UNIT-II

Overview of Machine Learning:

Machine learning is a branch of artificial intelligence, a science that researches machines to acquire new knowledge and new skills and to identify existing knowledge. The precise definition of machine learning is:

It's a computer program learning 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 E: Tom Mitchell 1998

Machine learning has been widely used in data mining, computer vision, natural language processing, biometrics, search engines, medical diagnostics, detection of credit card fraud, securities market analysis, DNA sequence sequencing, speech and handwriting recognition, strategy games and robotics.

Machine learning algorithm classification:

According to the way of learning, machine learning mainly includes:

- **Supervised learning**: Input data is tagged. Supervised learning establishes a learning process, compares the predicted results with the actual results of the "training data" (ie, input data), and continuously adjusts the predictive model until the predicted results of the model reach an expected accuracy, such as classification and regression problems. . Common algorithms include decision trees, Bayesian classification, least squares regression, logistic regression, support vector machines, neural networks, and so on.

Advantages of Supervised learning:

- With the help of supervised learning, the model can predict the output on the basis of prior experiences.
- In supervised learning, we can have an exact idea about the classes of objects.
- Supervised learning model helps us to solve various real-world problems such as **fraud detection, spam filtering**, etc.

Disadvantages of supervised learning:

- Supervised learning models are not suitable for handling the complex tasks.
- Supervised learning cannot predict the correct output if the test data is different from the training dataset.
- Training required lots of computation times.
- In supervised learning, we need enough knowledge about the classes of object.

Unsupervised learning: Input data has no tags, but algorithms to infer the intrinsic links of data, such as clustering and association rule learning. Common algorithms include independent component analysis, K-Means and Apriori algorithms.

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.
- Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

Semi-supervised learning: input data part tags, is an extension of supervised learning, often used for classification and regression. Common algorithms include graph theory inference algorithms, Laplacian support vector machines, and so on.

Reinforcement learning: Input data as feedback to the model, emphasizing how to act based on the environment to maximize the expected benefits. The difference between supervised learning is that it does not require the correct input/output pairs and does not require precise correction of sub-optimal behavior. Reinforcement learning is more focused on online planning and requires a balance between exploration (in the unknown) and compliance (existing knowledge).

Types of Reinforcement: There are two types of Reinforcement:

1. **Positive –**
   Positive Reinforcement is defined as when an event, occurs due to a particular behavior, increases the strength and the frequency of the behavior. In other words, it has a positive effect on behavior.
   Advantages of reinforcement learning are:

   - Maximizes Performance
   - Sustain Change for a long period of time

Disadvantages of reinforcement learning:

- Too much Reinforcement can lead to overload of states which can diminish the results

2. **Negative –**
   Negative Reinforcement is defined as strengthening of a behavior because a negative condition is stopped or avoided.
   Advantages of reinforcement learning:
   - Increases Behavior
   - Provide defiance to minimum standard of performance

Disadvantages of reinforcement learning:

- It Only provides enough to meet up the minimum behavior

Supervised Machine Learning

Supervised learning is the types of machine learning in which machines are trained using well "labelled" training data, and on basis of that data, machines predict the output. The labelled data means some input data is already tagged with the correct output.

In supervised learning, the training data provided to the machines work as the supervisor that teaches the machines to predict the output correctly. It applies the same concept as a student learns in the supervision of the teacher.

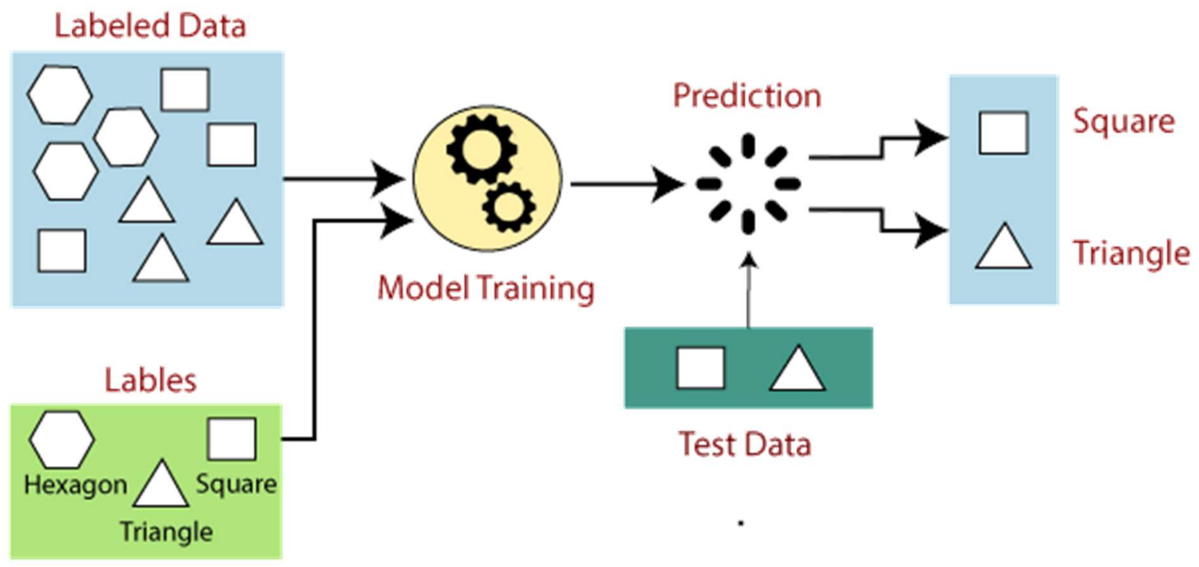
Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to **find a mapping function to map the input variable(x) with the output variable(y)**.

In the real-world, supervised learning can be used for **Risk Assessment, Image classification, Fraud Detection, spam filtering**, etc.

How Supervised Learning Works?

In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.

The working of Supervised learning can be easily understood by the below example and diagram:



Suppose we have a dataset of different types of shapes which includes square, rectangle, triangle, and Polygon. Now the first step is that we need to train the model for each shape.

- If the given shape has four sides, and all the sides are equal, then it will be labelled as a **Square**.
- If the given shape has three sides, then it will be labelled as a **triangle**.
- If the given shape has six equal sides then it will be labelled as **hexagon**.

Now, after training, we test our model using the test set, and the task of the model is to identify the shape.

The machine is already trained on all types of shapes, and when it finds a new shape, it classifies the shape on the bases of a number of sides, and predicts the output.

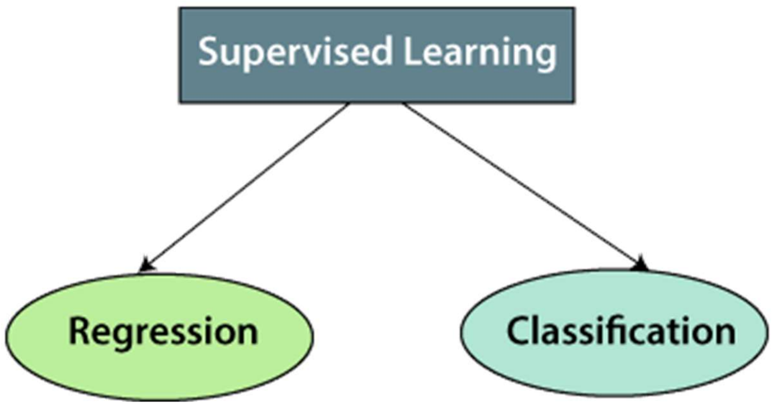
Steps Involved in Supervised Learning:

- First Determine the type of training dataset
- Collect/Gather the labelled training data.

- Split the training dataset into training **dataset, test dataset, and validation dataset**.
- Determine the input features of the training dataset, which should have enough knowledge so that the model can accurately predict the output.
- Determine the suitable algorithm for the model, such as support vector machine, decision tree, etc.
- Execute the algorithm on the training dataset. Sometimes we need validation sets as the control parameters, which are the subset of training datasets.
- Evaluate the accuracy of the model by providing the test set. If the model predicts the correct output, which means our model is accurate.

Types of supervised Machine learning Algorithms:

Supervised learning can be further divided into two types of problems:



1. Regression

Regression algorithms are used if there is a relationship between the input variable and the output variable. It is used for the prediction of continuous variables, such as Weather forecasting, Market Trends, etc. Below are some popular Regression algorithms which come under supervised learning:

- Linear Regression
- Regression Trees

- Non-Linear Regression
- Bayesian Linear Regression
- Polynomial Regression

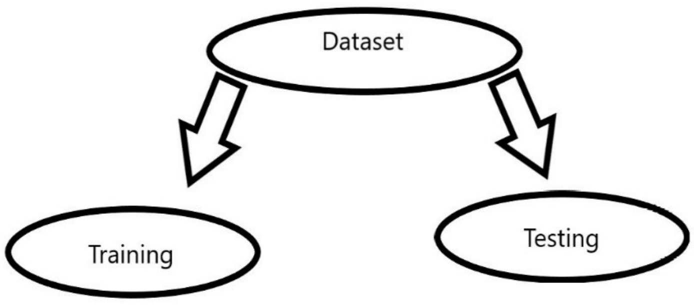
2. Classification

Classification algorithms are used when the output variable is categorical, which means there are two classes such as Yes-No, Male-Female, True-false, etc.

Spam Filtering,

- Random Forest
- Decision Trees
- Logistic Regression
- Support vector Machines

Splitting a dataset:



Train-Test Split Evaluation

The train-test split is a technique for evaluating the performance of a machine learning algorithm.

It can be used for classification or regression problems and can be used for any supervised learning algorithm.

The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model;

instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

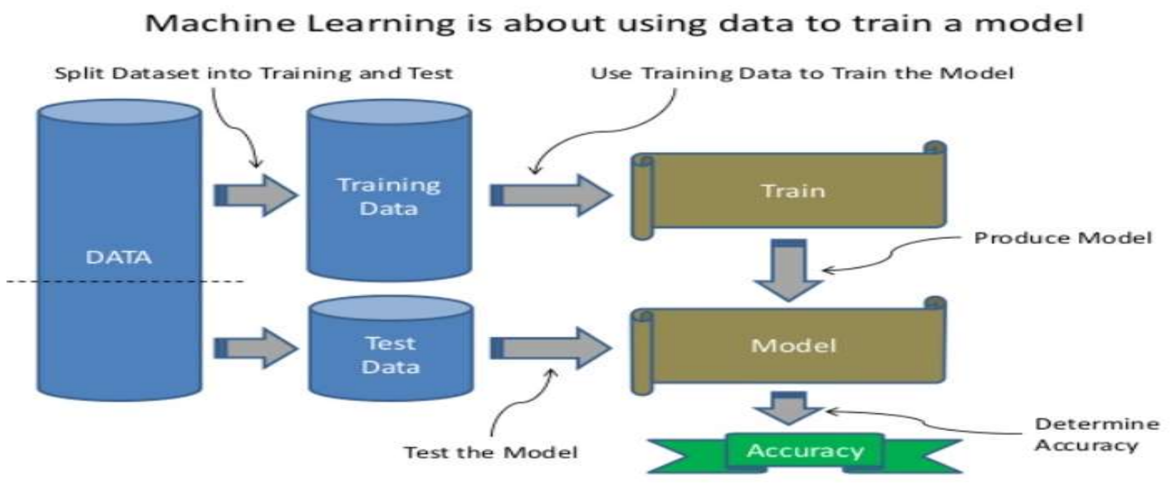
- **Train Dataset**: Used to fit the machine learning model.
- **Test Dataset**: Used to evaluate the fit machine learning model. The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values.

The train-test procedure is appropriate when there is a sufficiently large dataset available.

To train any machine learning model irrespective what type of dataset is being used you have to split the dataset into training data and testing data. So, let us look into how it can be done? Here I am going to use the 'train_test_split' library from sklearn

```
from sklearn.model_selection import train_test_split
```


Machine Learning is about using data to train a model
Split Dataset into Training and Test
Use Training Data to Train the Model
DATA
Training Data
Test Data
Train
Produce Model
Model
Determine Accuracy
Test the Model
Accuracy

The reason is that when the **dataset** is **split** into train and test sets, there will not be enough data in the training **dataset** for the model to learn an effective mapping of inputs to outputs. There will also not be enough data in the test set to effectively evaluate the model performance.

Bayes Theorem of Conditional Probability

Marginal probability is the probability of an event, irrespective of other random variables. If the random variable is independent, then it is the probability of the event directly, otherwise, if the variable is dependent upon other variables, then the marginal probability is the probability of the event summed over all outcomes for the dependent variables, called the sum rule.

- **Marginal Probability**: The probability of an event irrespective of the outcomes of other random variables, e.g. P(A).
  The joint probability is the probability of two (or more) simultaneous events, often described in terms of events A and B from two dependent random variables, e.g. X and Y. The joint probability is often summarized as just the outcomes, e.g. A and B.
- **Joint Probability**: Probability of two (or more) simultaneous events, e.g. P(A and B) or P(A, B).
  The conditional probability is the probability of one event given the occurrence of another event, often described in terms of events A and B from two dependent random variables e.g. X and Y.
- **Conditional Probability**: Probability of one (or more) event given the occurrence of another event, e.g. P(A given B) or P(A | B).
  The joint probability can be calculated using the conditional probability; for example:

  $$P(A, B) = P(A \mid B) * P(B)$$

  This is called the product rule. Importantly, the joint probability is symmetrical, meaning that:

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

  The conditional probability can be calculated using the joint probability; for example:

$$P(A \mid B) = P(A, B) / P(B)$$

The conditional probability is not symmetrical; for example:

$$P(A \mid B) != P(B \mid A)$$

Now, there is another way to calculate the conditional probability.

Specifically, one conditional probability can be calculated using the other conditional probability; for example:

$P(A|B) = P(B|A) * P(A) / P(B)$

The reverse is also true; for example:

$P(B|A) = P(A|B) * P(B) / P(A)$

This alternate approach of calculating the conditional probability is useful either when the joint probability is challenging to calculate (which is most of the time), or when the reverse conditional probability is available or easy to calculate.

This alternate calculation of the conditional probability is referred to as Bayes Rule or Bayes Theorem.

Bayes Theorem: Principled way of calculating a conditional probability without the joint probability.

It is often the case that we do not have access to the denominator directly, e.g. P(B).

We can calculate it an alternative way; for example:

$P(B) = P(B|A) * P(A) + P(B|\text{not } A) * P(\text{not } A)$

This gives a formulation of Bayes Theorem that we can use that uses the alternate calculation of P(B), described below:

$P(A|B) = P(B|A) * P(A) / P(B|A) * P(A) + P(B|\text{not } A) * P(\text{not } A)$

Or with brackets around the denominator for clarity:

P(A|B) = P(B|A) * P(A) / (P(B|A) * P(A) + P(B|not A) * P(not A))

P(not A) = 1 – P(A)
Additionally, if we have P(not B|not A), then we can calculate P(B|not A) as its complement; for example:

P(B|not A) = 1 – P(not B|not A)
Now that we are familiar with the calculation of Bayes Theorem, let’s take a closer look at the meaning of the terms in the equation.

Firstly, in general, the result P(A|B) is referred to as the **posterior probability** and P(A) is referred to as the **prior probability**.

P(A|B): Posterior probability.

P(A): Prior probability.

Sometimes P(B|A) is referred to as the **likelihood** and P(B) is referred to as the **evidence**.

P(B|A): Likelihood.

P(B): Evidence.

This allows Bayes Theorem to be restated as:

Posterior = Likelihood * Prior / Evidence

Bayes Theorem for Modeling Hypotheses:

Bayes Theorem is a useful tool in applied machine learning. It provides a way of thinking about the relationship between data and a model.

A machine learning algorithm or model is a specific way of thinking about the structured relationships in the data. In this way, a model can be thought of as a hypothesis about the relationships in the data, such as the relationship between input (*X*) and output (*y*). The practice of applied machine learning is the testing and analysis of different hypotheses (models) on a given dataset.

Bayes Theorem provides a probabilistic model to describe the relationship between data (*D*) and a hypothesis (h); for example:

P(h|D) = P(D|h) * P(h) / P(D)

Breaking this down, it says that the probability of a given hypothesis holding or being true given some observed data can be calculated as the probability of observing the data given the hypothesis multiplied by the probability of the hypothesis being true regardless of the data, divided by the probability of observing the data regardless of the hypothesis.

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself.

Under this framework, each piece of the calculation has a specific name; for example:

P(h|D): Posterior probability of the hypothesis (the thing we want to calculate).

P(h): Prior probability of the hypothesis.

This gives a useful framework for thinking about and modeling a machine learning problem.

If we have some prior domain knowledge about the hypothesis, this is captured in the prior probability. If we don't, then all hypotheses may have the same prior probability.

If the probability of observing the data P(D) increases, then the probability of the hypothesis holding given the data P(h|D) decreases. Conversely, if the probability of the hypothesis P(h) and the probability of observing the data given hypothesis increases, the probability of the hypothesis holding given the data P(h|D) increases.

The notion of testing different models on a dataset in applied machine learning can be thought of as estimating the probability of each hypothesis (h1, h2, h3, … in H) being true given the observed data.

The optimization or seeking the hypothesis with the maximum posterior probability in modeling is called maximum a posteriori or MAP for short.
Any such maximally probable hypothesis is called a maximum a posteriori (MAP) hypothesis. We can determine the MAP hypotheses by using Bayes theorem to calculate the posterior probability of each candidate hypothesis.

Under this framework, the probability of the data (D) is constant as it is used in the assessment of each hypothesis. Therefore, it can be removed from the calculation to give the simplified unnormalized estimate as follows:

max h in H P(h|D) = P(D|h) * P(h)

If we do not have any prior information about the hypothesis being tested, they can be assigned a uniform probability, and this term too will be a constant and can be removed from the calculation to give the following:

max h in H P(h|D) = P(D|h)

That is, the goal is to locate a hypothesis that best explains the observed data.

Fitting models like linear regression for predicting a numerical value, and logistic regression for binary classification can be framed and solved under the MAP probabilistic framework. This provides an alternative to the more common maximum likelihood estimation (MLE) framework.

Bayes Theorem for Classification

Classification is a predictive modeling problem that involves assigning a label to a given input data sample.

The problem of classification predictive modeling can be framed as calculating the conditional probability of a class label given a data sample, for example:

P(class|data) = (P(data|class) * P(class)) / P(data)

Where P(class|data) is the probability of class given the provided data.

This calculation can be performed for each class in the problem and the class that is assigned the largest probability can be selected and assigned to the input data.

In practice, it is very challenging to calculate full Bayes Theorem for classification.

The priors for the class and the data are easy to estimate from a training dataset, if the dataset is suitability representative of the broader problem.

The conditional probability of the observation based on the class P(data|class) is not feasible unless the number of examples is extraordinarily large, e.g. large enough to effectively estimate the probability distribution for all different possible combinations of values. This is almost never the case, we will not have sufficient coverage of the domain.

As such, the direct application of Bayes Theorem also becomes intractable, especially as the number of variables or features (n) increases.

Naive Bayes Classifier

The Bayes Theorem assumes that each input variable is dependent upon all other variables. This is a cause of complexity in the calculation. We can remove this assumption and consider each input variable as being independent from each other.

This changes the model from a dependent conditional probability model to an independent conditional probability model and dramatically simplifies the calculation.

This means that we calculate P(data|class) for each input variable separately and multiple the results together, for example:

P(class | X1, X2, …, Xn) = P(X1|class) * P(X2|class) * … * P(Xn|class) * P(class) / P(data)

We can also drop the probability of observing the data as it is a constant for all calculations, for example:

- P(class | X1, X2, …, Xn) = P(X1|class) * P(X2|class) * … * P(Xn|class) * P(class)

This simplification of Bayes Theorem is common and widely used for classification predictive modeling problems and is generally referred to as Naive Bayes.

Bayes Classifier: Probabilistic model that makes the most probable prediction for new examples.

What is the most probable classification of the new instance given the training data?

This is different from the MAP framework that seeks the most probable hypothesis (model). Instead, we are interested in making a specific prediction.

The equation below demonstrates how to calculate the conditional probability for a new instance (v_i) given the training data (*D*), given a space of hypotheses (*H*).

$P(v_j \mid D)$ = sum {h in H} $P(v_j \mid hi) * P(h_i \mid D)$

Where v_j is a new instance to be classified, *H* is the set of hypotheses for classifying the instance, h_i is a given hypothesis, $P(v_j \mid h_i)$ is the posterior probability for v_i given hypothesis h_i, and $P(h_i \mid D)$ is the posterior probability of the hypothesis h_i given the data *D*.

Linear Regression:

Linear Regression Model Representation

Linear regression is an attractive model because the representation is so simple.

The representation is a linear equation that combines a specific set of input values (x) the solution to which is the predicted output for that set of input values (y). As such, both the input values (x) and the output value are numeric.

The linear equation assigns one scale factor to each input value or column, called a coefficient and represented by the capital Greek letter Beta (B). One additional coefficient is also added, giving the line an additional degree of freedom (e.g. moving up and down on a two-dimensional plot) and is often called the intercept or the bias coefficient.

For example, in a simple regression problem (a single x and a single y), the form of the model would be:

$$y = B0 + B1*x$$

In higher dimensions when we have more than one input (x), the line is called a plane or a hyper-plane. The representation therefore is the form of the equation and the specific values used for the coefficients (e.g. B0 and B1 in the above example).

It is common to talk about the complexity of a regression model like linear regression. This refers to the number of coefficients used in the model.

When a coefficient becomes zero, it effectively removes the influence of the input variable on the model and therefore from the prediction made from the model (0 * x = 0). This becomes relevant if you look at regularization methods that change the learning algorithm to reduce the complexity of regression models by putting pressure on the absolute size of the coefficients, driving some to zero.

Now that we understand the representation used for a linear regression model, let's review some ways that we can learn this representation from data.

Linear Regression Learning the Model

Learning a linear regression model means estimating the values of the coefficients used in the representation with the data.

In this section we will take a brief look at four techniques to prepare a linear regression model.

1. Simple Linear Regression

With simple linear regression when we have a single input, we can use statistics to estimate the coefficients.

This requires that you calculate statistical properties from the data such as means, standard deviations, correlations and covariance. All of the data must be available to traverse and calculate statistics.

2. Ordinary Least Squares

When we have more than one input we can use Ordinary Least Squares to estimate the values of the coefficients.

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all of the squared errors together. This is the quantity that ordinary least squares seeks to minimize. This approach treats the data as a matrix and uses linear algebra operations to estimate the optimal values for the coefficients. It means that all of the data must be available and you must have enough memory to fit the data and perform matrix operations.

It is unusual to implement the Ordinary Least Squares procedure yourself unless as an exercise in linear algebra. It is more likely that you will call a procedure in a linear algebra library. This procedure is very fast to calculate.

3. Gradient Descent

When there are one or more inputs you can use a process of optimizing the values of the coefficients by iteratively minimizing the error of the model on your training data.

This operation is called Gradient Descent and works by starting with random values for each coefficient. The sum of the squared errors are calculated for each pair of input and output values. A learning rate is used as a scale factor and the coefficients are updated in the direction towards minimizing the error. The process is repeated until a minimum sum squared error is achieved or no further improvement is possible.

When using this method, you must select a learning rate (alpha) parameter that determines the size of the improvement step to take on each iteration of the procedure.

Gradient descent is often taught using a linear regression model because it is relatively straightforward to understand. In practice, it is useful when you have a very large dataset either in the number of rows or the number of columns that may not fit into memory.

4. Regularization

There are extensions of the training of the linear model called regularization methods. These seek to both minimize the sum of the squared error of the model on the training data (using ordinary least squares) but also to reduce the complexity of the model (like the number or absolute size of the sum of all coefficients in the model).

Two popular examples of regularization procedures for linear regression are:

Lasso Regression: where Ordinary Least Squares is modified to also minimize the absolute sum of the coefficients (called L1 regularization).

Ridge Regression: where Ordinary Least Squares is modified to also minimize the squared absolute sum of the coefficients (called L2 regularization).

These methods are effective to use when there is collinearity in your input values and ordinary least squares would overfit the training data.

Now that you know some techniques to learn the coefficients in a linear regression model, let’s look at how we can use a model to make predictions on new data.

Making Predictions with Linear Regression

Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let’s make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

$$y = B0 + B1 * x1$$

or

$$weight = B0 + B1 * height$$

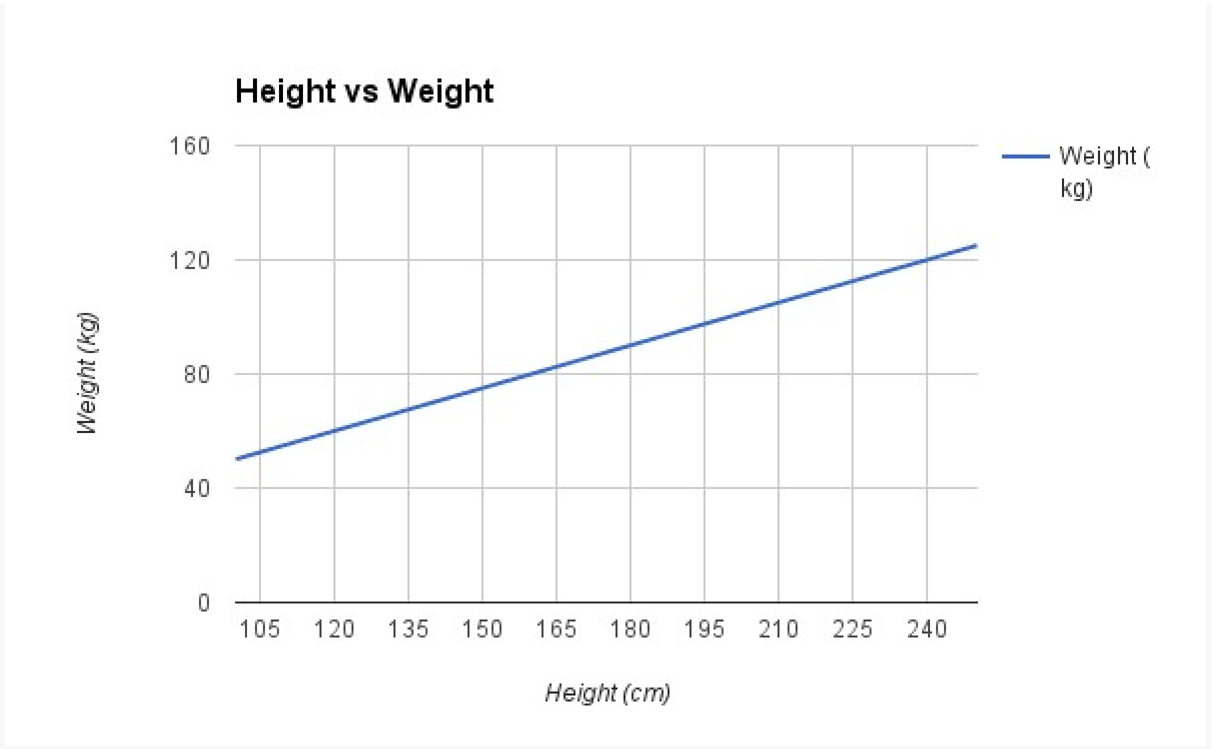
Where B0 is the bias coefficient and B1 is the coefficient for the height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

For example, lets use B0 = 0.1 and B1 = 0.5. Let’s plug them in and calculate the weight (in kilograms) for a person with the height of 182 centimeters.

$$weight = 0.1 + 0.5 * 182$$

$$weight = 91.1$$

You can see that the above equation could be plotted as a line in two-dimensions. The B0 is our starting point regardless of what height we have. We can run through a bunch of heights from 100 to 250 centimeters and plug them to the equation and get weight values, creating our line.



Sample Height vs Weight Linear Regression

Now that we know how to make predictions given a learned linear regression model, let’s look at some rules of thumb for preparing our data to make the most of this type of model.

- **Linear Assumption**. Linear regression assumes that the relationship between your input and output is linear. It does not support anything else. This may be obvious, but it is good to remember when you have a lot of attributes. You may need to transform data to make the relationship linear.

- **Remove Noise**. Linear regression assumes that your input and output variables are not noisy. Consider using data cleaning operations that let you better expose and clarify the signal in your data. This is most important for the output variable and you want to remove outliers in the output variable (y) if possible.

- **Remove Collinearity**. Linear regression will over-fit your data when you have highly correlated input variables. Consider calculating pairwise correlations for your input data and removing the most correlated.

- **Gaussian Distributions**. Linear regression will make more reliable predictions if your input and output variables have a Gaussian distribution. You may get some benefit using transforms (e.g. log or BoxCox) on you variables to make their distribution more Gaussian looking.
- **Rescale Inputs**: Linear regression will often make more reliable predictions if you rescale input variables using standardization or normalization.

Assumptions in Regression

Regression is a parametric approach. 'Parametric' means it makes assumptions about data for the purpose of analysis. Due to its parametric side, regression is restrictive in nature. It fails to deliver good results with data sets which doesn't fulfill its assumptions. Therefore, for a successful regression analysis, it's essential to validate these assumptions.

Let's look at the important assumptions in regression analysis:

1. There should be a linear and additive relationship between dependent (response) variable and independent (predictor) variable(s). A linear relationship suggests that a change in response Y due to one unit change in X^1 is constant, regardless of the value of X^1. An additive relationship suggests that the effect of X^1 on Y is independent of other variables.
2. There should be no correlation between the residual (error) terms. Absence of this phenomenon is known as Autocorrelation.
3. The independent variables should not be correlated. Absence of this phenomenon is known as multicollinearity.

4. The error terms must have constant variance. This phenomenon is known as homoskedasticity. The presence of non-constant variance is referred to heteroskedasticity.
5. The error terms must be normally distributed.

Regularization

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

A simple relation for linear regression looks like this. Here Y represents the learned relation and *β represents the coefficient estimates for different variables or predictors(X).*

Y ≈ β0 + β1X1 + β2X2 + …+ βpXp

The fitting procedure involves a loss function, known as residual sum of squares or RSS. The coefficients are chosen, such that they minimize this loss function.

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

Now, this will adjust the coefficients based on your training data. *If there is noise in the training data, then the estimated coefficients won’t generalize well to the future data. This is where regularization comes in and shrinks or regularizes these learned estimates towards zero.*

Ridge Regression

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

Above image shows ridge regression, where the ***RSS is modified by adding the shrinkage quantity.*** Now, the coefficients are estimated by minimizing this function. Here, ***λ is the tuning parameter that decides how much we want to penalize the flexibility of our model.*** The increase in flexibility of a model is represented by increase in its coefficients, and if we want to minimize the above function, then these coefficients need to be small. This is how the Ridge regression technique prevents coefficients from rising too high. Also, notice that we shrink the estimated association of each variable with the response, except the intercept β0, This intercept is a measure of the mean value of the response when xi1 = xi2 = …= xip = 0.

When λ = 0, the penalty term has no effect, and the estimates produced by ridge regression will be equal to least squares. However, ***as λ→∞, the impact of the shrinkage penalty grows, and the ridge regression coefficient estimates will approach zero***. As can be seen, selecting a good value of λ is critical. Cross validation comes in handy for this purpose. The coefficient estimates produced by this method are ***also known as the L2 norm***.

The coefficients that are produced by the standard least squares method are scale equivariant, i.e. if we multiply each input by c then the corresponding coefficients are scaled by a factor of 1/c. Therefore, regardless of how the predictor is scaled, the multiplication of predictor and coefficient(Xjβj) remains the same. ***However, this is not the case with ridge regression, and therefore, we need to standardize the predictors or bring the predictors to the same scale before performing ridge regression***. The formula used to do this is given below.

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}},$$

Lasso

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

Lasso is another variation, in which the above function is minimized. Its clear that ***this variation differs from ridge regression only in penalizing the high coefficients***. It uses |βj|(modulus)instead of squares of β, as its penalty. In statistics, this is ***known as the L1 norm***.

A standard least squares model tends to have some variance in it, i.e. this model won't generalize well for a data set different than its training data. ***Regularization, significantly reduces the variance of the model, without substantial increase in its bias***. So the tuning parameter λ, used in the regularization techniques described above, controls the impact on bias and variance. As the value of λ rises, it reduces the value of coefficients and thus reducing the variance. ***Till a point, this increase in λ is beneficial as it is only reducing the variance(hence avoiding overfitting), without loosing any important properties in the data.*** But after certain value, the model starts loosing important properties, giving rise to bias in the model and thus underfitting. Therefore, the value of λ should be carefully selected.

Elastic Net Regression

Linear regression refers to a model that assumes a linear relationship between input variables and the target variable.

With a single input variable, this relationship is a line, and with higher dimensions, this relationship can be thought of as a hyperplane that connects the input variables to the target variable. The coefficients of the model are found via an optimization process that seeks to minimize the sum squared error between the predictions (*yhat*) and the expected target values (*y*).
loss = sum i=0 to n (y_i – yhat_i)^2

A problem with linear regression is that estimated coefficients of the model can become large, making the model sensitive to inputs and possibly unstable. This is particularly true for problems with few observations (*samples*) or more samples (*n*) than input predictors (*p*) or variables (so-called *p >> n problems*).

One approach to addressing the stability of regression models is to change the loss function to include additional costs for a model that has large coefficients. Linear regression models that use these modified loss functions during training are referred to collectively as penalized linear regression.

One popular penalty is to penalize a model based on the sum of the squared coefficient values. This is called an L2 penalty. An L2 penalty minimizes the size of all coefficients, although it prevents any coefficients from being removed from the model.

l2_penalty = sum j=0 to p beta_j^2

Another popular penalty is to penalize a model based on the sum of the absolute coefficient values. This is called the L1 penalty. An L1 penalty minimizes the size of all coefficients and allows some coefficients to be minimized to the value zero, which removes the predictor from the model.

l1_penalty = sum j=0 to p abs(beta_j)

Elastic net is a penalized linear regression model that includes both the L1 and L2 penalties during training.

A hyperparameter "*alpha*" is provided to assign how much weight is given to each of the L1 and L2 penalties. Alpha is a value between 0 and 1 and is used to weight the contribution of the L1 penalty and one minus the alpha value is used to weight the L2 penalty.
elastic_net_penalty = (alpha * l1_penalty) + ((1 – alpha) * l2_penalty)

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.

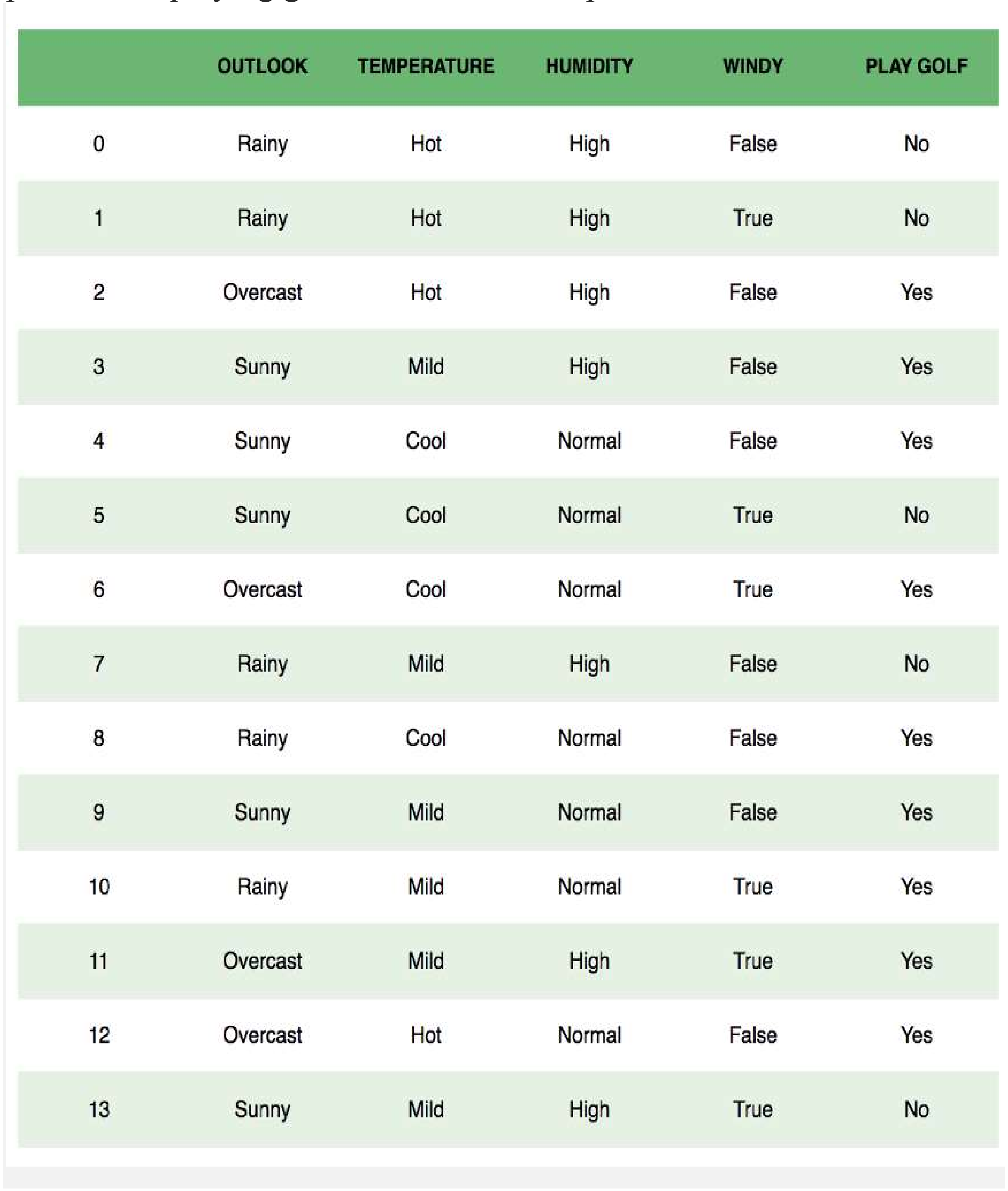
Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Using Bayes theorem, we can find the probability of **A** happening, given that **B** has occurred. Here, **B** is the evidence and **A** is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Example:

Let us take an example to get some better intuition. Consider the problem of playing golf. The dataset is represented as below.



	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.

X is given as,

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

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.

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

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.

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

Using the above function, we can obtain the class, given the predictors.

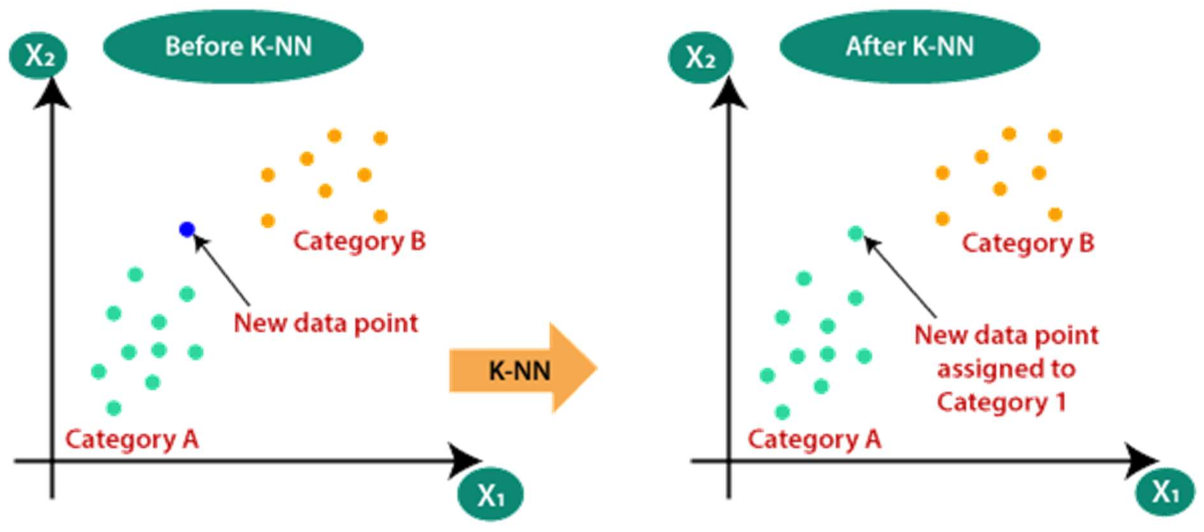
K-Nearest Neighbor(KNN) Algorithm for Machine Learning

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

KNN Classifier
Input value
Predicted Output

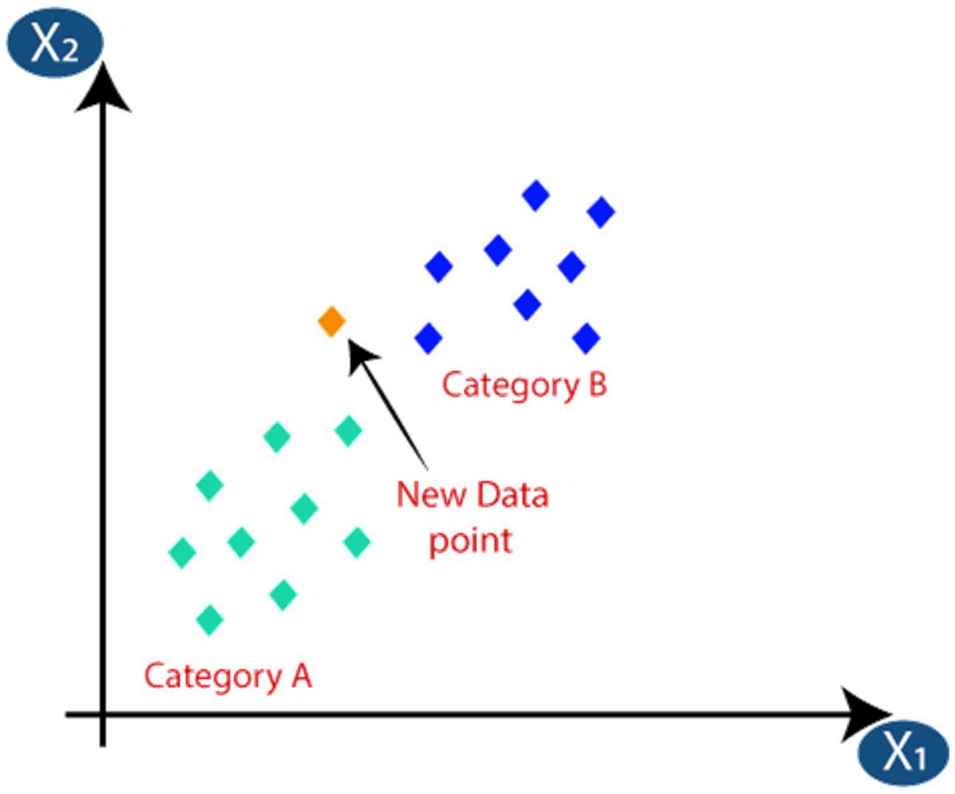
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



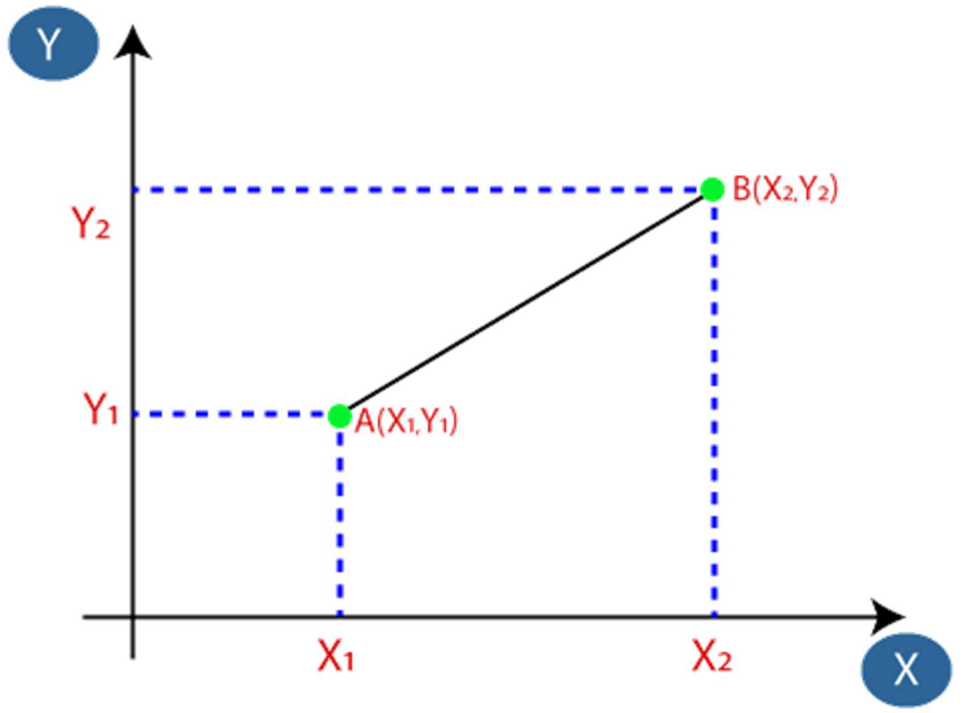
The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:

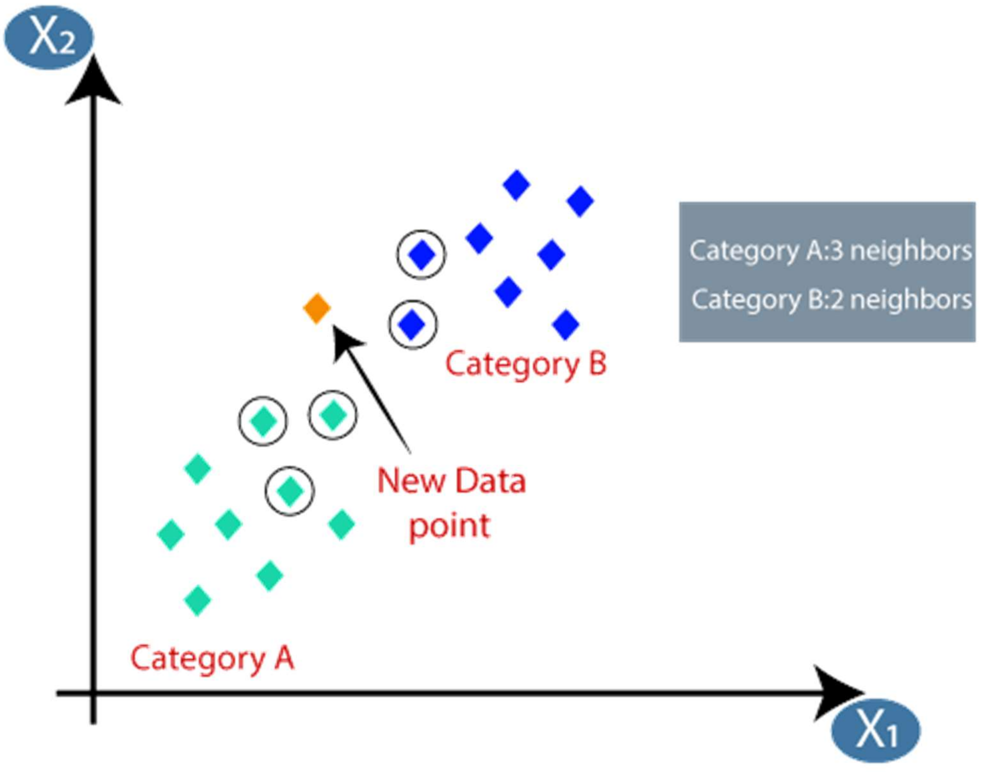


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and $B_2 = \sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

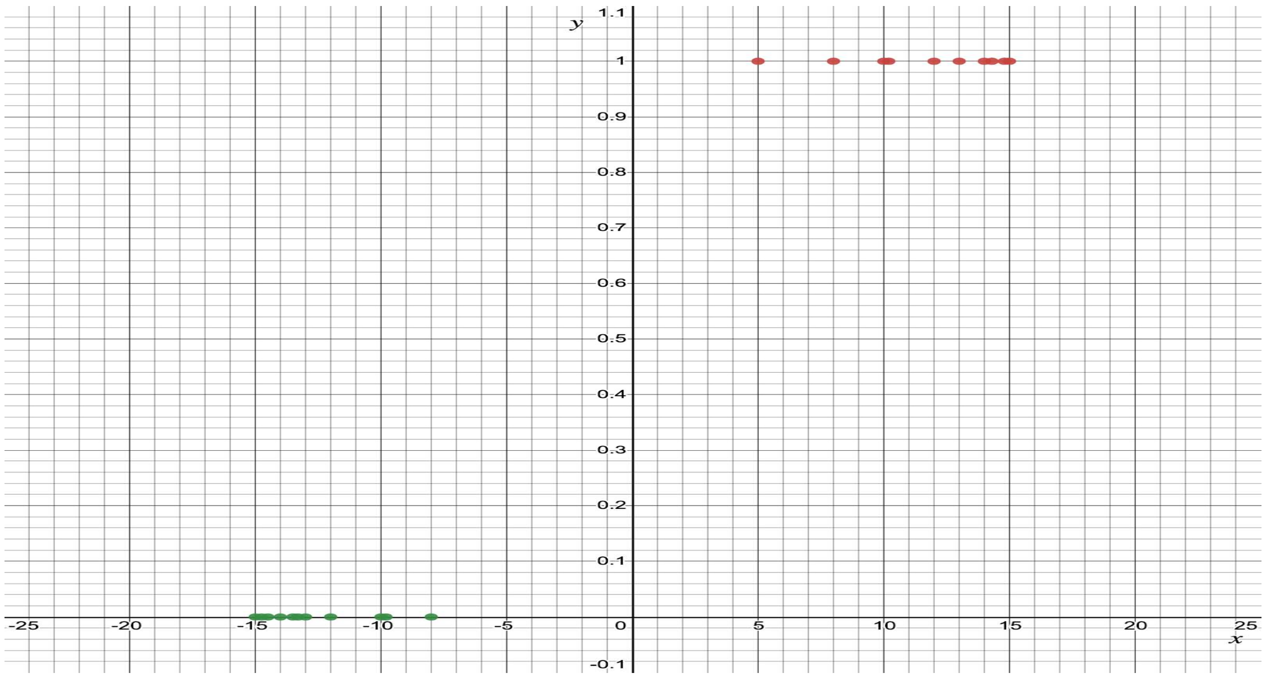
Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

Logistic Regression

Logistic regression is a classification algorithm, used when the value of the target variable is *categorical* in nature. Logistic regression is most commonly used when the data in question has binary output, so when it belongs to one class or another, or is either a 0 or 1. Remember that classification tasks have discrete categories, unlike regressions tasks.

Here, by the idea of using a regression model to solve the classification problem, we rationally raise a question of whether we can draw a hypothesis function to fit to the binary dataset. For simplification, we only concern the binary classification problem with the dataset below:



The answer is that you will have to use a type of function, different from linear functions, called a logistic function, or a **sigmoid function.**

Sigmoid Function

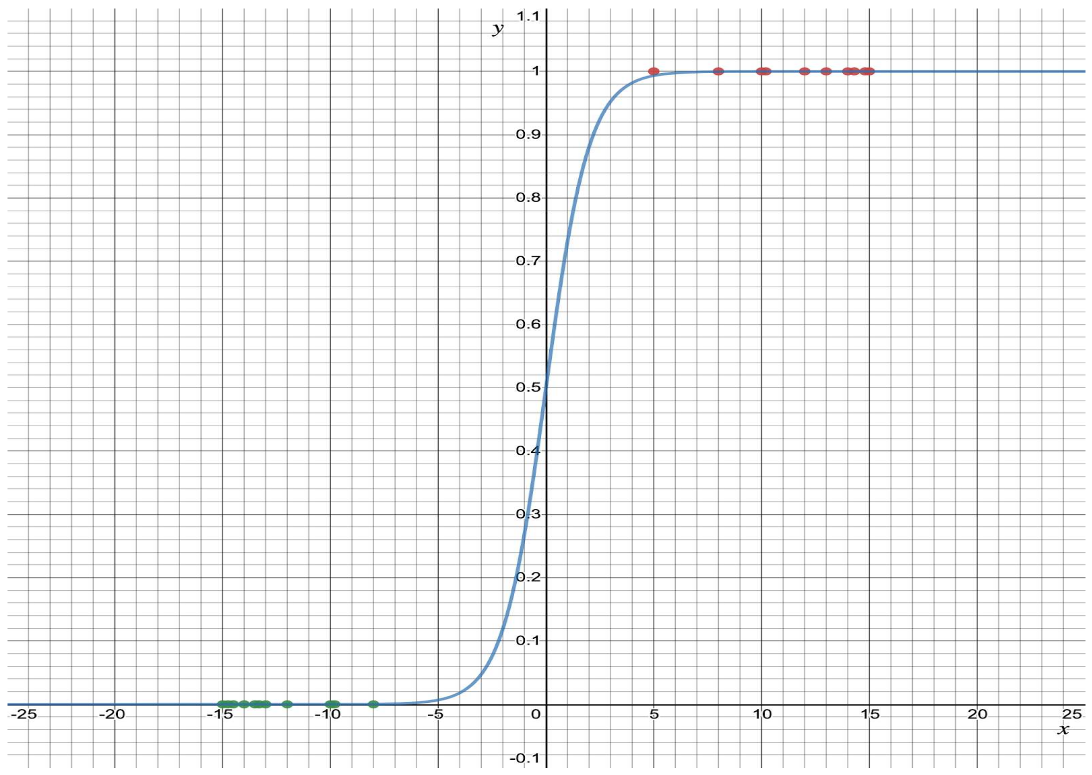
The sigmoid function/logistic function is a function that resembles an "S" shaped curve when plotted on a graph. It takes values between 0 and 1 and "squishes" them towards the margins at the top and bottom, labeling them as 0 or 1.

The equation for the Sigmoid function is this:

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

.

Let's see how the sigmoid function represent the given dataset.



This gives a value y that is extremely close to 0 if x is a large negative value and close to 1 if x is a large positive value. After the input value has been squeezed towards 0 or 1, the input can be run through a typical linear function, but the inputs can now be put into distinct categories.

It is important to understand that logistic regression should only be used when the target variables fall into discrete categories and that if there's a range of continuous values the target value might be, logistic regression should not be used. Examples of situations you might use logistic regression in include:

- Predicting if an email is spam or not spam
- Whether a tumor is malignant or benign
- Whether a mushroom is poisonous or edible.

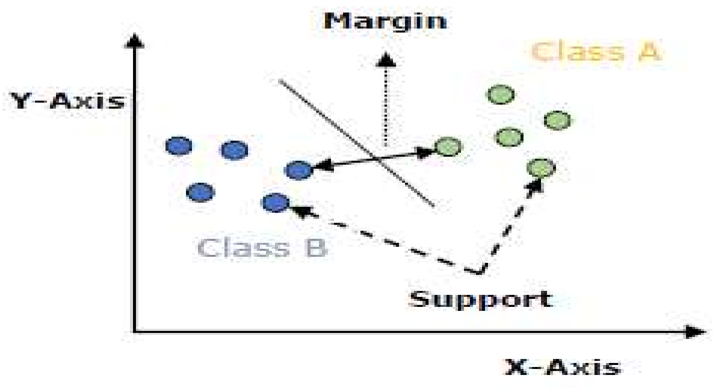
When using logistic regression, a threshold is usually specified that indicates at what value the example will be put into one class vs. the other class. In the spam classification task, a threshold of 0.5 might be set, which would cause an email with a 50% or greater probability of being spam to be classified as “spam” and any email with probability less than 50% classified as “not spam”.

Introduction to SVM

Support vector machines (SVMs) are powerful yet flexible supervised machine learning algorithms which are used both for classification and regression. But generally, they are used in classification problems. In 1960s, SVMs were first introduced but later they got refined in 1990. SVMs have their unique way of implementation as compared to other machine learning algorithms. Lately, they are extremely popular because of their ability to handle multiple continuous and categorical variables.

Working of SVM

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).



The followings are important concepts in SVM −

- **Support Vectors** − Datapoints that are closest to the hyperplane is called support vectors. Separating line will be defined with the help of these data points.
- **Hyperplane** − As we can see in the above diagram, it is a decision plane or space which is divided between a set of objects having different classes.

- **Margin** − It may be defined as the gap between two lines on the closet data points of different classes. It can be calculated as the perpendicular distance from the line to the support vectors. Large margin is considered as a good margin and small margin is considered as a bad margin.

The main goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH) and it can be done in the following two steps −

- First, SVM will generate hyperplanes iteratively that segregates the classes in best way.
- Then, it will choose the hyperplane that separates the classes correctly.

SVM Kernels

In practice, SVM algorithm is implemented with kernel that transforms an input data space into the required form. SVM uses a technique called the kernel trick in which kernel takes a low dimensional input space and transforms it into a higher dimensional space. In simple words, kernel converts non-separable problems into separable problems by adding more dimensions to it. It makes SVM more powerful, flexible and accurate. The following are some of the types of kernels used by SVM.

Linear Kernel

It can be used as a dot product between any two observations. The formula of linear kernel is as below −

$$K(x,xi)=sum(x*xi)K(x,xi)=sum(x*xi)$$

From the above formula, we can see that the product between two vectors say x & xi is the sum of the multiplication of each pair of input values.

Polynomial Kernel

It is more generalized form of linear kernel and distinguish curved or nonlinear input space. Following is the formula for polynomial kernel −

$$k(X,Xi)=1+sum(X*Xi)\text{^}dk(X,Xi)=1+sum(X*Xi)\text{^}d$$

Here d is the degree of polynomial, which we need to specify manually in the learning algorithm.

Pros and Cons of SVM Classifiers

Pros of SVM classifiers

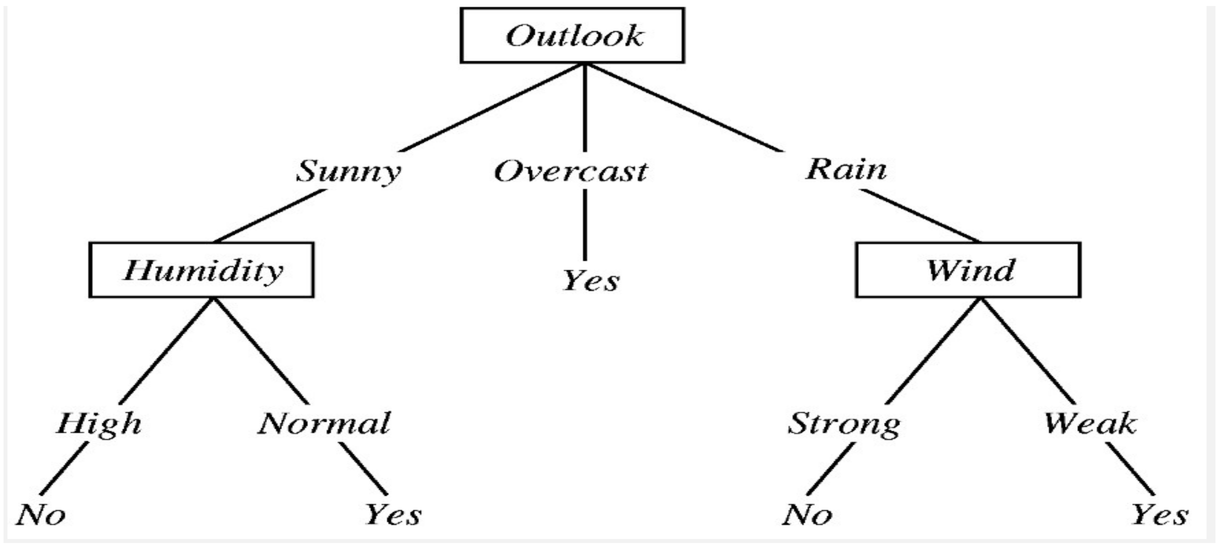
SVM classifiers offers great accuracy and work well with high dimensional space. SVM classifiers basically use a subset of training points hence in result uses very less memory.

Cons of SVM classifiers

They have high training time hence in practice not suitable for large datasets. Another disadvantage is that SVM classifiers do not work well with overlapping classes.

Decision Tree:

A decision tree is a flowchart-like structure in which each internal node represents a test on a feature (e.g. whether a coin flip comes up heads or tails) , each leaf node represents a class label (decision taken after computing all features) and branches represent conjunctions of features that lead to those class labels. The paths from root to leaf represent classification rules. Below diagram illustrate the basic flow of decision tree for decision making with labels (Rain(Yes), No Rain(No)).



Decision Tree for Rain Forecasting

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning.

Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric **supervised learning** method used for both **classification** and **regression** tasks.

Tree models where the target variable can take a discrete set of values are called **classification trees**. Decision trees where the target variable can take continuous values (typically real numbers) are called **regression trees**. Classification And Regression Tree (CART) is general term for this.

Throughout this post i will try to explain using the examples.

Data Format

Data comes in records of forms.

```
(x,Y)=(x1,x2,x3,....,xk,Y)
```

The dependent variable, Y, is the target variable that we are trying to understand, classify or generalize. The vector x is composed of the features, x1, x2, x3 etc., that are used for that task.

Approach to make decision tree

While making decision tree, at each node of tree we ask different type of questions. Based on the asked question we will calculate the information gain corresponding to it.

Information Gain

Information gain is used to decide which feature to split on at each step in building the tree. Simplicity is best, so we want to keep our tree small. To do so, at each step we should choose the split that results in the purest daughter nodes. A commonly used measure of purity is called information. For each node of the tree, the information value **measures how much information a feature gives us about the class. The split with the highest information gain will be taken as the first split and the process will continue until all children nodes are pure, or until the information gain is 0.**

Algorithm for constructing decision tree usually works top-down, by choosing a variable at each step that best splits the set of items. Different algorithms use different metrices for measuring best.

Gini Impurity

First let’s understand the meaning of **Pure** and **Impure**.

Pure

Pure means, in a selected sample of dataset all data belongs to same class (PURE).

Impure

Impure means, data is mixture of different classes.

Definition of Gini Impurity

Gini Impurity is a measurement of the likelihood of an incorrect classification of a new instance of a random variable, if that new instance were randomly classified according to the distribution of class labels from the data set.

If our dataset is Pure then likelihood of incorrect classification is 0. If our sample is mixture of different classes then likelihood of incorrect classification will be high.

Steps for Making decision tree

- Get list of rows (dataset) which are taken into consideration for making decision tree (recursively at each nodes).
- Calculate uncertanity of our dataset or Gini impurity or how much our data is mixed up etc.
- Generate list of all question which needs to be asked at that node.

- Partition rows into True rows and False rows based on each question asked.
- Calculate information gain based on gini impurity and partition of data from previous step.
- Update highest information gain based on each question asked.
- Update best question based on information gain (higher information gain).
- Divide the node on best question. Repeat again from step 1 again until we get pure node (leaf nodes).

Advantage of Decision Tree

- Easy to use and understand.
- Can handle both categorical and numerical data.
- Resistant to outliers, hence require little data preprocessing.

Disadvantage of Decision Tree

- Prone to overfitting.
- Require some kind of measurement as to how well they are doing.
- Need to be careful with parameter tuning.
- Can create biased learned trees if some classes dominate.

Overfitting the Decision tree model

Overfitting is one of the major problem for every model in machine learning. If model is overfitted it will poorly generalized to new samples. To avoid decision tree from overfitting **we remove the branches that make use of features having low importance.** This method is called as **Pruning or post-pruning.** This way we will reduce the complexity of tree, and hence imroves predictive accuracy by the reduction of overfitting.

Pruning should reduce the size of a learning tree without reducing predictive accuracy as measured by a cross-validation set. There are 2 major Pruning techniques.

- *Minimum Error:* The tree is pruned back to the point where the cross-validated error is a minimum.
- *Smallest Tree:* The tree is pruned back slightly further than the minimum error. Technically the pruning creates a decision tree with cross-validation error within 1 standard error of the minimum error.

Early Stop or Pre-pruning

An alternative method to prevent overfitting is to try and stop the tree-building process early, before it produces leaves with very small samples. This heuristic is known as *early stopping* but is also sometimes known as pre-pruning decision trees.

At each stage of splitting the tree, we check the cross-validation error. If the error does not decrease significantly enough then we stop. Early stopping may underfit by stopping too early. The current split may be of little benefit, but having made it, subsequent splits more significantly reduce the error.

Early stopping and pruning can be used together, separately, or not at all. Post pruning decision trees is more mathematically rigorous, finding a tree at least as good as early stopping. Early stopping is a quick fix heuristic. If used together with pruning, early stopping may save time. After all, why build a tree only to prune it back again?

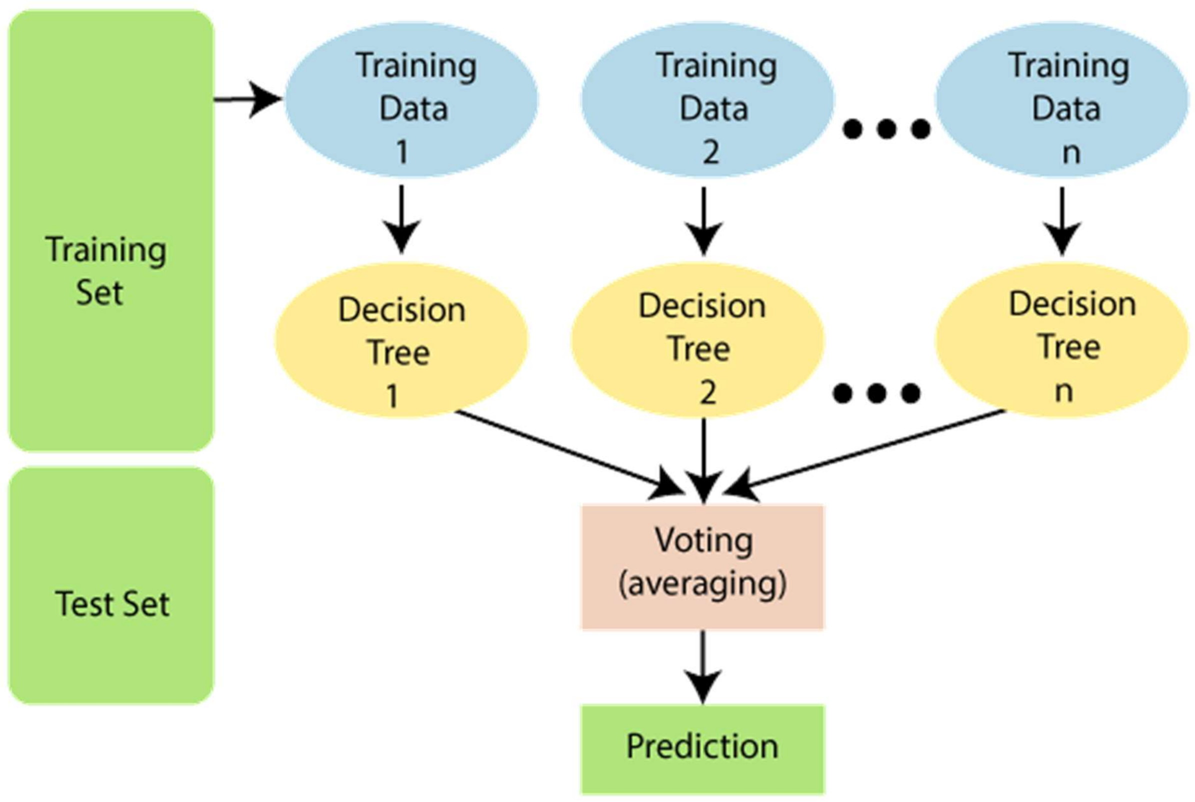
Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset".*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:



Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

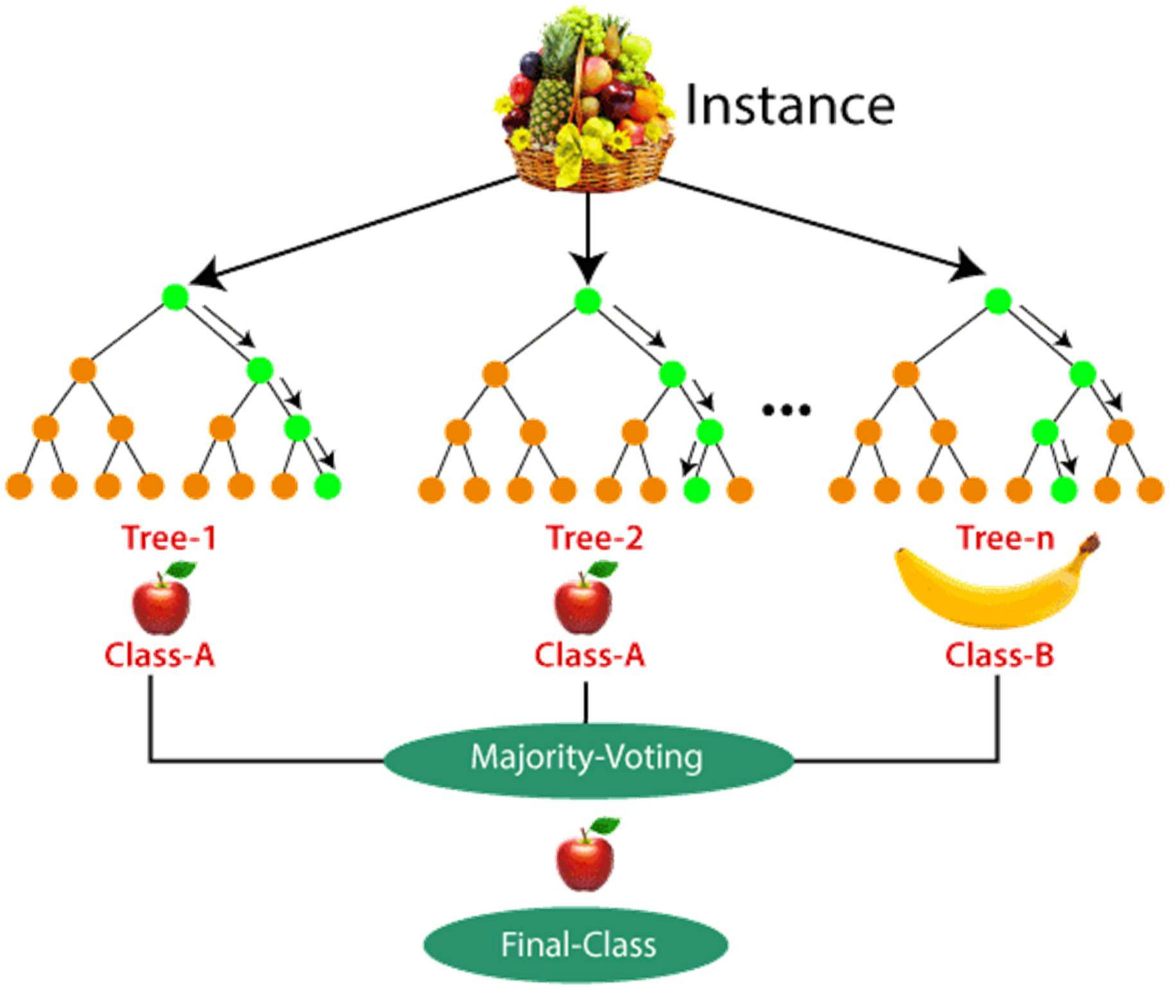
Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.

The working of the algorithm can be better understood by the below example:

Example: Suppose there is a dataset that contains multiple fruit images. So, this dataset is given to the Random forest classifier. The dataset is divided into subsets and given to each decision tree. During the training phase, each decision tree produces a prediction result, and when a new data point occurs, then based on the majority of results, the Random Forest classifier predicts the final decision. Consider the below image:



Advantages of Random Forest

- Random Forest is capable of performing both Classification and Regression tasks.
- It is capable of handling large datasets with high dimensionality.
- It enhances the accuracy of the model and prevents the overfitting issue.

Disadvantages of Random Forest

- Although random forest can be used for both classification and regression tasks, it is not more suitable for Regression tasks.

Model Performance Evaluation Metrics

Regression

There are many metrics for determining model performance for regression problems, but the most commonly used metric is known as the *mean square error* (*MSE*), or variation called the *root mean square error* (*RMSE*), which is calculated by taking the square root of the mean squared error.

The error in this case is the difference in value between a given model prediction and its actual value for an out-of-sample observation. The mean squared error is therefore the average of all of the squared errors across all new observations, which is the same as adding all of the squared errors (*sum of squares*) and dividing by the number of observations.

In addition to being used as a stand-alone performance metric, mean squared error (or RMSE) can also be used for model selection, controlling model complexity, and model tuning. Often many models are created and evaluated (e.g., cross-validation), and then MSE (or similar metric) is plotted on the y-axis, with the tuning or validation parameter given on the x-axis.

The tuning or validation parameter is changed in each model creation and evaluation step, and the plot described above can help determine the ideal tuning parameter value. The number of predictors is a great example of a potential tuning parameter in this case.

Before moving on to classification, it is worth mentioning R2 briefly. R2 is often thought of as a measure of model performance, but it's actually not. R2 is a measure of the amount of variance explained by the model, and is given as a number between 0 and 1. A value of 1 means the model explains all of the data perfectly, but when computed with training data is more of an indication of potential overfitting than high predictive performance.

As discussed earlier, the more complex the model, the more the model *tends* to fit the data better and potentially overfit, or contribute to additional model *variance*. Given this, adjusted R2 is a more robust and reliable metric in that it adjusts for any increases in model complexity (e.g., adding more predictors), so that one can better gauge underlying model improvement in lieu of the increased complexity.

Classification

Recall the different results from a binary classifier, which are true positives, true negatives, false positives, and false negatives. These are often shown in a *confusion matrix*. A confusion matrix is conceptually the basis of many classification performance metrics as shown. We will discuss a few of the more popular ones associated with machine learning here.

Accuracy is a key measure of performance, and is more specifically the rate at which the model is able to predict the correct value (classification or regression) for a given data point or observation. In other words, accuracy is the proportion of correct predictions out of all predictions made.

The other two metrics from the confusion matrix worth discussing are *precision* and *recall*. *Precision* (*positive predictive value*) is the ratio of true positives to the total amount of positive predictions made (i.e., true or false). Said another way, precision measures the proportion of accurate positive predictions out of all positive predictions made.

Recall on the other hand, or *true positive rate*, is the ratio of true positives to the total amount of actual positives, whether predicted correctly or not. So in other words, recall measures the proportion of accurate positive predictions out of all actual positive observations.

A metric that is associated with precision and recall is called the *F-score* (also called F1 score), which combines them mathematically, and somewhat like a weighted average, in order to produce a single measure of performance based on the simultaneous values of both. Its values range from 0 (worst) to 1 (best).

Another important concept to know about is the *receiver operating characteristic*, which when plotted, results in what's known as an *ROC curve*.

An ROC curve is a two-dimensional plot of *sensitivity* (recall, or true positive rate) vs *specificity* (false positive rate). The area under the curve is referred to as the *AUC*, and is a numeric metric used to represent the quality and performance of the classifier (model). An AUC of 0.5 is essentially the same as random guessing without a model, whereas an AUC of 1.0 is considered a perfect classifier. Generally, the higher the AUC value the better, and an AUC above 0.8 is considered quite good.

The higher the AUC value, the closer the curve gets to the upper left corner of the plot. One can easily see from the ROC curves then that the goal is to find and tune a model that maximises the true positive rate, while simultaneously minimising the false positive rate. Said another way, the goal as shown by the ROC curve is to correctly predict as many of the actual positives as possible, while also predicting as many of the actual negatives as possible, and therefore minimise errors for both.

Error Analysis and Tradeoffs

There are multiple types of *errors* associated with machine learning and predictive analytics. The primary types are *in-sample* and *out-of-sample* errors. In-sample errors are the error rate found from the training data, i.e., the data used to build predictive models.

Out-of-sample errors are the error rates found on a new data set, and are the most important since they represent the potential performance of a given predictive model on new and unseen data.

In-sample error rates may be very low and seem to be indicative of a high-performing model, but one must be careful, as this may be due to overfitting as mentioned, which would result in a model that is unable to generalise well to new data.

Training and validation data is used to build, validate, and tune a model, but test data is used to evaluate model performance and generalisation capability. One very important point to note is that prediction performance and error analysis should only be done on test data, when evaluating a model for use on non-training or new data.

Generally speaking, model performance on training data tends to be optimistic, and therefore data errors will be less than those involving test data. There are tradeoffs between the types of errors that a machine learning practitioner must consider and often choose to accept.

For binary classification problems, there are two primary types of errors. *Type 1* errors (false positives) and *Type 2* errors (false negatives). It's often possible through model selection and tuning to increase one while decreasing the other, and often one must choose which error type is more acceptable. This can be a major tradeoff consideration depending on the situation.

A typical example of this tradeoff dilemma involves cancer diagnosis, where the positive diagnosis of having cancer is based on some test. In this case, a false positive means that someone is told that have have cancer when they do not. Conversely, the false negative case is when someone is told that they do not have cancer when they actually do.

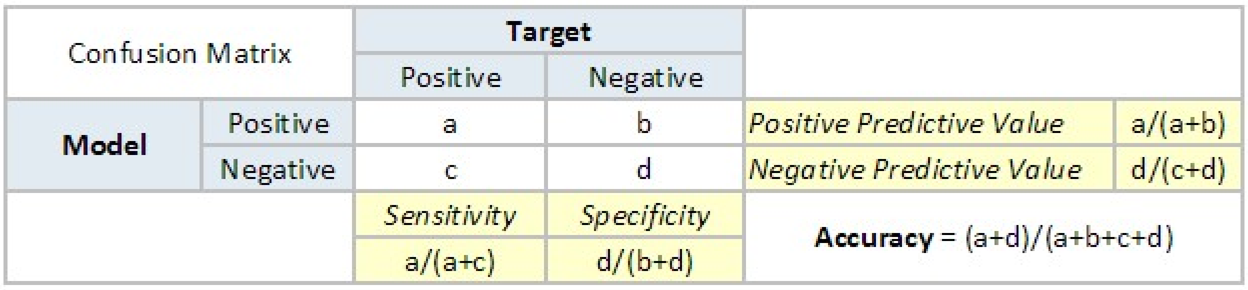
Telling someone they have cancer when they don't can result in tremendous emotional distress, stress, additional tests and medical costs, and so on. On the other hand, failing to detect cancer in someone that actually has it can mean the difference between life and death.

In the spam or ham case, neither error type is nearly as serious as the cancer case, but typically email vendors err slightly more on the side of letting some spam get into your inbox as opposed to you missing a very important email because the spam classifier is too aggressive.

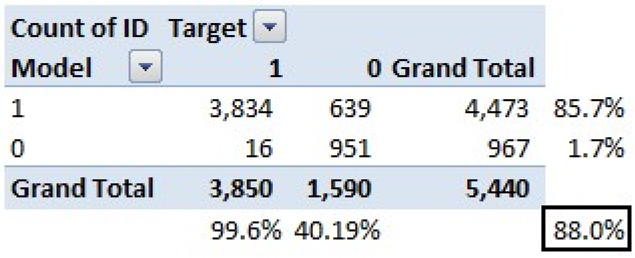
1. Confusion Matrix

A confusion matrix is an N X N matrix, where N is the number of classes being predicted. For the problem in hand, we have N=2, and hence we get a 2 X 2 matrix. Here are a few definitions, you need to remember for a confusion matrix :

- **Accuracy** : the proportion of the total number of predictions that were correct.
- **Positive Predictive Value or Precision** : the proportion of positive cases that were correctly identified.
- **Negative Predictive Value** : the proportion of negative cases that were correctly identified.
- **Sensitivity or Recall** : the proportion of actual positive cases which are correctly identified.
- **Specificity** : the proportion of actual negative cases which are correctly identified.



Confusion Matrix		**Target**			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		



Count of ID	**Target**			
Model	**1**	**0**	**Grand Total**	
1	3,834	639	4,473	85.7%
0	16	951	967	1.7%
Grand Total	**3,850**	**1,590**	**5,440**	
	99.6%	40.19%		88.0%

The accuracy for the problem in hand comes out to be 88%. As you can see from the above two tables, the Positive predictive Value is high, but negative predictive value is quite low. Same holds for Sensitivity and Specificity. This is primarily driven by the threshold value we have chosen. If we decrease our threshold value, the two pairs of starkly different numbers will come closer.

2. F1 Score

F1-Score is the harmonic mean of precision and recall values for a classification problem. The formula for F1-Score is as follows:

$$F_1 = \left(\frac{\text{recall}^{-1} + \text{precision}^{-1}}{2}\right)^{-1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Now, an obvious question that comes to mind is why are taking a harmonic mean and not an arithmetic mean. This is because HM punishes extreme values more. Let us understand this with an example. We have a binary classification model with the following results:

Precision: 0, Recall: 1

Here, if we take the arithmetic mean, we get 0.5. It is clear that the above result comes from a dumb classifier which just ignores the input and just predicts one of the classes as output. Now, if we were to take HM, we will get 0 which is accurate as this model is useless for all purposes.

This seems simple. There are situations however for which a data scientist would like to give a percentage more importance/weight to either precision or recall. Altering the above expression a bit such that we can include an adjustable parameter beta for this purpose, we get:

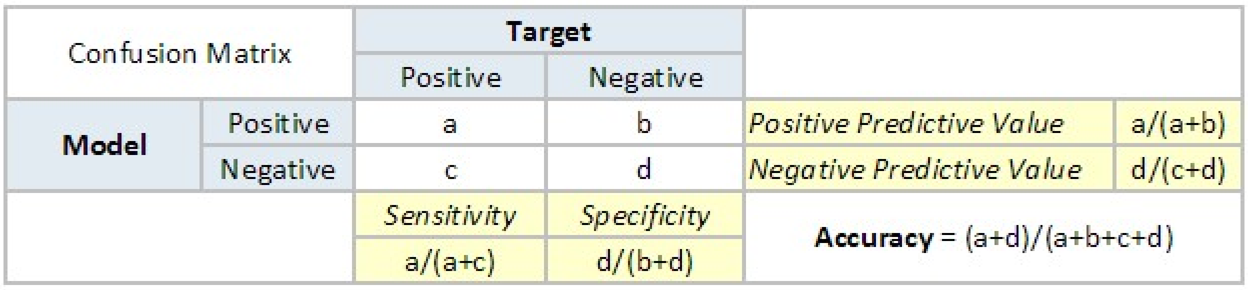
$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}.$$

Fbeta measures the effectiveness of a model with respect to a user who attaches β times as much importance to recall as precision.

3. Area Under the ROC curve (AUC – ROC)

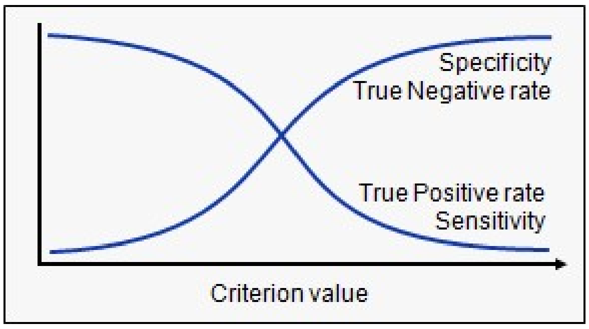
This is again one of the popular metrics used in the industry. The biggest advantage of using ROC curve is that it is independent of the change in proportion of responders. This statement will get clearer in the following sections.

Let’s first try to understand what is ROC (Receiver operating characteristic) curve. If we look at the confusion matrix below, we observe that for a probabilistic model, we get different value for each metric.

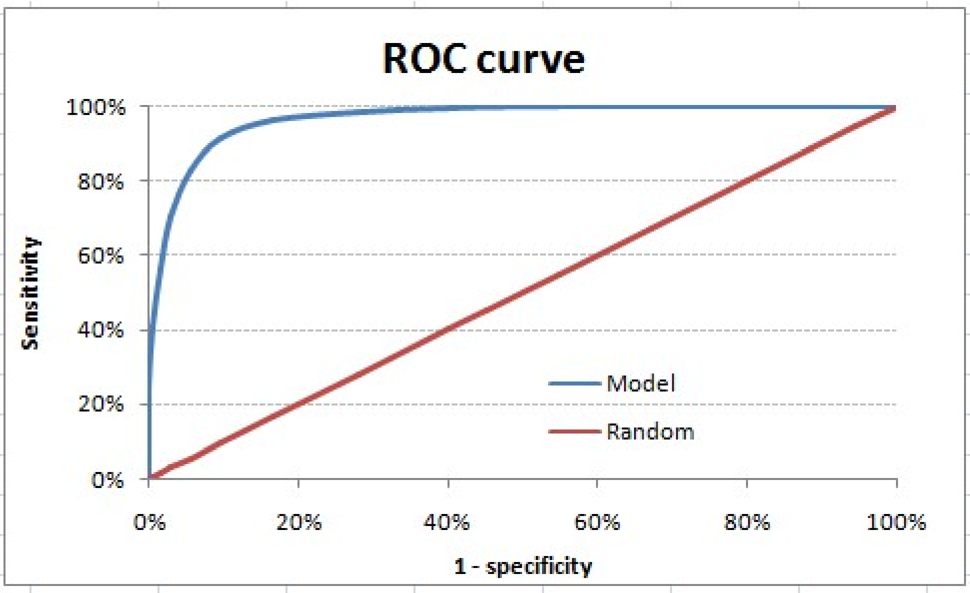


Confusion Matrix		**Target**			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

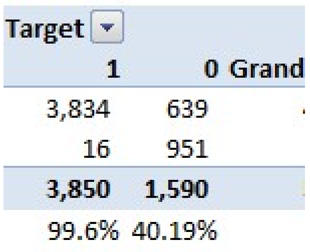
Hence, for each sensitivity, we get a different specificity.The two vary as follows:



The ROC curve is the plot between sensitivity and (1- specificity). (1- specificity) is also known as false positive rate and sensitivity is also known as True Positive rate. Following is the ROC curve for the case in hand.



Let’s take an example of threshold = 0.5 (refer to confusion matrix). Here is the confusion matrix :



Target ▾		
1	0	Grand
3,834	639	
16	951	
3,850	**1,590**	
99.6%	40.19%	

As you can see, the sensitivity at this threshold is 99.6% and the (1-specificity) is ~60%. This coordinate becomes on point in our ROC curve. To bring this curve down to a single number, we find the area under this curve (AUC).

Note that the area of entire square is 1*1 = 1. Hence AUC itself is the ratio under the curve and the total area. For the case in hand, we get AUC ROC as 96.4%. Following are a few thumb rules:

- .90-1 = excellent (A)
- .80-.90 = good (B)
- .70-.80 = fair (C)
- .60-.70 = poor (D)
- .50-.60 = fail (F)

We see that we fall under the excellent band for the current model. But this might simply be over-fitting. In such cases it becomes very important to to in-time and out-of-time validations

4. Log Loss

AUC ROC considers the predicted probabilities for determining our model's performance. However, there is an issue with AUC ROC, it only takes into account the order of probabilities and hence it does not take into account the model's capability to predict higher probability for samples more likely to be positive. In that case, we could us the log loss which is nothing but negative average of the log of corrected predicted probabilities for each instance.

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

- p(yi) is predicted probability of positive class
- 1-p(yi) is predicted probability of negative class
- yi = 1 for positive class and 0 for negative class (actual values)

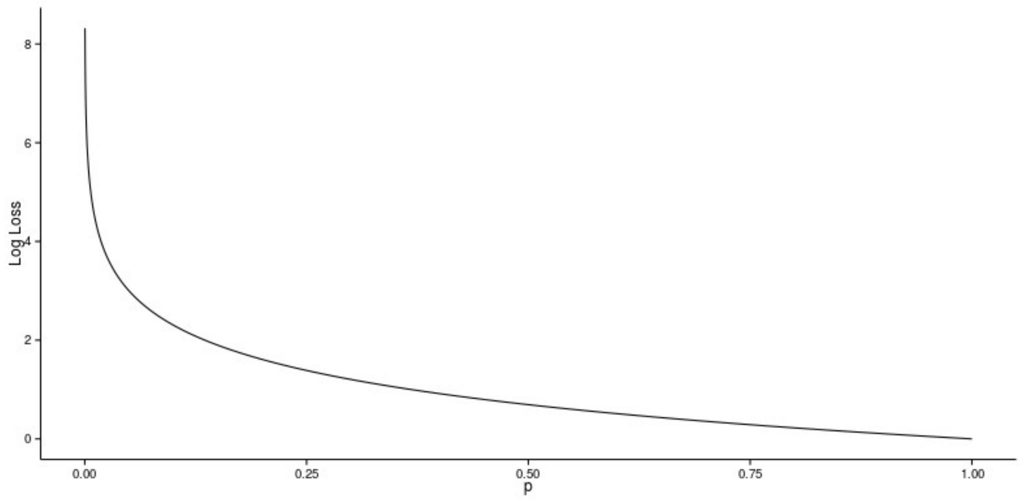
Let us calculate log loss for a few random values to get the gist of the above mathematical function:

Logloss(1, 0.1) = 2.303

Logloss(1, 0.5) = 0.693

Logloss(1, 0.9) = 0.105

If we plot this relationship, we will get a curve as follows:



It’s apparent from the gentle downward slope towards the right that the Log Loss gradually declines as the predicted probability improves. Moving in the opposite direction though, the Log Loss ramps up very rapidly as the predicted probability approaches 0.

So, lower the log loss, better the model. However, there is no absolute measure on a good log loss and it is use-case/application dependent.

Whereas the AUC is computed with regards to binary classification with a varying decision threshold, log loss actually takes “certainty” of classification into account.

5. Root Mean Squared Error (RMSE)

RMSE is the most popular evaluation metric used in regression problems. It follows an assumption that error are unbiased and follow a normal distribution. Here are the key points to consider on RMSE:

1. The power of ‘square root’ empowers this metric to show large number deviations.
2. The ‘squared’ nature of this metric helps to deliver more robust results which prevents cancelling the positive and negative error values. In other words, this metric aptly displays the plausible magnitude of error term.
3. It avoids the use of absolute error values which is highly undesirable in mathematical calculations.

4. When we have more samples, reconstructing the error distribution using RMSE is considered to be more reliable.
5. RMSE is highly affected by outlier values. Hence, make sure you've removed outliers from your data set prior to using this metric.
6. As compared to mean absolute error, RMSE gives higher weightage and punishes large errors.

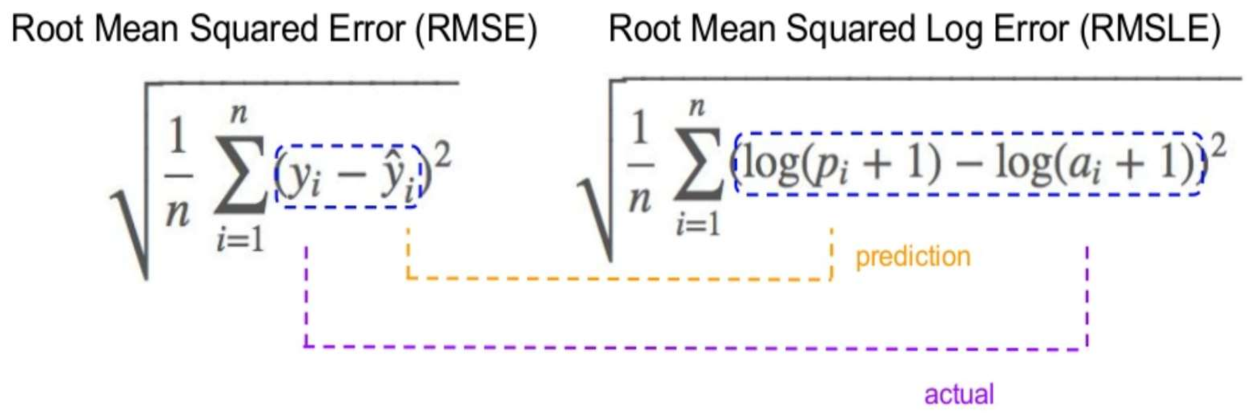
RMSE metric is given by:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

where, N is Total Number of Observations.

6. Root Mean Squared Logarithmic Error

In case of Root mean squared logarithmic error, we take the log of the predictions and actual values. So basically, what changes are the variance that we are measuring. RMSLE is usually used when we don't want to penalize huge differences in the predicted and the actual values when both predicted and true values are huge numbers.


Root Mean Squared Error (RMSE)
Root Mean Squared Log Error (RMSLE)
prediction
actual

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \qquad \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

1. If both predicted and actual values are small: RMSE and RMSLE are same.

2. If either predicted or the actual value is big: RMSE > RMSLE
3. If both predicted and actual values are big: RMSE > RMSLE (RMSLE becomes almost negligible)

11. R-Squared/Adjusted R-Squared

In the case of a classification problem, if the model has an accuracy of 0.8, we could gauge how good our model is against a random model, which has an accuracy of 0.5. So the random model can be treated as a benchmark. But when we talk about the RMSE metrics, we do not *have* a benchmark to compare.

This is where we can use R-Squared metric. The formula for R-Squared is as follows:

$$R^2 = 1 - \frac{\text{MSE(model)}}{\text{MSE(baseline)}}$$

$$\frac{\text{MSE(model)}}{\text{MSE(baseline)}} \qquad \frac{\sum_{i=1}^{N} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{N} (\overline{y}_i - \hat{y}_i)^2}$$

MSE(model): Mean Squared Error of the predictions against the actual values

MSE(baseline): Mean Squared Error of mean prediction against the actual values

In other words how good our regression model as compared to a very simple model that just predicts the mean value of target from the train set as predictions.

Adjusted R-Squared

A model performing equal to baseline would give R-Squared as 0. Better the model, higher the r2 value. The best model with all correct predictions would give R-Squared as 1. However, on adding new features to the model, the R-Squared value either increases or remains the same. R-Squared does not penalize for adding features that add no value to the model. So an improved version over the R-Squared is the adjusted R-Squared. The formula for adjusted R-Squared is given by:

$$\bar{R}^2 = 1 - \left(1 - R^2\right)\left[\frac{n-1}{n-\left(k+1\right)}\right]$$

k: number of features

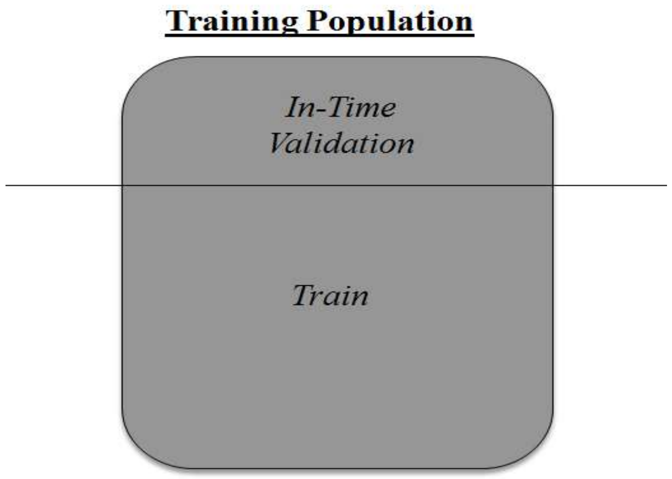
n: number of samples

As you can see, this metric takes the number of features into account. When we add more features, the term in the denominator n-(k +1) decreases, so the whole expression increases.

If R-Squared does not increase, that means the feature added isn’t valuable for our model. So overall we subtract a greater value from 1 and adjusted r2, in turn, would decrease.

12. Cross Validation

Cross Validation is one of the most important concepts in any type of data modelling. It simply says, try to leave a sample on which you do not train the model and test the model on this sample before finalizing the model.

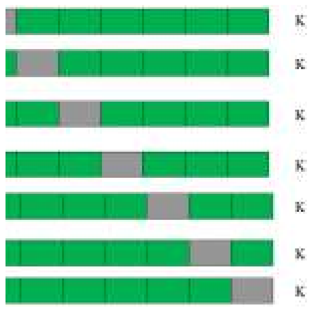


Above diagram shows how to validate model with in-time sample. We simply divide the population into 2 samples, and build model on one sample. Rest of the population is used for in-time validation.

We make a 50:50 split of training population and the train on first 50 and validate on rest 50. Then, we train on the other 50, test on first 50. This way we train the model on the entire population, however on 50% in one go. This reduces bias because of sample selection to some extent but gives a smaller sample to train the model on. This approach is known as 2-fold cross validation.

k-fold Cross validation:

Let's extrapolate the last example to k-fold from 2-fold cross validation. Now, we will try to visualize how does a k-fold validation work.



This is a 7-fold cross validation.

Here’s what goes on behind the scene : we divide the entire population into 7 equal samples. Now we train models on 6 samples (Green boxes) and validate on 1 sample (grey box). Then, at the second iteration we train the model with a different sample held as validation. In 7 iterations, we have basically built model on each sample and held each of them as validation. This is a way to reduce the selection bias and reduce the variance in prediction power. Once we have all the 7 models, we take average of the error terms to find which of the models is best.

k-fold cross validation is widely used to check whether a model is an overfit or not. If the performance metrics at each of the k times modelling are close to each other and the mean of metric is highest.

For a small k, we have a higher selection bias but low variance in the performances.

For a large k, we have a small selection bias but high variance in the performances.

Think of extreme cases :

k = 2 : We have only 2 samples similar to our 50-50 example. Here we build model only on 50% of the population each time. But as the validation is a significant population, the variance of validation performance is minimal.

k = number of observations (n) : This is also known as “Leave one out”. We have n samples and modelling repeated n number of times leaving only one observation out for cross validation. Hence, the selection bias is minimal but the variance of validation performance is very large.

Generally a value of k = 10 is recommended for most purpose.