Machine Learning

By Mehul

What is Machine Learning?

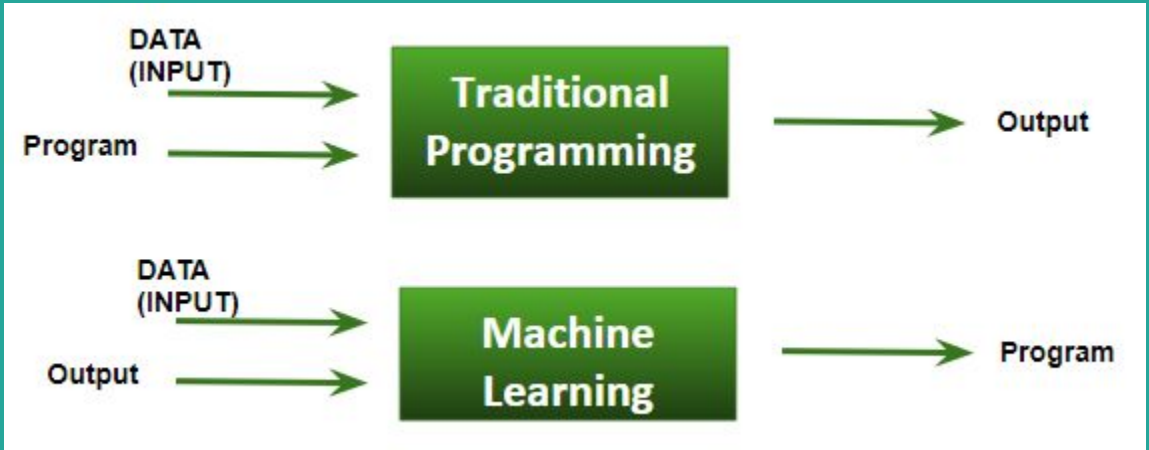
Field of study that gives computers the capability to learn without being explicitly programmed.

As it is evident from the name, it gives the computer that makes it more similar to humans: The ability to learn.

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

How is it different from traditional programming?

- **Traditional Programming :** We feed in DATA (Input) + PROGRAM (logic), run it on machine and get output.
- **Machine Learning :** We feed in DATA(Input) + Output, run it on machine during training and the machine creates its own program(logic), which can be evaluated while testing.



Terminologies of 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.**

  Note: Choosing informative, discriminating and independent features is a crucial step for effective algorithms. We generally employ a **feature extractor** to extract the relevant features from the raw data.

- **Target (Label)**

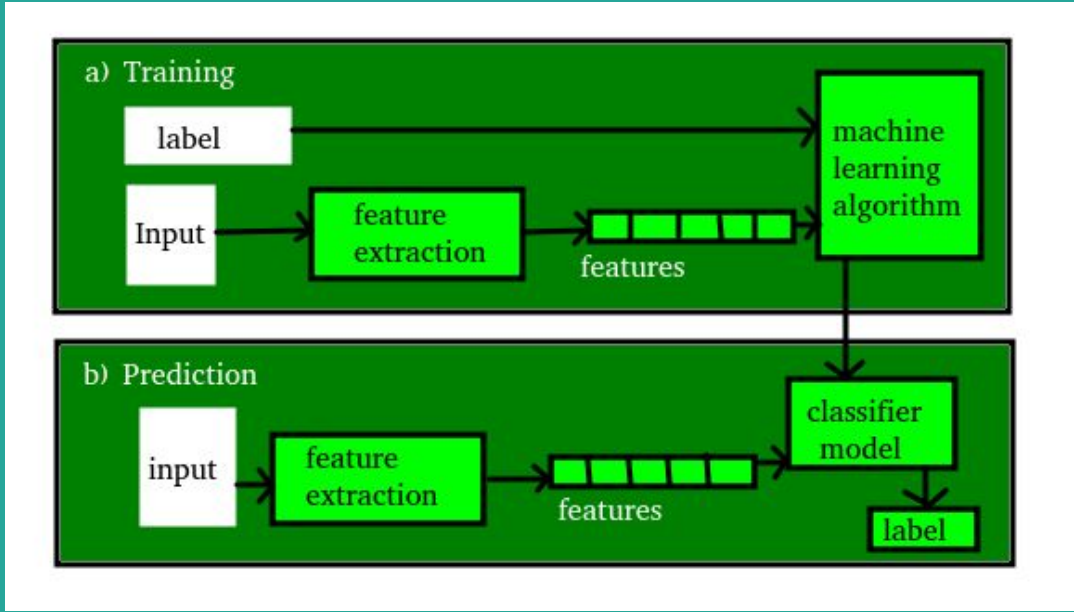
  A target variable or label is the value to be predicted by our model. For the fruit example discussed in the features section, the label with each set of input would be the name of the fruit like apple, orange, banana, etc.

- **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).



How ML works?
-The Process

- Gathering past data in any form suitable for processing.The better the quality of data, the more suitable it will be for modeling
- Data Processing – Sometimes, the data collected is in the raw form and it needs to be pre-processed.

  Example: Some tuples may have missing values for certain attributes, an, in this case, it has to be filled with suitable values in order to perform machine learning or any form of data mining. Missing values for numerical attributes such as the price of the house may be replaced with the mean value of the attribute whereas missing values for categorical attributes may be replaced with the attribute with the highest mode. This invariably depends on the types of filters we use. If data is in the form of text or images then converting it to numerical form will be required, be it a list or array or matrix. Simply, Data is to be made relevant and consistent. It is to be converted into a format understandable by the machine

- Divide the input data into training,cross-validation and test sets. The ratio between the respective sets must be 6:2:2
- Building models with suitable algorithms and techniques on the training set.
- Testing our conceptualized model with data which was not fed to the model at the time of training and evaluating its performance using metrics such as F1 score, precision and recall.

ML Applications -Examples

1. **Speech Recognition (Natural Language Processing in more technical terms) :** You talk to Cortana on Windows Devices. But how does it understand what you say? Along comes the field of Natural Language Processing, or N.L.P. It deals with the study of interactions between Machines and Humans, via Linguistics. Guess what is at the heart of NLP: Machine Learning Algorithms and Systems (Hidden Markov Models being one).
2. **Computer Vision :** Computer Vision is a subfield of AI which deals with a Machine's (probable) interpretation of the Real World. In other words, all Facial Recognition, Pattern Recognition, Character Recognition Techniques belong to Computer Vision. And Machine Learning once again, with it wide range of Algorithms, is at the heart of Computer Vision.
3. **Google's Self Driving Car** : Well. You can imagine what drives it actually. More Machine Learning goodness.

4. **Amazon's Product Recommendations:** Ever wondered how Amazon always has a recommendation that just tempts you to lighten your wallet. Well, that's a Machine Learning Algorithm(s) called "Recommender Systems" working in the backdrop. It learns every user's personal preferences and makes recommendations according to that.

5. **Youtube/Netflix :** They work just as above!

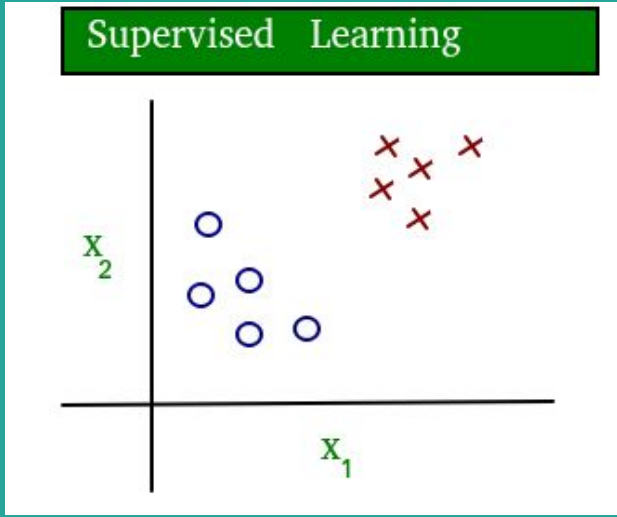
6. **Data Mining / Big Data :** This might not be so much of a shock to many. But Data Mining and Big Data are just manifestations of studying and learning from data at a larger scale. And wherever there's the objective of extracting information from data, you'll find Machine Learning lurking nearby.

7. **Stock Market/Housing Finance/Real Estate :** All of these fields, incorporate a lot of Machine Learning systems in order to better assess the market, namely "Regression Techniques", for things as mediocre as predicting the price of a House, to predicting and analyzing stock market trends.

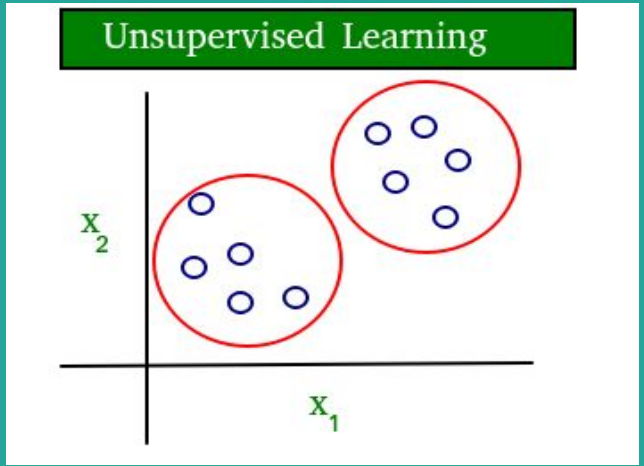
ypes of Machine Learning Algorithms

- Supervised Learning
- Unsupervised Learning
- Semi-supervised Learning
- Reinforcement Learning

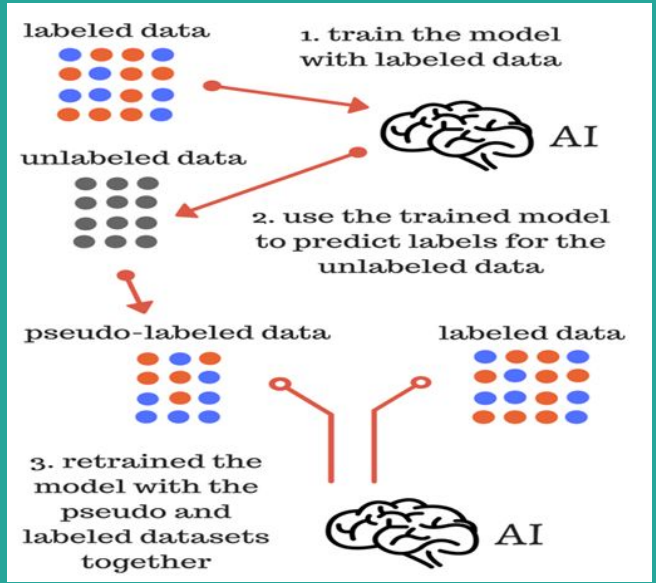
- **Supervised learning**: The computer is presented with example inputs and their desired outputs, given by a "teacher", and the goal is to learn a general rule that maps inputs to outputs. The training process continues until the model achieves the desired level of accuracy on the training data. Some real-life examples are:
    - **Image Classification:** You train with images/labels. Then in the future you give a new image expecting that the computer will recognize the new object.
    - **Market Prediction/Regression:** You train the computer with historical market data and ask the computer to predict the new price in the future.



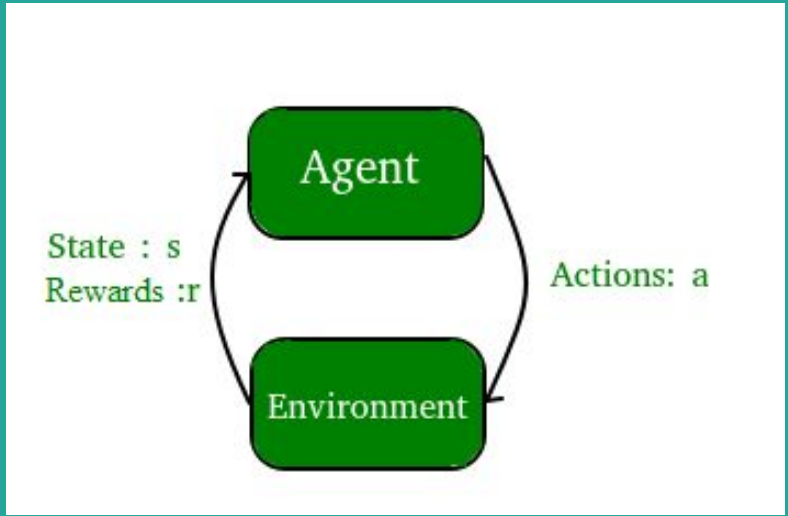
- **Unsupervised learning**: No labels are given to the learning algorithm, leaving it on its own to find structure in its input. It is used for clustering population in different groups. Unsupervised learning can be a goal in itself (discovering hidden patterns in data).
    - **Clustering:** You ask the computer to separate similar data into clusters, this is essential in research and science.
    - **High Dimension Visualization:** Use the computer to help us visualize high dimension data.
    - **Generative Models:** After a model captures the probability distribution of your input data, it will be able to generate more data. This can be very useful to make your classifier more robust.



- **Semi-supervised learning**: Problems where you have a large amount of input data and only some of the data is labeled, are called semi-supervised learning problems. These problems sit in between both supervised and unsupervised learning. For example, a photo archive where only some of the images are labeled, (e.g. dog, cat, person) and the majority are unlabeled.



- **Reinforcement learning**: A computer program interacts with a dynamic environment in which it must perform a certain goal (such as driving a vehicle or playing a game against an opponent). The program is provided feedback in terms of rewards and punishments as it navigates its problem space.



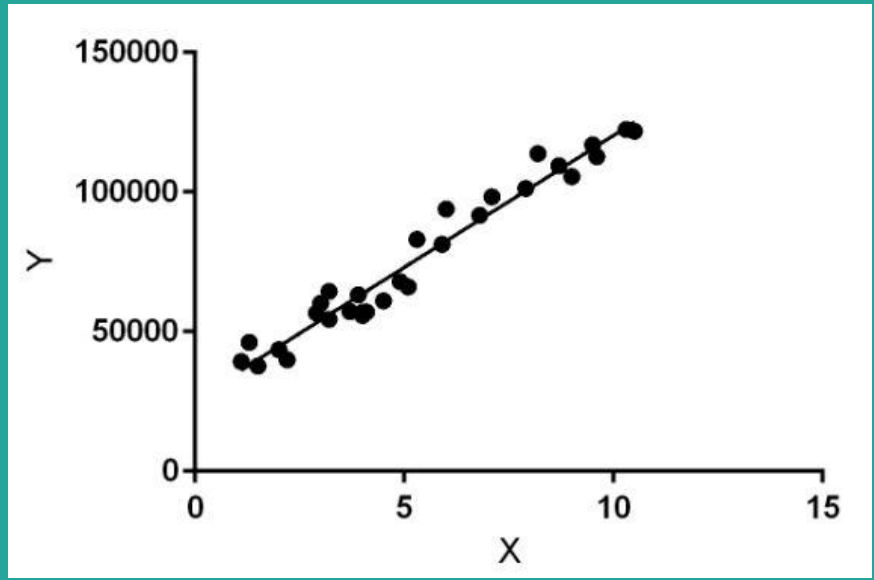
Supervised Learning

- Linear Regression
- Logistic Regression
- Support Vector Machines
- Naive Bayes

Linear Regression

- **Linear Regression** is a machine learning algorithm based on **supervised learning**.
- It performs a **regression task**. Regression models a target prediction value based on independent variables.
- It is mostly used for finding out the relationship between variables and forecasting.
- Different regression models differ based on – the kind of relationship between dependent and independent variables, they are considering and the number of independent variables being used

Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.
In the figure, X (input) is the work experience and Y (output) is the salary of a person. The regression line is the best fit line for our model.



Hypothesis function for Linear Regression

$$y = \theta_1 + \theta_2.x$$

While training the model we are given :

x: input training data (univariate – one input variable(parameter))

y: labels to data (supervised learning)

When training the model – it fits the best line to predict the value of y for a given value of x. The model gets the best regression fit line by finding the best θ_1 and θ_2 values.

θ_1: intercept

θ_2: coefficient of x

How to find the line with the best fit?

While training the model, the model calculates the cost function which measures the Root Mean Squared error between the predicted value (pred) and true value (y). The model targets to minimize the cost function.

Cost Function (J):

$$J = \frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

$$minimize\frac{1}{n}\sum_{i=1}^{n}(pred_i - y_i)^2$$

Gradient Descent

- To minimize the cost function, the model needs to have the best value of θ_1 and θ_2.
- Initially model selects θ_1 and θ_2 values randomly and then itertively update these value in order to minimize the cost function untill it reaches the minimum.
- By the time model achieves the minimum cost function, it will have the best θ_1 and θ_2 values.
- Using these finally updated values of θ_1 and θ_2 in the hypothesis equation of linear equation, model predicts the value of y in the best manner it can.

Therefore, the question arises – **How θ_1 and θ_2 values get updated ?**

Cost Function

$$J(\Theta_0, \Theta_1) = \frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y_i]^2$$

$h_\Theta(x_i)$: Predicted Value; y_i: True Value

Gradient Descent

$$\Theta_j = \Theta_j - \alpha\frac{\partial}{\partial\Theta_j}J(\Theta_0, \Theta_1)$$

α: Learning Rate

Now,

$$\frac{\partial}{\partial\Theta}J_\Theta = \frac{\partial}{\partial\Theta}\frac{1}{2m}\sum_{i=1}^{m}[h_\Theta(x_i) - y]^2$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\Theta(x_i) - y)\frac{\partial}{\partial\Theta_j}(\Theta x_i - y)$$

$$= \frac{1}{m}(h_\Theta(x_i) - y)x_i$$

Therefore,

$$\Theta_j := \Theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}[(h_\Theta(x_i) - y)x_i]$$

-> θ_j : Weights of the hypothesis.

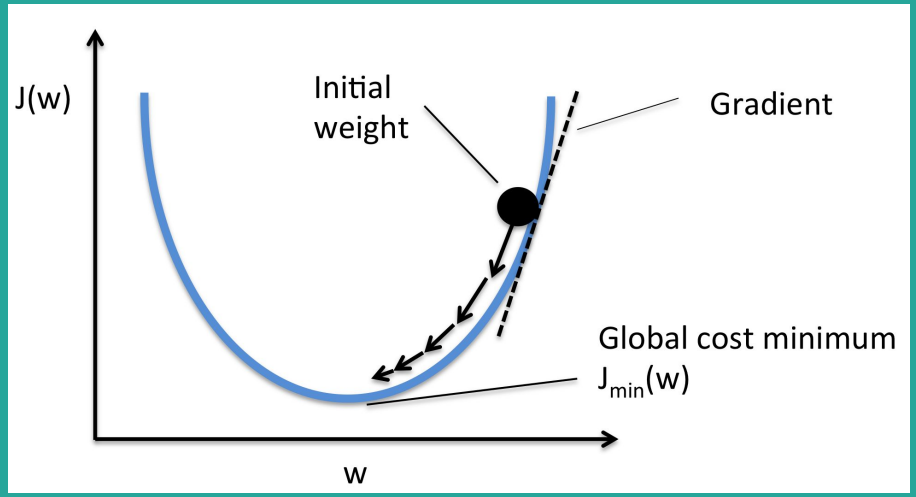
-> $h\theta(x_i)$: predicted y value for ith input.

-> **j** : Feature index number (can be 0, 1, 2, ..., n).

-> alpha : Learning Rate of Gradient Descent.

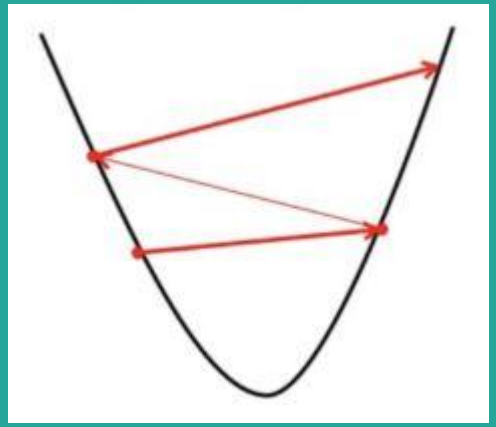
We graph cost function as a function of parameter estimates i.e. parameter range of our hypothesis function and the cost resulting from selecting a particular set of parameters. We move downward towards pits in the graph, to find the minimum value. Way to do this is taking derivative of cost function as explained in the previous slide.
In the Gradient Descent algorithm, one can infer two points :

1. **If slope is +ve** : $\theta j = \theta j$ – (+ve value). Hence value of θj decreases.
2. **If slope is -ve** : $\theta j = \theta j$ – (-ve value). Hence value of θj increases.

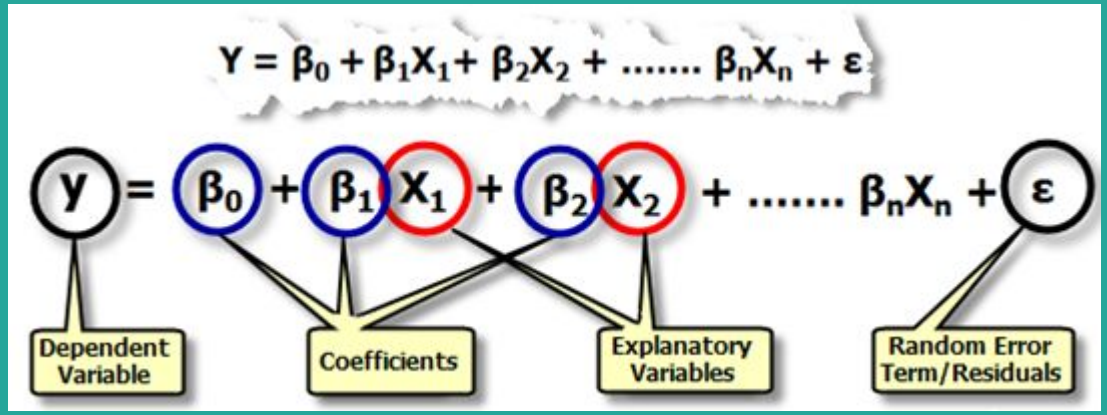


The choice of correct learning rate is very important as it ensures that Gradient Descent converges in a reasonable time. :

- If we choose **alpha to be very large**, Gradient Descent can overshoot the minimum. It may fail to converge or even diverge.
- If we choose alpha to be very small, Gradient Descent will take small steps to reach local minima and will take a longer time to reach minima.



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.

Polynomial Regression

- Quadratic – 2nd order

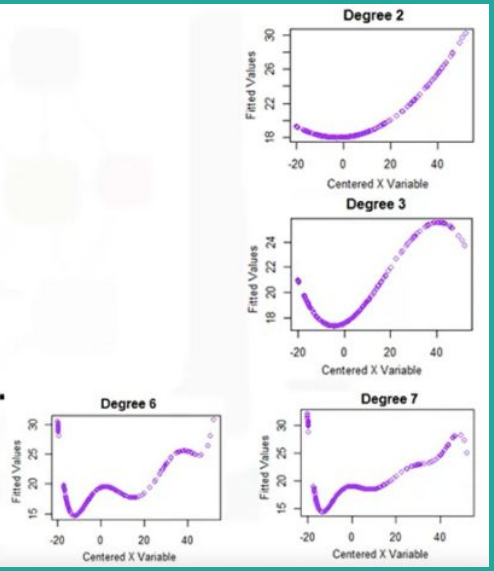
$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2$$

- Cubic – 3rd order

$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3$$

- Higher order

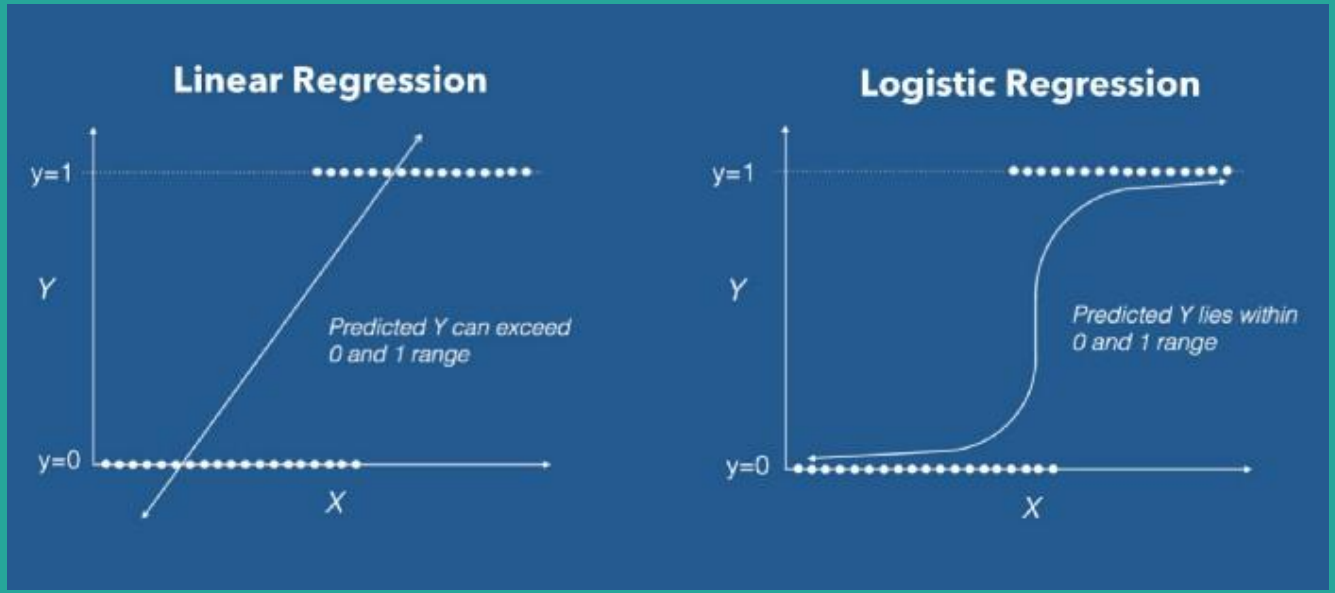
$$\hat{Y} = b_0 + b_1 x_1 + b_2 (x_1)^2 + b_3 (x_1)^3 + ..$$



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.

Logistic Regression

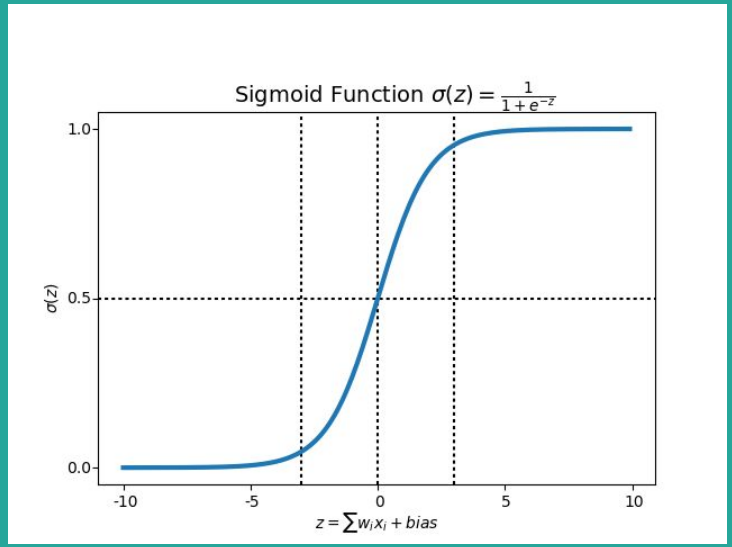
Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability.
What are the types of logistic regression?
Binary (eg. Tumor Malignant or Benign)
Multi-linear functions failsClass (eg. Cats, dogs or Sheep's)



Sigmoid Function

We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the '**Sigmoid function**'.

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



Hypothesis Function for Logistic Regression

$$Z = Wx + B$$

$$h_\theta(x) = sigmoid(Z)$$

$$also,\ sigmoid(x) = \frac{1}{\left(1 + e^{-x}\right)}$$

$$h_\theta(x) = \frac{1}{\left(1 + e^{-(Wx + B)}\right)}$$

(*hypothesis function for logistic regression*)

Decision Boundary

Threshold

If the value of hypothesis function is greater than or equal to threshold value, the output is 1 else the output is 0.

Understanding Decision Boundary with an example –

Let our hypothesis function be

$$h_\Theta(x) = g[\Theta_0 + \Theta_1 x_1 + \Theta_2 x_2 + \Theta_3 x_1^2 + \Theta_4 x_2^2]$$

Let the threshold be 0.5

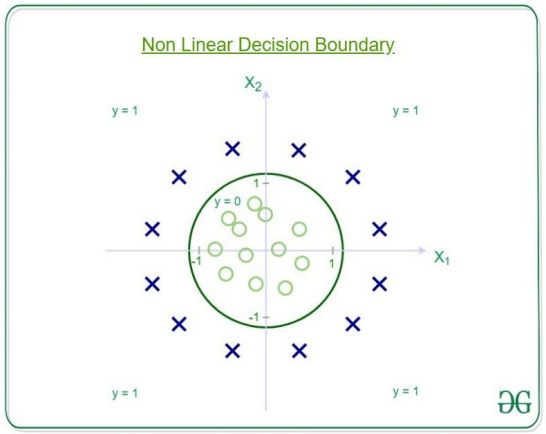
Let out weights or parameters be –

$$\Theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

So, it predicts y = 1 if

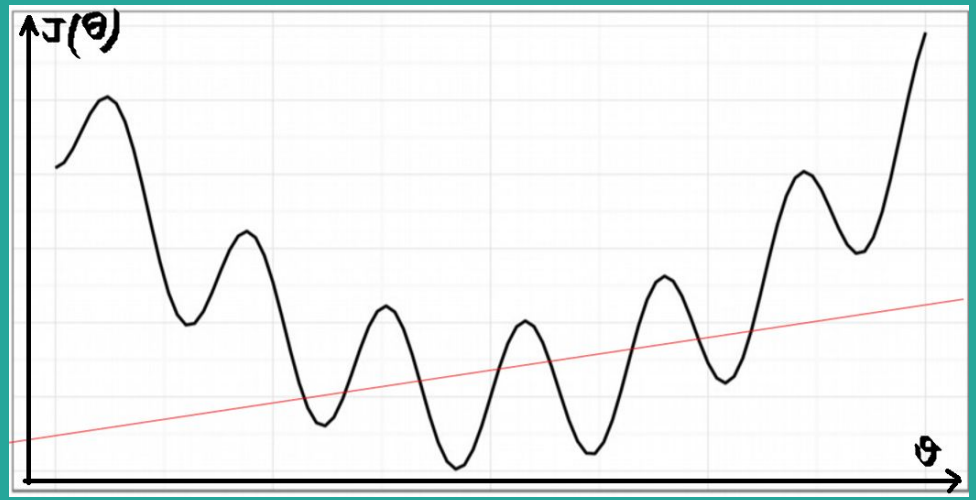
$$-1 + x_1^2 + x_2^2 \geqslant 0$$

$$\Rightarrow x_1^2 + x_2^2 \geqslant 1$$



Cost Function

If we try to use the cost function of the linear regression in ‘Logistic Regression’ then it would be of no use as it would end up being a **non-convex** function with many local minimums, in which it would be very **difficult** to **minimize the cost value** and find the global minimum.

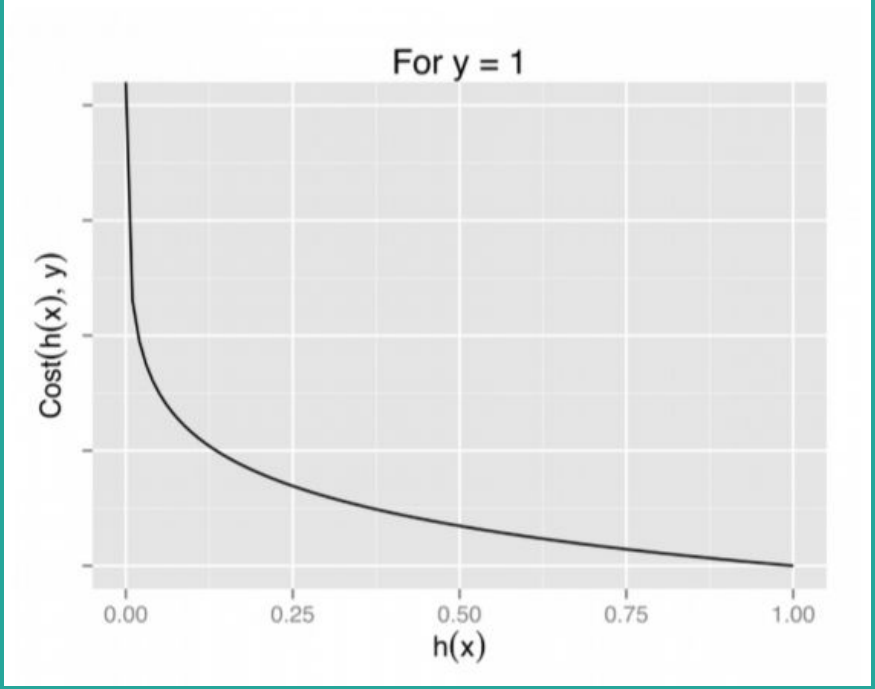


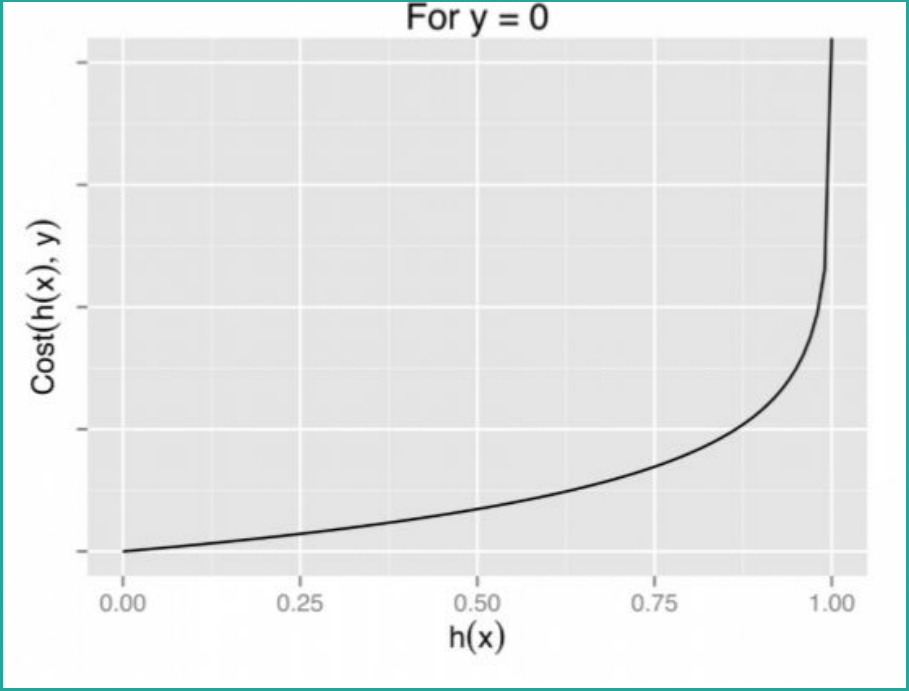
Therefore, the new cost function is:

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

$$J(\theta) = -\frac{1}{m}\sum \left[y^{(i)} \log(h\theta(x(i))) + \left(1 - y^{(i)}\right) \log(1 - h\theta(x(i))) \right]$$

Cost Function Graph





Gradient Descent

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

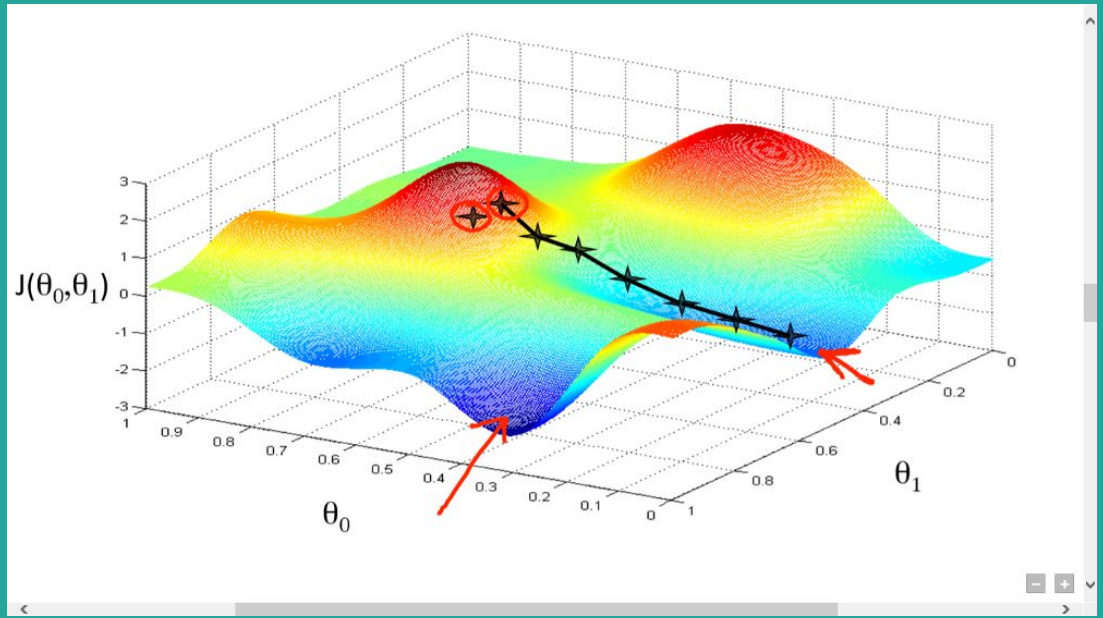
Repeat $\{$

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

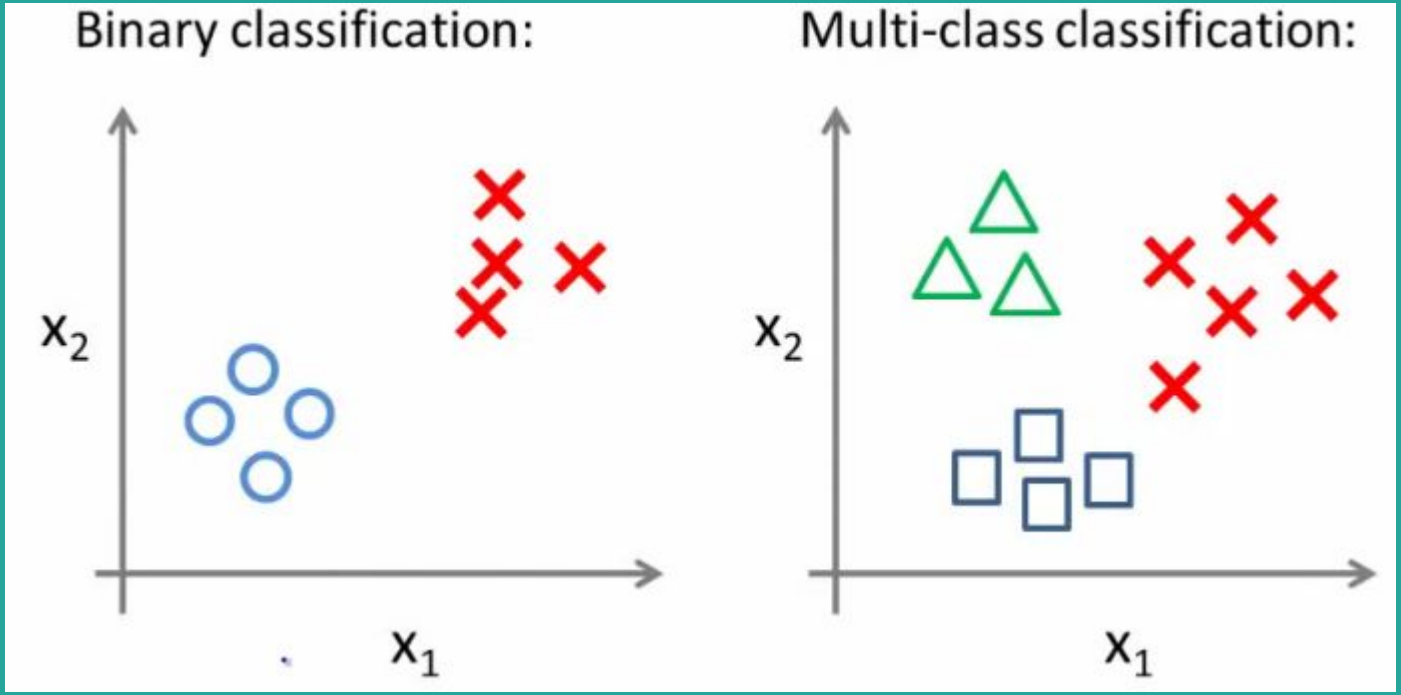
(simultaneously update all θ_j)

$\}$

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.



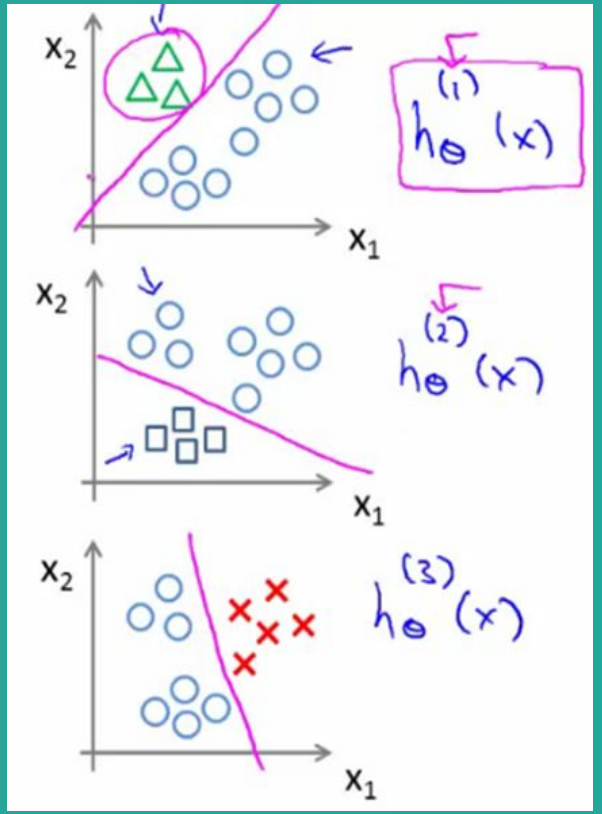
Multiclass Classification



One Vs All Method

Getting logistic regression for multiclass classification using **one vs. all**

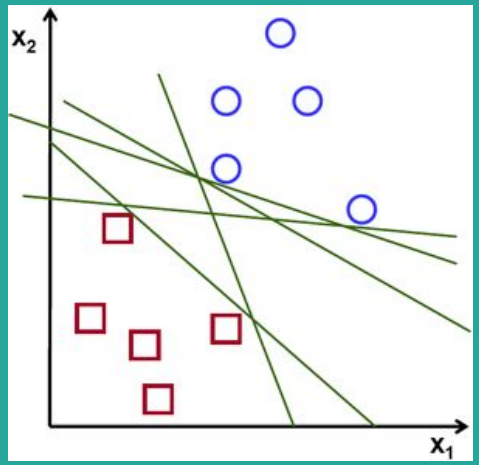
- Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that y = i
- On a new input, *x* to make a prediction, pick the class *i* that maximizes the probability that $h_\theta^{(i)}(x) = 1$

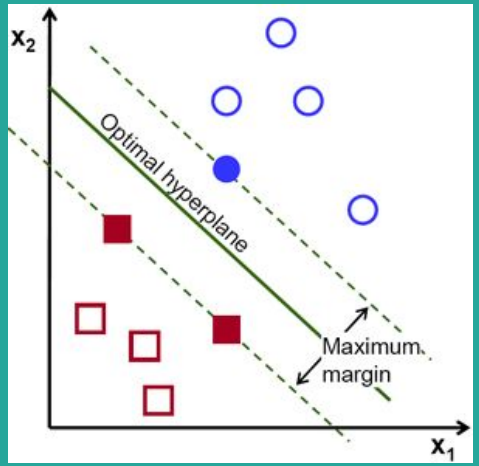


Support Vector Machines

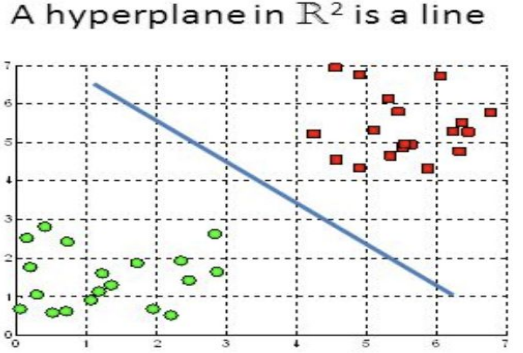
The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

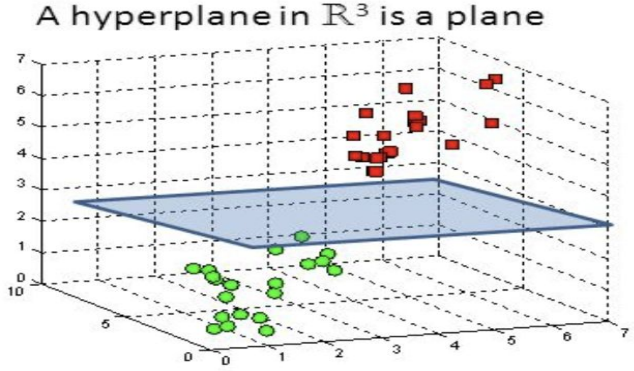
To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.





Hyperplanes





Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of input features is 3, then the hyperplane becomes a two-dimensional plane.

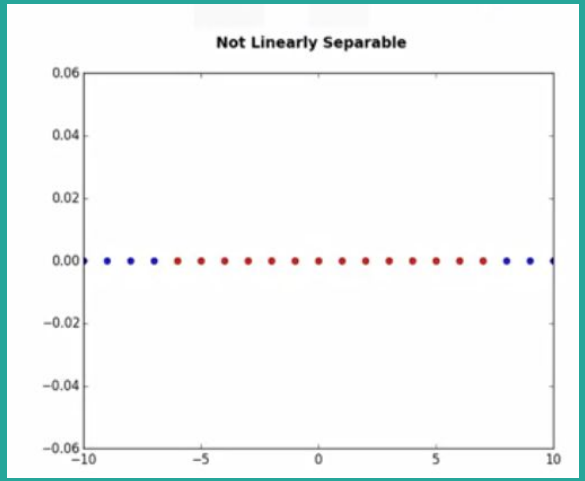
Steps of SVMs

- 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.

Example

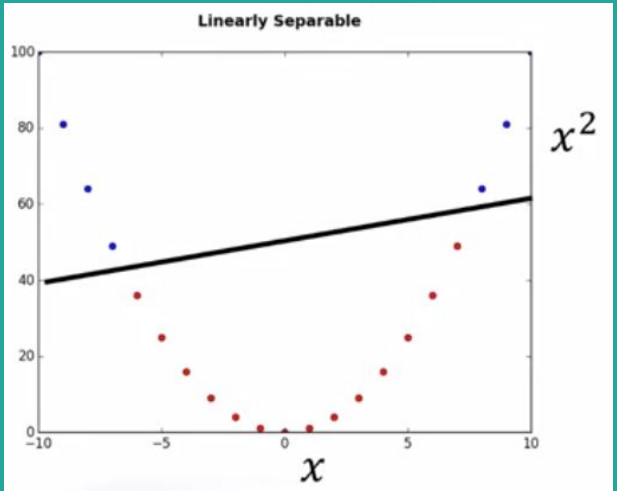
For the sake of simplicity, imagine that our dataset is one-dimensional data.
This means we have only one feature x.
As you can see, it is not linearly separable.



Data Transformation -Kernels

We can transfer the data 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.

Cost Function and Gradient Updates

Hinge Loss Function:

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases}$$

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions looks as below.

Cost Function:

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\delta}{\delta w_k} \lambda \| w \|^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} \left(1 - y_i \langle x_i, w \rangle\right)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

1. When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.
2. When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform

$$w = w - \alpha \cdot (2\lambda w)$$

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Naive Bayes Classifiers

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.

Bayes Theorem :

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred.

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

To start with, let us consider a dataset.
Given the weather conditions, each tuple classifies the conditions as fit(“Yes”) or unfit(“No”) for plaing golf.

	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

Assumption:

The fundamental Naive Bayes assumption is that each feature makes an independent and equal contribution to the outcome.

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 accuratey. None of the attributes is irrelevant and assumed to be contributing **equally** to the outcome.

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

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

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"

Now, as our features are independent (assumption), we can use the following theorem:
If any two events A and B are independent, then,

P(A,B) = P(A)P(B)

Hence, we reach to the result:

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

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

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

Now, we need to create a classifier model. For this, we find the probability of given set of inputs for all possible values of the class variable *y* and pick up the output with maximum probability. This can be expressed mathematically as:

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

So, finally, we are left with the task of calculating P(y) and P(xi | y).

We need to find P(xi | yj) for each xi in X and yj in y. All these calculations have been demonstrated in the tables below:

So, in the tables, we have calculated P(xi | yj) for each xi in X and yj 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.

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 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

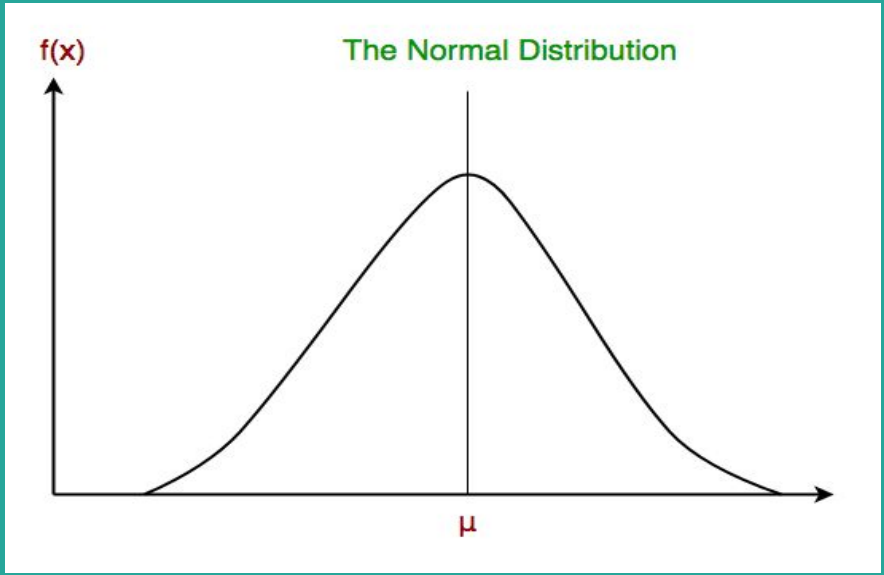
$$P(Yes|today) > P(No|today)$$

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

Types of Naive Bayes Classifiers

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

- **Multinomial Naive Bayes**: Feature vectors represent the frequencies with which certain events have been generated by a **multinomial distribution**. This is the event model typically used for document classification.
- **Bernoulli Naive Bayes**: In the multivariate Bernoulli event model, features are independent booleans (binary variables) describing inputs. Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence(i.e. a word occurs in a document or not) features are used rather than term frequencies(i.e. frequency of a word in the document).
- **Gaussian Naive Bayes**: 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:



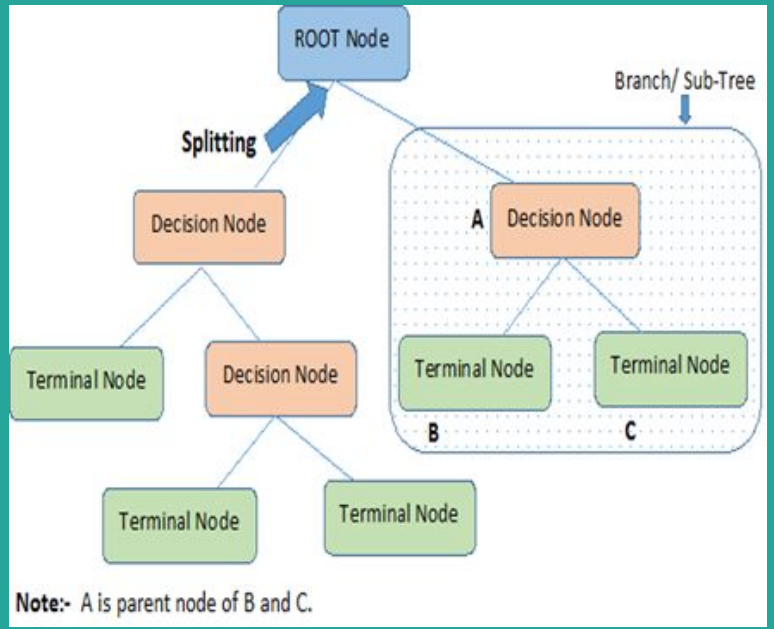
The likelihood of the features is assumed to be Gaussian, hence, conditional probability is given by:

$$p(x_i|y_j) = \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{(x_i-\mu_j)^2}{2\sigma_j^2}}$$

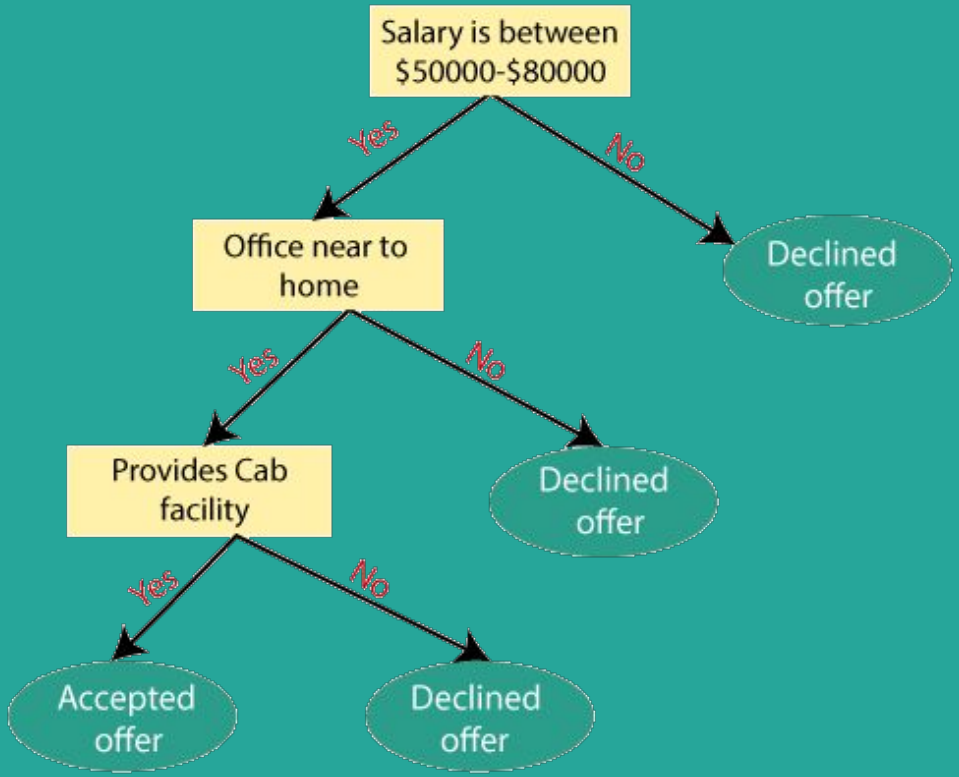
Decision Tree

- 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.**
- 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.

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.



Should you accept the job offer?



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

There are two popular techniques for ASM, which are:

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

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.

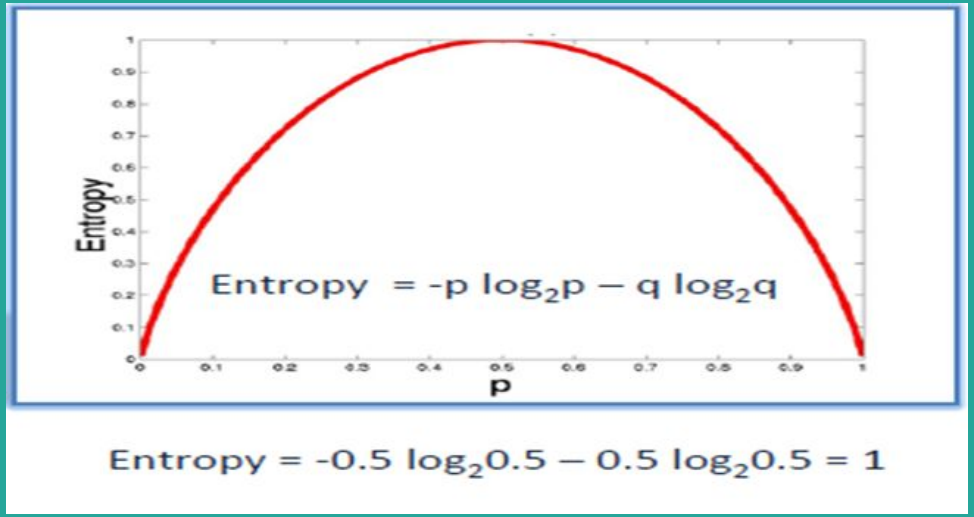
Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)]

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

Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)

Where,

- **S= Total number of samples**
- **P(yes)= probability of yes**
- **P(no)= probability of no**



Gini Index:

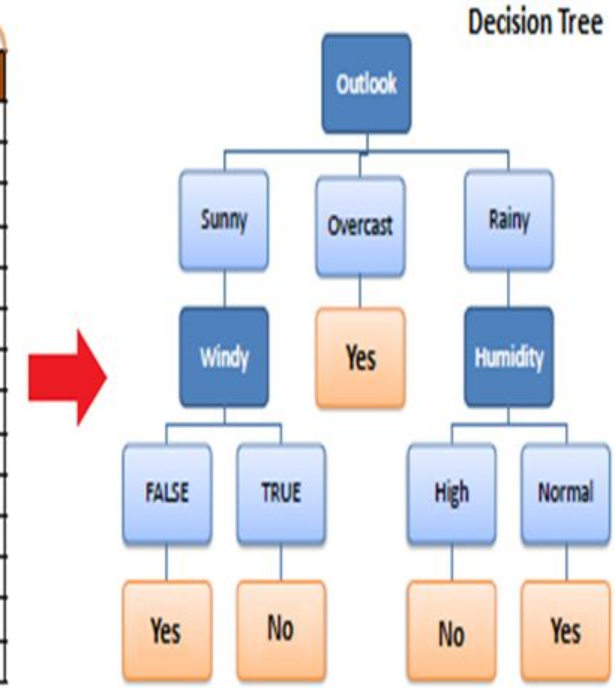
- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.

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

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

Step 1: Calculate Entropy of 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: Calculate Information Gain for each attribute

		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			

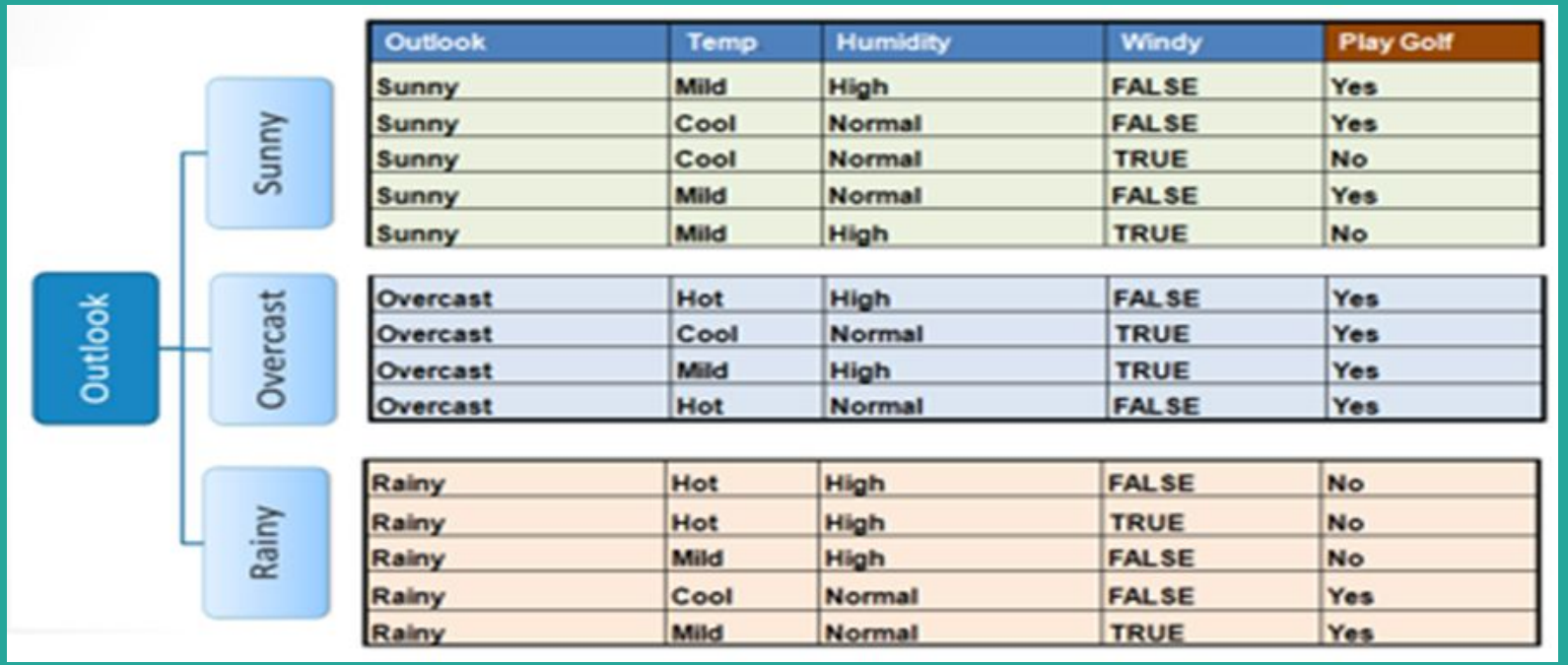
		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

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

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.



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

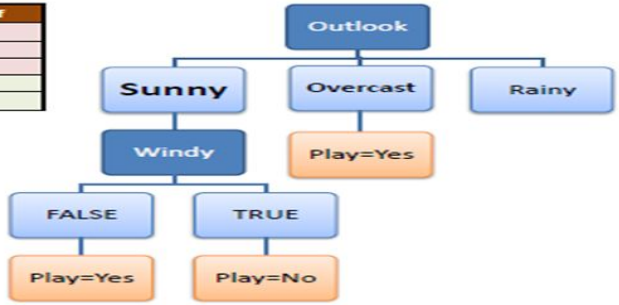
Rainy	Hot	High	FALSE	No
Rainy	Hot	High	TRUE	No
Rainy	Mild	High	FALSE	No
Rainy	Cool	Normal	FALSE	Yes
Rainy	Mild	Normal	TRUE	Yes

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 algorithm is run recursively on the non-leaf branches, until all data is classified.

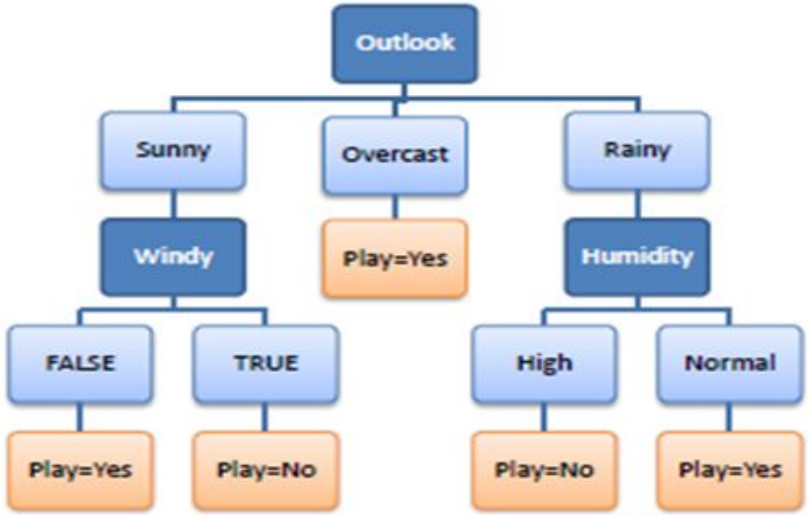
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

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

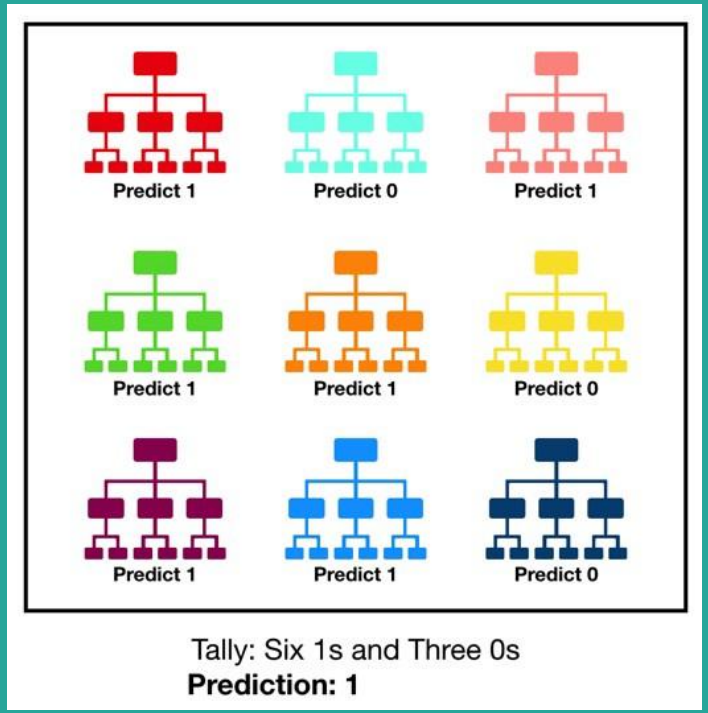
R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



Random Forest

The random forest is a classification algorithm consisting of many decisions trees that operate as an ensemble.

Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction



The reason that the random forest model works so well is:
A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key.
Uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions.

The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don’t constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction.

So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

Ensuring that the Models Diversify Each Other

So how does random forest ensure that the behavior of each individual tree is not too correlated with the behavior of any of the other trees in the model? It uses the following two methods:

1. Bagging (Bootstrap Aggregation)
2. Feature Randomness

Bagging (Bootstrap Aggregation)

Decisions trees are very sensitive to the data they are trained on — small changes to the training set can result in significantly different tree structures. Random forest takes advantage of this by allowing each individual tree to randomly sample from the dataset with replacement, resulting in different trees. This process is known as bagging.

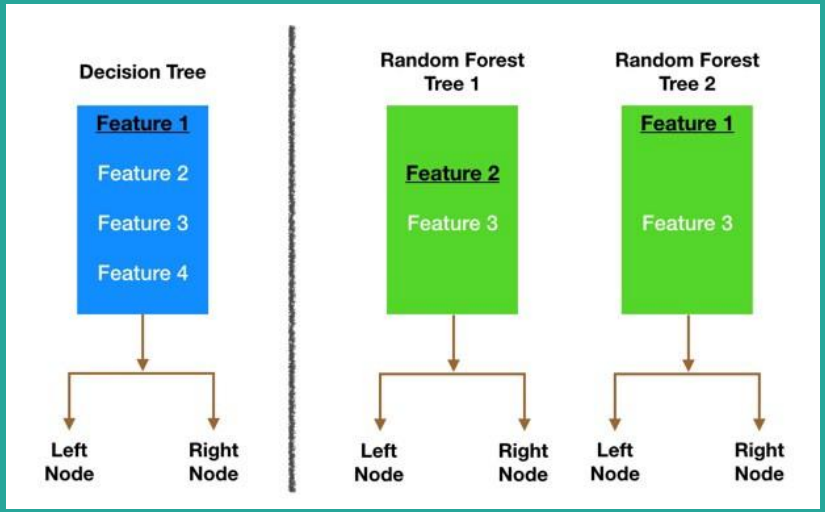
Notice that with bagging we are not subsetting the training data into smaller chunks and training each tree on a different chunk. Rather, if we have a sample of size N, we are still feeding each tree a training set of size N (unless specified otherwise). But instead of the original training data, we take a random sample of size N with replacement. For example, if our training data was [1, 2, 3, 4, 5, 6] then we might give one of our trees the following list [1, 2, 2, 3, 6, 6].

Feature Randomness

In a normal decision tree, when it is time to split a node, we consider every possible feature and pick the one that produces the most separation between the observations in the left node vs. those in the right node.
In contrast, each tree in a random forest can pick only from a random subset of features. This forces even more variation amongst the trees in the model and ultimately results in lower correlation across trees and more diversification.

For Example, when we check out random forest Tree 1, we find that it it can only consider Features 2 and 3 (selected randomly) for its node splitting decision. We know from our traditional decision tree (in blue) that Feature 1 is the best feature for splitting, but Tree 1 cannot see Feature 1 so it is forced to go with Feature 2 (black and underlined). Tree 2, on the other hand, can only see Features 1 and 3 so it is able to pick Feature 1.

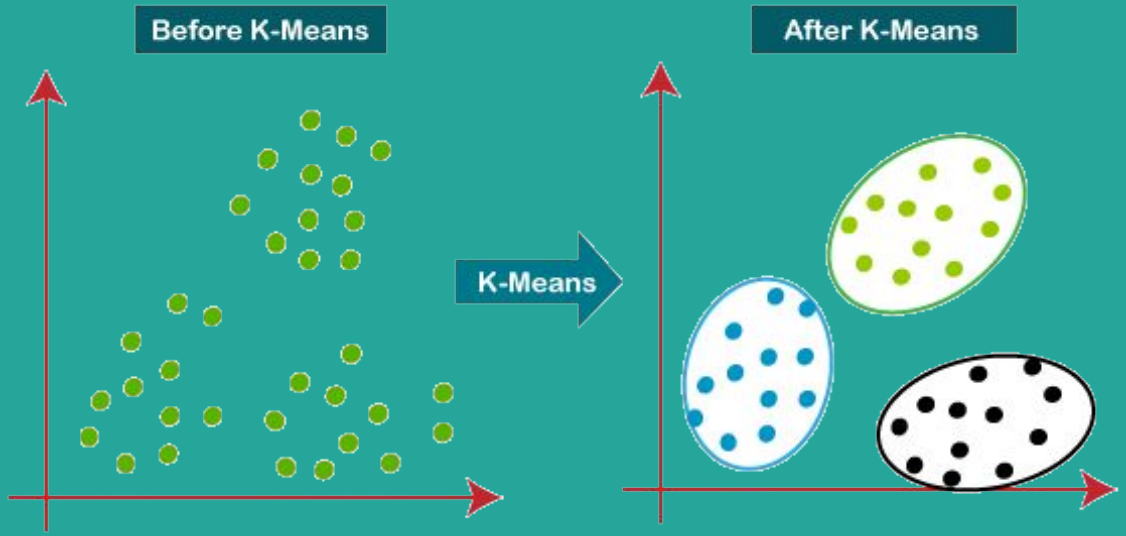
So in our random forest, we end up with trees that are not only trained on different sets of data (thanks to bagging) but also use different features to make decisions.



Unsupervised Learning

K Means Clustering

K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.
It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.
If K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.



Algorithm

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

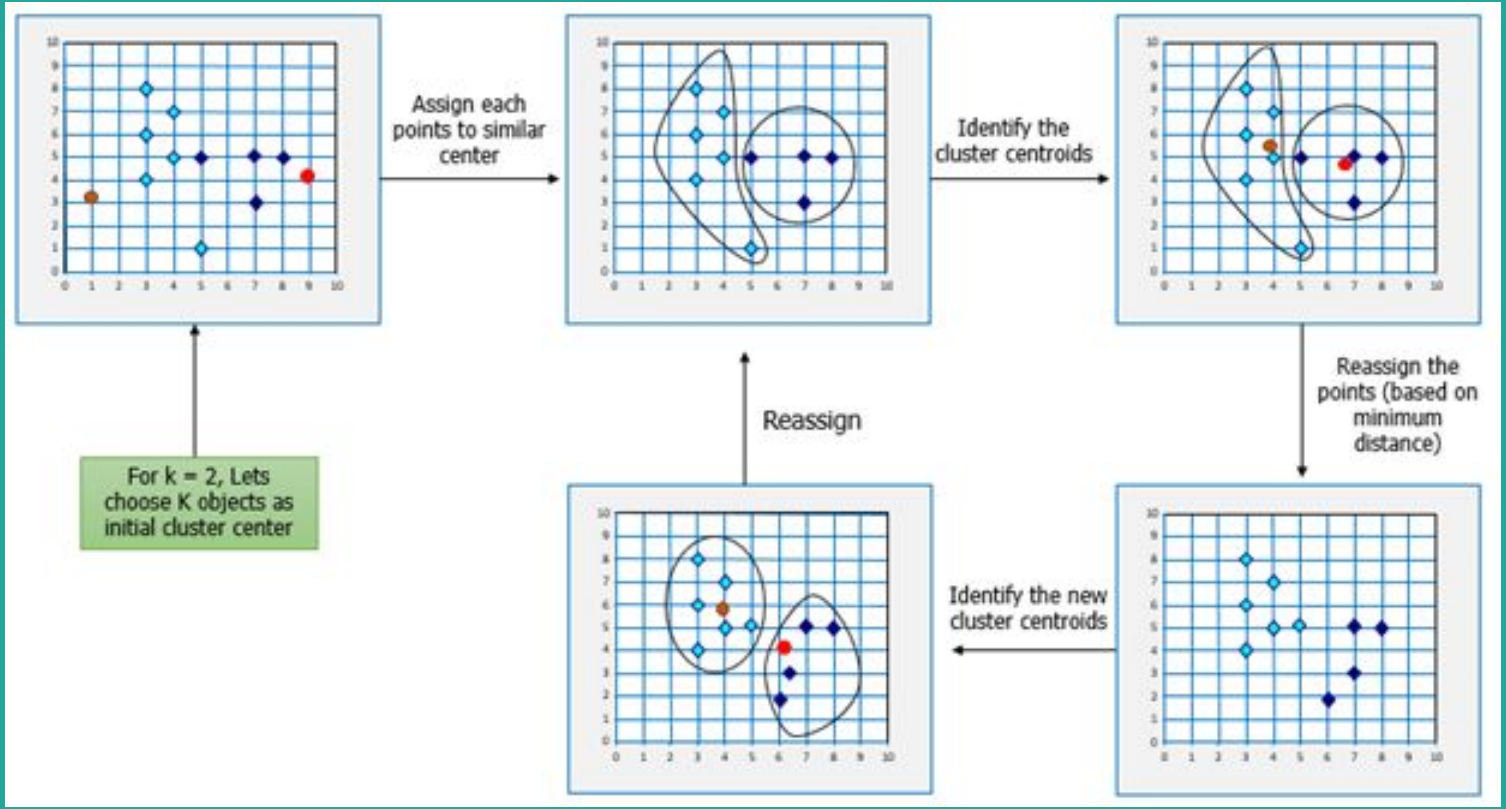
Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

Step-7: The model is ready.

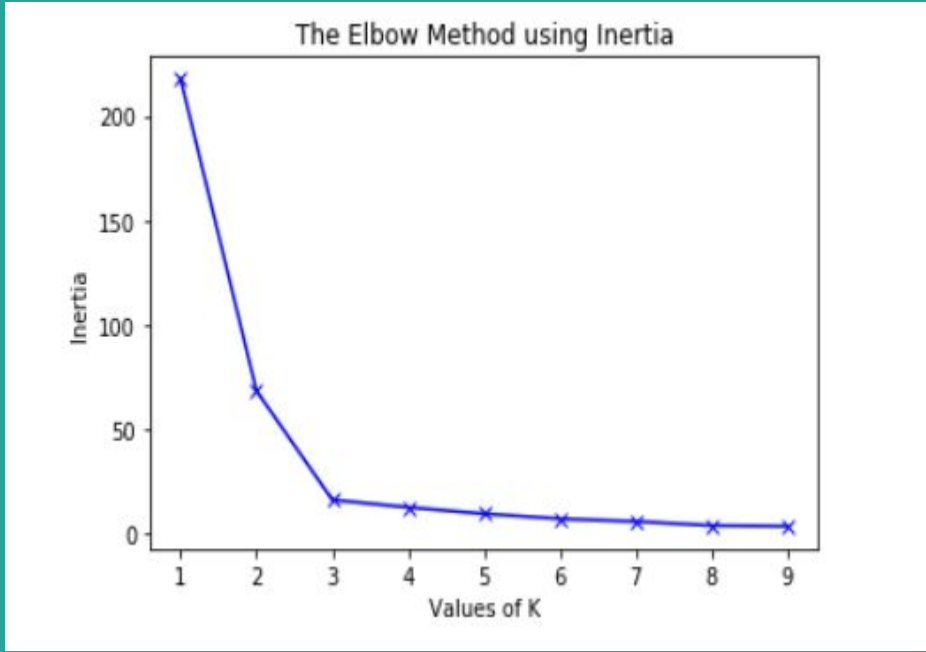

Assign each points to similar center
Identify the cluster centroids
Reassign the points (based on minimum distance)
Identify the new cluster centroids
Reassign
For k = 2, Lets choose K objects as initial cluster center

How to know the value of k?
-Elbow Method

Inertia : It is the sum of squared distances of samples to their closest cluster center.

Plot the graph between the Inertia and different k values

To determine the optimal number of clusters, we have to select the value of k at the “elbow” ie the point after which the distortion/inertia start decreasing in a linear fashion. Thus for the given data, we conclude that the optimal number of clusters for the data is **3**.

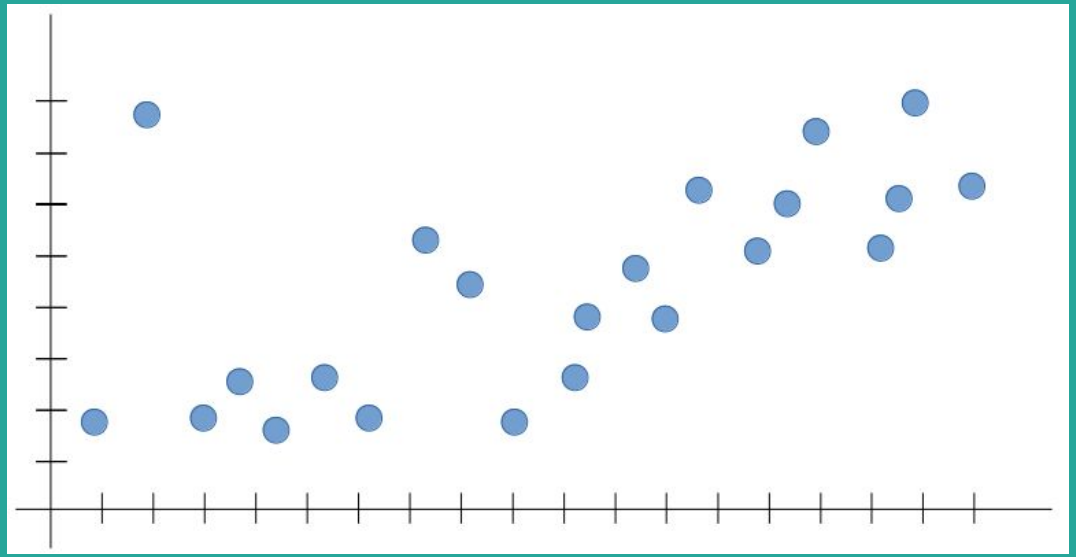


Overfitting and Underfitting

Arguably, Machine Learning models have one sole purpose; to generalize well.
Generalization is the model's ability to give sensible outputs to sets of input that it has never seen before.
Building on that idea, terms like overfitting and underfitting refer to deficiencies that the model's performance might suffer from.

A model that generizes well is the model that is neither overfit not underfit.

Example

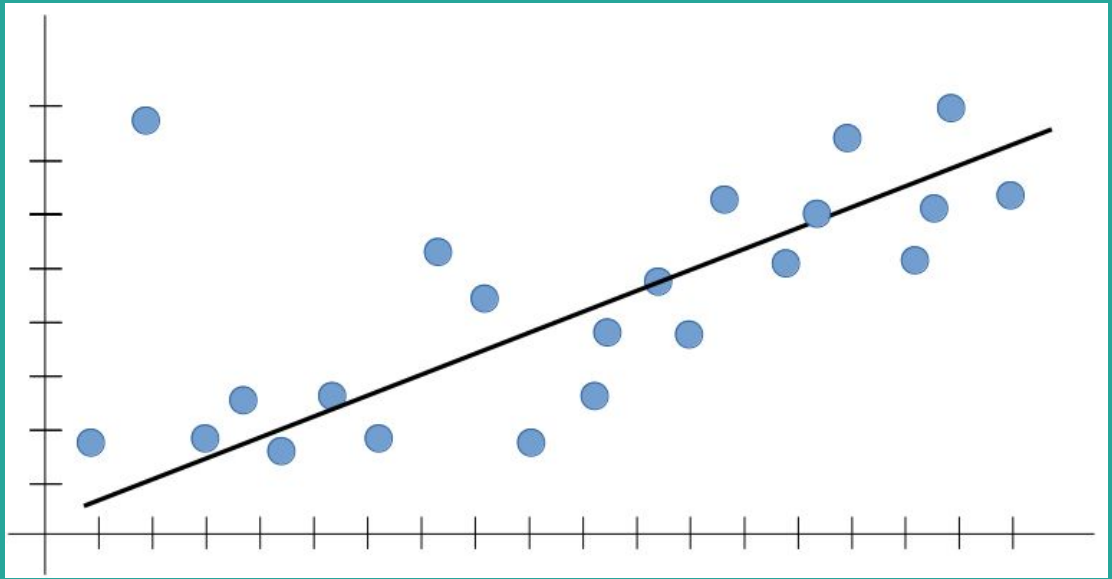


Let's say we're trying to build a Linear Regression Machine Learning model for the following data set.

Linear Regression allows us to map numeric inputs to numeric outputs, fitting a line into the data points.

The training stage

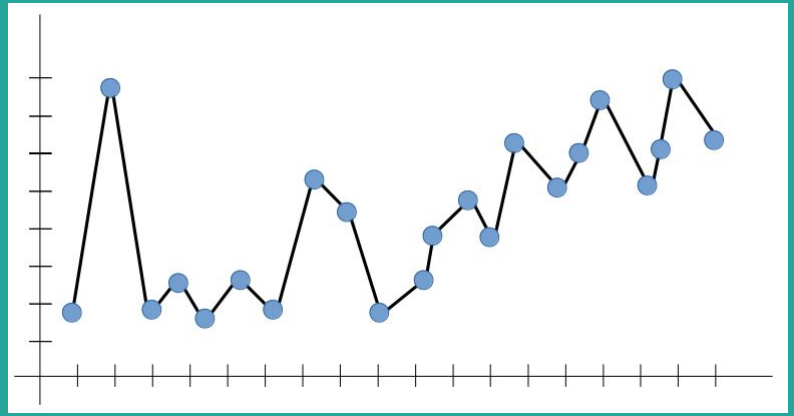
Training the Linear Regression model in our example is all about minimizing the total distance (i.e. cost) between the line we're trying to fit and the actual data points.



Even though the overall cost is not minimal, the line above fits within the trend very well, making the model reliable. **The line above could give a very likely prediction for the new input, as, in terms of Machine Learning, the outputs are expected to follow the trend seen in the training set.**

Overfitting (High Variance)

When we run our training algorithm on the data set, we allow the overall cost (i.e. distance from each point to the line) to become smaller with more iterations. Leaving this training algorithm run for long leads to minimal overall cost. However, this means that the line will be fit into all the points (including noise), catching secondary patterns that may not be needed for the generalizability of the model.



If the model does not capture the dominant trend that we can all see (positively increasing, in our case), it can't predict a likely output for an input that it has never seen before — defying the purpose of Machine Learning to begin with!

Overfitting is the case where the overall cost is really small, but the generalization of the model is unreliable. This is due to the model learning "too much" from the training data set.

The more we leave the model training the higher the chance of overfitting occurring. We always want to find the trend, not fit the line to all the data points.

In a nutshell, Overfitting – High variance and low bias

Techniques to reduce overfitting :

1. Increase training data.
2. Reduce model complexity.
3. Early stopping during the training phase (have an eye over the loss over the training period as soon as loss begins to increase stop training).
4. Ridge Regularization (L2) and Lasso Regularization(L1)
5. Use dropout for neural networks to tackle overfitting.

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

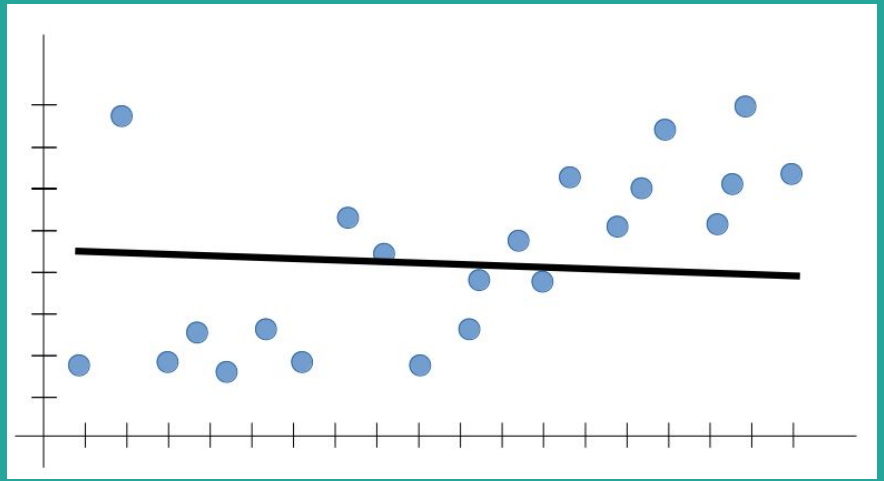
L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Underfitting (High Bias)

We want the model to learn from the training data, but we don't want it to learn too much (i.e. too many patterns). One solution could be to stop the training earlier. However, this could lead the model to not learn enough patterns from the training data, and possibly not even capture the dominant trend. This case is called underfitting.

Underfitting is the case where the model has " not learned enough" from the training data, resulting in low generalization and unreliable predictions.



In a nutshell, **Underfitting – High bias and low variance**

Techniques to reduce underfitting :

1. Increase model complexity

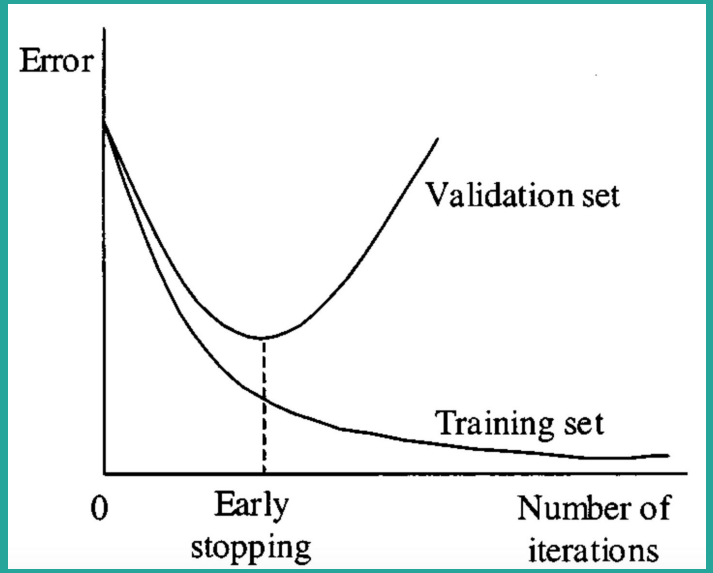
2. Increase number of features, performing feature engineering

3. Remove noise from the data.

4. Increase the number of epochs or increase the duration of training to get better results.

Early Stopping - for best fit!

With the passage of time, our model will keep on learning and thus the error for the model on the training and testing data will keep on decreasing. If it will learn for too long, the model will become more prone to overfitting due to the presence of noise and less useful details. Hence the performance of our model will decrease. In order to get a good fit, we will stop at a point just before where the error starts increasing. At this point the model is said to have good skills on training datasets as well as our unseen testing dataset.



Cross Validation

You need some kind of assurance that your model has got most of the patterns from the data correct, and its not picking up too much on the noise, or in other words its low on bias and variance.

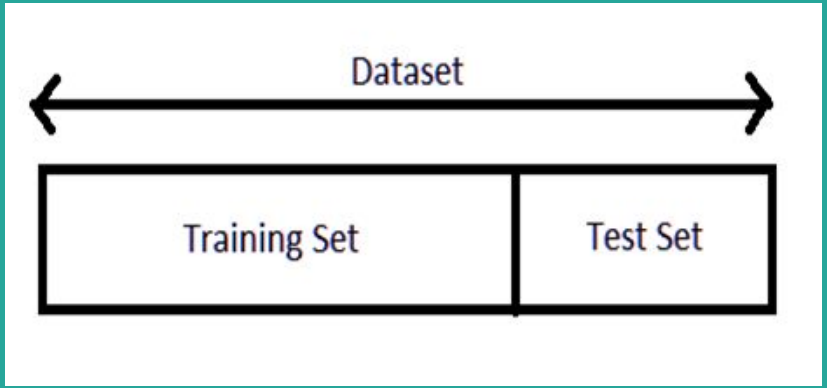
Cross Validation is *a very useful technique for assessing the effectiveness of your model, particularly in cases where you need to mitigate overfitting.*

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 :

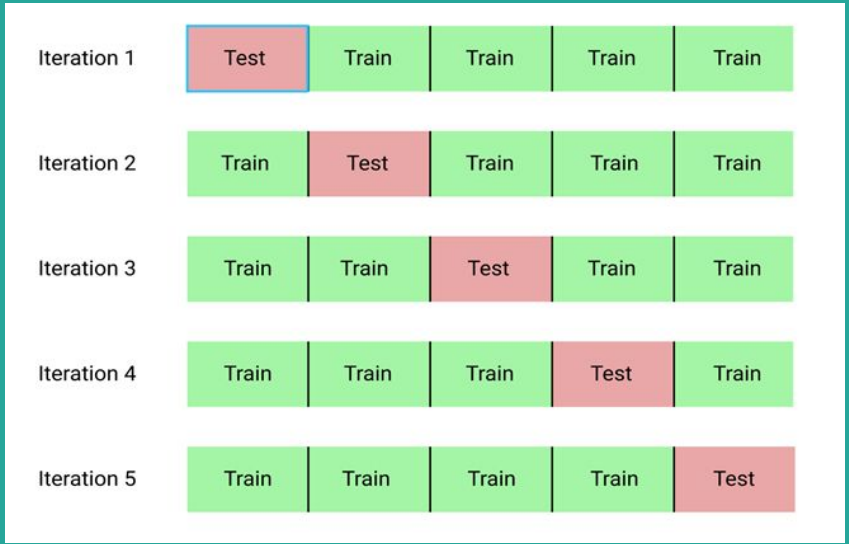
1. Split data set into training and test set
2. Using the training set train the model.
3. Test the model using the test set

USE: To get good out of sample accuracy



K-fold cross validation

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.



Hyperparameter Tuning

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:

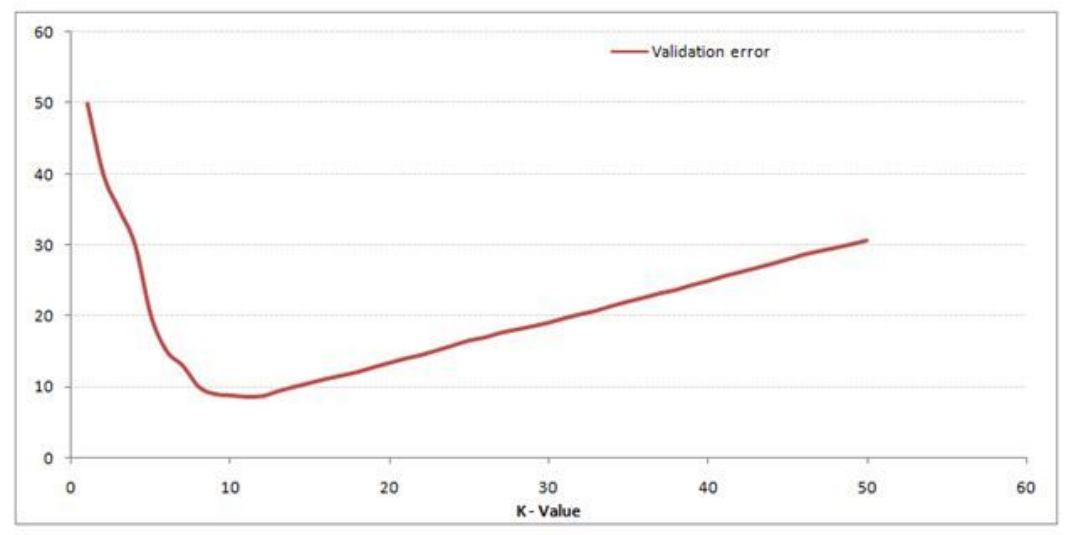
- **Learning** Rate.
- Number of Epochs.
- Regularization constant.
- Number of branches in a decision tree.
- Number of clusters in a clustering algorithm (like k-means)

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. GridSearchCV tries all the combinations of the values passed in the dictionary and evaluates the model for each combination using the Cross-Validation method. Hence after using this function we get accuracy/loss for every combination of hyperparameters and we can choose the one with the best performance.

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

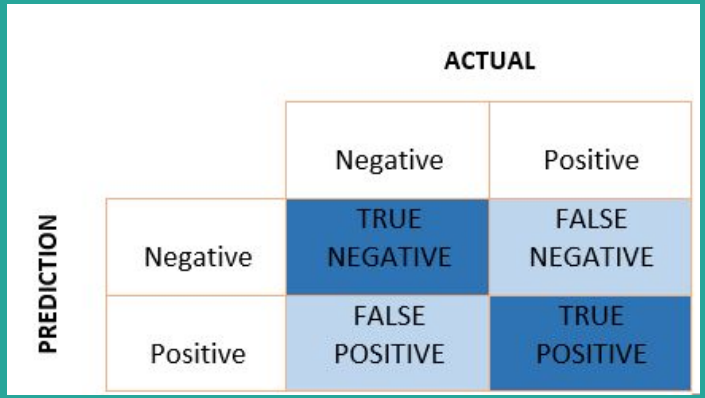


Model Evaluations

- Confusion Matrix
- Precision
- Recall
- F1 Score

What is a confusion matrix?

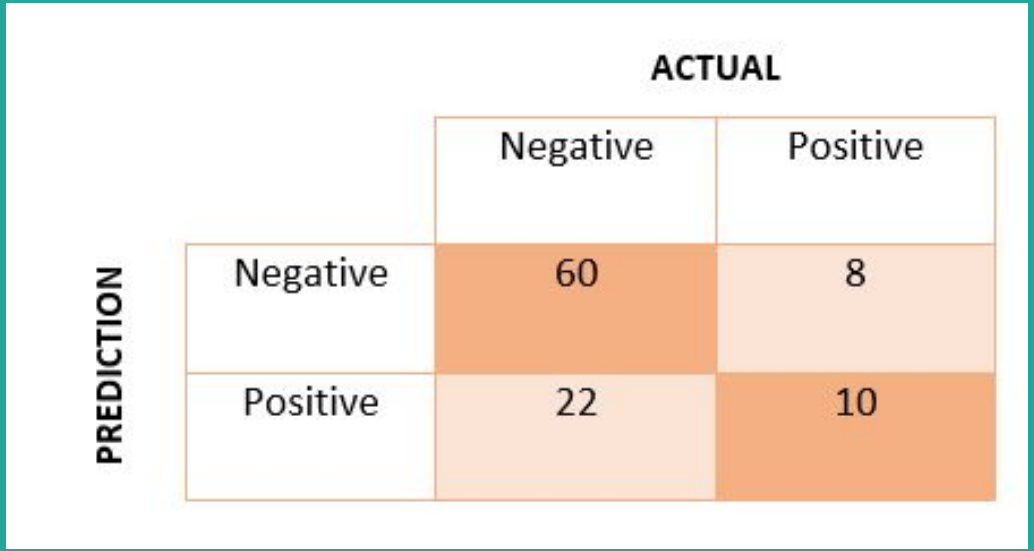
It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another.



Let's understand the confusing terms in the confusion matrix: **true positive, true negative, false negative, and false positive** with an example.

Example

A machine learning model is trained to predict tumor in patients. The test dataset consists of 100 people.



		ACTUAL	
		Negative	Positive
PREDICTION	Negative	60	8
	Positive	22	10

True Positive (TP) — model correctly predicts the positive class (prediction and actual both are positive). In the above example, **10 people** who have tumors are predicted positively by the model.

True Negative (TN) — model correctly predicts the negative class (prediction and actual both are negative). In the above example, **60 people** who don't have tumors are predicted negatively by the model.

False Positive (FP) — model gives the wrong prediction of the negative class (predicted-positive, actual-negative). In the above example, **22 people** are predicted as positive of having a tumor, although they don't have a tumor. FP is also called a **TYPE I** error.

False Negative (FN) — model wrongly predicts the positive class (predicted-negative, actual-positive). In the above example, **8 people** who have tumors are predicted as negative. FN is also called a **TYPE II** error.

With the help of these four values, we can calculate True Positive Rate (TPR), False Negative Rate (FPR), True Negative Rate (TNR), and False Negative Rate (FNR).

$$TPR = \frac{TP}{Actual\ Positive} = \frac{TP}{TP + FN}$$

$$FNR = \frac{FN}{Actual\ Positive} = \frac{FN}{TP + FN}$$

$$TNR = \frac{TN}{Actual\ Negative} = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{Actual\ Negative} = \frac{FP}{TN + FP}$$

Even if data is imbalanced, we can figure out that our model is working well or not. For that, **the values of TPR and TNR should be high, and FPR and FNR should be as low as possible.**

Precision

Out of all the positive predicted, what percentage is truly positive.

The precision value lies between 0 and 1.

$$Precision = \frac{TP}{TP + FP}$$

Recall

Out of the total positive, what percentage are predicted positive. It is the same as TPR (true positive rate).

$$Recall = \frac{TP}{TP + FN}$$

F1 Score

It is the harmonic mean of precision and recall. It takes both false positive and false negatives into account. Therefore, it performs well on an imbalanced dataset.

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

There is a **weighted F1 score** in which we can give different weightage to recall and precision. Different problems give different weightage to recall and precision.

$$F_\beta = (1 + \beta^2) * \frac{(Precision * Recall)}{(\beta^2 * Precision) + Recall}$$

Beta represents how many times recall is more important than precision. If the recall is twice as important as precision, the value of Beta is 2.

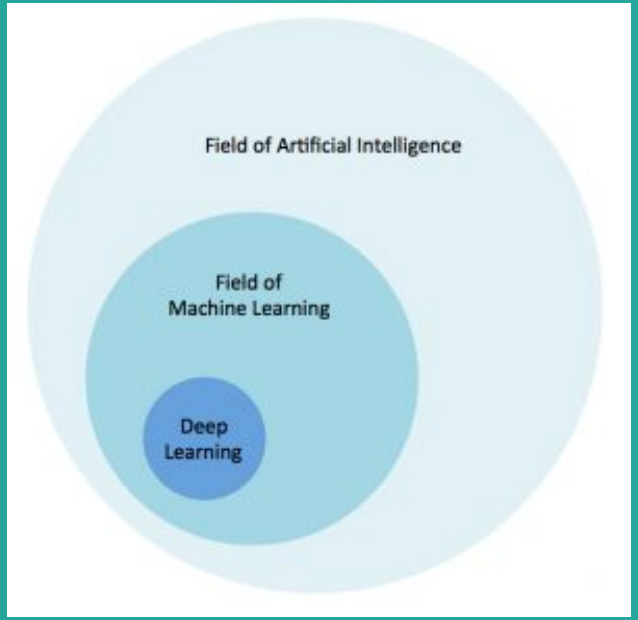
Deep Learning -Introduction

What is Deep Learning?

To understand what deep learning is, we first need to understand the relationship deep learning has with machine learning, neural networks, and artificial intelligence.

The best way to think of this relationship is to visualize them as concentric circles.

Deep learning is a specific subset of Machine Learning, which is a specific subset of Artificial Intelligence.

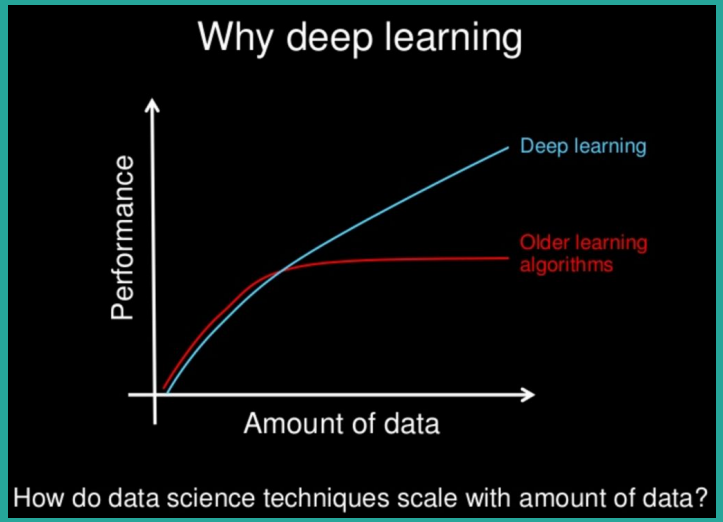


For individual definitions:

- Artificial Intelligence is the broad mandate of creating machines that can think intelligently.
- Machine Learning is one way of doing that, by using algorithms to glean insights from data .
- Deep Learning is one way of doing that, using a specific algorithm called a Neural Network.

Why is Deep Learning Important?

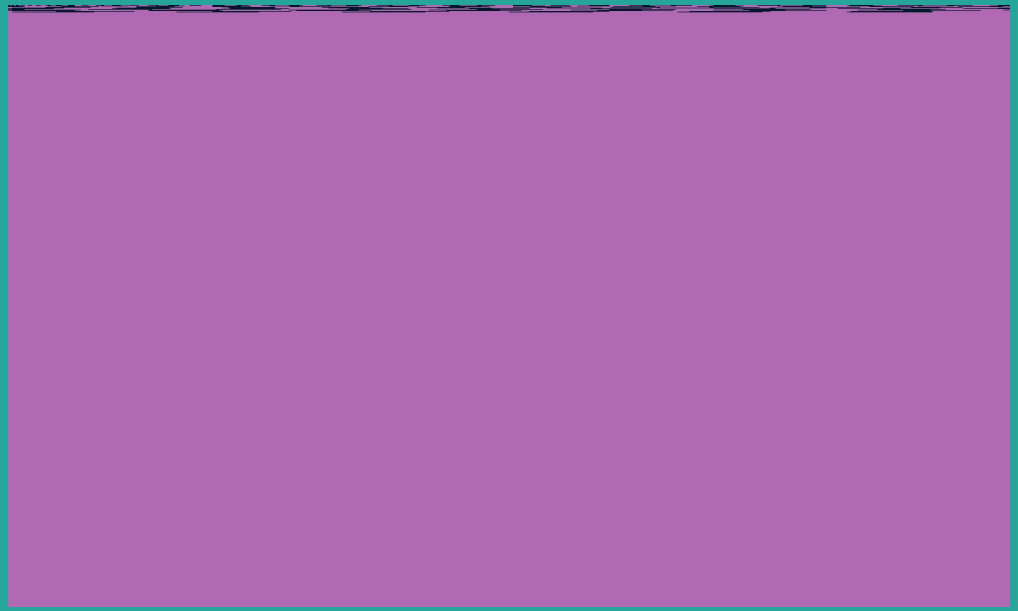
Deep Learning is important for one reason, and one reason only: we've been able to achieve meaningful, useful accuracy on tasks that matter. Machine Learning has been used for classification on images and text for decades, but it struggled to cross the threshold – there's a baseline accuracy that algorithms need to have to work in business settings. Deep Learning is finally enabling us to cross that line in places we weren't able to before.



Intuition

Given a large dataset of input and output pairs, a deep learning algorithm will try to minimize the difference between its prediction and expected output. By doing this, it tries to learn the association/pattern between given inputs and outputs — this in turn allows a deep learning model to generalize to inputs that it hasn't seen before.

As another example, let's say that inputs are images of dogs and cats, and outputs are labels for those images (i.e. is the input picture a dog or a cat). If an input has a label of a dog, but the deep learning algorithm predicts a cat, then my deep learning algorithm will learn that the features of my given image (e.g. sharp teeth, facial features) are going to be associated with a dog.



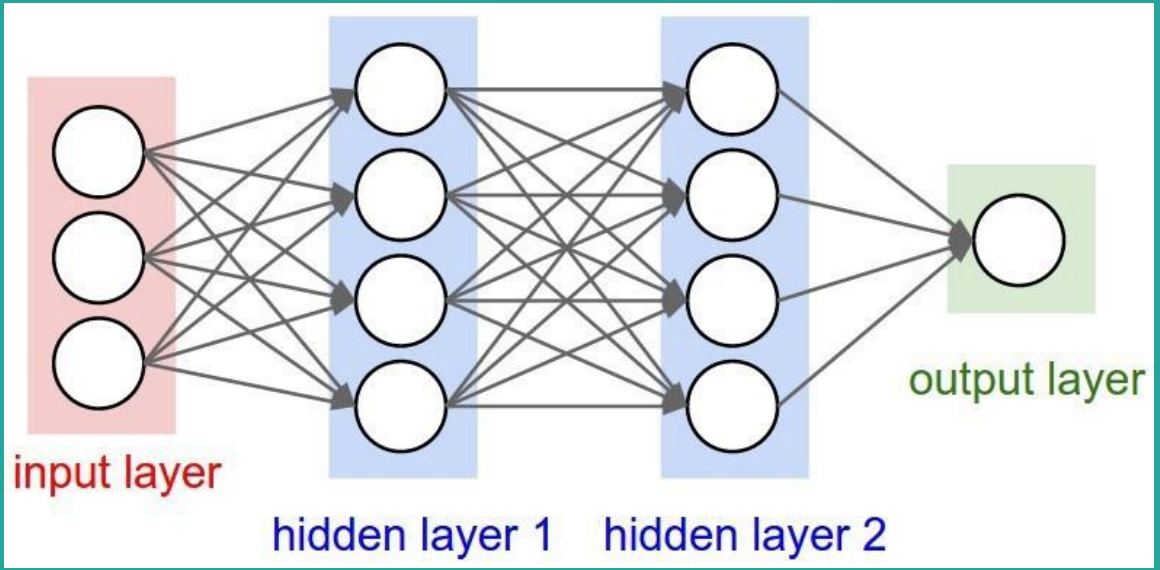
So why is it called “Deep” Learning?

The “deep” part of deep learning refers to creating deep neural networks. This refers a neural network with a large amount of layers — with the addition of more weights and biases, the neural network improves its ability to approximate more complex functions.

How Do Deep Learning algorithms “learn”?

- Neural Networks

Deep Learning Algorithms use something called a neural network to find associations between a set of inputs and outputs. The basic structure is seen below:



Neural Network Architecture

Input Nodes (input layer): No computation is done here within this layer, they just pass the information to the next layer (hidden layer most of the time). A block of nodes is also called layer.

Hidden nodes (hidden layer): In Hidden layers is where intermediate processing or computation is done, they perform computations and then transfer the weights (signals or information) from the input layer to the following layer (another hidden layer or to the output layer).

Output Nodes (output layer): Here we finally use an activation function that maps to the desired output format (e.g. softmax for classification).

Connections and weights: The network consists of connections, each connection transferring the output of a neuron i to the input of a neuron j. In this sense i is the predecessor of j and j is the successor of i, Each connection is assigned a weight Wij.

Activation function: the activation function of a node defines the output of that node given an input or set of inputs.

Learning rule: The learning rule is a rule or an algorithm which modifies the parameters of the neural network, in order for a given input to the network to produce a favored output.

Feedforward Neural Network

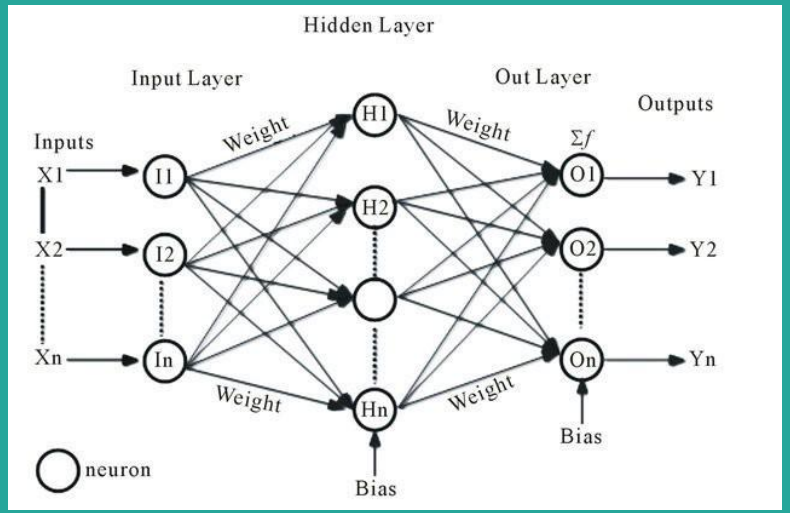
A feedforward neural network is an artificial neural network where connections between the units do *not* form a cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network.

Perceptron

The basic unit of computation in a neural network is the perceptron, often called a node or unit. It receives input from some other nodes, or from an external source and computes an output. Each input has an associated weight (w), which is assigned on the basis of its relative importance to other inputs. The node applies an activation function to the weighted sum of its inputs to get the output.

Multi-layer perceptron (MLP)

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function. MLP are very more useful and one good reason is that, they are able to learn non-linear representations



Activation Functions

Activation functions also known as transfer function is used to map input nodes to output nodes in certain fashion.
They are used to impart non linearity .
There are many activation functions used in Machine Learning out of which commonly used are listed below.

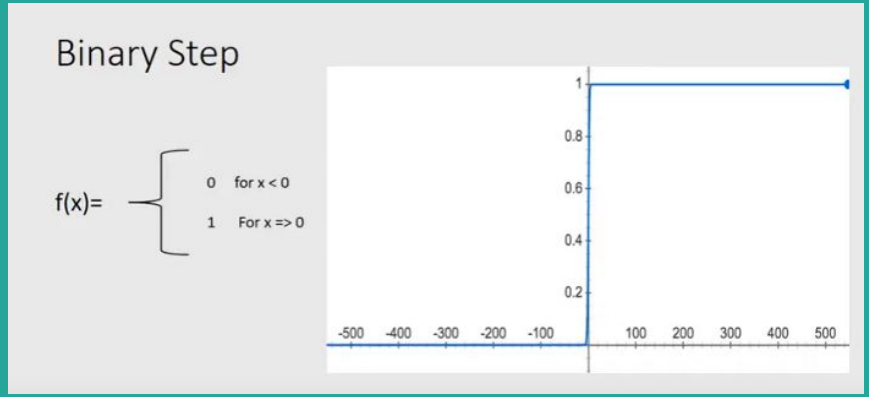
Identity or linear activation function

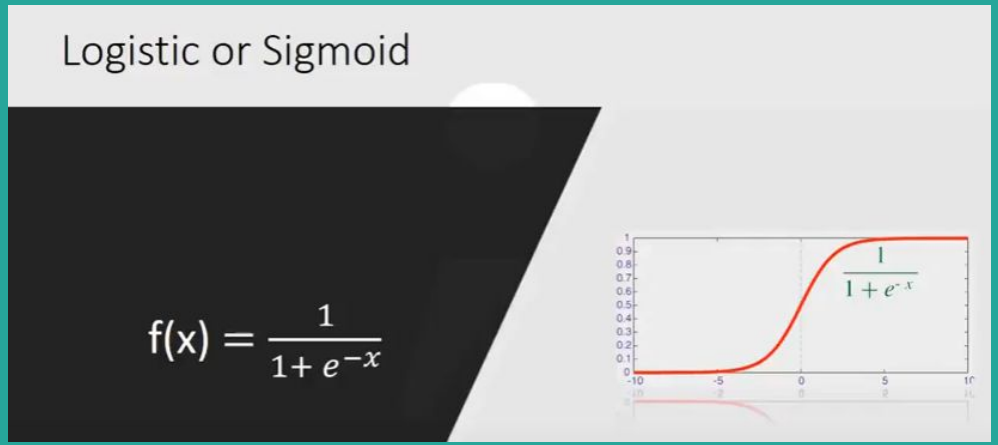
Input maps to same output.

F(x) = x

Binary Step

Very useful in classifiers



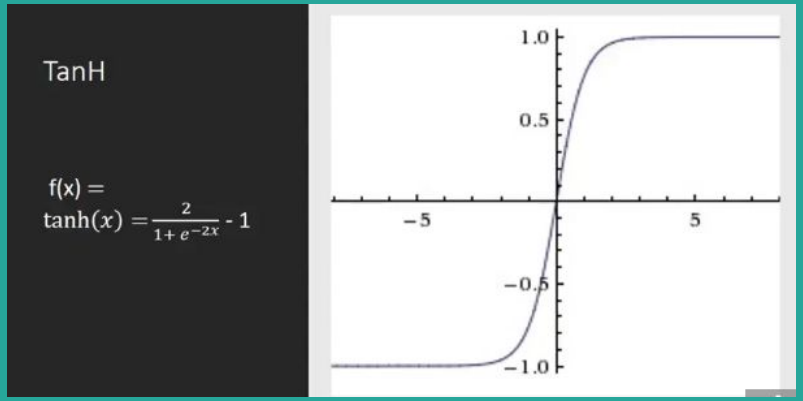


Sigmoid

Maps any sized inputs to outputs in range [0,1].

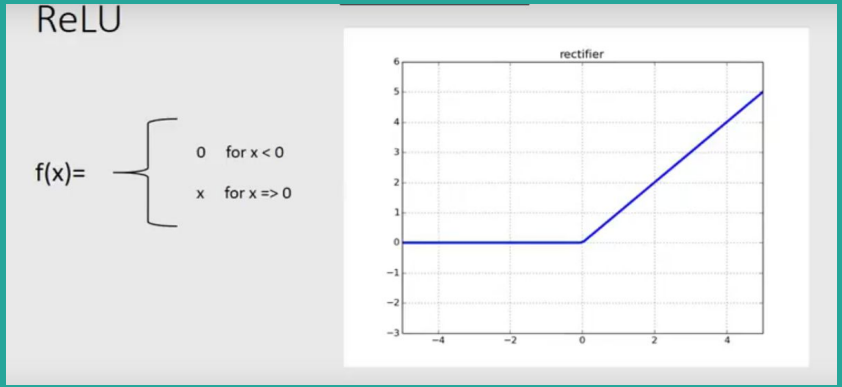
Tanh

→Maps input to output ranging in [-1,1].
→Similar to sigmoid function except it maps output in [-1,1] whereas sigmoid maps output to [0,1].



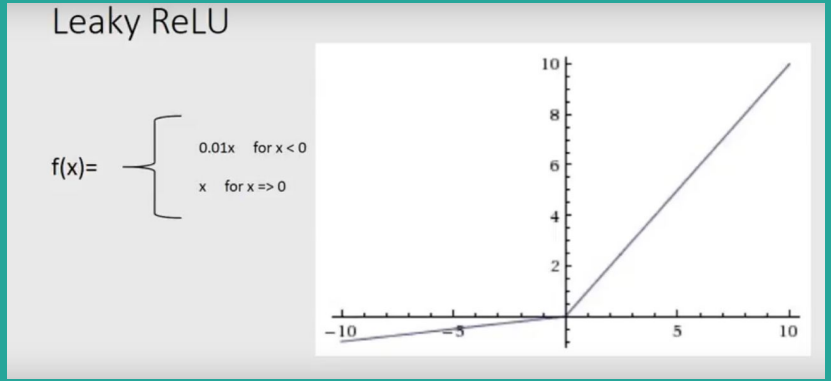
Rectified Linear Unit (ReLu)

→ It removes negative part of function.



Leaky ReLu

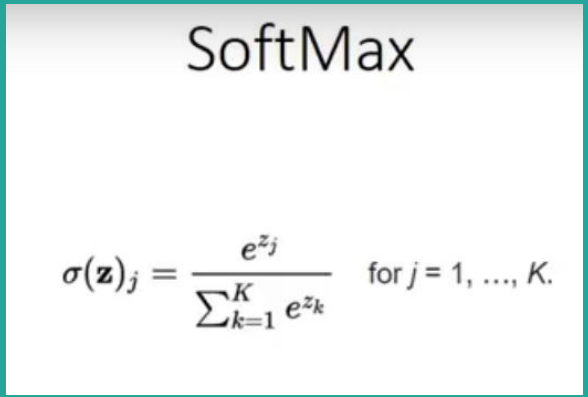
→ The only difference between ReLu and Leaky ReLu is it does not completely vanishes the negative part,it just lower its magnitude.



Softmax

→Softmax function is used to impart probabilities when you have more than one outputs you get probability distribution of outputs.

→Useful for finding most probable occurrence of output with respect to other outputs.



Backpropagation Algorithm

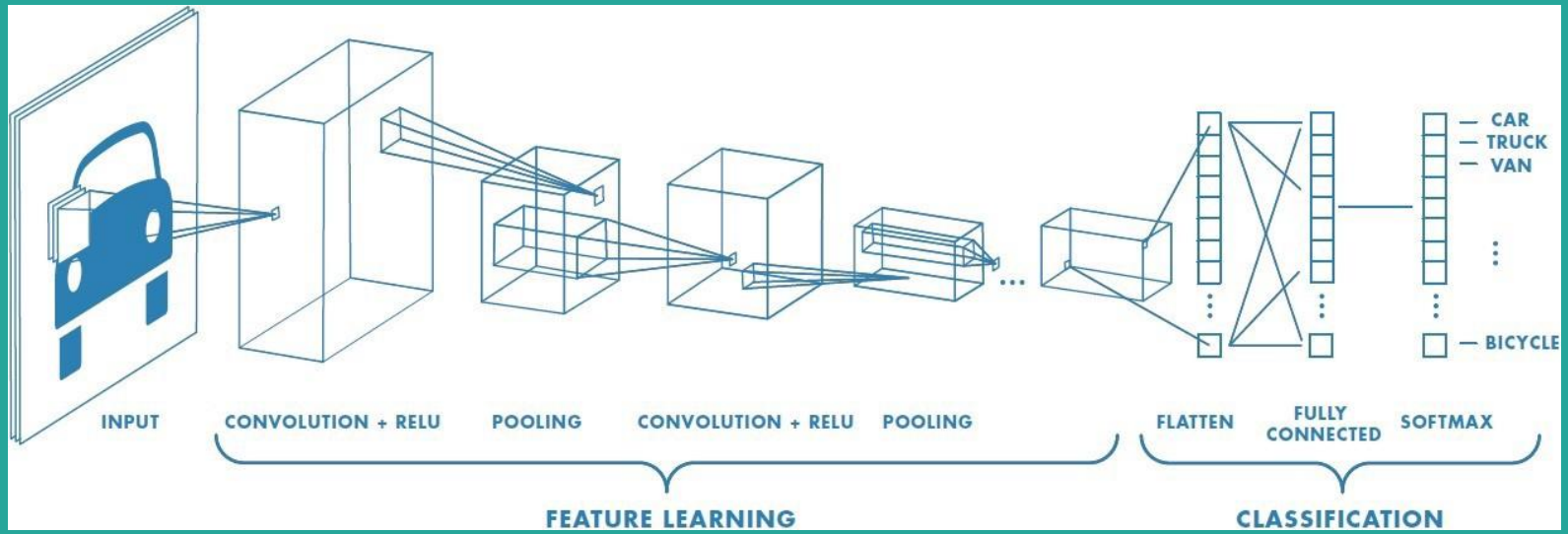
After the neural network passes its inputs all the way to its outputs, the network evaluates how good its prediction was (relative to the expected output) through something called a loss function.

The goal of my network is ultimately to minimize this loss by adjusting the weights and biases of the network. In using something called “back propagation” through gradient descent, the network backtracks through all its layers to update the weights and biases of every node in the opposite direction of the loss function — in other words, every iteration of back propagation should result in a smaller loss function than before.

The continuous updates of the weights and biases of the network ultimately turns it into a precise function approximator — one that models the relationship between inputs and expected outputs.

Convolutional Neural Networks

A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.



Why ConvNets over Feed-Forward Neural Nets?

An image is nothing but a matrix of pixel values, right? So why not just flatten the image (e.g. 3x3 image matrix into a 9x1 vector) and feed it to a Multi-Level Perceptron for classification purposes?

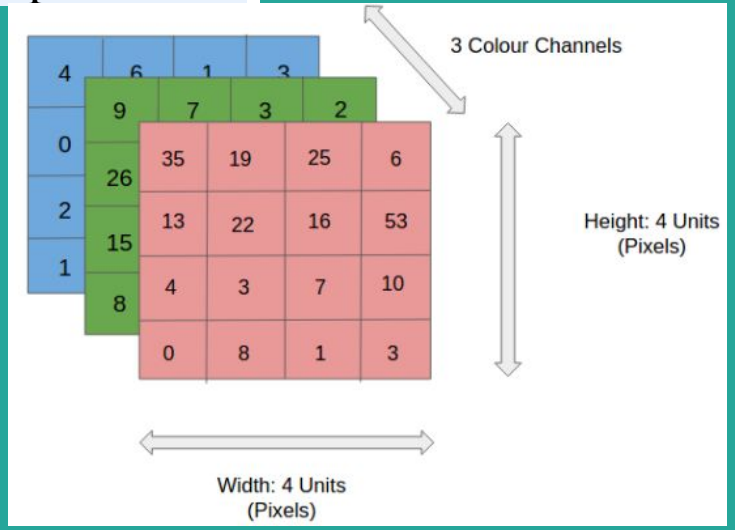
In cases of extremely basic binary images, the method might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout.

The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved through filters and reusability of weights.

Input Image

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue.

You can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction.



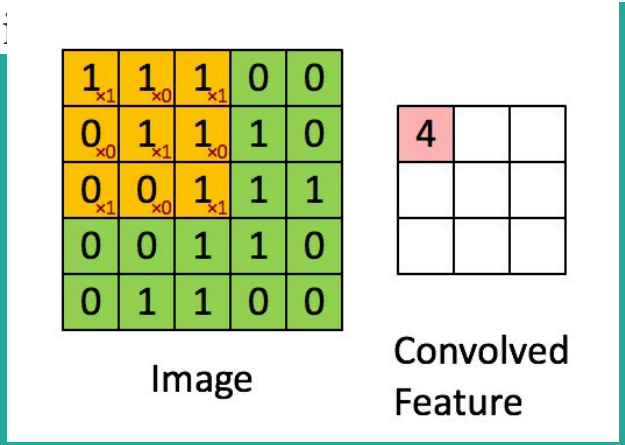
Convolution Layer — The Kernel

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

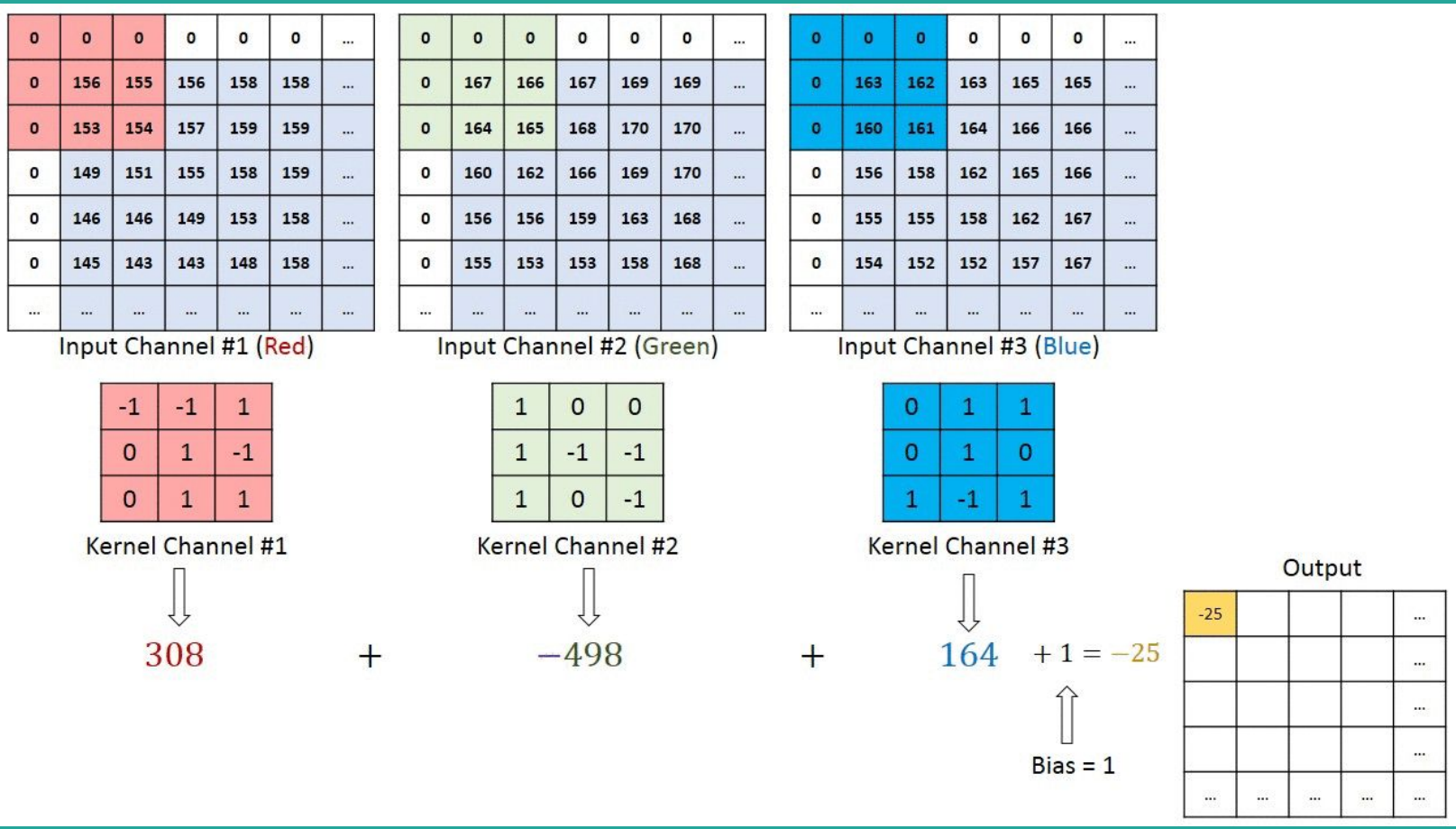
Kernel/Filter, K =

1 0 1
0 1 0
1 0 1

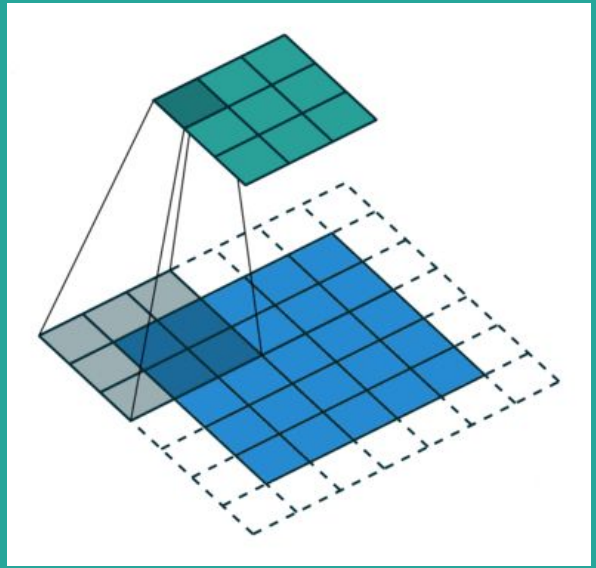
The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel i



In the case of images with multiple channels (e.g. RGB), the Kernel has the same depth as that of the input image. Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.


0 0 0 0 0 0 ...
0 156 155 156 158 158 ...
0 153 154 157 159 159 ...
0 149 151 155 158 159 ...
0 146 146 149 153 158 ...
0 145 143 143 148 158 ...
... ...
Input Channel #1 (Red)
0 0 0 0 0 0 ...
0 167 166 167 169 169 ...
0 164 165 168 170 170 ...
0 160 162 166 169 170 ...
0 156 156 159 163 168 ...
0 155 153 153 158 168 ...
... ...
Input Channel #2 (Green)
0 0 0 0 0 0 ...
0 163 162 163 165 165 ...
0 160 161 164 166 166 ...
0 156 158 162 165 166 ...
0 155 155 158 162 167 ...
0 154 152 152 157 167 ...
... ...
Input Channel #3 (Blue)
-1 -1 1
0 1 -1
0 1 1
Kernel Channel #1
1 0 0
1 -1 -1
1 0 -1
Kernel Channel #2
0 1 1
0 1 0
1 -1 1
Kernel Channel #3
308
+
−498
+
164
+ 1 = −25
Bias = 1
Output
-25 ...
... ...
... ...
... ...
... ...

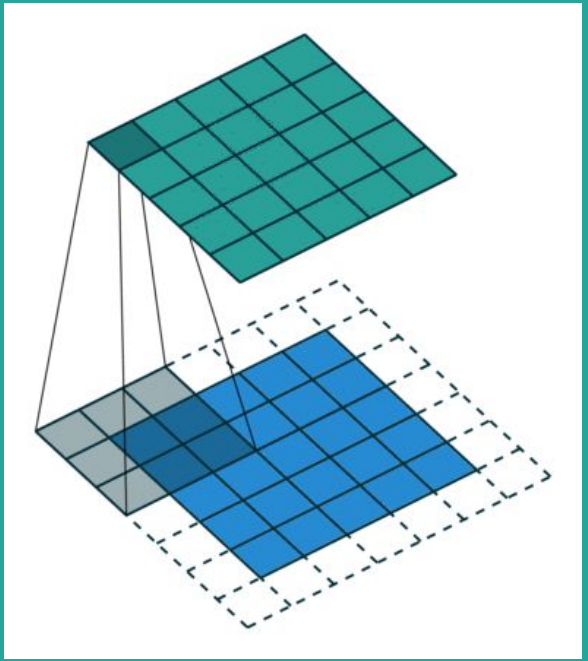
The objective of the Convolution Operation is to extract the high-level features such as edges, from the input image. ConvNets need not be limited to only one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network which has the wholesome understanding of images in the dataset, similar to how we would.



Padding

When we augment the 5x5x1 image into a 6x6x1 image and then apply the 3x3x1 kernel over it, we find that the convolved matrix turns out to be of dimensions 5x5x1. Hence the name — **Same Padding**. In the example, 5x5x1 image is padded with 0s to create a 6x6x1 image

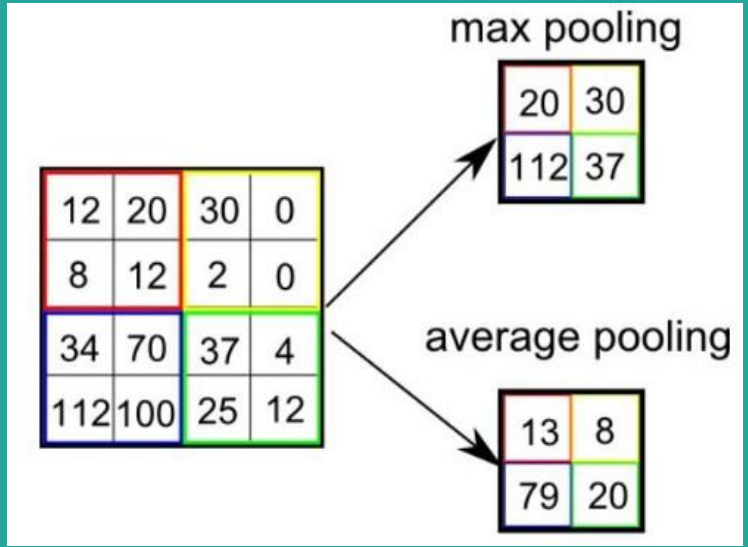
On the other hand, if we perform the same operation without padding, we are presented with a matrix which has dimensions of the Kernel (3x3x1) itself — **Valid Padding**.



Pooling Layer

Similar to the Convolutional Layer, the Pooling layer is responsible for reducing the spatial size of the Convolved Feature. This is to **decrease the computational power required to process the data** through dimensionality reduction. Furthermore, it is useful for **extracting dominant features** which are rotational and positional invariant, thus maintaining the process of effectively training of the model.

There are two types of Pooling: Max Pooling and Average Pooling. **Max Pooling** returns the **maximum value** from the portion of the image covered by the Kernel. On the other hand, **Average Pooling** returns the **average of all the values** from the portion of the image covered by the Kernel.

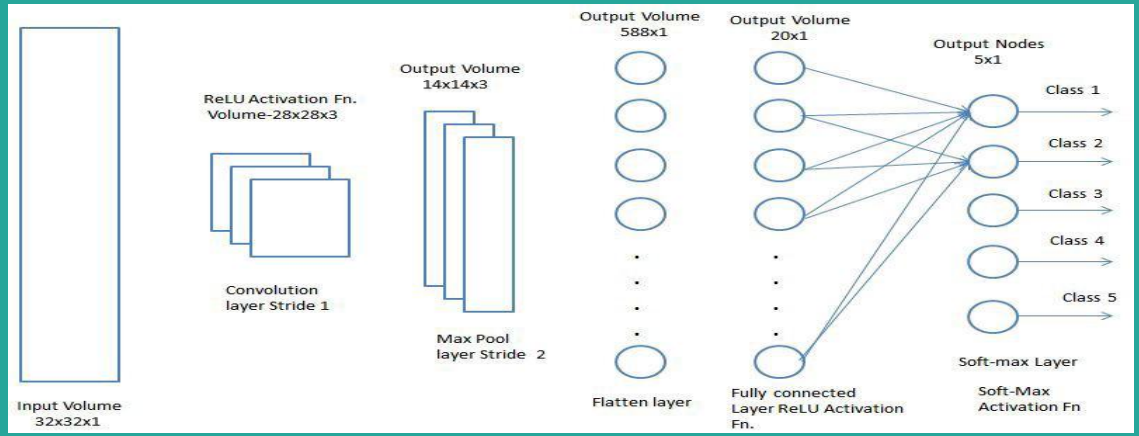


The Convolutional Layer and the Pooling Layer, together form the i-th layer of a Convolutional Neural Network. Depending on the complexities in the images, the number of such layers may be increased for capturing low-levels details even further, but at the cost of more computational power.

After going through the above process, we have successfully enabled the model to understand the features. Moving on, we are going to flatten the final output and feed it to a regular Neural Network for classification purposes.

Fully Connected Layer (FC Layer)

Adding a Fully-Connected layer is a (usually) cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolutional layer. The Fully-Connected layer is learning a possibly non-linear function in that space.
Now that we have converted our input image into a suitable form for our Multi-Level Perceptron, we shall flatten the image into a column vector. The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training. Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.



Object Detection

How do we make the machine to identify an object? That's what gave rise to the domain of Computer Vision that we call **"Object Detection"**. Object detection is a field of Computer Vision and Image Processing that deals with detecting instances of various classes of objects (like a person, book, chair, car, bus, etc.) in a digitally captured Image or Video.
This domain is further divided into sub-domains like Face detection, Activity recognition, Image annotation, and many more. Object Detection has found its applications in various important areas like Self-Driving cars, robots, Video Surveillance, Object Tracking, etc.

YOLO Algorithm (You Only Look Once)

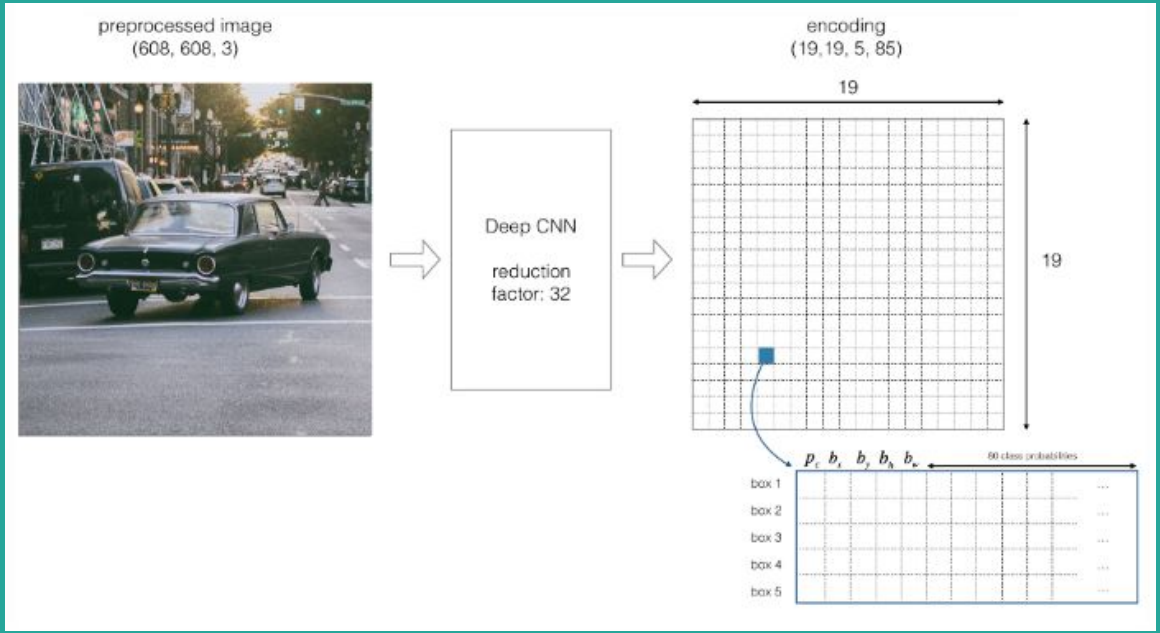
YOLO algorithm is an algorithm used for fast real-time object detection based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in **one run of the Algorithm.**

To understand the YOLO algorithm, first we need to understand what is actually being predicted. Ultimately, we aim to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

1. Center of the box (**bx, by**)
2. Width (**bw**)
3. Height (**bh**)
4. Value **c** corresponding to the class of an object

Along with that we predict a real number **pc**, which is the probability that there is an object in the bounding box.

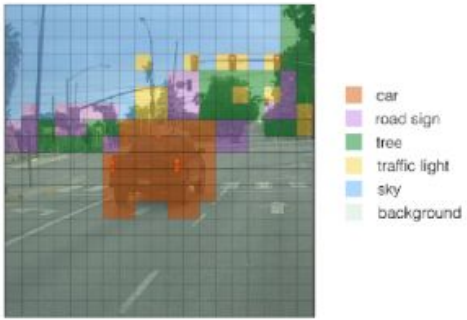
YOLO doesn’t search for interested regions in the input image that could contain an object, instead it splits the image into cells, typically 19x19 grid. Each cell is then responsible for predicting K bounding boxes.



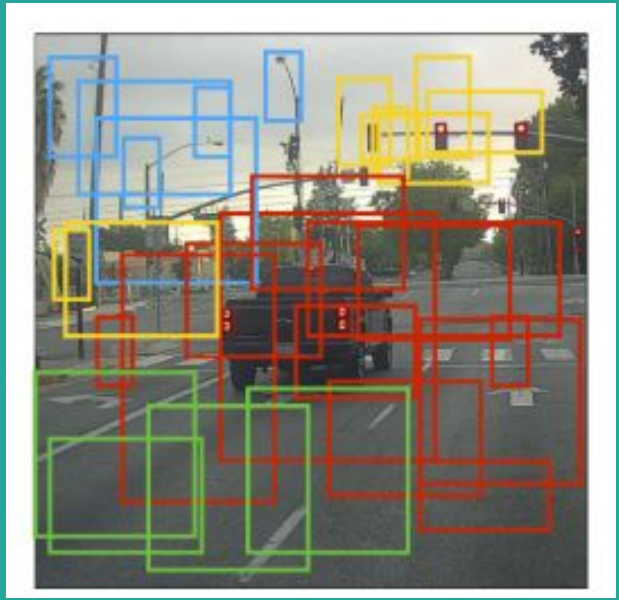
During the one pass of forwards propagation, YOLO determines the probability that the cell contains a certain class. The equation for the same is :

$$score_{c,i} = p_c \times c_i$$

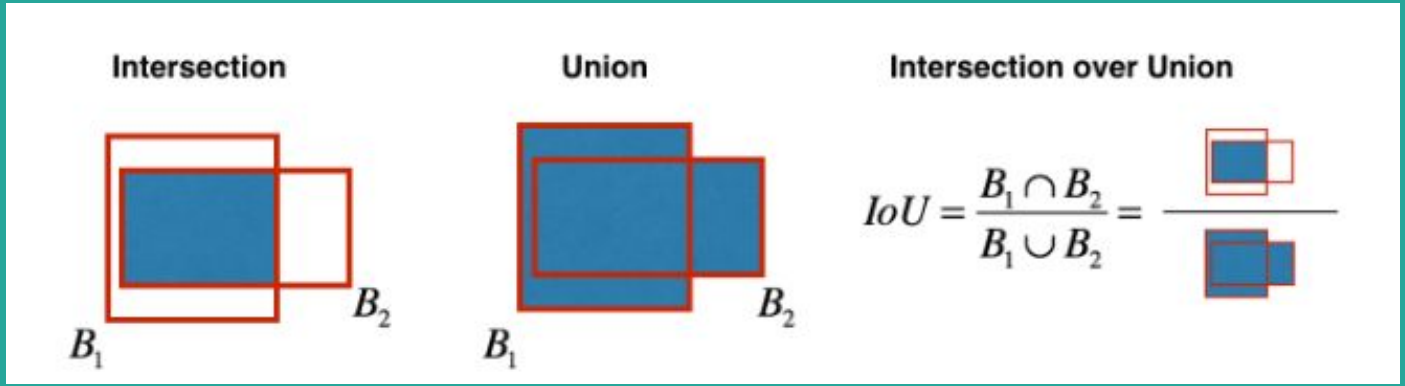
The class with the maximum probability is chosen and assigned to that particular grid cell. Similar process happens for all the grid cells present in the image.
After computing the above class probabilities, the image may look like this :



This shows the before and after of predicting the class probabilities for each grid cell. After predicting the class probabilities, the next step is Non-max suppression, it helps the algorithm to get rid of the unnecessary anchor boxes, like you can see that in the figure below, there are numerous anchor boxes calculated based on the class probabilities.

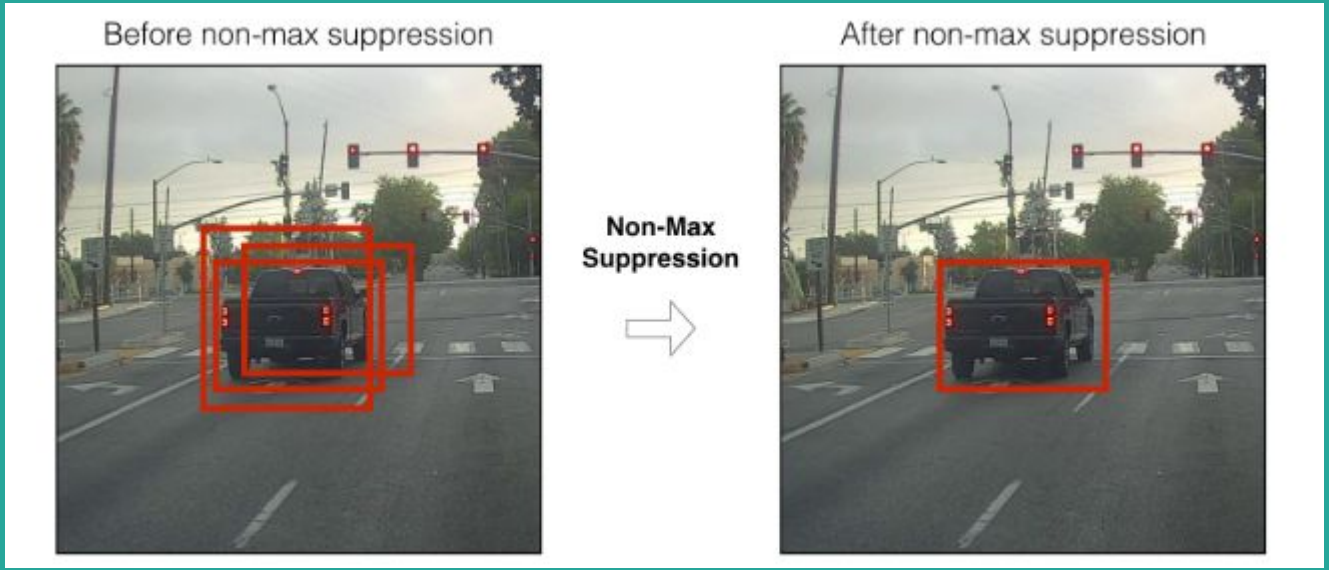


To resolve this problem Non-max suppression eliminates the bounding boxes that are very close by preforming the IoU (Intersection over Union) with the one having the highest class probability among them.



It calculates the value of IoU for all the bounding boxes respective to the one having the highest class probability, it then rejects the bounding boxes whose value of IoU is greater than a threshold. It signifies that those two bounding boxes are covering the same object but the other one has a low probabilty for the same, thus it is eliminated.

Once done, algorithm finds the bunding box with next highest class probabilities and does the same process, it is done until we are left with all the different bounding boxes.



After this, almost all of our work is done, the algorithm finally outputs the required vector showing the details of the bounding box of the respective class. The overall architecture of the algorithm can be viewed below :