Data Mining and Predictive Modelling – Module 3

Dr. Dinesh Kumar
Associate Professor
Bennett University (Times of India Group)
Plot Nos 8, 11, TechZone 2, Greater Noida, Uttar Pradesh 201310

Course Brief

CSET228 - Data Mining and Predictive Modelling

Course Type - Specialized Core – II L-T-P Format 3-0-2 Credits - 4

COURSE SUMMARY

This course exposes multiple techniques of understanding and analysing the data from a mathematical point of view. In addition, they will also use multiple predictive models to analyse the future trend. This will be done in a purely statistical manner.

COURSE-SPECIFIC LEARNING OUTCOMES (CO)

CO1: To articulate data preparation for data mining and analyzing based on pre-processing techniques.

CO2: To examine predictive analysis in various use cases.

CO3: To make use of exploratory data analysis to gain insights and prepare data for predictive modelling.

Tentative Evaluation Components

Components	Percentage
Mid-Term	20
End-Term	40
Course Certification	10
Lab Continuous Evaluation	10
End Term Lab Examination	20

Syllabus

Module 1 (8 hours)

Purpose of Data mining, Procedures of Data Mining, Functionality of Data Mining, Knowledge data discovery process, Data, and attribute type, Properties of data, Discrete and continuous attributes, Dataset types, Data quality measurement, Noise Analysis and its importance, Techniques of Data pre-processing, Aggregation, Sampling, Curse of dimensionality, Dimensionality reduction, Feature selection and generation, Discretization and vectorization, Binarization, Attribute transformation correlation, Association rule mining, Apriori algorithm, Rule generation, Pattern Mining in: Multilevel, Multidimensional Space Pattern Mining.

Module 2 (7 hours)

Rule-based reasoning, Memory-based reasoning, measuring data similarity, Similarity Metrics: Distance-based measure, Information based measures, Set similarity measure, Jaccard Index, Sorenson Dice Coefficient, Model Selection Problem, Error Analysis, Case study, Startups in DataAnalysis.

Module 3 (14 hours)

Supervised Learning; Classification: K-NN, Performance Matrix, Linear Regression, Logistic Regression, Decision Tree, SVM, ANN **Unsupervised Learning;** Clustering: K-means, Outlier analysis in classification and clustering, **Exploratory data analysis**, Data summarization and visualization, Dataset exploration, Data Exploration Tools,Interactive Data Exploration, Predictive models, Design Principles, Parametric Models, Non-Parametric Models, one way and two way ANOVA, Regression Analysis

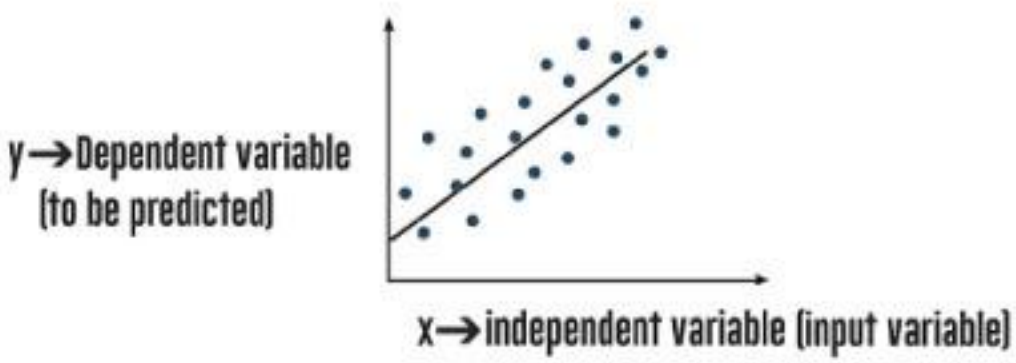
Module 4 (8 hours)

Linear discriminant analysis, Fisher discriminant analysis, Time series Model: ARMA, ARIMA, ARFIMA, Factor Analysis, Recommendation System and Collaborative Filtering, Multidimensional Scaling, Mining Textual Data, Temporal mining, Spatial mining, Visual and audio data mining, Social Impacts of data mining.

What is a Regression ?

- Linear Regression is one of the machine learning algorithms where the result is predicted by the use of known parameters which are correlated with the output.
- It is used to predict values within a continuous range rather than trying to classify them into categories.
- The known parameters are used to make a continuous and constant slope which is used to predict the unknown or the result.
- Majority of the machine learning algorithms fall under the supervised learning category. It is the process where an algorithm is used to predict a result based on the previously entered values and the results generated from them. Suppose we have an input variable ‘x’ and an output variable ‘y’ where y is a function of x (y=f{x}). Supervised learning reads the value of entered variable ‘x’ and the resulting variable ‘y’ so that it can use those results to later predict a highly accurate output data of ‘y’ from the entered value of ‘x’. A regression problem is when the resulting variable contains a real or a continuous value. It tries to draw the line of best fit from the data gathered from a number of points.

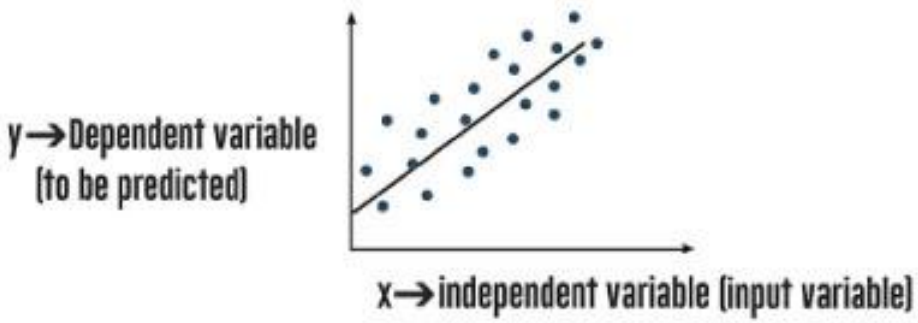
Regression Problem



For example, ***which of these is a regression problem?***

- How much gas will I spend if I drive for 100 miles?
- What is the nationality of a person?
- What is the age of a person?
- Which is the closest planet to the Sun?

Regression Problem



For example, ***which of these is a regression problem?***

- How much gas will I spend if I drive for 100 miles?
- What is the nationality of a person?
- What is the age of a person?
- Which is the closest planet to the Sun?

Ans: Predicting the amount of gas to be spent and the age of a person are regression problems. Predicting nationality is categorical and the closest planet to the Sun is discrete.

What is a Linear Regression ?

- Let's say we have a dataset which contains information about the relationship between 'number of hours studied' and 'marks obtained'. A number of students have been observed and their hours of study along with their grades are recorded. This will be our training data. Our goal is to design a model that can predict the marks if number of hours studied is provided. **Using the training data, a regression line is obtained which will give minimum error. This linear equation is then used to apply for a new data.** That is, if we give the number of hours studied by a student as an input, our model should be able to predict their mark with minimum error.

Hypothesis of Linear Regression

- The linear regression model can be represented by the following equation:

$$Y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

where,

Y is the predicted value

θ_0 is the bias term.

$\theta_1,...,\theta n$ are the model parameters

$x_1, x_2,...,xn$ are the feature values.

The above hypothesis can also be represented by

$$Y = \theta^T x$$

Where, θ is the model's parameter vector including the bias term θ_0; x is the feature vector with $x_0 = 1$

$$Y\ (pred) = b_0 + b_1 * x$$

The values b_0 and b_1 must be chosen so that the error is minimum. If sum of squared error is taken as a metric to evaluate the model, then the goal is to obtain a line that best reduces the error.

$$\text{Error} = \sum_{i=1}^{n} (actual\_output - predicted\_output) ** 2$$

If we don't square the error, then the positive and negative points will cancel each other out.

Hypothesis of Linear Regression

- It is a **function** that measures the performance of a model for any given data. **Cost Function** quantifies the error between predicted values and expected values and presents it in the form of a single real number

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

Parameters: θ_0, θ_1

Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Linear Regression: Ordinary Least Squares (OLS)

OLS is a method for estimating the unknown parameters in a linear regression model. The goal is to find the line (or hyperplane in higher dimensions) that best fits the data by minimizing the sum of the squared differences between the observed values and the values predicted by the model.

A linear regression model expresses the relationship between a dependent variable y and one or more independent variables X. The model can be written as:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i$$

where:

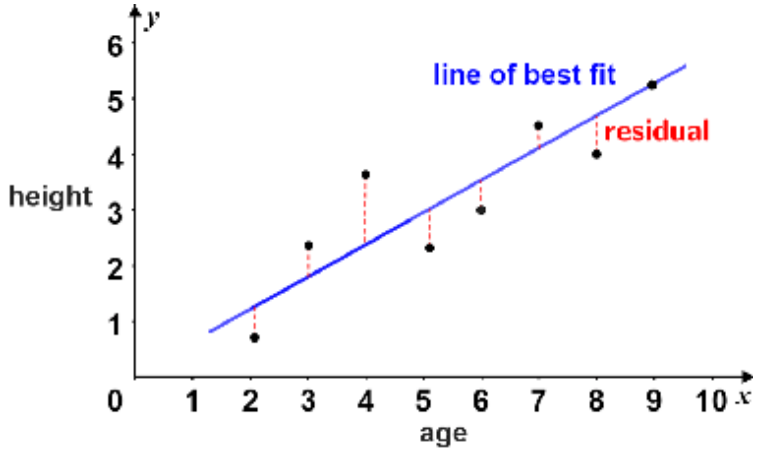
- y_i is the dependent variable.
- x_ij are the independent variables.
- β_j are the coefficients to be estimated.
- ε_i is the error term.

Linear Regression: Ordinary Least Squares (OLS)

Step-by-Step Estimation of Coefficients Using OLS

In matrix form, the model can be written as:

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$



where:

- y is an n×1 vector of the dependent variable.
- X is an n×(p+1) matrix of the independent variables.
- β is a (p+1)×1 vector of the coefficients.
- ε is an n×1 vector of the errors.

Linear Regression: Ordinary Least Squares (OLS)

Step-by-Step Estimation of Coefficients Using OLS

Step 1: Define the Least Squares Cost Function

The objective is to minimize the **Sum of Squared Errors (SSE)**:

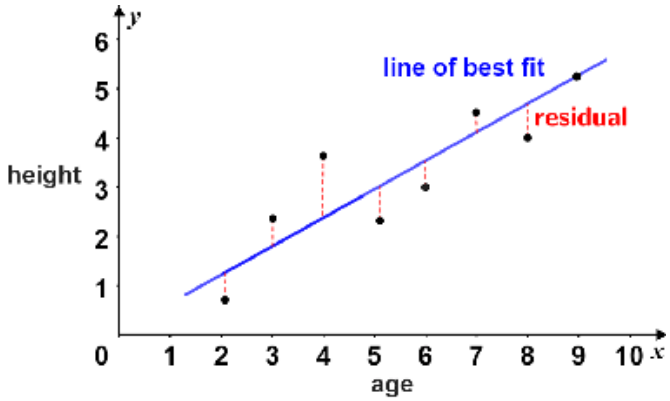
$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Expanding:

$$SSE = (Y - X\beta)^T (Y - X\beta)$$

To find the best coefficients (β), we take the derivative of SSE with respect to β and set it to zero:

$$\frac{\partial SSE}{\partial \beta} = -2X^T(Y - X\beta) = 0$$



Linear Regression: Ordinary Least Squares (OLS)

Step-by-Step Estimation of Coefficients Using OLS

Step 2: Solve for β

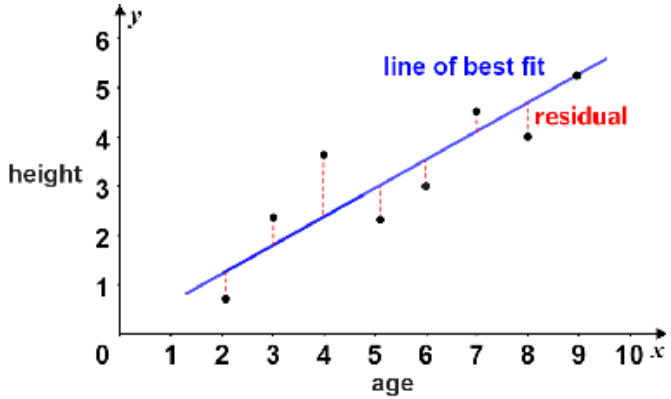
Rearrange the equation:

$$X^T X\beta = X^T Y$$

Solving for β:

$$\beta = (X^T X)^{-1} X^T Y$$

This equation provides the **closed-form solution** to compute the coefficients in **one step**, making it much faster than iterative methods like gradient descent.



Linear Regression: Ordinary Least Squares (OLS)

Step-by-Step Estimation of Coefficients Using OLS

x_1	x_2	y
1	2	3
2	3	4
3	5	6

Step 1: Construct the Matrices

We include a **bias column** (all ones) in X:

$$X = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 3 \\ 1 & 3 & 5 \end{bmatrix}$$

Target values:

$$Y = \begin{bmatrix} 3 \\ 4 \\ 6 \end{bmatrix}$$

Step 2: Compute $X^T X$ and $X^T Y$

$$X^T X = \begin{bmatrix} 3 & 6 & 10 \\ 6 & 14 & 23 \\ 10 & 23 & 38 \end{bmatrix}$$

$$X^T Y = \begin{bmatrix} 13 \\ 29 \\ 48 \end{bmatrix}$$

Step 3: Compute $(X^T X)^{-1}$

By inverting $X^T X$, we get:

$$(X^T X)^{-1} = \begin{bmatrix} 1.88 & -0.88 & 0.25 \\ -0.88 & 0.55 & -0.25 \\ 0.25 & -0.25 & 0.25 \end{bmatrix}$$

Step 4: Compute β

$$\beta = (X^T X)^{-1} X^T Y$$

Multiplying the matrices gives the estimated coefficients:

$$\beta = \begin{bmatrix} 1.00 \\ 0.50 \\ 1.00 \end{bmatrix}$$

Thus, the final equation for prediction is:

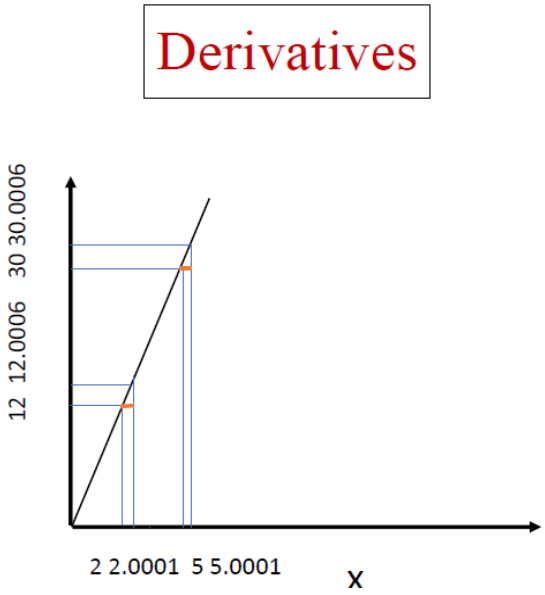
$$\hat{y} = 1.00 + 0.50x_1 + 1.00x_2$$

In linear regression, the bias column (also called the intercept term) is added to account for the constant term in the linear equation. This ensures that the model can fit data properly when the independent variables (xi) are zero.

What is Gradient Descent?

- Let’s say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?
- Take a moment to think about this before you read on.
- The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Linear Regression (Supervised Learning)



f(x)=6x

$\frac{df(x)}{dx} = 6$

x=2 f(x)=12

x=2.0001 f(x) = 12.0006

Slope at x=2 is .0006/.0001= 6

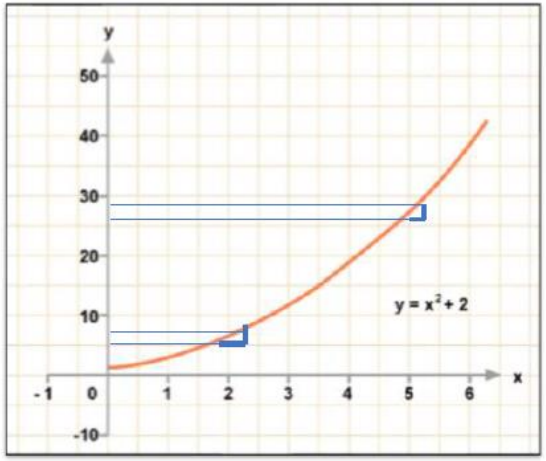
x=5 f(x) =30

x=5.0001 f(x) = 30.0006

Slope at x =5 is .0006/.0001= 6

Linear Regression (Supervised Learning)

Another Example



$y=f(x)=x^2+2$

$\frac{df(x)}{dx} = 2x$

x=2 f(x)=6

x=2.0001 f(x)=6.00040001

Slope at x=2 is .0004/.0001= 4

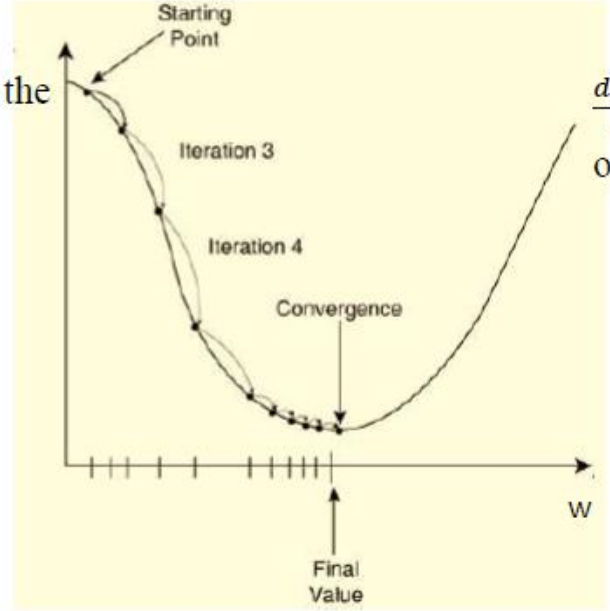
x=5 f(x) =27

x=5.0001 f(x) = 27.00100001

Slope at a =5 is .0010/.0001= 10

Gradient Descent Single Dimension

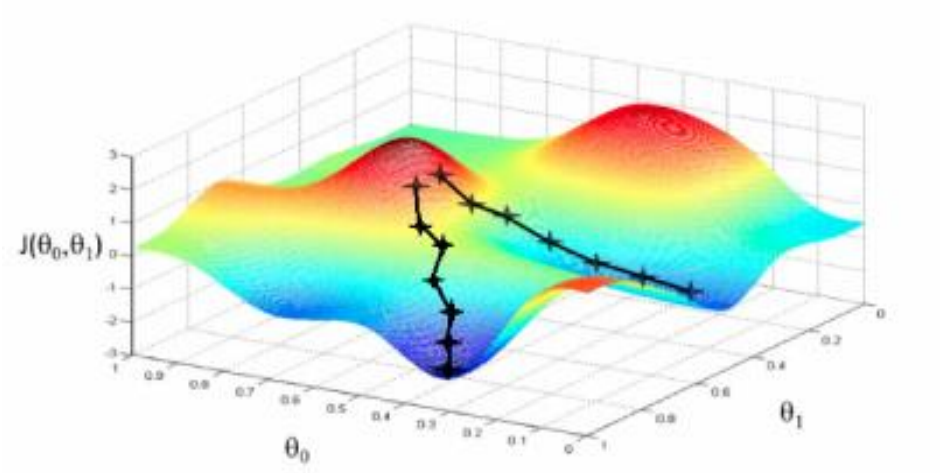
$\frac{dj(w)}{dw}$ is less than the optimum value



$\frac{dj(w)}{dw}$ is more than the optimum value

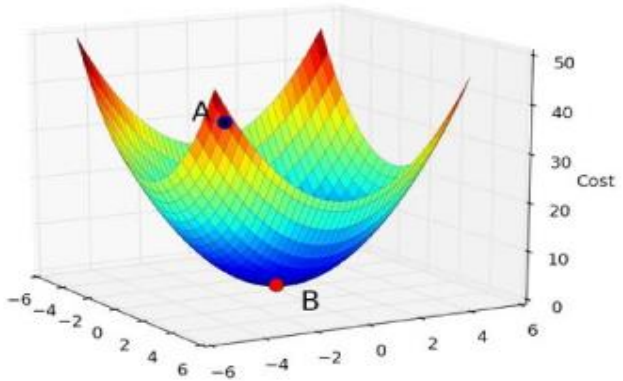
What is Gradient Descent?

- Gradient descent is an optimization algorithm which is mainly used to find the minimum of a function. In machine learning, gradient descent is used to update parameters in a model. Parameters can vary according to the algorithms, such as *coefficients* in Linear Regression and weights in Neural Networks.



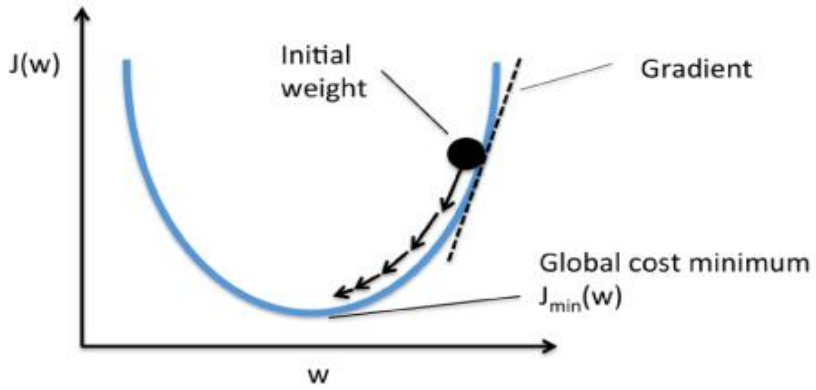
What is Gradient Descent?

- Similarly, let us consider another analogy. Suppose you have a ball and you place it on an inclined plane (at position A). As per laws, it will start rolling until it travels to a gentle plane where it will be stationary (at position B as shown in the figure below).



What is Gradient Descent?

This is exactly what happens in gradient descent. The inclined and/or irregular is the cost function when it is plotted and the role of gradient descent is to provide direction and the velocity (learning rate) of the movement in order to attain the minima of the function i.e where the cost is minimum.



How does Gradient Descent work?

- The primary goal of machine learning algorithms is always to build a model, which is basically a hypothesis which can be used to find an estimation for Y based on X.
- Let us consider an example of a model based on certain housing data which comprises of the sale price of the house, the size of the house etc. Suppose we want to predict the pricing of the house based on its size. It is clearly a regression problem where given some inputs, we would like to predict a continuous output.

The hypothesis is usually presented as

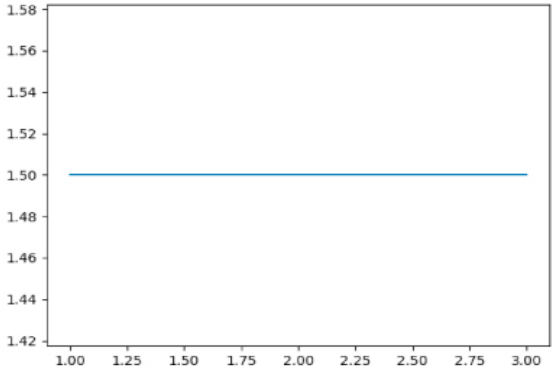
$$\theta_0 = 1.5$$
$$\theta_1 = 0$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

where the theta values are the *parameters*.

This yields h(x) = 1.5 + 0x. 0x means no slope, and y will always be the constant 1.5.

How does Gradient Descent work?

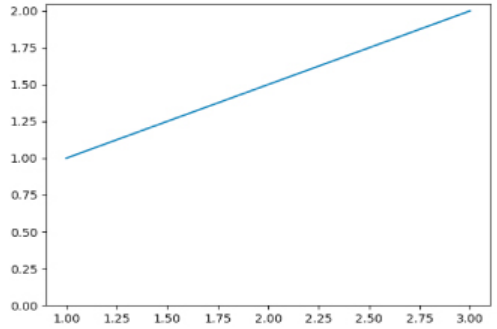


$$\theta_0 = 1.5$$
$$\theta_1 = 0$$

This yields h(x) = 1.5 + 0x. 0x means no slope, and y will always be the constant 1.5.

How does Gradient Descent work?

$$\theta_0 = 1$$
$$\theta_1 = 0.5$$



Where, h(x) = 1 + 0.5x

This yields h(x) = 1.5 + 0x. 0x means no slope, and y will always be the constant 1.5.

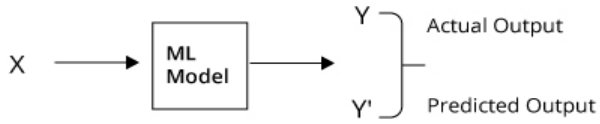
Cost Function

- The objective in the case of gradient descent is to find a line of best fit for some given *inputs*, or X values, and any number of Y values, or *outputs*.

- A cost function is defined as *"**a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.**"*

Cost Function

With a known set of inputs and their corresponding outputs, a machine learning model attempts to make predictions according to the new set of inputs.



Error = Y'(Predicted) - Y(Actual)

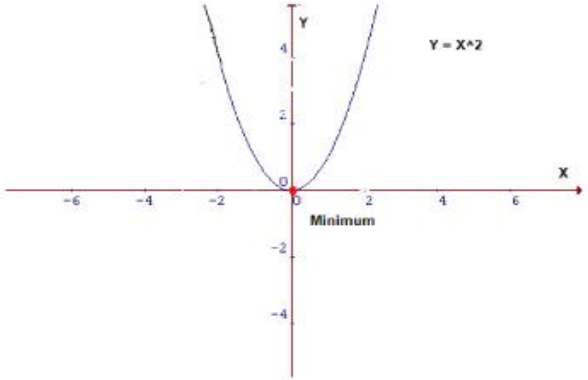
This relates to the idea of a **Cost function or Loss function**.

A **Cost Function/Loss Function** tells us “how good” our model is at making predictions for a given set of parameters. The cost function has a curve and a gradient, the slope of this curve helps us to update our parameters and make an accurate model.

Minimizing the Cost Function

- It is always the primary goal of any Machine Learning Algorithm to minimize the Cost Function. Minimizing cost functions will also result in a lower error between the predicted values and the actual values which also denotes that the algorithm has performed well in learning.

- Generally, the cost function is in the form of $\mathbf{Y = X^2}$. In a Cartesian coordinate system, this represents an equation for a parabola which can be graphically represented as :



Minimizing the Cost Function

- Now a function is required which will minimize the parameters over a dataset. The most common function which is often used is the **mean squared error**. It measures the difference between the estimated value (the prediction) and the estimator (the dataset).

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

It turns out we can adjust the equation a little to make the calculation down the track a little more simple.

Minimizing the Cost Function

Why do we take the squared differences and simply not the absolute differences?

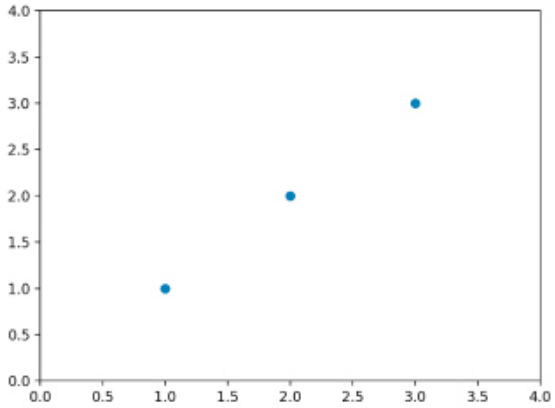
- Because the squared differences make it easier to derive a regression line. Indeed, to find that line we need to compute the first derivative of the Cost function, and it is much harder to compute the derivative of absolute values than squared values.

- The squared differences increase the error distance, thus, making the bad predictions more pronounced than the good ones.

$$\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Minimizing the Cost Function: Example

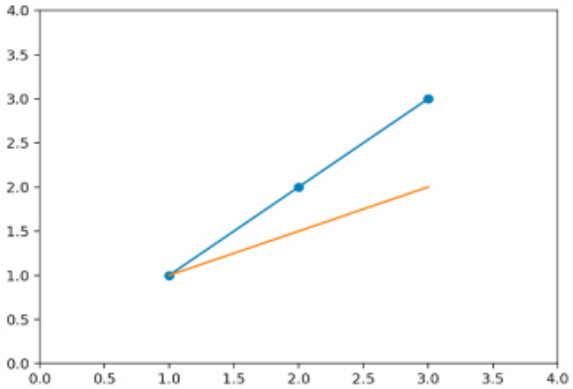
- Let us apply this cost function to the following data:



Here we will calculate some of the theta values and then plot the cost function by hand. Since this function passes through (0, 0), we will look only at a single value of theta. Also, let us refer to the cost function as J(Θ) from now on.

Minimizing the Cost Function: Example

- When the value of Θ is 1, for J(1), we get a 0. You will notice the value of J(1) gives a straight line which fits the data perfectly. Now let us try with Θ = 0.5



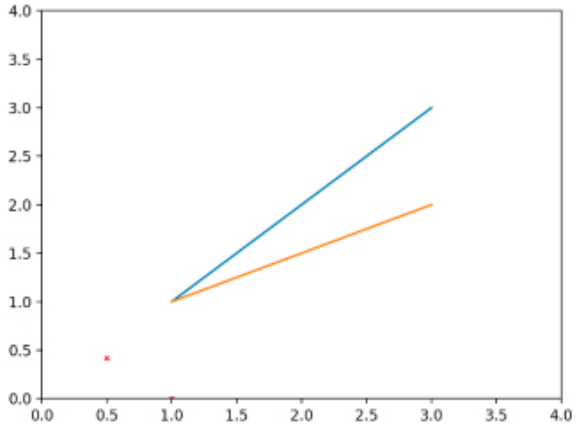
Here we will calculate some of the theta values and then plot the cost function by hand. Since this function passes through (0, 0), we will look only at a single value of theta.

Minimizing the Cost Function: Example

- The MSE function gives us a value of 0.58. Let's plot both our values so far:

J(1) = 0

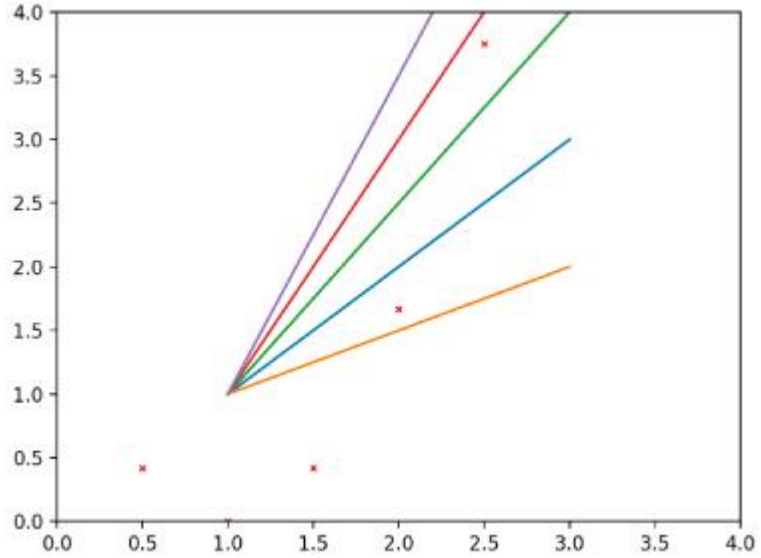
J(0.5) = 0.58



With J(1) and J(0.5)

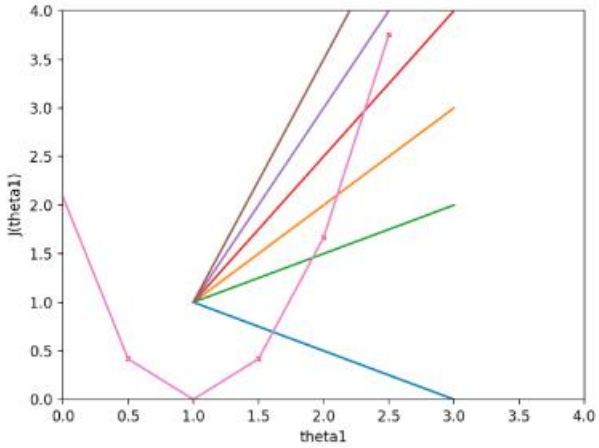
Minimizing the Cost Function: Example

- Let us go ahead and calculate some more values of J(Θ).



Minimizing the Cost Function: Example

Now if we join the dots carefully, we will get -



As we can see, the cost function is at a minimum when theta = 1, which means the initial data is a straight line with a slope or gradient of 1 as shown by the orange line in the above figure.

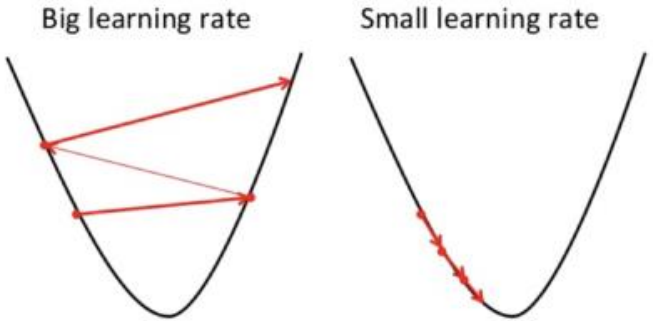
Using a trial and error method, we minimized J(Θ). We did all of these by trying out a lot of values and with the help of visualizations. **Gradient Descent** does the same thing in a much better way, by changing the theta values or parameters until it descends to the minimum value.

Learning Rate

- Let us now start by initializing theta0 and theta1 to any two values, say 0 for both, and go from there. The algorithm is as follows:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \text{(for } j = 0 \text{ and } j = 1\text{)}$$

where α, alpha, is the **learning rate**, or how rapidly do we want to move towards the minimum. We can always overshoot if the value of α is too large.



Learning Rate

The derivative which refers to the slope of the function is calculated. Here we calculate the partial derivative of the cost function. It helps us to know the direction (sign) in which the coefficient values should move so that they attain a lower cost on the following iteration.

$$\frac{\partial}{\partial \theta_j}$$

Once we know the direction from the derivative, we can update the coefficient values. Now you need to specify a learning rate parameter which will control how much the coefficients can change on each update.

coefficient = coefficient – (alpha * delta)

$$\begin{aligned}&\text{repeat until convergence } \{\\ &\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)\\ &\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)}\\ &\}\end{aligned}$$

Derivation of Simple Linear Regression Using Gradient Descent

1. Introduction to Simple Linear Regression

Simple Linear Regression models the relationship between a dependent variable y and an independent variable x using the equation:

$$y = \theta_1 x + \theta_0$$

where:

- θ_1 is the slope (coefficient),
- θ_0 is the intercept.

Given a dataset $\{(x_i, y_i)\}_{i=1}^{n}$, we aim to find the optimal values of θ_1 and θ_0 that minimize the error.

Derivation of Simple Linear Regression Using Gradient Descent

2. Defining the Cost Function

The **Mean Squared Error (MSE)** is used as the cost function:

$$J(\theta_1, \theta_0) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where:

$$\hat{y}_i = \theta_1 x_i + \theta_0$$

Thus,

$$J(\theta_1, \theta_0) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - (\theta_1 x_i + \theta_0))^2$$

The factor $\frac{1}{2}$ is added for mathematical convenience when differentiating.

Derivation of Simple Linear Regression Using Gradient Descent

3. Applying Gradient Descent

Gradient Descent is an iterative optimization algorithm to minimize $J(\theta_1, \theta_0)$. The update rules for θ_1 and θ_0 are:

$$\theta_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$$

$$\theta_0 := \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}$$

where α is the learning rate.

Derivation of Simple Linear Regression Using Gradient Descent

4. Computing Partial Derivatives

Derivative w.r.t θ_1:

$$\frac{\partial J}{\partial \theta_1} = \frac{1}{n}\sum_{i=1}^{n}(-x_i(y_i - (\theta_1 x_i + \theta_0)))$$

$$= -\frac{1}{n}\sum_{i=1}^{n} x_i(y_i - \hat{y}_i)$$

Derivative w.r.t θ_0:

$$\frac{\partial J}{\partial \theta_0} = \frac{1}{n}\sum_{i=1}^{n}(-(y_i - (\theta_1 x_i + \theta_0)))$$

$$= -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)$$

Challenges in executing Gradient Descent

There are many cases where gradient descent fails to perform well. There are mainly three reasons when this would happen:

1. Data challenges
2. Gradient challenges
3. Implementation challenges

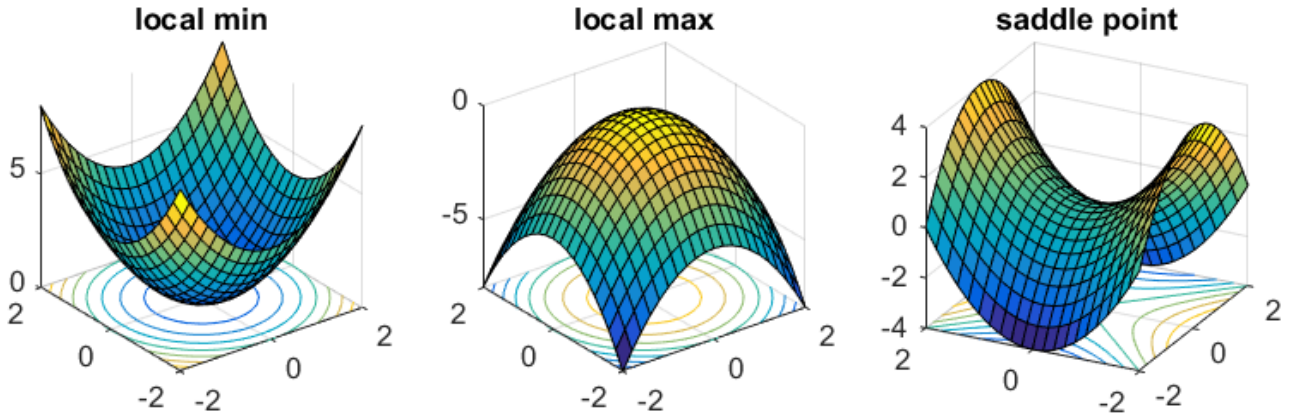
Challenges in executing Gradient Descent

1. **Data challenges**

- The arrangement of data sometimes leads to challenges. If it is arranged in such a way that it poses a **non-convex optimization problem** then it becomes difficult to perform optimization using gradient descent. ***Gradient descent works for problems which are arranged with a well-defined convex optimization problem.***
- During the optimization of a convex optimization problem, you will come across several minimal points. The lowest among all the points is called the global minimum, and other points are called the local minima. You will have to make sure you go to the global minimum and avoid local minima.
- There is also a saddle point problem. This is a situation where the gradient is zero but is not an optimal point. It cannot be avoided and is still an active part of the research.

Challenges in executing Gradient Descent

1. Data challenges



Challenges in executing Gradient Descent

2. Gradient challenges

- While using gradient descent, if the execution is not proper, it leads to certain problems like vanishing gradient. This happens when the gradient is either too small or too large which results in no convergence.

3. Implementation challenges

- Smaller memory results in the failure of network. A lot of neural network practitioners do not pay attention but it is very important to look at the resource utilization by the network.
- Another important thing to look at is to keep track of things like floating point considerations and hardware/software prerequisites.

Multilinear Regression (MLR)

- In the previous topic, we have learned about Simple Linear Regression, where a single Independent/Predictor(X) variable is used to model the response variable (Y). But there may be various cases in which the response variable is affected by more than one predictor variable; for such cases, the Multiple Linear Regression algorithm is used.

- Moreover, Multiple Linear Regression is an extension of Simple Linear regression as it takes more than one predictor variable to predict the response variable. We can define it as:

Multiple Linear Regression is one of the important regression algorithms which models the linear relationship between a single dependent continuous variable and more than one independent variable.

Assumptions for Multiple Linear Regression

- In Multiple Linear Regression, the target variable(Y) is a linear combination of multiple predictor variables x_1, x_2, x_3, ...,x_n. Since it is an enhancement of Simple Linear Regression, so the same is applied for the multiple linear regression equation, the equation becomes:

$$Y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + b_nx_n$$

Where,

Y= Output/Response variable

b_0, b_1, b_2, b_3 , ….b_n= Coefficients of the model.

x_1, x_2, x_3, x_4,…….x_n= Various Independent/feature variable

Assumptions for Multiple Linear Regression

- A **linear relationship** should exist between the Target and predictor variables.
- The regression residuals must be **normally distributed**.
- MLR assumes little or **no multicollinearity** (correlation between the independent variable) in data.

Implementation Multiple Linear Regression model using Python

Problem Description:

- We have a dataset of **50 start-up companies**. This dataset contains five main information: **R&D Spend, Administration Spend, Marketing Spend, State, and Profit for a financial year**. Our goal is to create a model that can easily determine which company has a maximum profit, and which is the most affecting factor for the profit of a company.

- Since we need to find the Profit, so it is the dependent variable, and the other four variables are independent variables. Below are the main steps of deploying the MLR model:

    1. **Data Pre-processing Steps**
    2. **Fitting the MLR model to the training set**
    3. **Predicting the result of the test set**

Step-1: Data Pre-processing Step:

The very first step is data pre-processing, which we have already discussed in this tutorial. This process contains the below steps:

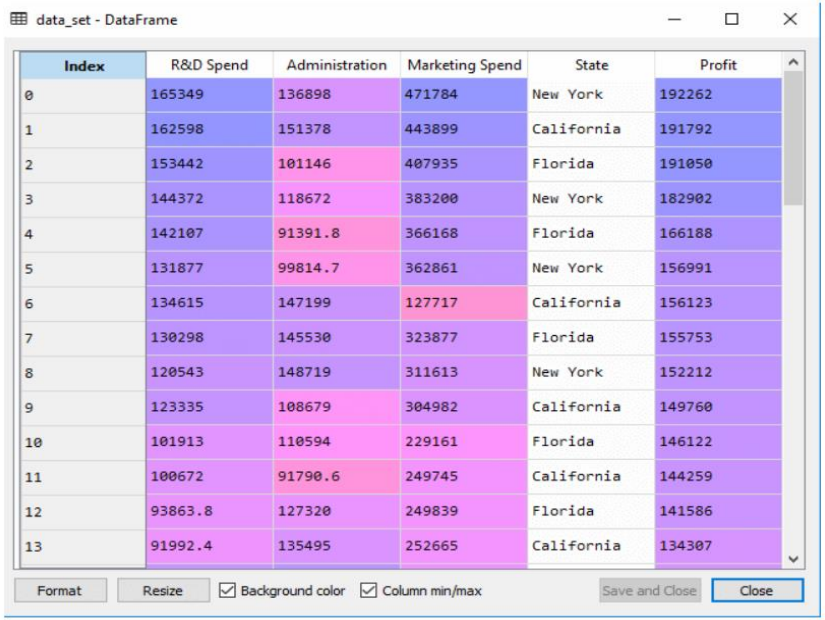
- **Importing libraries:** Firstly we will import the library which will help in building the model. Below is the code for it:

```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

```
#importing datasets
data_set= pd.read_csv('50_CompList.csv')
```

Step-1: Data Pre-processing Step:

Output: We will get the dataset as:



Index	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349	136898	471784	New York	192262
1	162598	151378	443899	California	191792
2	153442	101146	407935	Florida	191050
3	144372	118672	383200	New York	182902
4	142107	91391.8	366168	Florida	166188
5	131877	99814.7	362861	New York	156991
6	134615	147199	127717	California	156123
7	130298	145530	323877	Florida	155753
8	120543	148719	311613	New York	152212
9	123335	108679	304982	California	149760
10	101913	110594	229161	Florida	146122
11	100672	91790.6	249745	California	144259
12	93863.8	127320	249839	Florida	141586
13	91992.4	135495	252665	California	134307

Step-1: Data Pre-processing Step:

- **Extracting dependent and independent Variables:**

```
#Extracting Independent and dependent Variable
x= data_set.iloc[:, :-1].values
y= data_set.iloc[:, 4].values
```

Encoding Dummy Variables:

As we have one categorical variable (State), which cannot be directly applied to the model, so we will encode it. To encode the categorical variable into numbers, we will use the **LabelEncoder** class. But it is not sufficient because it still has some relational order, which may create a wrong model. So in order to remove this problem, we will use **OneHotEncoder**, which will create the dummy variables. Below is code for it:

Step: 2- Fitting our MLR model to the

- Now, we have well prepared our dataset in order to provide training, which means we will fit our regression model to the training set. It will be similar to as we did in Simple Linear Regression model. The code for this will be:

```
#Fitting the MLR model to the training set:
from sklearn.linear_model import LinearRegression
regressor= LinearRegression()
regressor.fit(x_train, y_train)
```

Output:

```
Out[9]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

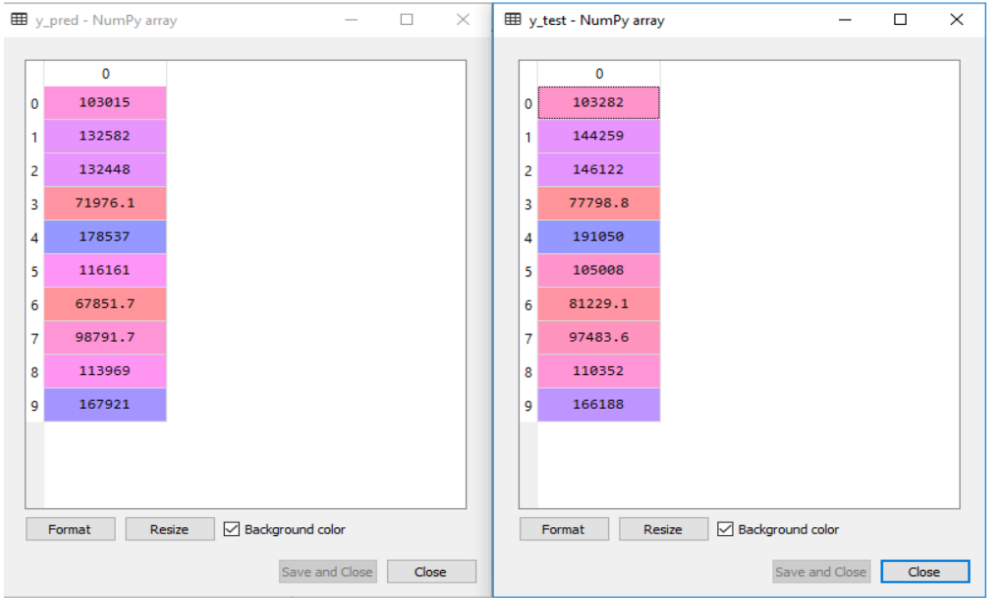
Step: 3- Prediction of Test set results

- The last step for our model is checking the performance of the model. We will do it by predicting the test set result. For prediction, we will create a **y_pred** vector. Below is the code for it:

```
#Predicting the Test set result;
y_pred= regressor.predict(x_test)
```

By executing the above lines of code, a new vector will be generated under the variable explorer option. We can test our model by comparing the predicted values and test set values.

Step: 3- Prediction of Test set results:



- In the above output, we have predicted result set and test set. We can check model performance by comparing these two value index by index. For example, the first index has a predicted value of **103015$** profit and test/real value of **103282$** profit. The difference is only of **267$**, which is a good prediction, so, finally, our model is completed here.

Step: 3- Prediction of Test set results:

- We can also check the score for training dataset and test dataset. Below is the code for it:

```
print('Train Score: ', regressor.score(x_train, y_train))
print('Test Score: ', regressor.score(x_test, y_test))
```

Output: The score is:

```
Train Score:  0.9501847627493607
Test Score:  0.9347068473282446
```

The above score tells that our model is 95% accurate with the training dataset and 93% accurate with the test dataset.

ML Polynomial Regression

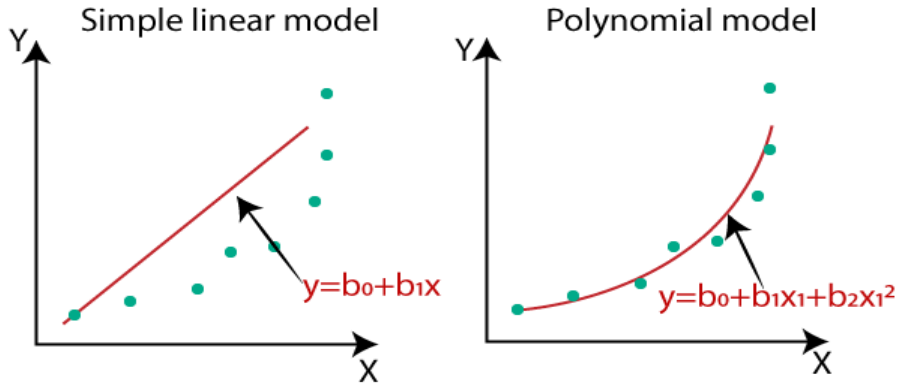
- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

$$y= b_0+b_1x_1+ b_2x_1^2+ b_2x_1^3+..... b_nx_1^n$$

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.
- **Hence, *"In Polynomial regression, the original features are converted into Polynomial features of required degree (2,3,..,n) and then modeled using a linear model."***

Need for Polynomial Regression:

- If we apply a linear model on a **linear dataset**, then it provides us a good result as we have seen in Simple Linear Regression, but if we apply the same model without any modification on a **non-linear dataset**, then it will produce a drastic output. Due to which loss function will increase, the error rate will be high, and accuracy will be decreased.
- So for such cases, **where data points are arranged in a non-linear fashion, we need the Polynomial Regression model**. We can understand it in a better way using the below comparison diagram of the linear dataset and non-linear dataset.



Need for Polynomial Regression:

- In the image, we have taken a dataset which is arranged non-linearly. So if we try to cover it with a linear model, then we can clearly see that it hardly covers any data point. On the other hand, a curve is suitable to cover most of the data points, which is of the Polynomial model.

- Hence, *if the datasets are arranged in a non-linear fashion, then we should use the Polynomial Regression model instead of Simple Linear Regression.*

Regression Equations

Simple Linear Regression equation:

$$y = b_0 + b_1 x \qquad \text{.....(a)}$$

Multiple Linear Regression equation:

$$y = b_0 + b_1 x + b_2 x_2 + b_3 x_3 + ... + b_n x_n \qquad \text{.....(b)}$$

Polynomial Regression equation:

$$y = b_0 + b_1 x + b_2 x^2 + b_3 x^3 + ... + b_n x^n \qquad \text{.....(c)}$$

When we compare the above three equations, we can clearly see that all three equations are Polynomial equations but differ by the degree of variables. The Simple and Multiple Linear equations are also Polynomial equations with a single degree, and the Polynomial regression equation is Linear equation with the nth degree. So if we add a degree to our linear equations, then it will be converted into Polynomial Linear equations.

Implementation of Polynomial Regression using Python:

- Here we will implement the Polynomial Regression using Python. We will understand it by comparing Polynomial Regression model with the Simple Linear Regression model. So first, let's understand the problem for which we are going to build the model.

Problem Description: There is a Human Resource company, which is going to hire a new candidate. The candidate has told his previous salary 160K per annum, and the HR have to check whether he is telling the truth or bluff. So to identify this, they only have a dataset of his previous company in which the salaries of the top 10 positions are mentioned with their levels. By checking the dataset available, we have found that there is a **non-linear relationship between the Position levels and the salaries**. Our goal is to build a **Bluffing detector regression** model, so HR can hire an honest candidate. Below are the steps to build such a model.

Implementation of Polynomial Regression using Python: Example

Position	Level(X-variable)	Salary(Y-Variable)
Business Analyst	1	45000
Junior Consultant	2	50000
Senior Consultant	3	60000
Manager	4	80000
Country Manager	5	110000
Region Manager	6	150000
Partner	7	200000
Senior Partner	8	300000
C-level	9	500000
CEO	10	1000000

Steps for Polynomial Regression:

- The main steps involved in Polynomial Regression are given below:
- Data Pre-processing
- Build a Linear Regression model and fit it to the dataset
- Build a Polynomial Regression model and fit it to the dataset
- Visualize the result for Linear Regression and Polynomial Regression model.
- Predicting the output.

Here, a Linear regression model as well as Polynomial Regression to see the results between the predictions is built. And Linear regression model is for reference.

Data Pre-processing Step:

The data pre-processing step will remain the same as in previous regression models, except for some changes. In the Polynomial Regression model, we will not use feature scaling, and also we will not split our dataset into training and test set. It has two reasons:

- The dataset contains very less information which is not suitable to divide it into a test and training set, else our model will not be able to find the correlations between the salaries and levels.

- In this model, we want very accurate predictions for salary, so the model should have enough information.

Data Pre-processing Step:

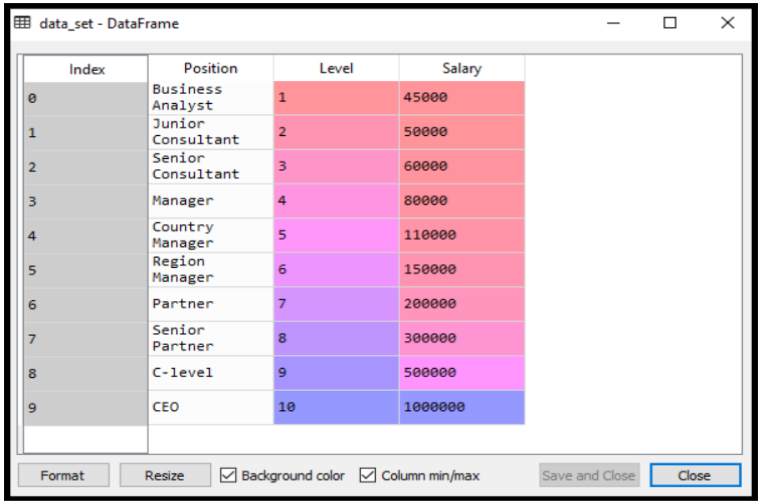
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

#importing datasets
data_set= pd.read_csv('Position_Salaries.csv')

#Extracting Independent and dependent Variable
x= data_set.iloc[:, 1:2].values
y= data_set.iloc[:, 2].values
```

- In the above lines of code, we have imported the important Python libraries to import dataset and operate on it.
- Next, we have imported the dataset '**Position_Salaries.csv**', which contains three columns (Position, Levels, and Salary), but we will consider only two columns (Salary and Levels).
- After that, we have extracted the dependent(Y) and independent variable(X) from the dataset. For x-variable, we have taken parameters as [:,1:2], because we want 1 index(levels), and included :2 to make it as a matrix.

Data Pre-processing Step:



Index	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

- As we can see in the above output, there are three columns present (Positions, Levels, and Salaries). But we are only considering two columns because Positions are equivalent to the levels or may be seen as the encoded form of Positions.
- Here we will predict the output for level **6.5** because the candidate has 4+ years' experience as a regional manager, so he must be somewhere between levels 7 and 6.

Building the Linear regression model:

- Now, we will build and fit the Linear regression model to the dataset. In building polynomial regression, we will take the Linear regression model as reference and compare both the results. The code is given below:

```
#Fitting the Linear Regression to the dataset
from sklearn.linear_model import LinearRegression
lin_regs= LinearRegression()
lin_regs.fit(x,y)
```

In the above code, we have created the Simple Linear model using **lin_regs** object of **LinearRegression** class and fitted it to the dataset variables (x and y).

Building the Polynomial regression model:

- Now we will build the Polynomial Regression model, but it will be a little different from the Simple Linear model. Because here we will use **PolynomialFeatures** class of **preprocessing** library. We are using this class to add some extra features to our dataset.

```
#Fitting the Polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_regs= PolynomialFeatures(degree= 2)
x_poly= poly_regs.fit_transform(x)
lin_reg_2 =LinearRegression()
lin_reg_2.fit(x_poly, y)
```

In the above lines of code, we have used **poly_regs.fit_transform(x)**, because first we are converting our feature matrix into polynomial feature matrix, and then fitting it to the Polynomial regression model. The parameter value(degree= 2) depends on our choice. We can choose it according to our Polynomial features.

Building the Polynomial regression model:

- After executing the code, we will get another matrix **x_poly**, which can be seen under the variable explorer option:

	0	1	2
0	1	1	1
1	1	2	4
2	1	3	9
3	1	4	16
4	1	5	25
5	1	6	36
6	1	7	49
7	1	8	64
8	1	9	81
9	1	10	100

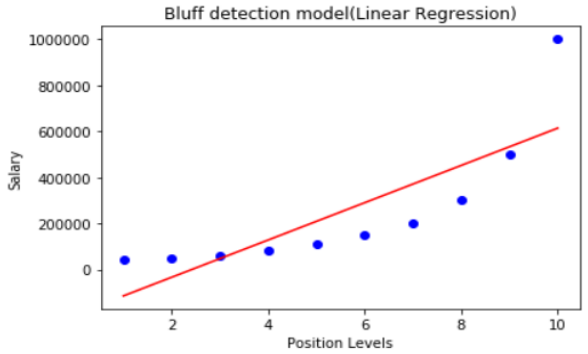
Next, we have used another LinearRegression object, namely **lin_reg_2**, to fit our **x_poly** vector to the linear model

Visualizing the result for Linear regression:

- Linear regression model as we did in Simple Linear Regression

```
#Visulaizing the result for Linear Regression model
mtp.scatter(x,y,color="blue")
mtp.plot(x,lin_regs.predict(x), color="red")
mtp.title("Bluff detection model(Linear Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```

Output:



Output

- In the above output image, we can clearly see that the regression line is so far from the datasets. Predictions are in a red straight line, and blue points are actual values. If we consider this output to predict the value of CEO, it will give a salary of approx. 600000$, which is far away from the real value.
- So we need a curved model to fit the dataset other than a straight line.

Visualizing the result for Polynomial Regression

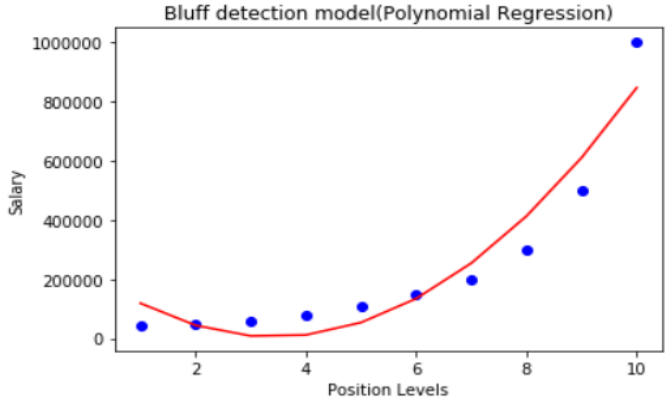
- Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.

```
#Visulaizing the result for Polynomial Regression
mtp.scatter(x,y,color="blue")
mtp.plot(x, lin_reg_2.predict(poly_regs.fit_transform(x)), color="red")
mtp.title("Bluff detection model(Polynomial Regression)")
mtp.xlabel("Position Levels")
mtp.ylabel("Salary")
mtp.show()
```

In the above code, we have taken lin_reg_2.predict(poly_regs.fit_transform(x), instead of x_poly, because we want a Linear regressor object to predict the polynomial features matrix.

Visualizing the result for Polynomial Regression

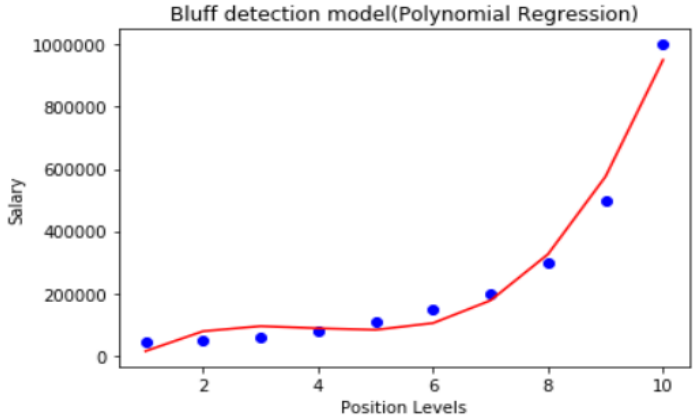
- Here we will visualize the result of Polynomial regression model, code for which is little different from the above model.



As we can see in the above output image, the predictions are close to the real values. The above plot will vary as we will change the degree.

Visualizing the result for Polynomial Regression

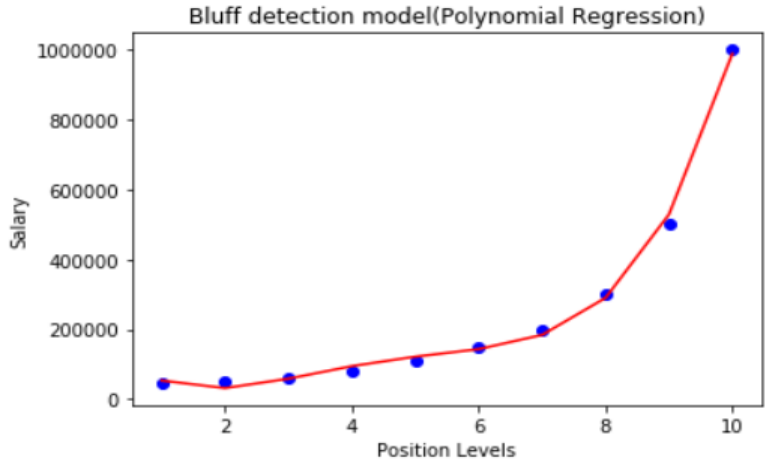
- **For degree= 3:**
- If we change the degree=3, then we will give a more accurate plot, as shown in the below image.



SO as we can see here in the above output image, the predicted salary for level 6.5 is near to 170K$-190k$, which seems that future employee is saying the truth about his salary.

Visualizing the result for Polynomial Regression

- **Degree= 4:**
- Let's again change the degree to 4, and now will get the most accurate plot. Hence we can get more accurate results by increasing the degree of Polynomial.



Predicting the final result with the Linear Regression model:

- Now, we will predict the final output using the Linear regression model to see whether an employee is saying truth or bluff. So, for this, we will use the **predict()** method and will pass the value 6.5. Below is the code for it:

```
lin_pred = lin_regs.predict([[6.5]])
print(lin_pred)
```

Output:

```
[330378.78787879]
```

Predicting the final result with the Polynomial Regression model:

- Now, we will predict the final output using the Polynomial Regression model to compare with Linear model. Below is the code for it:

```
poly_pred = lin_reg_2.predict(poly_regs.fit_transform([[6.5]]))
print(poly_pred)
```

Output:

```
[158862.45265153]
```

As we can see, the predicted output for the Polynomial Regression is [158862.45265153], which is much closer to real value hence, we can say that future employee is saying true.

Performance Evaluation Metrics: Linear Regression

1. R Square Score

2. Mean Square Error(MSE)/Root Mean Square Error(RMSE)

3. Mean Absolute Error(MAE)

Decision Tree Fundamentals

- Tree-based machine learning methods are among the most commonly used supervised learning methods. They are constructed by two entities; branches and nodes. Tree-based ML methods are built by recursively splitting a training sample, using different features from a dataset at each node that splits the data most effectively. The splitting is based on learning simple decision rules inferred from the training data.

- Generally, tree-based ML methods are simple and intuitive; to predict a class label or value, we start from the top of the tree or the root and, using branches, go to the nodes by comparing features on the basis of which will provide the best split.

Decision Tree Fundamentals

- Tree-based methods also use the mean for continuous variables or mode for categorical variables when making predictions on training observations in the regions they belong to.

- Since the set of rules used to segment the predictor space can be summarized in a visual representation with branches that show all the possible outcomes, these approaches are commonly referred to as **decision tree methods**.

- The methods are flexible and can be applied to either classification or regression problems. ***Classification and Regression Trees*** **(CART)** is a commonly used term by Leo Breiman, referring to the flexibility of the methods in solving both linear and non-linear predictive modeling problems.

Decision Tree Fundamentals

Types of Decision Trees

- Decision trees can be classified based on the type of target or response variable.

i. Classification Trees

- The default type of decision trees, used when the response variable is categorical—i.e. predicting whether a team will win or lose a game.

ii. Regression Trees

- Used when the target variable is continuous or numerical in nature—i.e. predicting house prices based on year of construction, number of rooms, etc.

Decision Tree Fundamentals

Advantages of Tree-based Machine Learning Methods

- **Interpretability**: Decision tree methods are easy to understand even for non-technical people.
- The **data type isn't a constraint**, as the methods can handle both categorical and numerical variables.
- **Data exploration** — Decision trees help us easily identify the most significant variables and their correlation.

Decision Tree Fundamentals

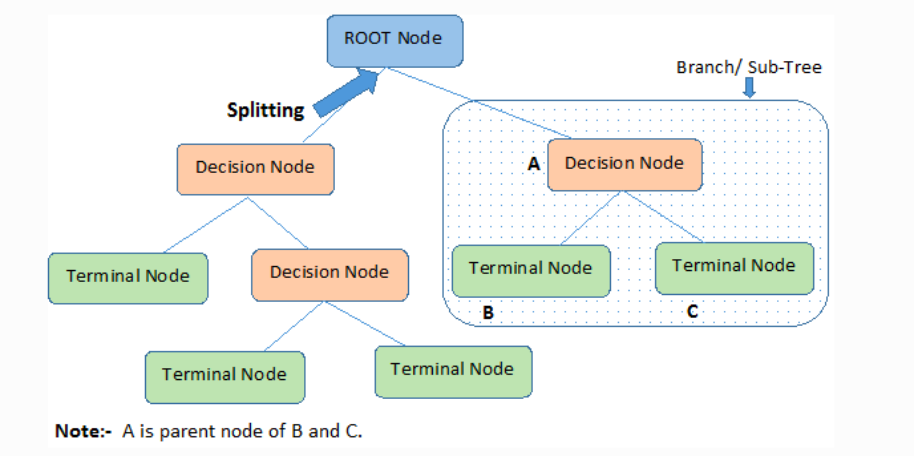
Disadvantages of Tree-based Machine Learning Methods

- Large decision trees are **complex, time-consuming and less accurate** in predicting outcomes.

- Decision trees **don’t fit well for continuous variables**, as they lose important information when segmenting the data into different regions.

Decision Tree Fundamentals

Common Terminology



i) **Root node** — this represents the entire population or the sample, which gets divided into two or more homogenous subsets.
ii) **Splitting** — subdividing a node into two or more sub-nodes

Decision Tree Fundamentals

Common Terminology

iii) **Decision node** — this is when a sub-node is divided into further sub-nodes.

iv) **Leaf/Terminal node** — this is the final/last node that we consider for our model output. It cannot be split further.

v) **Pruning —** removing unnecessary sub-nodes of a decision node to combat overfitting.

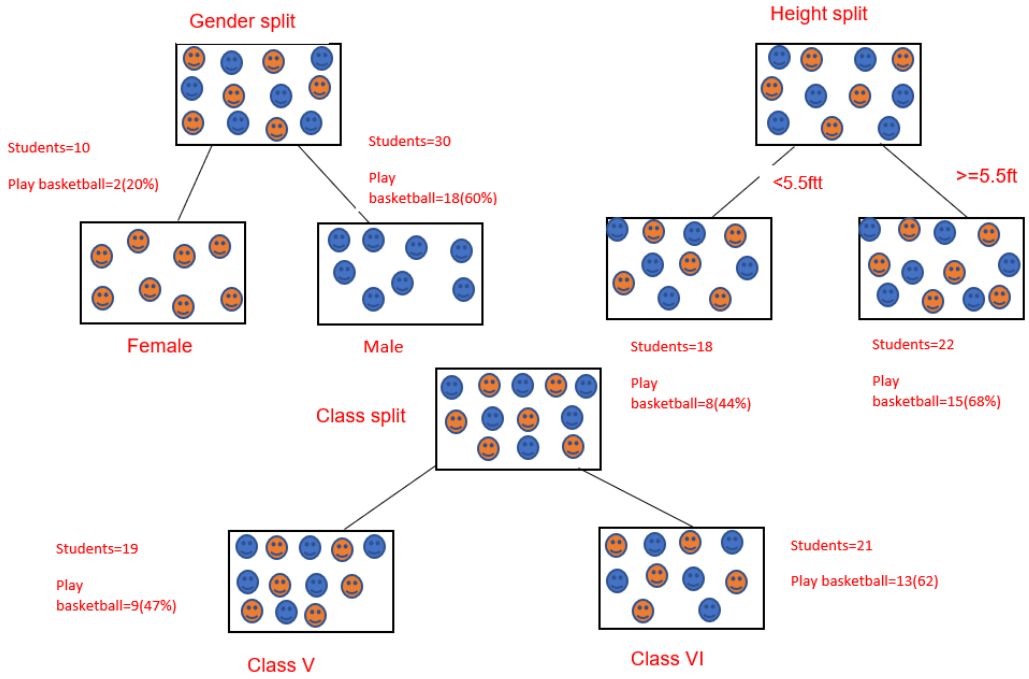
vi) **Branch/Sub-tree** — the sub-section of the entire tree.

vii) **Parent and Child node** — a node that's subdivided into a sub-node is a parent, while the sub-node is the child node.

Algorithms in Tree-based Machine Learning Models

- The decision of splitting a tree affects its accuracy. Tree-based machine learning models use multiple algorithms to decide where to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of the resultant sub-nodes. **Algorithm selection is based on the type of target variable.**
- Suppose you're the basketball coach of a grade school. The inter-school basketball competitions are nearby, and you want to do a survey to determine which students play basketball in their leisure time. The sample selected is 40 students. The selection criterion is based on a number of factors such as gender, height, and class.
- As a coach, you'd want to select the students based on the most significant input variable among the three variables.
- Decision tree algorithms will help the coach identify the right sample of students using the variable, which creates the best homogenous set of student players.

Algorithms in Tree-based Machine Learning Models



Tree Based Methods

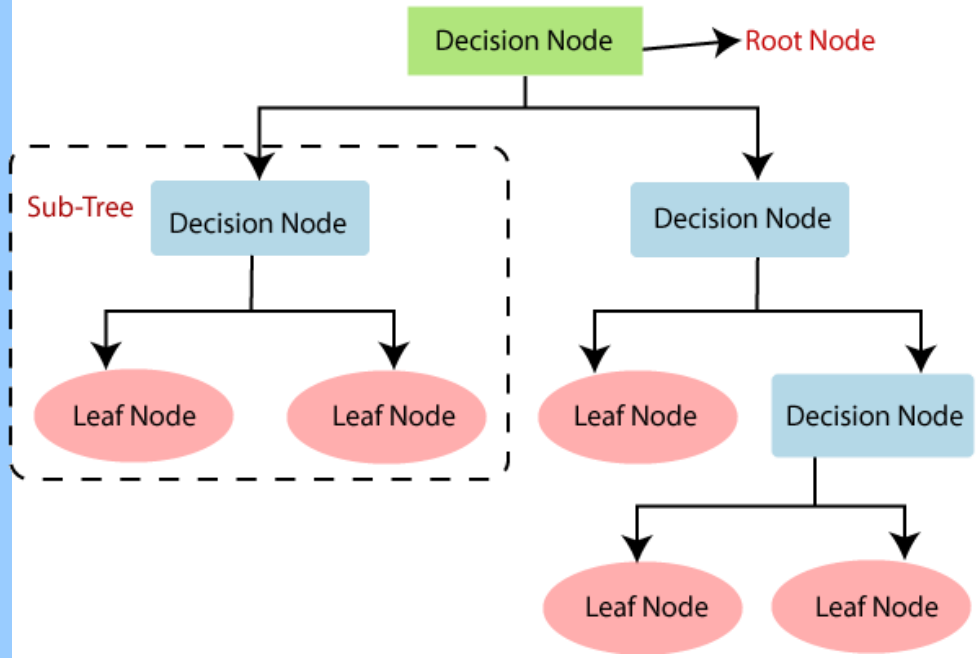
There are different tree-based algorithms that you can use, such as

- Decision Trees
- Random Forest
- Gradient Boosting
- Bagging (Bootstrap Aggregation)

Decision Tree Classification Algorithm

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- ***It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.***
- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

Decision Tree Classification Algorithm



Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main point to remember while creating a machine learning model. Below are the two reasons for using the Decision tree:

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

How does the Decision Tree algorithm Work?

- In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node.
- For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree. The complete process can be better understood using the below algorithm:

Step-1: Begin the tree with the root node, says S, which contains the complete dataset.

Step-2: Find the best attribute in the dataset using **Attribute Selection Measure (ASM).**

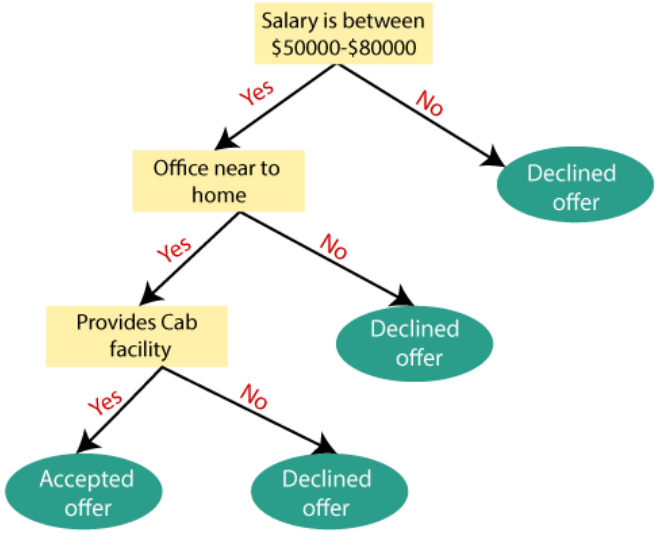
Step-3: Divide the S into subsets that contains possible values for the best attributes.

Step-4: Generate the decision tree node, which contains the best attribute.

Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

How does the Decision Tree algorithm Work?

- **Example:** Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer). Consider the below diagram:



Attribute Selection Measures

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

1. Information Gain:

- Information gain is the measurement of changes in **entropy** after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain**, and a node/attribute having the highest information gain is split first**. It can be calculated using the below formula:

Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)

1. Information Gain

Entropy: Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

$$H(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

Intuitively, it tells us about the predictability of a certain event. Example, consider a coin toss whose probability of heads is 0.5 and probability of tails is 0.5. Here the entropy is the highest possible, since there's no way of determining what the outcome might be. Alternatively, consider a coin which has heads on both the sides, the entropy of such an event can be predicted perfectly since we know beforehand that it'll always be heads. In other words, this event has **no randomness** hence it's entropy is zero. In particular, lower values imply less uncertainty while higher values imply high uncertainty.

1. Information Gain

Information gain is also called as Kullback-Leibler divergence denoted by IG(S,A) for a set S is the effective change in entropy after deciding on a particular attribute A. It measures the relative change in entropy with respect to the independent variables.

$$IG(S,A) = H(S) - H(S,A)$$

$$IG(S,A) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

where IG(S, A) is the information gain by applying feature A. H(S) is the Entropy of the entire set, while the second term calculates the Entropy after applying the feature A, where P(x) is the probability of event x.

2. Gini Index

- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the **low Gini index should be preferred** as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

where j is the number classes.

Decision tree

- Decision tree learning is a widely used method in data mining, celebrated for its simplicity and clarity. Several algorithms are employed to build decision trees, each with distinct characteristics.

Algorithm	Key Features	Best Used For
ID3	Uses information gain, handles categorical data	Simple classification tasks
C4.5	Handles continuous and discrete attributes, uses gain ratio	More complex classification tasks
CART	Supports both classification and regression, uses Gini index	Versatile applications

Pruning: Getting an Optimal Decision tree

- *Pruning is a process of deleting the unnecessary nodes from a tree in order to get the optimal decision tree.*

- A too-large tree increases the risk of overfitting, and a small tree may not capture all the important features of the dataset. Therefore, a technique that decreases the size of the learning tree without reducing accuracy is known as Pruning. There are mainly two types of tree **pruning** technology used:

- **Cost Complexity Pruning**
- **Reduced Error Pruning**

DT: Advantages and Disadvantages

Advantages

- It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
- It can be very useful for solving decision-related problems.
- It helps to think about all the possible outcomes for a problem.
- **There is less requirement of data cleaning compared to other algorithms**

Disadvantages

- The decision tree contains lots of layers, which makes it complex.
- It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
- For more class labels, the computational complexity of the decision tree may increase.

Decision tree: Example

Here the decision or the outcome variable is **Continuous**, e.g. a number like 123. **Working** Now that we know what a Decision Tree is, we'll see how it works internally. There are many algorithms out there which construct Decision Trees, but one of the best is called as **ID3 Algorithm**. ID3 Stands for **Iterative Dichotomiser 3**. Before discussing the ID3 algorithm, we'll go through few definitions.

•**Entropy:**

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision tree: Example

ID3 Algorithm will perform following tasks recursively

1. Create root node for the tree
2. If all examples are positive, return leaf node ‘positive’
3. Else if all examples are negative, return leaf node ‘negative’
4. Calculate the entropy of current state H(S)
5. For each attribute, calculate the entropy with respect to the attribute ‘x’ denoted by H(S, x)
6. Select the attribute which has maximum value of IG(S, x)
7. Remove the attribute that offers highest IG from the set of attributes
8. Repeat until we run out of all attributes, or the decision tree has all leaf nodes.

Decision tree: Example

The initial step is to calculate H(S), the Entropy of the current state. In the above example, we can see in total there are 5 No's and 9 Yes's.

Yes	No	Total
9	5	14

$$Entropy(S) = \sum_{x \in X} p(x) \log_2 \frac{1}{p(x)}$$

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

$$= 0.940$$

Remember that the Entropy is 0 if all members belong to the same class, and 1 when half of them belong to one class and other half belong to other class that is perfect randomness. Here it's 0.94 which means the distribution is fairly random.

Decision tree: Example

the next step is to choose the attribute that gives us highest possible Information Gain which we'll choose as the root node. Let's start with 'Wind'

$$IG(S, Wind) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

where 'x' are the possible values for an attribute. Here, attribute 'Wind' takes two possible values in the sample data, hence x = {Weak, Strong} We'll have to calculate:

1. $H(S_{weak})$
2. $H(S_{strong})$
3. $P(S_{weak})$
4. $P(S_{strong})$
5. $H(S) = 0.94$ which we had already calculated in the previous example

Amongst all the 14 examples we have **8 places where the wind is weak and 6 where the wind is Strong**.

Decision tree: Example

Wind = Weak	Wind = Strong	Total
8	6	14

$$P(S_{weak}) = \frac{Number\ of\ Weak}{Total}$$

$$= \frac{8}{14}$$

$$P(S_{strong}) = \frac{Number\ of\ Strong}{Total}$$

$$= \frac{6}{14}$$

Now, out of the 8 Weak examples, 6 of them were 'Yes' for Play Golf and 2 of them were 'No' for 'Play Golf'. So, we have,

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

$$= 0.811$$

Similarly, out of 6 Strong examples, we have **3 examples where the outcome was 'Yes' for Play Golf and 3 where we had 'No' for Play Golf**.

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

$$= 1.000$$

Decision tree: Example

Remember, here half items belong to one class while other half belong to other. Hence we have perfect randomness. Now we have all the pieces required to calculate the Information Gain,

$$IG(S, Wind) = H(S) - \sum_{i=0}^{n} P(x) * H(x)$$

$$IG(S, Wind) = H(S) - P(S_{weak}) * H(S_{weak}) - P(S_{strong}) * H(S_{strong})$$

$$= 0.940 - \left(\frac{8}{14}\right)(0.811) - \left(\frac{6}{14}\right)(1.00)$$

$$= 0.048$$

Which tells us the Information Gain by considering 'Wind' as the feature and give us information gain of **0.048**. Now we must similarly calculate the Information Gain for all the features.

$$IG(S, Outlook) = 0.246$$

$$IG(S, Temperature) = 0.029$$

$$IG(S, Humidity) = 0.151$$

$$IG(S, Wind) = 0.048 \text{ (Previous example)}$$

We can clearly see that IG(S, Outlook) has the highest information gain of 0.246, **hence we chose Outlook attribute as the root node**. At this point, the decision tree looks like.

Decision tree: Example

We can clearly see that IG(S, Outlook) has the highest information gain of 0.246, **hence we chose Outlook attribute as the root node**. At this point, the decision tree looks like.

$$IG(S, Outlook) = 0.246$$

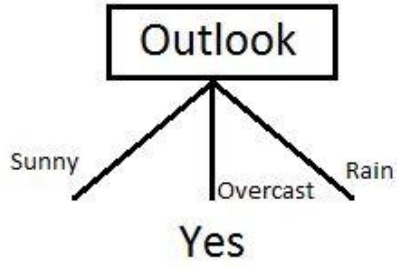
$$IG(S, Temperature) = 0.029$$

$$IG(S, Humidity) = 0.151$$

$$IG(S, Wind) = 0.048 \text{ (Previous example)}$$

Which tells us the Information Gain by considering 'Wind' as the feature and give us information gain of **0.048**. Now we must similarly calculate the Information Gain for all the features.

We can clearly see that IG(S, Outlook) has the highest information gain of 0.246, **hence we chose Outlook attribute as the root node**. At this point, the decision tree looks like.



Here we observe that whenever the outlook is Overcast, Play Golf is always 'Yes', it's no coincidence by any chance, the simple tree resulted because of **the highest information gain is given by the attribute Outlook**.

Decision tree: Example

Now how do we proceed from this point? We can simply apply recursion, you might want to look at the algorithm steps described Earlier.

Now that we've used Outlook, we've got three of them remaining Humidity, Temperature, and Wind. And, we had three possible values of Outlook: Sunny, Overcast, Rain. Where the Overcast node already ended up having leaf node 'Yes', so we're left with two subtrees to compute: Sunny and Rain

Next step would be computing $H(S_{sunny})$.

Table where the value of Outlook is Sunny looks like:

Temperature	Humidity	Wind	Play Golf
Hot	High	Weak	No
Hot	High	Strong	No
Mild	High	Weak	No
Cool	Normal	Weak	Yes
Mild	Normal	Strong	Yes

$$H(S_{sunny}) = \left(\frac{3}{5}\right)\log_2\left(\frac{3}{5}\right) - \left(\frac{2}{5}\right)\log_2\left(\frac{2}{5}\right) = 0.96$$

In the similar fashion, we compute the following values

$IG(S_{sunny}, Humidity) = 0.96$

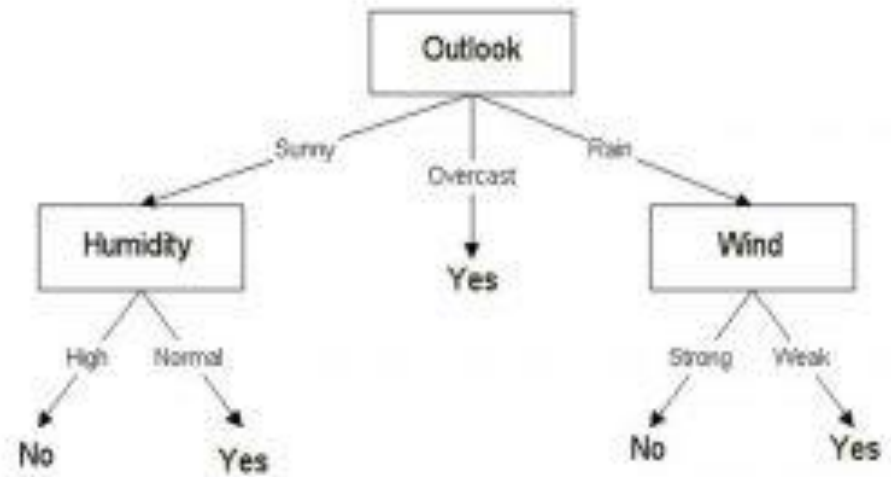
$IG(S_{sunny}, Temperature) = 0.57$

$IG(S_{sunny}, Wind) = 0.019$

Decision tree: Example

As we can see the highest Information Gain is given by **Humidity**. Proceeding in the same way with will give us **Wind** as the one with highest information gain.

The final Decision Tree looks something like this



Clustering

Clustering is a type of unsupervised learning wherein data points are grouped into different sets based on their degree of similarity.

The various types of clustering are:

1. Hierarchical clustering
2. Partitioning clustering

Hierarchical clustering is further subdivided into:

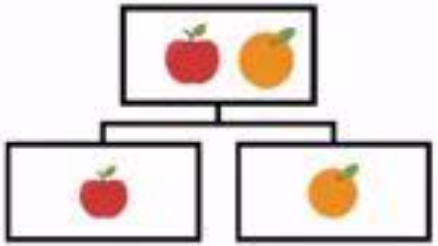
- Agglomerative clustering
- Divisive clustering

Partitioning clustering is further subdivided into:

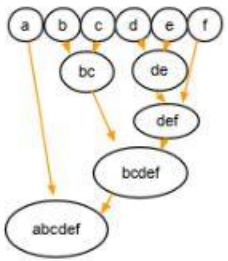
- K-Means clustering
- Fuzzy C-Means clustering

Hierarchical Clustering

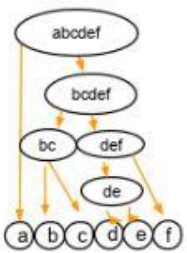
- Hierarchical clustering uses a tree-like structure, like so:



- In agglomerative clustering, there is a bottom-up approach. We begin with each element as a separate cluster and merge them into successively more massive clusters, as shown on the right:



- Divisive clustering is a top-down approach. We begin with the whole set and proceed to divide it into successively smaller clusters, as you can see on the right:



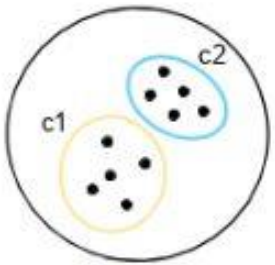
Partitioning Clustering

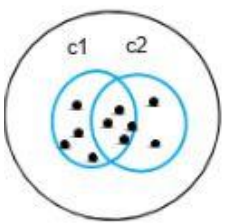
- Partitioning clustering is split into two subtypes - K-Means clustering and Fuzzy C-Means.

- In k-means clustering, the objects are divided into several clusters mentioned by the number 'K.' So if we say K = 2, the objects are divided into two clusters, c1 and c2, as shown:

- Here, the features or characteristics are compared, and all objects having similar characteristics are clustered together.

- Fuzzy c-means is very similar to k-means in the sense that it clusters objects that have similar characteristics together. In k-means clustering, a single object cannot belong to two different clusters. But in c-means, objects can belong to more than one cluster, as shown.





K-means algorithm

- K-Means clustering is an unsupervised learning algorithm. There is no labeled data for this clustering, unlike in supervised learning. K-Means performs the division of objects into clusters that share similarities and are dissimilar to the objects belonging to another cluster.

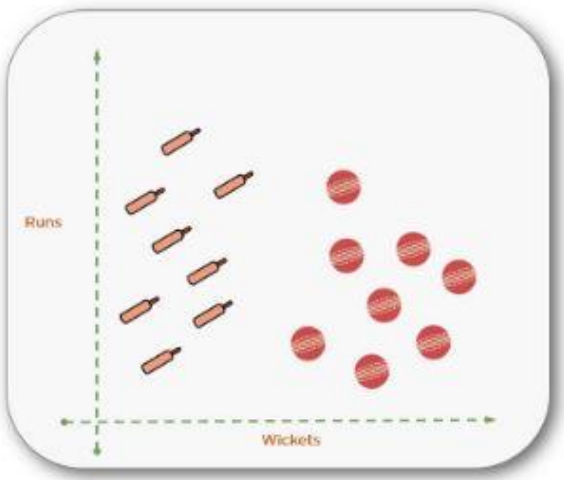
- The term 'K' is a number. You need to tell the system how many clusters you need to create. For example, K = 2 refers to two clusters. There is a way of finding out what is the best or optimum value of K for a given data.

- For a better understanding of k-means, let's take an example from cricket. Imagine you received data on a lot of cricket players from all over the world, which gives information on the runs scored by the player and the wickets taken by them in the last ten matches. Based on this information, we need to group the data into two clusters, namely batsman and bowlers.

Solution:

Assign data points

- Here, we have our data set plotted on 'x' and 'y' coordinates. The information on the y-axis is about the runs scored, and on the x-axis about the wickets taken by the players.

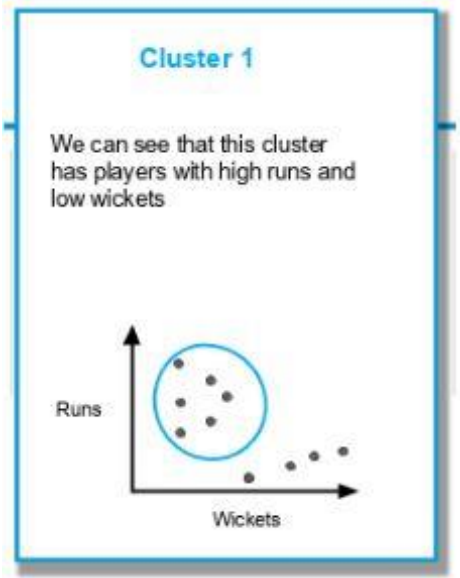


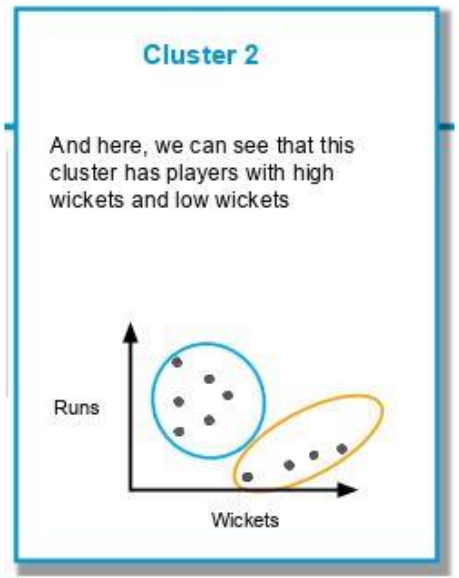
When we plot the data, we can see a clear separation between the class of batsman & bowler

Batsman Bowler

simplilearn

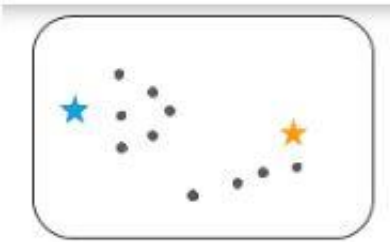
Solution:


Cluster 1
We can see that this cluster has players with high runs and low wickets
Runs
Wickets


Cluster 2
And here, we can see that this cluster has players with high wickets and low wickets
Runs
Wickets

K-Mean Clustering: Centroids

- The first step in k-means clustering is the allocation of two centroids randomly (as K=2). Two points are assigned as centroids. Note that the points can be anywhere, as they are random points. They are called centroids, but initially, they are not the central point of a given data set.



- The next step is to determine the distance between each of the randomly assigned centroids' data points. For every point, the distance is measured from both the centroids, and whichever distance is less, that point is assigned to that centroid. You can see the data points attached to the centroids and represented here in blue and yellow.

K-Mean Clustering: Centroids

- The next step is to determine the actual centroid for these two clusters. The original randomly allocated centroid is to be repositioned to the actual centroid of the clusters.

- This process of calculating the distance and repositioning the centroid continues until we obtain our final cluster. Then the centroid repositioning stops.

Distance Measure

Distance measure determines the similarity between two elements and influences the shape of clusters.

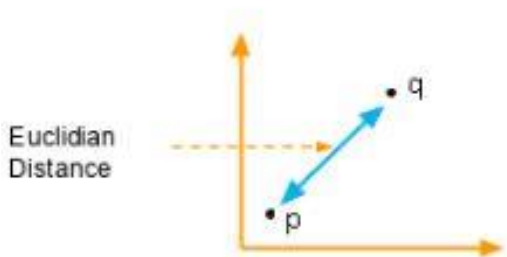
K-Means clustering supports various kinds of distance measures, such as:

- Euclidean distance measure
- Manhattan distance measure
- A squared Euclidean distance measure
- Cosine distance measure

Euclidean Distance Measure

- The most common case is determining the distance between two points. If we have a point P and point Q, the euclidean distance is an ordinary straight line. It is the distance between the two points in Euclidean space.
- The formula for distance between two points is shown below:

$$d=\sqrt{\sum_{i=1}^{n}(q_i-p_i)^2}$$



Squared Euclidean Distance Measure

Distance Measure

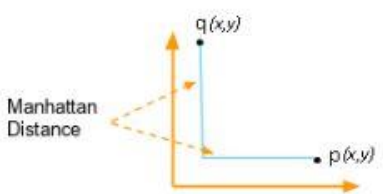
- This is identical to the Euclidean distance measurement but does not take the square root at the end. The formula is shown below:

$$d = \sum_{i=1}^{n} (q_i - p_i)^2$$

Manhattan Distance Measure

- The Manhattan distance is the simple sum of the horizontal and vertical components or the distance between two points measured along axes at right angles.

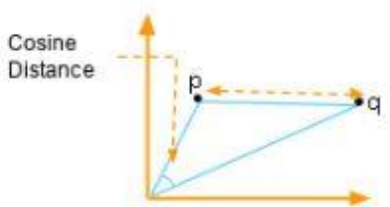
$$d = \sum_{i=1}^{n} |q_x - p_x| + |q_x - p_y|$$



Cosine Distance Measure

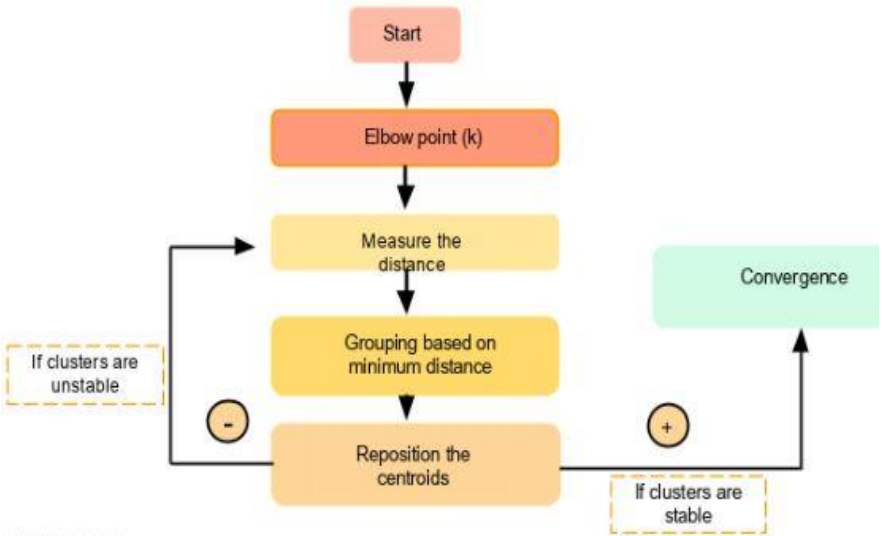
- In this case, we take the angle between the two vectors formed by joining the origin point. The formula is shown below:

$$d = \frac{\sum_{i=0}^{n-1} q_i - p_x}{\sum_{i=0}^{n-1}(q_i)^2 \times \sum_{i=0}^{n-1}(p_i)^2}$$



How Does K-Means Clustering Work?

- The flowchart below shows how k-means clustering works:



How Does K-Means Clustering Work?
Step 1

- The Elbow method is the best way to find the number of clusters. The elbow method constitutes running K-Means clustering on the dataset.

- Next, we use within-sum-of-squares as a measure to find the optimum number of clusters that can be formed for a given data set. Within-the-sum of squares (WSS) is defined as the sum of the squared distance between each member of the cluster and its centroid.

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

Where x_i = data point and c_i = closest point to centroid

The WSS is measured for each value of K. The value of K, which has the least amount of WSS, is taken as the optimum value.

How Does K-Means Clustering Work?
Step 1

- The Elbow method is the best way to find the number of clusters. The elbow method constitutes running K-Means clustering on the dataset.

- Next, we use within-sum-of-squares as a measure to find the optimum number of clusters that can be formed for a given data set. **Within the sum of squares** (WSS) is defined as the sum of the squared distance between each member of the cluster and its centroid.

$$WSS = \sum_{i=1}^{m} (x_i - c_i)^2$$

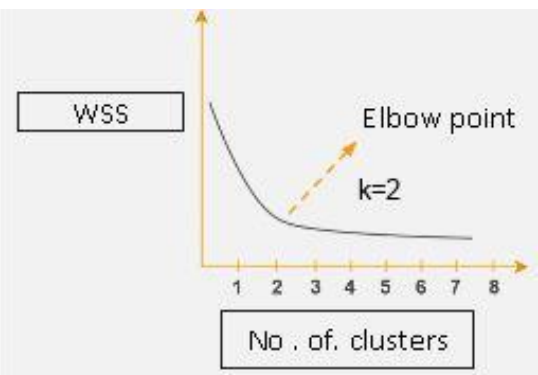
Where x_i = data point and c_i = closest point to centroid

The WSS is measured for each value of K. The value of K, which has the least amount of WSS, is taken as the optimum value.

How Does K-Means Clustering Work?

Step 1

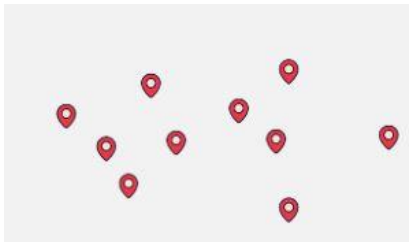
- WSS is on the y-axis and number of clusters on the x-axis.
- You can see that there is a very gradual change in the value of WSS as the K value increases from 2.
- So, you can take the elbow point value as the optimal value of K. It should be either two, three, or at most four. But, beyond that, increasing the number of clusters does not dramatically change the value in WSS, it gets stabilized.

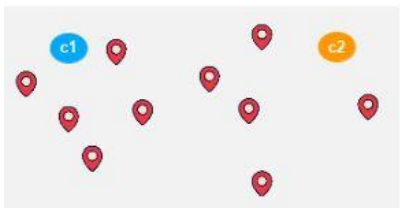


How Does K-Means Clustering Work?

Step 2

- Let's assume that these are our delivery points:





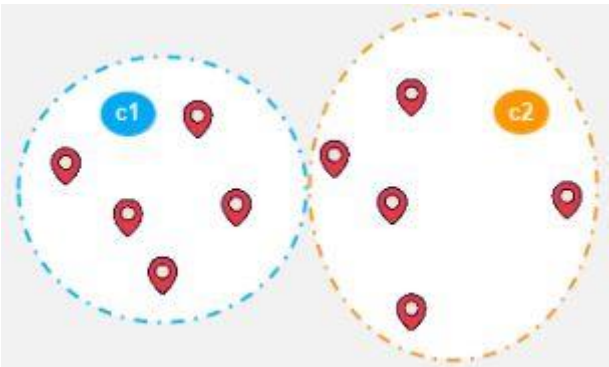
We can randomly initialize two points called the cluster centroids.

Here, C1 and C2 are the centroids assigned randomly.

How Does K-Means Clustering Work?

Step 3

- Now the distance of each location from the centroid is measured, and each data point is assigned to the centroid, which is closest to it.
- This is how the initial grouping is done:



How Does K-Means Clustering Work?

Step 4

Compute the actual centroid of data points for the first group.

Step 5:

Reposition the random centroid to the actual centroid.

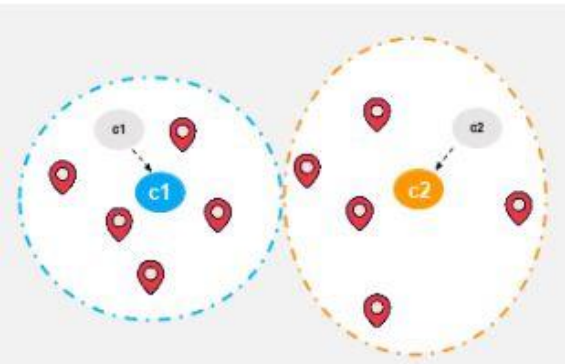
How Does K-Means Clustering Work?

Step 6

Compute the actual centroid of data points for the second group.

Step 7:

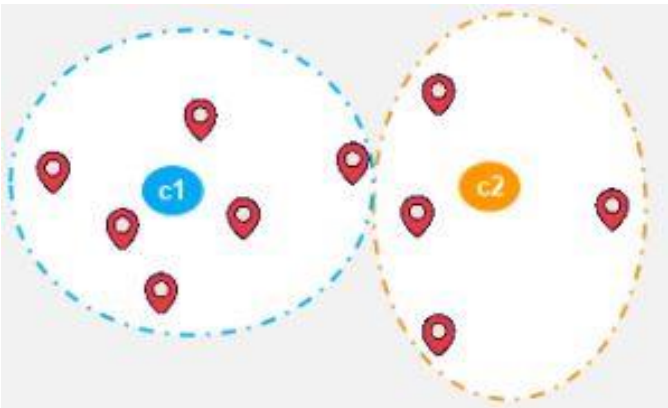
Reposition the random centroid to the actual centroid.



How Does K-Means Clustering Work?

Step 8

- Once the cluster becomes static, the k-means algorithm is said to be converged.
- The final cluster with centroids c1 and c2 is as shown below:



K-Means Clustering Algorithm

Let's say we have x1, x2, x3..... x(n) as our inputs, and we want to split this into K clusters.

The steps to form clusters are:

- Step 1: Choose K random points as cluster centers called centroids.
- Step 2: Assign each x(i) to the closest cluster by implementing euclidean distance (i.e., calculating its distance to each centroid)
- Step 3: Identify new centroids by taking the average of the assigned points.
- Step 4: Keep repeating step 2 and step 3 until convergence is achieved

Illustration of k-Means clustering algorithms

16 Objects with 2 Attributes

A_1	A_2
6.8	12.6
0.8	9.8
1.2	11.6
2.8	9.6
3.8	9.9
4.4	6.5
4.8	1.1
6.0	19.9
6.2	18.5
7.6	17.4
7.8	12.2
6.6	7.7
8.2	4.5
8.4	6.9
9.0	3.4
9.6	11.1

Plotting data of Table

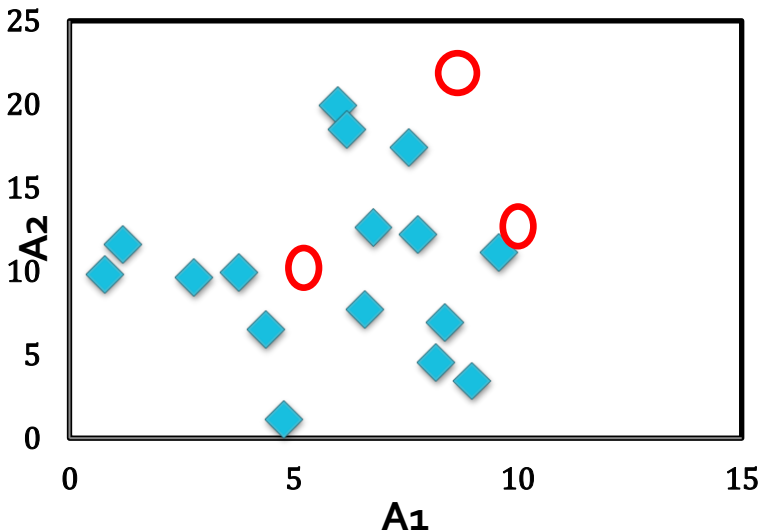


Illustration of k-Means clustering algorithms

- Suppose, k=3. Three objects are chosen at random shown as circled (see Fig). These three centroids are shown below.

Centroid	Objects	
	A1	A2
c_1	3.8	9.9
c_2	7.8	12.2
c_3	6.2	18.5

- Let us consider the Euclidean distance measure (L_2 Norm) as the distance measurement in our illustration.
- Let d_1, d_2 and d_3 denote the distance from an object to c_1, c_2 and c_3 respectively. The distance calculations are shown in Table.
- Assignment of each object to the respective centroid is shown in the right-most column and the clustering so obtained is shown.

Illustration of k-Means clustering algorithms

Distance

A_1	A_2	d_1	d_2	d_3	cluster
6.8	12.6	4.0	1.1	5.9	2
0.8	9.8	3.0	7.4	10.2	1
1.2	11.6	3.1	6.6	8.5	1
2.8	9.6	1.0	5.6	9.5	1
3.8	9.9	0.0	4.6	8.9	1
4.4	6.5	3.5	6.6	12.1	1
4.8	1.1	8.9	11.5	17.5	1
6.0	19.9	10.2	7.9	1.4	3
6.2	18.5	8.9	6.5	0.0	3
7.6	17.4	8.4	5.2	1.8	3
7.8	12.2	4.6	0.0	6.5	2
6.6	7.7	3.6	4.7	10.8	1
8.2	4.5	7.0	7.7	14.1	1
8.4	6.9	5.5	5.3	11.8	2
9.0	3.4	8.3	8.9	15.4	1
9.6	11.1	5.9	2.1	8.1	2

Initial cluster

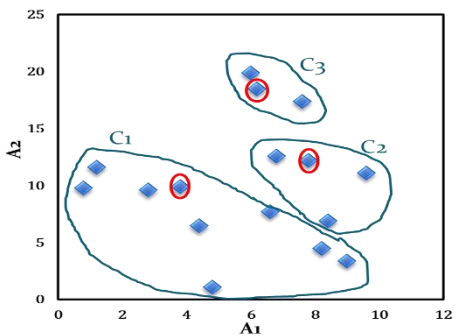
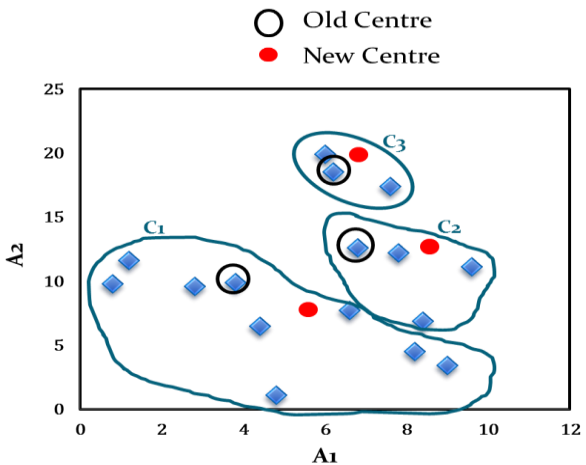


Illustration of k-Means clustering algorithms

The calculation new centroids of the three cluster using the mean of attribute values of A_1 and A_2 is shown in the Table below. The cluster with new centroids are shown in Fig.

Calculation of new centroids

New Centroid	Objects	
	A1	A2
c_1	4.6	7.1
c_2	8.2	10.7
c_3	6.6	18.6

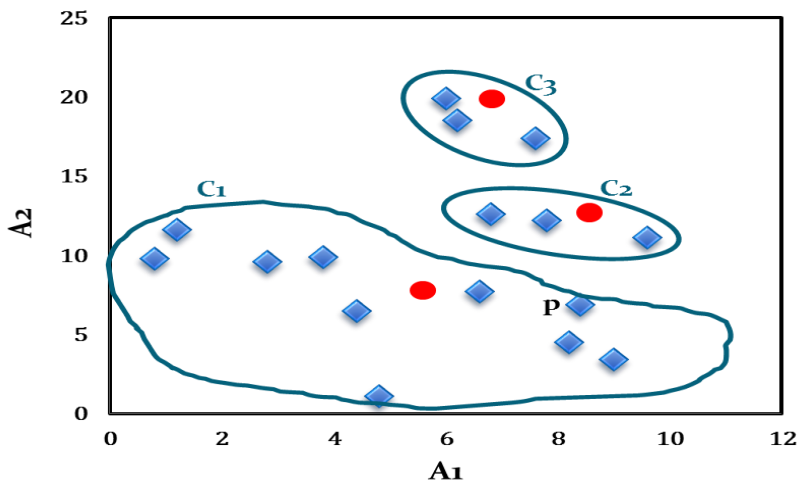


Initial cluster with new centroids

Illustration of k-Means clustering algorithms

We next reassign the 16 objects to three clusters by determining which centroid is closest to each one. This gives the revised set of clusters shown in Fig.

Note that point p moves from cluster C_2 to cluster C_1.



Cluster after first iteration

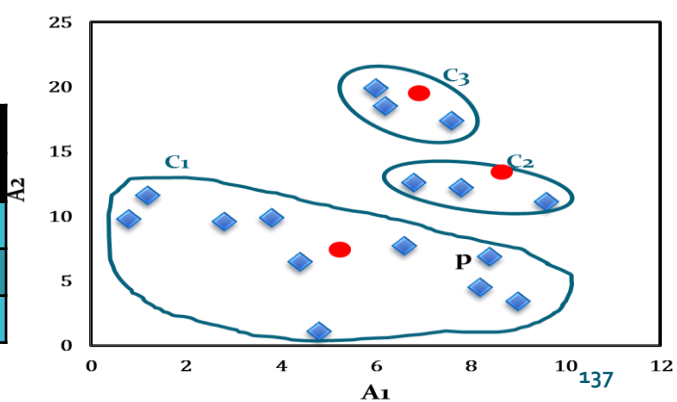
Illustration of k-Means clustering algorithms

- The newly obtained centroids after second iteration are given in the table below. Note that the centroid c_3 remains unchanged, where c_2 and c_1 changed a little.
- With respect to newly obtained cluster centres, 16 points are reassigned again. These are the same clusters as before. Hence, their centroids also remain unchanged.
- Considering this as the termination criteria, the k-means algorithm stops here. Hence, the final cluster in earlier Fig is same as Fig below.

Cluster centres after second iteration

Centroid	Revised Centroids	
	A1	A2
c_1	5.0	7.1
c_2	8.1	12.0
c_3	6.6	18.6

Cluster after Second iteration



Comments on k-Means algorithm

1. **Value of k:**
   - The k-means algorithm produces only one set of clusters, for which, user must specify the desired number, k of clusters.
   - In fact, k should be the best guess on the number of clusters present in the given data. Choosing the best value of k for a given dataset is, therefore, an issue.
   - We may not have an idea about the possible number of clusters for high dimensional data, and for data that are not scatter-plotted.
   - Further, possible number of clusters is hidden or ambiguous in image, audio, video and multimedia clustering applications etc.
   - There is no principled way to know what the value of k ought to be. We may try with successive value of k starting with 2.
   - The process is stopped when two consecutive k values produce more-or-less identical results (with respect to some cluster quality estimation).
   - Normally $k \ll n$ and there is heuristic to follow $k \approx \sqrt{n}$.

Comments on k-Means algorithm

K- Versus Cluster Quality

- Usually, there is some objective function to be met as a goal of clustering. One such objective function is sum-square-error denoted by SSE and defined as

$$SSE = \sum_{i=1}^{k} \sum_{x \in C_i} (x - c_i)^2$$

- Here, $x - c_i$ denotes the error, if x is in cluster C_i with cluster centroid c_i.
- Usually, this error is measured as distance norms like L_1, L_2, L_3 or Cosine similarity, etc.

Comments on k-Means algorithm

Example : k versus cluster quality

- With reference to an arbitrary experiment, suppose the following results are obtained.

k	SSE
1	62.8
2	12.3
3	9.4
4	9.3
5	9.2
6	9.1
7	9.05
8	9.0

- With respect to this observation, we can choose the value of $k \approx 3$, as with this smallest value of k it gives reasonably good result.
- Note: If $k = n$, then SSE=0; However, the cluster is useless! This is another example of overfitting.

Cluster Performance Metrics

Silhouette Score

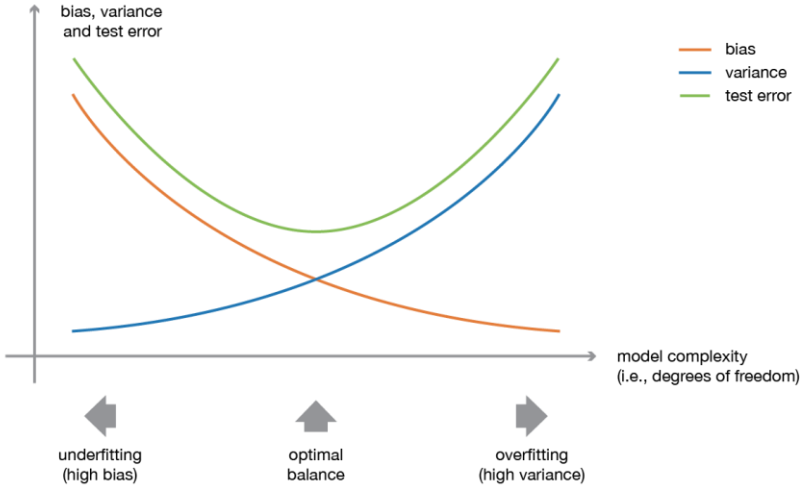
- A popular statistic for assessing the homogeneity and separation of clusters is the silhouette score. An average silhouette coefficient is calculated for each sample, and defined as the difference between the average intercluster and the nearest cluster distance, normalized by the greater of these two distances. **A higher silhouette score indicates a different cluster and well separated, indicating that the clusters are adequate.** Similarity between data points in one group and data points in other groups is measured using silhouette scores.

- The mathematical Formula for the Silhouette Score is

$$S = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

Here, (a(i)) represents the average distance from data point (i) to other points in the same cluster and (b(i)) is the smallest average distance from data point (i) to points in a different cluster.

bias-variance tradeoff?



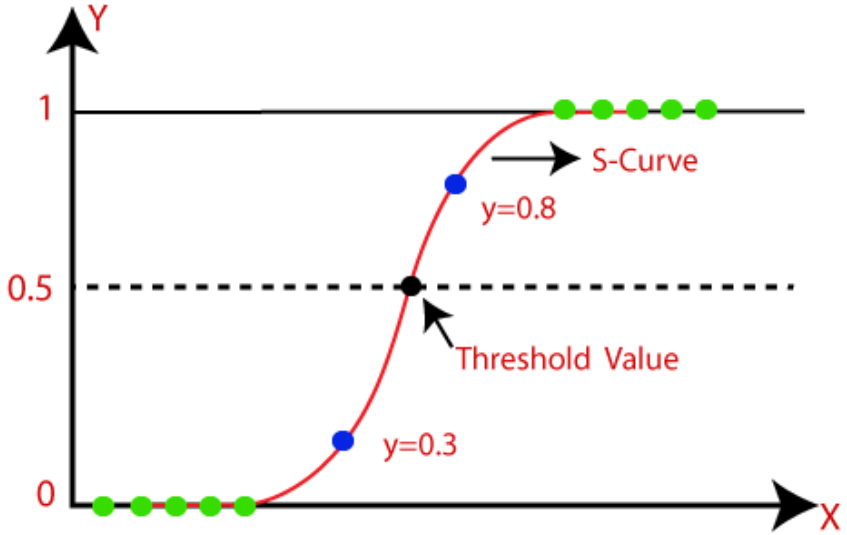
- A low bias and a low variance, although they most often vary in opposite directions, are the two most fundamental features expected for a model. Indeed, to be able to “solve” a problem, we want our model to have enough degrees of freedom to resolve the underlying complexity of the data we are working with, but we also want it to have not too much degrees of freedom to avoid high variance and be more robust. This is the well known **bias-variance tradeoff**.

K-nearest neighbours (kNN) Algorithm

k-Nearest Neighbors (kNN) is considered a **lazy learning** algorithm because it doesn't build an explicit model during the training phase. Here's why:

1. **No training phase (or very little)**: In kNN, the training phase consists of simply storing the training data. There's no real computation or model-building done at this stage, unlike other algorithms like decision trees, SVMs, or neural networks, which perform extensive calculations during training to create a model.
2. **Computation during prediction**: The main computational effort happens during the prediction phase. When you want to classify a new data point, the algorithm looks through the entire training dataset to find the "k" nearest neighbors (based on some distance metric, like Euclidean distance). This makes kNN a "lazy" algorithm because it defers the computation until the prediction is actually needed.
3. **Instance-based learning**: Lazy learning is also called **instance-based learning**, meaning that the algorithm stores all the training data and uses it directly to make predictions at the time of query. It doesn't generalize a model in advance, which contrasts with **eager learning** algorithms, where a model is built before making predictions.

Logistic Function (Sigmoid Function)



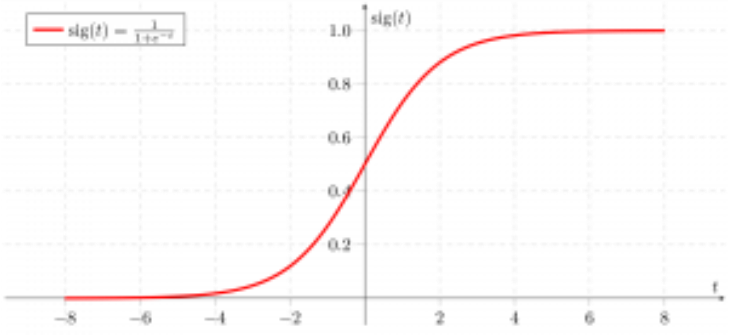
$$f(x) = \frac{1}{1 + e^{-x}}$$

Logistic regression uses the concept of predictive modeling as regression; therefore, it is called logistic regression, but is used to classify samples; Therefore, it falls under the classification algorithm.

Logistic Function (Sigmoid Function)

As shown above, the figure sigmoid function converts the continuous variable data into the probability i.e. between 0 and 1.

- $\sigma(z)$ tends towards 1 as $z \to \infty$
- $\sigma(z)$ tends towards 0 as $z \to -\infty$
- $\sigma(z)$ is always bounded between 0 and 1



where the probability of being a class can be measured as:

$$P(y = 1) = \sigma(z)$$

$$P(y = 0) = 1 - \sigma(z)$$

Logistic Function (Sigmoid Function)

- The sigmoid function is a mathematical function used to map the predicted values to probabilities.
- It maps any real value into another value within a range of 0 and 1.
- The value of the logistic regression must be between 0 and 1, which cannot go beyond this limit, so it forms a curve like the "S" form. The S-form curve is called the **Sigmoid function** or the logistic function.
- In logistic regression, we use the concept of the threshold value, which defines the probability of either 0 or 1. Such as values above the threshold value tends to 1, and a value below the threshold values tends to 0.

Assumptions for Logistic Regression

- The dependent variable must be categorical in nature.
- The independent variable should not have multi-collinearity.

Logistic Regression Equation

- The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:
  - We know the equation of the straight line can be written as:

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

  - In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$; 0 for y= 0, and infinity for y=1

  - But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

Logistic Regression Equation

- Logistic Regression is a statistical model used for **binary classification** problems, where the output is either **0 or 1** (e.g., spam vs. not spam, cancer vs. no cancer). It is derived from **linear regression** but uses the **sigmoid function** to map predictions to probabilities.

1. Linear Regression and Its Limitation

A standard **linear regression model** predicts an output y as:

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

However, in classification, we need **probabilities** (values between 0 and 1). A linear model can output values beyond this range, which is not ideal for classification tasks.

Thus, we apply a **sigmoid function** to convert the output into a probability.

Logistic Regression Equation

2. Sigmoid (Logistic) Function

To map the output to a probability, we use the **sigmoid function**:

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

where z is the linear combination of inputs:

$$z = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

Thus, the probability that $y = 1$ (i.e., the positive class) is:

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

Similarly, the probability that $y = 0$ (negative class) is:

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - \sigma(z)$$

This gives a model that predicts a **probability** between **0 and 1** for classification.

Logistic Regression Equation

3. Log-Odds and Logistic Transformation

The logistic function can be rewritten in terms of **odds** (the ratio of the probability of success to failure):

$$\frac{P(y=1|x)}{P(y=0|x)} = e^z$$

Taking the **logarithm** on both sides gives the **log-odds (logit function)**:

$$\log\left(\frac{P(y=1|x)}{P(y=0|x)}\right) = z = w_0 + w_1x_1 + \cdots + w_nx_n$$

This transformation ensures that the probability output is always between **0 and 1**.

Logistic Regression Equation

4. Maximum Likelihood Estimation (MLE)

To find the best parameters w, we use **Maximum Likelihood Estimation (MLE)** instead of least squares (used in linear regression).

For a given dataset (x_i, y_i) **where** $y_i \in \{0, 1\}$, the likelihood function is:

$$L(w) = \prod_{i=1}^{m} P(y_i|x_i)$$

Since y_i can be either 0 or 1, we write:

$$L(w) = \prod_{i=1}^{m} \sigma(z_i)^{y_i}(1 - \sigma(z_i))^{(1-y_i)}$$

Taking the **log-likelihood** (since log transforms products into sums, making it easier to differentiate):

$$\log L(w) = \sum_{i=1}^{m} [y_i \log \sigma(z_i) + (1 - y_i) \log(1 - \sigma(z_i))]$$

This is the **log-likelihood function**, which we aim to **maximize** to find the best parameters w.

Logistic Regression Equation

6. Final Logistic Regression Model

The final model for binary classification is:

$$P(y=1|x) = \frac{1}{1+e^{-(w_0+w_1x_1+\cdots+w_nx_n)}}$$

During prediction:

- If $P(y=1|x) > 0.5$, classify as **1 (positive class)**
- Otherwise, classify as **0 (negative class)**

- **Logistic Regression** is a classification model derived from linear regression by applying the **sigmoid function**.
- The **log-odds transformation** ensures output is in the range **(0,1)**.
- The **Maximum Likelihood Estimation (MLE)** is used to estimate parameters.
- **Gradient Descent** is used to optimize the parameters.

Logistic Regression: Example

Example: Predicting if a Patient has Diabetes

- Suppose we have data on patients and want to predict whether they have diabetes (1 = Yes, 0 = No) based on their blood sugar level (x).

Step 1: Data (Blood Sugar vs. Diabetes)

Patient	Blood Sugar Level (x)	Diabetes (y)
1	80	0 (No)
2	85	0 (No)
3	90	0 (No)
4	95	0 (No)
5	100	1 (Yes)
6	110	1 (Yes)
7	120	1 (Yes)
8	130	1 (Yes)

We want to build a **logistic regression model** to predict y (diabetes status) given x (blood sugar level).

Step 2: Define the Logistic Regression Model

The probability of having diabetes is modeled as:

$$P(y=1|x) = \frac{1}{1+e^{-(w_0+w_1x)}}$$

where:

- w_0 (intercept) and w_1 (coefficient) are parameters that we need to learn.
- x is the blood sugar level.

Logistic Regression: Example

Step 3: Convert to Log-Odds

To simplify, we take the **log-odds (logit function)**:

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right) = w_0 + w_1 x$$

This equation is **linear** in terms of the log-odds.

Step 4: Train the Model

Using **Maximum Likelihood Estimation (MLE)**, we find the best parameters w_0 and w_1 using the given data. Assume we get:

$$w_0 = -15, \quad w_1 = 0.15$$

So, the model becomes:

$$P(y=1|x) = \frac{1}{1+e^{-(-15+0.15x)}}$$

Logistic Regression: Example

Step 5: Make Predictions

Now, let's use this model to **predict** if a new patient with a given blood sugar level has diabetes.

Case 1: Patient with Blood Sugar = 105

$$P(y=1|x=105) = \frac{1}{1+e^{-(-15+0.15\times 105)}}$$

$$= \frac{1}{1+e^{-0.75}} \approx \frac{1}{1+0.472} \approx 0.68$$

Since $P = 0.68 > 0.5$, we classify **this patient as diabetic (1)**.

Case 2: Patient with Blood Sugar = 90

$$P(y=1|x=90) = \frac{1}{1+e^{-(-15+0.15\times 90)}}$$

$$= \frac{1}{1+e^{-1.5}} \approx \frac{1}{1+0.223} \approx 0.82$$

Since $P = 0.82 > 0.5$, we classify **this patient as diabetic (1)**.

Step 6: Interpretation

- As **blood sugar level increases**, the probability of having diabetes **increases**.
- The logistic function ensures predictions remain between **0 and 1**.
- We use **0.5 as the decision boundary**: If $P(y=1) > 0.5$, classify as **1 (diabetic)**; otherwise, classify as **0 (not diabetic)**.

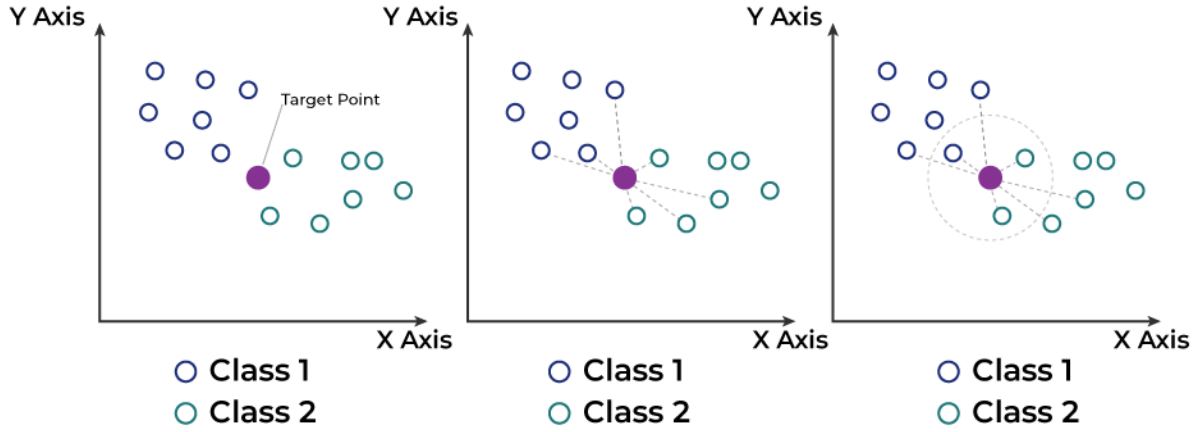
Type of Logistic Regression:

On the basis of the categories, Logistic Regression can be classified into three types:

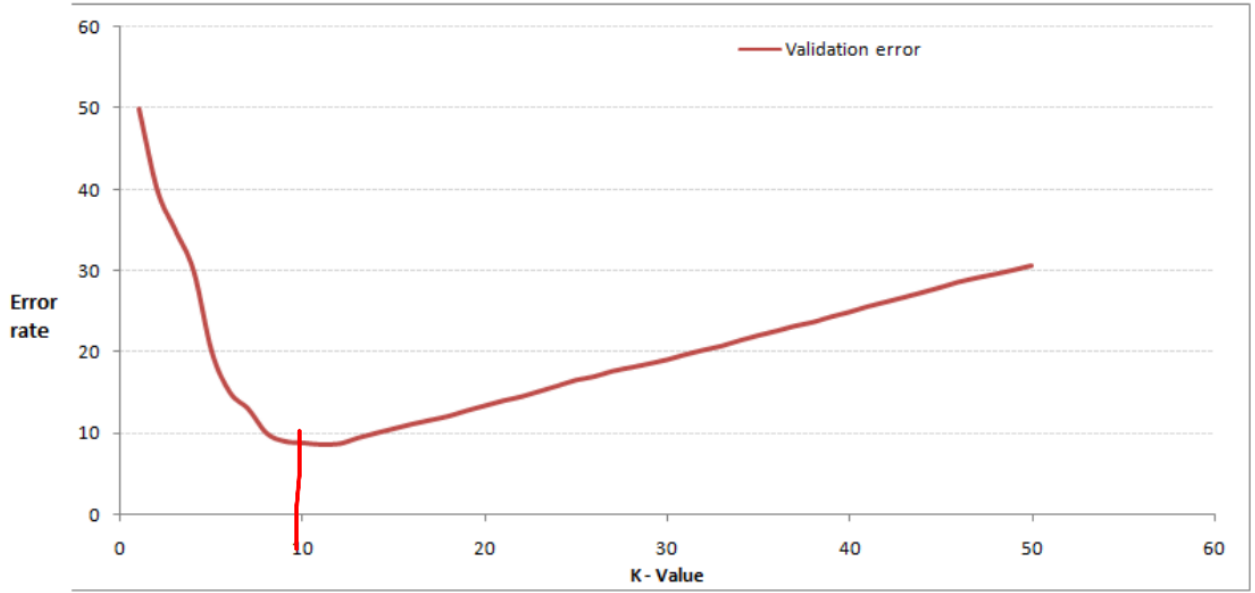
- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Working of kNN algorithm

- The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Choosing appropriate k



Choosing appropriate k

- In practice, choosing k depends on the difficulty of the concept to be learned and the number of records in the training data.
- Typically, k is set somewhere between 3 and 10. One common practice is to set k equal to the square root of the number of training examples.
- In the classifier, we might set k = 4, because there were 15 example ingredients in the training data and the square root of 15 is 3.87.

Rule of thumb is K < sqrt(n), n is number of examples.

Working of KNN algorithm

Step-by-Step explanation of how KNN works is discussed below:

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

- To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

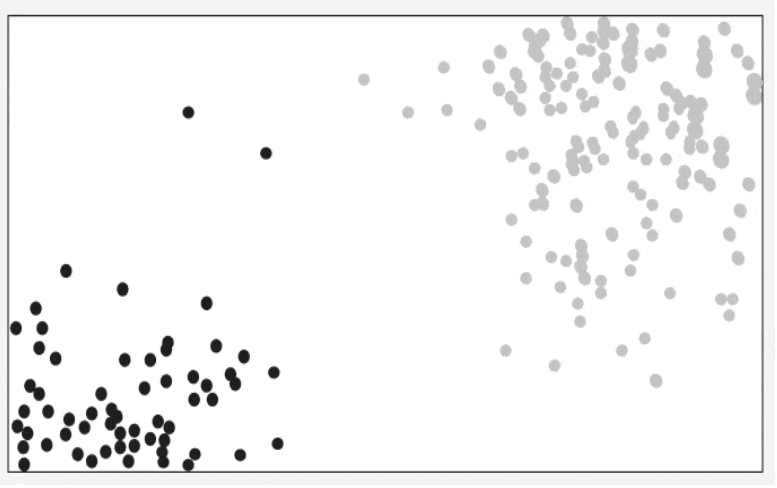
- The k data points with the smallest distances to the target point are nearest neighbors.

Working of KNN algorithm

Step 4: Voting for Classification or Taking Average for Regression

- When we want to classify a data point into a category (like spam or not spam), the K-NN algorithm looks at the **K closest points** in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called **majority voting**.

- In regression, the algorithm still looks for the **K closest points**. But instead of voting for a class in classification, it takes the **average** of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.



Examples on KNN algorithm

- **Example -2:** You receive an email and want to determine whether it is **Spam** or **Not Spam** based on past emails.

Step 1: Training Data (Labeled Emails)

We have a dataset where each email is labeled as **Spam (1) or Not Spam (0)** based on word frequency.

Email ID	Contains "Free"	Contains "Win"	Word Count	Spam (1) or Not Spam (0)
1	Yes	Yes	50	1 (Spam)
2	No	Yes	20	0 (Not Spam)
3	Yes	No	100	0 (Not Spam)
4	Yes	Yes	60	1 (Spam)

Step 2: New Email to Classify

A new email contains **"Free"**, **"Win"**, and has **55 words**. Is it spam?

Step 3: Compute Distance (Using Euclidean Distance)

We compare this email to the existing dataset using the features **(Contains "Free", Contains "Win", Word Count)**:

- **Email 1 (Spam)** → Distance = $\sqrt{((1-1)^2 + (1-1)^2 + (55-50)^2)} = \mathbf{5}$
- **Email 2 (Not Spam)** → Distance = $\sqrt{((1-0)^2 + (1-1)^2 + (55-20)^2)} = \mathbf{35.02}$
- **Email 3 (Not Spam)** → Distance = $\sqrt{((1-1)^2 + (1-0)^2 + (55-100)^2)} = \mathbf{45.01}$
- **Email 4 (Spam)** → Distance = $\sqrt{((1-1)^2 + (1-1)^2 + (55-60)^2)} = \mathbf{5}$

Step 4: Select k Nearest Neighbors

For **k = 3**, the closest emails are:

1. **Email 1 (Spam, Distance = 5)**
2. **Email 4 (Spam, Distance = 5)**
3. **Email 2 (Not Spam, Distance = 35.02)**

Step 5: Majority Voting

Since **2 out of 3 neighbors are Spam**, the **new email is classified as Spam.**

Using **k-NN**, we predict that the new email is **Spam** based on the similarity to previous emails.

k-Nearest Neighbors (k-NN) Regression

- **Example -3:** You want to predict the price of a new house based on its **size (square feet)** using past housing data.

Step 1: Training Data (Labeled Data)

House ID	Size (sq. ft.)	Price ($1000s)
1	1000	150
2	1200	180
3	1500	220
4	1800	260
5	2000	300

Step 2: New House to Predict

A new house has **size = 1600 sq. ft.**. What is its predicted price?

Step 3: Compute Distance (Using Euclidean Distance)

We compare the new house's size with the existing houses.

- Distance to **House 1 (1000 sq. ft.)** = |1600 - 1000| = **600**
- Distance to **House 2 (1200 sq. ft.)** = |1600 - 1200| = **400**
- Distance to **House 3 (1500 sq. ft.)** = |1600 - 1500| = **100**
- Distance to **House 4 (1800 sq. ft.)** = |1600 - 1800| = **200**
- Distance to **House 5 (2000 sq. ft.)** = |1600 - 2000| = **400**

Step 4: Select k Nearest Neighbors

For **k = 3**, the closest houses are:

1. **House 3 (1500 sq. ft., $220k)** → Distance **100**
2. **House 4 (1800 sq. ft., $260k)** → Distance **200**
3. **House 2 (1200 sq. ft., $180k)** → Distance **400**

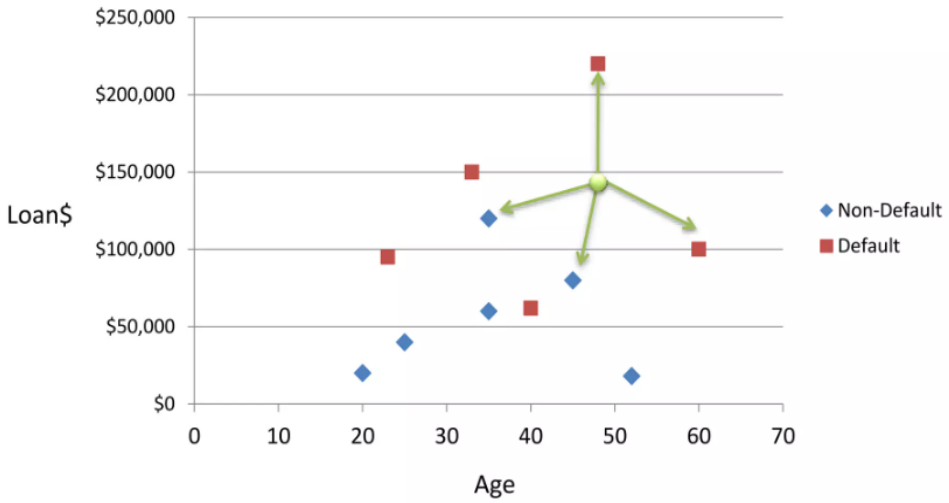
Step 5: Compute Average Price (Regression Output)

The predicted price is the **average** of the k nearest neighbors' prices:

$$\text{Predicted Price} = \frac{220 + 260 + 180}{3} = \frac{660}{3} = 220$$

k-NN Regression, we predict that the **new house (1600 sq. ft.) will cost around $220,000**.

K-NN: Example


$250,000
$200,000
$150,000
$100,000
$50,000
$0
Loan$
0
10
20
30
40
50
60
70
Age
Non-Default
Default

K-NN: Example

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

Euclidean Distance

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

Disadvantages of the KNN Algorithm

- **Does not scale** – As we have heard about this that the KNN algorithm is also considered a Lazy Algorithm. The main significance of this term is that this takes lots of computing power as well as data storage. This makes this algorithm both time-consuming and resource exhausting.

- **Curse of Dimensionality** – There is a term known as the peaking phenomenon according to this the KNN algorithm is affected by the curse of dimensionality which implies the algorithm faces a hard time classifying the data points properly when the dimensionality is too high.

- **Prone to Overfitting** – As the algorithm is affected due to the curse of dimensionality it is prone to the problem of overfitting as well. Hence generally feature selection as well as dimensionality reduction techniques are applied to deal with this problem.

Bayes Theorem: Unveiling Probabilistic Reasoning

- Bayes theorem revolutionizes how we update beliefs based on new evidence. It's a cornerstone of probabilistic reasoning, widely applied in science, technology, and decision-making.
- Naive Bayes classifiers are a collection of classification algorithms based on **Bayes' Theorem**. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

The Essence of Bayes Theorem

Bayes' Theorem is a fundamental principle in probability theory that describes how to update the probability of a hypothesis based on new evidence. The theorem is expressed mathematically as:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

Where:

- $P(H|E)$ is the **posterior probability**: the probability of hypothesis H given evidence E.
- $P(E|H)$ is the **likelihood**: the probability of observing evidence E given that H is true.
- $P(H)$ is the **prior probability**: the initial probability of hypothesis H before observing evidence E.
- $P(E)$ is the **marginal likelihood**: the total probability of observing evidence E under all possible hypotheses.

Example: Medical Diagnosis

Scenario: Determine the probability that a person has a disease (Disease A) given a positive test result.

1. **Define the probabilities:**
   - Let H: the event that the person has Disease A.
   - Let E: the event that the person tests positive for Disease A.
   - Assume:
     - $P(H) = 0.01$ (1% of the population has the disease).
     - $P(E|H) = 0.9$ (90% probability of testing positive if they have the disease).
     - $P(E|\neg H) = 0.05$ (5% probability of testing positive if they do not have the disease).
2. **Calculate $P(E)$:**

$$P(E) = P(E|H) \cdot P(H) + P(E|\neg H) \cdot P(\neg H)$$

   Where $P(\neg H) = 1 - P(H) = 0.99$:

$$P(E) = (0.9 \cdot 0.01) + (0.05 \cdot 0.99) = 0.009 + 0.0495 = 0.0585$$

3. **Apply Bayes' Theorem:**

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)} = \frac{0.9 \cdot 0.01}{0.0585} \approx 0.154$$

Naive Bayes Classifiers

The dataset is divided into two parts, namely, **feature matrix** and the **response vector**.

- Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of **dependent features**. In above dataset, features are ‘Outlook’, ‘Temperature’, ‘Humidity’ and ‘Windy’.

- Response vector contains the value of **class variable** (prediction or output) for each row of feature matrix. In above dataset, the class variable name is ‘Play golf’.

Assumptions

The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

Note: The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Assumptions

With relation to our dataset, this concept can be understood as:

- We assume that no pair of features are dependent. For example, the temperature being ‘Hot’ has nothing to do with the humidity or the outlook being ‘Rainy’ has no effect on the winds. Hence, the features are assumed to be **independent**.

- Secondly, each feature is given the same weight (or importance). For example, knowing only temperature and humidity alone can’t predict the outcome accurately. None of the attributes is irrelevant and assumed to be contributing **equally** to the outcome.

Naive Bayes Classifiers

The dataset is divided into two parts, namely, **feature matrix** and the **response vector**.

•Feature matrix contains all the vectors(rows) of dataset in which each vector consists of the value of **dependent features**. In above dataset, features are ‘Outlook’, ‘Temperature’, ‘Humidity’ and ‘Windy’.

•Response vector contains the value of **class variable**(prediction or output) for each row of feature matrix. In above dataset, the class variable name is ‘Play golf’.

Index	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

Naive Bayes Classifiers

- Now, with regards to our dataset, we can apply Bayes' theorem in following way:

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

- where, y is class variable and X is a dependent feature vector (of size *n*) where:

$$X = (x_1, x_2, x_3, ..., x_n)$$

- Just to clear, an example of a feature vector and corresponding class variable can be: (refer 1st row of dataset)

```
X = (Rainy, Hot, High, False)
y = No
```

So basically, P(y|X) here means, the probability of "Not playing golf" given that the weather conditions are "Rainy outlook", "Temperature is hot", "high humidity" and "no wind".

Naive Bayes Classifiers

- Now, its time to put a naive assumption to the Bayes' theorem, which is, **independence** among the features. So now, we split **evidence** into the independent parts. Hence, we reach to the result:

$$P(y|x_1, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)}$$

- Which can be expressed as

$$P(y|x_1, ..., x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1)P(x_2)...P(x_n)}$$

- Now, as the denominator remains constant for a given input, we can remove that term:

$$P(y|x_1, ..., x_n) \propto P(y)\prod_{i=1}^{n} P(x_i|y)$$

Naive assumption: in example

Outlook

	Yes	No	P(yes)	P(no)
Sunny	2	3	2/9	3/5
Overcast	4	0	4/9	0/5
Rainy	3	2	3/9	2/5
Total	9	5	100%	100%

Temperature

	Yes	No	P(yes)	P(no)
Hot	2	2	2/9	2/5
Mild	4	2	4/9	2/5
Cool	3	1	3/9	1/5
Total	9	5	100%	100%

Humidity

	Yes	No	P(yes)	P(no)
High	3	4	3/9	4/5
Normal	6	1	6/9	1/5
Total	9	5	100%	100%

Wind

	Yes	No	P(yes)	P(no)
False	6	2	6/9	2/5
True	3	3	3/9	3/5
Total	9	5	100%	100%

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

So, in the figure above, we have calculated $P(x_i \mid y_j)$ for each x_i in X and y_j in y manually in the tables 1-4. For example, probability of playing golf given that the temperature is cool, i.e P(temp. = cool | play golf = Yes) = 3/9.

Also, we need to find class probabilities (P(y)) which has been calculated in the table 5. For example, P(play golf = Yes) = 9/14.

So now, we are done with our pre-computations and the classifier is ready!

Let us test it on a new set of features (let us call it today):

Naive assumption: in example

- Let us test it on a new set of features (let us call it today):

```
today = (Sunny, Hot, Normal, False)
```

- So, probability of playing golf is given by:

$$P(Yes|today) = \frac{P(SunnyOutlook|Yes)P(HotTemperature|Yes)P(NormalHumidity|Yes)P(NoWind|Yes)P(Yes)}{P(today)}$$

- and probability to not play golf is given by:

$$P(No|today) = \frac{P(SunnyOutlook|No)P(HotTemperature|No)P(NormalHumidity|No)P(NoWind|No)P(No)}{P(today)}$$

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

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

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

Naive assumption: in example

- 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

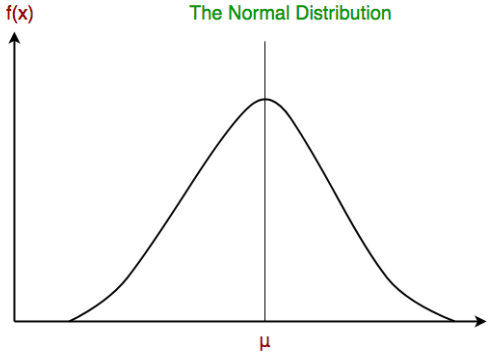
$$P(Yes|today) > P(No|today)$$

So, prediction that golf would be played is ‘Yes’.

The method that we discussed above is applicable for discrete data.

Gaussian Naive Bayes classifier

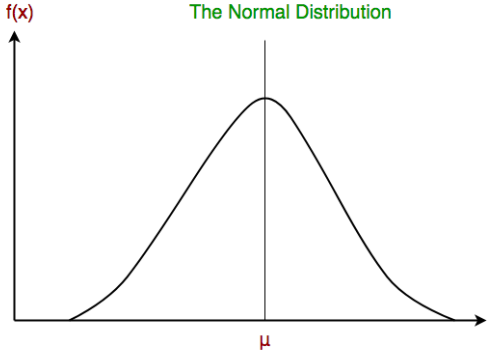
- In case of continuous data, we need to make some assumptions regarding the distribution of values of each feature. The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.
- In Gaussian Naive Bayes, continuous values associated with each feature are assumed to be distributed according to a **Gaussian distribution**. A Gaussian distribution is also called Normal distribution. When plotted, it gives a bell shaped curve which is symmetric about the mean of the feature values as shown below:



Gaussian Naive Bayes classifier

- The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right)$$



Gaussian Naive Bayes classifier

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

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

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

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

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

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

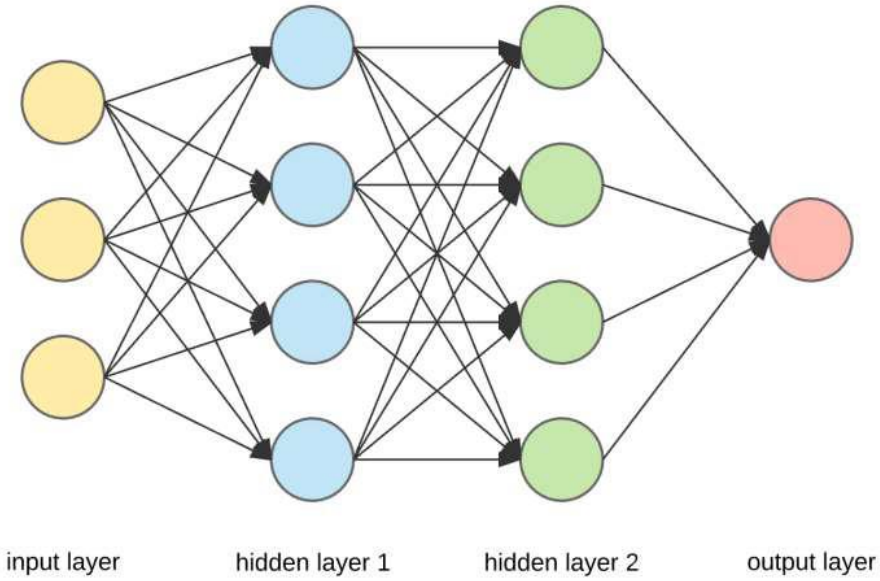
Advantages/Disadvantages of Naïve Bayes Classifier

- Naïve Bayes is one of the fast and easy ML algorithms to predict a class of datasets.
- It can be used for Binary as well as Multi-class Classifications.
- It performs well in Multi-class predictions as compared to the other Algorithms.
- It is the most popular choice for **text classification problems**.

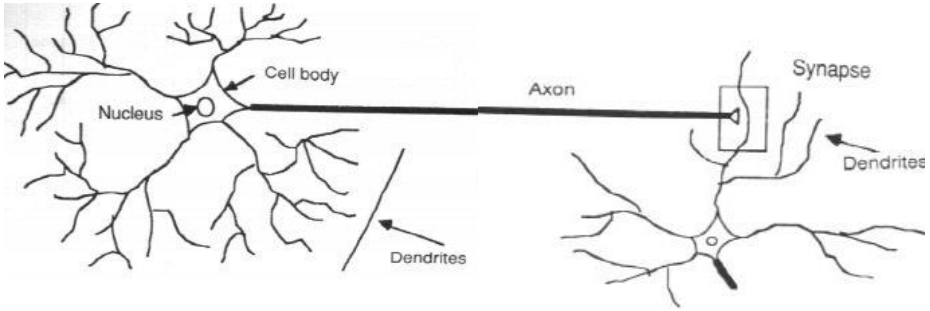
- Naive Bayes assumes that all features are independent or unrelated, so it cannot learn the relationship between features.

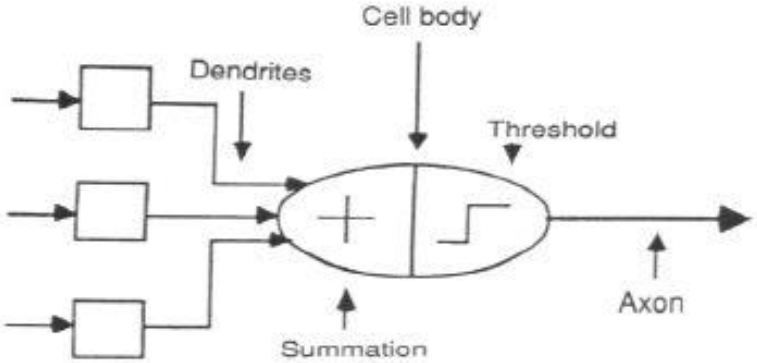
Mathematical Model

- An Artificial Neural Network (ANN) is a network of interconnected artificial neurons.
- Like in a biological neural network, artificial neurons communicate by sending signals to one another.
- Each input to an artificial neuron can either inhibit or excite the artificial neuron.



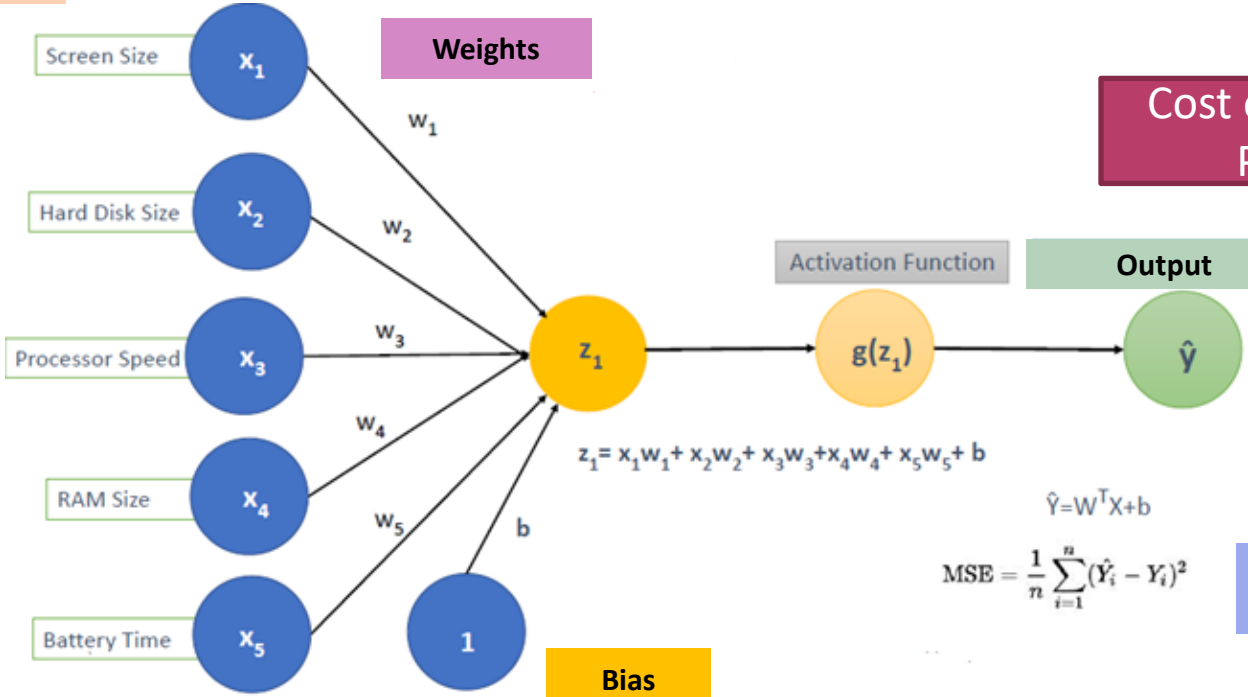
Biological Neuron Vs Artificial





Main Players of ANN

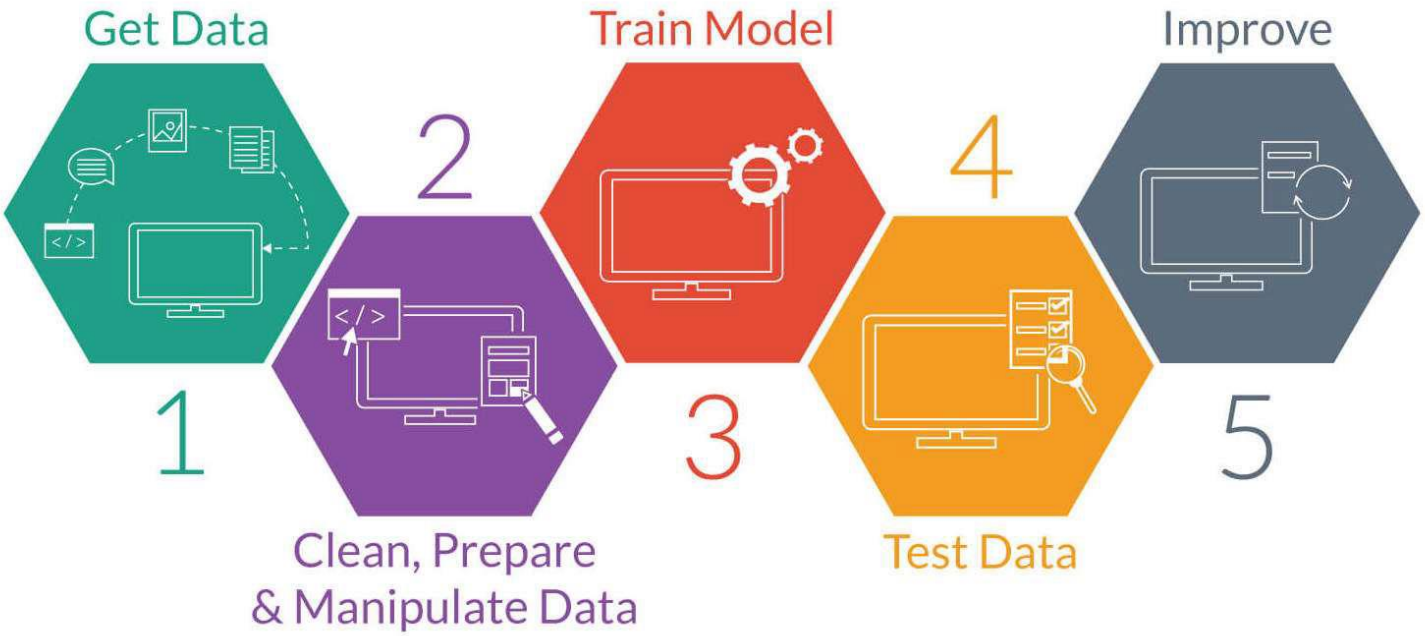
Inputs/Features



Cost of Computer Problem

Comparison with actual output / Error

Steps in Implimentation



Learning Process

Learning means to do and adapt the change in itself as and when there is a change in environment.

Artificial Neural Network Model is built after learning process which is subjected to iterative algorithms.

Learning in the process of ANN model is finding weights (w11,w12.....) and activation function at the cost of minimizing error between actual output and calculated ones.

Features

The features are the elements of your input vectors. The number of features is equal to the number of nodes in the input layer of the network.

Category	Features
Housing Prices	No. of Rooms, House Area, Air Pollution, Distance from facilities, Economic Index city, Security Ranking etc.
Speech Recognition	noise ratios, length of sounds, relative power of sounds, filter matches
Cancer Detection	Clump thickness, Uniformity of cell size, Uniformity of cell shape, Marginal adhesion, Single epithelial cell size, Number of bare nuclei, Bland chromatin, Number of normal nuclei, Mitosis etc.
Video Recommendations	Text matches, Ranking of the video, Interest overlap, history of seen videos, browsing patterns etc
Image Classification	Pixel values, Curves, Edges etc.

Types of Learning Algorithms

Supervised learning	• Learning with a teacher • Learns from examples which provide desired outputs for given inputs
Unsupervised learning	• Learning without a teacher • Learns patterns in input data when no specific output values are given.
Reinforced learning	• Learning with limited feedback • Learns by an indication of correctness at end of some reasoning.

Weights

Weights correspond to each feature. Weights denote how much the feature matters in the model. Higher weight of a particular feature means that it is more important in deciding the outcome of the model.

Weights of a feature represent that how much evidence it gives in favor or against the current hypothesis in context of the existence or non-existence of the pattern you are trying to identify in the current input.

Generally weights are initialized randomly and then with different methods we try to bring them to near optimal values so that they are able to fit the model well and can help in prediction of unseen values.

Activation Function

Their main purpose is to convert a input signal of a node in an ANN to an output signal.

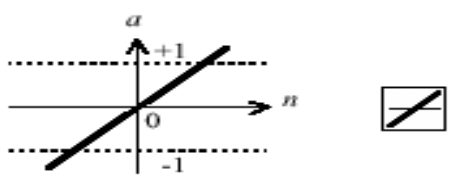
Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it.

Without activation function ANN model is a Linear regression model.

The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

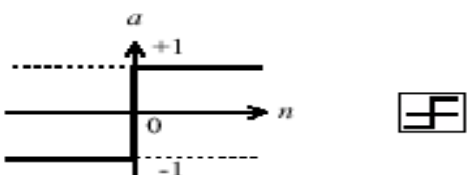
Types of Activation (Squashing) functions

- The activation function describes the output behavior of the neurons Generally is non-linear. Linear functions are limited because the output is simply proportional to the input.



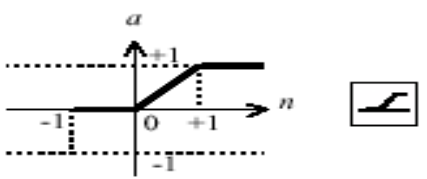
$a = purelin(n)$

Linear Transfer Function



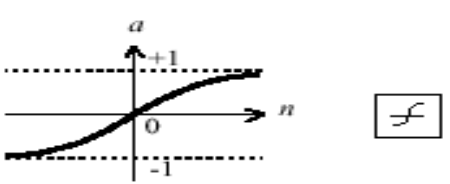
$a = hardlims(n)$

Symmetric Hard Limit Trans. Funct.



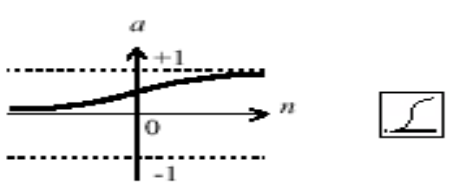
$a = satlin(n)$

Satlin Transfer Function



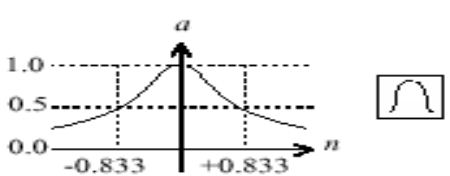
$a = tansig(n)$

Tan-Sigmoid Transfer Function



$a = logsig(n)$

Log-Sigmoid Transfer Function



$a = radbas(n)$

Radial Basis Function

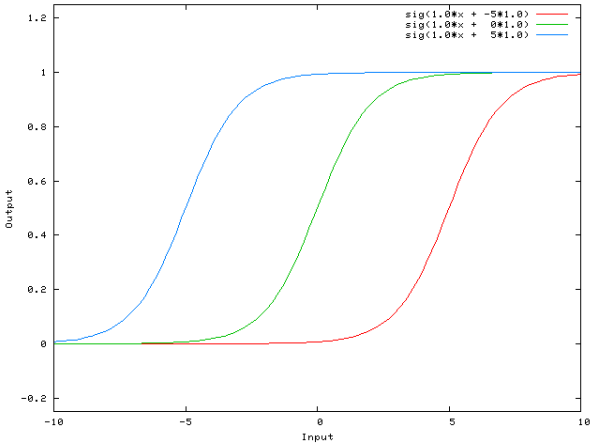
Bias

The bias value allows the activation function to be shifted to the left or right, to better fit the data.

Bias only influences the output values, it doesn't interact with the actual input data.

Biases are tuned alongside weights by learning algorithms such as gradient descent.

Biases differ from weights. They are independent of the output from previous layers.



Squaring the Error difference

The benefits of squaring include:

Squaring always gives a positive value, so the sum will not be zero.

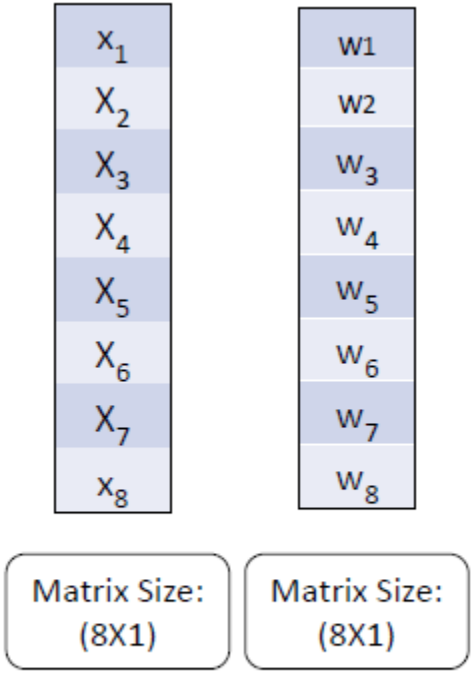
Squaring emphasizes larger differences—a feature that turns out to be both good and bad (think of the effect outliers have).

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

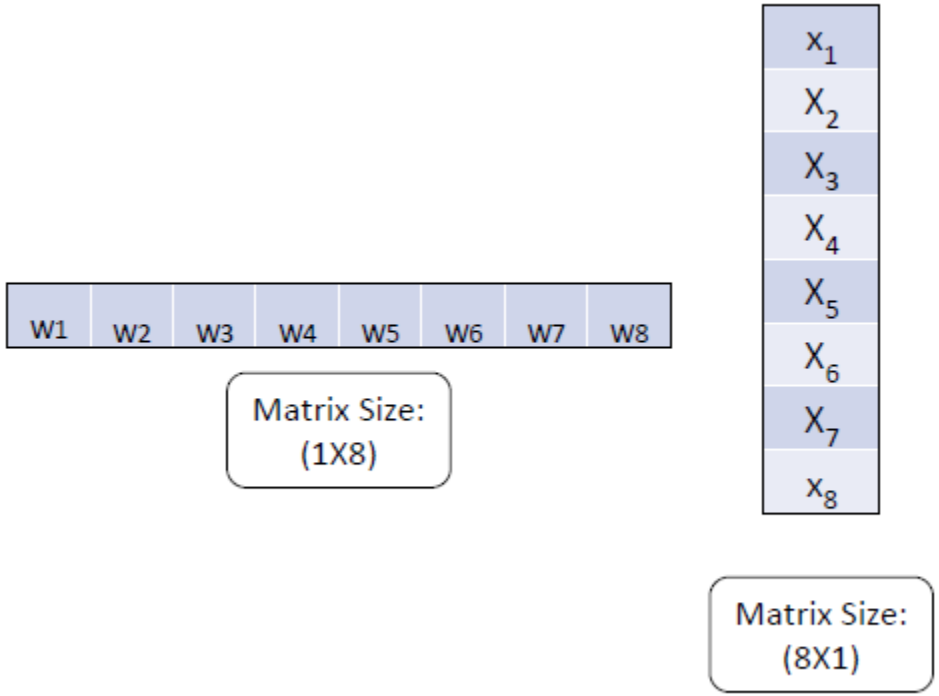
What we calculate in every neuron

z= (wTx + b) and then apply activation function on z which is shown as a(z)

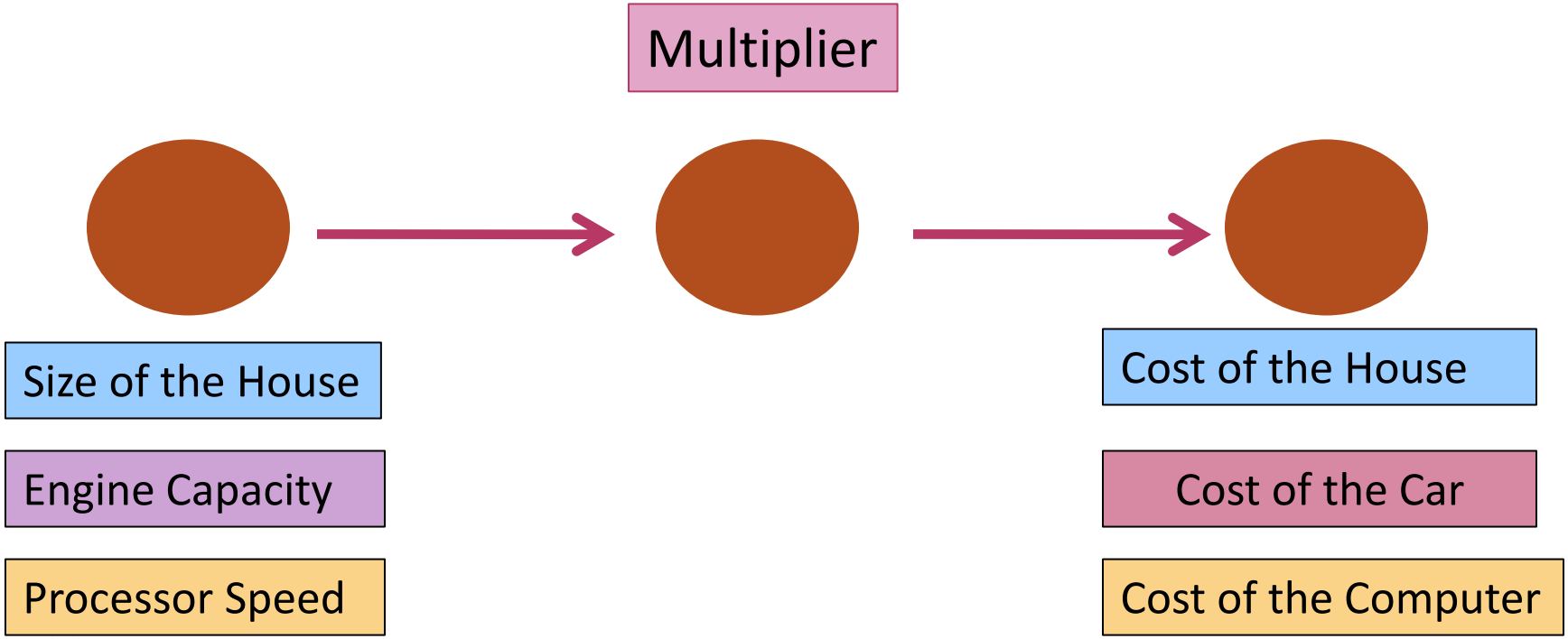
Common Index in both matrices is number of input data points for Single input. We take the transpose of w to make the multiplication work



Multiplication of these two matrices is not possible

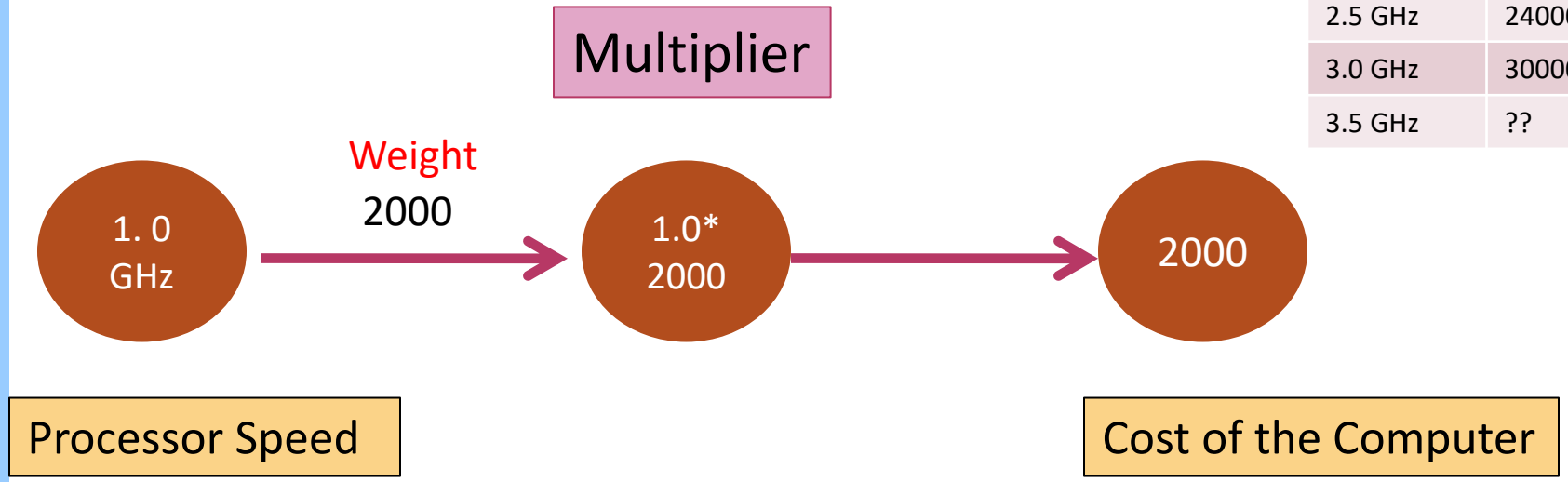


Single Neuron Perceptron



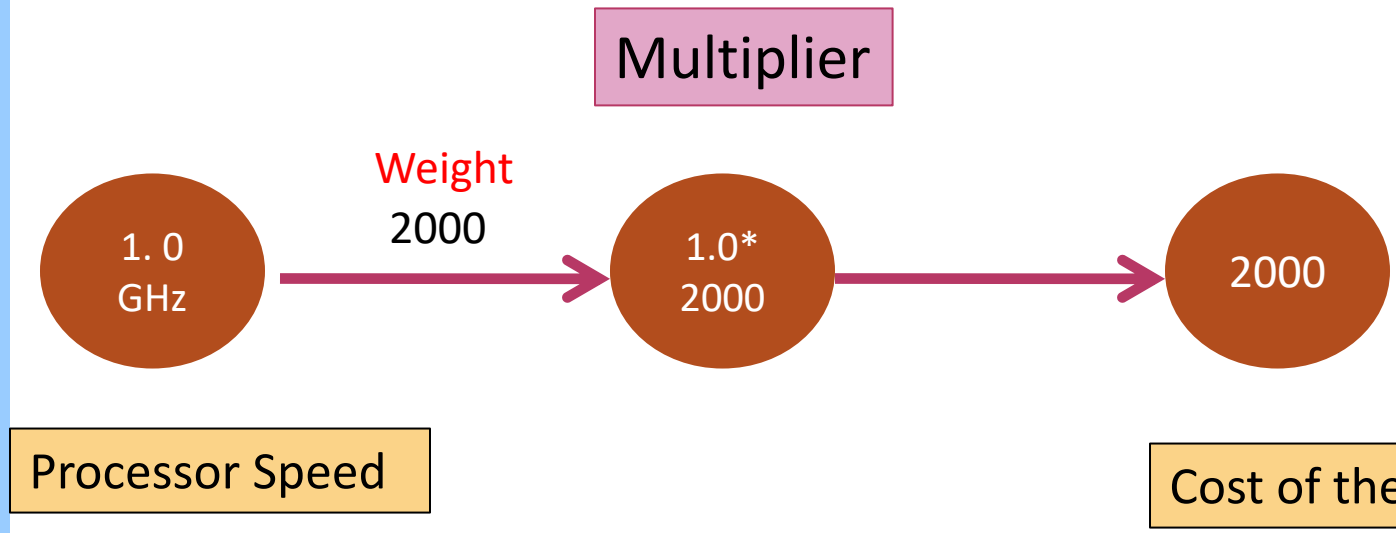
Understanding Single Neuron Perceptron

Speed	Cost
1.0 GHz	8000 INR
1.5 GHz	13000 INR
2.0 GHz	16000 INR
2.5 GHz	24000 INR
3.0 GHz	30000 INR
3.5 GHz	??



Understanding Single Neuron Perceptron

Speed	Cost
1.0 GHz	8000 INR
1.5 GHz	13000 INR
2.0 GHz	16000 INR
2.5 GHz	24000 INR
3.0 GHz	30000 INR
3.5 GHz	??

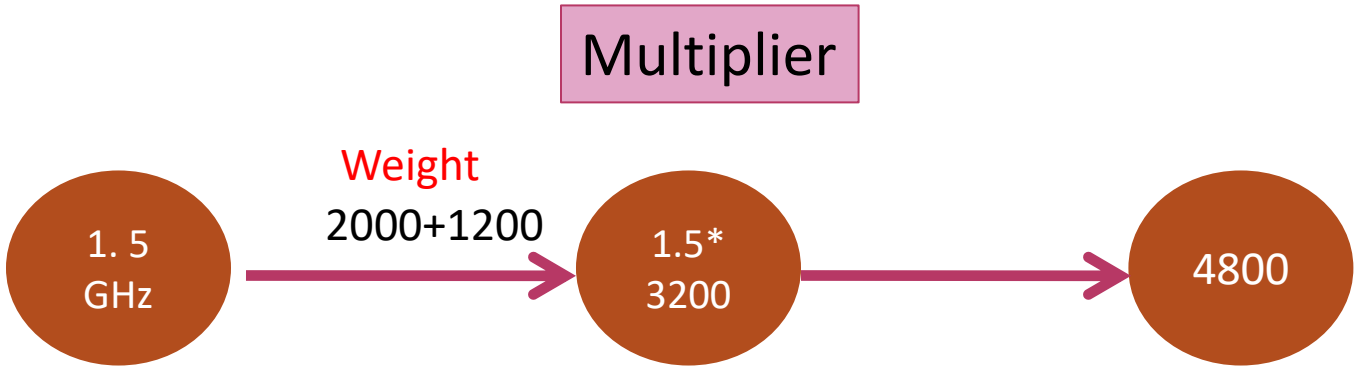


Error= Actual cost-calculated cost = 8000-2000
= 6000

Change in Weight=(Error/5)=1200

Understanding Single Neuron Perceptron

Speed	Cost
1.0 GHz	8000 INR
1.5 GHz	13000 INR
2.0 GHz	16000 INR
2.5 GHz	24000 INR
3.0 GHz	30000 INR
3.5 GHz	??

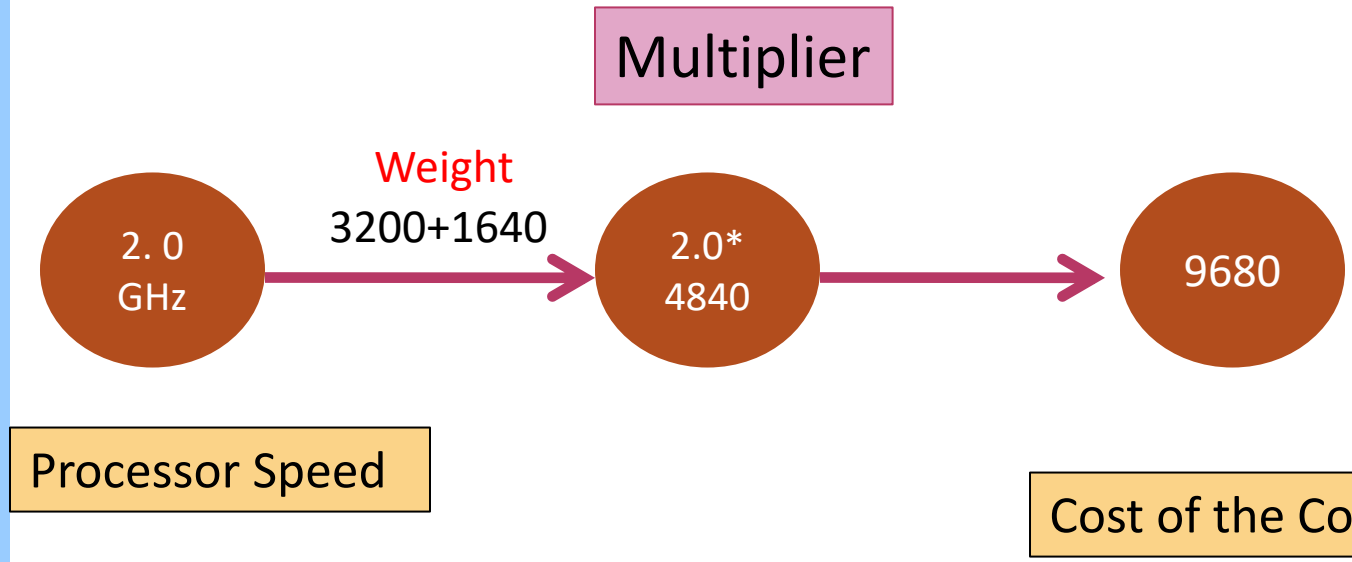


Error= Actual cost-calculated cost = 13000-4800
= 8200

Change in Weight=(Error/5)=1640

Understanding Single Neuron Perceptron

Speed	Cost
1.0 GHz	8000 INR
1.5 GHz	13000 INR
2.0 GHz	16000 INR
2.5 GHz	24000 INR
3.0 GHz	30000 INR
3.5 GHz	??

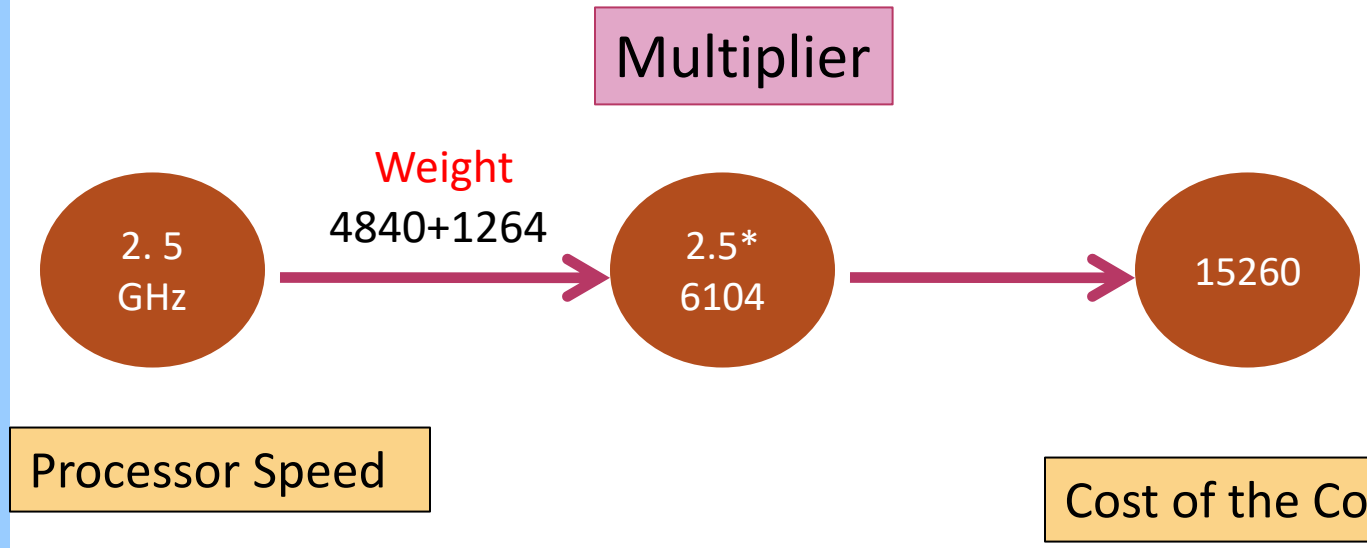


Error= Actual cost-calculated cost
=16000-9680
=6320

Change in
Weight=(Error/5)=1264

Understanding Single Neuron Perceptron

Speed	Cost
1.0 GHz	8000 INR
1.5 GHz	13000 INR
2.0 GHz	16000 INR
2.5 GHz	24000 INR
3.0 GHz	30000 INR
3.5 GHz	??

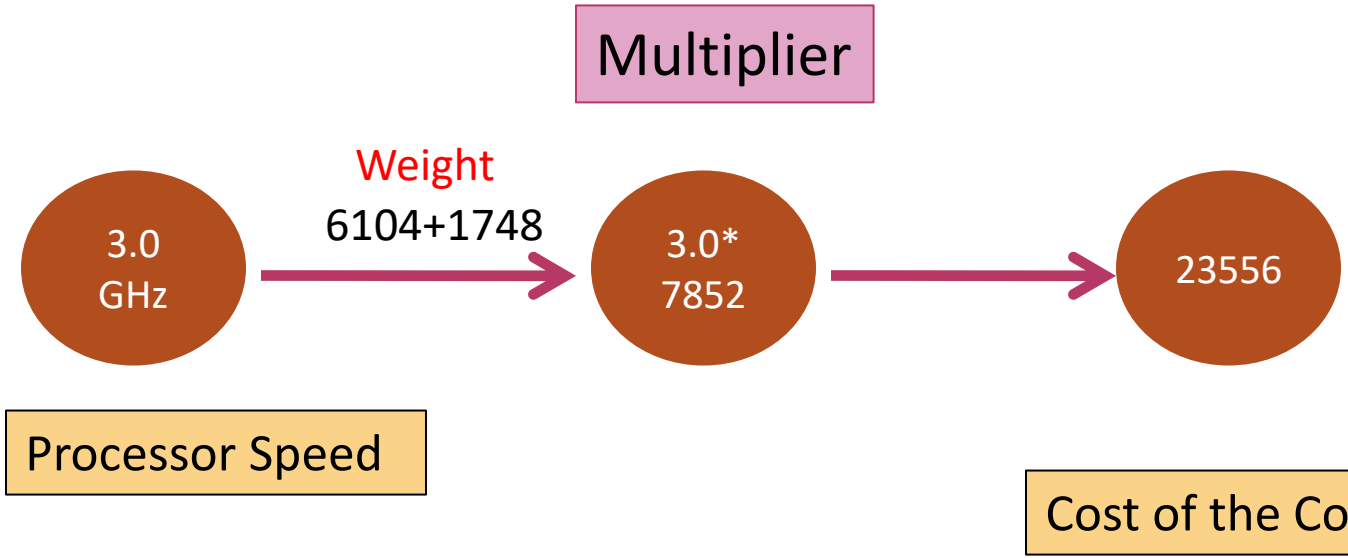


Error= Actual Cost- calculated cost = 24000-15260
=8740

Change in Weight=(Error/5)=1748

Understanding Single Neuron Perceptron

Speed	Cost
1.0 GHz	8000 INR
1.5 GHz	13000 INR
2.0 GHz	16000 INR
2.5 GHz	24000 INR
3.0 GHz	30000 INR
3.5 GHz	??

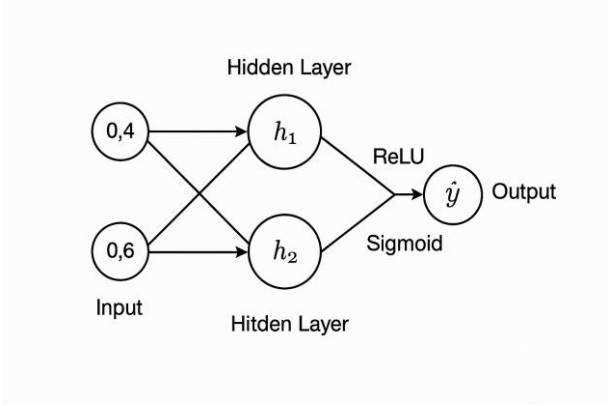


Error= Actual Cost-calculated cost = 30000-23556
= 6444

Change in Weight=(Error/5)=1289

	Speed	Actual Cost	Weight	Speed*Weight	Actual Error	Feed Error	Average Error
1st Epoch	1	8000	2000	2000	6000	1200	
	1.5	13000	3200	4800	8200	1640	
	2	16000	4840	9680	6320	1264	
	2.5	24000	6104	15260	8740	1748	
	3	30000	7852	23556	6444	1289	1428.2
2nd Epoch	1	8000	9141	9141	-1141	-228	
	1.5	13000	8913	13369.5	-369.5	-74	
	2	16000	8839	17678	-1678	-336	
	2.5	24000	8503	21257.5	2742.5	549	
	3	30000	9052	27156	2844	569	96
3rd Epoch	1	8000	9621	9621	-1621	-324	
	1.5	13000	9297	13945.5	-945.5	-189	
	2	16000	9108	18216	-2216	-443	
	2.5	24000	8665	21662.5	2337.5	468	
	3	30000	9133	27399	2601	520	6.4
4th Epoch	1	8000	9653	9653	-1653	-331	
	1.5	13000	9322	13983	-983	-197	
	2	16000	9125	18250	-2250	-450	
	2.5	24000	8675	21687.5	2312.5	463	
	3	30000	9138	27414	2586	517	0.4
5th Epoch	1	8000	9655	9655	-1655	-331	
	1.5	13000	9324	13986	-986	-197	
	2	16000	9127	18254	-2254	-451	
	2.5	24000	8676	21690	2310	462	
	3	30000	9138	27414	2586	517	0
	3.5		9138	31983			

Example



Network Structure

- **Inputs**: $x_1 = 0.4, x_2 = 0.6$
- **Hidden Layer**: 2 neurons (ReLU activation)
- **Output Layer**: 1 neuron (Sigmoid activation)

Weights & Biases

Hidden Layer:

- Neuron h_1:
  - Weights: $w_{1,1} = 0.2, w_{2,1} = 0.8$
  - Bias: $b_1 = 0.1$
- Neuron h_2:
  - Weights: $w_{1,2} = 0.6, w_{2,2} = -0.4$
  - Bias: $b_2 = 0.05$

Output Layer:

- Weights: $w_{h1} = -0.3, w_{h2} = 0.7$
- Bias: $b_{out} = 0.2$

Example

Forward pass calculation

Step 1: Hidden Layer Neurons

Neuron h_1:

$$z_1 = (0.4)(0.2) + (0.6)(0.8) + 0.1 = 0.08 + 0.48 + 0.1 = 0.66$$

$$h_1 = \text{ReLU}(0.66) = 0.66$$

Neuron h_2:

$$z_2 = (0.4)(0.6) + (0.6)(-0.4) + 0.05 = 0.24 - 0.24 + 0.05 = 0.05$$

$$h_2 = \text{ReLU}(0.05) = 0.05$$

Example

Forward pass calculation

Step 2: Output Neuron

$$z_{out} = (0.66)(-0.3) + (0.05)(0.7) + 0.2 = -0.198 + 0.035 + 0.2 = 0.037$$

$$\hat{y} = \text{Sigmoid}(0.037) = \frac{1}{1 + e^{-0.037}} \approx 0.509$$

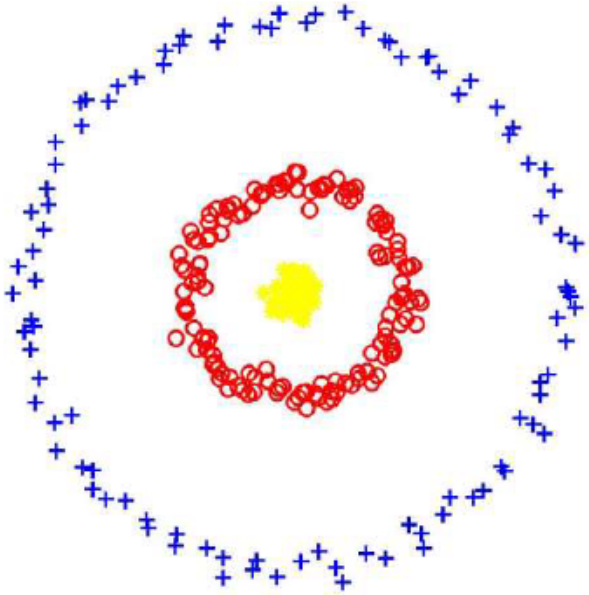
Final Output

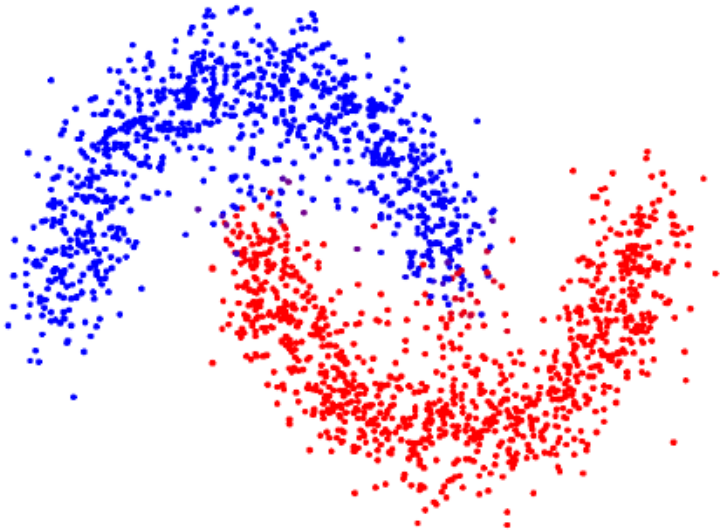
$$\hat{y} \approx 0.509 \quad \Rightarrow \text{Predicted Class: 1 (if threshold is 0.5)}$$

Non-Linearity

- Simple multiplication of weights with inputs and giving the outputs will be linear function.
- A linear function is just a polynomial of one degree and will not be able to learn complex functions. It has been found that linear functions don’t have much expressive power and will loose out when solving complex problems.
- Without activation functions (which will help us to achieve non-linearity) Neural Network will not be able to learn unstructured data like images, audio data, videos and text.

Non Linear Patterns





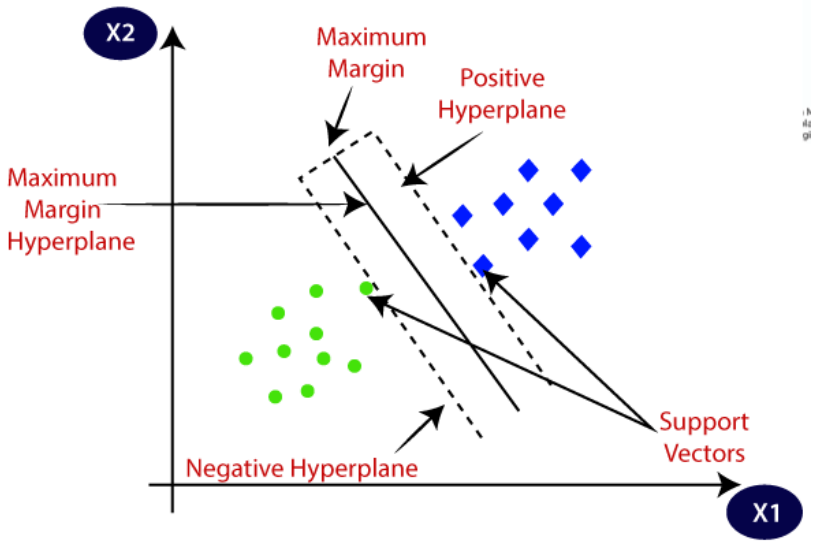
Support Vector Machine Algorithm

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate *n-dimensional* space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

- SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane:

Support Vector Machine Algorithm

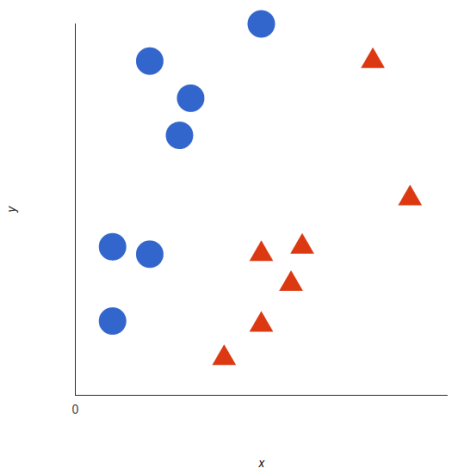


Hyperplane and Support Vectors in the SVM algorithm:

- There can be multiple lines/decision boundaries to segregate the classes in *n-dimensional* space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.
- The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.
- We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

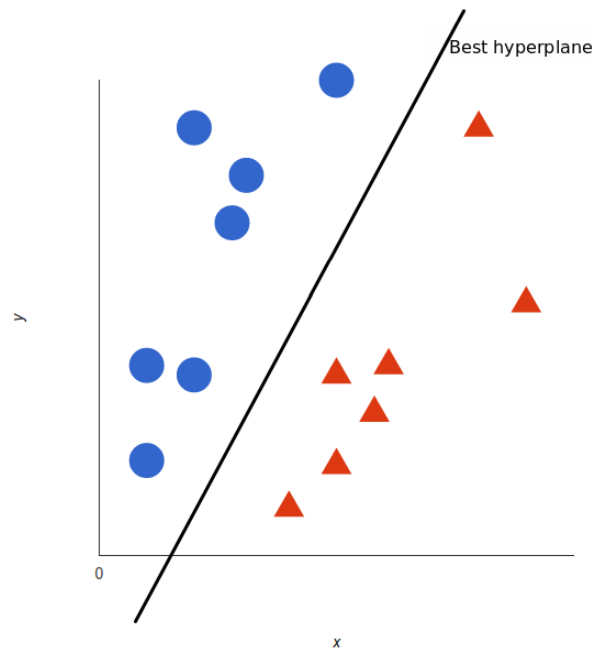
SVM Working : Linear Data

- *red* and *blue*, and our data has two features: *x* and *y*. We want a classifier that, given a pair of *(x,y)* coordinates, outputs if it's either *red* or *blue*. We plot our already labeled training data on a plane:



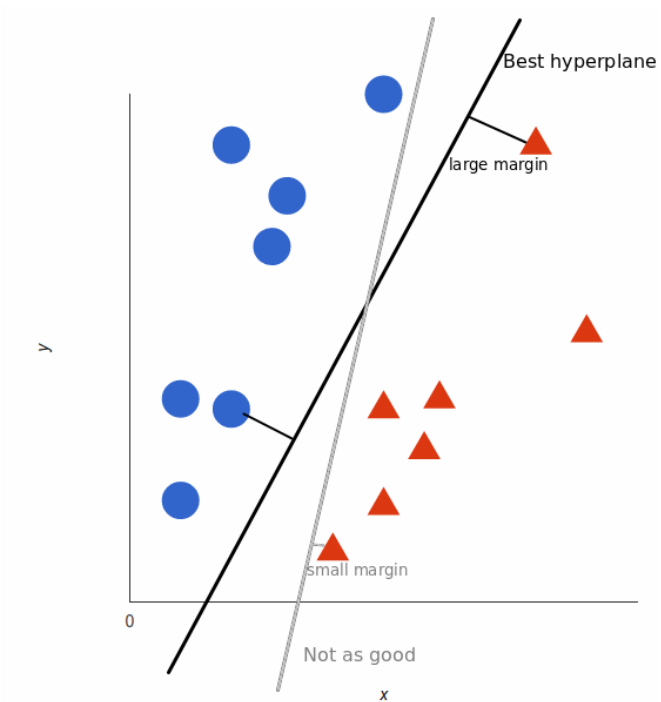
SVM Working : Linear Data

- A support vector machine takes these data points and outputs the hyperplane (which in two dimensions it's simply a line) that best separates the tags. This line is the **decision boundary**: anything that falls to one side of it we will classify as *blue*, and anything that falls to the other as *red*.



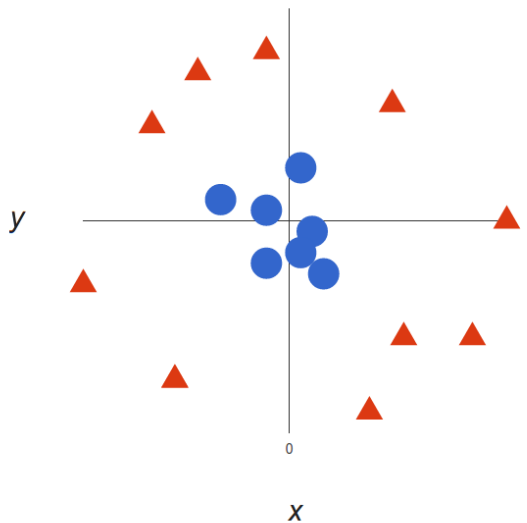
SVM Working : Linear Data

- But, what exactly is *the best* hyperplane? For SVM, it's the one that maximizes the margins from both tags. In other words: the hyperplane (remember it's a line in this case) whose distance to the nearest element of each tag is the largest.



SVM Working : Non-Linear Data

- Now this example was easy, since clearly the data was linearly separable — we could draw a straight line to separate *red* and *blue*. Sadly, usually things aren't that simple. Take a look at this case:



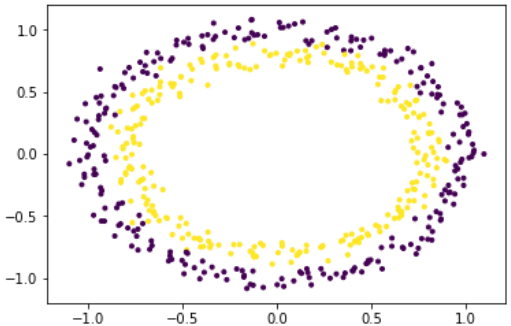
SVM Working : Non-Linear Data

- It's clear that there's not a linear decision boundary (a single straight line that separates both tags). However, the vectors are very clearly segregated and it looks as though it should be easy to separate them.

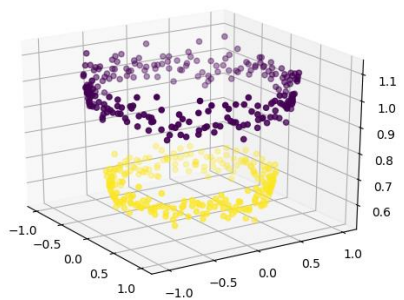
- So here's, third dimension is added. Up until now we had two dimensions: *x* and *y*. We create a new *z* dimension, and we rule that it be calculated a certain way that is convenient for us: $z = x^2 + y^2$ (you'll notice that's the equation for a circle).

- This will give us a three-dimensional space. Taking a slice of that space, it looks like this:

SVM Working : Non-Linear Data



Lets assume value of points on z plane, $w = x^2 + y^2$. In this case we can manipulate it as distance of point from z-origin. Now if we plot in z-axis, a clear separation is visible and a line can be drawn.



Tuning parameters

- Kernel
- Regularization
- Gamma

Kernel

- **Kernel Function** is a method used to take data as input and transform into the required form of processing data.
- "Kernel" is used due to set of mathematical functions used in Support Vector Machine provides the window to manipulate the data.
- So, Kernel Function generally transforms the training set of data so that a non-linear decision surface is able to transformed to a linear equation in a higher number of dimension spaces.
- Basically, it returns the inner product between two points in a standard feature dimension.

Kernel

- The learning of the hyperplane in linear SVM is done by transforming the problem using some linear algebra. This is where the kernel plays role.

- For **linear kernel** the equation for prediction for a new input using the dot product between the input (x) and each support vector (xi) is calculated as follows:

$$f(x) = B(o) + sum(ai * (x,xi))$$

- This is an equation that involves calculating the inner products of a new input vector (x) with all support vectors in training data. The coefficients Bo and ai (for each input) must be estimated from the training data by the learning algorithm.

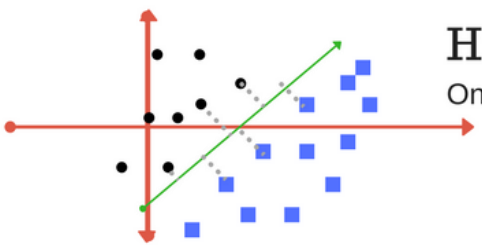
- The **polynomial kernel** can be written as $K(x,xi) = 1 + sum(x * xi)^d$ and **exponential** as $K(x,xi) = exp(-gamma * sum((x — xi^2))$.

Regularization

- Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.

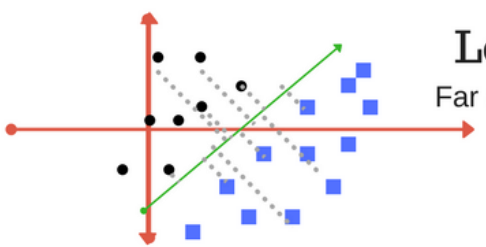
Gamma

- The gamma parameter defines how far the influence of a single training example reaches, with low values meaning ‘far’ and high values meaning ‘close’. In other words, with low gamma, points far away from plausible separation line are considered in calculation for the separation line. Where as high gamma means the points close to plausible line are considered in calculation.



High Gamma

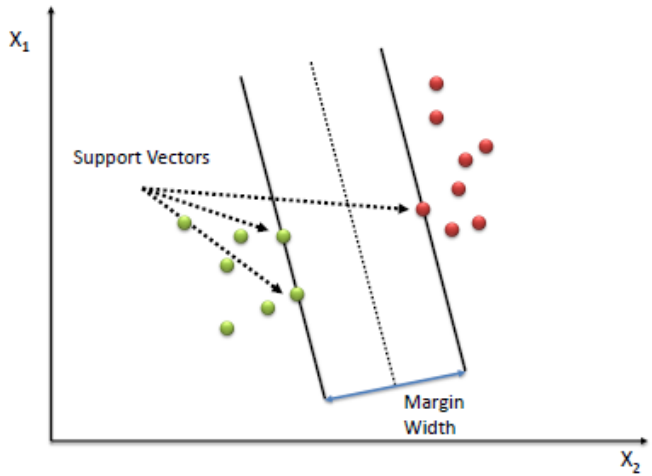
Only nearby points are considered.



Low Gamma

Far away points are also considered.

Support Vector Machine - Classification (SVM) : Algorithm



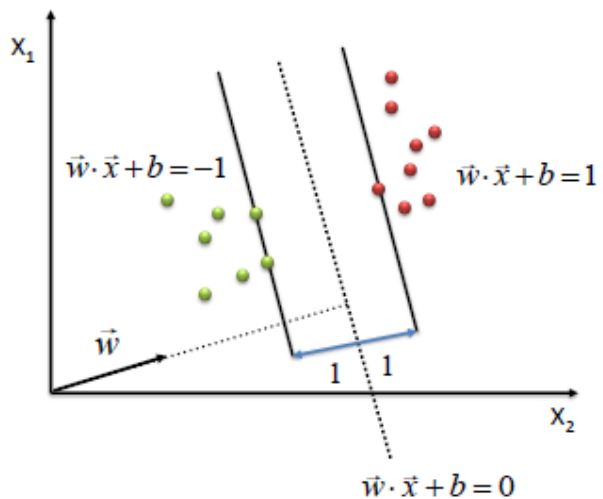
1. Define an optimal hyperplane: maximize margin

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

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

Support Vector Machine - Classification (SVM) : Algorithm

1. To define an optimal hyperplane we need to maximize the width of the margin (*w*).



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

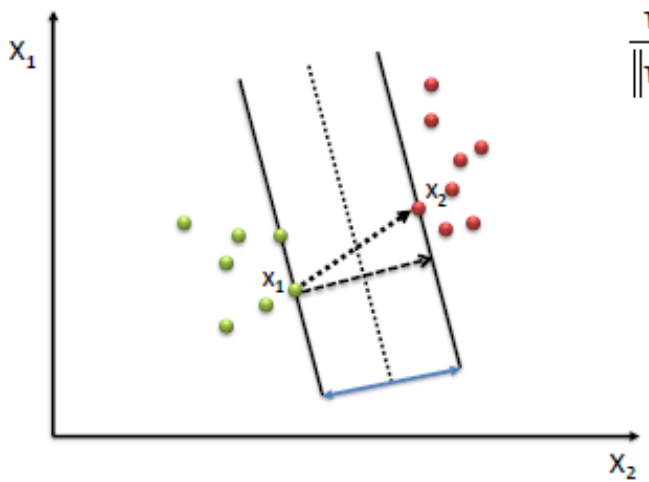
s.t.

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

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

Support Vector Machine - Classification (SVM) : Algorithm

1. To define an optimal hyperplane we need to maximize the width of the margin (*w*).



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

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

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

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

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

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

Support Vector Machine - Classification (SVM) : Algorithm

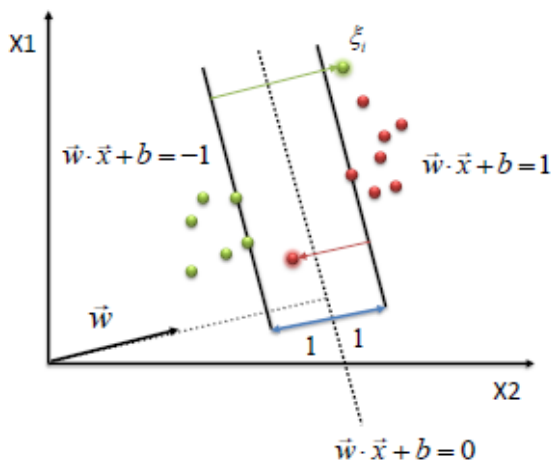
- We find *w* and *b* by solving the following objective function using **Quadratic Programming.**

$$\min \frac{1}{2}\|w\|^2$$

$$s.t.\ y_i(w \cdot x_i + b) \geq 1,\ \forall x_i$$

Support Vector Machine - Classification (SVM) : Algorithm

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



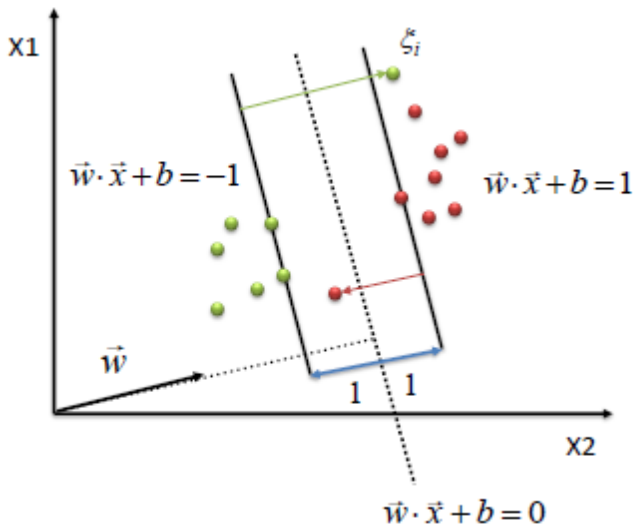
slack variable:

ξ_i

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

Support Vector Machine - Classification (SVM) : Algorithm

- The algorithm tries to maintain the **slack** variable to zero while maximizing margin. However, it does not minimize the number of misclassifications (NP-complete problem) but the sum of distances from the margin hyperplanes.



Constraint becomes :

$$y_i(w\cdot x_i+b)\geq 1-\xi_i,\ \forall x_i$$
$$\xi_i\geq 0$$

Objective function penalizes for misclassified instances and those within the margin

$$\min\frac{1}{2}\|w\|^2+C\sum_i\xi_i$$

C trades-off margin width and misclassifications

Support Vector Machine - Classification (SVM) : Algorithm

- The simplest way to separate two groups of data is with a straight **line (1 dimension), flat plane (2 dimensions) or an N-dimensional hyperplane**. However, there are situations where a nonlinear region can separate the groups more efficiently.

- SVM handles this by using a kernel function (nonlinear) to map the data into a different space where a hyperplane (linear) cannot be used to do the separation. It means a non-linear function is learned by a linear learning machine in a high-dimensional feature space while the capacity of the system is controlled by a parameter that does not depend on the dimensionality of the space. This is called *kernel trick* which means the kernel function transform the data into a higher dimensional feature space to make it possible to perform the linear separation.

Support Vector Machine - Classification (SVM) : Algorithm

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

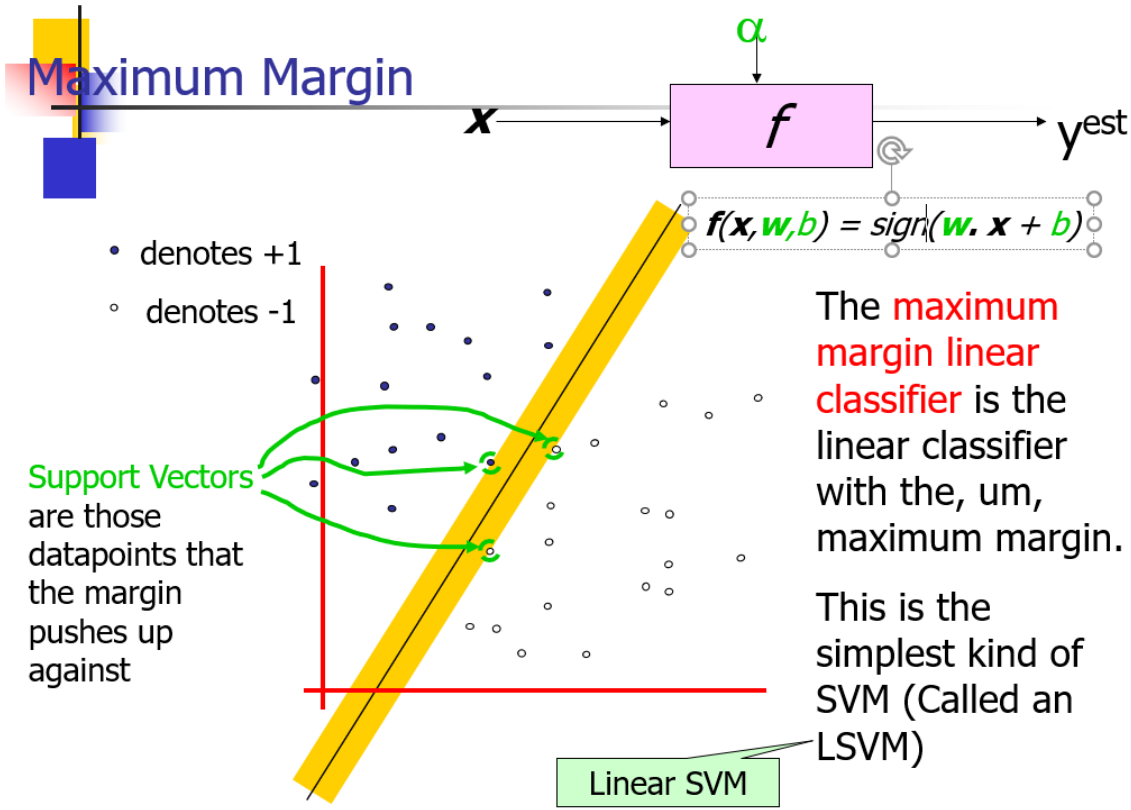
Polynomial

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

Gaussian Radial Basis function

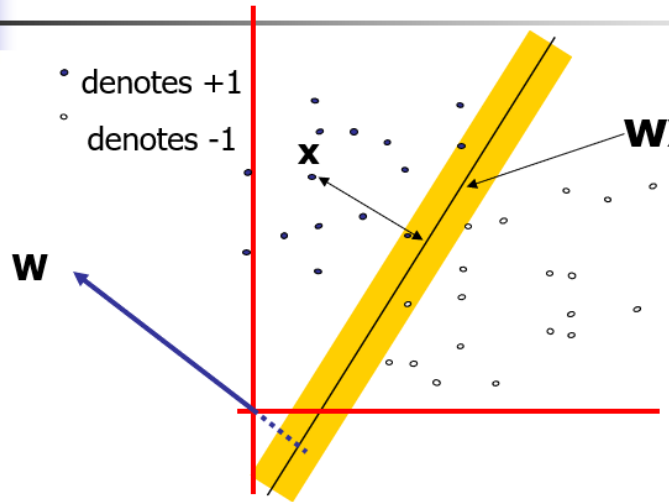
$$k(\mathrm{x}_i, \mathrm{x}_j) = \exp\left(-\frac{\left\| \mathrm{x}_i - \mathrm{x}_j \right\|^2}{2\sigma^2} \right)$$

Support Vector Machine - Classification (SVM) : Algorithm



Support Vector Machine - Classification (SVM) : Algorithm

Estimate the Margin



x – Vector

w – Normal Vector

b – Scale Value

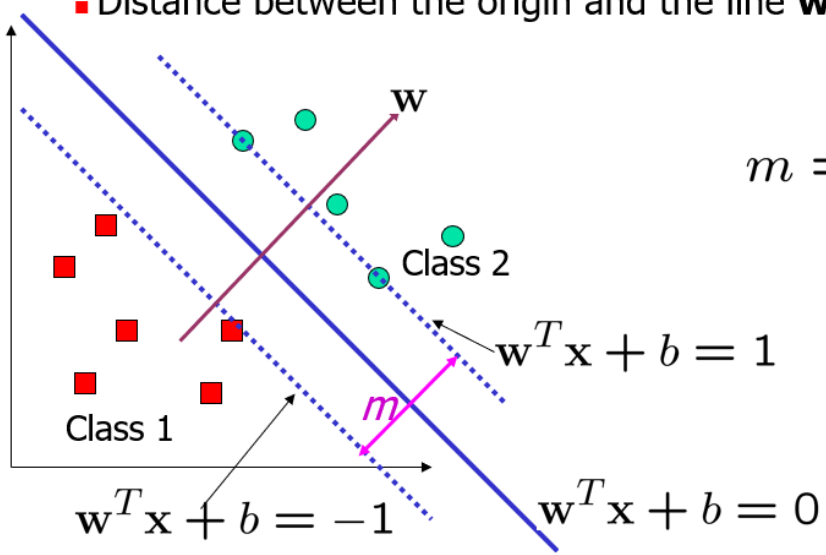
- What is the distance expression for a point **x** to a line **wx**+b= 0?

$$d(\mathbf{x}) = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

Support Vector Machine - Classification (SVM) : Algorithm

Large-margin Decision Boundary

- The decision boundary should be as far away from the data of both classes as possible
  - We should maximize the margin, *m*
  - Distance between the origin and the line $\mathbf{w}^t\mathbf{x}=-b$ is $b/||\mathbf{w}||$



$$m = \frac{2}{||\mathbf{w}||}$$

Support Vector Machine - Classification (SVM) : Algorithm

Finding the Decision Boundary

- Let $\{x_1, ..., x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly $\Rightarrow \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$
- To see this: when y=-1, we wish (wx+b)<1, when y=1, we wish (wx+b)>1. For support vectors, we wish y(wx+b)=1.
- The decision boundary can be found by solving the following constrained optimization problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

Support Vector Machine - Classification (SVM) : Algorithm

- Converting SVM to a form we can solve
  - Dual form
- Allowing a few errors
  - Soft margin
- Allowing nonlinear boundary
  - Kernel functions

The Dual Problem (we ignore the derivation)

- The new objective function is in terms of α_i only
- It is known as the dual problem: if we know **w**, we know all α_i; if we know all α_i, we know **w**
- The original problem is known as the primal problem
- The objective function of the dual problem needs to be maximized!
- The dual problem is therefore:

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \qquad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

Support Vector Machine - Classification (SVM) : Algorithm

The Dual Problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- This is a quadratic programming (QP) problem
  - A global maximum of α_i can always be found
- **w** can be recovered by $\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

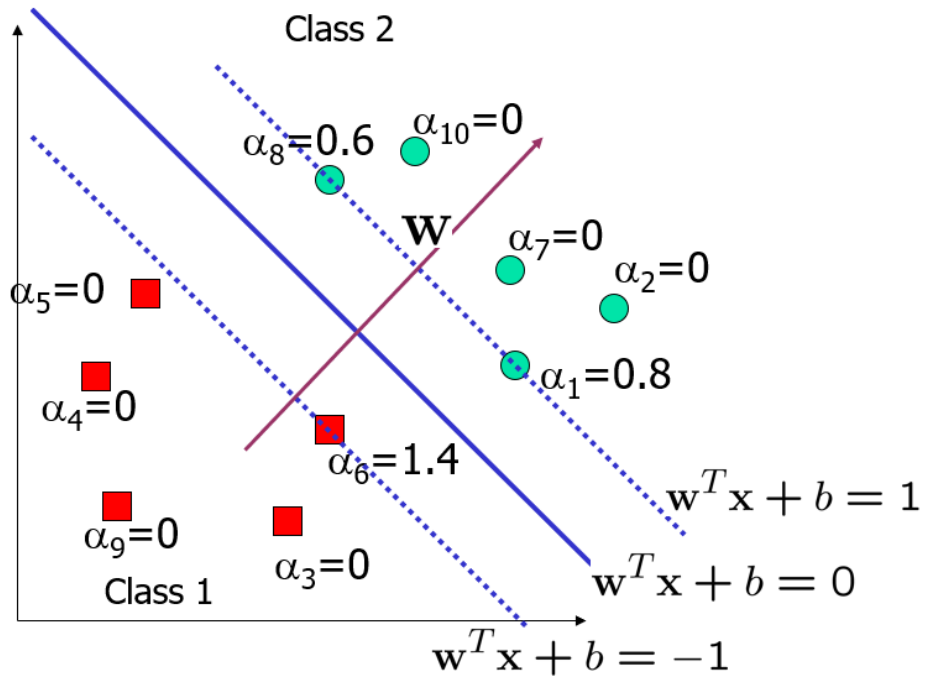
Support Vector Machine - Classification (SVM) : Algorithm

Characteristics of the Solution

- Many of the α_i are zero (see next page for example)
  - **w** is a linear combination of a small number of data points
  - This "sparse" representation can be viewed as data compression as in the construction of knn classifier
- $\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)
  - The decision boundary is determined only by the SV
  - Let t_j (j=1, ..., s) be the indices of the s support vectors. We can write $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} \mathbf{x}_{t_j}$
- For testing with a new data **z**
  - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{s} \alpha_{t_j} y_{t_j} (\mathbf{x}_{t_j}^T \mathbf{z}) + b$ and classify **z** as class 1 if the sum is positive, and class 2 otherwise
  - Note: **w** need not be formed explicitly

Support Vector Machine - Classification (SVM) : Algorithm

A Geometrical Interpretation



Support Vector Machine - Classification (SVM) : Algorithm

Soft Margin Hyperplane

- If we minimize $\sum_i \xi_i$, ξ_i can be computed by

$$\begin{cases} \mathbf{w}^T\mathbf{x}_i + b \geq 1 - \xi_i & y_i = 1 \\ \mathbf{w}^T\mathbf{x}_i + b \leq -1 + \xi_i & y_i = -1 \\ \xi_i \geq 0 & \forall i \end{cases}$$

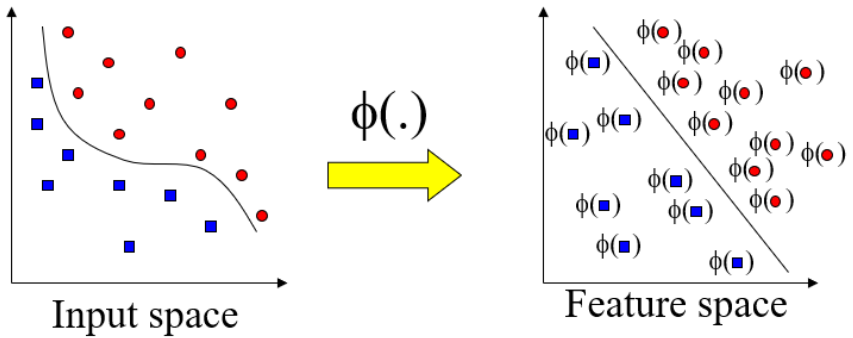
  - ξ_i are "slack variables" in optimization
  - Note that ξ_i=0 if there is no error for $\mathbf{x}_i$
  - ξ_i is an upper bound of the number of errors
- We want to minimize $\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$
  - C: tradeoff parameter between error and margin
- The optimization problem becomes

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$
$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

Extension to Non-linear Decision Boundary

- So far, we have only considered large-margin classifier with a linear decision boundary
- How to generalize it to become nonlinear?
- Key idea: transform $\mathbf{x}_i$ to a higher dimensional space to "make life easier"
  - Input space: the space the point $\mathbf{x}_i$ are located
  - Feature space: the space of $\phi(\mathbf{x}_i)$ after transformation

Transforming the Data (c.f. DHS Ch. 5)



Note: feature space is of higher dimension than the input space in practice

- Computation in the feature space can be costly because it is high dimensional
  - The feature space is typically infinite-dimensional!
- The kernel trick comes to rescue

Support Vector Machine - Classification (SVM) : Algorithm

The Kernel Trick

- Recall the SVM optimization problem

$$\text{max. } W(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1,j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{subject to } C \geq \alpha_i \geq 0, \sum_{i=1}^{n} \alpha_i y_i = 0$$

- The data points only appear as inner product
- As long as we can calculate the inner product in the feature space, we do not need the mapping explicitly
- Many common geometric operations (angles, distances) can be expressed by inner products
- Define the kernel function K by

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

An Example for $\phi(.)$ and K(.,.)

- Suppose $\phi(.)$ is given as follows

$$\phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$$

- An inner product in the feature space is

$$\langle \phi(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}), \phi(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}) \rangle = (1 + x_1y_1 + x_2y_2)^2$$

- So, if we define the kernel function as follows, there is no need to carry out $\phi(.)$ explicitly

$$K(\mathbf{x}, \mathbf{y}) = (1 + x_1y_1 + x_2y_2)^2$$

- This use of kernel function to avoid carrying out $\phi(.)$ explicitly is known as the kernel trick

More on Kernel Functions

- Not all similarity measures can be used as kernel function, however
  - The kernel function needs to satisfy the Mercer function, i.e., the function is "positive-definite"
- This implies that
  - the *n* by *n* kernel matrix,
  - in which the (i,j)-th entry is the $K(\mathbf{x}_i, \mathbf{x}_j)$, is always positive definite
- This also means that optimization problem can be solved in polynomial time!

Examples of Kernel Functions

- Polynomial kernel with degree *d*

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function kernel with width σ

$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional

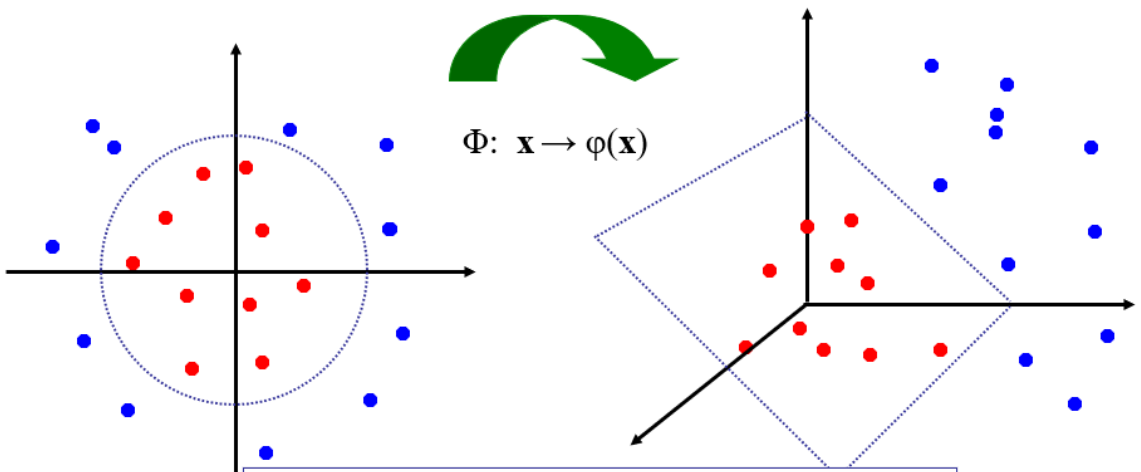
- Sigmoid with parameter κ and θ

$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$

  - It does not satisfy the Mercer condition on all κ and θ

Non-linear SVMs: Feature spaces

General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable:



$$\Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

Example

- Suppose we have 5 one-dimensional data points
  - x_1=1, x_2=2, x_3=4, x_4=5, x_5=6, with 1, 2, 6 as class 1 and 4, 5 as class 2 $\Rightarrow$ y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1
- We use the polynomial kernel of degree 2
  - $K(x,y) = (xy+1)^2$
  - C is set to 100
- We first find α_i (i=1, …, 5) by

$$\max. \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j (x_i x_j + 1)^2$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^{5} \alpha_i y_i = 0$$

Example

- By using a QP solver, we get
  - α_1=0, α_2=2.5, α_3=0, α_4=7.333, α_5=4.833
  - Note that the constraints are indeed satisfied
  - The support vectors are {x_2=2, x_4=5, x_5=6}
- The discriminant function is

$$f(z) = 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + 4.833(1)(6z+1)^2 + b$$
$$= 0.6667z^2 - 5.333z + b$$

(α_5, y_5, $K(z, x_5)$)

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x_2 and x_5 lie on the line $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = 1$ and x_4 lies on the line $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = -1$
- All three give b=9 $\Rightarrow$ $f(z) = 0.6667z^2 - 5.333z + 9$

Software

- A list of SVM implementation can be found at http://www.kernel-machines.org/software.html
- Some implementation (such as LIBSVM) can handle multi-class classification
- SVMLight is among one of the earliest implementation of SVM
- Several Matlab toolboxes for SVM are also available

Summary: Steps for Classification

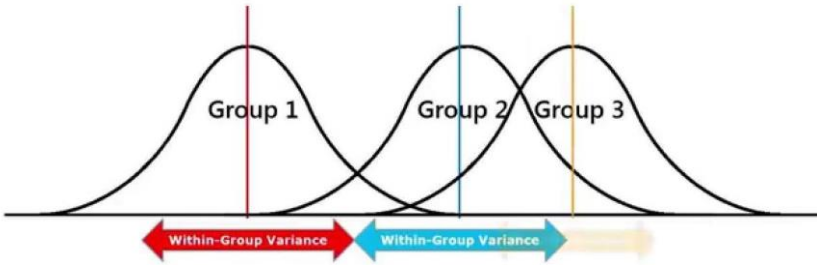
- Prepare the pattern matrix
- Select the kernel function to use
- Select the parameter of the kernel function and the value of C
  - You can use the values suggested by the SVM software, or you can set apart a validation set to determine the values of the parameter
- Execute the training algorithm and obtain the α_i
- Unseen data can be classified using the α_i and the support vectors

ANOVA: Analysis of Variation

- What is analysis of variance (ANOVA)? Analysis of Variance (ANOVA) is a statistical formula used to compare variances across the means (or average) of different groups.

- A range of scenarios use it to determine if there is any difference between the means of different groups.

- Two common types of ANOVA: one-way and two-way ANOVA.



One-Way ANOVA

- One-way ANOVA is a statistical test used to determine whether there are any **statistically significant differences between the means of three or more independent (unrelated) groups**. It compares the means of the groups to see if at least one of them is significantly different from the others.

When to Use One-Way ANOVA?

One-way ANOVA is used when we have:

- One independent variable (factor) with three or more levels (groups).
- A continuous dependent variable.

One-Way ANOVA

The one way ANOVA test is used to determine whether there is any difference between the means of three or more groups. A one way ANOVA will have only one independent variable. The hypothesis for a one way ANOVA test can be set up as follows:

Null Hypothesis, H0**:** $\mu 1 = \mu 2 = \mu 3 = ... = \mu k$

Alternative Hypothesis, H1**:** The means are not equal

Decision Rule: If test statistic > critical value then reject the null hypothesis and conclude that the means of at least two groups are statistically significant.

The p-value is the probability of obtaining a test statistic (like an F-statistic, t-statistic, etc.) at least as extreme as the one observed, assuming that the null hypothesis is true.

One-Way ANOVA

The steps to perform the one way ANOVA test are given below:

- **Step 1:** Calculate the mean for each group.
- **Step 2:** Calculate the total mean. This is done by adding all the means and dividing it by the total number of means.
- **Step 3:** Calculate the SSB.
- **Step 4:** Calculate the between groups degrees of freedom.
- **Step 5:** Calculate the SSE/SSW.
- **Step 6:** Calculate the degrees of freedom of errors.
- **Step 7:** Determine the MSB and the MSE.
- **Step 8:** Find the f test statistic.
- **Step 9:** Using the f table for the specified level of significance, αα, find the critical value. This is given by F(αα, df_1. df_2).
- **Step 10:** If f > F then reject the null hypothesis.

One-Way ANOVA : Calculation

Total Sum of Squares (SST)

This measures the total variation in the data.

$$\text{SST} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (X_{ij} - \bar{X})^2$$

Where:

- X_{ij} is the j-th observation in group i
- $\bar{X}$ is the grand mean (overall mean)
- k is the number of groups
- n_i is the number of observations in group i

Sum of Squares Between Groups (SSB)

This measures the variation between the group means and the overall mean.

$$\text{SSB} = \sum_{i=1}^{k} n_i (\bar{X}_i - \bar{X})^2$$

Where:

- $\bar{X}_i$ is the mean of group i
- $\bar{X}$ is the grand mean
- n_i is the number of observations in group i

Sum of Squares Within Groups (SSW)

This measures the variation within each group.

$$\text{SSW} = \sum_{i=1}^{k} \sum_{j=1}^{n_i} (X_{ij} - \bar{X}_i)^2$$

Degrees of Freedom

- Between groups: $df_{\text{between}} = k - 1$
- Within groups: $df_{\text{within}} = N - k$
- Total: $df_{\text{total}} = N - 1$

Where N is the total number of observations across all groups.

One-Way ANOVA : Calculation

Mean Squares (MS)

- Between groups: $MS_{\text{between}} = \frac{SSB}{df_{\text{between}}}$
- Within groups: $MS_{\text{within}} = \frac{SSW}{df_{\text{within}}}$

F-statistic

$$F = \frac{MS_{\text{between}}}{MS_{\text{within}}}$$

p-value

To calculate the p-value, use the F-distribution with $df_1 = df_{\text{between}}$ and $df_2 = df_{\text{within}}$.

Find p-value from F-statistic -
Using F-distribution Table (Manual Method)

We can look up the **F-table** with your degrees of freedom. Find the critical F value for your α\alphaα level (e.g., 0.05). If your observed F > critical F, then the p-value < 0.05 (i.e., statistically significant).

One-Way ANOVA

- For example, suppose a researcher wants to test the effect of three different diets on weight loss. The diets are labeled as Diet A, Diet B, and Diet C. The weight loss (in pounds) of participants on each diet is recorded, and one-way ANOVA is used to determine if there is a significant difference in weight loss among the diets.

Assumptions of One-Way ANOVA

1. **Independence of Observations:** The data collected from the groups should be independent of each other.
2. **Normality:** The data in each group should be approximately normally distributed.
3. **Homogeneity:** The variances among the groups should be approximately equal.

One-Way ANOVA

To determine whether there is a **statistically significant difference** in **mean weight loss** among three different diet programs: **Diet A**, **Diet B**, and **Diet C**.

Weight loss in pounds over a 4-week program

Participant	Diet A	Diet B	Diet C
1	6	8	10
2	5	7	9
3	4	6	8
4	5	9	7

Step1:Hypotheses

- **Null Hypothesis** H_0:

  $\mu_A = \mu_B = \mu_C$

  (All diet programs have the same average weight loss)

- **Alternative Hypothesis** H_1:

  At least one mean is different.

Step 2: ANOVA Table Calculations

- **Group Means**
  - $\bar{X}_A = \frac{6+5+4+5}{4} = 5.0$
  - $\bar{X}_B = \frac{8+7+6+9}{4} = 7.5$
  - $\bar{X}_C = \frac{10+9+8+7}{4} = 8.5$
- **Overall Mean** $\bar{X}$

$$\bar{X} = \frac{6+5+4+5+8+7+6+9+10+9+8+7}{12} = 84 / 12$$

= 7.00

One-Way ANOVA

Step 3: Compute Sum of Squares

(a) Between Group Sum of Squares (SSB)

$$SSB = n \sum_{i=1}^{k} (\bar{X}_i - \bar{X})^2$$

Where $n = 4$ per group, $k = 3$

$$SSB = 4(5-7)^2 + 4(7.5-7)^2 + 4(8.5-7)^2$$
$$= 4(4) + 4(0.25) + 4(2.25) = 16 + 1 + 9 = \boxed{26}$$

One-Way ANOVA

Step 3: Compute Sum of Squares

(b) Within Group Sum of Squares (SSW)

$$SSW = \sum_{i=1}^{k} \sum_{j=1}^{n} (X_{ij} - \bar{X}_i)^2$$

Compute deviations within each group:

- **Diet A**: $(6-5)^2 + (5-5)^2 + (4-5)^2 + (5-5)^2 = 1 + 0 + 1 + 0 = 2$
- **Diet B**: $(8-7.5)^2 + (7-7.5)^2 + (6-7.5)^2 + (9-7.5)^2 = 0.25 + 0.25 + 2.25 + 2.25 = 5$
- **Diet C**: $(10-8.5)^2 + (9-8.5)^2 + (8-8.5)^2 + (7-8.5)^2 = 2.25 + 0.25 + 0.25 + 2.25 = 5$

$$SSW = 2 + 5 + 5 = 12$$

One-Way ANOVA

Step 4: Degrees of Freedom

df_between = k - 1 = 3 - 1 = 2
df_within = N - k = 12 - 3 = 9

dfB = Degrees of Freedom Between = k−1, where k is number of groups
dfW = Degrees of Freedom Within = N−k, where N is total number of observations

Step 5: Mean Squares

MSB = SSB / df_between = 26 / 2 = 13.00
MSW = SSW / df_within = 12 / 9 = 1.33

One-Way ANOVA

Step 6: F-Statistic

$$F = \frac{MSB}{MSW} = \frac{13.00}{1.33} \approx \boxed{9.77}$$

Step 7: p-value (from F-distribution table or calculator)

Using F-distribution with df1 = 2, df2 = 9, F = 9.77

p-value ≈ 0.0055

Since p-value < 0.05, we reject the null hypothesis.
There is a significant difference in weight loss among the three diets.

http://www.socr.ucla.edu/Applets.dir/F_Table.html

https://users.sussex.ac.uk/~grahamh/RM1web/F-ratio%20table%202005.pdf

One-Way ANOVA

ANOVA Table

Source	SS	df	MS	F	p-value
Between Groups	26	2	13.0	9.77	0.0055
Within Groups	12	9	1.33		
Total	38	11			

Question

Consider a simple feedforward neural network with the following characteristics:
Input Layer: 2 neurons (representing two features, x1 and x2).
Hidden Layer: 2 neurons with a ReLU activation function.
Output Layer: 1 neuron with a sigmoid activation function.

Given the following weights and biases:
Weights from the input layer to the hidden layer:

W1=[0.5−0.3; −0.20 .8]
W2=[−0.7 0.6]

Biases for the hidden layer:
b1=0.1
b2=−0.4

Bias for the output layer:
b3=0.2

Calculate the output of the neural network for the input values x1=0.5, x2=−0.3

Resources

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.support-vector.net/icml-tutorial.pdf
- http://www.kernel-machines.org/papers/tutorial-nips.ps.gz
- http://www.clopinet.com/isabelle/Projects/SVM/applist.html