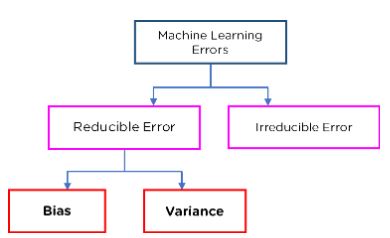
**Machine Learning Resource File**

**Bias(Underfitting) and Variance(Overfitting) :**

There are two main types of errors present in any [machine learning](https://www.simplilearn.com/tutorials/machine-learning-tutorial) model. They are Reducible Errors and Irreducible Errors.

* Irreducible errors are errors which will always be present in a machine learning model, because of unknown variables, and whose values cannot be reduced.
* Reducible errors are those errors whose values can be further reduced to improve a model. They are caused because our model’s output function does not match the desired output function and can be optimized.

We can further divide reducible errors into two: Bias and Variance.



What is Bias?

To make predictions, our model will analyze our data and find patterns in it. Using these patterns, we can make generalizations about certain instances in our data. Our model after training learns these patterns and applies them to the test set to predict them.

Bias is the difference between our actual and predicted values. Bias is the simple assumptions that our model makes about our data to be able to predict new data.

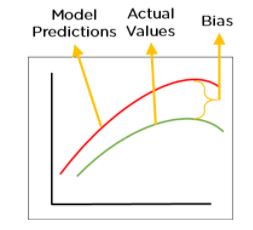


Figure 2: Bias

When the Bias is high, assumptions made by our model are too basic, the model can’t capture the important features of our data. This means that our model hasn’t captured patterns in the training data and hence cannot perform well on the testing data too. If this is the case, our model cannot perform on new data and cannot be sent into production.

This instance, where the model cannot find patterns in our training set and hence fails for both seen and unseen data, is called Underfitting.

The below figure shows an example of Underfitting. As we can see, the model has found no patterns in our data and the line of best fit is a straight line that does not pass through any of the data points. The model has failed to train properly on the data given and cannot predict new data either.

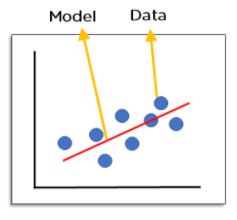


                          Figure 3: Underfitting

What is Variance?

Variance is the very opposite of Bias. During training, it allows our model to ‘see’ the data a certain number of times to find patterns in it. If it does not work on the data for long enough, it will not find patterns and bias occurs. On the other hand, if our model is allowed to view the data too many times, it will learn very well for only that data. It will capture most patterns in the data,  but it will also learn from the unnecessary data present, or from the noise.

We can define variance as the model’s sensitivity to fluctuations in the data. Our model may learn from noise. This will cause our model to consider trivial features as important.

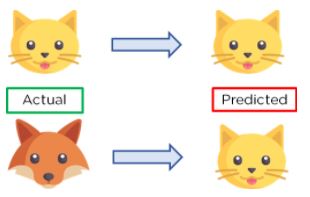


                                                      Figure 4: Example of Variance

In the above figure, we can see that our model has learned extremely well for our training data, which has taught it to identify cats. But when given new data, such as the picture of a fox, our model predicts it as a cat, as that is what it has learned. This happens when the Variance is high, our model will capture all the features of the data given to it, including the noise, will tune itself to the data, and predict it very well but when given new data, it cannot predict on it as it is too specific to training data.

Hence, our model will perform really well on testing data and get high accuracy but will fail to perform on new, unseen data. New data may not have the exact same features and the model won’t be able to predict it very well. This is called Overfitting.

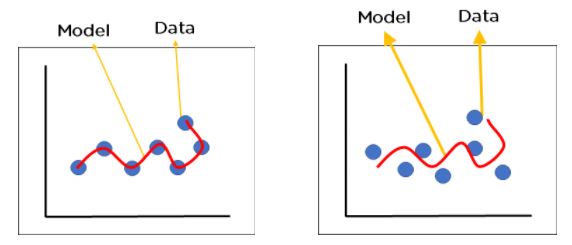


           Figure 5: Over-fitted model where we see model performance on, a) training data b) new data

Bias-Variance Tradeoff

For any model, we have to find the perfect balance between Bias and Variance. This just ensures that we capture the essential patterns in our model while ignoring the noise present it in. This is called Bias-Variance Tradeoff. It helps optimize the error in our model and keeps it as low as possible.

An optimized model will be sensitive to the patterns in our data, but at the same time will be able to generalize to new data. In this, both the bias and variance should be low so as to prevent [overfitting and underfitting](https://www.simplilearn.com/tutorials/machine-learning-tutorial/overfitting-and-underfitting).

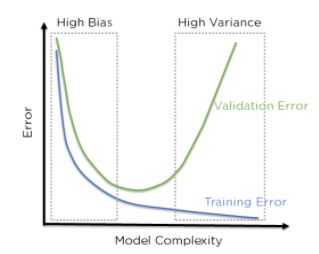


                                     Figure 6: Error in Training and Testing with high Bias and Variance

In the above figure, we can see that when bias is high, the error in both testing and training set is also high. If we have a high variance, the model performs well on the testing set, we can see that the error is low, but gives high error on the training set. We can see that there is a region in the middle, where the error in both training and testing set is low and the bias and variance is in perfect balance.

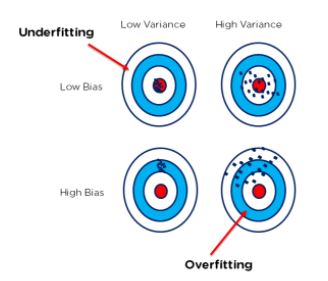


  Figure 7: Bull’s Eye Graph for Bias and Variance

The above bull’s eye graph helps explain bias and variance tradeoff better. The best fit is when the data is concentrated in the center, ie: at the bull’s eye. We can see that as we get farther and farther away from the center, the error increases in our model. The best model is one where bias and variance are both low.

**Regularization :**

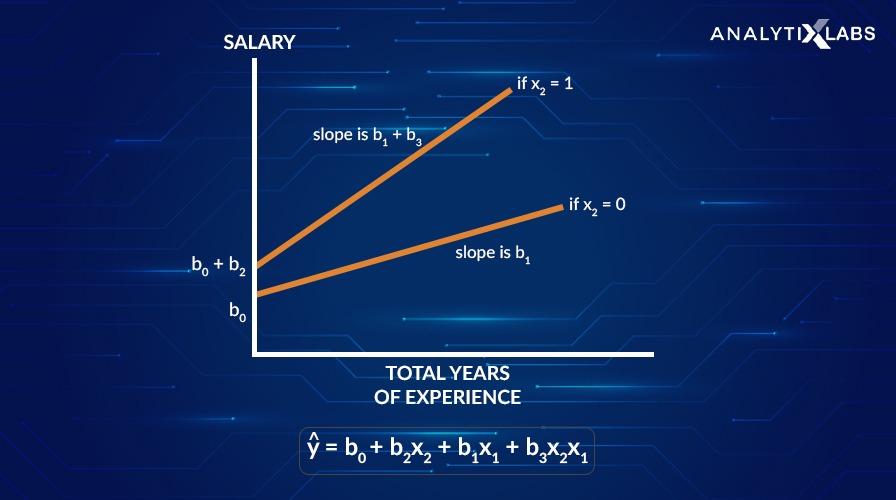
Generalization error is “ a measure of how accurately an algorithm can predict outcome values for previously unseen data.”

Regularization refers to the modifications that can be made to a learning algorithm that helps to reduce this generalization error and not the training error. It reduces by ignoring the less important features. It also helps prevent overfitting, making the model more robust and decreasing the complexity of a model.

**The regularization techniques in machine learning are:**

* Lasso regression: having the L1 norm
* Ridge regression: with the L2 norm
* Elastic net regression: It is a combination of Ridge and Lasso regression.

**How Does Regularization Work?**

Regularization works by shrinking the beta coefficients of a linear regression model. To understand why we need to shrink the coefficients, let us see the below example: 

In the above graph, the two lines represent the relationship between total years of experience and salary, where salary is the target variable. These are slopes indicating the change in salary per unit change in total years of experience. As the slope b1 + b3 decreases to slope b1, we see that the salary is less sensitive to the total years of experience.

By decreasing the slope, the target variable (salary) became less sensitive to the change in the independent X variables, which increases the bias into the model. Remember, bias is the difference between the predicted and the actual values.

With the increase in bias to the model, the variance (which is the difference between the predictions when the model fits different datasets.) decreases. And, by decreasing the variance, the overfitting gets reduced.

The models having the higher variance leads to overfitting, and we saw above, we will shrink or reduce the beta coefficients to overcome the overfitting. The beta coefficients or the weights of the features converge towards zero, which is known as shrinkage.

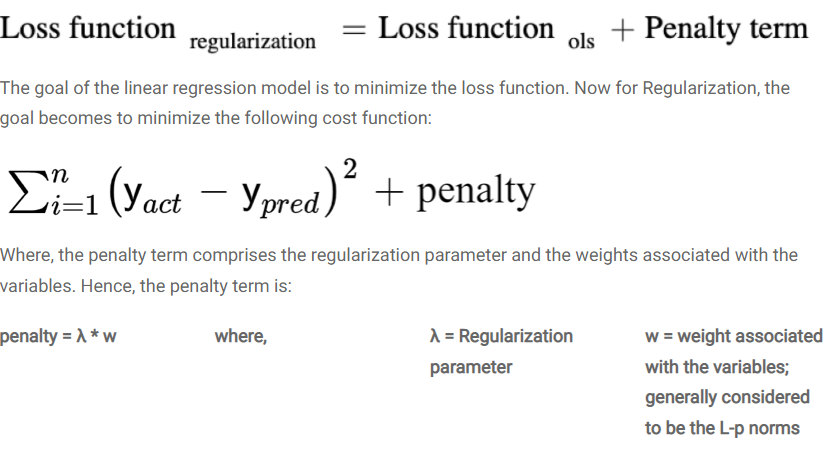
Now, to do this, we penalize the model that has higher variance. So, Regularization adds a penalty term to the loss function of the linear regression model such that the model with higher variance gets a larger penalty.

**What Is the Regularization Parameter?**

**For linear regression, the regularization has two terms in the loss function:**

1. The Ordinary Least Squares (OLS) function, and
2. The penalty term

It becomes :

****

**The regularization parameter in machine learning is λ :**

* It imposes a higher penalty on the variable having higher values, and hence, it controls the strength of the penalty term.
* This tuning parameter controls the bias-variance trade-off.
* λ can take values 0 to infinity
* If λ = 0, then means there is no difference between a model with and without regularization.

We shall see the L-p norms in the following section while discussing the various regularization techniques.

## ****Regularization Techniques in Machine Learning****

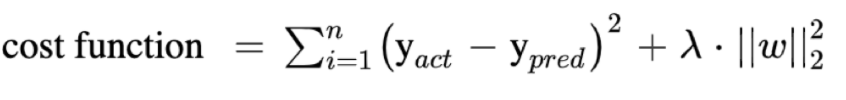
Each of the following techniques uses different regularization norms (L-p) based on the mathematical methodology that creates different kinds of regularization. These methodologies have different effects on the beta coefficients of the features. The regularization techniques in machine learning as follows:

### **1. Ridge Regression**

The Ridge regression technique is used to analyze the model where the variables may be having multicollinearity. It reduces the insignificant independent variables though it does not remove them completely. This type of regularization uses the L2 norm for regularization.

* It uses the L2-norm as the penalty.
* L2 penalty is the square of the magnitudes of beta coefficients.
* It is also known as L2-regularization.
* L2 shrinks the coefficients, however never make them to zero.
* The output of L2 regularization is non-sparse.

The cost function of the Ridge regression becomes:

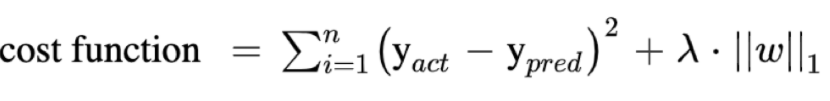


### **2. Lasso Regression**

Least Absolute Shrinkage and Selection Operator (or LASSO) Regression penalizes the coefficients to the extent that it becomes zero. It eliminates the insignificant independent variables. This regularization technique uses the L1 norm for regularization.

* It adds L1-norm as the penalty.
* L1 is the absolute value of the beta coefficients.
* It is also known as the L-1 regularization.
* The output of L1 regularization is sparse.

It is useful when there are many variables, as this technique can be used as a feature selection method by itself. The cost function for the LASSO regression is:

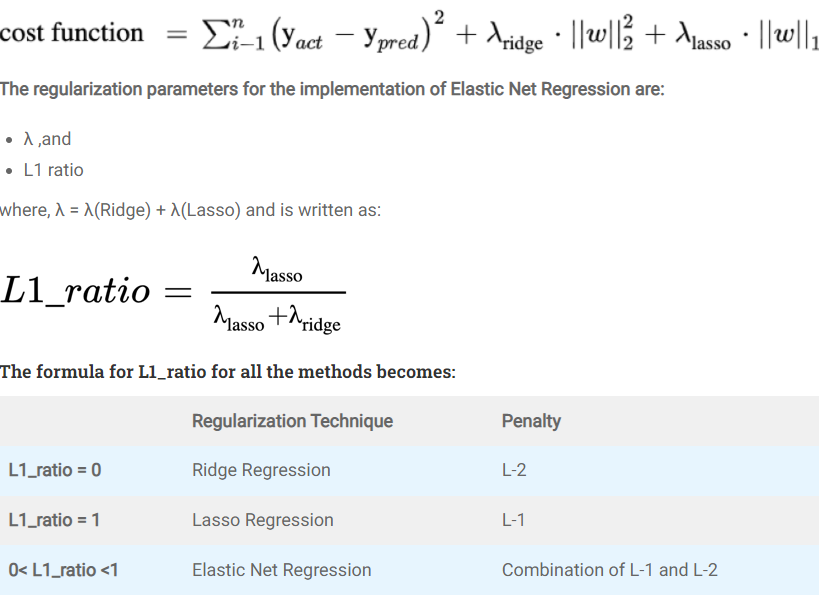


### **3. Elastic Net Regression**

The Elastic Net Regression technique is a combination of the Ridge and Lasso regression technique. It is the linear combination of penalties for both the L1-norm and L2-norm regularization.

The model using elastic net regression allows the learning of the sparse model where some of the points are zero, similar to Lasso regularization, and yet maintains the Ridge regression properties. Therefore, the model is trained on both the L1 and L2 norms.

The cost function of Elastic Net Regression is:



**In Python, the strength of the penalty is controlled by the argument called *alpha*. If:**

* alpha = 0, then will become Ridge, and
* alpha = 1, then it becomes Lasso.

**The Elastic Net Regression happens in two stages:**

* the first stage is Ridge Regression, and
* then the second stage is the LASSO Regression.

In each stage, the betas get reduced or shrunken, resulting in an increased bias in the model. An increase in bias means a lot of deviation between the predicted values and the actual values and therefore makes more predictions. Hence, we rescale the coefficients of Elastic Net Regularization by multiplying the estimated coefficients by (1+λridge) to improve the prediction performance and decrease the bias.

## ****When to Use Which Regularization Technique?****

The regularization in machine learning is used in following scenarios:

* Ridge regression is used when it is important to consider all the independent variables in the model or when many interactions are present. That is where collinearity or codependency is present amongst the variables.
* Lasso regression is applied when there are many predictors available and would want the model to make feature selection as well for us.
* When many variables are present, and we can’t determine whether to use Ridge or Lasso regression, then the Elastic-Net regression is your safe bet.

## ****Summary****

Based on Occam’s Razor, Regularization is one of the key concepts in Machine learning. It helps prevent the problem of overfitting, makes the model more robust, and decreases the complexity of a model.

In summary, regularization chooses a model (making Occam’s Razor applicable) with smaller weights of the features (or shrunken beta coefficients) that have less generalization error. In addition, it penalizes the model having higher variance by adding a penalty term to the loss function to prevent the larger values from being weighed too heavily.

The regularization techniques are Lasso Regression, Ridge Regression, and Elastic Net Regression. Regularization can be used for feature reduction.

### **Q1. What are L1 and L2 regularization?**

L1 and L2 are the norms of regularization. These are the types of regularization that help in reducing the overfitting problem by reducing the less important features. It is done by shrinking the beta coefficients of the linear regression model. The differences between L1 and L2 norms are as follows:

|  |  |
| --- | --- |
| **L1 regularization** | **L2 regularization** |
| L1 is equal to the absolute value of the beta coefficients. | L2 is equal to the squares of magnitudes of beta coefficients. |
| It is used in Lasso regression. | It is used in Ridge regression. |
| To prevent overfitting, L1 estimates the median of the data. | L2 estimates the mean of the data to avoid overfitting. |
| L1 has built-in feature selection. | L2 is not used for feature selection. |
| L1 returns sparse outputs. | L2 results in non-sparse outputs. |
| It is computationally inefficient on the non-sparse cases. | It is computationally efficient since it has analytical solutions. |

### **Q2. What is L1 and L2 Penalty?**

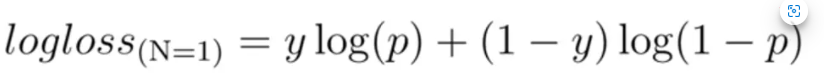
* The L1 penalty is the absolute of the weights which are added to the loss function and are used in Lasso regression, whereas
* L2 penalty is the square of the weights. This penalty term is added to the loss function, which is used in Ridge regression.

### **Q3. What is an advantage of L1 regularization over L2 regularization?**

L1 regularization shrinks the beta coefficients or the weights of the features in the model to zero and hence, is useful for eliminating the unimportant features. On the other hand, L2 regularization shrinks the weights evenly and is applicable when multicollinearity is present in the data. The advantage of L1 regularization over L2 is that L1 can be used for feature selection.

## ****Q4. Is regularization helpful for logistic regression?****

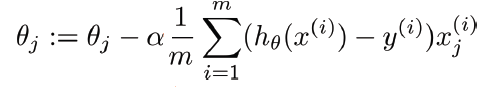
Yes, regularization is helpful for logistic regression. As the logistic regression is applied for classification problems, wherein the case of binary classification predicts whether it belongs to class 1 or class 0, therefore the loss function changes to:



**Feature Scaling :**

### **Gradient Descent Based Algorithms**

**Machine learning algorithms like**[**linear regression**](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization)**,**[**logistic regression**](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization)**,**[**neural network**](https://www.analyticsvidhya.com/blog/2017/05/neural-network-from-scratch-in-python-and-r/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization)**, etc. that use gradient descent as an optimization technique require data to be scaled.** Take a look at the formula for gradient descent below:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/gradient-descent.png)

The presence of feature value X in the formula will affect the step size of the gradient descent. The difference in ranges of features will cause different step sizes for each feature. To ensure that the gradient descent moves smoothly towards the minima and that the steps for gradient descent are updated at the same rate for all the features, we scale the data before feeding it to the model.

Having features on a similar scale can help the gradient descent converge more quickly towards the minima.

### **Distance-Based Algorithms**

Distance algorithms like [KNN](https://www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), [K-means](https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), and [SVM](https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization) are most affected by the range of features. This is because behind the scenes **they are using distances between data points to determine their similarity.**

### **Tree-Based Algorithms**

[Tree-based algorithms](https://www.analyticsvidhya.com/blog/2016/04/tree-based-algorithms-complete-tutorial-scratch-in-python/?utm_source=blog&utm_medium=feature-scaling-machine-learning-normalization-standardization), on the other hand, are fairly insensitive to the scale of the features. Think about it, a decision tree is only splitting a node based on a single feature. The decision tree splits a node on a feature that increases the homogeneity of the node. This split on a feature is not influenced by other features.

So, there is virtually no effect of the remaining features on the split. This is what makes them invariant to the scale of the features!

## What is Normalization?

**Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.**

Here’s the formula for normalization:

[Normalization equation](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Norm_eq.gif)

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

* When the value of X is the minimum value in the column, the numerator will be 0, and hence X’ is 0
* On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X’ is 1
* If the value of X is between the minimum and the maximum value, then the value of X’ is between 0 and 1

## What is Standardization?

**Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.**

Here’s the formula for standardization:

[Standardization equation](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/03/Stand_eq.gif)

Feature scaling: Muis the mean of the feature values andFeature scaling: Sigmais the standard deviation of the feature values. Note that in this case, the values are not restricted to a particular range.

Now, the big question in your mind must be when should we use normalization and when should we use standardization? Let’s find out!

 The Big Question – Normalize or Standardize?

Normalization vs. standardization is an eternal question among machine learning newcomers. Let me elaborate on the answer in this section.

* Normalization is good to use when you know that the distribution of your data does not follow a Gaussian distribution. This can be useful in algorithms that do not assume any distribution of the data like K-Nearest Neighbors and Neural Networks.
* Standardization, on the other hand, can be helpful in cases where the data follows a Gaussian distribution. However, this does not have to be necessarily true. Also, unlike normalization, standardization does not have a bounding range. So, even if you have outliers in your data, they will not be affected by standardization.

However, at the end of the day, the choice of using normalization or standardization will depend on your problem and the machine learning algorithm you are using. There is no hard and fast rule to tell you when to normalize or standardize your data. **You can always start by fitting your model to raw, normalized and standardized data and compare the performance for best results.**

It is a good practice to fit the scaler on the training data and then use it to transform the testing data. This would avoid any data leakage during the model testing process. Also, the scaling of target values is generally not required*.*

**Difference between Normalization and Standardization**

| S.NO. | Normalization | Standardization |
| --- | --- | --- |
| 1. | Minimum and maximum value of features are used for scaling | Mean and standard deviation is used for scaling. |
| 2. | It is used when features are of different scales. | It is used when we want to ensure zero mean and unit standard deviation. |
| 3. | Scales values between [0, 1] or [-1, 1]. | It is not bounded to a certain range. |
| 4. | It is really affected by outliers. | It is much less affected by outliers. |
| 5. | Scikit-Learn provides a transformer called MinMaxScaler for Normalization. | Scikit-Learn provides a transformer called StandardScaler for standardization. |
| 6. | This transformation squishes the n-dimensional data into an n-dimensional unit hypercube. | It translates the data to the mean vector of original data to the origin and squishes or expands. |
| 7. | It is useful when we don’t know about the distribution | It is useful when the feature distribution is Normal or Gaussian. |
| 8. | It is a often called as Scaling Normalization | It is a often called as Z-Score Normalization. |

**Gradient Descent :**

Optimization refers to a procedure for finding the input parameters or arguments to a function that result in the minimum or maximum output of the function.

Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).

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

Gradient Descent is an iterative process that finds the minima of a function. This is an optimisation algorithm that finds the parameters or coefficients of a function where the function has a minimum value. Although this function does not always guarantee to find a global minimum and can get stuck at a local minimum.

To understand the difference between local minima and global minima, take a look at the figure above. The global minimum is the least value of a function while a local minimum is the least value of a function in a certain neighbourhood.

The cost function is used to quantify the error between the predicted values and the real values of a Machine Learning model built.

## Gradient Descent Procedure

This process of gradient descent begins with allocating values initially to the coefficients of the cost function. This could be either a value close to 0 or a small random value.

**coefficient = 0.0**

Next, the cost of the coefficients is obtained by applying it to the cost function and calculating the cost.

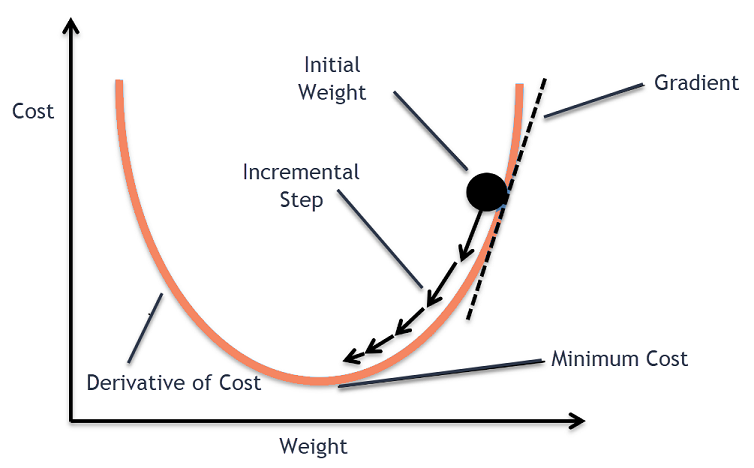
**cost = f(coefficient)**

Then, the derivative of the cost function is calculated. This derivative of the cost function is obtained by the mathematical concept of differential calculus. It gives us the slope of the function at the given point where its derivative is calculated. This slope is needed to know in which direction the coefficient is to be moved in the next iteration to get a lower cost value. This is done by observing the sign of the derivative calculated.

**delta = derivative(cost)**

Once we know which direction is downhill from the derivative calculated, we need to update the coefficient values. For this, a parameter is known as the learning parameter, alpha (α) is utilized. This is used to control to what extent the coefficients can change with every update.

**coefficient = coefficient – (alpha \* delta)**



In this way, this process is repeated till the cost of the coefficients is equal to 0.0 or close enough to zero. This is the procedure for the gradient descent algorithm.

**Gradient Descent in Deep Learning:**

Gradient Descent is a process that occurs in the **backpropagation** phase where the goal is to continuously resample the gradient of the model’s parameter in the opposite direction based on the weight *w*, updating consistently until we reach the **global minimum**of function *J(w)*.

To put it simply, we use gradient descent to minimize the cost function, *J(w)*.

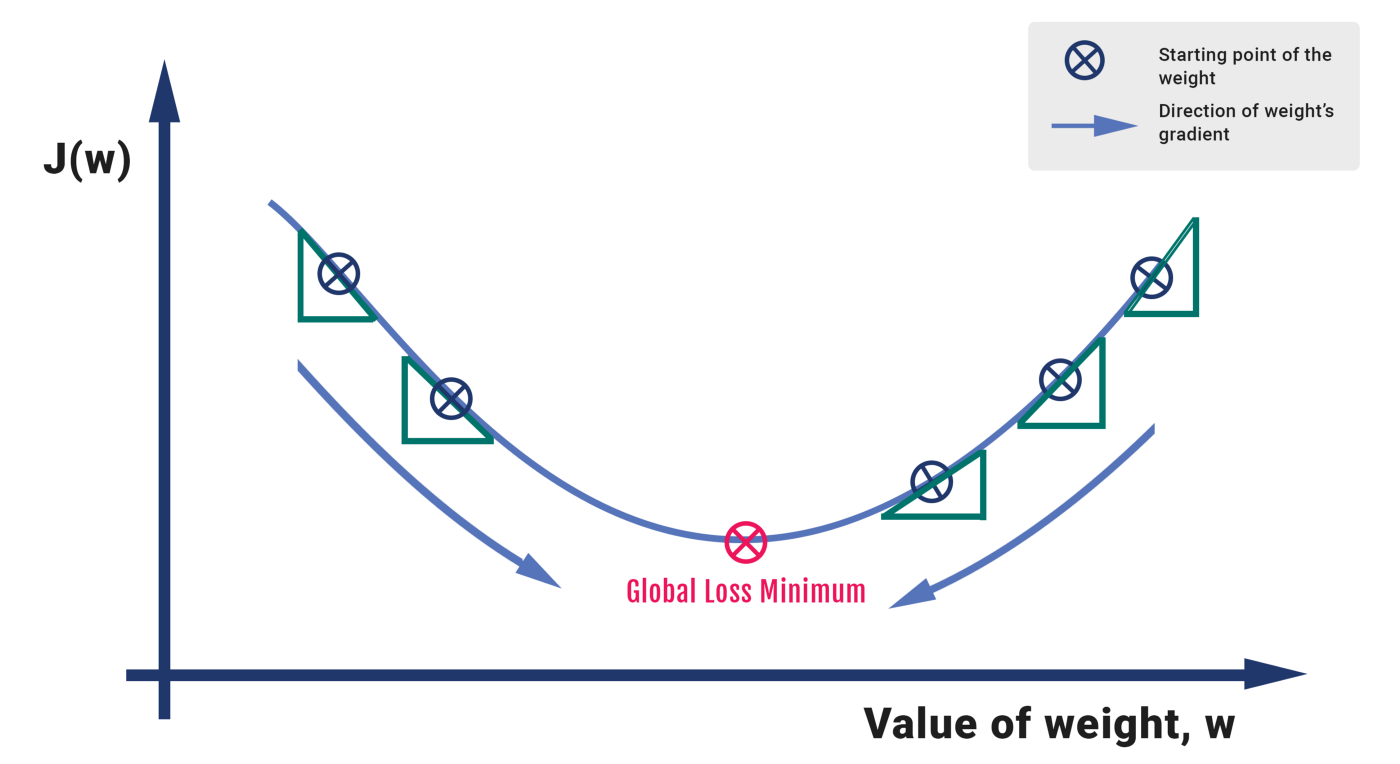
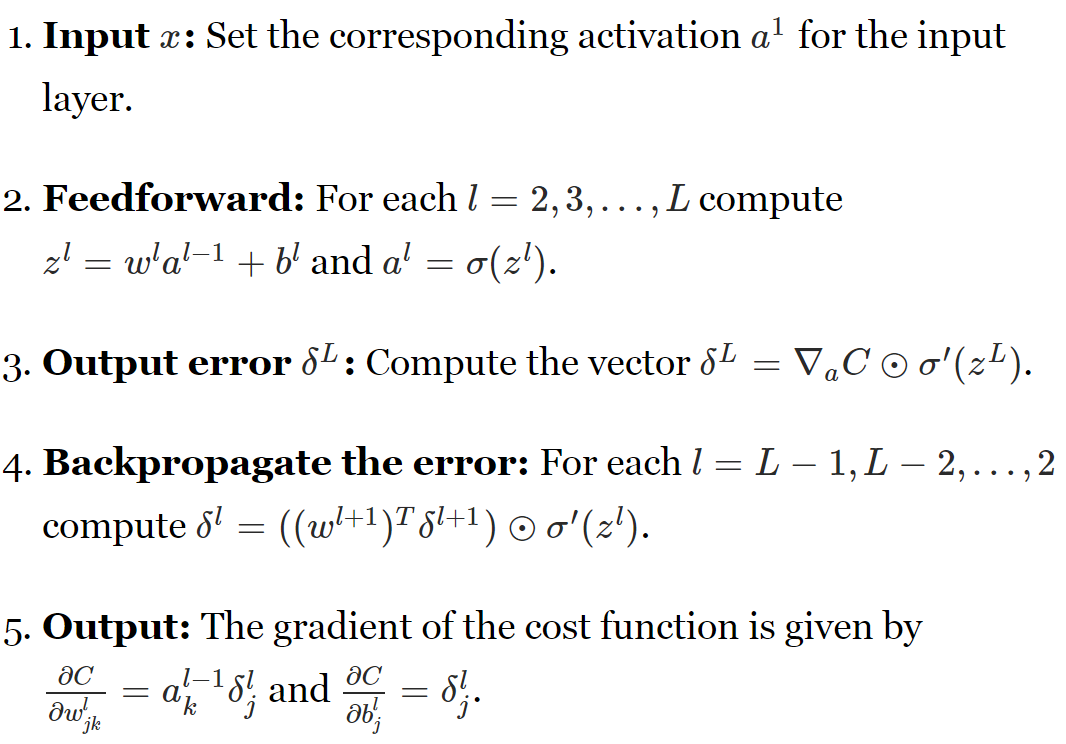


Fig 1: How Gradient Descent works for one parameter, *w*

## Mathematics

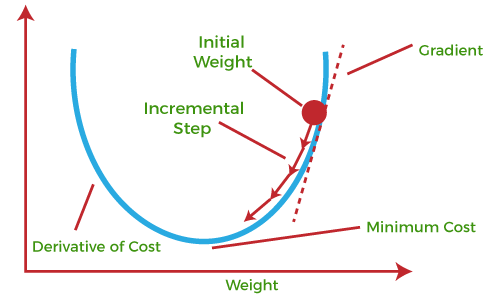
Let’s formalize the analogy into an algorithmic form. We compute the activation for the incoming parameters, we carry out feedforward by taking the weighted sum of the activation and its bias. We extract the error term of the output sample by subtracting it with the actual ‘target’ value.



The gradient descent process is exhibited in the form of the backpropagation step where we compute the error vectors δ backward, starting from the final layer. Depending upon the activation function, we identify how much change is required by much change is required by taking the partial derivative of the function with respect to **w.**The change value gets multiplied by the learning rate. As part of the output, we subtract this value from the previous output to get the updated value. We continue this till we reach convergence.

The best way to define the local minimum or local maximum of a function using gradient descent is as follows:

* If we move towards a negative gradient or away from the gradient of the function at the current point, it will give the **local minimum** of that function.
* Whenever we move towards a positive gradient or towards the gradient of the function at the current point, we will get the **local maximum** of that function.



This entire procedure is known as Gradient Ascent, which is also known as steepest descent. ***The main objective of using a gradient descent algorithm is to minimize the cost function using iteration.*** To achieve this goal, it performs two steps iteratively:

* Calculates the first-order derivative of the function to compute the gradient or slope of that function.
* Move away from the direction of the gradient, which means slope increased from the current point by alpha times, where Alpha is defined as Learning Rate. It is a tuning parameter in the optimization process which helps to decide the length of the steps.

### **What is Cost-function?**

**The cost function is defined as the measurement of difference or error between actual values and expected values at the current position and present in the form of a single real number.** It helps to increase and improve machine learning efficiency by providing feedback to this model so that it can minimize error and find the local or global minimum. Further, it continuously iterates along the direction of the negative gradient until the cost function approaches zero. At this steepest descent point, the model will stop learning further. Although cost function and loss function are considered synonymous, also there is a minor difference between them. The slight difference between the loss function and the cost function is about the error within the training of machine learning models, as loss function refers to the error of one training example, while a cost function calculates the average error across an entire training set.

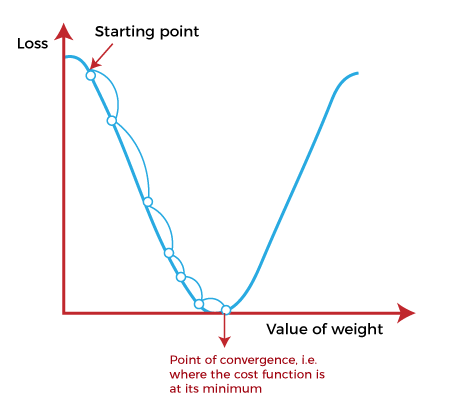
The cost function is calculated after making a hypothesis with initial parameters and modifying these parameters using gradient descent algorithms over known data to reduce the cost function.

### **How does Gradient Descent work?**

Before starting the working principle of gradient descent, we should know some basic concepts to find out the slope of a line from linear regression. The equation for simple linear regression is given as:

Y=mX+c

Where 'm' represents the slope of the line, and 'c' represents the intercepts on the y-axis.



The starting point(shown in above fig.) is used to evaluate the performance as it is considered just as an arbitrary point. At this starting point, we will derive the first derivative or slope and then use a tangent line to calculate the steepness of this slope. Further, this slope will inform the updates to the parameters (weights and bias).

The slope becomes steeper at the starting point or arbitrary point, but whenever new parameters are generated, then steepness gradually reduces, and at the lowest point, it approaches the lowest point, which is called **a point of convergence.**

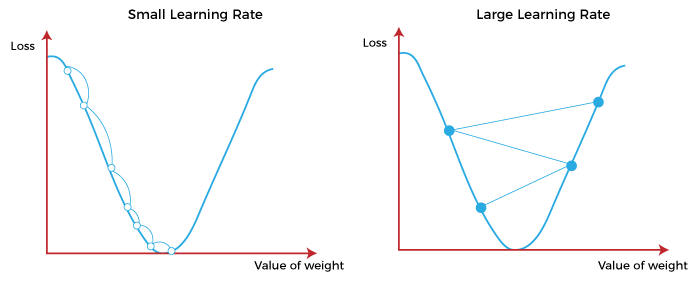
The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required:

* **Direction & Learning Rate**

These two factors are used to determine the partial derivative calculation of future iteration and allow it to the point of convergence or local minimum or global minimum. Let's discuss learning rate factors in brief;

### **Learning Rate:**

It is defined as the step size taken to reach the minimum or lowest point. This is typically a small value that is evaluated and updated based on the behavior of the cost function. If the learning rate is high, it results in larger steps but also leads to risks of overshooting the minimum. At the same time, a low learning rate shows the small step sizes, which compromises overall efficiency but gives the advantage of more precision.



## Types of Gradient Descent

Based on the error in various training models, the Gradient Descent learning algorithm can be divided into **Batch gradient descent, stochastic gradient descent, and mini-batch gradient descent.** Let's understand these different types of gradient descent:

### **1. Batch Gradient Descent:**

Batch gradient descent (BGD) is used to find the error for each point in the training set and update the model after evaluating all training examples. This procedure is known as the training epoch. In simple words, it is a greedy approach where we have to sum over all examples for each update.

**Advantages of Batch gradient descent:**

* It produces less noise in comparison to other gradient descent.
* It produces stable gradient descent convergence.
* It is Computationally efficient as all resources are used for all training samples.

### **2. Stochastic gradient descent**

Stochastic gradient descent (SGD) is a type of gradient descent that runs one training example per iteration. Or in other words, it processes a training epoch for each example within a dataset and updates each training example's parameters one at a time. As it requires only one training example at a time, hence it is easier to store in allocated memory. However, it shows some computational efficiency losses in comparison to batch gradient systems as it shows frequent updates that require more detail and speed. Further, due to frequent updates, it is also treated as a noisy gradient. However, sometimes it can be helpful in finding the global minimum and also escaping the local minimum.

**Advantages of Stochastic gradient descent:**

In Stochastic gradient descent (SGD), learning happens on every example, and it consists of a few advantages over other gradient descent.

* It is easier to allocate in desired memory.
* It is relatively fast to compute than batch gradient descent.
* It is more efficient for large datasets.

### **3. MiniBatch Gradient Descent:**

Mini Batch gradient descent is the combination of both batch gradient descent and stochastic gradient descent. It divides the training datasets into small batch sizes then performs the updates on those batches separately. Splitting training datasets into smaller batches make a balance to maintain the computational efficiency of batch gradient descent and speed of stochastic gradient descent. Hence, we can achieve a special type of gradient descent with higher computational efficiency and less noisy gradient descent.

**Advantages of Mini Batch gradient descent:**

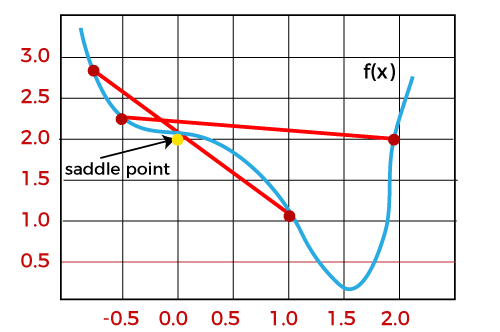
* It is easier to fit in allocated memory.
* It is computationally efficient.
* It produces stable gradient descent convergence.

## Challenges with the Gradient Descent

Although we know Gradient Descent is one of the most popular methods for optimization problems, it still also has some challenges. There are a few challenges as follows:

### **1. Local Minima and Saddle Point:**

For convex problems, gradient descent can find the global minimum easily, while for non-convex problems, it is sometimes difficult to find the global minimum, where the machine learning models achieve the best results.



Whenever the slope of the cost function is at zero or just close to zero, this model stops learning further. Apart from the global minimum, there occur some scenarios that can show this slop, which is saddle point and local minimum. Local minima generate the shape similar to the global minimum, where the slope of the cost function increases on both sides of the current points.

In contrast, with saddle points, the negative gradient only occurs on one side of the point, which reaches a local maximum on one side and a local minimum on the other side. The name of a saddle point is taken by that of a horse's saddle.

The name of local minima is because the value of the loss function is minimum at that point in a local region. In contrast, the name of the global minima is given so because the value of the loss function is minimum there, globally across the entire domain the loss function.

### **2. Vanishing and Exploding Gradient**

In a deep neural network, if the model is trained with gradient descent and backpropagation, there can occur two more issues other than local minima and saddle point.

### **Vanishing Gradients:**

Vanishing Gradient occurs when the gradient is smaller than expected. During backpropagation, this gradient becomes smaller that causing the decrease in the learning rate of earlier layers than the later layer of the network. Once this happens, the weight parameters update until they become insignificant.

### **Exploding Gradient:**

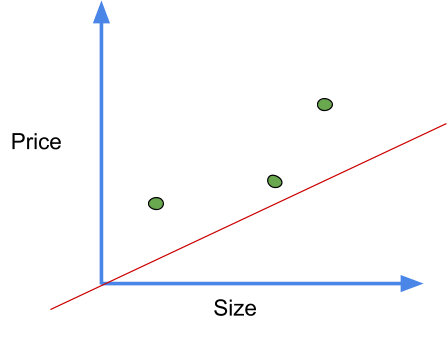
Exploding gradient is just opposite to the vanishing gradient as it occurs when the Gradient is too large and creates a stable model. Further, in this scenario, model weight increases, and they will be represented as NaN. This problem can be solved using the dimensionality reduction technique, which helps to minimize complexity within the model.

**Explanation of Gradient Descent**

Take an example of a data-set where we are given prices of various houses depending upon their area. For simplicity, we ‘ll only consider a few examples from the dataset with the following price and area.

|  |  |
| --- | --- |
| Area (Acre sq) | Price(in millions) |
| 0.5 | 1.4 |
| 2.3 | 1.9 |
| 2.9 | 3.2 |

Here is a representation of this data on the graph. To fit the best fit line we have to optimise the slope of the line and the intercept of the line. For simplicity, we take a constant slope of 0.64, so that we can understand how gradient descent would optimise the intercept. In the next section, we implement gradient descent on the slope and intercept simultaneously.

[](data:image/svg+xml,%3Csvg%20xmlns=)

First, we calculate the residual errors for each. Follow the below steps to calculate it

The gradient descent is provided with a random guess for the value of the intercept.

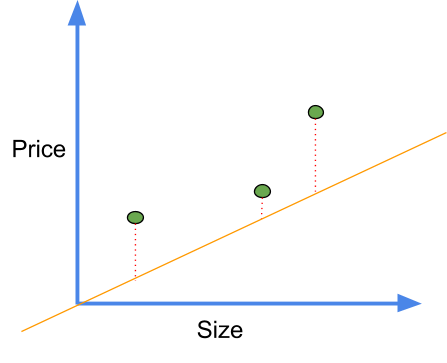
In our case, we take a random guess of zero, so the equation becomes

Predicted value = intercept + slope \* x ( If you are not familiar with this formula refer to Linear Regression)

The predicted values for the above can be calculated like this.

predicted value = 0 + 0.64 \* 0.5=0.32

The rest can be calculated in similar manner

[](data:image/svg+xml,%3Csvg%20xmlns=)Residual errors represented by red dotted lines

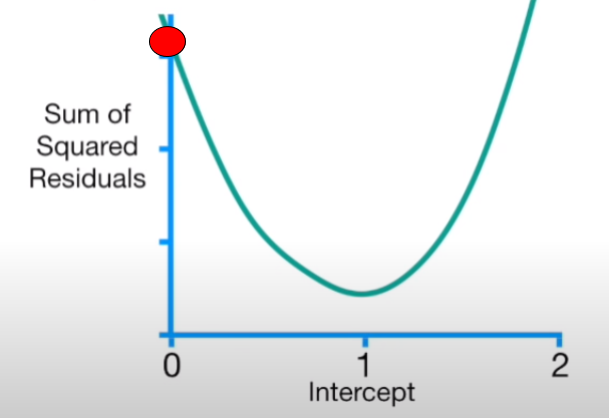
Next, we calculate the squared residual error for each point

Squared Residual error= (actual error - predicted)^2

For the first point, squared residual error = (1.4-0.32)^2 = (1.1)^2

Thus the sum of squared error = (1.1)^2 + (0.4)^2 + (1.3)^2 =3.1

Now we plot this point in a graph with the value of intercept as X-axis and value of a sum of squared error as Y-axis. In a similar manner, we plot points for many values of intercept. The plot represents the cost functions and looks like this.

[](data:image/svg+xml,%3Csvg%20xmlns=)

The primary task of Gradient Descent is to find the minimum of this cost function. To find the minimum point, we find its derivatives with respect to intercept. So the equation of this cost function is given by

f(intercept) = (1.4-(intercept+ 0.64 \* 0.5))^2 +

(1.9-(intercept+0.64 \* 2.3))^2 +

(3.2-(intercept+0.64 \* 2.9))^2

The derivative of this function with respect to intercept is given by

Derivative= d/d(intercept)(1.4-(intercept+ 0.64 \* 0.5))^2

+ d/d(intercept) (1.9-(intercept+0.64 \* 2.3))^2

+ d/d(intercept)(3.2-(intercept+0.64 \* 2.9))^2

Applying chain rule, we find derivative of each term individually and add them up. Note that here slope is taken constant so its derivative is zero.

Derivative of (1.4-(intercept+0.64 \* 0.5))^2 = - 2 (1.4-(intercept+0.64 \* 0.5))

In a similar way we find derivatives of next two terms and the value we get is

Derivative= - 2 (1.4-(intercept+0.64 \* 0.5))+

-2 (1.9-(intercept+0.64 \* 2.3))+

-2 (3.2-(intercept+0.64 \* 2.9))

Let us put the value of intercept=0 to find the value of the next intercept

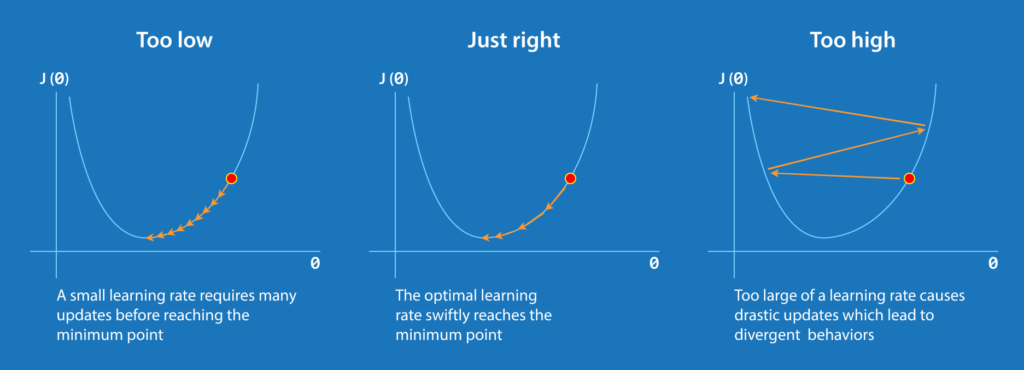
Derivative= - 2 (1.4-(0+0.64 \* 0.5))+

-2 (1.9-(0+0.64 \* 2.3))+

-2 (3.2-(0+0.64 \* 2.9))

= -5.7

Gradient descent subtracts the step size from the current value of intercept to get the new value of intercept. This step size is calculated by multiplying the derivative which is -5.7 here to a small number called the learning rate. Usually, we take the value of the learning rate to be 0.1, 0.01 or 0.001. The value of the step should not be too big as it can skip the minimum point and thus the optimisation can fail. It is a hyper-parameter and you need to experiment with its values.

[](data:image/svg+xml,%3Csvg%20xmlns=)

In this case, let us take the learning rate 0.1, then the step size is equal to

Step size=-5.7\*0.1

New intercept = old intercept-step size

= 0-(-0.57)=0.57

Let us now put the new intercept in the derivative function

d sum of squared error /d(intercept)= -2 (1.4-(0.57+0.64 \* 0.5))+

-2 (1.9-(0.57+0.64 \* 2.3))+

-2 (3.2-(0.57+0.64 \* 2.9))

= -2.3

Now calculate the next step size

Step size=-2.3\*0.1

New intercept = old intercept-step size

= 0.57-(-0.23)=0.8

Again let us now put the new intercept in the derivative function

d sum of squared error /d(intercept)= - 2 (1.4-(0.8+0.64 \* 0.5))+

-2 (1.9-(0.8+0.64 \* 2.3))+

-2 (3.2-(0.8+0.64 \* 2.9))

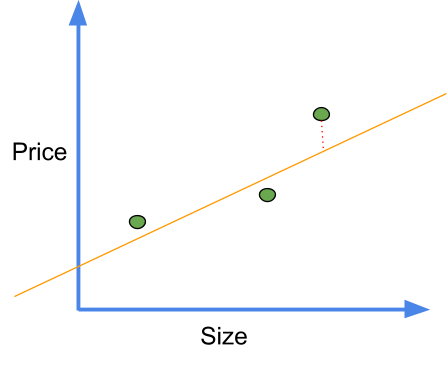
= -0.9

Step size= -0.9\*0.1

New intercept= old intercept-step size

= 0.8-(-0.09)=0.89

You might have noticed that the value of the step is high when the optimal solution is far away and this value is less as we approached an optimal solution. Thus we can say that gradient descent takes a bigger step when away from the solution and takes small steps when nearer to an optimal solution. This is the reason why gradient descent is efficient and fast.

[](data:image/svg+xml,%3Csvg%20xmlns=)

Now as we can see the line with intercept 0.89 is a much better fit. But is this our optimal solution? No, we continue to find new intercept values until the value of step tends to zero(less than 0.001) or even in some cases we predefine the number of steps that are to be taken. In practice, this number can go to 1000 or even greater.

**Optimising Linear Regression**

Now let us come to the real problem and see how gradient descent optimises slope and intercept simultaneously. As before we take the derivatives but this time of this equation

f(intercept) = (1.4-(intercept+ slope \* 0.5))^2+

(1.9-(intercept+slope \* 2.3))^2+

(3.2-(intercept+slope \* 2.9))^2

Here we again use the chain rule, first as before we find the derivative of D with respect to intercept keeping slope as constant

Derivative w.r.t intercept = -2 (1.4-(intercept+slope \* 0.5))+

-2 (1.9-(intercept+slope \* 2.3))+

-2 (3.2-(intercept+slope \* 2.9))

Now we find derivative of D with respect to slope and consider intercept as constant

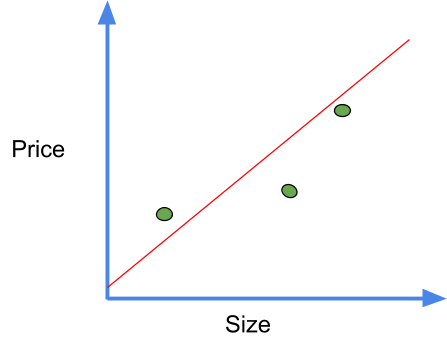
Derivative w.r.t slope= -2(0.5) (1.4-(intercept+slope \* 0.5))+

-2(2.3) (1.9-(intercept+slope \* 2.3))+

-2(2.9)(3.2-(intercept+slope \* 2.9))

When we have two or more derivatives of the same function, they are called gradients. We use these gradients to descend down the cost function. Thus the algorithm is called gradient descent. Note here the cost function we have been using so far is the sum of the square residuals.

As before we initialise intercept and slope randomly as zero and one. Now putting these values in the above gradients.

[](data:image/svg+xml,%3Csvg%20xmlns=)

the plot when intercept=0 and slope=1

Derivative w.r.t intercept= -2 (1.4-(0+1 \* 0.5))+

-2 (1.9-(0+1 \* 2.3))+

-2 (3.2-(0+1 \* 2.9))

= -1.6

We take a different learning rate here

Step size= -1.6\*0.01=-0.016

New intercept=0-(-0.016)=0.016

d/d(slope)=- 2(0.5) (1.4-(0+1 \* 0.5))+

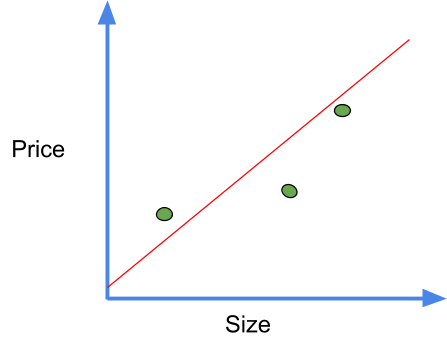
-2(2.3) (1.9-(0+1 \* 2.3))+

-2 (2.9)(3.2-(0+1 \* 2.9))

=-0.8

Step size= -0.8\*0.01=-0.008

New slope=1-(-0.008)=1.008

[](data:image/svg+xml,%3Csvg%20xmlns=)

This is definitely a better fit than random initialisation. Repeating this process until we get step size near zero for both slope and intercept gives us an optimal solution and best fit line.

If we have more than one parameter, such as the number of rooms, the process remains the same but the number of derivatives increases. Also here we used the sum of squared residuals as loss function, but we can use any other loss function as well such as least squares.

To briefly summarise the process, here are some points

1. Take the gradient of the loss function or in simpler words, take the derivative of the loss function for each parameter in it.
2. Randomly select the initialisation values.
3. Substitute these parameter values in the gradient
4. Calculate step size by using appropriate learning rate.
5. Calculate new parameters
6. Repeat from step 3 until an optimal solution is obtained.

**PCA(Principle Component Analysis) :**

Principal Component Analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data much easier and faster for machine learning algorithms without extraneous variables to process.

So to sum up, the idea of PCA is simple — reduce the number of variables of a data set, while preserving as much information as possible

Principal component analysis can be broken down into five steps:

1. Standardize the range of continuous initial variables
2. Compute the covariance matrix to identify correlations
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components
4. Create a feature vector to decide which principal components to keep
5. Recast the data along the principal components axes

## Step by Step Explanation of PCA

### STEP 1: STANDARDIZATION

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

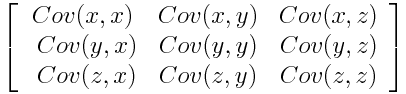
Principal Component Analysis Standardization

Once the standardization is done, all the variables will be transformed to the same scale.

### STEP 2: COVARIANCE MATRIX COMPUTATION

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a p × psymmetric matrix (where p is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables x, y, and z, the covariance matrix is a 3×3 matrix of this from:

Covariance Matrix for 3-Dimensional Data

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

**What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?**

It’s actually the sign of the covariance that matters :

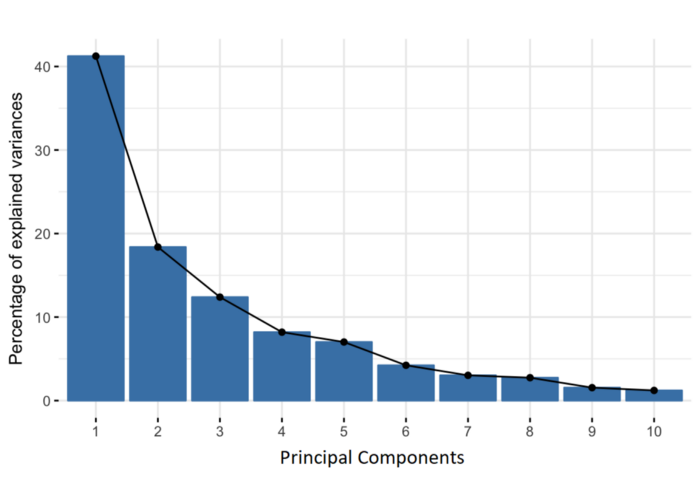
* if positive then : the two variables increase or decrease together (correlated)
* if negative then : One increases when the other decreases (Inversely correlated)

Now, that we know that the covariance matrix is not more than a table that summaries the correlations between all the possible pairs of variables, let’s move to the next step.

### STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the **principal components** of the data. Before getting to the explanation of these concepts, let’s first understand what do we mean by principal components.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.

Percentage of Variance (Information) for each by PC

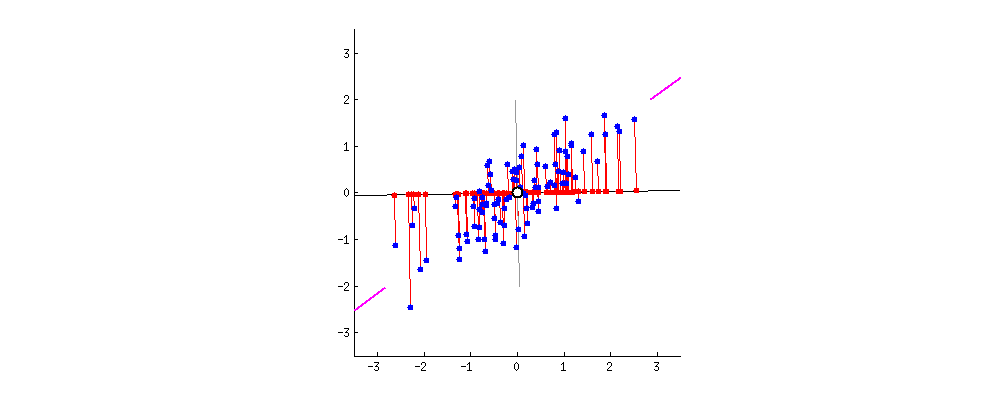
Organizing information in principal components this way, will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

An important thing to realize here is that, the principal components are less interpretable and don’t have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance**, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more the information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

## **How PCA Constructs the Principal Components**

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the **largest possible variance** in the data set. For example, let’s assume that the scatter plot of our data set is as shown below, can we guess the first principal component ? Yes, it’s approximately the line that matches the purple marks because it goes through the origin and it’s the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it’s the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

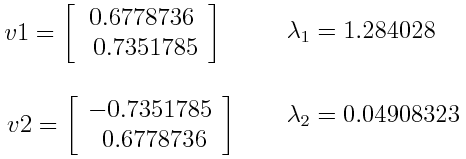
Now that we understood what we mean by principal components, let’s go back to eigenvectors and eigenvalues. What you firstly need to know about them is that they always come in pairs, so that every eigenvector has an eigenvalue. And their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

Without further ado, it is eigenvectors and eigenvalues who are behind all the magic explained above, because the eigenvectors of the Covariance matrix are actually thedirections of the axes where there is the most variance(most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the amount of variance carried in each Principal Component.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

**Example:**

Let’s suppose that our data set is 2-dimensional with 2 variables **x,y**and that the eigenvectors and eigenvalues of the covariance matrix are as follows:



If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is v1 and the one that corresponds to the second component (PC2) isv2.

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96% and 4% of the variance of the data.

### STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call Feature vector.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only **p** eigenvectors (components) out of **n**, the final data set will have only **p** dimensions.

**Example**:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors v1 and v2:

Principal Component Analysis eigen vectors

Or discard the eigenvector v2, which is the one of lesser significance, and form a feature vector with v1 only:

Principal Component Analysis eigen vectors 2

Discarding the eigenvector v2 will reduce dimensionality by 1, and will consequently cause a loss of information in the final data set. But given that v2 was carrying only 4% of the information, the loss will be therefore not important and we will still have 96% of the information that is carried by v1.

So, as we saw in the example, it’s up to you to choose whether to keep all the components or discard the ones of lesser significance, depending on what you are looking for. Because if you just want to describe your data in terms of new variables (principal components) that are uncorrelated without seeking to reduce dimensionality, leaving out lesser significant components is not needed.

### LAST STEP: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

Principal Component Analysis feature vector

## ****Applications of PCA****

Data is generated in many sectors, and there is a need to analyse data for the growth of any firm/company. PCA will help in reducing the dimensions of the data, thus making it easier to analyse. The applications of PCA are:

• Neuroscience – Neuroscientists use PCA to identify any neuron or to map the brain structure during phase transitions.

• Finance – PCA is used in the finance sector for reducing the dimensionality of data to create fixed income portfolios. Many other facets of the finance sector involve PCA like forecasting returns, making asset allocation algorithms or equity algorithms, etc.

• Image Technology – PCA is also used for image compression or digital image processing. Each image can be represented via a matrix by plotting the intensity values of each pixel, and then we can apply PCA on it.

• [Facial Recognition](https://www.upgrad.com/blog/face-detection-project-in-python/) – PCA in facial recognition leads to the creation of eigenfaces which makes facial recognition more accurate.

• Medical – PCA is used on a lot of medical data to find the correlation among different variables. For example, doctors use PCA to show the correlation between cholesterol & low-density lipoprotein.

• Security – Anomalies can be found easily using PCA. It is used to identify cyber/computer attacks and visualise them with the help of PCA.

## ****Takeaway Points****

PCA can also lead to low model performance after applying it if the original dataset has a weak correlation or no correlation. The variables need to be related to one other to apply PCA perfectly. PCA provides us with a combination of features, and individual feature importance from the original dataset is eradicated. The principal axes with the most variance are the ideal principal components.

## Can PCA be used on all data?

Yes. Principal Component Analysis (PCA) is a data analysis technique that provides a way of looking at and understanding data which is very high dimensional. In other words, PCA can be applied to data that has a large number of variables. There is a common misconception that PCA can only be used on data that is in a certain form. For example, many people think PCA is only useful on variables that are numerical. This is not the case. In fact, PCA can be used on variables of all types. For example, PCA can be applied to categorical variables, ordinal variables, and so on.

## What are the limitations of Principal Component Analysis?

PCA is a great tool to analyze your data and extract two or three most important factors. It is great to spot the outliers and the trend. But, it has some limitations like: It is not suitable for small data sets (Generally, data set should have more than 30 rows). It does not find the important factors but selects them based on the values. So, it is difficult to find the important factors. It does not have a strong mathematical structure behind it. It is difficult to compare the data with PCA. It cannot find any non-linear relationships.

## What are the advantages of principal component analysis?

Principal component analysis (PCA) is a statistical method used to transform a large number of possibly correlated variables into a much smaller number of uncorrelated variables referred to as principal components. PCA can be used as a data reduction technique as it allows us to find the most important variables that are needed to describe a dataset. PCA can also be used to reduce the dimensionality of the data space in order to get insight on the inner structure of the data. This is helpful when dealing with large datasets.

**Regression :**

## What is Regression Analysis?

Regression analysis is a form of predictive modelling technique which investigates the relationship between a **dependent**(target) and **independent variable (s)** (predictor). This technique is used for forecasting, time series modelling and finding the [causal effect relationship](https://www.analyticsvidhya.com/blog/2015/06/establish-causality-events/) between the variables. For example, relationship between rash driving and number of road accidents by a driver is best studied through regression.

Why do we use Regression Analysis?

As mentioned above, regression analysis estimates the relationship between two or more variables. Let’s understand this with an easy example:

Let’s say, you want to estimate growth in sales of a company based on current economic conditions. You have the recent company data which indicates that the growth in sales is around two and a half times the growth in the economy. Using this insight, we can predict future sales of the company based on current & past information.

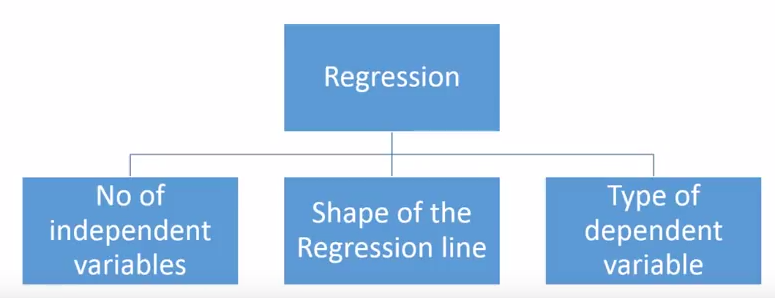
There are multiple benefits of using regression analysis. They are as follows:

1. It indicates the **significant relationships** between dependent variable and independent variable.
2. It indicates the **strength of impact** of multiple independent variables on a dependent variable.

Regression analysis also allows us to compare the effects of variables measured on different scales, such as the effect of price changes and the number of promotional activities. These benefits help market researchers / data analysts / data scientists to eliminate and evaluate the best set of variables to be used for building predictive models.

## How many types of regression techniques do we have?

There are various kinds of regression techniques available to make predictions. These techniques are mostly driven by three metrics (number of independent variables, type of dependent variables and shape of regression line). We’ll discuss them in detail in the following sections.

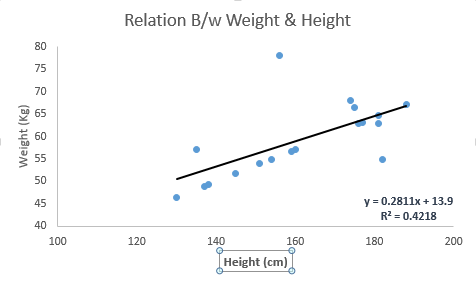
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Regression_Type.png)

## 1. Linear Regression

It is one of the most widely known modeling technique. Linear regression is usually among the first few topics which people pick while learning predictive modeling. In this technique, the dependent variable is continuous, independent variable(s) can be [continuous or discrete](https://en.wikipedia.org/wiki/Continuous_and_discrete_variables), and nature of regression line is linear.

Linear Regression establishes a relationship between **dependent variable (Y)** and one or more **independent variables (X)** using a **best fit straight line** (also known as regression line).

It is represented by an equation **Y=a+b\*X + e**, where a is intercept, b is slope of the line and e is error term. This equation can be used to predict the value of target variable based on given predictor variable(s).

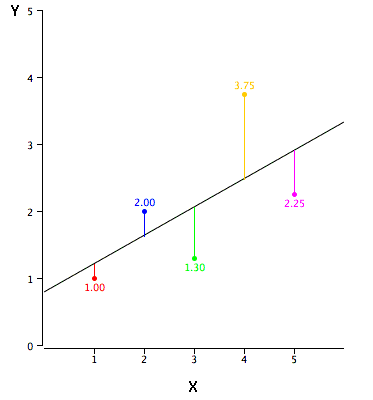
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Linear_Regression1.png)

The difference between simple linear regression and multiple linear regression is that, multiple linear regression has (>1) independent variables, whereas simple linear regression has only 1 independent variable.  Now, the question is “How do we obtain best fit line?”.

#### **How to obtain best fit line (Value of a and b)?**

This task can be easily accomplished by Least Square Method. It is the most common method used for fitting a regression line. It calculates the best-fit line for the observed data by minimizing the sum of the squares of the vertical deviations from each data point to the line. Because the deviations are first squared, when added, there is no cancelling out between positive and negative values.

[least square, regression line](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Least_Square.png)

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/reg_error.gif)

#### **Important Points:**

* There must be **linear relationship** between independent and dependent variables
* Multiple regression suffers from **multicollinearity, autocorrelation, heteroskedasticity**.
* Linear Regression is very sensitive to **Outliers**. It can terribly affect the regression line and eventually the forecasted values.
* Multicollinearity can increase the variance of the coefficient estimates and make the estimates very sensitive to minor changes in the model. The result is that the coefficient estimates are unstable
* In case of multiple independent variables, we can go with **forward selection**, **backward elimination** and **step wise approach** for selection of most significant independent variables

## 2. Logistic Regression

Logistic regression is used to find the probability of event=Success and event=Failure. We should use logistic regression when the dependent variable is binary (0/ 1, True/ False, Yes/ No) in nature. Here the value of Y ranges from 0 to 1 and it can represented by following equation.

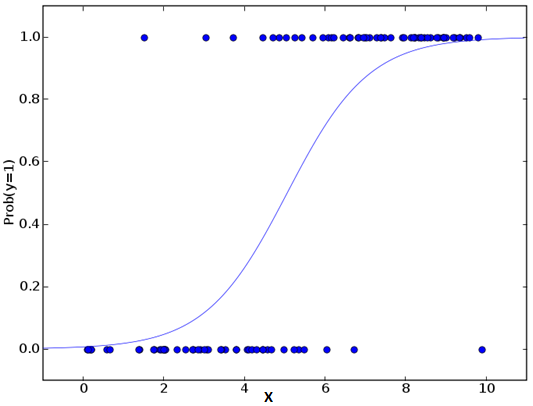
odds= p/ (1-p) = probability of event occurrence / probability of not event occurrence

ln(odds) = ln(p/(1-p))

logit(p) = ln(p/(1-p)) = b0+b1X1+b2X2+b3X3....+bkXk

Above, p is the probability of presence of the characteristic of interest. A question that you should ask here is “why have we used log in the equation?”.

Since we are working here with a binomial distribution (dependent variable), we need to choose a link function which is best suited for this distribution. And, it is [**logit**](https://en.wikipedia.org/wiki/Logistic_function) function. In the equation above, the parameters are chosen to maximize the likelihood of observing the sample values rather than minimizing the sum of squared errors (like in ordinary regression).

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Logistic_Regression.png)

#### **Important Points:**

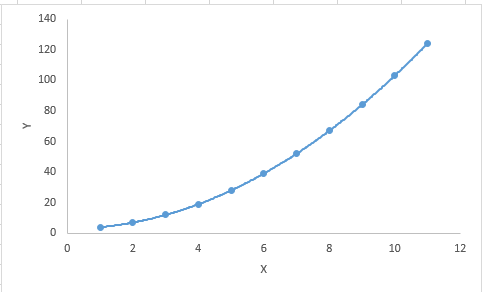
* Logistic regression is widely used for **classification problems**
* Logistic regression doesn’t require linear relationship between dependent and independent variables.  It can handle various types of relationships because it applies a non-linear log transformation to the predicted odds ratio
* To avoid over fitting and under fitting, we should include all significant variables. A good approach to ensure this practice is to use a step wise method to estimate the logistic regression
* It requires **large sample sizes** because maximum likelihood estimates are less powerful at low sample sizes than ordinary least square
* The independent variables should not be correlated with each other i.e. **no multi collinearity**.  However, we have the options to include interaction effects of categorical variables in the analysis and in the model.
* If the values of dependent variable is ordinal, then it is called as **Ordinal logistic regression**
* If dependent variable is multi class then it is known as **Multinomial Logistic regression**.

## 3. Polynomial Regression

A regression equation is a polynomial regression equation if the power of independent variable is more than 1. The equation below represents a polynomial equation:

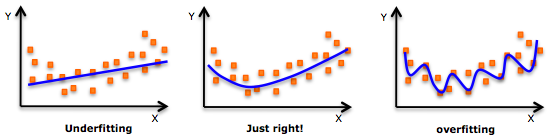
y=a+b\*x^2

In this regression technique, the best fit line is not a straight line. It is rather a curve that fits into the data points.

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Polynomial.png)

#### **Important Points:**

* While there might be a temptation to fit a higher degree polynomial to get lower error, this can result in over-fitting. Always plot the relationships to see the fit and focus on making sure that the curve fits the nature of the problem. Here is an example of how plotting can help:

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/02/underfitting-overfitting.png)

* Especially look out for curve towards the ends and see whether those shapes and trends make sense. Higher polynomials can end up producing wierd results on extrapolation.

## 4. Stepwise Regression

This form of regression is used when we deal with multiple independent variables. In this technique, the selection of independent variables is done with the help of an automatic process, which involves no human intervention.

This feat is achieved by observing statistical values like R-square, t-stats and AIC metric to discern significant variables. Stepwise regression basically fits the regression model by adding/dropping co-variates one at a time based on a specified criterion. Some of the most commonly used Stepwise regression methods are listed below:

* Standard stepwise regression does two things. It adds and removes predictors as needed for each step.
* Forward selection starts with most significant predictor in the model and adds variable for each step.
* Backward elimination starts with all predictors in the model and removes the least significant variable for each step.

The aim of this modeling technique is to maximize the prediction power with minimum number of predictor variables. It is one of the method to handle[higher dimensionality](https://www.analyticsvidhya.com/blog/2015/07/dimension-reduction-methods/) of data set.

## 5. Ridge Regression

Ridge Regression is a technique used when the data suffers from multicollinearity (independent variables are highly correlated). In multicollinearity, even though the least squares estimates (OLS) are unbiased, their variances are large which deviates the observed value far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

Above, we saw the equation for linear regression. Remember? It can be represented as:

y=a+ b\*x

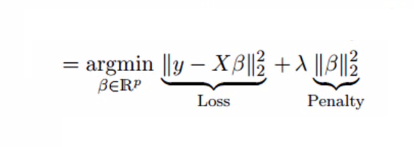
This equation also has an error term. The complete equation becomes:

y=a+b\*x+e (error term),  [error term is the value needed to correct for a prediction error between the observed and predicted value]

=> y=a+y= a+ b1x1+ b2x2+....+e, for multiple independent variables.

In a linear equation, prediction errors can be decomposed into two sub components. First is due to the **biased** and second is due to the **variance**. Prediction error can occur due to any one of these two or both components. Here, we’ll discuss about the error caused due to variance.

Ridge regression solves the multicollinearity problem through [shrinkage parameter](https://en.wikipedia.org/wiki/Shrinkage_estimator) λ (lambda). Look at the equation below.

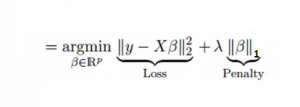
[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Ridge2.png)

In this equation, we have two components. First one is least square term and other one is lambda of the summation of β2 (beta- square) where β is the coefficient. This is added to least square term in order to shrink the parameter to have a very low variance.

#### **Important Points:**

* The assumptions of this regression is same as least squared regression except normality is not to be assumed
* Ridge regression shrinks the value of coefficients but doesn’t reaches zero, which suggests no feature selection feature
* This is a regularization method and uses [l2 regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics)).

## 6. Lasso Regression

[](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Lasso.png)

Similar to Ridge Regression, Lasso (Least Absolute Shrinkage and Selection Operator) also penalizes the absolute size of the regression coefficients. In addition, it is capable of reducing the variability and improving the accuracy of linear regression models.  Look at the equation below: Lasso regression differs from ridge regression in a way that it uses absolute values in the penalty function, instead of squares. This leads to penalizing (or equivalently constraining the sum of the absolute values of the estimates) values which causes some of the parameter estimates to turn out exactly zero. Larger the penalty applied, further the estimates get shrunk towards absolute zero. This results to variable selection out of given n variables.

#### **Important Points:**

* The assumptions of lasso regression is same as least squared regression except normality is not to be assumed
* Lasso Regression shrinks coefficients to zero (exactly zero), which certainly helps in feature selection
* Lasso is a regularization method and uses [l1 regularization](https://en.wikipedia.org/wiki/Regularization_(mathematics))
* If group of predictors are highly correlated, lasso picks only one of them and shrinks the others to zero

## 7. ElasticNet Regression

ElasticNet is hybrid of Lasso and Ridge Regression techniques. It is trained with L1 and L2 prior as regularizer. Elastic-net is useful when there are multiple features which are correlated. Lasso is likely to pick one of these at random, while elastic-net is likely to pick both.

[elastic net regression](https://www.analyticsvidhya.com/wp-content/uploads/2015/08/Elastic_Net.png)

A practical advantage of trading-off between Lasso and Ridge is that, it allows Elastic-Net to inherit some of Ridge’s stability under rotation.

#### **Important Points:**

* It encourages group effect in case of highly correlated variables
* There are no limitations on the number of selected variables
* It can suffer with double shrinkage

## How to select the right regression model?

Life is usually simple, when you know only one or two techniques. One of the training institutes I know of tells their students – if the outcome is continuous – apply linear regression. If it is binary – use logistic regression! However, higher the number of options available at our disposal, more difficult it becomes to choose the right one. A similar case happens with regression models.

Within multiple types of regression models, it is important to choose the best suited technique based on type of independent and dependent variables, dimensionality in the data and other essential characteristics of the data. Below are the key factors that you should practice to select the right regression model:

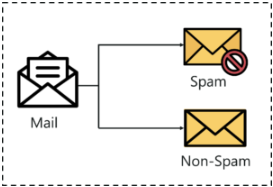
1. Data exploration is an inevitable part of building predictive model. It should be you first step before selecting the right model like identify the relationship and impact of variables
2. To compare the goodness of fit for different models, we can analyse different metrics like statistical significance of parameters, R-square, Adjusted r-square, AIC, BIC and error term. Another one is the [Mallow’s Cp](http://support.minitab.com/en-us/minitab/17/topic-library/modeling-statistics/regression-and-correlation/goodness-of-fit-statistics/what-is-mallows-cp/) criterion. This essentially checks for possible bias in your model, by comparing the model with all possible submodels (or a careful selection of them).
3. Cross-validation is the best way to evaluate models used for prediction. Here you divide your data set into two group (train and validate). A simple mean squared difference between the observed and predicted values give you a measure for the prediction accuracy.
4. If your data set has multiple confounding variables, you should not choose automatic model selection method because you do not want to put these in a model at the same time.
5. It’ll also depend on your objective. It can occur that a less powerful model is easy to implement as compared to a highly statistically significant model.
6. Regression regularization methods(Lasso, Ridge and ElasticNet) works well in case of high dimensionality and multicollinearity among the variables in the data set.

**Classification:-**

## ****What is Classification In Machine Learning****

Classification is a process of categorizing a given set of data into classes, It can be performed on both structured or unstructured data. The process starts with predicting the class of given data points. The classes are often referred to as target, label or categories.

The classification predictive modeling is the task of approximating the mapping function from input variables to discrete output variables. The main goal is to identify which class/category the new data will fall into.

****

**Types Of Learners In Classification**

* **Lazy Learners** – Lazy learners simply store the training data and wait until a testing data appears. The classification is done using the most related data in the stored training data. They have more predicting time compared to eager learners. Eg – k-nearest neighbor, case-based reasoning.
* **Eager Learners** – Eager learners construct a classification model based on the given training data before getting data for predictions. It must be able to commit to a single hypothesis that will work for the entire space. Due to this, they take a lot of time in training and less time for a prediction. Eg – Decision Tree, Naive Bayes, Artificial Neural Networks.

**Different types of Classification Algorithm:**

**1.Logistic Regression:**

Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Unlike linear regression which outputs continuous number values, logistic regression transforms its output using the logistic sigmoid function to return a probability value which can then be mapped to two or more discrete classes.

### [Types of logistic regression](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#toc-entry-3)

* Binary (Pass/Fail)
* Multi (Cats, Dogs, Sheep)
* Ordinal (Low, Medium, High)

### [Sigmoid activation](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#toc-entry-5)

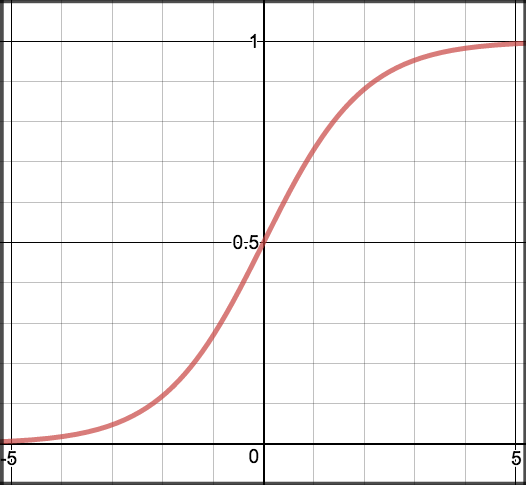
In order to map predicted values to probabilities, we use the [sigmoid](https://ml-cheatsheet.readthedocs.io/en/latest/activation_functions.html#activation-sigmoid) function. The function maps any real value into another value between 0 and 1. In machine learning, we use sigmoid to map predictions to probabilities.

**Math**

****

**Note**

* s(z)s(z) = output between 0 and 1 (probability estimate)
* zz = input to the function (your algorithm’s prediction e.g. mx + b)
* ee = base of natural log

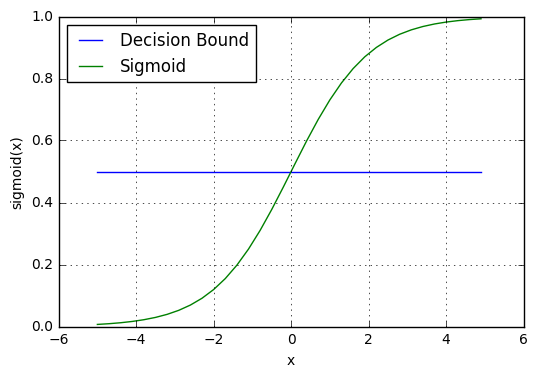


### [Decision boundary](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#toc-entry-6)

Our current prediction function returns a probability score between 0 and 1. In order to map this to a discrete class (true/false, cat/dog), we select a threshold value or tipping point above which we will classify values into class 1 and below which we classify values into class 2.

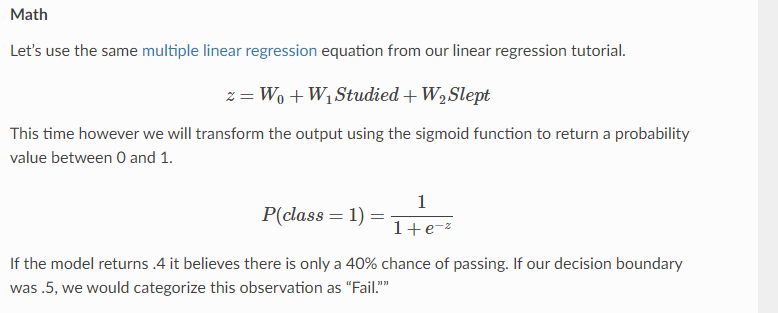
p≥0.5,class=1p<0.5,class=0p≥0.5,class=1p<0.5,class=0

For example, if our threshold was .5 and our prediction function returned .7, we would classify this observation as positive. If our prediction was .2 we would classify the observation as negative. For logistic regression with multiple classes we could select the class with the highest predicted probability.



### [Making predictions](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#toc-entry-7)

Using our knowledge of sigmoid functions and decision boundaries, we can now write a prediction function. A prediction function in logistic regression returns the probability of our observation being positive, True, or “Yes”. We call this class 1 and its notation is P(class=1). As the probability gets closer to 1, our model is more confident that the observation is in class 1.

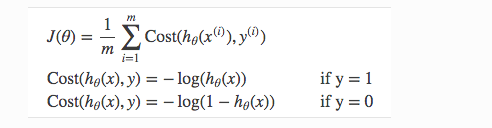
****

### [Cost function](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#toc-entry-8)

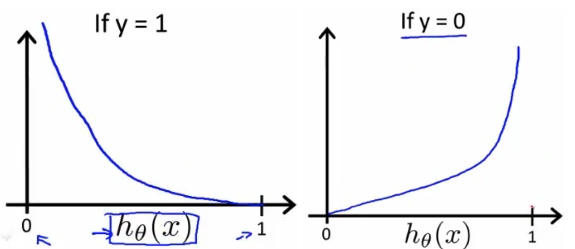
Unfortunately we can’t (or at least shouldn’t) use the same cost function [MSE (L2)](https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#mse) as we did for linear regression. Why? There is a great math explanation in chapter 3 of Michael Neilson’s deep learning book [[5]](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#footnote-5), but for now I’ll simply say it’s because our prediction function is non-linear (due to sigmoid transform). Squaring this prediction as we do in MSE results in a non-convex function with many local minimums. If our cost function has many local minimums, gradient descent may not find the optimal global minimum.

**Math**

Instead of Mean Squared Error, we use a cost function called [Cross-Entropy](https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#loss-cross-entropy), also known as Log Loss. Cross-entropy loss can be divided into two separate cost functions: one for y=1y=1 and one for y=0y=0.



The benefits of taking the logarithm reveal themselves when you look at the cost function graphs for y=1 and y=0. These smooth monotonic functions [[7]](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#footnote-7) (always increasing or always decreasing) make it easy to calculate the gradient and minimize cost. Image from Andrew Ng’s slides on logistic regression [[1]](https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html#footnote-1).



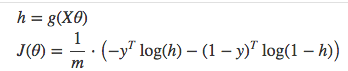
The key thing to note is the cost function penalizes confident and wrong predictions more than it rewards confident and right predictions! The corollary is increasing prediction accuracy (closer to 0 or 1) has diminishing returns on reducing cost due to the logistic nature of our cost function.

**Above functions compressed into one**



Multiplying by yy and (1−y)(1−y) in the above equation is a sneaky trick that let’s us use the same equation to solve for both y=1 and y=0 cases. If y=0, the first side cancels out. If y=1, the second side cancels out. In both cases we only perform the operation we need to perform.

**Vectorized cost function**



**Prepare Data for Logistic Regression**

The assumptions made by logistic regression about the distribution and relationships in your data are much the same as the assumptions made in linear regression.

Much study has gone into defining these assumptions and precise probabilistic and statistical language is used. My advice is to use these as guidelines or rules of thumb and experiment with different data preparation schemes.

Ultimately in predictive modeling machine learning projects you are laser focused on making accurate predictions rather than interpreting the results. As such, you can break some assumptions as long as the model is robust and performs well.

* **Binary Output Variable**: This might be obvious as we have already mentioned it, but logistic regression is intended for binary (two-class) classification problems. It will predict the probability of an instance belonging to the default class, which can be snapped into a 0 or 1 classification.
* **Remove Noise**: Logistic regression assumes no error in the output variable (y), consider removing outliers and possibly misclassified instances from your training data.
* **Gaussian Distribution**: Logistic regression is a linear algorithm (with a non-linear transform on output). It does assume a linear relationship between the input variables with the output. Data transforms of your input variables that better expose this linear relationship can result in a more accurate model. For example, you can use log, root, Box-Cox and other univariate transforms to better expose this relationship.
* **Remove Correlated Inputs**: Like linear regression, the model can overfit if you have multiple highly-correlated inputs. Consider calculating the pairwise correlations between all inputs and removing highly correlated inputs.
* **Fail to Converge**: It is possible for the expected likelihood estimation process that learns the coefficients to fail to converge. This can happen if there are many highly correlated inputs in your data or the data is very sparse (e.g. lots of zeros in your input data).

2.Decision Tree:

***A Decision Tree is a supervised Machine learning algorithm. It is used in both classification and regression algorithms***. The decision tree is like a tree with nodes. The branches depend on a number of factors. It splits data into branches like these till it achieves a threshold value. A decision tree consists of the root nodes, children nodes, and leaf nodes.

Every decision tree consists following list of elements:

**a) Nodes:** It is The point where the tree splits according to the value of some attribute/feature of the dataset

**b) Edges:** It directs the outcome of a split to the next node we can see in the figure above that there are nodes for features like outlook, humidity and windy. There is an edge for each potential value of each of those attributes/features.

**c) Root:** This is the node where the first split takes place

**d) Leaves:** These are the terminal nodes that predict the outcome of the decision tree

## How to Build Decision Trees from Scratch?

While building a Decision tree, the main thing is to select the best attribute from the total features list of the dataset for the root node as well as for sub-nodes. The selection of best attributes is being achieved with the help of a technique known as the Attribute selection measure (ASM).

With the help of ASM, we can easily select the best features for the respective nodes of the decision tree.

There are two techniques for ASM:

**a) Information Gain**

**b) Gini Index**

## How to Build Decision Trees from Scratch?

While building a Decision tree, the main thing is to select the best attribute from the total features list of the dataset for the root node as well as for sub-nodes. The selection of best attributes is being achieved with the help of a technique known as the Attribute selection measure (ASM).

With the help of ASM, we can easily select the best features for the respective nodes of the decision tree.

There are two techniques for ASM:

**a) Information Gain**

**b) Gini Index**

### a) Information Gain:

1Information gain is the measurement of changes in entropy value after the splitting/segmentation of the dataset based on an attribute.

2 It tells how much information a feature/attribute provides us.

3 Following the value of the information gain, splitting of the node and decision tree building is being done.

4 decision tree always tries to maximize the value of the information gain, and a node/attribute having the highest value of the information gain is being split first. Information gain can be calculated using the below formula:

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

Entropy: Entropy signifies the randomness in the dataset. It is being defined as a metric to measure impurity. 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.

### b) Gini Index:

Gini index is also being defined as a measure of impurity/ purity used while creating a decision tree in the CART(known as Classification and Regression Tree) algorithm.

### How Does the Decision Tree Algorithm works?

The basic idea behind any decision tree algorithm is as follows:

1. Select the best Feature using Attribute Selection Measures(ASM) to split the records.

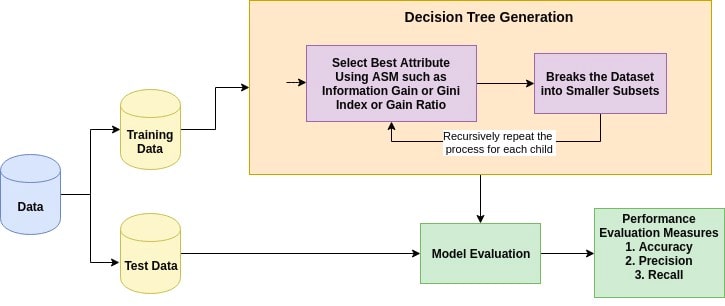
2. Make that attribute/feature a decision node and break the dataset into smaller subsets.

3 Start the tree-building process by repeating this process recursively for each child until one of the following condition is being achieved :

a) All tuples belonging to the same attribute value.

b) There are no more of the attributes remaining.

c ) There are no more instances remaining.

[source](https://www.datacamp.com/community/tutorials/decision-tree-classification-python)

An attribute having a low Gini index value should be preferred in contrast to the high Gini index value.

It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.

Gini index can be calculated using the below formula:

**Gini Index= 1- ∑jPj2**

Where pj stands for the probability

### How Does the Decision Tree Algorithm works?

The basic idea behind any decision tree algorithm is as follows:

1. Select the best Feature using Attribute Selection Measures(ASM) to split the records.

2. Make that attribute/feature a decision node and break the dataset into smaller subsets.

3 Start the tree-building process by repeating this process recursively for each child until one of the following condition is being achieved :

a) All tuples belonging to the same attribute value.

b) There are no more of the attributes remaining.

c ) There are no more instances remaining.

**Advantages & Disadvantages of Decision Trees**

### **Advantages**

* Decision trees generate understandable rules.
* Decision trees perform classification without requiring much computation.
* Decision trees are capable of handling both continuous and categorical variables.
* Decision trees provide a clear indication of which fields are most important for prediction or classification.

### **Disadvantages**

* Decision trees are less appropriate for estimation tasks where the goal is to predict the value of a continuous attribute.
* Decision trees are prone to errors in classification problems with many class and a relatively small number of training examples.
* Decision trees can be computationally expensive to train. The process of growing a decision tree is computationally expensive. At each node, each candidate splitting field must be sorted before its best split can be found. In some algorithms, combinations of fields are used and a search must be made for optimal combining weights. [***Pruning algorithms***](https://www.edureka.co/blog/implementation-of-decision-tree/) can also be expensive since many candidate sub-trees must be formed and compared

Random Forest :-

## What is bagging?

Bagging, also known as bootstrap aggregation, is the ensemble learning method that is commonly used to reduce variance within a noisy dataset. In bagging, a random sample of data in a training set is selected with replacement—meaning that the individual data points can be chosen more than once. After several data samples are generated, these weak models are then trained independently, and depending on the type of task—regression or classification, for example—the average or majority of those predictions yield a more accurate estimate.

As a note, the random forest algorithm is considered an extension of the bagging method, using both bagging and feature randomness to create an uncorrelated forest of decision trees.

### **Ensemble learning**

Ensemble learning gives credence to the idea of the “wisdom of crowds,” which suggests that the decision-making of a larger group of people is typically better than that of an individual expert. Similarly, ensemble learning refers to a group (or ensemble) of base learners, or models, which work collectively to achieve a better final prediction. A single model, also known as a base or weak learner, may not perform well individually due to high variance or high bias. However, when weak learners are aggregated, they can form a strong learner, as their combination reduces bias or variance, yielding better model performance.

Ensemble methods are frequently illustrated using decision trees as this algorithm can be prone to overfitting (high variance and low bias) when it hasn’t been pruned and it can also lend itself to underfitting (low variance and high bias) when it’s very small, like a decision stump, which is a decision tree with one level. Remember, when an algorithm overfits or underfits to its training set, it cannot generalize well to new datasets, so ensemble methods are used to counteract this behavior to allow for generalization of the model to new datasets. While decision trees can exhibit high variance or high bias, it’s worth noting that it is not the only modeling technique that leverages ensemble learning to find the “sweet spot” within the bias-variance tradeoff.

### **Bagging vs. boosting**

Bagging and boosting are two main types of ensemble learning methods. As highlighted in this [study](https://www.d.umn.edu/~rmaclin/cs5751/notes/opitz-jair99.pdf) (PDF, 248 KB) (link resides outside IBM), the main difference between these learning methods is the way in which they are trained. In bagging, weak learners are trained in parallel, but in boosting, they learn sequentially. This means that a series of models are constructed and with each new model iteration, the weights of the misclassified data in the previous model are increased. This redistribution of weights helps the algorithm identify the parameters that it needs to focus on to improve its performance. AdaBoost, which stands for “adaptative boosting algorithm,” is one of the most popular boosting algorithms as it was one of the first of its kind. Other types of boosting algorithms include XGBoost, GradientBoost, and BrownBoost.

Another difference in which bagging and boosting differ are the scenarios in which they are used. For example, bagging methods are typically used on weak learners which exhibit high variance and low bias, whereas boosting methods are leveraged when low variance and high bias is observed.

## How bagging works

bagging algorithm has three basic steps:

1. **Bootstrapping:** Bagging leverages a bootstrapping sampling technique to create diverse samples. This resampling method generates different subsets of the training dataset by selecting data points at random and with replacement. This means that each time you select a data point from the training dataset, you are able to select the same instance multiple times. As a result, a value/instance repeated twice (or more) in a sample.
2. **Parallel training:** These bootstrap samples are then trained independently and in parallel with each other using weak or base learners.
3. **Aggregation:**Finally, depending on the task (i.e. regression or classification), an average or a majority of the predictions are taken to compute a more accurate estimate. In the case of regression, an average is taken of all the outputs predicted by the individual classifiers; this is known as soft voting. For classification problems, the class with the highest majority of votes is accepted; this is known as hard voting or majority voting.

## Benefits and challenges of bagging

There are a number of key advantages and challenges that the bagging method presents when used for classification or regression problems. The key benefits of bagging include:

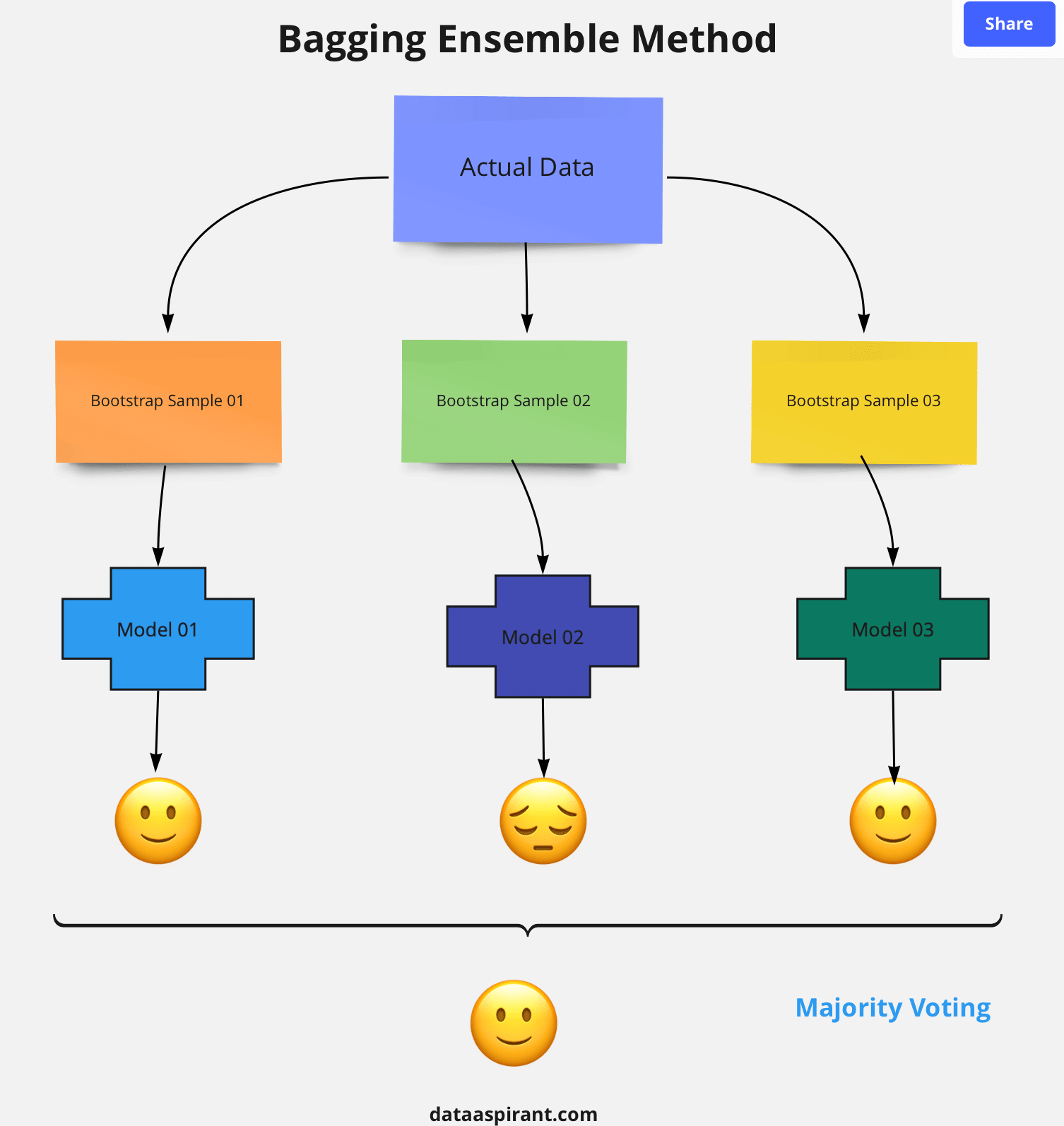
* **Ease of implementation:**Python libraries such as scikit-learn (also known as sklearn) make it easy to combine the predictions of base learners or estimators to improve model performance. Their [documentation](https://scikit-learn.org/stable/modules/ensemble.html) (link resides outside IBM) lays out the available modules that you can leverage in your model optimization.
* **Reduction of variance**: Bagging can reduce the variance within a learning algorithm. This is particularly helpful with high-dimensional data, where missing values can lead to higher variance, making it more prone to overfitting and preventing accurate generalization to new datasets.

The key challenges of bagging include:

* **Loss of interpretability**: It’s difficult to draw very precise business insights through bagging because due to the averaging involved across predictions. While the output is more precise than any individual data point, a more accurate or complete dataset could also yield more precision within a single classification or regression model.
* **Computationally expensive:** Bagging slows down and grows more intensive as the number of iterations increase. Thus, it’s not well-suited for real-time applications. Clustered systems or a large number of processing cores are ideal for quickly creating bagged ensembles on large test sets.
* **Less flexible:**As a technique, bagging works particularly well with algorithms that are less stable. One that are more stable or subject to high amounts of bias do not provide as much benefit as there’s less variation within the dataset of the model. As noted in the [Hands-On Guide to Machine Learning](https://bradleyboehmke.github.io/HOML/) (link resides outside of IBM), “bagging a linear regression model will effectively just return the original predictions for large enough b.”

The random forest algorithm is based on supervised learning. It can be used for both regression and classification problems. As the name suggests, Random Forest can be viewed as a collection of multiple decision trees algorithm with random sampling. This algorithm is made to eradicate the shortcomings of the Decision tree algorithm.

Random forest is a combination of Breiman’s “[bagging](https://en.wikipedia.org/wiki/Bootstrap_aggregating)” idea and a random selection of features. The idea is to make the prediction precise by taking the average or mode of the output of multiple decision trees. The greater the number of decision trees is considered, the more precise output will be.



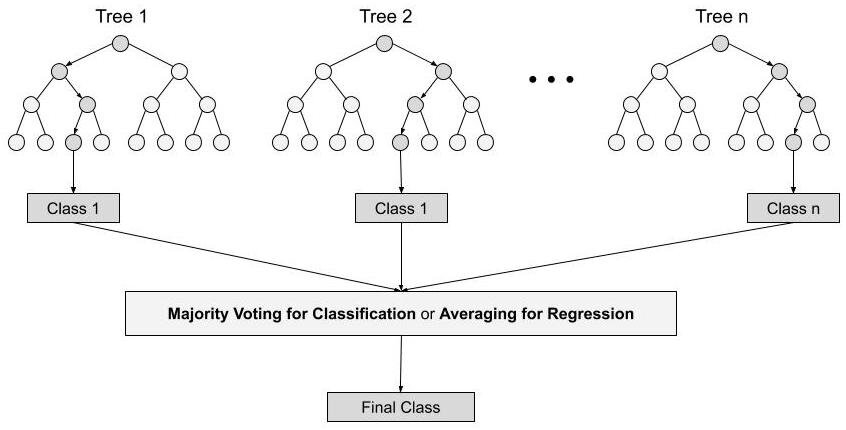
**Steps involved in random forest algorithm:**

Step 1: In Random forest n number of random records are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on ***Majority Voting or Averaging***for Classification and regression respectively.



### **Difference Between Decision Tree & Random Forest**

Random forest is a collection of decision trees; still, there are a lot of differences in their behavior.

|  |  |
| --- | --- |
| **Decision trees** | **Random Forest** |
| 1. Decision trees normally suffer from the problem of overfitting if it’s allowed to grow without any control. | 1. Random forests are created from subsets of data and the final output is based on average or majority ranking and hence the problem of overfitting is taken care of. |
| 2. A single decision tree is faster in computation. | 2. It is comparatively slower. |
| 3. When a data set with features is taken as input by a decision tree it will formulate some set of rules to do prediction. | 3. Random forest randomly selects observations, builds a decision tree and the average result is taken. It doesn’t use any set of formulas. |

Thus random forests are much more successful than decision trees only if the trees are diverse and acceptable.

### **Important Hyperparameters**

Hyperparameters are used in random forests to either enhance the performance and predictive power of models or to make the model faster.

*Following hyperparameters increases the predictive power:*

1. **n\_estimators**– number of trees the algorithm builds before averaging the predictions.

2. ***max\_features****–*maximum number of features random forest considers splitting a node.

*3.****mini\_sample\_leaf****–*determines the minimum number of leaves required to split an internal node.

*Following hyperparameters increases the speed:*

*1.****n\_jobs****–*it tells the engine how many processors it is allowed to use. If the value is 1, it can use only one processor but if the value is -1 there is no limit.

*2.****random\_state****–*controls randomness of the sample. The model will always produce the same results if it has a definite value of random state and if it has been given the same hyperparameters and the same training data.

*3.****oob\_score****– OOB* means out of the bag. It is a random forest cross-validation method. In this one-third of the sample is not used to train the data instead used to evaluate its performance. These samples are called out of bag samples.

Support Vector Machine ;-

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

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

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.



SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

**SVM can be of two types:**

* **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
* **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

## Hyperplane and Support Vectors in the SVM algorithm:

**Hyperplane:** There can be multiple lines/decision boundaries to segregate the classes in n-dimensional space, but we need to find out the best decision boundary that helps to classify the data points. This best boundary is known as the hyperplane of SVM.

The dimensions of the hyperplane depend on the features present in the dataset, which means if there are 2 features (as shown in image), then hyperplane will be a straight line. And if there are 3 features, then hyperplane will be a 2-dimension plane.

We always create a hyperplane that has a maximum margin, which means the maximum distance between the data points.

**Support Vectors:**

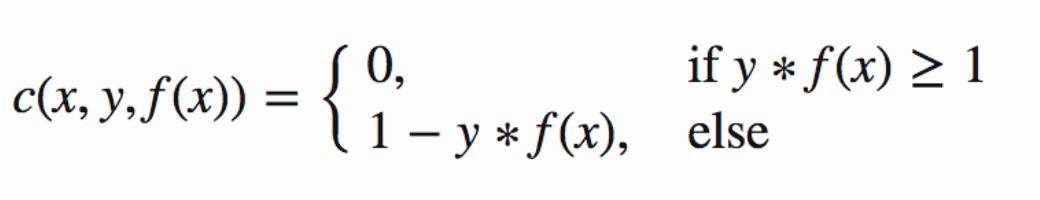
The data points or vectors that are the closest to the hyperplane and which affect the position of the hyperplane are termed as Support Vector. Since these vectors support the hyperplane, hence called a Support vector.

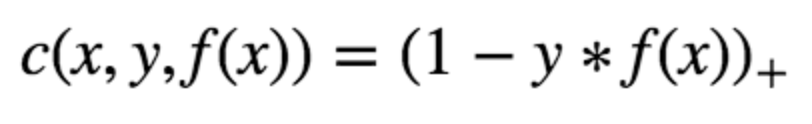
## Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value(0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

## Cost Function and Gradient Updates

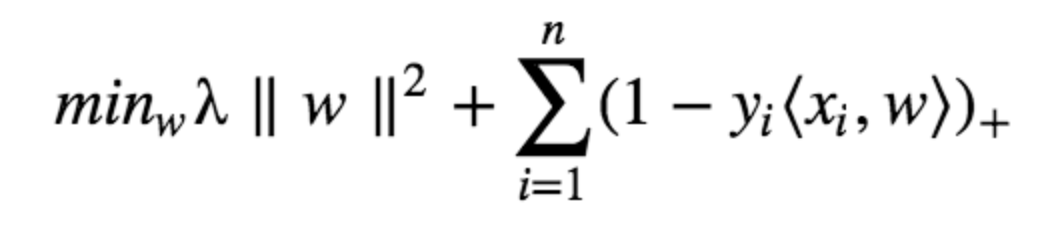
In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.





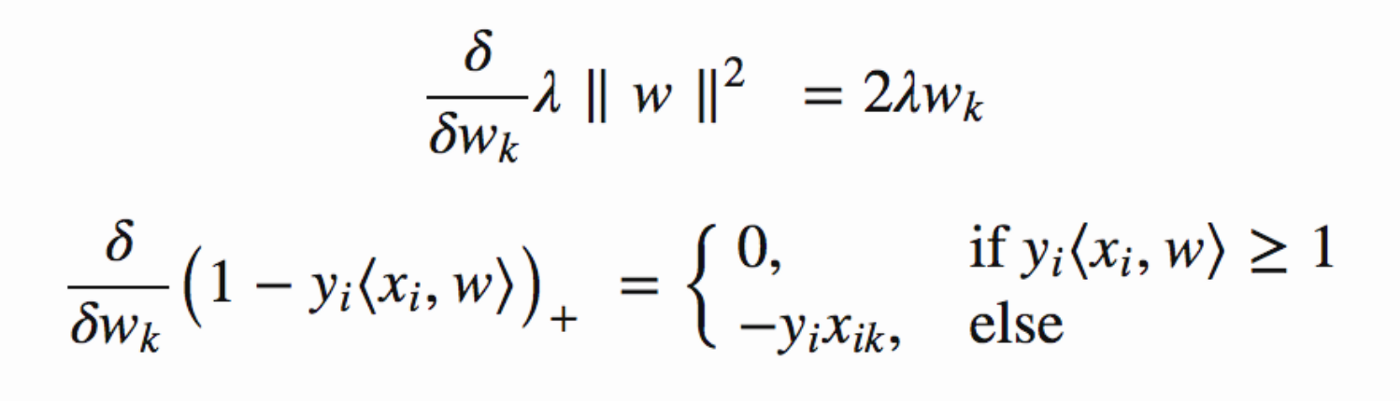
Hinge loss function (function on left can be represented as a function on the right)

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.



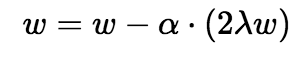
Loss function for SVM

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.



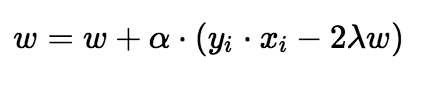
Gradients

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.



Gradient Update — No misclassification

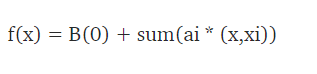
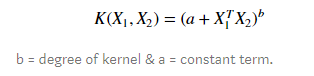
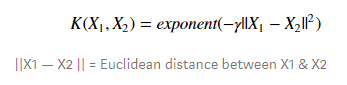
When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.



Gradient Update — Misclassification

## ****SVM Kernels****

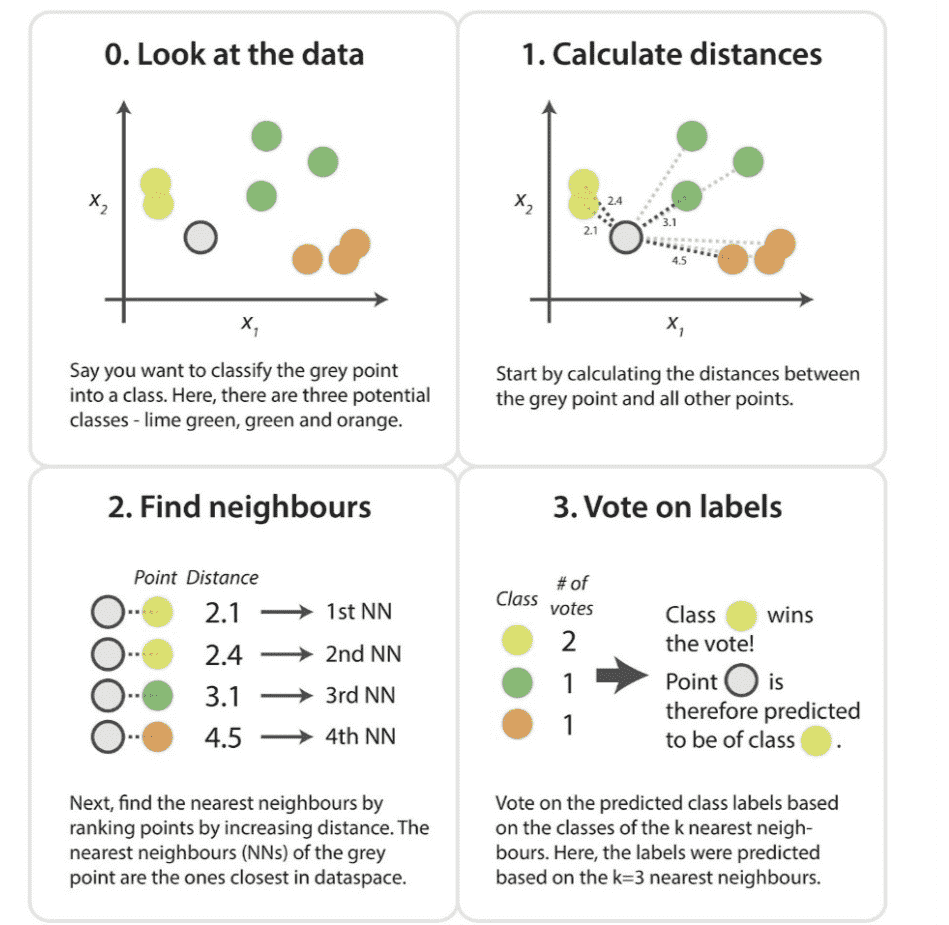
An SVM kernel basically adds more dimensions to a low dimensional space to make it easier to segregate the data. It converts the inseparable problem to separable problems by adding more dimensions using the kernel trick. A support vector machine is implemented in practice by a kernel. The kernel trick helps to make a more accurate classifier. Let us take a look at the different kernels in the Support vector machine.

* **Linear Kernel** – A linear kernel can be used as a normal dot product between any two given observations. The product between the two vectors is the sum of the multiplication of each pair of input values. Following is the linear kernel equation.  
  
* **Polynomial Kernel** – It is a rather generalized form of the linear kernel. It can distinguish curved or nonlinear input space. Following is the polynomial kernel equation.  
  
* **Radial Basis Function Kernel** – The radial basis function kernel is commonly used in SVM classification, it can map the space in infinite dimensions. Following is the RBF kernel equation.  
  

KNN Algorithm:

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closet to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of ‘K’ training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the ‘K’ selected training points.

Let see the below example to make it a better understanding



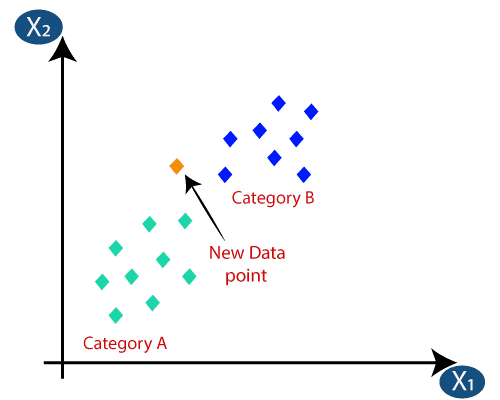
* K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
* K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
* It is also called a **lazy learner algorithm** because it does not 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.

**How does K-NN work?**

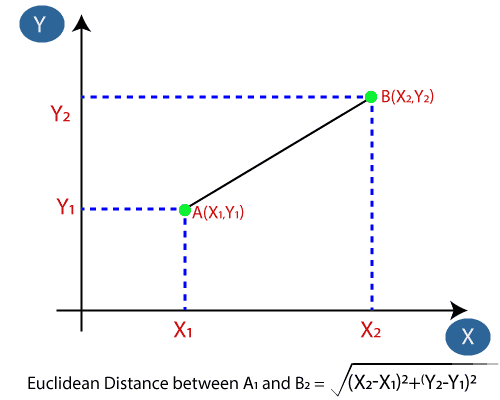
The K-NN working can be explained on the basis of the below algorithm:

* 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.

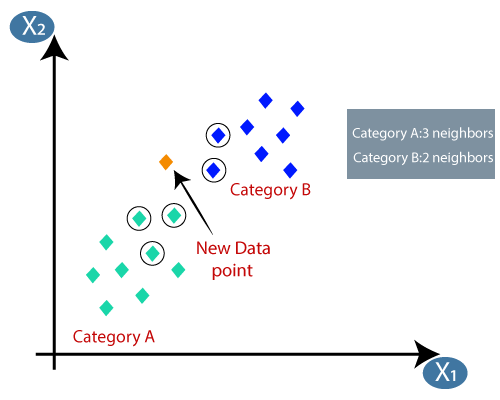
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



* 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:

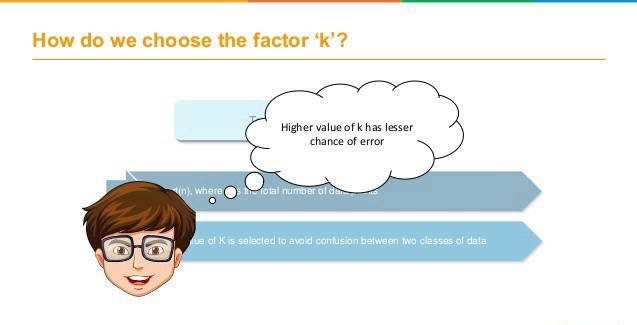


* By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

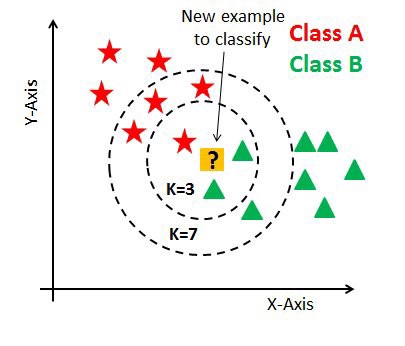


* As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

**How to choose a K value?**



Kvalue indicates the count of the nearest neighbors. We have to compute distances between test points and trained labels points. Updating distance metrics with every iteration is computationally expensive, and that’s why KNN is a lazy learning algorithm.



* As you can verify from the above image, if we proceed with K=3, then we predict that test input belongs to class B, and if we continue with K=7, then we predict that test input belongs to class A.
* That’s how you can imagine that the K value has a powerful effect on KNN performance.

**Then how to select the optimal K value?**

* There are no pre-defined statistical methods to find the most favorable value of K.
* Initialize a random K value and start computing.
* Choosing a small value of K leads to unstable decision boundaries.
* The substantial K value is better for classification as it leads to smoothening the decision boundaries.
* Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate.

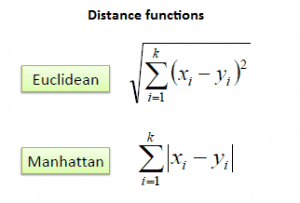
Now you will get the idea of choosing the optimal K value by implementing the model.

**Calculating distance:**

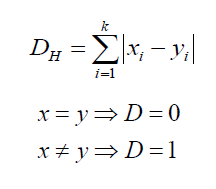
The first step is to calculate the distance between the new point and each training point. There are various methods for calculating this distance, of which the most commonly known methods are — Euclidian, Manhattan (for continuous) and Hamming distance (for categorical).

**Euclidean Distance:** Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (y).

**Manhattan Distance:** This is the distance between real vectors using the sum of their absolute difference.



**Hamming Distance:** It is used for categorical variables. If the value (x) and the value (y) are the same, the distance D will be equal to 0 . Otherwise D=1.



**Ways to perform K-NN**

*KNeighborsClassifier(n\_neighbors=5, \*, weights=’uniform’, algorithm=’auto’, leaf\_size=30, p=2, metric=’minkowski’, metric\_params=None, n\_jobs=None, \*\*kwargs)* ***algorithm : {‘auto’, ‘ball\_tree’, ‘kd\_tree’, ‘brute’}, default=’auto’***

**Brute Force**

Lets consider for simple case with two dimension plot. If we look mathematically, the simple intuition is to calculate the euclidean distance from point of interest ( of whose class we need to determine) to all the points in training set. Then we take class with majority points. This is called brute force method.

Naïve Bayes Algorithm

## What is Naive Bayes Algorithm?

The naive Bayes Algorithm is one of the popular classification machine learning algorithms that helps to classify the data based upon the conditional probability values computation. It implements the Bayes theorem for the computation and used class levels represented as feature values or vectors of predictors for classification. Naive Bayes Algorithm is a fast algorithm for classification problems. This algorithm is a good fit for real-time prediction, multi-class prediction, recommendation system, text classification, and sentiment analysis use cases. Naive Bayes Algorithm can be built using Gaussian, Multinomial and Bernoulli distribution. This algorithm is scalable and easy to implement for a large data set.

Bayes' Theorem:

* Bayes' theorem is also known as **Bayes' Rule** or **Bayes' law**, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.
* The formula for Bayes' theorem is given as:

Naïve Bayes Classifier Algorithm

**Where,**

**P(A|B) is Posterior probability**: Probability of hypothesis A on the observed event B.

**P(B|A) is Likelihood probability**: Probability of the evidence given that the probability of a hypothesis is true.

**P(A) is Prior Probability**: Probability of hypothesis before observing the evidence.

**P(B) is Marginal Probability**: Probability of Evidence.

### How Naive Bayes Algorithm works?

Let us understand the working of the Naive Bayes Algorithm using an example. We assume a training data set of weather and the target variable ‘Going shopping’. Now we will classify whether a girl will go to shopping based on weather conditions.

The given Data Set is:

|  |  |
| --- | --- |
| **Weather** | **Going Shopping** |
| Sunny | No |
| Rainy | Yes |
| Overcast | Yes |
| Sunny | Yes |
| Overcast | Yes |
| Rainy | No |
| Sunny | Yes |
| Sunny | Yes |
| Rainy | No |
| Rainy | Yes |
| Overcast | Yes |
| Rainy | No |
| Overcast | Yes |
| Sunny | No |

The following steps would be performed:

**Step 1:**Make Frequency Tables Using Data Sets.

|  |  |  |
| --- | --- | --- |
| **Weather** | **Yes** | **No** |
| Sunny | 3 | 2 |
| Overcast | 4 | 0 |
| Rainy | 2 | 3 |
| **Total** | **9** | **5** |

**Step 2:** Make a likelihood table by calculating the probabilities of each weather condition and going shopping.

|  |  |  |  |
| --- | --- | --- | --- |
| **Weather** | **Yes** | **No** | **Probability** |
| Sunny | 3 | 2 | 5/14 = 0.36 |
| Overcast | 4 | 0 | 4/14 = 0.29 |
| Rainy | 2 | 3 | 5/14 = 0.36 |
| **Total** | **9** | **5** |  |
| Probability | 9/14 = 0.64 | 5/14 = 0.36 |  |

**Step 3:** Now, we need to calculate the posterior probability using the Naive Bayes equation for each class.

**Problem instance:**A girl will go shopping if the weather is overcast. Is this statement correct?

**Solution:**

* P(Yes|Overcast) = (P(Overcast|Yes) \* P (Yes)) / P(Overcast)
* P(Overcast|Yes) = 4/9 = 0.44
* P(Yes) = 9/14 = 0.64
* P(Overcast) = 4/14 = 0.39

Now put all the calculated values in the above formula

 P(Yes|Overcast) = (0.44 \* 0.64) / 0.39

* P(Yes|Overcast) = 0.722

The class having the highest probability would be the outcome of the prediction. Using the same approach, probabilities of different classes can be predicted.

### What is Naive Bayes Algorithm used for?

* **Real-time Prediction:** Naive Bayes Algorithm is fast and always ready to learn hence best suited for real-time predictions.
* **Multi-class Prediction:** The probability of multi-classes of any target variable can be predicted using a Naive Bayes algorithm.
* **Recommendation system:** Naive Bayes classifier with the help of Collaborative Filtering builds a Recommendation System. This system uses [data mining and machine learning](https://www.educba.com/data-mining-vs-machine-learning/) techniques to filter the information which is not seen before and then predict whether a user would appreciate a given resource or not.
* **Text Classification/ Sentiment Analysis/ Spam Filtering:** Due to its better performance with multi-class problems and its independence rule, the Naive Bayes algorithm performs better or has a higher success rate in text classification; therefore, it is used in Sentiment Analysis and Spam filtering.

### Advantages and Disadvantages of Naive Bayes Algorithm

Given below are the advantages and disadvantages mentioned:

#### Advantages:

* Easy to implement.
* Fast
* If the independence assumption holds, then it works more efficiently than other algorithms.
* It requires less training data.
* It is highly scalable.
* It can make probabilistic predictions.
* Can handle both continuous and discrete data.
* Insensitive towards irrelevant features.
* It can work easily with missing values.
* Easy to update on the arrival of new data.
* Best suited for text classification problems.

#### Disadvantages:

* The strong assumption about the features to be independent is hardly true in real-life applications.
* Data scarcity.
* Chances of loss of accuracy.
* Zero Frequency, i.e. if the category of any categorical variable is not seen in the training data set, then the model assigns a zero probability to that category, and then a prediction cannot be made.

Types of Naïve Bayes Model:

There are three types of Naive Bayes Model, which are given below:

* **Gaussian**: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.
* **Multinomial**: The Multinomial Naïve Bayes classifier is used when the data is multinomial distributed. It is primarily used for document classification problems, it means a particular document belongs to which category such as Sports, Politics, education, etc.  
  The classifier uses the frequency of words for the predictors.
* **Bernoulli**: The Bernoulli classifier works similar to the Multinomial classifier, but the predictor variables are the independent Booleans variables. Such as if a particular word is present or not in a document. This model is also famous for document classification tasks.

**Evaluation Matrix in Machine Learning**

**Classification Evaluation Metrics**

Classification evaluation metrics score generally indicates how correct we are about our prediction. The higher the score, the better our model is.

Before diving into the evaluation metrics for classification, it is important to understand the confusion matrix.

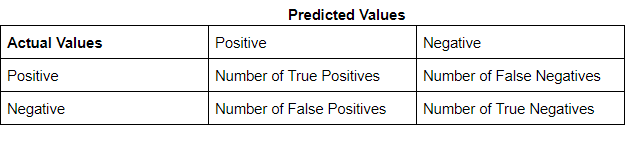
**Confusion Matrix:**

A confusion matrix is a technique for summarizing the performance of a classification algorithm.

A few terms associated with the confusion matrix are

1. True positive: An instance for which both predicted and actual values are positive.
2. True negative: An instance for which both predicted and actual values are negative.
3. False Positive: An instance for which predicted value is positive but actual value is negative.
4. False Negative: An instance for which predicted value is negative but actual value is positive.

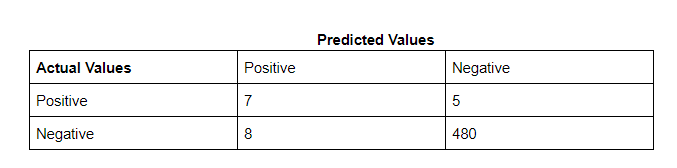
A confusion matrix follows the below format:



Example: Cancer Detection:

Consider a problem where we are required to classify whether a patient has cancer or not.

In the below table the columns represent the rows that present the number of predicted values and the columns present the number of actual values for each class. There are 500 total instances. This is the example we will use throughout the blog for classification purposes. Below is the confusion matrix.



**Note** Post this point in the blog, I’ll refer to True positive as TP, False positive as FP, True Negative as TN, and False Negative as FN.

For the above confusion matrix

Number of TP:7

Number of TN: 480

Number of FP: 8

Number of FN: 5

**Accuracy:**

Accuracy can be defined as the percentage of correct predictions made by our classification model.

The formula is:

Accuracy = Number of Correct predictions/number of rows in data

Which can also be written as:

Accuracy = (TP+TN)/number of rows in data

So, for our example:

Accuracy = 7+480/500 = 487/500 = 0.974.

Our model has a 97.4% prediction accuracy, which seems exceptionally good.

Accuracy is a good metric to use when the classes are balanced, i.e proportion of instances of all classes are somewhat similar. However, it is to be noted that accuracy is not a reliable metric for datasets having class imbalance, i.e The total number of instances of a class of data is far less than the total number of instances for another class of data. In this case, the number of positives in the dataset is 12 (TP+FN = 7+ 5 = 12) and the number of negatives is 488. Since we are dealing with an imbalanced class problem, it’s better to check our performance on other metrics before concluding our model’s performance.

**Precision:**

Precision indicates out of all positive predictions, how many are actually positive. It is defined as a ratio of correct positive predictions to overall positive predictions.

Precision = Predictions actually positive/Total predicted positive.

Precision = TP/TP+FP

For our cancer detection example, precision will be 7/7+8 = 7/15 = 0.46

**Recall:**

Recall indicates out of all actually positive values, how many are predicted positive. It is a ratio of correct positive predictions to the overall number of positive instances in the dataset.

Recall = Predictions actually positive/Actual positive values in the dataset.

Recall = TP/TP+FN

For our cancer detection example, recall will be 7/7+5 = 7/12 = 0.58

As we can see, the precision and recall are both lower than accuracy, for our example.

Deciding whether to use precision or recall:

It is mathematically impossible to increase both precision and recall at the same time, as both are inversely proportional to each other.. Depending on the problem at hand we decide which of them is more important to us.

We will first need to decide whether it’s important to avoid false positives or false negatives for our problem. Precision is used as a metric when our objective is to minimize false positives and recall is used when the objective is to minimize false negatives. We optimize our model performance on the selected metric.

Below are a couple of cases for using precision/recall.

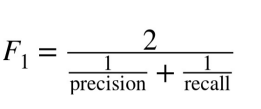
1. An AI is leading an operation for finding criminals hiding in a housing society. The goal should be to arrest only criminals, since arresting innocent citizens can mean that an innocent can face injustice. However, if the criminal manages to escape, there can be multiple chances to arrest him afterward. In this case, false positive(arresting an innocent person) is more damaging than false negative(letting a criminal walk free). Hence, we should select precision in order to minimize false positives.
2. We are all aware of the intense security checks at airports. It is of utmost importance to ensure that people do not carry weapons along them to ensure the safety of all passengers. Sometimes these systems can lead to innocent passengers getting flagged, but it is still a better scenario than letting someone dangerous onto the flight. Each flagged individual is then checked thoroughly once more and innocent people are released. In this case, the emphasis is on ensuring false negatives(people with weapons getting into flights) are avoided during initial scanning, while detected false positives(innocent passengers flagged) are eventually let free. This is a scenario for minimizing false negatives and recall is the ideal measure of how the system has performed.

**F1 score:**

Consider a scenario where your model needs to predict if a particular employee has to be promoted or not and promotion is the positive outcome. In this case, promoting an incompetent employee(false positive) and not promoting a deserving candidate(false negative) can both be equally risky for the company.

When avoiding both false positives and false negatives are equally important for our problem, we need a trade-off between precision and recall. This is when we use the f1 score as a metric. An f1 score is defined as the harmonic mean of precision and recall.

Formula:



**Threshold:**

Any machine learning algorithm for classification gives output in the probability format, i.e probability of an instance belonging to a particular class. In order to assign a class to an instance for binary classification, we compare the probability value to the threshold, i.e if the value is greater than or less than the threshold.

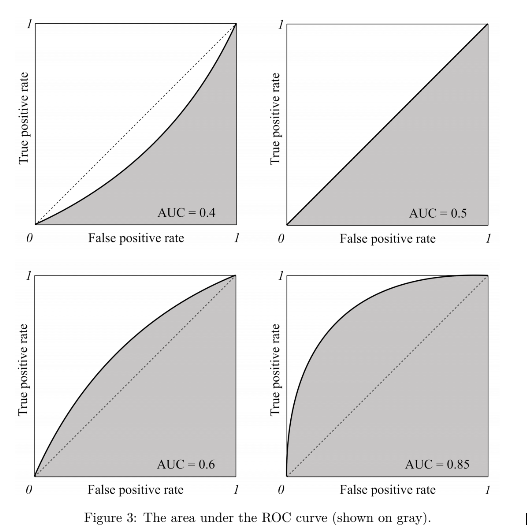
For probability >= threshold, class = 1

probability < threshold, class = 0

For multi-class classification, we can assign the class for which the instance has maximum probability value as the final class value.

**AUC-ROC:**

We use the receiver operating curve to check model performance. Wikipedia defines ROC as: “A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied”. Below is an example:



The x-axis represents the false positive rate and the y-axis represents the true positive rate.

True Positive Rate is also known as recall and False positive rate is the proportion of negative examples predicted incorrectly, both of them have a range of 0 to 1. Below are the formulas:

True Positive Rate(tpr) = TP/TP+FN

False Positive Rate(fpr) = FP/FP+TN

The shaded region is the area under the curve(AUC). Mathematically the roc curve is the region between the origin and the coordinates(tpr,fpr).

The higher the area under the curve, the better the performance of our model. We can improve the AUC-ROC score by changing true and false-positive rates, which in turn can be changed using the threshold value.

# **Logarithmic Loss**

Logarithmic Loss or Log Loss, works by penalising the false classifications. It works well for multi-class classification. When working with Log Loss, the classifier must assign probability to each class for all the samples. Suppose, there are N samples belonging to M classes, then the Log Loss is calculated as below :



where,

y\_ij, indicates whether sample i belongs to class j or not

p\_ij, indicates the probability of sample i belonging to class j

Log Loss has no upper bound and it exists on the range [0, ∞). Log Loss nearer to 0 indicates higher accuracy, whereas if the Log Loss is away from 0 then it indicates lower accuracy.

In general, minimising Log Loss gives greater accuracy for the classifier.

*The cost function used in Logistic Regression is****Log Loss****.*

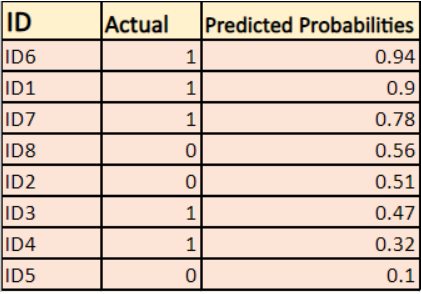
**What is Log Loss?**

Log Loss is the most important classification metric based on probabilities. It’s hard to interpret raw log-loss values, but log-loss is still a good metric for comparing models. For any given problem, a lower log loss value means better predictions.

*Mathematical interpretation:*

Log Loss is the negative average of the log of corrected predicted probabilities for each instance.

*Let us understand it with an example:*

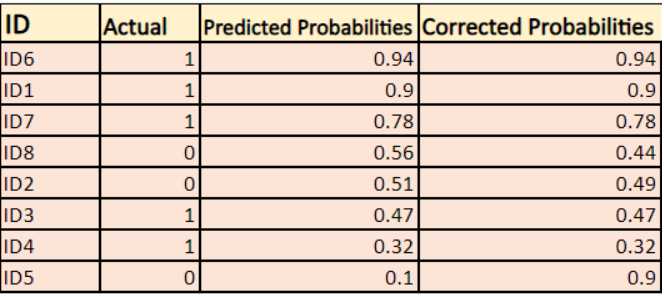


The model is giving predicted probabilities as shown above.

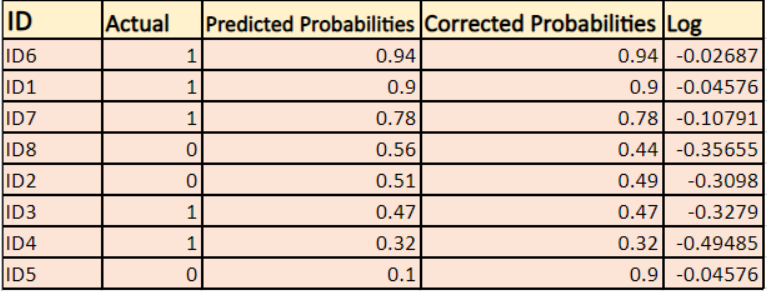
**What are the corrected probabilities?**

-> By default, the output of the logistics regression model is the probability of the sample being positive(indicated by 1) i.e if a logistic regression model is trained to classify on a `company  dataset` then the predicted probability column says What is the probability that the person has bought jacket. Here in the above data set the probability that a person with ID6 will buy a jacket is 0.94.

In the same way, the probability that a person with ID5 will buy a jacket (i.e. belong to class 1) is 0.1 but the actual class for ID5 is 0, so the probability for the class is (1-0.1)=0.9. 0.9 is the correct probability for ID5.



**We will find a log of corrected probabilities for each instance**.



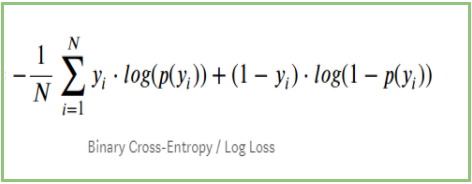
As you can see these log values are negative. To deal with the negative sign, we take the**negative average of these values**, to maintain a common convention that lower loss scores are better.



**In short, there are three steps to find Log Loss:**

1. **To find corrected probabilities.**
2. **Take a log of corrected probabilities.**
3. **Take the negative average of the values we get in the 2nd step.**

***If we summarize all the above steps, we can use the formula:-***



Here Yi represents the actual class and log(p(yi)is the probability of that class.

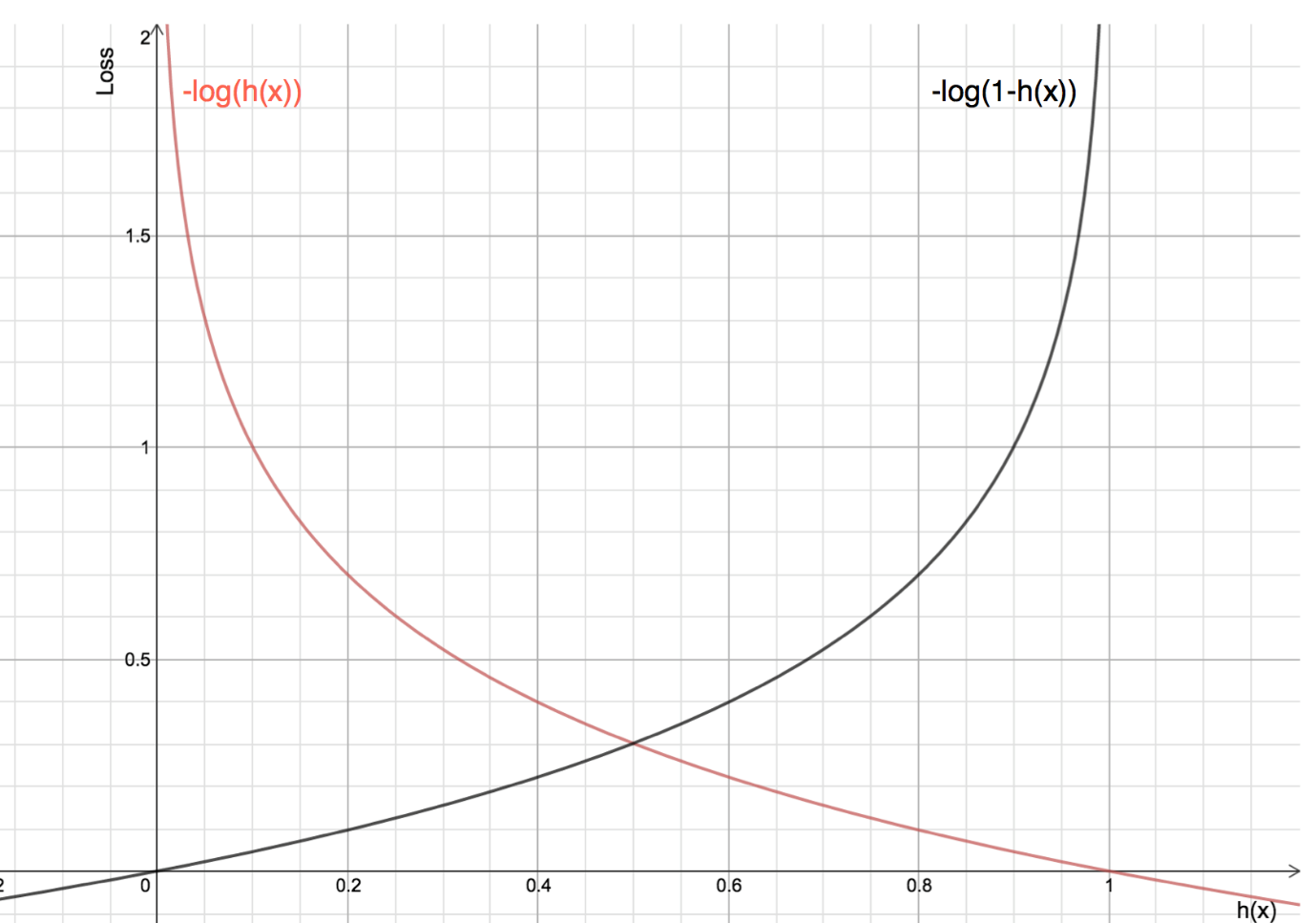
* p(yi) is the probability of 1.
* 1-p(yi) is the probability of 0.

**Now Let’s see how the above formula is working in two cases:**

1. When the actual class is 1: second term in the formula would be 0 and we will left with first term i.e. yi.log(p(yi)) and (1-1).log(1-p(yi) this will be 0.
2. When the actual class is 0: First-term would be 0 and will be left with the second term i.e (1-yi).log(1-p(yi)) and 0.log(p(yi)) will be 0.

*wow!! we got back to the original formula for binary cross-entropy/log loss 🙂*

**The benefits of taking logarithm reveal themselves when you look at the cost function graphs for actual class 1 and 0 :**



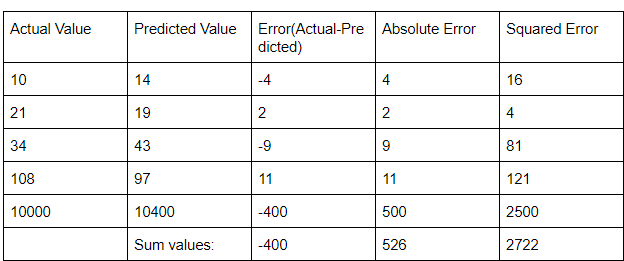
* The Red line represents 1 class. As we can see, when the predicted probability (x-axis) is close to 1, the loss is less and when the predicted probability is close to 0, loss approaches infinity.
* The Black line represents 0 class. As we can see, when the predicted probability (x-axis) is close to 0, the loss is less and when the predicted probability is close to 1, loss approaches infinity.

**Regression Evaluation Metrics**

Unlike classification, where we measure a model’s performance by checking how correct it’s predictions are, in regression we check it by measuring the difference in predicted and actual values, our objective is to minimize the metric score in order to improve our model. We will use the below example to understand more.

Example:

Below is the example we will use, the difference between the actual and predicted value is the error our model makes during prediction.



Number of samples(N) = 5.

When we add the error values (containing both positive and negative values) these elements cancel out each other and we may get an error value lower than it should be. For example, in the first 4 rows of our example -4+2-9+11 = 0. The sum of differences, i.e error will be 0. However, we can clearly see that none of the 4 rows got the prediction correct. This can lead to a problem when calculating the effectiveness of the model. In order to mitigate this issue, we use other evaluation metrics.

**Mean Absolute Error:**

As the name suggests the mean absolute error can be defined as the mean of the sum of absolute differences between the predicted and actual values of the continuous target variable.

MAE = Σ | y\_actual – y\_predicted | / n

For our example, MAE will be 526/5 = 105.2

**Mean Squared Error:**

There can be instances where large errors are undesirable. Let’s say that one of the predictions of a model is an outlier. In this case, we should penalize this higher error to a greater extent. Which is where we can use mean squared error.

The average of the sum of squares of differences between the predicted and actual values of the continuous target variable.

MSE = Σ (y\_actual – y\_predicted)2 / n

For our example, mse will be 544.4

**Root Mean Squared Error:**

The metric of the attribute changes when we calculate the error using mean squared error. For e.g, if the unit of a distance-based attribute is meters(m) the unit of mean squared error will be m2, which could make calculations confusing. In order to avoid this, we use the root of mean squared error.

RMSE = √MSE = √ Σ (y\_actual – y\_predicted)2 / n

For our example, rmse will be 23.33

**R-Squared**

In classification, where metrics output a value between 0 to 1, and the score can be used to objectively judge a model’s performance. However, in regression the target variable may not always be in the same range, e.g the price of a house can be a 6 digits number but a student’s exam marks in a subject are between 0-100. This means that the metric scores for marks will mostly be a 2 digit number, but that for housing prices can be anything between a 1-6 digit number.

In simpler words, an accuracy of 0.90 or 90% is a good performance, but does an RMSE of 90 indicate good performance. For house price which is a 6 digit number, it’s a good score, but for a student’s marks, it is a terrible one! Predicting a value of 10, when the actual value is 100 is much different than predicting a value of 200,000 when the actual value is 200,090.

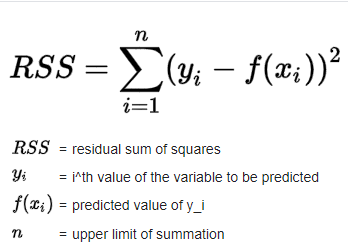
R-squared acts as a benchmark metric for judging a regression model’s performance, irrespective of the range of values the target variable presents. The range of r-squared is between 0 and 1. The greater the r-squared value the better our model’s performance is.

When new features are added to data, the R-squared value either increases or remains the same. However, adding features does not always guarantee a better performance for the model and r-squared fails to adequately capture the negative impact of adding a feature to our model, i.e whether the feature actually improves model predictions or not. In order to address this problem, the adjusted r-squared metric is used.

Before we get into the formula, let’s look into what Residual sum of squares and the total sum of squares.

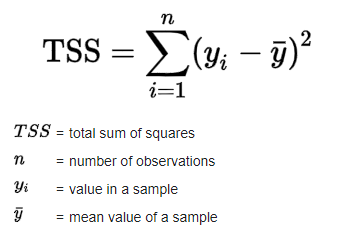
Residual sum of squares(RSS):

RSS is defined as the sum of squares of the difference between the actual and predicted values. Below is the formula

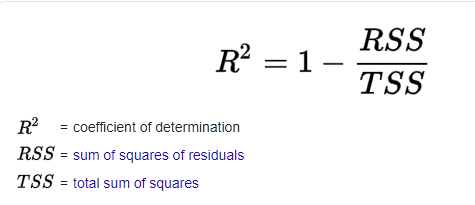


Total sum of squares(TSS):

TSS is defined as the sum of squares of the difference between the mean value and actual values. Below is the formula



The formula for r-squared will be:



**Adjusted R-Squared:**

The only difference between r-squared and adjusted r-squared is that the adjusted r-squared value increases only if the feature added improves the model performance, thus capturing the impact of adding features more adequately. Below is the formula for adjusted r-squared

