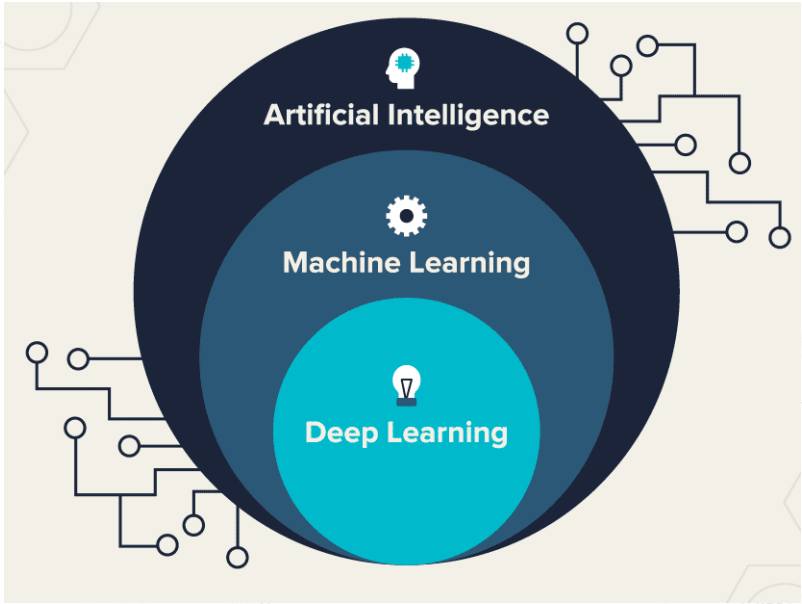
Machine Learning

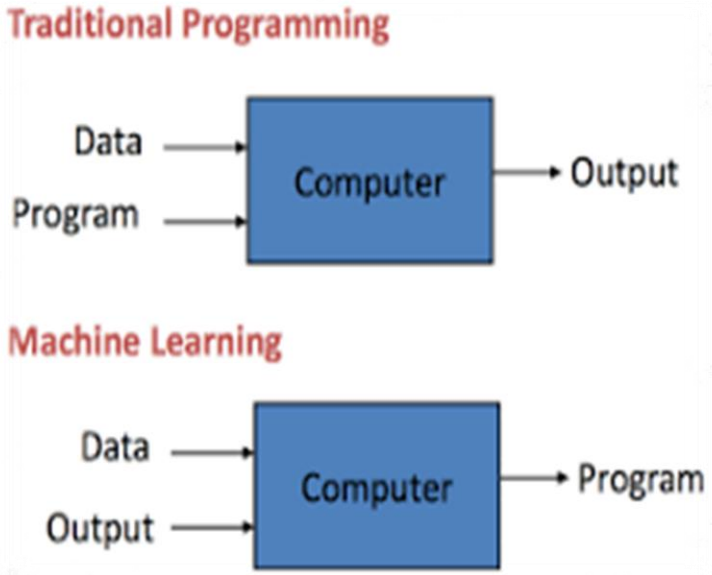


Machine Learning

Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term **"Machine Learning"** as – **"Field of study that gives computers the capability to learn without being explicitly programmed"**.

How it is different from traditional Programming:

- In Traditional Programming, we feed the Input, Program logic and run the program to get output.
- In Machine Learning, we feed the input, output and run it on machine during training and the machine creates its own logic, which is being evaluated while testing.



Terminologies that one should know before starting Machine Learning:

- **Model:** A model is a **specific representation** learned from data by applying some machine learning algorithm. A model is also called **hypothesis**.

- **Feature:** A feature is an individual measurable property of our data. A set of numeric features can be conveniently described by a **feature vector**. Feature vectors are fed as input to the model. For example, in order to predict a fruit, there may be features like color, smell, taste, **etc.**

- **Target(Label):** A target variable or label is the value to be predicted by our model. For the fruit example discussed in the features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.

- **Training:** The idea is to give a set of inputs(features) and it's expected outputs(labels), so after training, we will have a model (hypothesis) that will then map new data to one of the categories trained on.

- **Prediction:** Once our model is ready, it can be fed a set of inputs to which it will provide a predicted output(label).

Types of Learning

- **Supervised Learning**
- **Unsupervised Learning**
- **Semi-Supervised Learning**

1. **Supervised Learning**: Supervised learning is when the model is getting trained on a labelled dataset. **Labelled** dataset is one which have both input and output parameters. In this type of learning both training and validation datasets are labelled as shown in the figures below.

User ID	Gender	Age	Salary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	1
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	1
15728773	Male	27	58000	1
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	1
15727311	Female	35	65000	0
15570769	Female	26	80000	1
15606274	Female	26	52000	0

Classification

Temperature	Pressure	Relative Humidity	Wind Direction	Wind Speed
10.69261758	986.882019	54.19337313	195.7150879	3.278597116
13.59184184	987.8729248	48.0648859	189.2951202	2.909167767
17.70494885	988.1119385	39.11965597	192.9273834	2.973036289
20.95430404	987.8500366	30.66273218	202.0752869	2.965289593
22.9278274	987.2833862	26.06723423	210.6589203	2.798230886
24.04233986	986.2907104	23.46918024	221.1188507	2.627005816
24.41475295	985.2338867	22.25082295	233.7911987	2.448749781
23.93361956	984.8914795	22.35178837	244.3504333	2.454271793
22.68800023	984.8461304	23.7538641	253.0864716	2.418341875
20.56425726	984.8380737	27.07867944	264.5071106	2.318677425
17.76400389	985.4262085	33.54900114	280.7827454	2.343950987

Regression

Types of Supervised Learning:

- **Classification**
- **Regression**

Classification : It is a Supervised Learning task where output is having defined labels(discrete value). For example in above Figure A, Output – Purchased has defined labels i.e. 0 or 1 ; 1 means the customer will purchase and 0 means that customer won't purchase. It can be either binary or multi class classification. In **binary** classification, model predicts either 0 or 1 ; yes or no but in case of **multi class** classification, model predicts more than one class.

Example: Gmail classifies mails in more than one classes like social, promotions, updates, offers.

Regression : It is a Supervised Learning task where output is having continuous value. Example in before regression Figure, Output – Wind Speed is not having any discrete value but is continuous in the particular range. The goal here is to predict a value as much closer to actual output value as our model can and then evaluation is done by calculating error value. The smaller the error the greater the accuracy of our regression model.

Example of Supervised Learning Algorithms:

- Linear Regression
- Nearest Neighbor
- Gaussian Naive Bayes
- Decision Trees
- Support Vector Machine (SVM)
- Random Forest

Unsupervised Learning:

Unsupervised learning is the training of machine using information that is neither classified nor labeled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unsupervised machine learning is more challenging than supervised learning due to the absence of labels.

Types of Unsupervised Learning:

- **Clustering**
- **Association**

Unsupervised Learning:

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.

Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Examples of unsupervised learning algorithms are:

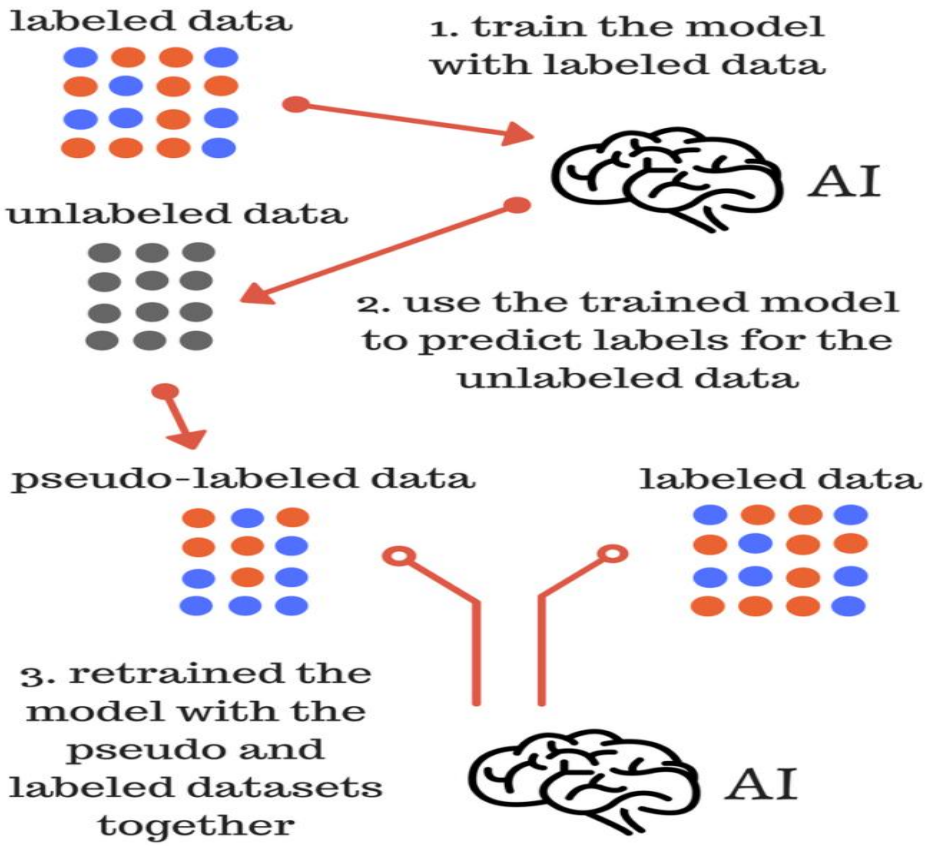
- k-means for clustering problems.
- Apriori algorithm for association rule learning problems

The most basic disadvantage of any **Supervised Learning** algorithm is that the dataset has to be hand-labeled either by a Machine Learning Engineer or a Data Scientist. This is a very *costly process*, especially when dealing with large volumes of data. The most basic disadvantage of any **Unsupervised Learning** is that it's **application spectrum is limited**.

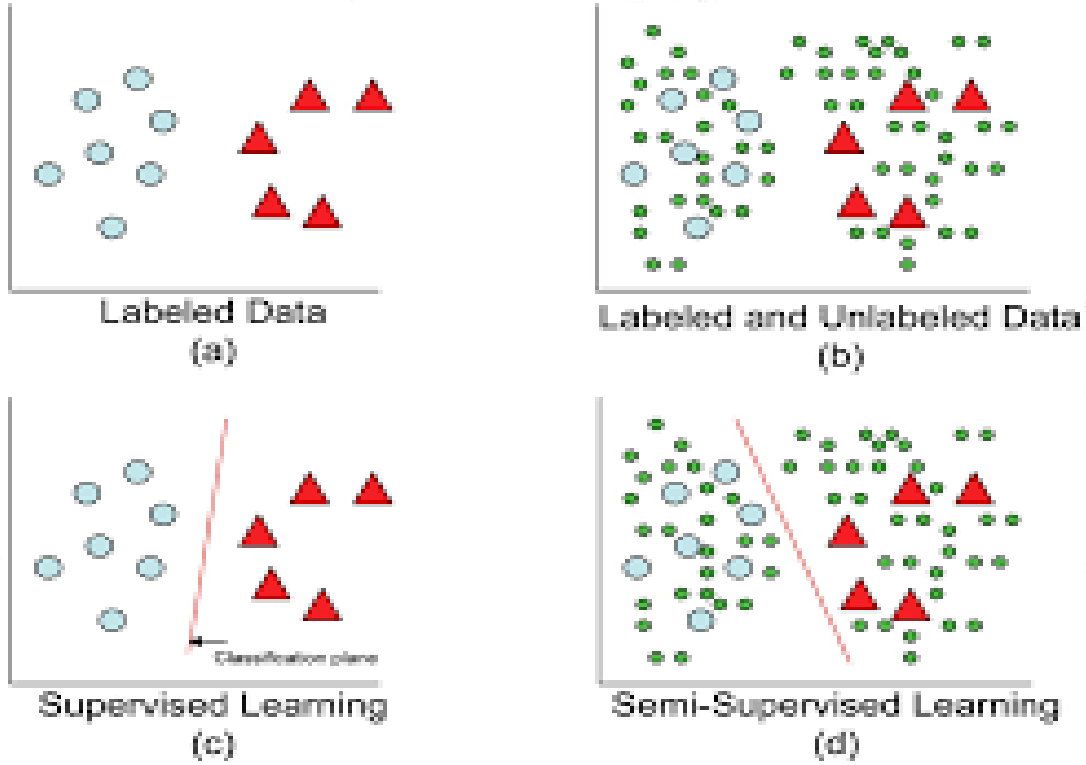
Semi-supervised machine learning:

To counter these disadvantages, the concept of **Semi-Supervised Learning** was introduced. In this type of learning, the algorithm is trained upon a combination of labeled and unlabeled data. Typically, this combination will contain a very small amount of labeled data and a very large amount of unlabeled data.

- In semi supervised learning labelled data is used to learn a model and using that model unlabeled data is labelled called pseudo labelling now using whole data model is trained for further use



Model with labellled data and model with both labelled and unlabelled data



Intuitively, one may imagine the three types of learning algorithms as Supervised learning where a student is under the supervision of a teacher at both home and school, Unsupervised learning where a student has to figure out a concept himself and Semi-Supervised learning where a teacher teaches a few concepts in class and gives questions as homework which are based on similar concepts.

REGRESSION

Regression is a statistical measurement used in finance, investing, and other disciplines that attempts to determine the strength of the relationship between one dependent variable and a series of other changing variables or independent variable

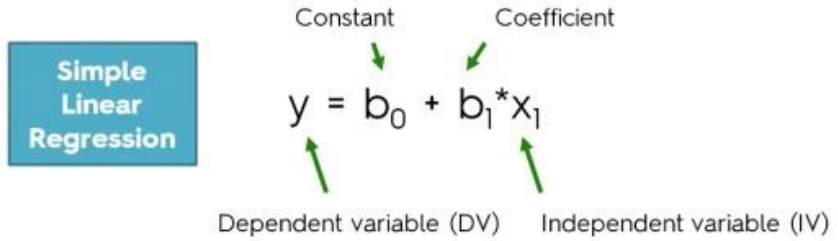
Types of regression

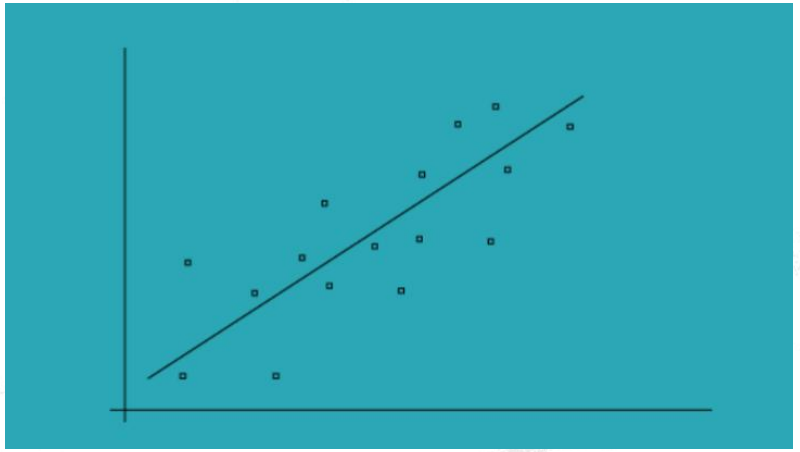
- linear regression
  - Simple linear regression
  - Multiple linear regression
- Polynomial regression
- Decision tree regression
- Random forest regression

Simple Linear regression

- The simple linear regression models are used to show or predict the relationship between the two variables or factors

- The factor that being predicted is called dependent variable and the factors that is are used to predict the dependent variable are called independent variables

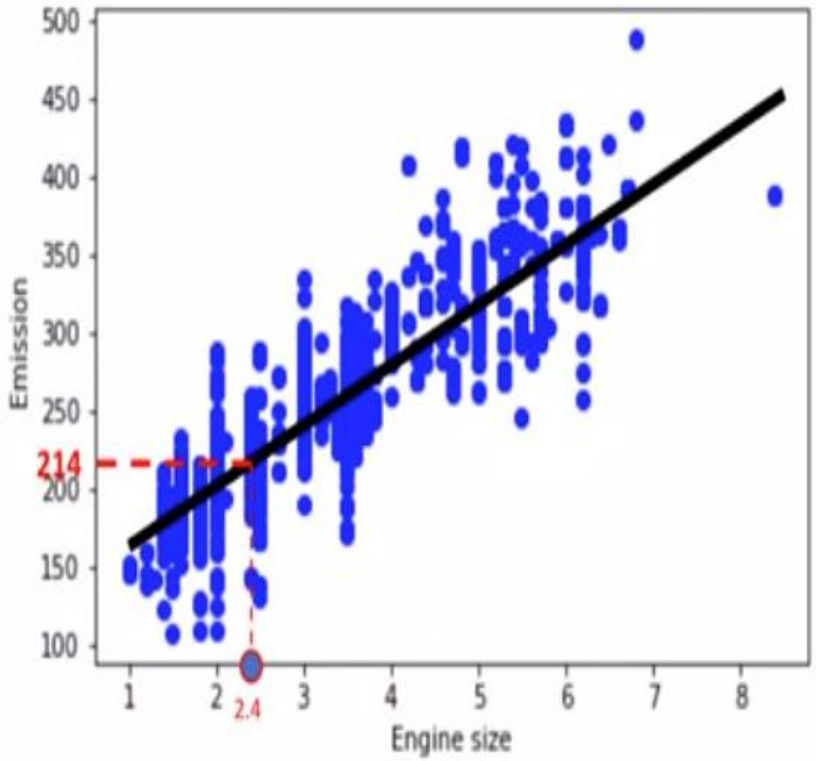




Simple Linear regression

Predicting C02 emission with engine size feature using simple linear regression

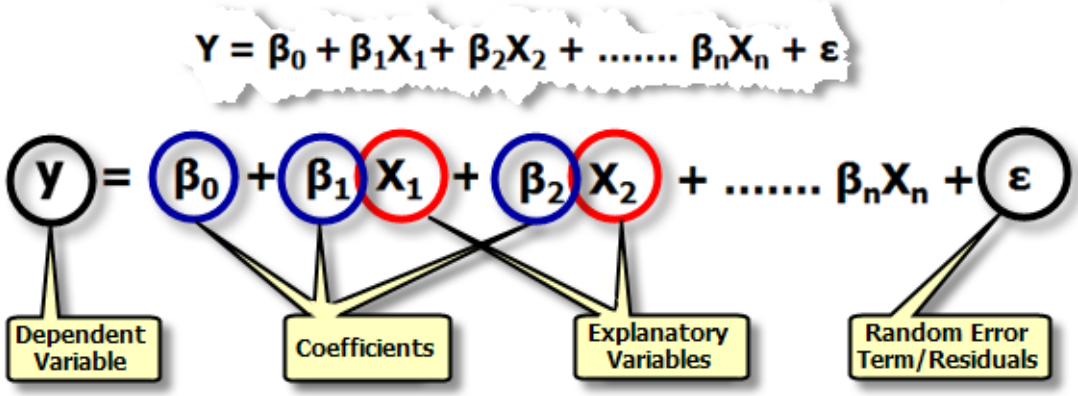
	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267
9	2.4	4	9.2	?



```
from sklearn import linear_model

regr = linear_model.LinearRegression()

train_x = np.asanyarray(train[['ENGINESIZE']])

train_y = np.asanyarray(train[['CO2EMISSIONS']])

regr.fit (train_x, train_y)

# The coefficients

print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

Multiple linear regression

- Multiple regression is an extension of simple linear regression. It is used when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable).



Simple linear regression

- Predict CO2 emission vs Engine size of all cars
  - Independent variable(x) : Engine size
  -Dependent variable(y):CO2 emission

Multiple linear regression

- Predict CO2 emission vs Engine size and cylinders of all car
  -Independent variable(x) : engine size,cylinders
  -Dependent variable(y):CO2 emission

	ENGINESIZE	CYLINDERS	FUELCONSUMPTION_COMB	CO2EMISSIONS
0	2.0	4	8.5	196
1	2.4	4	9.6	221
2	1.5	4	5.9	136
3	3.5	6	11.1	255
4	3.5	6	10.6	244
5	3.5	6	10.0	230
6	3.5	6	10.1	232
7	3.7	6	11.1	255
8	3.7	6	11.6	267
9	2.4	4	9.2	?

```
from sklearn import linear_model

regr = linear_model.LinearRegression()

train_x =
np.asanyarray(train[['ENGINESIZE','CYLINDERS']])

train_y = np.asanyarray(train[['CO2EMISSIONS']])

regr.fit (train_x, train_y)

# The coefficients

print ('Coefficients: ', regr.coef_)
print ('Intercept: ',regr.intercept_)
```

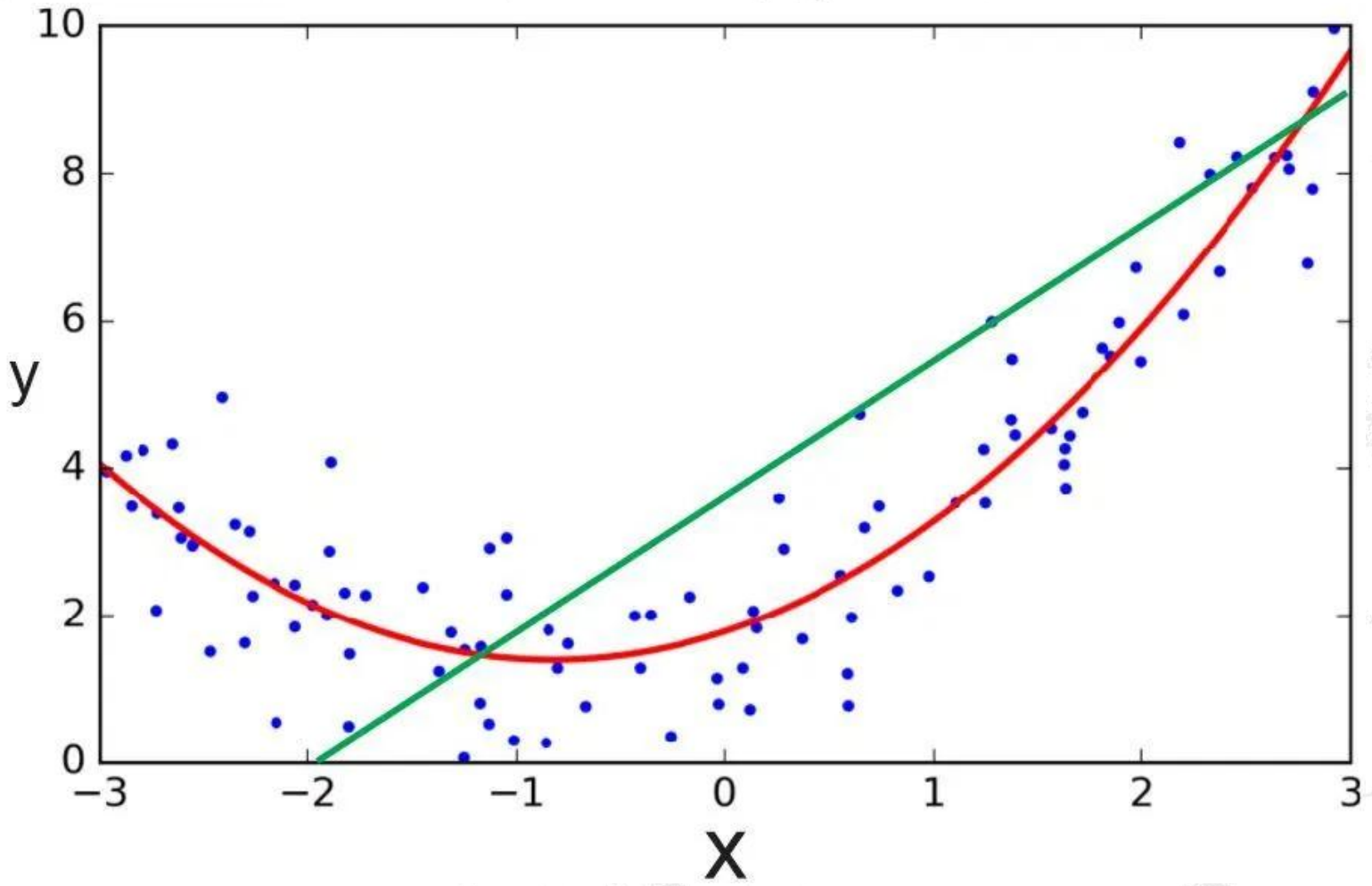
Polynomial regression

- Polynomial Regression is a form of linear regression in which the relationship between the independent variable x and dependent variable y is modelled as an *nth* degree polynomial. Polynomial regression fits a nonlinear relationship between the value of x and the corresponding conditional mean of y, denoted E(y |x)

$$y_i = \beta_0 + \beta_1 \tilde{X}_i + \beta_2 Z_i + \beta_3 \tilde{X}_i Z_i + \beta_4 \tilde{X}_i^2 + \beta_5 \tilde{X}_i^2 Z_i + e_i$$

where:

- y_i = outcome score for the i^{th} unit
- β_0 = coefficient for the *intercept*
- β_1 = linear pretest coefficient
- β_2 = mean difference for treatment
- β_3 = linear interaction
- β_4 = quadratic pretest coefficient
- β_5 = quadratic interaction
- X_i = transformed pretest
- Z_i = dummy variable for treatment(0 = control, 1= treatment)
- e_i = residual for the i^{th} unit


10
8
6
4
2
0
y
−3
−2
−1
0
1
2
3
X

```
from sklearn.preprocessing import PolynomialFeatures

from sklearn import linear_model

train_x = np.asanyarray(train[['ENGINESIZE','CYLINDERS']])
train_y = np.asanyarray(train[['CO2EMISSIONS']])


test_x = np.asanyarray(test[['ENGINESIZE']])
test_y = np.asanyarray(test[['CO2EMISSIONS']])


poly = PolynomialFeatures(degree=2)
train_x_poly = poly.fit_transform(train_x)
train_x_poly.shape
```

fit_transform takes our x values, and output a list of our data raised from power of 0 to power of 2 (since we set the degree of our polynomial to 2).

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \longrightarrow \begin{bmatrix} [1 & v_1 & v_1^2] \\ [1 & v_2 & v_2^2] \\ \vdots & \vdots & \vdots \\ [1 & v_n & v_n^2] \end{bmatrix}$$

in our example

$$\begin{bmatrix} 2. \\ 2.4 \\ 1.5 \\ \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} [1 & 2. & 4.] \\ [1 & 2.4 & 5.76] \\ [1 & 1.5 & 2.25] \\ \vdots & \vdots & \vdots \end{bmatrix}$$

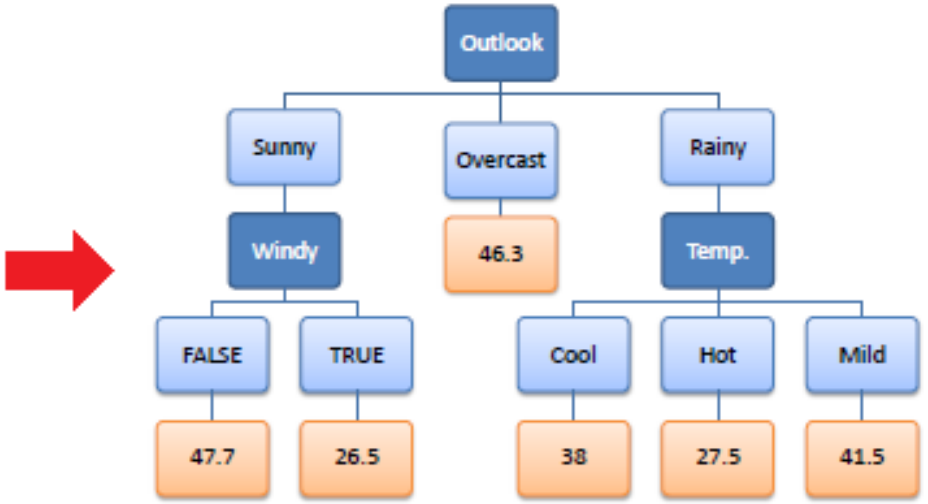
Now, we can deal with it as 'linear regression' problem. Therefore, this polynomial regression is considered to be a special case of traditional multiple linear regression. So, you can use the same mechanism as linear regression to solve such a problems.
so we can use **LinearRegression()** function to solve it:

```
clf = linear_model.LinearRegression()
train_y_ = clf.fit(train_x_poly, train_y)

# The coefficients
print ('Coefficients: ', clf.coef_)
print ('Intercept: ',clf.intercept_)
```

Decision tree regression

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

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	35
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



Decision tree regression observes features of an object and trains a model in the structure of a tree to predict data in the future to produce meaningful continuous output. Continuous output means that the output/result is not discrete, i.e., it is not represented just by a discrete, known set of numbers or values.

***Discrete output example*:** A weather prediction model that predicts whether or not there'll be rain in a particular day.

***Continuous output example*:** A profit prediction model that states the probable profit that can be generated from the sale of a product.

Code:

```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X, y)
```

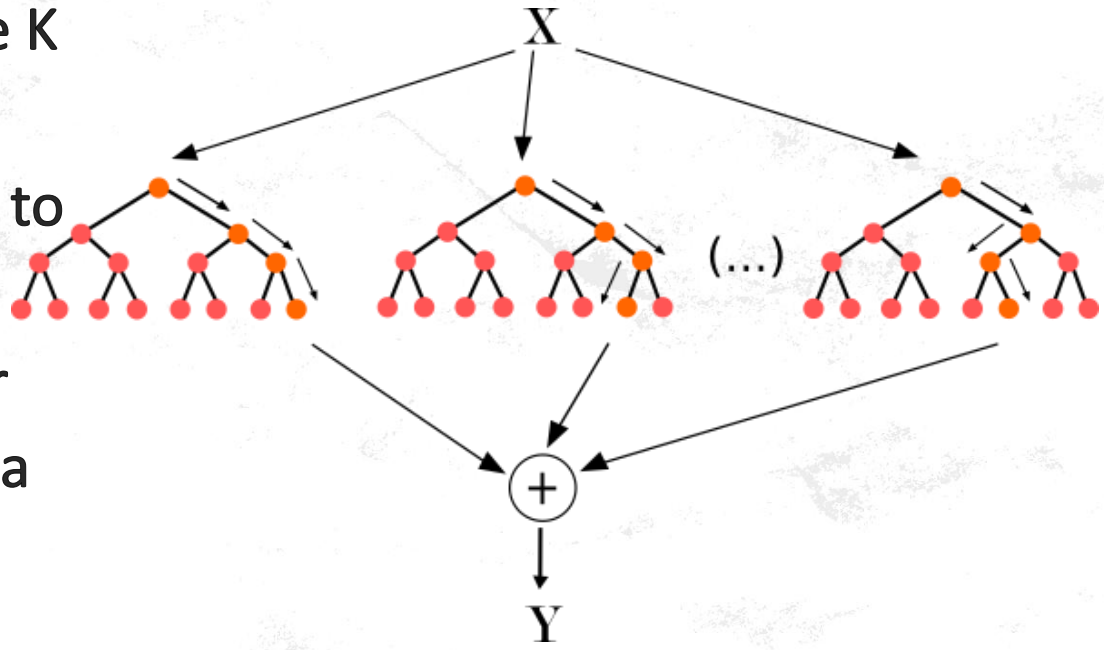
Random forest regression

The Random Forest is one of the most effective machine learning models for predictive analytics, making it an industrial workhorse for machine learning.

The random forest model is a type of additive model that makes predictions by combining decisions from a sequence of base models. Here, each base classifier is a simple decision tree. This broad technique of using multiple models to obtain better predictive performance is called model ensembling. In random forests, all the base models are constructed independently using a different subsample of the data

Approach :

- Pick at random K data points from the training set.
- Build the decision tree associated with those K data points.
- Choose the number Ntree of trees you want to build and repeat step 1 & 2.
- For a new data point, make each one of your Ntree trees predict the value of Y for the data point, and assign the new data point the average across all of the predicted Y values.



Code

```
# import the regressor
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0)

# fit the regressor with X and Y data
regressor.fit(X, y)
```

Pros and cons

Regression model	Pros	Cons
Linear regression	Works on any size of dataset, gives information about features.	The Linear regression assumptions.
Polynomial regression	Works on any size of dataset, works very well on non linear problems	Need to choose right polynomial degree for a. Good bias and trade off.
SVR	Easily adaptable, works very well on non linear problems, not biased by outliers	Compulsory to apply feature scaling, not well known ,more difficult to understand.
Decision tree recession	Interpretability ,no need for feature scaling ,works on both linear and non linear problems	Poor results on small datasets, overfitting can easily occur
Random forest regression	Powerful and accurate ,good performance many problems ,including non linear	No Interpretability , overfitting can easily occur, need to choose number of trees

LOGISTIC REGRESSION

In statistics, the logistic model is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc

Based on the number of categories, Logistic regression can be classified as:

binomial: Target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc.

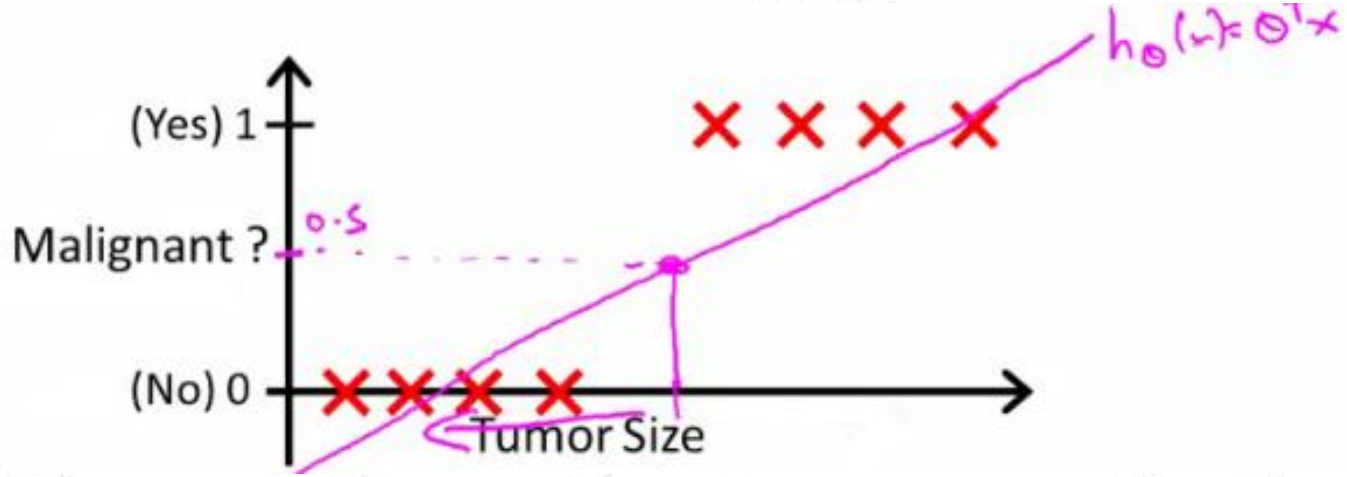
multinomial: Target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like “disease A” vs “disease B” vs “disease C”.

ordinal: It deals with target variables with ordered categories. For example, a test score can be categorized as:“very poor”, “poor”, “good”, “very good”. Here, each category can be given a score like 0, 1, 2, 3.

Start with **binary class problems**

How do we develop a classification algorithm?

- Tumour size vs malignancy (0 or 1)
- We *could* use linear regression
  - Then threshold the classifier output (i.e. anything over some value is yes, else no)
  - In our example below linear regression with thresholding seems to work



•We can see above this does a reasonable job of stratifying the data points into one of two classes
  - But what if we had a single Yes with a very small tumour
  - This would lead to classifying all the existing yeses as nos

•Another issues with linear regression
  - We know Y is 0 or 1
  - Hypothesis can give values large than 1 or less than 0

•So, logistic regression generates a value where is always either 0 or 1
  - Logistic regression is a **classification algorithm** - don't be confused

Hypothesis representation

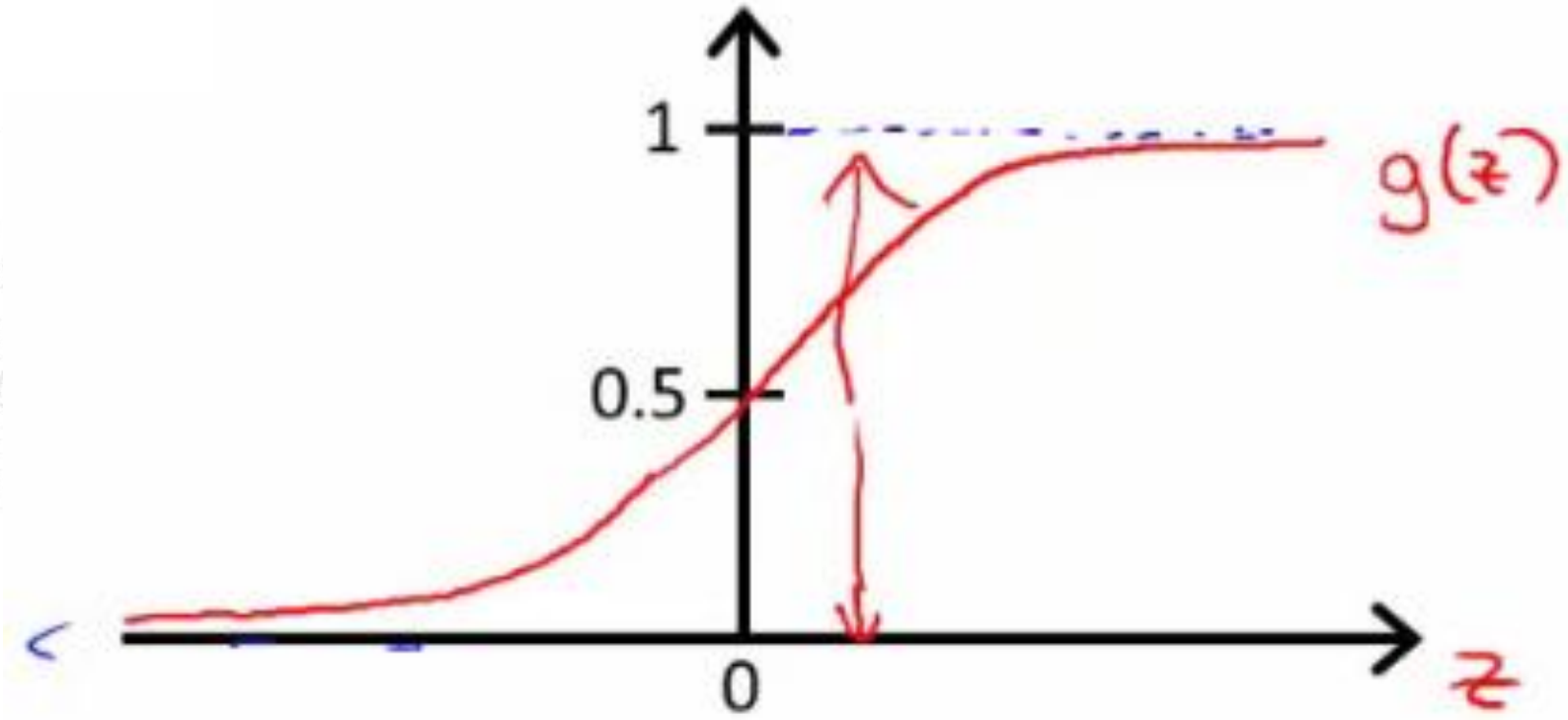
•What function is used to represent our hypothesis in classification

•We want our classifier to output values between 0 and 1
  - When using linear regression we did $h_\theta(x) = (\theta^T x)$
  - For classification hypothesis representation we do $h_\theta(x) = g((\theta^T x))$
    - Where we define g(z)
      - z is a real number
      - This is the **sigmoid function**, or the **logistic function**
    - If we combine these equations we can write out the hypothesis as

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

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

- How does the sigmoid function look like



- Crosses 0.5 at the origin, then flattens out
    - Asymptotes at 0 and 1

• Interpreting hypothesis output

When our hypothesis ($h_\theta(x)$) outputs a number, we treat that value as the estimated probability that y=1 on input x

- Example
  - If X is a feature vector with $x_0 = 1$ (as always) and x_1 = tumourSize
  - $h_\theta(x) = 0.7$
    - Tells a patient they have a 70% chance of a tumor being malignant

  $h_\theta(x) = P(y=1|x\ ;\ \theta)$
- What does this mean?
  - Probability that y=1, given x, parameterized by θ

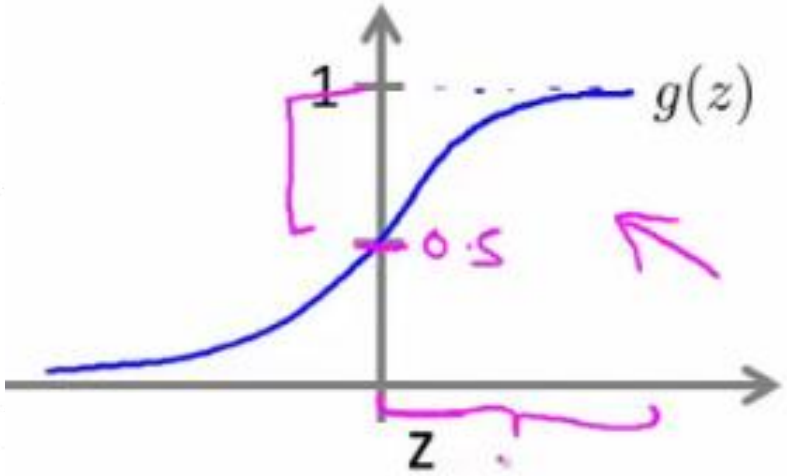
• Since this is a binary classification task we know y = 0 or 1

- So the following must be true
  - $P(y=1|x\ ;\ \theta) + P(y=0|x\ ;\ \theta) = 1$
  - $P(y=0|x\ ;\ \theta) = 1 - P(y=1|x\ ;\ \theta)$

Decision boundary

- This gives a better sense of what the hypothesis function is computing
  - One way of using the sigmoid function is;
    - When the probability of y being 1 is greater than 0.5 then we can predict y = 1
    - Else we predict y = 0
  - When is it exactly that $h_\theta(x)$ is greater than 0.5?
    - Look at sigmoid function
      - g(z) is greater than or equal to 0.5 when z is greater than or equal to 0

- So if z is positive, g(z) is greater than 0.5
  - $z = (\theta^T x)$
- So when
  - $\theta^T x >= 0$
- Then $h_\theta >= 0.5$

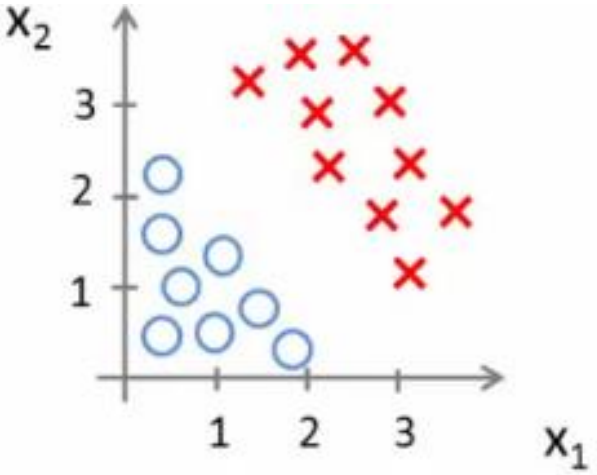


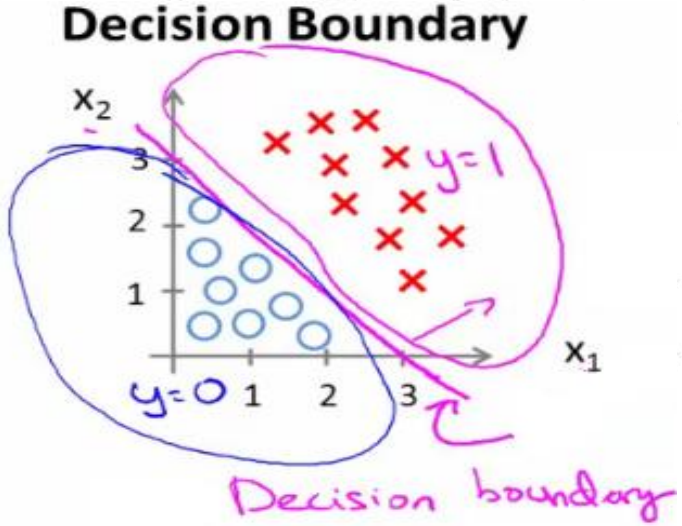
- So what we've shown is that the hypothesis predicts y = 1 when $\theta^T x >= 0$
  - The corollary of that when $\theta^T x <= 0$ then the hypothesis predicts y = 0
  - Let's use this to better understand how the hypothesis makes its predictions

Consider,

$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

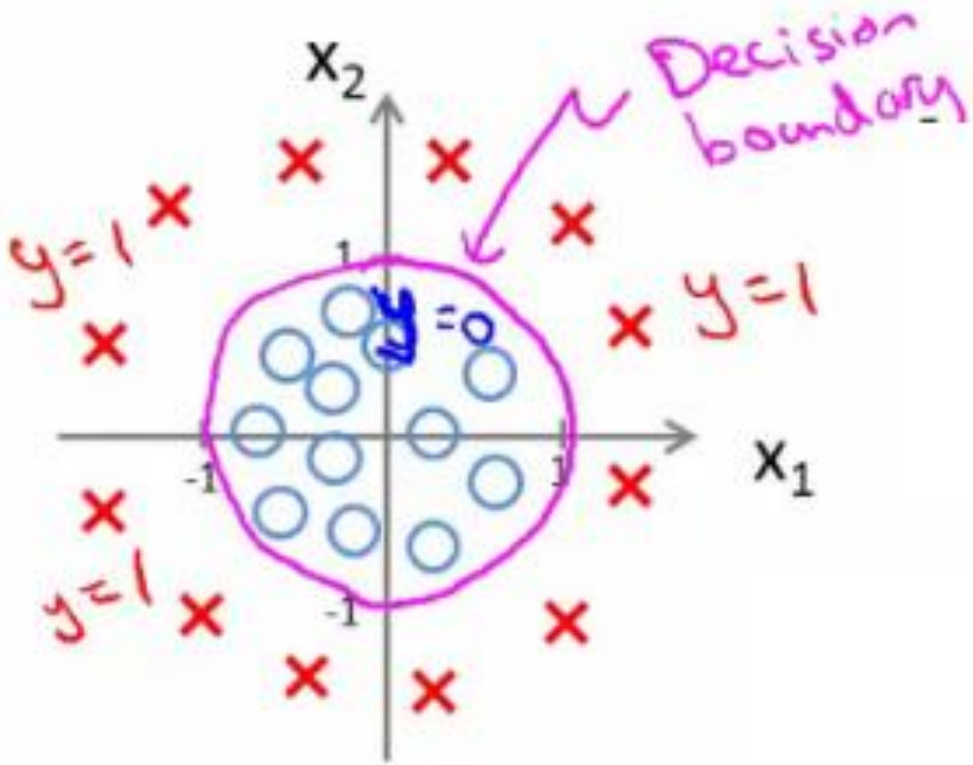
- So, for example
  - $\theta_0 = -3$
  - $\theta_1 = 1$
  - $\theta_2 = 1$
- So our parameter vector is a column vector with the above values
  - So, θ^T is a row vector = [-3,1,1]
- What does this mean?
  - The z here becomes $\theta^T x$
  - We predict "y = 1" if
    - $-3x_0 + 1x_1 + 1x_2 >= 0$
    - $-3 + x_1 + x_2 >= 0$
- We can also re-write this as
  - If ($x_1 + x_2 >= 3$) then we predict y = 1
  - If we plot
    - $x_1 + x_2 = 3$ we graphically plot our **decision boundary**





$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_3 x_1^2 + \theta_4 x_2^2)$

- Say θ^T was [-1,0,0,1,1] then we say;
- Predict that "y = 1" *if*
  - $-1 + x_1^2 + x_2^2 >= 0$
    or
  - $x_1^2 + x_2^2 >= 1$
- If we plot $x_1^2 + x_2^2 = 1$



Cost function for logistic regression

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

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

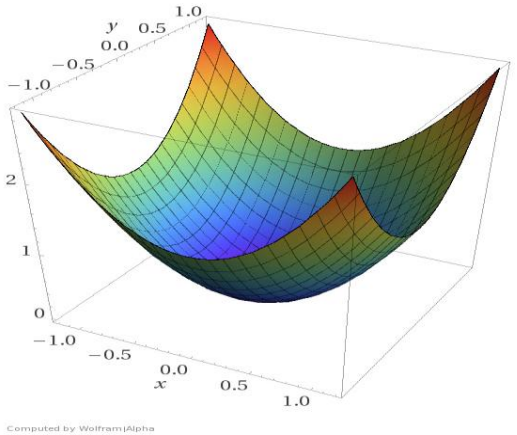
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

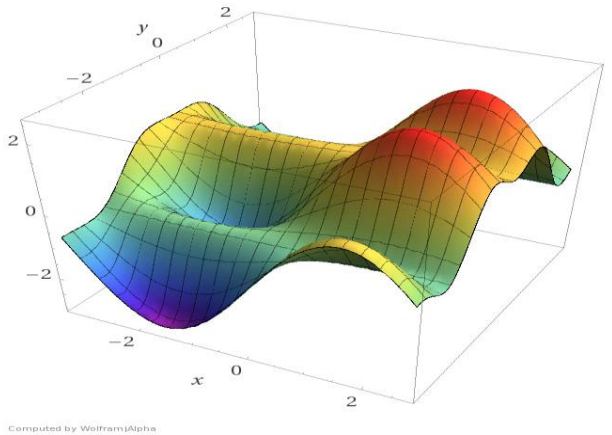
Linear regression uses the following function to determine θ

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

• If we use this function for logistic regression this is a **non-convex function** for parameter optimization Could work !!!

• What do we mean by non convex?
  - We have some function - J(θ) - for determining the parameters
  - Our hypothesis function has a non-linearity (sigmoid function of $h_\theta(x)$)
    - This is a complicated non-linear function
  - If you take $h_\theta(x)$ and plug it into the Cost() function, and them plug the Cost() function into J(θ) and plot J(θ) we find many local optimum -> *non convex function*
  - Why is this a problem
    - Lots of local minima mean gradient descent may not find the global optimum - may get stuck in a global minimum
  - We would like a convex function so if you run gradient descent you converge to a global minimum





A convex logistic regression cost function

•To get around this we need a different, convex Cost() function which means we can apply gradient descent

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

The above two functions can be compressed into a single function i.e.

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

Gradient Descent

Now the question arises, how do we reduce the cost value. Well, this can be done by using **Gradient Descent.** The main goal of Gradient descent is to **minimize the cost value.** i.e. min J(***ϑ***).
Now to minimize our cost function we need to run the gradient descent function on each parameter i.e.

$$\theta j := \theta j - \alpha \frac{\partial}{\partial \theta j} J(\theta)$$

Want $\min_\theta J(\theta)$:

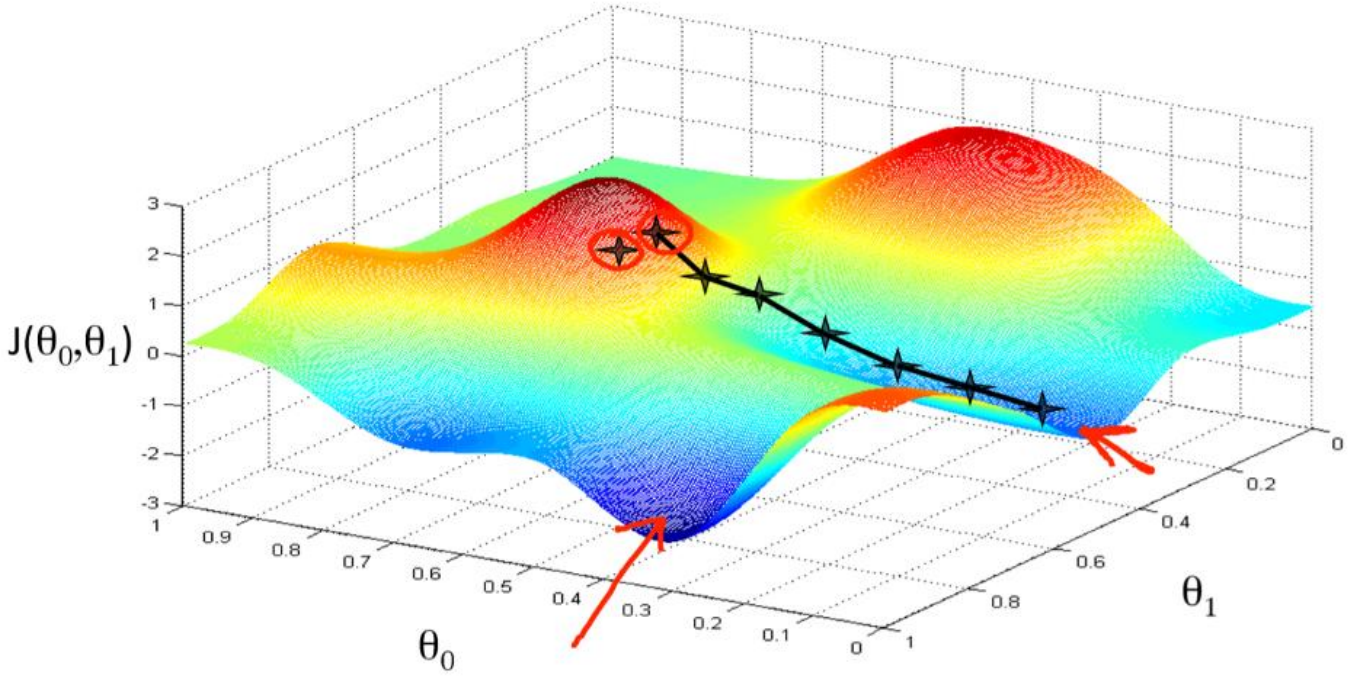
Repeat $\{$

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

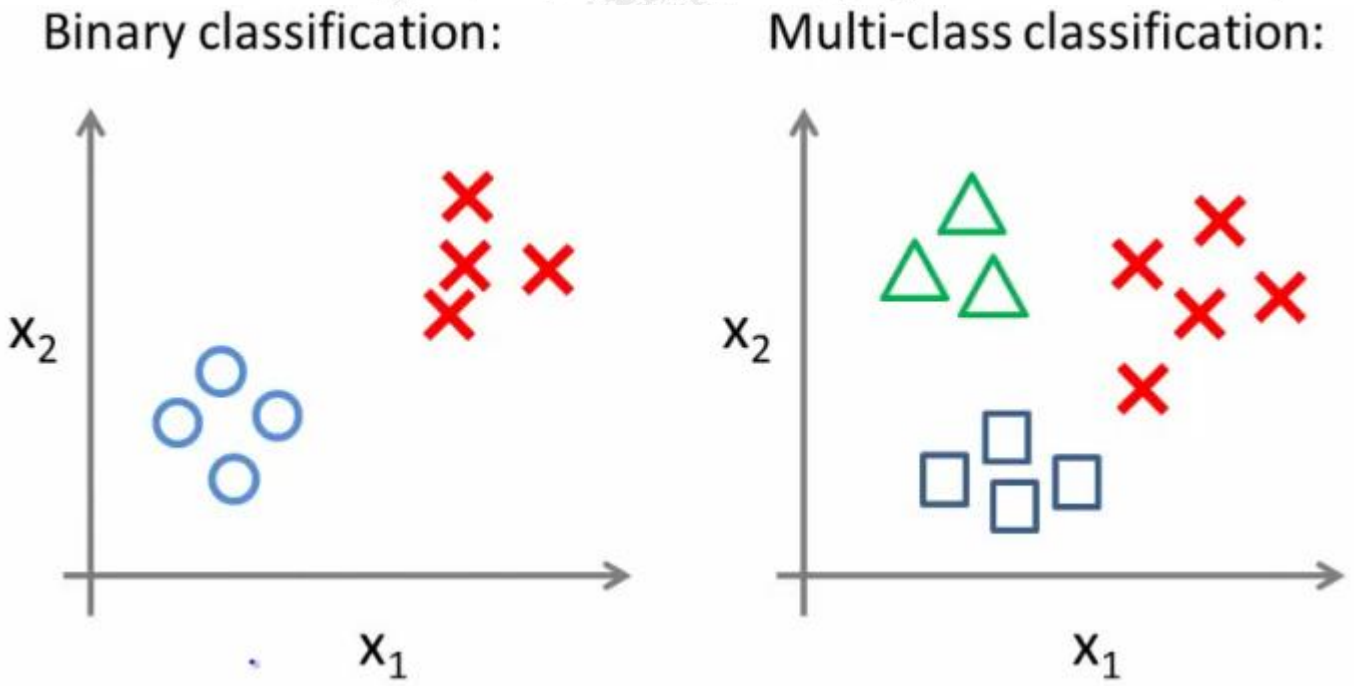
$\}$

Gradient descent has an analogy in which we have to imagine ourselves at the top of a mountain valley and left stranded and blindfolded, our objective is to reach the bottom of the hill. Feeling the slope of the terrain around you is what everyone would do. Well, this action is analogous to calculating the gradient descent, and taking a step is analogous to one iteration of the update to the parameters.



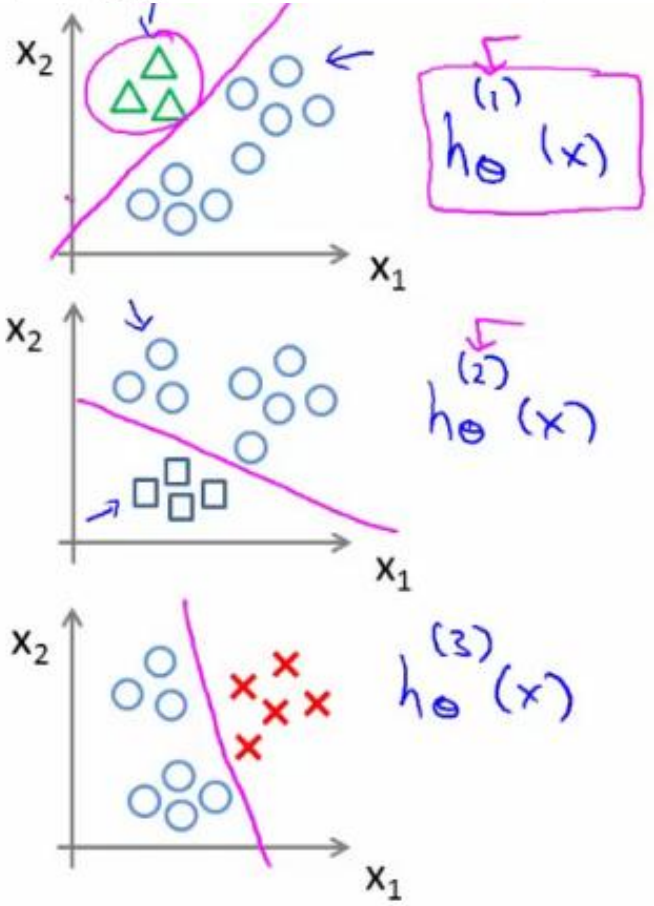
Multiclass classification problems

- Getting logistic regression for multiclass classification using **one vs. all**
- Multiclass - more than yes or no (1 or 0)
    - Classification with multiple classes for assignment



- Given a dataset with three classes, how do we get a learning algorithm to work?
  - Use one vs. all classification make binary classification work for multiclass classification

- **One vs. all classification**
  - Split the training set into three separate binary classification problems
    - i.e. create a new fake training set
      - Triangle (1) vs crosses and squares (0) $h_\theta^1(x)$
        - $P(y=1 \mid x_1; \theta)$
      - Crosses (1) vs triangle and square (0) $h_\theta^2(x)$
        - $P(y=1 \mid x_2; \theta)$
      - Square (1) vs crosses and square (0) $h_\theta^3(x)$
        - $P(y=1 \mid x_3; \theta)$



- Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that y = i
- On a new input, *x* to make a prediction, pick the class *i* that maximizes the probability that $h_\theta^{(i)}(x) = 1$

K-Nearest Neighbours

X: Independent variable

Y: Dependent variable

	region	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	44	1	9	64	4	5	0	0	2	1
1	3	33	1	7	136	5	5	0	0	6	4
2	3	52	1	24	116	1	29	0	1	2	3
3	2	33	0	12	33	2	0	0	1	1	1
4	2	30	1	9	30	1	2	0	0	4	3
5	2	39	0	17	78	2	16	0	1	1	3
6	3	22	1	2	19	2	4	0	1	5	2
7	2	35	0	5	76	2	10	0	0	3	4
8	3	50	1	7	166	4	31	0	0	5	?

Value	Label
1	Basic Service
2	E-Service
3	Plus Service
4	Total Service

This algorithm classifies cases based on their similarity to other cases.

In K-Nearest Neighbors, data points that are near each other are said to be neighbors.

K-Nearest Neighbors is based on this paradigm.

Similar cases with the same class labels are near each other.
Thus, the distance between two cases is a measure of their dissimilarity.
There are different ways to calculate the similarity or conversely,
the distance or dissimilarity of two data points.
For example, this can be done using Euclidean distance.

the K-Nearest Neighbors algorithm works as follows.

- pick a value for K.
- calculate the distance from the new case hold out from each of the cases in the dataset.
- search for the K-observations in the training data that are nearest to the measurements of the unknown data point.
- predict the response of the unknown data point using the most popular response value from the K-Nearest Neighbors.

There are two parts in this algorithm that might be a bit confusing.

- First, how to select the correct K
- second, how to compute the similarity between cases,

Let's first start with the second concern.

Calculating the similarity/distance in a multi-dimensional space

Customer 1		
Age	Income	Education
54	190	3

Customer 2		
Age	Income	Education
50	200	8

$$\text{Dis}\,(x_1, x_2) = \sqrt{\sum_{i=0}^{n}(x_{1i} - x_{2i})^2}$$

$$= \sqrt{(54-50)^2+(190-200)^2+(3-8)^2} = 11.87$$

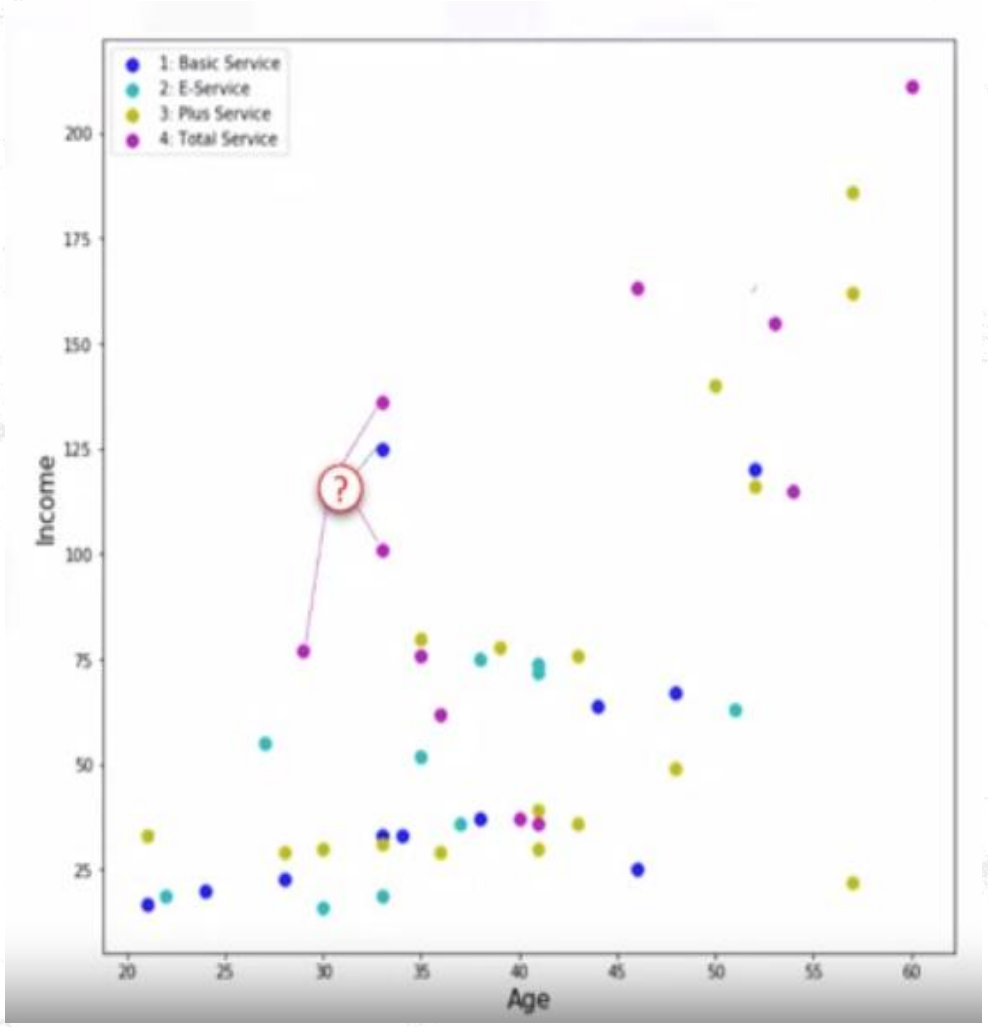
How to select the correct K

As mentioned, K and K-Nearest Neighbors is the number of nearest neighbors to examine.

It is supposed to be specified by the user.
So, how do we choose the right K?

Assume that we want to find the class of
the customer noted as question mark on the chart.

What happens if we choose a very low value of K?
Let's say, K equals one.
The first nearest point would be blue,
which is class one.
This would be a bad prediction,
since more of the points around it are magenta or class four.



In fact, since its nearest neighbor is blue we can say that we capture the noise in the data or we chose one of the points that was an anomaly in the data.

A low value of K causes a highly complex model as well, which might result in overfitting of the model.

It means the prediction process is not generalized enough to be used for out-of-sample cases.

Out-of-sample data is data that is outside of the data set used to train the model.

In other words, it cannot be trusted to be used for prediction of unknown samples. It's important to remember that overfitting is bad, as we want a general model that works for any data, not just the data used for training.

Now, on the opposite side of the spectrum, if we choose a very high value of K such as K equals 20,
then the model becomes overly generalized.

So, how can we find the best value for K?

The general solution is to reserve a part of your data for testing the accuracy of the model.
Once you've done so, choose K equals one and then use the training part for modeling and calculate the accuracy of prediction using all samples in your test set.

Repeat this process increasing the K and see which K is best for your model.
For example, in our case,

K equals four will give us the best accuracy.

Advantages of KNN

1. No Training Period: KNN is called Lazy Learner (Instance based learning). It does not learn anything in the training period. It does not derive any discriminative function from the training data. In other words, there is no training period for it. It stores the training dataset and learns from it only at the time of making real time predictions. This makes the KNN algorithm much faster than other algorithms that require training e.g. SVM, Linear Regression etc.

2. Since the KNN algorithm requires no training before making predictions, new data can be added seamlessly which will not impact the accuracy of the algorithm.

3. KNN is very easy to implement. There are only two parameters required to implement KNN i.e. the value of K and the distance function (e.g. Euclidean or Manhattan etc.)

Disadvantages of KNN

1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.

2. Does not work well with high dimensions: The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

3.Sensitive to noisy data, missing values and outliers: KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.66666667 0.33333333]]
```

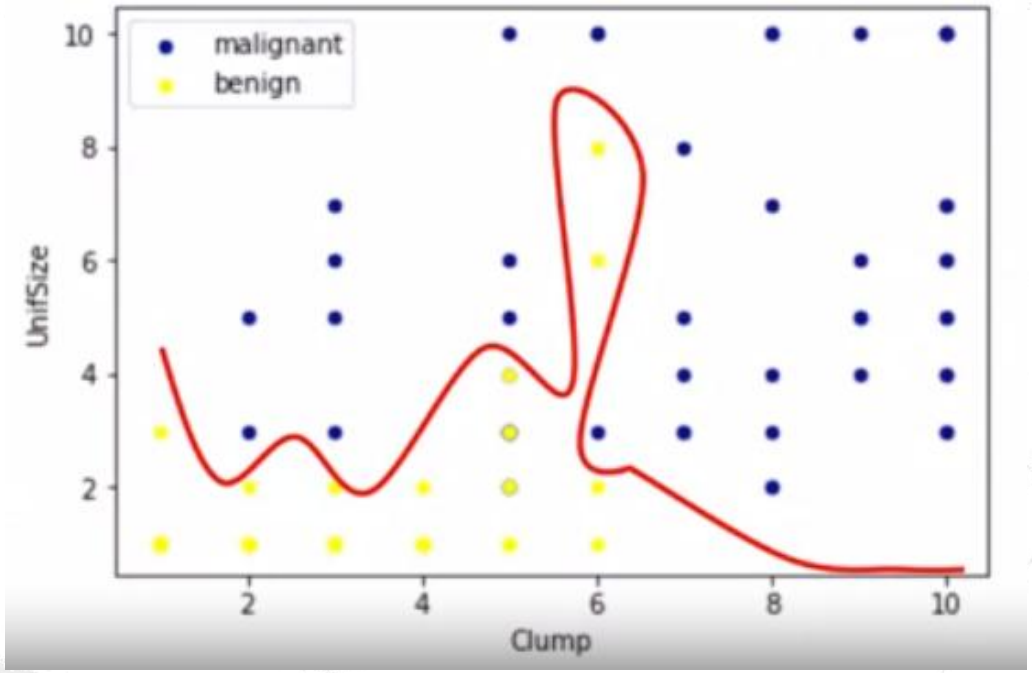
SUPPORT VECTOR MACHINE(SVM)

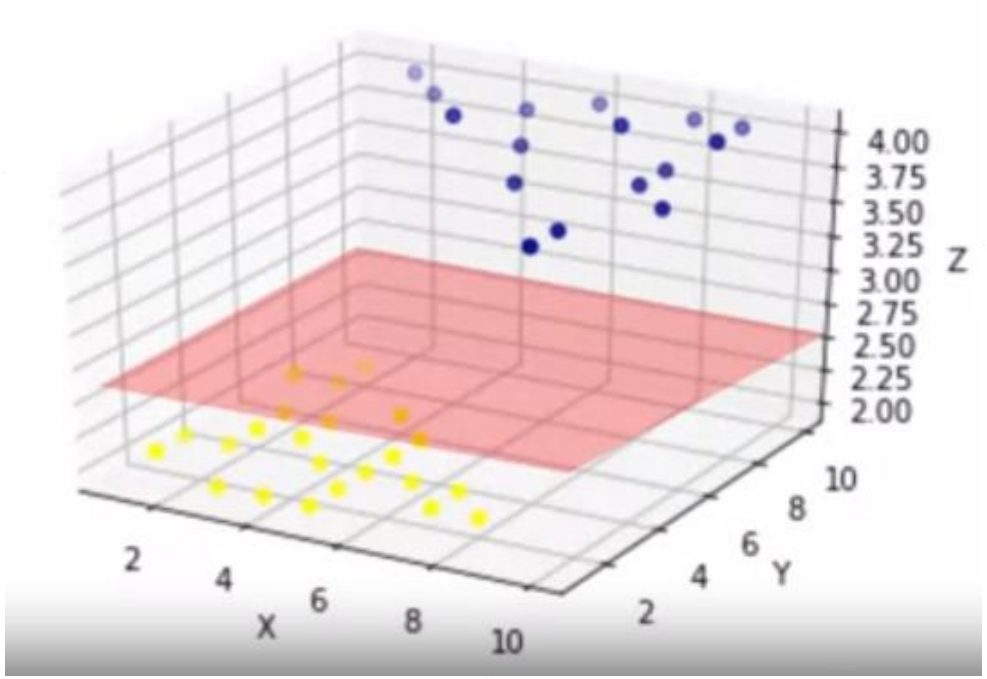
A Support Vector Machine is a supervised algorithm that can classify cases by finding a separator.

SVM works by first mapping data to a high dimensional feature space so that data points can be categorized, even when the data are not linearly separable.

Then, a separator is estimated for the data. The data should be transformed in such a way that a separator could be drawn as a hyperplane.

Clump	UnifSize	UnifShape	MargAdh	SingEpiSize	BareNuc	BlandChrom	NormNucl	Mit	Class
5	1	1	1	2	1	3	1	1	benign
5	4	4	5	7	10	3	2	1	benign
3	1	1	1	2	2	3	1	1	malignant
6	8	8	1	3	4	3	7	1	benign
4	1	1	3	2	1	3	1	1	benign
8	10	10	8	7	10		7	1	malignant
1	1	1	1	2	10	3	1	1	benign
2	1	2	H	2	1	3	1	1	benign
2	1	1	1	2	1	1	1	5	benign
4	2	1	1	2	1	2	1	1	benign





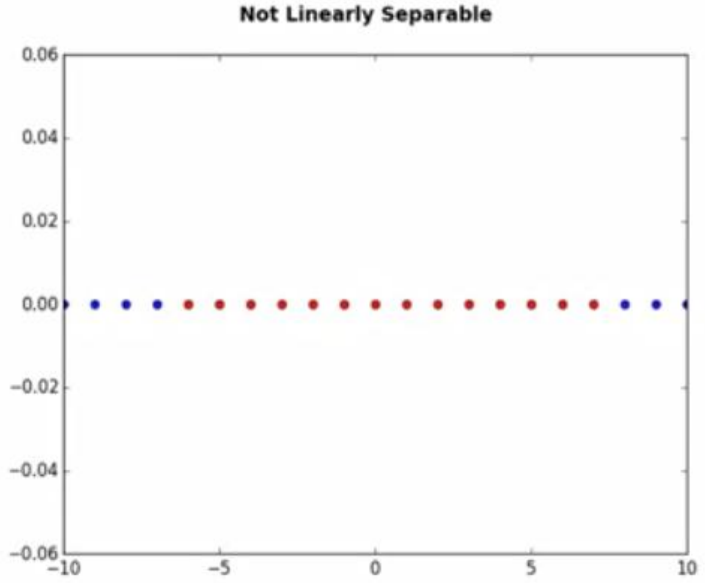
Therefore, the SVM algorithm outputs an optimal hyperplane that categorizes new examples.

DATA TRANFORMATION

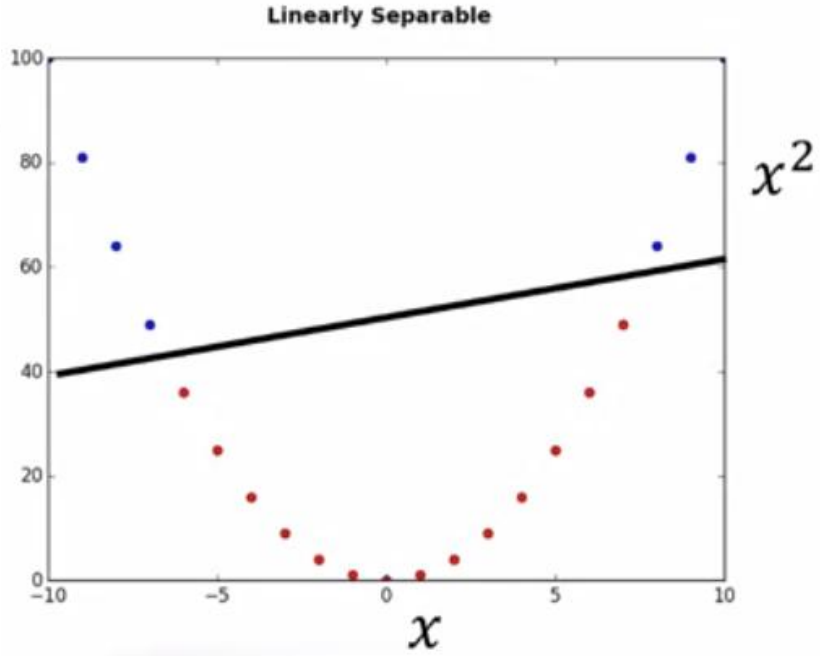
For the sake of simplicity, imagine that our dataset is one-dimensional data.
This means we have only one feature x.

As you can see, it is not linearly separable.

Well, we can transfer it into a two-dimensional space. For example, you can increase the dimension of data by mapping x into a new space using a function with outputs x and x squared.



$$\phi(x) = [x, x^2]$$



Basically, mapping data into a higher-dimensional space is called, kernelling.
The mathematical function used for the transformation is known as the kernel function, and can be of different types,such as linear, polynomial, Radial Basis Function, or RBF, and sigmoid.

SVMs are based on the idea of finding a hyperplane that best divides a data set into two classes as shown here.

As we're in a two-dimensional space, you can think of the hyperplane as a line that linearly separates the blue points from the red points.

ADVANTAGES

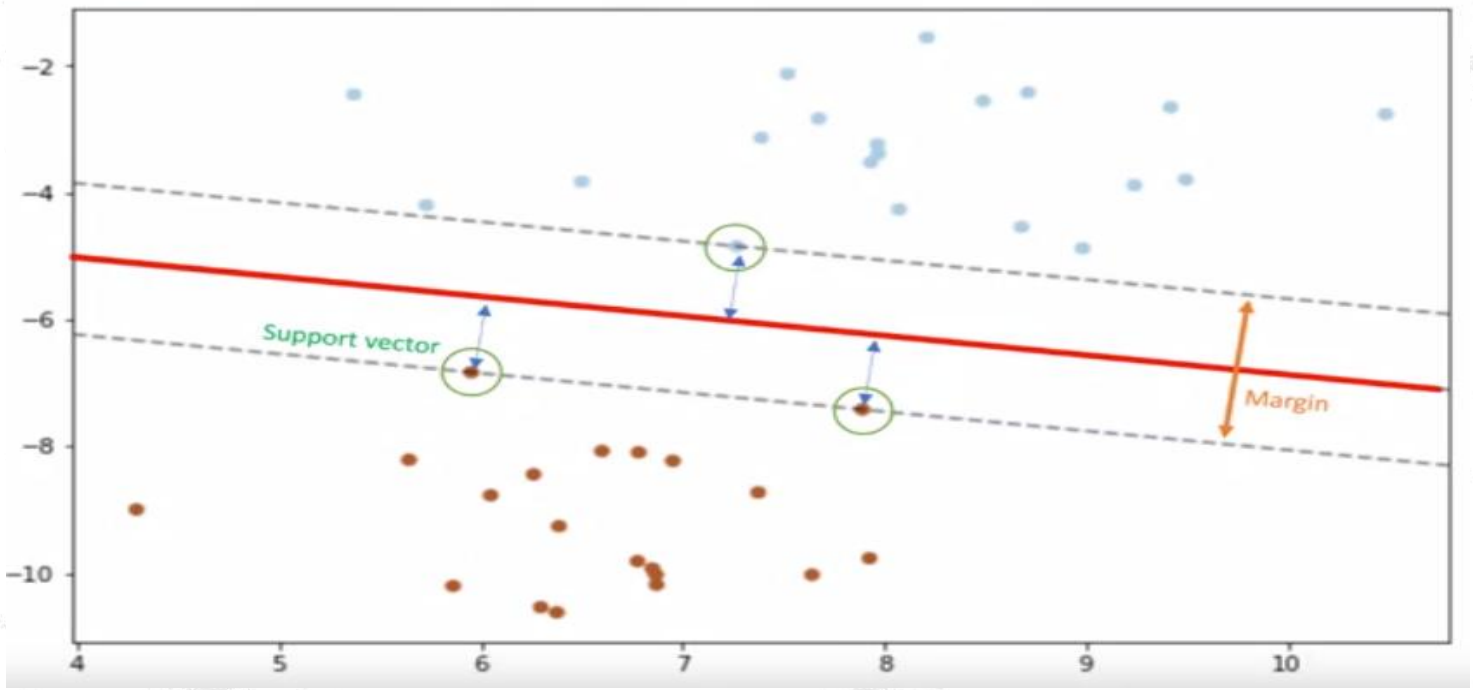
- Accurate in high dimension place
- Memory efficient

DISADVANTAGES

- Small datasets
- Prone to overfitting

APPLICATIONS

- Image Recognition
- Spam detection



Naive Bayes Classifiers

COLLECTION OF CLASSIFICATION ALGORITHMS

Principle of Naive Bayes Classifier:

- A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

- Bayes theorem can be rewritten as:

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

- 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.

	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

- We classify whether the day is suitable for playing golf, given the features of the day. The columns represent these features and the rows represent individual entries. If we take the first row of the dataset, we can observe that is not suitable for playing golf if the outlook is rainy, temperature is hot, humidity is high and it is not windy. We make two assumptions here, one as stated above we consider that these predictors are independent. That is, if the temperature is hot, it does not necessarily mean that the humidity is high. Another assumption made here is that all the predictors have an equal effect on the outcome. That is, the day being windy does not have more importance in deciding to play golf or not.
- According to this example, Bayes theorem can be rewritten as:

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

- The variable **y** is the class variable(play golf), which represents if it is suitable to play golf or not given the conditions.
Variable **X** represent the parameters/features.

In our case, the class variable(**y**) has only two outcomes, yes or no. There could be cases where the classification could be multivariate. Therefore, we need to find the class **y** with maximum probability.

- **X** is given as, $X = (x_1, x_2, x_3, ..., x_n)$
- Here x_1,x_2....x_n represent the features, i.e they can be mapped to outlook, temperature, humidity and windy. By substituting for **X** and expanding using the chain rule we get,

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

- Now, you can obtain the values for each by looking at the dataset and substitute them into the equation. For all entries in the dataset, the denominator does not change, it remain static. Therefore, the denominator can be removed and a proportionality can be introduced.

- In our case, the $P(y|x_1, ..., x_n) \propto P(y) \prod_{i=1}^{n} P(x_i|y)$ no. There could be cases where the classification could be multivariate. Therefore, we need to find the class **y** with maximum probability.

- Using the abov $y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$ redictors.

- We need to find $P(x_i \mid y_j)$ for each x_i in X and y_j in y. All these calculations have been demonstrated in the tables below:

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_i 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):

```
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:

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

and

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

Now, since

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

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

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

and

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

Since

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

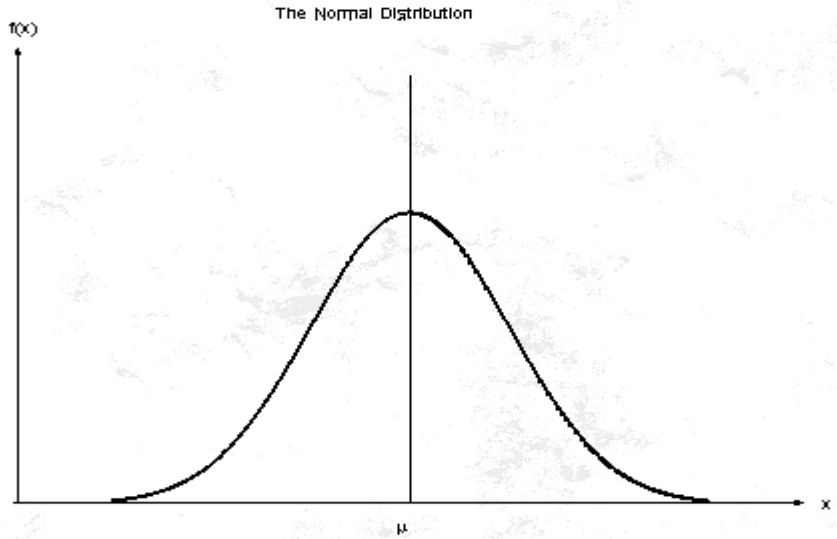
So, prediction that golf would be played is 'Yes'.

Types of Naive Bayes Classifier:

- **Multinomial Naive Bayes:** This is mostly used for document classification problem, i.e whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present.

- **Bernoulli Naive Bayes:** This is similar to the multinomial naive bayes but the predictors are boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

- **Gaussian Naive Bayes:** When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.



Gaussian Distribution(Normal Distribution)

Conclusion:

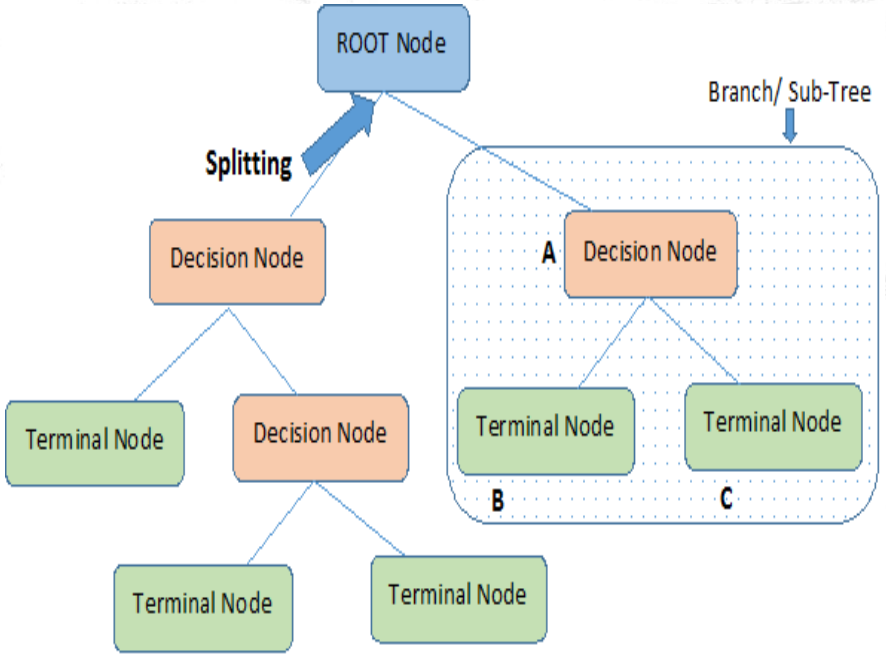
Naive Bayes algorithms are mostly used in sentiment analysis, spam filtering, recommendation systems etc. They are fast and easy to implement but their biggest disadvantage is that the requirement of predictors to be independent. In most of the real life cases, the predictors are dependent, this hinders the performance of the classifier.

Decision Tree

CLASSIFICATION ALGORITHM

- Decision tree algorithm falls under the category of supervised learning. They can be used to solve both regression and classification problems..
- Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.
- We can represent any boolean function on discrete attributes using the decision tree.
- **Types of decision trees**
- Categorical Variable Decision Tree: Decision Tree which has categorical target variable then it called as categorical variable decision tree.
- Continuous Variable Decision Tree: Decision Tree which has continuous target variable then it is called as Continuous Variable Decision Tree.

- **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
- **Splitting:** It is a process of dividing a node into two or more sub-nodes.
- **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
- **Leaf/ Terminal Node:** Nodes with no children (no further split) is called Leaf or Terminal node.
- **Pruning:** When we reduce the size of decision trees by removing nodes (opposite of Splitting), the process is called pruning.
- **Branch / Sub-Tree:** A sub section of decision tree is called branch or sub-tree.
- **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as sub-nodes are the child of parent node.

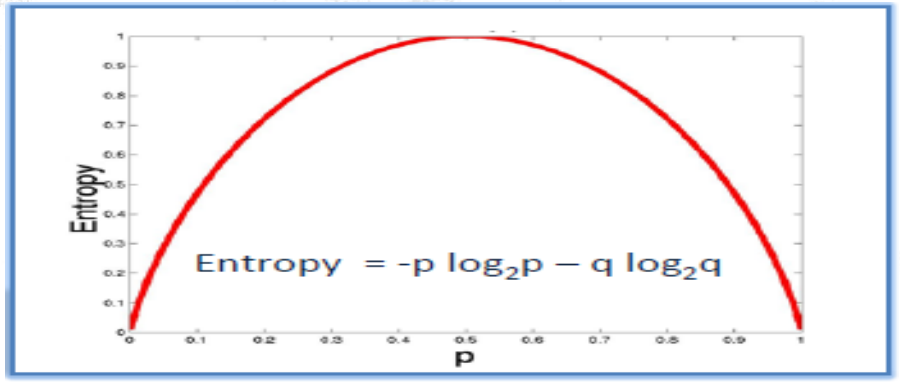


Algorithm

- **Algorithms used in decision trees:**
- ID3
- Gini Index
- Chi-Square
- Reduction in Variance
- The core algorithm for building decision trees is called **ID3.** Developed by J. R. Quinlan and it uses *Entropy* and *Information Gain* to construct a decision tree.
- The ID3 algorithm begins with the original set S as the root node. On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy H(S)or information gain IG(S) of that attribute. It then selects the attribute which has the smallest entropy (or largest information gain) value. The set S is then split or partitioned by the selected attribute to produce subsets of the data

Entropy

- Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Decision tree algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

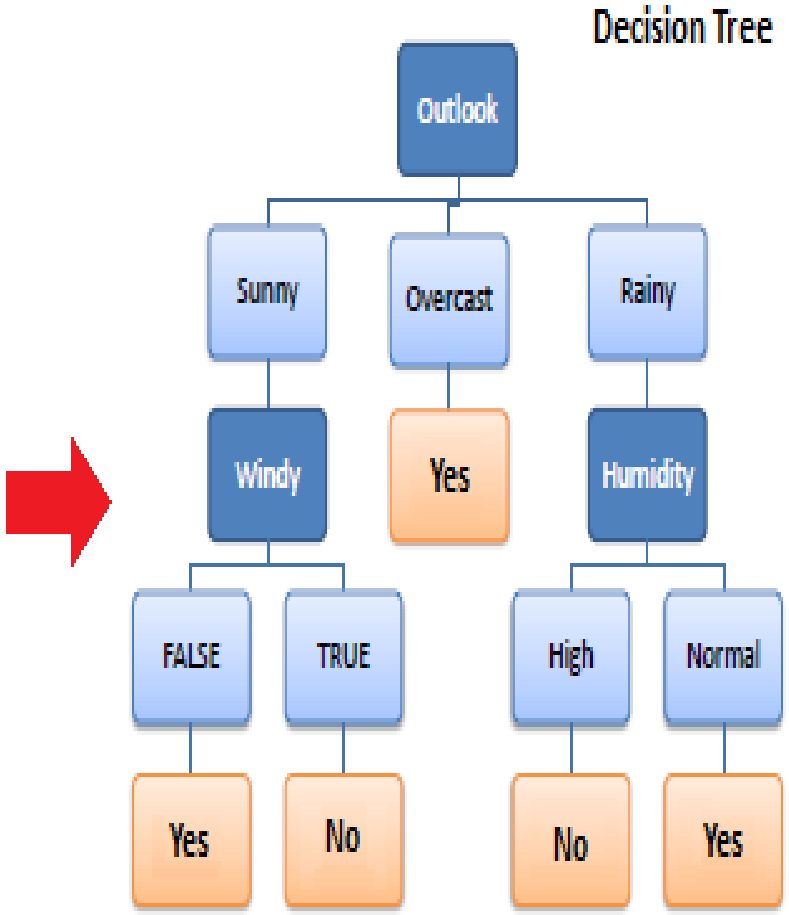


Entropy = $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$

Example:

Predictors | Target

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



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

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

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

b) Entropy using the frequency table of two attributes:

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

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

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

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

= 0.693

Information gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

Step 1: Calculate entropy of the target.

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

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

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

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

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

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

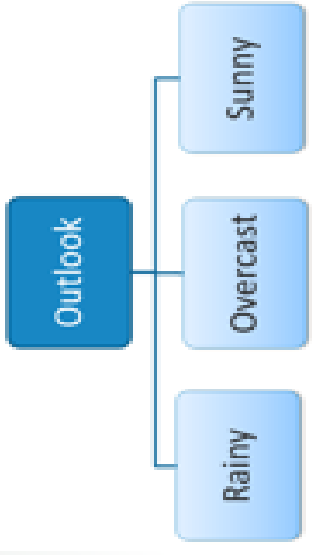
$$Gain(T, X) = Entropy(T) - Entropy(T, X)$$

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

= 0.940 – 0.693 = 0.247

- *Step 3*: Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

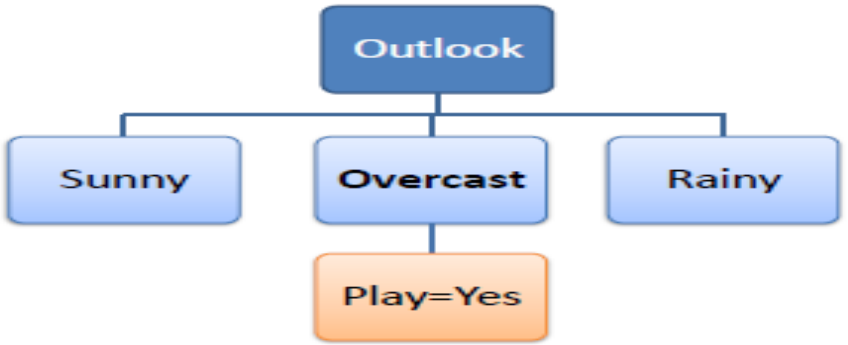
★		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			



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

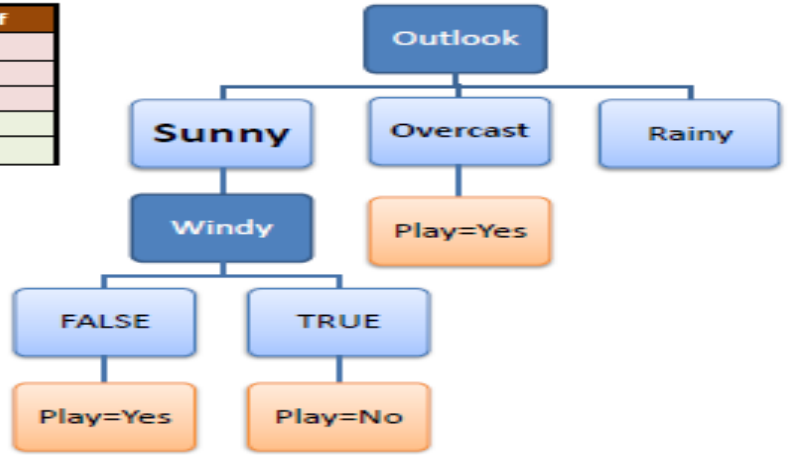
- *Step 4a*: A branch with entropy of 0 is a leaf node

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



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

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



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

- Decision Tree to Decision Rules

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

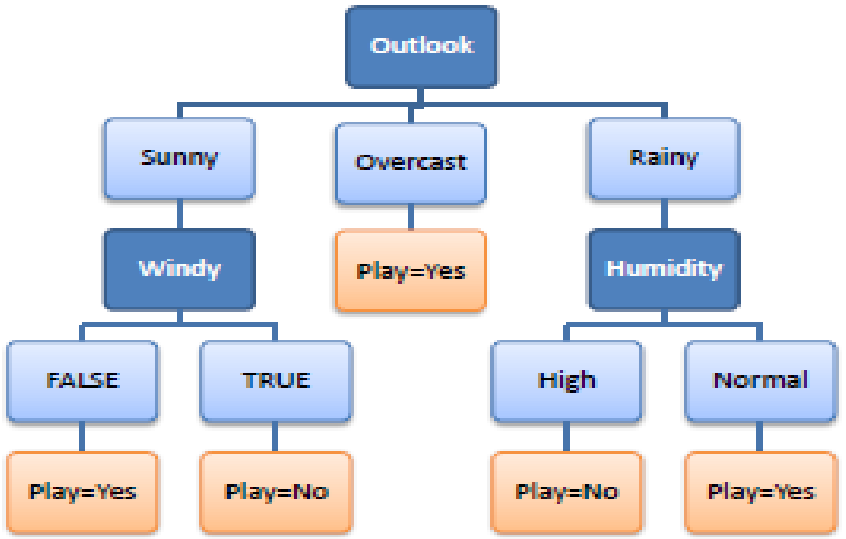
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

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

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

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

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



Limitations to Decision Trees

- Decision trees tend to have high variance when they utilize different training and test sets of the same data, since they tend to overfit on training data. This leads to poor performance on unseen data. Unfortunately, this limits the usage of decision trees in predictive modeling.
- To overcome these problems we use ensemble methods, we can create models that utilize underlying(weak) decision trees as a foundation for producing powerful results and this is done in Random Forest Algorithm

Random forest

Definition:

Random forest algorithm is a supervised classification algorithm Based on Decision Trees, also known as random decision forests, are a popular ensemble method that can be used to build predictive models for both classification and regression problems.

- Ensemble we mean(In Random Forest Context), Collective Decisions of Different Decision Trees. In RFT(Random Forest Tree), we make a prediction about the class, not simply based on One Decision Trees, but by an (almost) Unanimous Prediction, made by '**K**' Decision Trees.
- **Construction**:
- 'K' Individual Decision Trees are made from given Dataset, by randomly dividing the Dataset and the Feature Subspace by process called as **Bootstrap Aggregation**(Bagging), which is process of random selection with replacement. Generally 2/3rd of the Dataset (row-wise 2/3rd) is selected by bagging, and On that Selected Dataset we perform what we call is Attribute Bagging.

- Now **Attribute Bagging** is done to select 'm' features from given M features,(this Process is also called Random Subspace Creation.) Generally value of 'm' is square-root of M. Now we select say, 10 such values of m, and then Build 10 Decision Trees based on them, and test the 1/3rd remaining Dataset on these(10 Decision Trees).We would then Select the Best Decision Tree out of this. And Repeat the whole Process 'K' times again to build such 'K' decision trees.
- **Classification:**
- Prediction in Random Forest (a collection of 'K' Decision Trees) is truly ensemble ie, For Each Decision Tree, Predict the class of Instance and then return the class which was predicted the most often.

Using Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier //Importing library
TRAIN_DIR = "../train-mails"
TEST_DIR = "../test-mails"
dictionary = make_Dictionary(TRAIN_DIR)
print "reading and processing emails from file."
features_matrix, labels = extract_features(TRAIN_DIR)
test_feature_matrix, test_labels = extract_features(TEST_DIR)
model = RandomForestClassifier() //Creating model
print "Training model."
model.fit(features_matrix, labels) //training model
predicted_labels = model.predict(test_feature_matrix)
print "FINISHED classifying. accuracy score : "
print accuracy_score(test_labels, predicted_labels) //Predicting
```

we will get accuracy around 95.7%.

Parameters

- Lets understand and try with some of the tuning parameters.
- **n_estimators :** Number of trees in forest. Default is 10.
- **criterion:** “gini” or “entropy” same as decision tree classifier.
- **min_samples_split:** minimum number of working set size at node required to split. Default is 2.
- Play with these parameters by changing values individually and in combination and check if you can improve accuracy.
- trying following combination and obtained the accuracy as shown in next slide image .

Test Case	n_estimators	criterion	min_samples_split	accuracy
1	30	-	-	97.6
2	50	-	-	97.6
3	100	-	-	97.6
4	-	gini	-	96.9
5	-	entropy	-	98.07
6	-	-	10	95.7
7	-	-	20	95.7
8	-	-	50	97.3
9	30	entropy	2	97.3
10	30	entropy	10	98.07
11	50	entropy	10	97.6

Final Thoughts

- Random Forest Classifier being ensembled algorithm tends to give more accurate result. This is because it works on principle,
- Number of weak estimators when combined forms strong estimator.
- Even if one or few decision trees are prone to a noise, overall result would tend to be correct. Even with small number of estimators = 30 it gave us high accuracy as 97%.

Clustering

UNSUPERVISED LEARNING

Clustering

- A cluster is a subset of data which are similar.
- Clustering (also called unsupervised learning) is the process of dividing a dataset into groups such that the members of each group are as similar (close) as possible to one another, and different groups are as dissimilar (far) as possible from one another.
- Generally, it is used as a process to find meaningful structure, generative features, and groupings inherent in a set of examples.
- Clustering can uncover previously undetected relationships in a dataset. There are many applications for cluster analysis. For example, in business, cluster analysis can be used to discover and characterize customer segments for marketing purposes and in biology, it can be used for classification of plants and animals given their features.

Clustering Algorithms

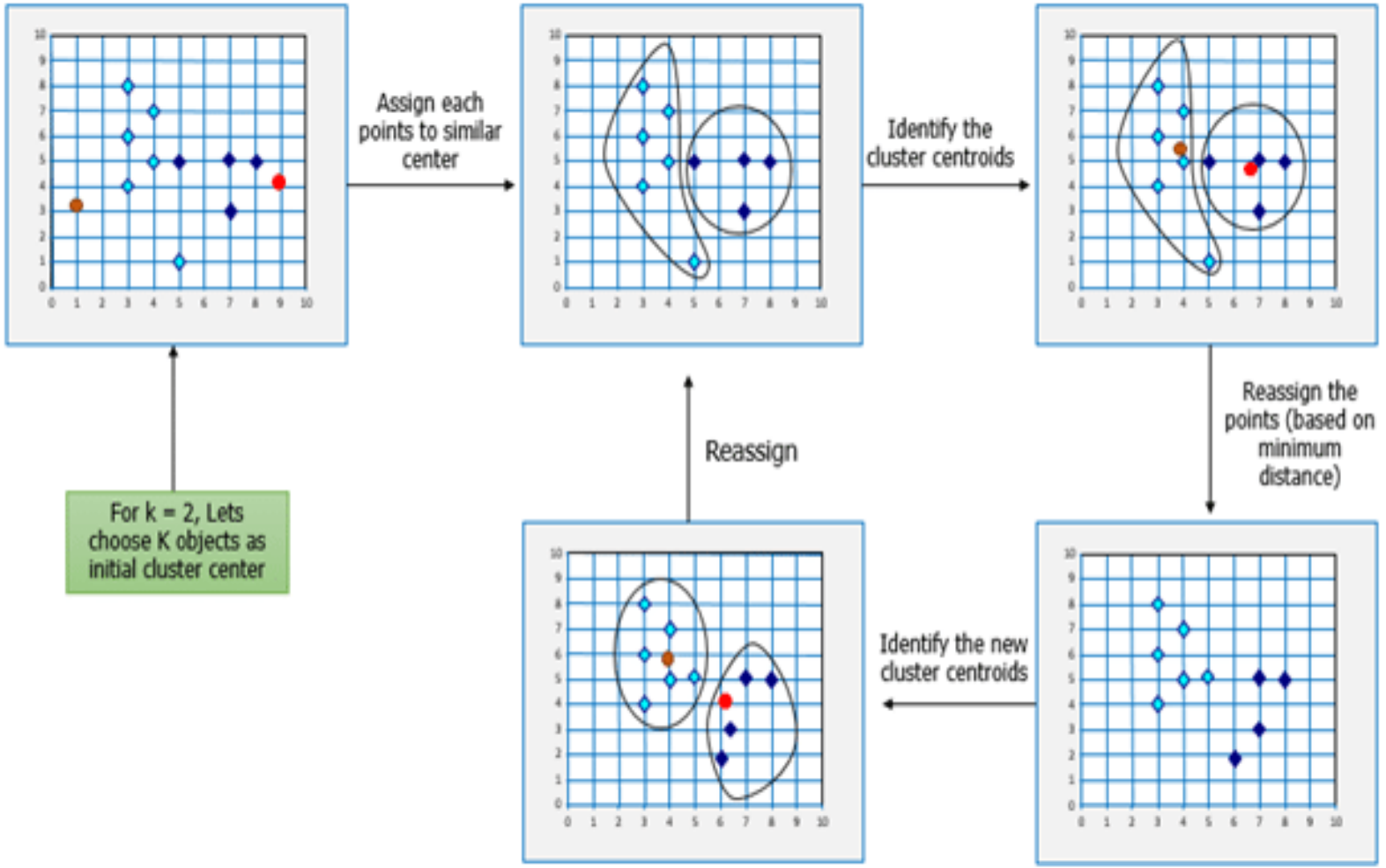
- K-means Algorithm
- The simplest among unsupervised learning algorithms. This works on the principle of k-means clustering. This actually means that the clustered groups (clusters) for a given set of data are represented by a variable ‘k’. For each cluster, a centroid (arithmetic mean of all the data points that belong to that cluster) is defined.
- The centroid is a data point present at the centre of each cluster (considering Euclidean distance). The trick is to define the centroids far away from each other so that the variation is less. After this, each data point in the cluster is assigned to the nearest centroid such that the sum of the squared distance between the data points and the cluster’s centroid is at the minimum.

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

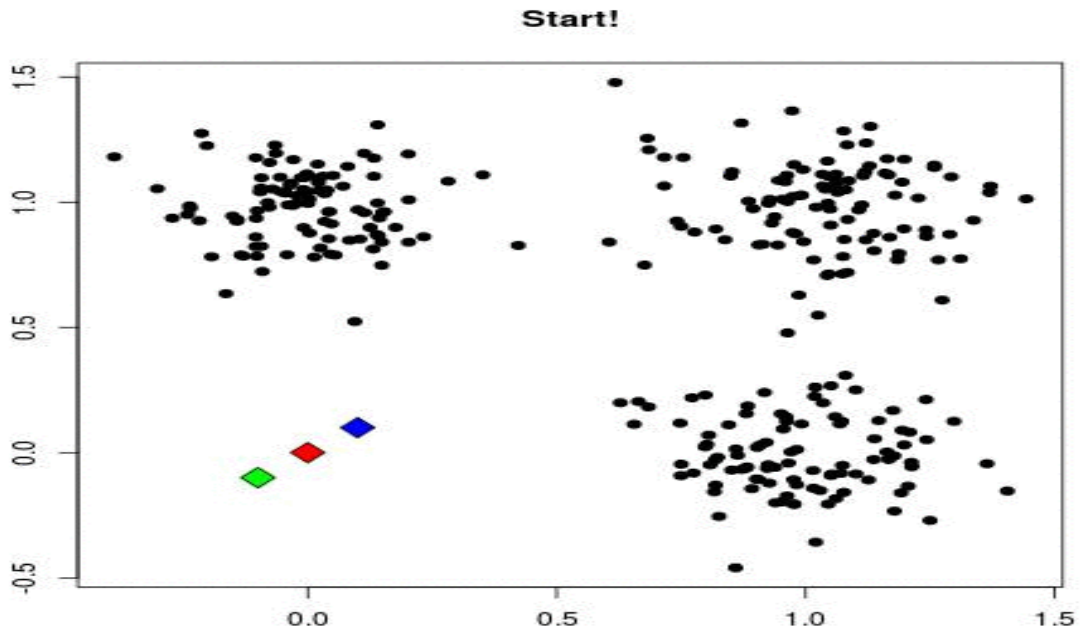
$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

- **Algorithm**
- 1.Clusters the data into k groups where k is predefined.
- 2. k points at random as cluster centers.
- 3.Assign objects to their closest cluster center according to the Euclidean distance function.
- 4.Calculate the centroid or mean of all objects in each cluster.
- 5.Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds.
- The Euclidean distance between two points in either the plane or 3-dimensional space measures the length of a segment connecting the two points.

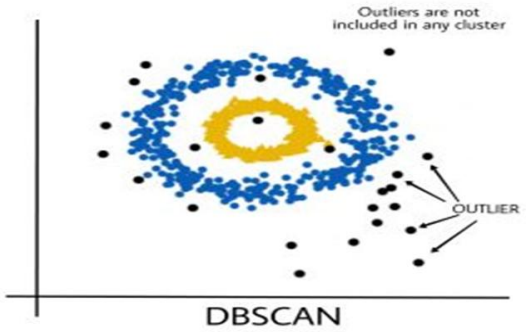
The step by step process:

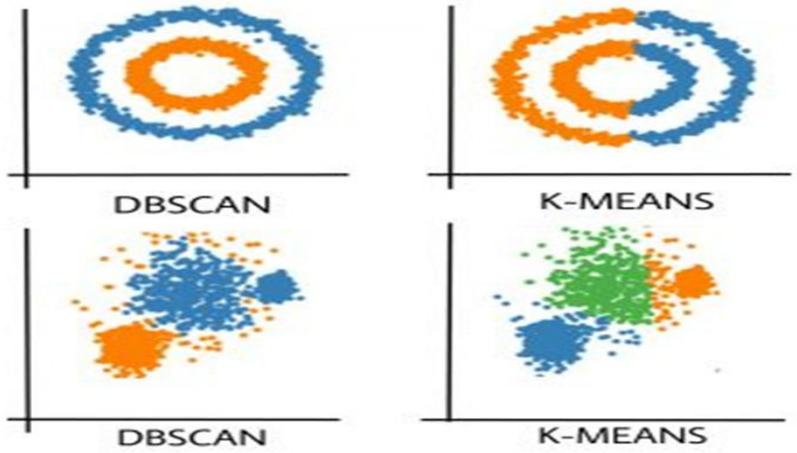


- K-means clustering algorithm has found to be very useful in grouping new data. Some practical applications which use k-means clustering are sensor measurements, activity monitoring in a manufacturing process, audio detection and image segmentation.



- **Disadvantage Of K-MEANS:**
- K-Means forms spherical clusters only. This algorithm fails when data is not spherical (i.e. same variance in all directions).
- K-Means algorithm is sensitive towards outlier. Outliers can skew the clusters in K-Means in very large extent.
- K-Means algorithm requires one to specify the number of clusters and for which there is no global method to choose best value.





- **Hierarchical Clustering Algorithms**

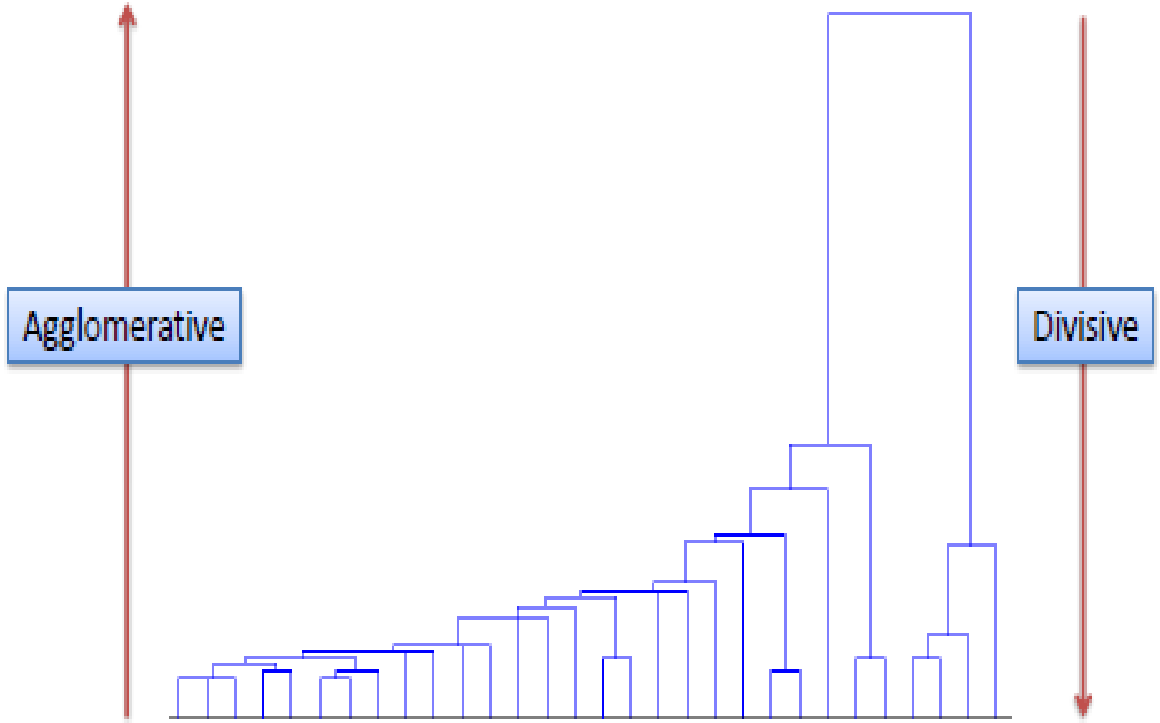
- Last but not the least are the hierarchical clustering algorithms. These algorithms have clusters sorted in an order based on the hierarchy in data similarity observations. Hierarchical clustering is categorised into two types, divisive(top-down) clustering and agglomerative (bottom-up) clustering.

- **Agglomerative Hierarchical clustering Technique:** In this technique, initially each data point is considered as an individual cluster. At each iteration, the similar clusters merge with other clusters until one cluster or K clusters are formed.

- **Divisive Hierarchical clustering Technique:**Divisive Hierarchical clustering is exactly the opposite of the **Agglomerative Hierarchical clustering.** In Divisive Hierarchical clustering, we consider all the data points as a single cluster and in each iteration, we separate the data points from the cluster which are not similar. Each data point which is separated is considered as an individual cluster.

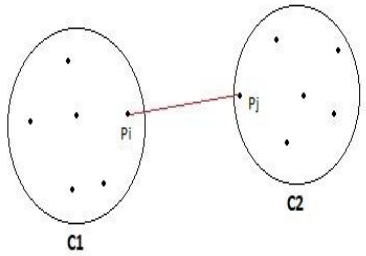
- Most of the hierarchical algorithms such as single linkage, complete linkage, median linkage, Ward's method, among others, follow the agglomerative approach.

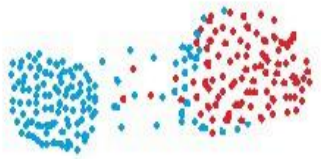
Hierarchical Clustering



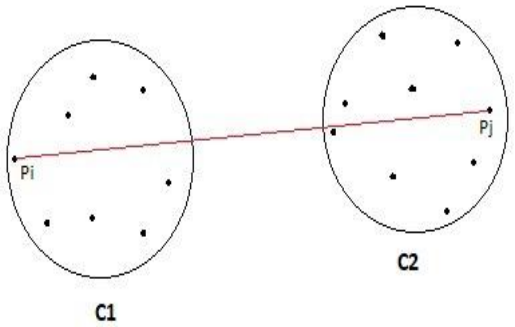
Calculating the similarity between two clusters is important to merge or divide the clusters. There are certain approaches which are used to calculate the similarity between two clusters:

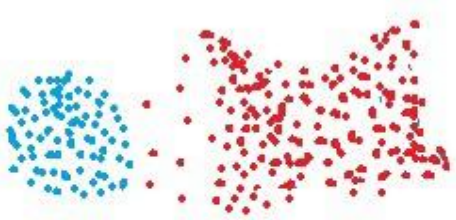
- **MIN:** Also known as single linkage algorithm can be defined as the similarity of two clusters C1 and C2 is equal to the **minimum** of the similarity between points Pi and Pj such that Pi belongs to C1 and Pj belongs to C2.
- This approach can separate non-elliptical shapes as long as the gap between two clusters is not small.
- MIN approach cannot separate clusters properly if there is noise between clusters.

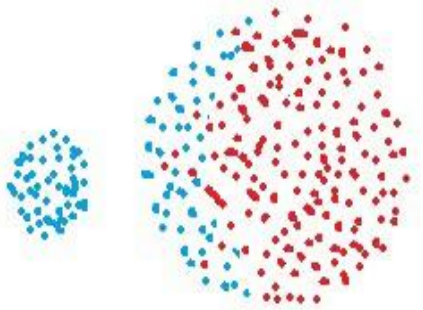




- **MAX:** Also known as the complete linkage algorithm, this is exactly opposite to the **MIN** approach. The similarity of two clusters C1 and C2 is equal to the **maximum** of the similarity between points Pi and Pj such that Pi belongs to C1 and Pj belongs to C2.
- MAX approach does well in separating clusters if there is noise between clusters but Max approach tends to break large clusters.





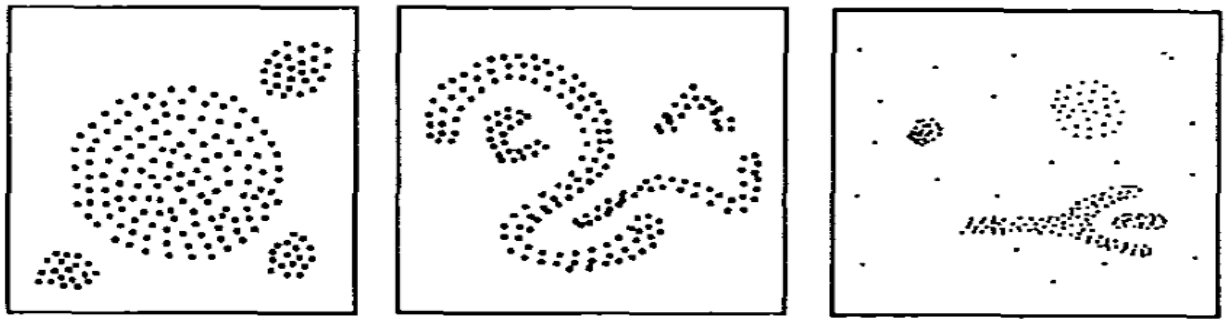


- **Group Average:** Take all the pairs of points and compute their similarities and calculate the average of the similarities.
- The group Average approach does well in separating clusters if there is noise between clusters but it is less popular technique in the real world.
- **Limitations of Hierarchical clustering Technique:**
- There is no mathematical objective for Hierarchical clustering.
- All the approaches to calculate the similarity between clusters has its own disadvantages.
- High space and time complexity for Hierarchical clustering. Hence this clustering algorithm cannot be used when we have huge data.

Density-based spatial clustering of applications with noise (DBSCAN) is a well-known data clustering algorithm that is commonly used in data mining and machine learning.

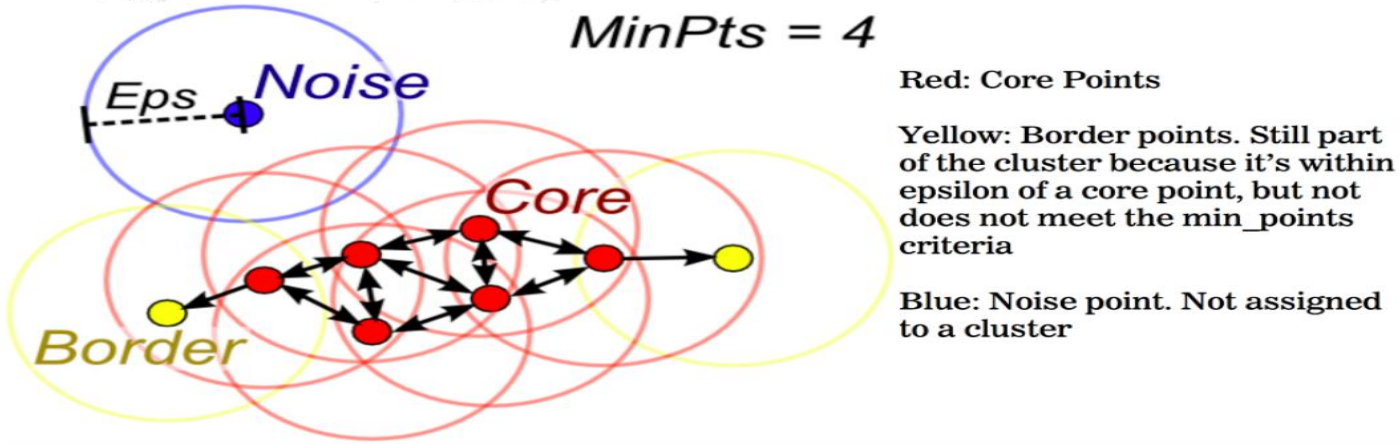
- Unlike to **K-means**, DBSCAN does not require the user to specify the number of clusters to be generated
- DBSCAN can find any shape of clusters. The cluster doesn't have to be circular.
- DBSCAN can identify outliers

The basic idea behind **density-based clustering** approach is derived from a human intuitive clustering method. by looking at the figure below, one can easily identify four clusters along with several points of noise, because of the differences in the density of points



DBSCAN algorithm has two parameters:

- ε: The radius of our neighborhoods around a data point *p*.
- *minPts*: The minimum number of data points we want in a neighborhood to define a cluster.
- Using these two parameters, DBSCAN categories the data points into three categories:
- *Core Points*: A data point *p* is a *core point* if **Nbhd**(*p*,ε) [ε-neighborhood of *p*] contains at least *minPts* ; |**Nbhd**(*p*,ε)| >= *minPts*.
- *Border Points: A data point *q* is a *border point* if **Nbhd**(*q*, ε) contains less than *minPts* data points, but *q* is *reachable* from some *core point p*.
- *Outlier*: A data point *o* is an *outlier* if it is neither a core point nor a border point. Essentially, this is the "other" class.



- The steps to the DBSCAN algorithm are:
- Pick a point at random that has not been assigned to a cluster or been designated as an *outlier*. Compute its neighborhood to determine if it's a *core point*. If yes, start a cluster around this point. If no, label the point as an *outlier*.
- Once we find a *core point* and thus a cluster, expand the cluster by adding all *directly-reachable* points to the cluster. Perform "neighborhood jumps" to find all *density-reachable* points and add them to the cluster. If an an *outlier* is added, change that point's status from *outlier* to *border point*.
- Repeat these two steps until all points are either assigned to a cluster or designated as an *outlier*.

- **Below is the DBSCAN clustering algorithm in pseudocode:**

```
DBSCAN(dataset, eps, MinPts){
# cluster index
C = 1
for each unvisited point p in dataset {
      mark p as visited
      # find neighbors
      Neighbors N = find the neighboring points of p

      if |N|>=MinPts:
        N = N U N'
        if p' is not a member of any cluster:
          add p' to cluster C
}
```

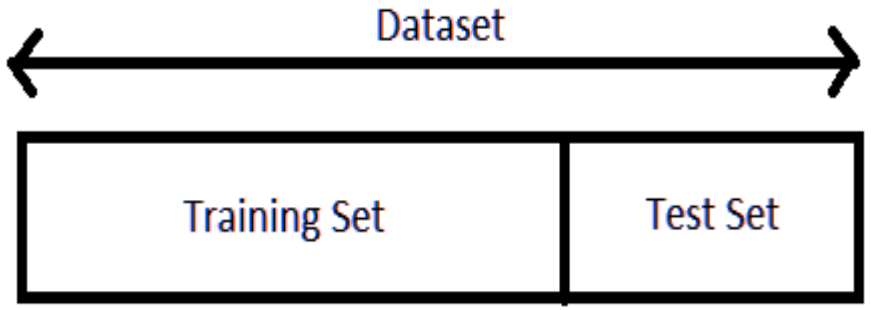
CROSS VALIDATION:

Cross-validation is a technique in which we train our model using the subset of the data-set and then evaluate using the complementary subset of the data-set.

The three steps involved in cross-validation are as follows :

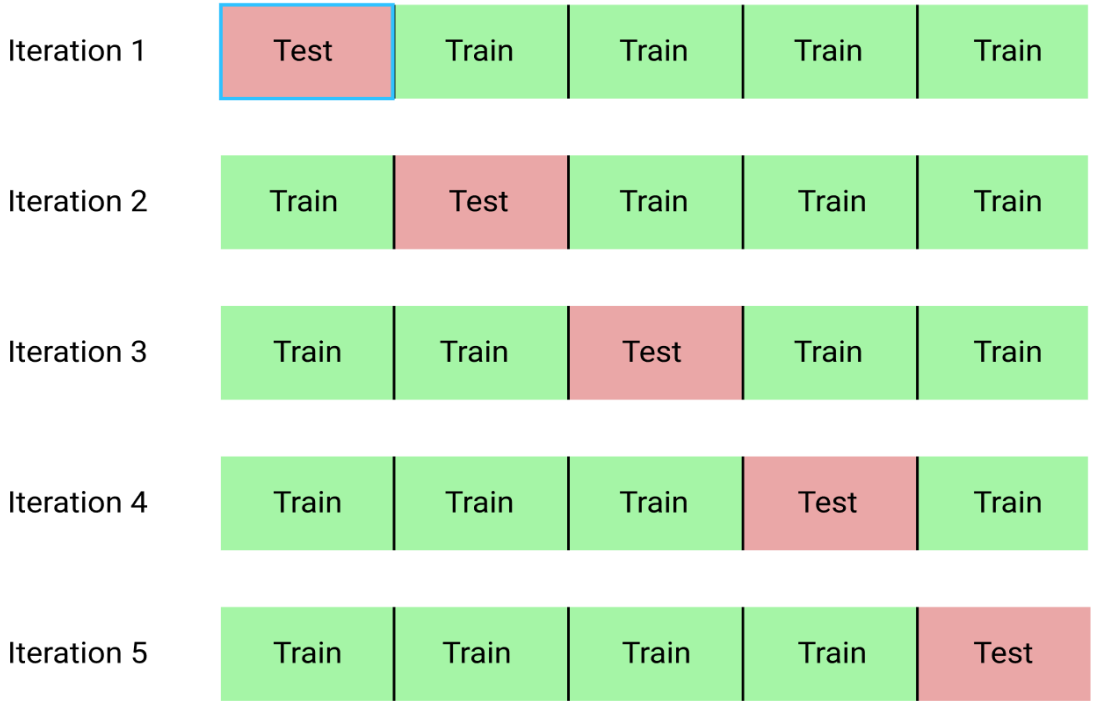
1. Split data set into training and test set
2. Using the training set train the model.
3. Test the model using the test set

USE: To get good out of sample accuracy



Even though we use cross validation technique we get variation in accuracy when we train our model for that we use K-fold cross validation technique

In **K-fold cross validation**, we split the data-set into k number of subsets(known as folds) then we perform training on the all the subsets but leave one(k-1) subset for the evaluation of the trained model. In this method, we iterate k times with a different subset reserved for testing purpose each time.



Code in python for k-cross validation:

from sklearn.model_selection import cross_val_score
List=cross_val_score(estimator**=#name of your model object** ,X**=#your trained input,**y**=#correct output**,cv=**#number of folds(k)**)

- First line is importing k fold cross validation function from model_selection sublibrary from sklearn library

- second line will give a list of accuracies based on k value. we need to average them to get the model accurate accuracy

How to choose the optimal values for the hyperparameters ?

Hyperparameters, are the parameters that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Examples:

1. The k in k-nearest neighbours.

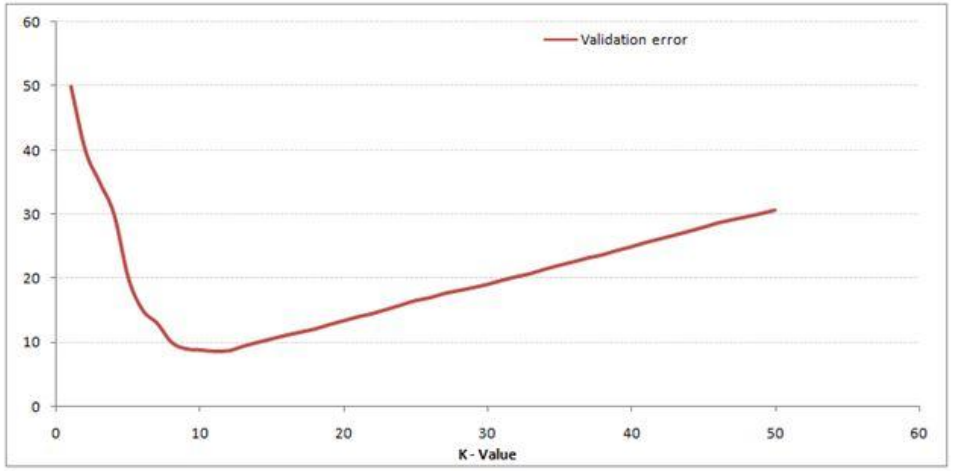
Models can have many hyperparameters and finding the best combination of parameters can be treated as a search problem. One of the best strategies for Hyperparameter tuning is grid_search.

GridSearchCV:

In GridSearchCV approach, machine learning model is evaluated for a range of hyperparameter values. This approach is called GridSearchCV, it searches for best set of hyperparameters from a grid of hyperparameters values.

For example, if we want to set hyperparameter K-nearest neighbours model, with different set of values. The gridsearch technique will check model with all possible combinations of hyperparameters, and will return the best one.

From graph we can say best value for k is 10 and grid search will search all the values of k that we given in range and return the best one



Code in python for getting optimal hyperparameter using gridsearch for support vector machine:

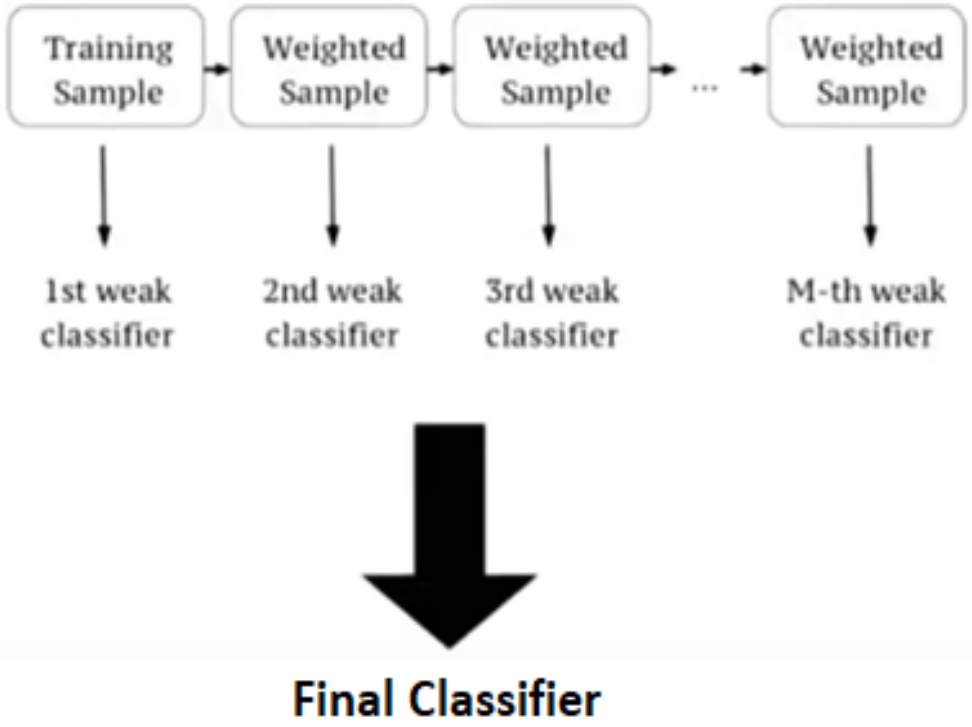
```
#importing svm from svc library
from sklearn.svm import SVC
Classifier=SVC()
#To import gridsearcv class from sklearn library
from sklearn.model_selection import GridSearchCV
#creating a list of dictonarties that need to be inputed for grid search
parameters=[{'C':[1,10,100,1000],'kernel':['linear']},{'C':[1,10,100,1000],'kernel':['rbf'],
'gamma':[0.5,0.1,0.01,0.001]}]
#creating grid search object
gridsearch=GridSearchCV(estimator=classifier,param_grid=parameters,scoring='accur
acy',cv=10,n_jobs=-1)
#fitting gridsearch with data set
gd=gridsearch.fit(X_train,y_train)
```

```
#fitting gridsearch with data set
gd=gridsearch.fit(X_train,y_train)
#best score among all models in grid search
bestaccuracy=gd.best_score_
#return the parameters of best model
best_param=gd.best_params_
```

XGBOOST

- XGBoost is an implementation of gradient boosted decision trees designed for speed and performance

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results.



Weight of variables predicted wrong by the tree is increased and these the variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model. It can work on regression, classification, ranking, and user-defined prediction problems.

Final value of the classifier is the class which is repeated more number of times among all classifier for classification problem and for regression problem final value is the average of all the regressor values got in sequential trees

Code in python for XGBOOST

```
#Fitting XGBoost to the training data for classifier
Import xgboost as xgb
my_model=xgb.XGBClassifier()
my_model.fit(X_train,y_train)

#predicting the test set results
Y_pred=my_model.predict(X_test)
```

THANKYOU