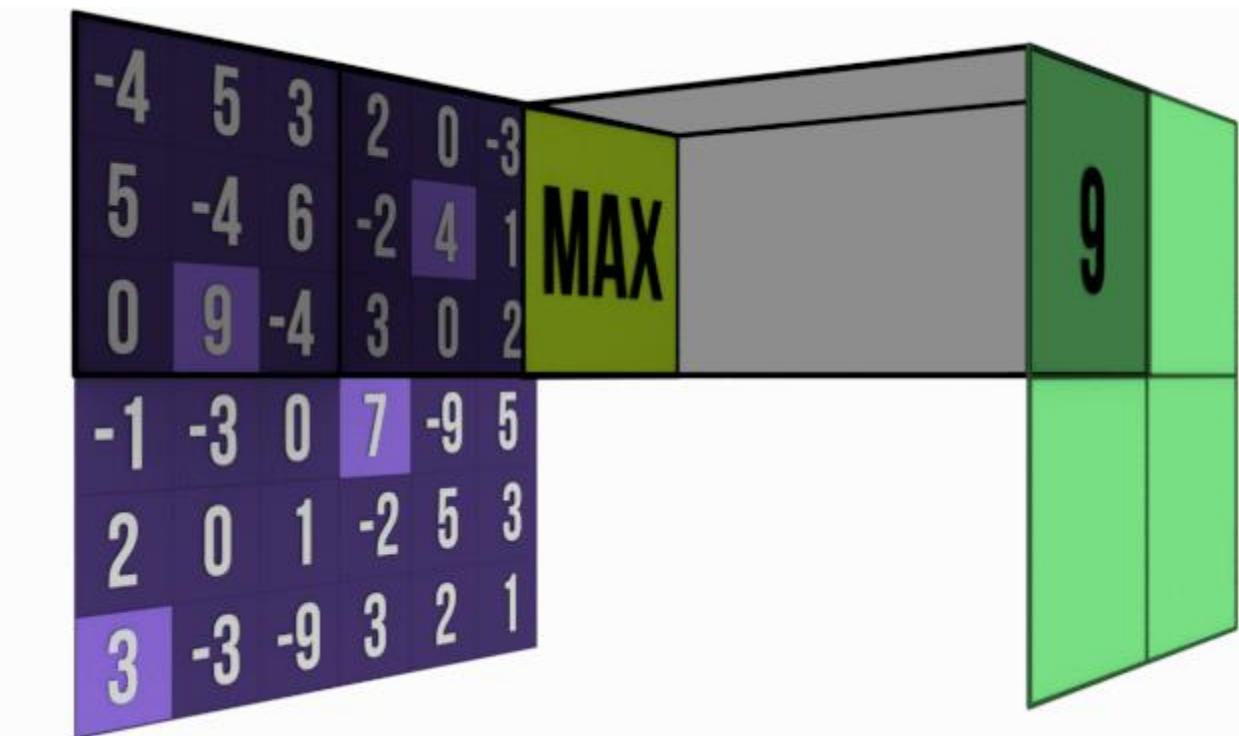
1	2	1
0	0	0
-1	-2	-1

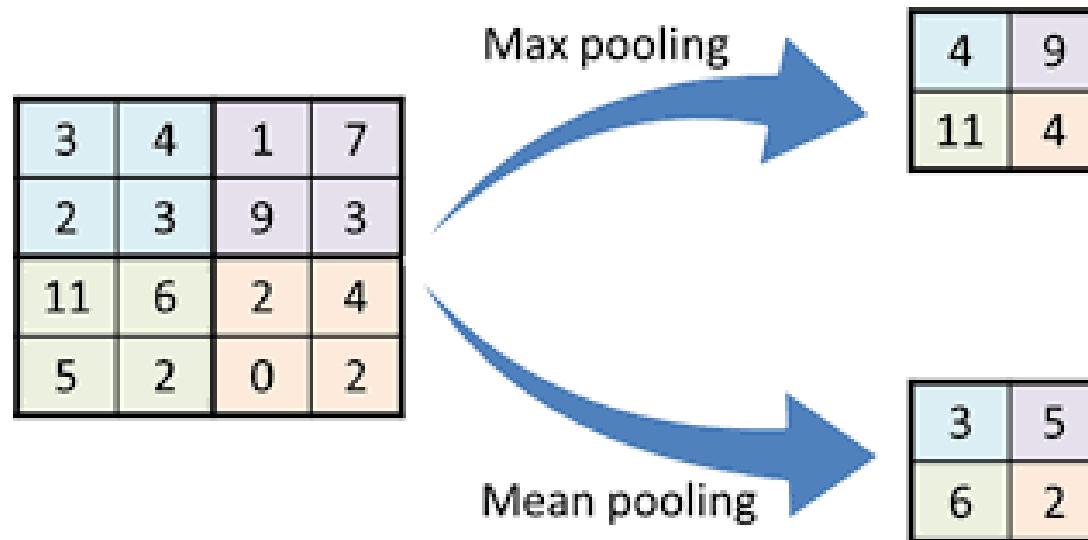
NEURAL NETWORK PERCEPTION



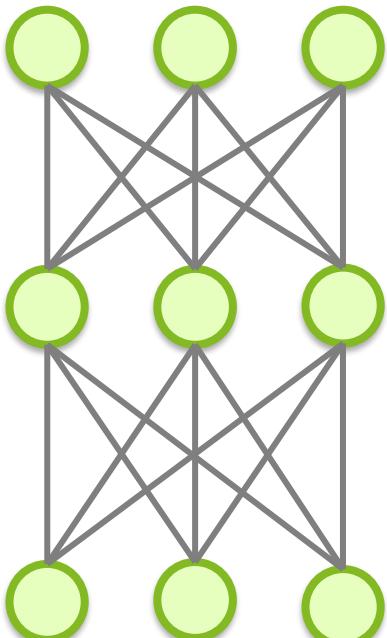
MAX POOLING



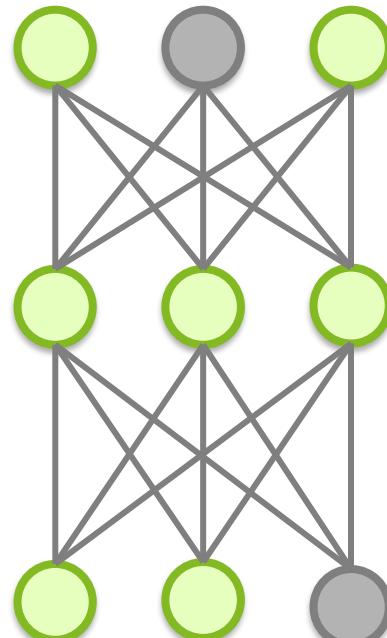




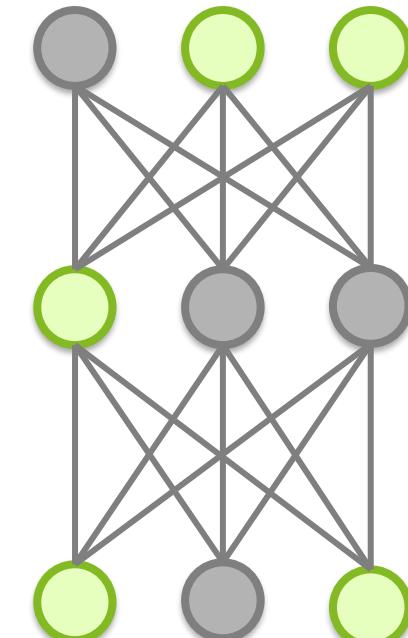
DROPOUT



rate = 0



rate = .2

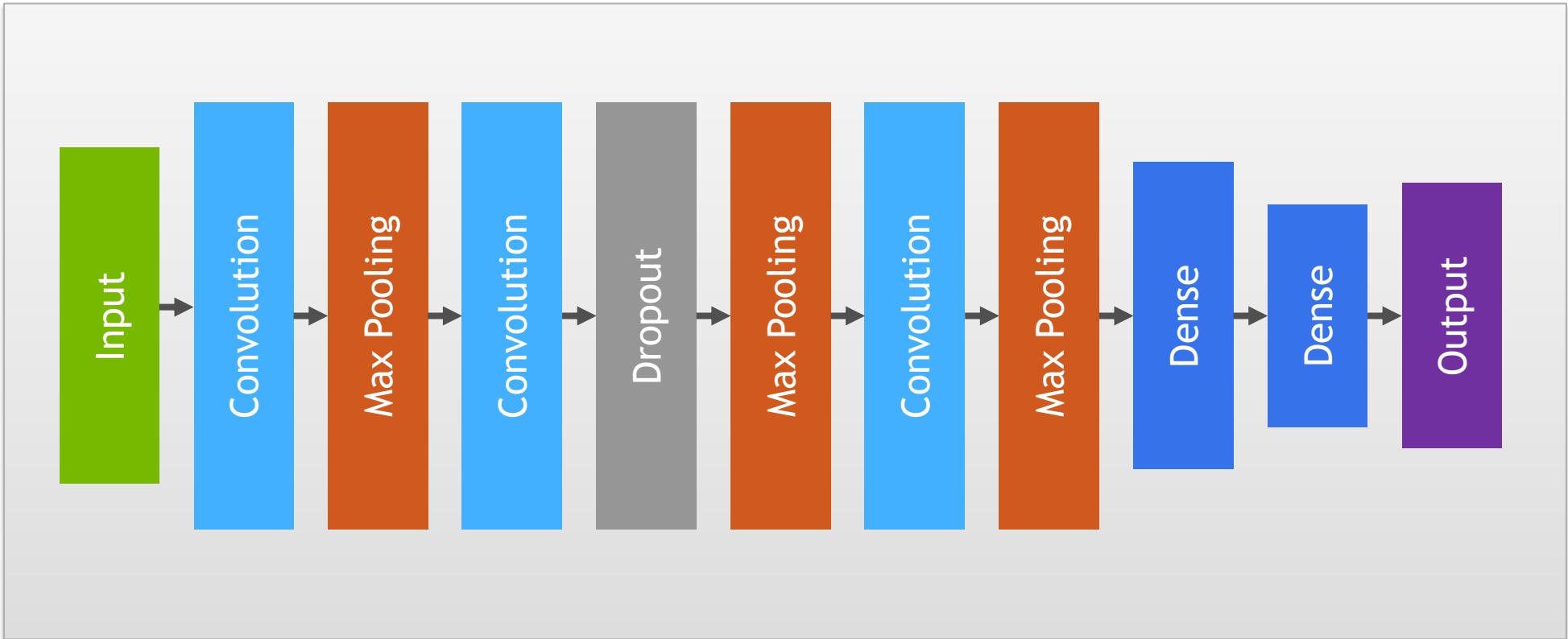


rate = .4

What if rate = 1?



WHOLE ARCHITECTURE



Back Propagation

- Back propagation algorithm learns the weights for a multilayer network, given a network with fixed set of inputs and interconnections.

- Computational Graph

$$\text{Equation: } (x+y) \cdot z$$

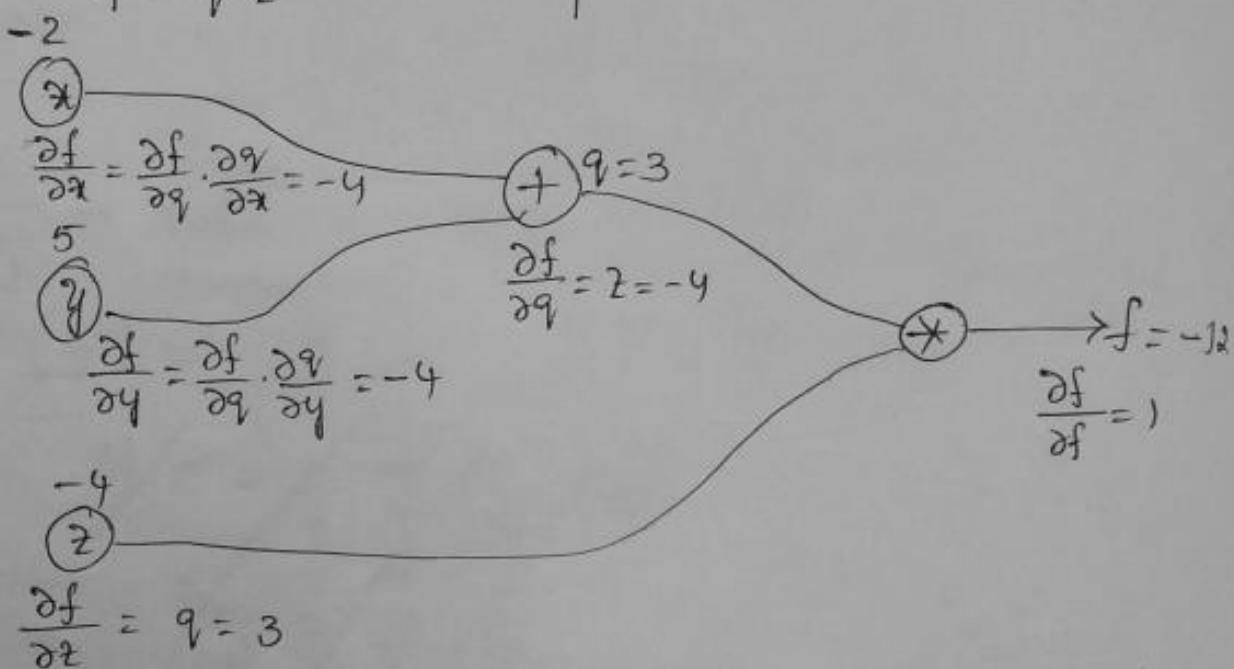
$$q: x+y$$

$$f: q \cdot z$$

$$x = -2$$

$$y = 5$$

$$z = -4$$



$\frac{\partial f}{\partial z} = 3 \Rightarrow$ a unit change in z causes an increase of f value by 3.

- Q) for the equation $f = (a+b) \cdot (c+d)$, find the change in f w.r.t to i/p variables using C.G.

- Notations used in BP

$w_{j,k}^l$ - weight from node k of layer $(l-1)$ to node j of layer l

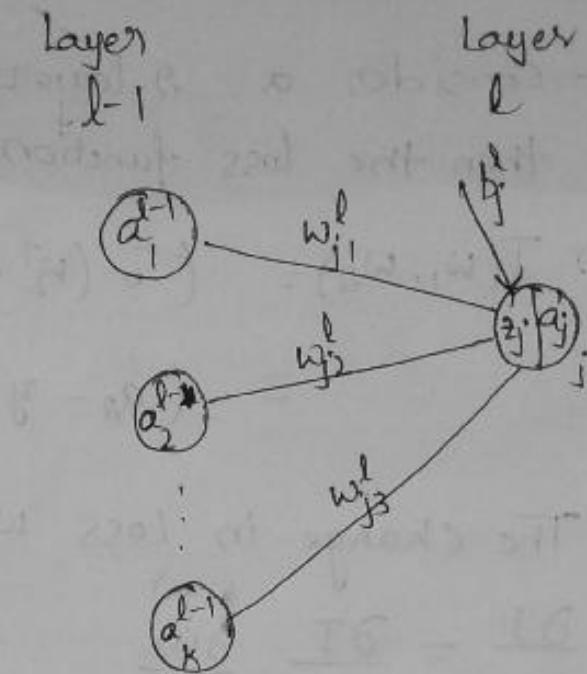
b_j^l - bias of node j of layer l

z_j^l - weighted i/p emitted from node j

a_j^l - activated value to the weighted i/p z_j^l

$$z_j^l = \sum_k w_{jk}^l \cdot a_k^{l-1} + b_j^l$$

$$a_j^l = \sigma(z_j^l)$$



- Loss function

The loss function that need to be minimized is

$$J(w_1, w_2, \dots, w_L) = \sum_{i=1}^N (\sigma(w_L^T \dots \sigma(w_2^T \cdot \sigma(w_1^T \cdot x_i))) - y_i)^2$$

To optimize, we need gradient descent

for eg: for \$w_1\$

$$w_1^{t+1} = w_1^t - \alpha \cdot \nabla J(w_1^t)$$

we need to do this for all \$w_1, w_2, \dots, w_L\$.

- The weighted sum and activation values for a L-layer network are

$$z_1 = w_1^T \cdot x$$

$$a_1 = \sigma(z_1)$$

$$z_2 = w_2^T \cdot a_1$$

$$a_2 = \sigma(z_2)$$

$$z_L = w_L^T \cdot a_{L-1}$$

$$a_L = \sigma(z_L)$$

— Consider a 2-layer network.
then the loss function is

$$J(w_1, w_2) = (\sigma(w_2^T \cdot \sigma(w_1^T x)) - y)^2$$

$$= (a_2 - y)^2$$

$$a_2 = \sigma(w_2^T \cdot \sigma(w_1^T x))$$

The change in Loss w.r.t w_2 is

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_2}$$

$$= \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}$$

The change in Loss w.r.t w_1 is

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$$

$$= \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1}$$

$$= \frac{\partial J}{\partial a_2} \cdot \frac{\partial a_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1} \quad \text{(chain rule)}$$

— As shown above the loss is $(a_2 - y)^2$
For a neural network the loss function takes the
form of

$$C = \sum_j (a_j^L - y_j)^2$$

L - output layer
j - no. of neurons in
output layer

$$(a_1 - y_1)^2$$

$$(a_2 - y_2)^2$$

$$(a_3 - y_3)^2$$

$$C = \sum$$

$$\text{overall Loss } C = \sum_j (q_j^L - y_j)^2$$

The Error is defined as

$$\delta_j^L = \frac{\partial C}{\partial z_j^L}$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L}$$

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)$$

Fundamental equations for Backpropagation

1. Error in the output layer
2. An equation for error δ^L in terms of error in next layer δ^{L+1}
3. Rate of change w.r.t bias
4. Rate of change w.r.t weight

Equation - 1

Error in the output layer

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \sigma'(z_j^L)$$

↑ ↗

Rate of change Rate of change
w.r.t a_j^L w.r.t z_j^L

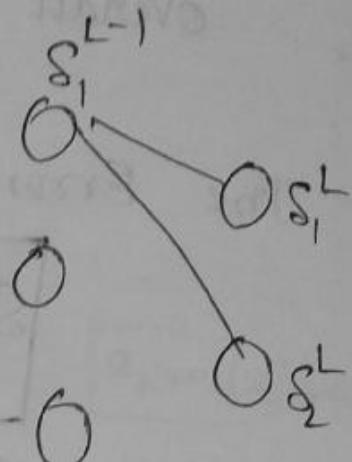
Matrix vector form:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

Equation 2:

An equation for δ^l in terms of the error in the next layer δ^{l+1}

$$\boxed{\delta^l = ((w^{l+1})^T \cdot \delta^{l+1}) \odot \sigma^{-1}(z^l)}$$



The equations 1 and 2 can help you determine error at any layer.

Equation 3:

An equation for the rate of change of cost w.r.t any bias in the network

$$\boxed{\frac{\partial C}{\partial b_j} = \delta_j^l}$$

Equation 4:

An equation for the rate of change of cost w.r.t any weight in the network

$$\boxed{\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l}$$

Summary of the Backpropagation algorithm

① Input \mathbf{z} : Set the corresponding activation a' for the input layer

② Feed forward: for each $l = 2, 3, \dots, L$ compute
 $z^l = w^l \cdot a^{l-1} + b^l$ and
 $a^l = \sigma(z^l)$

③ Output the error δ^L : compute the vector
 $\delta^L = \nabla_a C \odot \sigma'(z^L)$

④ Backpropagate the error:

for each $l = L-1, L-2, \dots, 2$

compute $\delta^l = ((w^{l+1})^T \cdot \delta^{l+1}) \odot \sigma'(z^l)$

⑤ The gradient of the cost function is given by

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \text{ and}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$