**Unit-2-----Model Evaluation and Feature engineering**

Introduction

Model selection is an essential phase in the development of powerful and precise predictive models in the field of machine learning. Model selection is the process of deciding which algorithm and model architecture is best suited for a particular task or dataset. It entails contrasting various models, assessing their efficacy, and choosing the one that most effectively addresses the issue at hand.

The choice of an appropriate machine learning model is crucial since there are various levels of complexity, underlying assumptions, and capabilities among them. A model's ability to generalize to new, untested data may not be as strong as its ability to perform effectively on a single dataset or problem. Finding a perfect balance between the complexity of models & generalization is therefore key to model selection.

Choosing a model often entails a number of processes. The first step in this process is to define a suitable evaluation metric that matches the objectives of the particular situation. According to the nature of the issue, this statistic may refer to precision, recall, accuracy, F1-score, or any other relevant measure.

The selection of numerous candidate models is then made in accordance with the problem at hand and the data that are accessible. These models might be as straightforward as decision trees or linear regression or as sophisticated as deep neural networks, random forests, or support vector machines. During the selection process, it is important to take into account the assumptions, constraints, and hyperparameters that are unique to each model.

Using a suitable methodology, such as cross-validation, the candidate models are trained and evaluated after being selected. To do this, the available data must be divided into validation and training sets, with each model fitting on the training set before being evaluated on the validation set. The models are compared using their performance metrics, then the model with the highest performance is chosen.

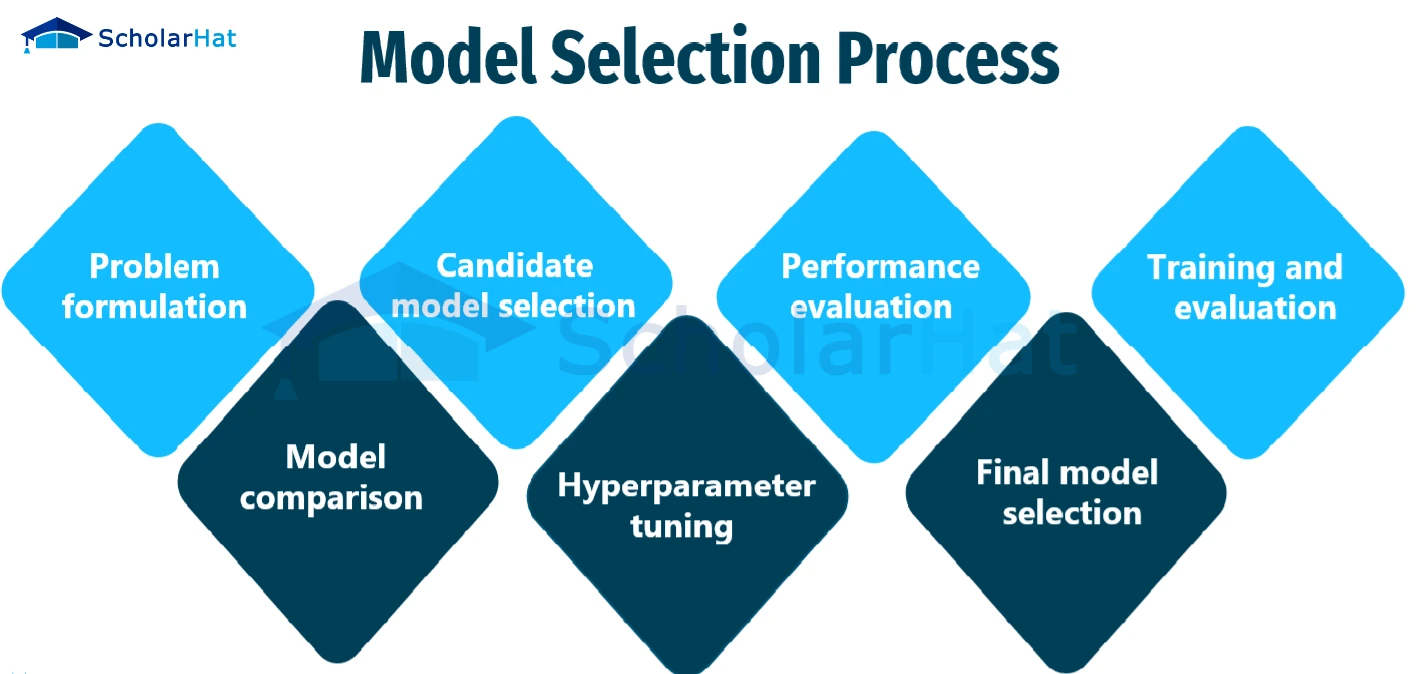
Model selection is a continuous process, though. In order to make wise selections, it frequently calls for an iterative process that involves testing several models and hyperparameters. The models are improved through this iterative process, which also aids in choosing the ideal mix of algorithms & hyperparameters.

Model Selection

In machine learning, the process of selecting the top model or algorithm from a list of potential models to address a certain issue is referred to as model selection. It entails assessing and contrasting various models according to how well they function and choosing the one that reaches the highest level of accuracy or prediction power.

Because different models have varied levels of complexity, underlying assumptions, and capabilities, model selection is a crucial stage in the machine-learning pipeline. Finding a model that fits the training set of data well and generalizes well to new data is the objective. While a model that is too complex may overfit the data and be unable to generalize, a model that is too simple could underfit the data and do poorly in terms of prediction.

The following steps are frequently included in the model selection process:

****

* **Problem formulation:** Clearly express the issue at hand, including the kind of predictions or task that you'd like the model to carry out (for example, classification, regression, or clustering).
* **Candidate model selection:** Pick a group of models that are appropriate for the issue at hand. These models can include straightforward methods like decision trees or linear regression as well as more sophisticated ones like deep neural networks, random forests, or support vector machines.
* **Performance evaluation:** Establish measures for measuring how well each model performs. Common measurements include area under the receiver's operating characteristic curve (AUC-ROC), recall, F1-score, mean squared error, and accuracy, precision, and recall. The type of problem and the particular requirements will determine which metrics are used.
* **Training and evaluation:**Each candidate model should be trained using a subset of the available data (the training set), and its performance should be assessed using a different subset (the validation set or via cross-validation). The established evaluation measures are used to gauge the model's effectiveness.
* **Model comparison:** Evaluate the performance of various models and determine which one performs best on the validation set. Take into account elements like data handling capabilities, interpretability, computational difficulty, and accuracy.
* **Hyperparameter tuning:** Before training, many models require that certain hyperparameters, such as the learning rate, regularisation strength, or the number of layers that are hidden in a neural network, be configured. Use methods like grid search, random search, and Bayesian optimization to identify these hyperparameters' ideal values.
* **Final model selection:**After the models have been analyzed and fine-tuned, pick the model that performs the best. Then, this model can be used to make predictions based on fresh, unforeseen data.

Model Selection in machine learning:

Model selection in machine learning is the process of selecting the best algorithm and model architecture for a specific job or dataset. It entails assessing and contrasting various models to identify the one that best fits the data & produces the best results. Model complexity, data handling capabilities, and generalizability to new examples are all taken into account while choosing a model. Models are evaluated and contrasted using methods like cross-validation, and grid search, as well as indicators like accuracy and mean squared error. Finding a model that balances complexity and performance to produce reliable predictions and strong generalization abilities is the aim of model selection.

There are numerous important considerations to bear in mind while selecting a model for machine learning. These factors assist in ensuring that the chosen model is effective in solving the issue at its core and has an opportunity for outstanding performance. Here are some crucial things to remember:

* **The complexity of the issue:** Determine how complex the issue you're trying to resolve is. Simple models might effectively solve some issues, but more complicated models can be necessary to fully represent complex relationships in the data. Take into account the size of the dataset, the complexity of the input features, and any potential for non-linear connections.
* **Data Availability & Quality:**Consider the accessibility and caliber of the data you already have. Using complicated models with a lot of parameters on a limited dataset may result in overfitting. Such situations may call for simpler models with fewer parameters. Take into account missing data, outliers, and noise as well as how various models respond to these difficulties.
* **Interpretability:**Consider whether the model's interpretability is crucial in your particular setting. Some models, like decision trees or linear regression, offer interpretability by giving precise insights into the correlations between the input data and the desired outcome. Complex models, such as neural networks, may perform better but offer less interpretability.
* **Model Assumptions:** Recognise the presumptions that various models make. For instance, although decision trees assume piecewise constant relationships, linear regression assumes a linear relationship between the input characteristics and the target variable. Make sure the model you choose is consistent with the fundamental presumptions underpinning the data and the issue.
* **Scalability and Efficiency:**If you're working with massive datasets or real-time applications, take the model's scalability and computing efficiency into consideration. Deep neural networks and support vector machines are two examples of models that could need a lot of time and computing power to train.
* **Regularisation and Generalisation:** Assess the model's capacity to apply to fresh, untested data. By including penalty terms to the objective function of the model, regularisation approaches like L1 or L2 regularisation can help prevent overfitting. When the training data is sparse, regularised models may perform better in terms of generalization.
* **Domain Expertise:**Consider your expertise and domain knowledge. On the basis of previous knowledge of the data or particular features of the domain, consider if particular models are appropriate for the task. Models that are more likely to capture important patterns can be found by using domain expertise to direct the selection process.
* **Resource Constraints:** Take into account any resource limitations you may have, such as constrained memory space, processing speed, or time. Make that the chosen model can be successfully implemented using the resources at hand. Some models require significant resources during training or inference.
* **Ensemble Methods:** Examine the potential advantages of ensemble methods, which integrate the results of various models in order to perform more effectively. By utilizing the diversity of several models' predictions, ensemble approaches, such as bagging, boosting, and stacking, frequently outperform individual models.
* **Evaluation and Experimentation:** experimentation and assessment of several models should be done thoroughly. Utilize the right evaluation criteria and statistical tests to compare their performance. To evaluate the models' performance on unknown data and reduce the danger of overfitting, use hold-out or cross-validation.

Model Selection Techniques

Model selection in machine learning can be done using a variety of methods and tactics. These methods assist in comparing and assessing many models to determine which is best suited to solve a certain issue. Here are some methods for selecting models that are frequently used:

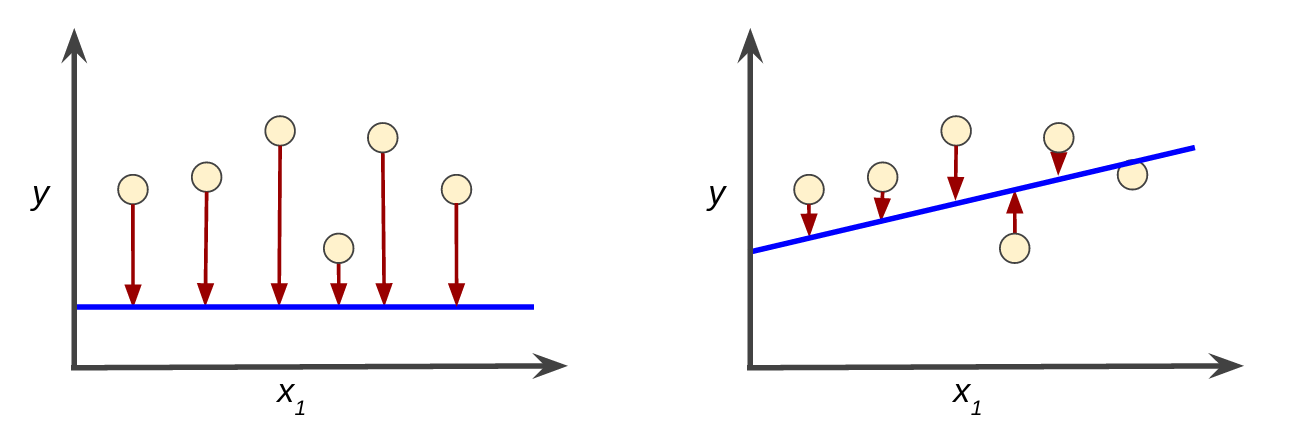
* **Train-Test Split:** With this strategy, the available data is divided into two sets: a training set & a separate test set. The models are evaluated using a predetermined evaluation metric on the test set after being trained on the training set. This method offers a quick and easy way to evaluate a model's performance using hypothetical data.
* **Cross-Validation:**A resampling approach called cross-validation divides the data into various groups or folds. Several folds are used as the test set & the rest folds as the training set, and the models undergo training and evaluation on each fold separately. Lowering the variance in the evaluation makes it easier to generate an accurate assessment of the model's performance. Cross-validation techniques that are frequently used include leave-one-out, stratified, and k-fold cross-validation.
* **Grid Search:**Hyperparameter tuning is done using the grid search technique. In order to do this, a grid containing hyperparameter values must be defined, and all potential hyperparameter combinations must be thoroughly searched. For each combination, the models are trained, assessed, and their performances are contrasted. Finding the ideal hyperparameter settings to optimize the model's performance is made easier by grid search.
* **Random Search:**A set distribution for hyperparameter values is sampled at random as part of the random search hyperparameter tuning technique. In contrast to grid search, which considers every potential combination, random search only investigates a portion of the hyperparameter field. When a thorough search is not possible due to the size of the search space, this strategy can be helpful.
* **Bayesian optimization:** A more sophisticated method of hyperparameter tweaking, Bayesian optimization. It models the relationship between the performance of the model and the hyperparameters using a probabilistic model. It intelligently chooses which set of hyperparameters to investigate next by updating the probabilistic model and iteratively assessing the model's performance. When the search space is big and expensive to examine, Bayesian optimization is especially effective.
* **Model averaging:** This technique combines forecasts from various models to get a single prediction. For regression issues, this can be accomplished by averaging the predictions, while for classification problems, voting or weighted voting systems can be used. Model averaging can increase overall prediction accuracy by lowering the bias and variation of individual models.
* **Information Criteria:**Information criteria offer a numerical assessment of the trade-off between model complexity and goodness of fit. Examples include the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC). These criteria discourage the use of too complicated models and encourage the adoption of simpler models that adequately explain the data.
* **Domain Expertise & Prior Knowledge:**Prior understanding of the problem and the data, as well as domain expertise, can have a significant impact on model choice. The models that are more suitable given the specifics of the problem and the details of the data may be known by subject matter experts.
* **Model Performance Comparison:** Using the right assessment measures, it is vital to evaluate the performance of various models. Depending on the issue at hand, these measurements could include F1-score, mean squared error, accuracy, precision, recall, or the area beneath the receiver's operating characteristic curve (AUC-ROC). The best-performing model can be found by comparing many models.

**Training Model**

**Training** a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called **empirical risk minimization**.

Loss is the penalty for a bad prediction. That is, **loss** is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have *low* loss, on average, across all examples. For example, Figure 3 shows a high loss model on the left and a low loss model on the right. Note the following about the figure:

* The arrows represent loss.
* The blue lines represent predictions.



**Figure 3. High loss in the left model; low loss in the right model.**

Notice that the arrows in the left plot are much longer than their counterparts in the right plot. Clearly, the line in the right plot is a much better predictive model than the line in the left plot.

You might be wondering whether you could create a mathematical function—a loss function—that would aggregate the individual losses in a meaningful fashion.

### Squared loss: a popular loss function

The linear regression models we'll examine here use a loss function called **squared loss** (also known as **L2 loss**). The squared loss for a single example is as follows:

= the square of the difference between the label and the prediction

= (observation - prediction(**x**))2

= (y - y')2

**Mean square error** (**MSE**) is the average squared loss per example over the whole dataset. To calculate MSE, sum up all the squared losses for individual examples and then divide by the number of examples:

MSE=1/N E(y-prediction(x))2

(x,y)eD

where:

* (x,y) is an example in which
  + x is the set of features (for example, chirps/minute, age, gender) that the model uses to make predictions.
  + y is the example's label (for example, temperature).
* Prediction(x)  is a function of the weights and bias in combination with the set of features Z.
* D is a data set containing many labeled examples, which are (x,y) pairs.
* Nis the number of examples in D.

Although MSE is commonly-used in machine learning, it is neither the only practical loss function nor the best loss function for all circumstances.

# **Representation Learning**

3345 papers with code • 5 benchmarks • 9 datasets

**Representation Learning** is a process in machine learning where algorithms extract meaningful patterns from raw data to create representations that are easier to understand and process. These representations can be designed for interpretability, reveal hidden features, or be used for transfer learning. They are valuable across many fundamental machine learning tasks like [image classification](https://paperswithcode.com/task/image-classification) and [retrieval](https://paperswithcode.com/task/image-retrieval).

Deep neural networks can be considered representation learning models that typically encode information which is projected into a different subspace. These representations are then usually passed on to a linear classifier to, for instance, train a classifier.

Representation learning can be divided into:

* **Supervised representation learning**: learning representations on task A using annotated data and used to solve task B
* **Unsupervised representation learning**: learning representations on a task in an unsupervised way (label-free data). These are then used to address downstream tasks and reducing the need for annotated data when learning news tasks. Powerful models like [GPT](https://paperswithcode.com/method/gpt) and [BERT](https://paperswithcode.com/method/bert) leverage unsupervised representation learning to tackle language tasks.

More recently, [self-supervised learning (SSL)](https://paperswithcode.com/task/self-supervised-learning) is one of the main drivers behind unsupervised representation learning in fields like computer vision and NLP.

Here are some additional readings to go deeper on the task:

* [Representation Learning: A Review and New Perspectives](https://paperswithcode.com/paper/representation-learning-a-review-and-new) - Bengio et al. (2012)
* [A Few Words on Representation Learning](https://sthalles.github.io/a-few-words-on-representation-learning/) - Thalles Silva

# **Interpretability**

It is difficult to (mathematically) define interpretability. A (non-mathematical) definition of interpretability that I like by Miller (2017)[3](https://christophm.github.io/interpretable-ml-book/interpretability.html#fn3) is: **Interpretability is the degree to which a human can understand the cause of a decision.** Another one is: **Interpretability is the degree to which a human can consistently predict the model’s result** [4](https://christophm.github.io/interpretable-ml-book/interpretability.html#fn4). The higher the interpretability of a machine learning model, the easier it is for someone to comprehend why certain decisions or predictions have been made. A model is better interpretable than another model if its decisions are easier for a human to comprehend than decisions from the other model. I will use both the terms interpretable and explainable interchangeably. Like Miller (2017), I think it makes sense to distinguish between the terms interpretability/explainability and explanation. I will use “explanation” for explanations of individual predictions. See the [section about explanations](https://christophm.github.io/interpretable-ml-book/explanation.html#explanation) to learn what we humans see as a good explanation.

Interpretable machine learning is a useful umbrella term that captures the “extraction of relevant knowledge from a machine-learning model concerning relationships either contained in data or learned by the model”. [5](https://christophm.github.io/interpretable-ml-book/interpretability.html#fn5)

1. Miller, Tim. “Explanation in artificial intelligence: Insights from the social sciences.” arXiv Preprint arXiv:1706.07269. (2017).[↩︎](https://christophm.github.io/interpretable-ml-book/interpretability.html#fnref3)
2. Kim, Been, Rajiv Khanna, and Oluwasanmi O. Koyejo. “Examples are not enough, learn to criticize! Criticism for interpretability.” Advances in Neural Information Processing Systems (2016).[↩︎](https://christophm.github.io/interpretable-ml-book/interpretability.html#fnref4)
3. Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., & Yu, B. “Definitions, methods, and applications in interpretable machine learning.” Proceedings of the National Academy of Sciences, 116(44), 22071-22080. (2019).[↩︎](https://christophm.github.io/interpretable-ml-book/interpretability.html#fnref5)

# Machine Learning Model Evaluation

Machine Learning Model does not require hard-coded algorithms. We feed a large amount of data to the model and the model tries to figure out the features on its own to make future predictions. So we must also use some techniques to determine the predictive power of the model.

### **Machine Learning Model Evaluation**

Model evaluation is the process that uses some metrics which help us to analyze the performance of the model. As we all know that model development is a multi-step process and a check should be kept on how well the model generalizes future predictions. Therefore evaluating a model plays a vital role so that we can judge the performance of our model. The evaluation also helps to analyze a model’s key weaknesses. There are many metrics like Accuracy, Precision, Recall, F1 score, Area under Curve, Confusion Matrix, and Mean Square Error. Cross Validation is one technique that is followed during the training phase and it is a model evaluation technique as well.

#### **Cross Validation and Holdout**

Cross Validation is a method in which we do not use the whole dataset for training. In this technique, some part of the dataset is reserved for testing the model. There are many types of Cross-Validation out of which K Fold Cross Validation is mostly used. In K Fold Cross Validation the original dataset is divided into k subsets. The subsets are known as folds. This is repeated k times where 1 fold is used for testing purposes. Rest k-1 folds are used for training the model. So each data point acts as a test subject for the model as well as acts as the training subject. It is seen that this technique generalizes the model well and reduces the error rate

Holdout is the simplest approach. It is used in neural networks as well as in many classifiers.  In this technique, the dataset is divided into train and test datasets. The dataset is usually divided into ratios like 70:30 or 80:20. Normally a large percentage of data is used for training the model and a small portion of the dataset is used for testing the model.

## Evaluation Metrics for Classification Task

In this Python code, we have imported the iris dataset which has features like the length and width of sepals and petals. The target values are Iris setosa, Iris virginica, and Iris versicolor. After importing the dataset we divided the dataset into train and test datasets in the ratio 80:20. Then we called [Decision Trees](https://www.geeksforgeeks.org/decision-tree/) and trained our model. After that, we performed the prediction and calculated the [accuracy score](https://www.geeksforgeeks.org/metrics-for-machine-learning-model/), [precision, recall](https://www.geeksforgeeks.org/precision-recall-curve-ml/), and f1 score. We also plotted the [confusion matrix](https://www.geeksforgeeks.org/confusion-matrix-machine-learning/).

#### Importing Libraries and Dataset

[**Python**](https://www.geeksforgeeks.org/python-programming-language/) libraries make it very easy for us to handle the data and perform typical and complex tasks with a single line of code.

* [**Pandas**](https://www.geeksforgeeks.org/python-pandas-dataframe/)– This library helps to load the data frame in a 2D array format and has multiple functions to perform analysis tasks in one go.
* [**Numpy**](https://www.geeksforgeeks.org/python-numpy/)– Numpy arrays are very fast and can perform large computations in a very short time.
* [**Matplotlib**](https://www.geeksforgeeks.org/matplotlib-tutorial/)/[**Seaborn**](https://www.geeksforgeeks.org/introduction-to-seaborn-python/)– This library is used to draw visualizations.
* Sklearn – This module contains multiple libraries having pre-implemented functions to perform tasks from data preprocessing to model development and evaluation.

# import pandas as pd

# import numpy as np

# from sklearn import tree

# from sklearn import datasets

# from sklearn.datasets import load\_iris

# from sklearn.tree import DecisionTreeClassifier

# from sklearn.model\_selection import train\_test\_split

# import seaborn as sns

# import matplotlib.pyplot as plt

# from sklearn.metrics import precision\_score,\

# recall\_score, f1\_score, accuracy\_score

Now let’s load the toy dataset iris flowers from the sklearn.datasets library and then split it into training and testing parts (for model evaluation) in the 80:20 ratio.

* Python3

|  |
| --- |
| iris **=** load\_iris()  X **=** iris.data  y **=** iris.target    # Holdout method.Dividing the data into train and test  X\_train, X\_test,\      y\_train, y\_test **=** train\_test\_split(X, y,                                         random\_state**=**20,                                         test\_size**=**0.20) |

Now, let’s train a Decision Tree Classifier model on the training data, and then we will move on to the evaluation part of the model using different metrics.

* Python3

|  |
| --- |
| tree **=** DecisionTreeClassifier()  tree.fit(X\_train, y\_train)  y\_pred **=** tree.predict(X\_test) |

# Improving Performance of ML Models

## Performance Improvement with Ensembles

Ensembles can give us boost in the machine learning result by combining several models. Basically, ensemble models consist of several individually trained supervised learning models and their results are merged in various ways to achieve better predictive performance compared to a single model. Ensemble methods can be divided into following two groups −

### **Sequential ensemble methods**

As the name implies, in these kind of ensemble methods, the base learners are generated sequentially. The motivation of such methods is to exploit the dependency among base learners.

### **Parallel ensemble methods**

As the name implies, in these kind of ensemble methods, the base learners are generated in parallel. The motivation of such methods is to exploit the independence among base learners.

## Ensemble Learning Methods

The following are the most popular ensemble learning methods i.e. the methods for combining the predictions from different models −

### **Bagging**

The term bagging is also known as bootstrap aggregation. In bagging methods, ensemble model tries to improve prediction accuracy and decrease model variance by combining predictions of individual models trained over randomly generated training samples. The final prediction of ensemble model will be given by calculating the average of all predictions from the individual estimators. One of the best examples of bagging methods are random forests.

### **Boosting**

In boosting method, the main principle of building ensemble model is to build it incrementally by training each base model estimator sequentially. As the name suggests, it basically combine several week base learners, trained sequentially over multiple iterations of training data, to build powerful ensemble. During the training of week base learners, higher weights are assigned to those learners which were misclassified earlier. The example of boosting method is AdaBoost.

### **Voting**

In this ensemble learning model, multiple models of different types are built and some simple statistics, like calculating mean or median etc., are used to combine the predictions. This prediction will serve as the additional input for training to make the final prediction.

## Bagging Ensemble Algorithms

The following are three bagging ensemble algorithms −

* [Bagged Decision Tree](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_bagged_decision_tree.htm)
* [Random Forest](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_random_forest.htm)
* [Extra Trees](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_extra_trees.htm)

## Boosting Ensemble Algorithms

The followings are the two most common boosting ensemble algorithms −

* [AdaBoost](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_adaboost.htm)
* [Stochastic Gradient Boosting](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_stochastic_gradient_boosting.htm)

## Voting Ensemble Algorithms

As discussed, voting first creates two or more standalone models from training dataset and then a voting classifier will wrap the model along with taking the average of the predictions of sub-model whenever needed new data.

In the following Python recipe, we are going to build Voting ensemble model for classification by using VotingClassifier class of sklearn on Pima Indians diabetes dataset. We are combining the predictions of logistic regression, Decision Tree classifier and SVM together for a classification problem as follows −

First, import the required packages as follows −

from pandas import read\_csv

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.ensemble import VotingClassifier

Now, we need to load the Pima diabetes dataset as did in previous examples −

path = r"C:\pima-indians-diabetes.csv"

headernames = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']

data = read\_csv(path, names = headernames)

array = data.values

X = array[:,0:8]

Y = array[:,8]

Next, give the input for 10-fold cross validation as follows −

kfold = KFold(n\_splits = 10, random\_state = 7)

Next, we need to create sub-models as follows −

estimators = []

model1 = LogisticRegression()

estimators.append(('logistic', model1))

model2 = DecisionTreeClassifier()

estimators.append(('cart', model2))

model3 = SVC()

estimators.append(('svm', model3))

Now, create the voting ensemble model by combining the predictions of above created sub models.

ensemble = VotingClassifier(estimators)

results = cross\_val\_score(ensemble, X, Y, cv = kfold)

print(results.mean())

**Output**

0.7382262474367738

The output above shows that we got around 74% accuracy of our voting classifier ensemble model.

# Feature Transformation Techniques in Machine Learning

 **Read**

 Discuss

 Courses

Most machine learning algorithms are statistics dependent, meaning that all of the algorithms are indirectly using a statistical approach to solve the complex problems in the data. In statistics, the normal distribution of the data is one that a statistician desires to be. A [normal distribution](https://www.geeksforgeeks.org/mathematics-probability-distributions-set-3-normal-distribution/) of the data helps statisticians to solve the complex patterns of the data and gain valuable insights from the same. But for the algorithm scenario, a normal distribution of the data can not be desired every time with every type of dataset, which means the data which is not normally distributed needs preprocessing and cleaning before applying the[machine learning](https://www.geeksforgeeks.org/machine-learning/) algorithm to it.

In this article, we will be discussing the feature transformation techniques in machine learning which are used to transform the data from one form to another form, keeping the essence of the data. In simple words, the transformers are the type of functions that are applied to data that is not normally distributed, and once applied there is a high of getting normally distributed data.

There are 3 types of Feature transformation techniques:

1. Function Transformers
2. Power Transformers
3. Quantile Transformers

## Function Transformers

Function transformers are the type of feature transformation technique that uses a particular function to transform the data to the normal distribution. Here the particular function is applied to the data observations.

There is not any thumb rule for the selection of function transformers, the function can be designed by anyone good at domain knowledge of the data, but mostly there are 5 types of function transformers that are used and which also solve the issue of normal distribution almost every time.

1. Log Transform
2. Square Transform
3. Square Root Transform
4. Reciprocal Transform
5. Custom Transform

Let us try to discuss the core intuition of every transformation one by one.

### Log Transform

Log transform is one of the simplest transformations on the data in which the log is applied to every single distribution of the data and the result from the log is considered the final day to feed the machine learning algorithms.

Through experiments, it is proven that [log transforms](https://www.geeksforgeeks.org/numpy-log-python/) performs so well on the right-skewed data. It transforms the right-skewed data into normally distributed data so well.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** FunctionTransformer  transform **=** FunctionTransformer(func**=**np.log1p)  transformed\_data **=** transform.fit\_transform(data) |

### Square Transform

Square transform is the type of transformer in which the square of the data is considered instead of the normal data. In simple words, in this transformed the data is applied with the [square function](https://www.geeksforgeeks.org/numpy-square-python/), where the square of every single observation will be considered as the final transformed data.

* Python3

|  |
| --- |
| **import** numpy as np  tranformed\_data **=** np.square(data) |

### Square Root Transform

In this transform, the [square root](https://www.geeksforgeeks.org/floor-square-root-without-using-sqrt-function-recursive/) of the data is calculated. This transform performs so well on the left-skewed data and efficiently transformed the left-skewed data into normally distributed data.

* Python3

|  |
| --- |
| **import** numpy as np  tranformed\_data **=** np.sqrt(data) |

### Reciprocal Transform

In this transformation, the reciprocal of every observation is considered. This transform is useful in some of the datasets as the reciprocal of the observations works well to achieve normal distributions.

* Python3

|  |
| --- |
| **import** numpy as np  tranformed\_data **=** np.reciprocal(data) |

### Custom Transforms

In every dataset, the log and square root transforms can not be used, as every data can have different patterns and complexity. Based on the domain knowledge of the data, custom transformations can be applied to transform the data into a normal distribution. The custom transforms here can be any function or parameter like sin, cos, tan, cube, etc.

* Python3

|  |
| --- |
| importy numpy as np  sin\_tranformed\_data **=** np.sin(data)  cos\_tranformed\_data **=** np.cos(data)  tan\_tranformed\_data **=** np.tan(data) |

## Power Transformers

Power Transformation techniques are the type of feature transformation technique where the power is applied to the data observations for transforming the data.

There are two types of Power Transformation techniques:

1. [Box-Cox Transform](https://www.geeksforgeeks.org/box-cox-transformation-using-python/)
2. Yeo-Johnson Transform

### Box-Cox Transform

This transform technique is mainly used for transforming the data observations by applying power to them. The power of the data observations is denoted by Lambda(λ). There are mainly two conditions associated with the power in this transform, which is lambda equals zero and not equal to zero. The mathematical formulation of this transform is as follows:

Here the lambda is the power applied to every data observation. Based upon the iteration technique every single value of the lambda is examined and the best fit value of the lambda is then applied to the data to transform it.

Here the transformed value of every data observation will lie between 5 to -5. One major disadvantage associated with this transformation technique is that this technique can only be applied to positive observations. it is not applicable for negative and zero values of the data observations.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** PowerTransformer  boxcox **=** PowerTransformer(method**=**'box-cox')  data\_transformed **=** boxcox.fit\_transform(data) |

### Yeo Johnson Transform

This transformation technique is also a power transform technique, where the power of the data observations is applied to transform the data. This is an advanced form of a box cox transformations technique where it can be applied to even zero and negative values of data observations also.

The mathematical formulations of this transformations technique are as follows:

In this transformation technique, y represents the appropriate value of Xi. In scikit learn the default parameter is set to Yeo Johnson in the Power Transformer class.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** PowerTransformer  boxcox **=** PowerTransformer()  data\_transformed **=** boxcox.fit\_transform(data) |

## Quantile Transformers

Quantile transformation techniques are the type of feature transformation technique that can be applied to NY numerical data observations. This transformation technique can be implemented using sklearn.

In this transformation technique, the input data can be fed to this transformer where this transformer makes the distribution of the output data normal to fed to the further machine learning algorithm.

Here there is a paramere called *output\_distribution*, which value can be set to *uniform*or *normal*.

* Python3

|  |
| --- |
| **from** sklearn.preprocessing **import** QuantileTransformer  quantile\_trans **=** QuantileTransformer(output\_distribution**=**'normal')  data\_transformed **=** quantile.fit\_transform(data) |

## Key Takeaways

* The featured transformation techniques are used to transform the data to normal distribution for better performance of the algorithm.
* The Log transforms perform so well on the right-skewed data. Whereas the square root transformers perform so well on left-skewed data.
* Based on the domain knowledge of the problem statement and the data, the custom data transformations technique can be also applied efficiently.
* Box-Cox transformations can be applied to only positive data observations which return the transformed values between -5 to 5.
* Yeo Johnson’s transformations technique can be applied to zero and negative values as well.

## Conclusion

In this article, we discussed some of the famous and most used data transformation techniques that are used to transform the data from any other distribution to normal distribution. this will help one to apply data preprocessing and cleaning techniques n the complex data easily and will help one to answer some of the interview questions related to it very efficiently.

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you.

# Feature Subset Selection Process

 **Read**

 Discuss

 Courses

**Feature Selection** is the most critical pre-processing activity in any machine learning process. It intends to select a subset of attributes or features that makes the most meaningful contribution to a machine learning activity. In order to understand it, let us consider a small example i.e. **Predict the weight of students based on the past information about similar students**, which is captured inside a ‘Student Weight’ data set. The data set has 04 features like **Roll Number, Age, Height & Weight.**Roll Number has no effect on the weight of the students, so we eliminate this feature. So now the new data set will be having only 03 features. This subset of the data set is expected to give better results than the full set.

| **Age** | **Height** | **Weight** |
| --- | --- | --- |
| 12 | 1.1 | 23 |
| 11 | 1.05 | 21.6 |
| 13 | 1.2 | 24.7 |
| 11 | 1.07 | 21.3 |
| 14 | 1.24 | 25.2 |
| 12 | 1.12 | 23.4 |

**The above data set is a reduced dataset.** Before proceeding further, we should look at the fact why we have reduced the dimensionality of the above dataset OR what are the issues in **High Dimensional Data?**

**High Dimensional**refers to the high number of variables or attributes or features present in certain data sets, more so in the domains like DNA analysis, geographic information system (GIS), etc.  It may have sometimes hundreds or thousands of dimensions which is not good from the machine learning aspect because it may be a big challenge for any ML algorithm to handle that. On the other hand, a high quantity of computational and a high amount of time will be required. Also, a model built on an extremely high number of features may be very difficult to understand. **For these reasons, it is necessary to take a subset of the features instead of the full set.** So we can deduce that the objectives of feature selection are:

1. Having a faster and more cost-effective (less need for computational resources) learning model
2. Having a better understanding of the underlying model that generates the data.
3. Improving the efficacy of the learning model.

**Main Factors Affecting Feature Selection**

**a. Feature Relevance:** In the case of supervised learning, the input data set (which is the training data set), has a class label attached. A model is inducted based on the training data set — so that the inducted model can assign class labels to new, unlabeled data. Each of the predictor variables, ie expected to contribute information to decide the value of the class label. In case of a variable is not contributing any information, it is said to be irrelevant. In case the information contribution for prediction is very little, the variable is said to be weakly relevant. The remaining variables, which make a significant contribution to the prediction task are said to be strongly relevant variables.

In the case of unsupervised learning, there is no training data set or labelled data. Grouping of similar data instances are done and the similarity of data instances are evaluated based on the value of different variables. Certain variables do not contribute any useful information for deciding the similarity of dissimilar data instances. Hence, those variable makes no significant contribution to the grouping process. These variables are marked as irrelevant variables in the context of the unsupervised machine learning task.

We can understand the concept by taking a real-world example: At the start of the article, we took a random dataset of the student. In that, Roll Number doesn’t contribute any significant information in predicting what the Weight of a student would be. Similarly, if we are trying to group together students with similar academic capabilities, *Roll No* can really not contribute any information. So, in the context of grouping students with similar academic merit, the variable *Roll No* is quite irrelevant.  Any feature which is irrelevant in the context of a machine learning task is a candidate for rejection when we are selecting a subset of features.

**b. Feature Redundancy:** A feature may contribute to information that is similar to the information contributed by one or more features. For example, in the Student Data-set, both the features **Age & Height** contribute similar information. This is because, with an increase in age, weight is expected to increase. Similarly, with the increase in Height also weight is expected to increase. So, in context to that problem, Age and Height contribute similar information. In other words, irrespective of whether the feature **Height** is present or not, the learning model will give the same results. In this kind of situation where one feature is similar to another feature, **the feature is said to be potentially redundant** in the context of a machine learning problem.

All features having potential redundancy are candidates for rejection in the final feature subset. Only a few representative features out of a set of potentially redundant features are considered for being a part of the final feature subset. So in short, the main objective of feature selection is to remove all features which are irrelevant and take a representative subset of the features which are potentially redundant. This leads to a meaningful feature subset in the context of a specific learning task.

**The measure of feature relevance and redundancy**

**a. Measures of Feature Relevance:**In the case of supervised learning, **mutual information** is considered as a good measure of information contribution of a feature to decide the value of the class label. That is why it is a good indicator of the relevance of a feature with respect to the class variable. The higher the value of mutual information of a feature, the more relevant is that feature. Mutual information can be calculated as follows:

**Where, marginal entropy of the class,**(

**Marginal entropy of the feature** ‘x’,

And **K** = number of classes, **C** = class variable, **f** = feature set that take discrete values. In the case of unsupervised learning, there is no class variable. Hence, feature-to-class mutual information cannot be used to measure the information contribution of the features. In the case of unsupervised learning, the entropy of the set of features without one feature at a time is calculated for all features. Then the features are ranked in descending order of information gain from a feature and the top percentage (value of beta is a design parameter of the algorithm) of features are selected as relevant features. The entropy of a feature f is calculated using Shannon’s formula below:

 is used only for features that take the discrete values. For continuous features, it should be replaced by discretization performed first to estimate the probabilities p(f=x).

**b. Measures of Feature Redundancy:** There are multiple measures of similarity of information contribution, the main ones are:

* Correlation-based Measures
* Distance-based Measures
* Other coefficient-based Measure

**1. Correlation Based Similarity Measure**

Correlation is a measure of linear dependency between two random variables. Pearson’s product correlation coefficient is one of the most popular and accepted measures correlation between two random variables. For two random feature variables F1 and F2 , the Pearson coefficient is defined as:

 where

  where

Correlation value ranges between +1 and -1. A correlation of 1 (+/-) indicates perfect correlation. In case the correlation is zero, then the features seem to have no linear relationship. Generally for all feature selection problems, a threshold value is adopted to decide whether two features have adequate similarity or not.

**2. Distance-Based Similarity Measure**

The most common distance measure is the **Euclidean distance,** which, between two features F1 and F2 are calculated as:

Where the features represent an n-dimensional dataset. Let us consider that the dataset has two features, Subjects (F1) and marks (F2) under consideration. The Euclidean distance between the two features will be calculated like this:

| **Subjects (F1)** | **Marks (F2)** | **(F1-F2)** | **(F1-F2)2** |
| --- | --- | --- | --- |
| 2 | 6 | -4 | 16 |
| 3 | 5.5 | -2.5 | 6.25 |
| 6 | 4 | 2 | 4 |
| 7 | 2.5 | 4.5 | 20.25 |
| 8 | 3 | 5 | 25 |
| 6 | 5.5 | 0.5 | 0.25 |
| 6 | 7 | -1 | 1 |
| 7 | 6 | 1 | 1 |
| 8 | 6 | 2 | 4 |
| 9 | 7 | 2 | 4 |

A more generalized form of the Euclidean distance is the **Minkowski Distance,** measured as

Minkowski distance takes the form of Euclidean distance (also called **L2 norm**) where r = 2. At r=1, it takes the form of **Manhattan** distance (also called **L1 norm**) :

**3. Other Similarity Measures**

**Jaccard index/coefficient** is used as a measure of dissimilarity between two features is complementary of Jaccard Index. For two features having binary values, Jaccard Index is measured as:

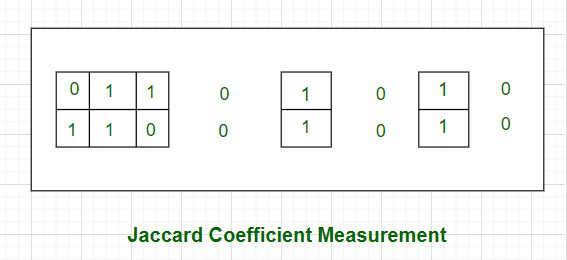
Where   = number of cases when both the feature have value 1,

=  number of cases where the feature 1 has value 0 and feature 2 has value 1,

=  the number of cases where feature 1 has value 1 and feature 2 has value 0.

Jaccard distance:

Let us take an example to understand it better. Consider two features, F1 and F2 having values (0, 1, 1, 0, 1, 0, 1, 0) and (1, 1, 0, 0, 1, 0, 0, 0).



As shown in the above picture, the cases where both the values are 0 have been left out without border- as an indication of the fact that they will be excluded in the calculation of the Jaccard coefficient.

Jaccard coefficient of F1 and F2 , J =

**Therefore, Jaccard Distance between those two features is dj = (1 – 0.4) = 0.6**

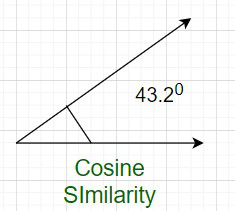
**Note:** One more measure of similarity using similarity coefficient calculation is **Cosine Similarity**. For the sake of understanding, let u stake an example of the **text classification problem.** The text needs to be first transformed into features with a word token being a feature and the number of times the word occurs in a document comes as a value in each row. There are thousands of features in such a text dataset. However, the data set is sparse in nature as only a few words do appear in a document and hence in a row of the data set. So each row has very few non-zero values. However, the non-zero values can be anything integer value as the same word may occur any number of times. Also, considering the sparsity of the dataset, the 0-0 matches need to be ignored. **Cosine similarity** which is one of the most popular measures in text classification is calculated as:

Where, x.y is the vector dot product of x and y  =

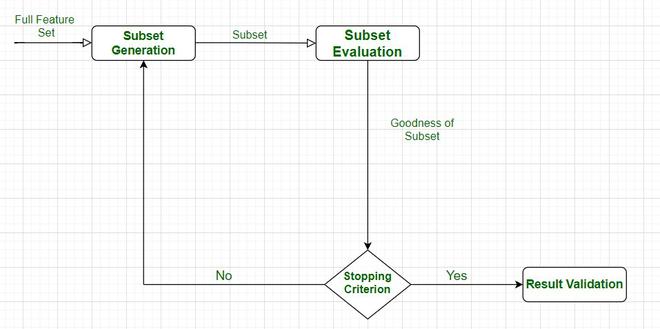
  and

So let’s calculate the cosine similarity of x and y, where x = (2,4,0,0,2,1,3,0,0) and y = (2,1,0,0,3,2,1,0,1). In this case, dot product of x and y will be **x.y = 2\*2 + 4\*1 + 0\*0 + 0\*0 + 2\*3 + 1\*2 + 3\*1 + 0\*0 + 0\*1 = 19.**

**Cosine Similarity** measures the angle between x and y vectors. Hence, if cosine similarity has a value of 1, the angles between x and y is 0 degrees which means x and y are the same except for the magnitude. If the cosine similarity is 0, the angle between x and y is 900. Hence, they do not share any similarity. In the case of the above example, the angle comes out to be 43.20.



**Even after all these steps, there are some few more steps.**You can understand it by the following flowchart:



***Feature Selection Process***

After the successful completion of this cycle, we get the desired features, and we have finally tested them also.

The feature subset selection process involves identifying and selecting a subset of relevant features from a given dataset. It aims to improve model performance, reduce overfitting, and enhance interpretability. Here is a general outline of the feature subset selection process:

**Data Preparation:**

Clean the data: Handle missing values, outliers, and data inconsistencies.  
Encode categorical variables: Convert categorical features into numerical representations, such as one-hot encoding or label encoding.  
Normalize or standardize numerical features: Scale numerical features to a common range to avoid bias.

**Feature Ranking/Scoring:**

Select an appropriate feature scoring or ranking method based on the nature of the data and the problem you are addressing. Common scoring methods include correlation coefficient, information gain, chi-square test, mutual information, or statistical tests like t-test or ANOVA.  
Calculate the score or rank for each feature based on its relationship with the target variable.

**Feature Selection Techniques:**

Filter-based methods: Select features based on their individual scores or rankings. Set a threshold and select features above that threshold.  
Wrapper-based methods: Use a machine learning model with different feature subsets and evaluate their performance to determine the optimal subset. Techniques like forward selection, backward elimination, or recursive feature elimination fall under this category.  
Embedded methods: Incorporate feature selection as part of the model training process. Algorithms like Lasso (L1 regularization) and Ridge (L2 regularization) can automatically select relevant features during model training.

**Evaluate Subset Performance:**

Split the dataset into training and validation sets.  
Train a machine learning model using the selected subset of features.  
Evaluate the model’s performance metrics (e.g., accuracy, precision, recall, F1-score, or area under the ROC curve) on the validation set.  
If the performance is not satisfactory, go back to the feature selection step and try different techniques or adjust the parameters

.  
**Iterative Refinement:**

Iterate through steps 2-4, trying different feature scoring methods, selection techniques, thresholds, or algorithms to find the optimal feature subset.  
Utilize cross-validation techniques, such as k-fold cross-validation, to obtain more robust estimates of model performance and feature relevance.  
Final Model Training and Testing:

Once the optimal feature subset is selected, train the final machine learning model on the entire training dataset using the selected features.  
Evaluate the model’s performance on an independent test dataset to assess its generalization ability.

**Interpretability and Validation:**

Analyze the selected features and their relationship with the target variable to gain insights into the problem domain.  
Validate the selected features on new, unseen data to ensure their robustness and effectiveness.  
Remember that the feature subset selection process is iterative and may require experimenting with different techniques, thresholds, and evaluation metrics to find the most suitable subset of features for your specific problem.

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses)

**Unit-3**

# **1.Probability for Machine Learning**

Probability and statistics both are the most important concepts for Machine Learning. Probability is about predicting the likelihood of future events, while statistics involves the analysis of the frequency of past events.

Nowadays, Machine Learning has become one of the first choices for most freshers and IT professionals. But, in order to enter this field, one must have some pre-specified skills and one of those skills in Mathematics. Yes, Mathematics is very much important to learn ML technology and develop efficient applications for the business. When talking about mathematics for Machine Learning, it especially focuses on Probability and Statistics, which are the essential topics to get started with ML. Probability and statistics are considered as the base foundation for ML and data science to develop ML algorithms and build decision-making capabilities. Also, Probability and statistics are the primary prerequisites to learn ML.

In this topic, we will discuss a few important books on Probability and statistics that help you in making the ML process easy and implementing algorithms to business scenarios too. Here, we will discuss some of the best books for Probability and Statistics from basic to advanced levels.

## Probability in Machine Learning

Probability is the bedrock of ML, which tells how likely is the event to occur. The value of Probability always lies between 0 to 1. It is the core concept as well as a primary prerequisite to understanding the ML models and their applications.

**Probability can be calculated by the number of times the event occurs divided by the total number of possible outcomes**. Let's suppose we tossed a coin, then the probability of getting head as a possible outcome can be calculated as below formula:

P (H) = Number of ways to head occur/ total number of possible outcomes

P (H) = ½

P (H) = 0.5

Where;

P (H) = Probability of occurring Head as outcome while tossing a coin.

## Types of Probability

For better understanding the Probability, it can be categorized further in different types as follows:

**Empirical Probability:** Empirical Probability can be calculated as the number of times the event occurs divided by the total number of incidents observed.

**Theoretical Probability:**Theoretical Probability can be calculated as the number of ways the particular event can occur divided by the total number of possible outcomes.

**Joint Probability:**It tells the Probability of simultaneously occurring two random events.

P(A ∩ B) = P(A). P(B)

Where;

P(A ∩ B) = Probability of occurring events A and B both.

P (A) = Probability of event A

P (B) = Probability of event B

**Conditional Probability:**It is given by the Probability of event A given that event B occurred.

The Probability of an event A conditioned on an event B is denoted and defined as;

P(A|B) = P(A∩B)/P(B)

Similarly, P(B|A) = P(A ∩ B)/ P(A) . We can write the joint Probability of as A and B as P(A ∩ B)= p(A).P(B|A), which means: "The chance of both things happening is the chance that the first one happens, and then the second one is given when the first thing happened."

We have a basic understanding of Probability required to learn Machine Learning. Now, we will discuss the basic introduction of Statistics for ML.

## 2.What is Machine Learning Inference

Inference in machine learning (ML) is the method of applying an ML model to a dataset and producing an output or “prediction.” This output could be a number score, image, or text. So any kind of organized or unstructured data.

An ML model is often software code that implements a mathematical method. The ML inference process places this code in a production environment, allowing it to create predictions based on input from real end users.

The ML life span is divided into two parts:

* **The training** step entails developing a ML, learning it by executing it on data sets examples, and then evaluating and confirming the model on unseen instances.
* **Inference Machine learning**entails running the model on real data to get actionable results. During this stage, the inference system takes end-user inputs, analyzes the information, feeds it into the model, and returns outputs to users.

**TESTING. CI/CD. MONITORING.**

## How does it work?

In addition to the model, 3 key components are required to construct an ML inference environment:

* **Data sources** – A data source is often a system that collects live data from the mechanism that creates the data. A data source might, for example, be a cluster that stores data. A data source might also be a simple web application that captures user clicks and provides data to the server that contains the ML model.
* **Host system** – The ML model’s host system takes data from sources and feeds it all into the model. The infrastructure required to transform the inference code in machine learning into a completely running application is provided by the host system. After the ML model generates an output, the host system delivers that production to the data endpoints.
* **Data destinations**– are the locations to which the host system should send the ML model’s output score. An endpoint can be any form of data storage, from which downstream applications act on the scores.

**Causal Inference in Machine Learning**

The goal of causal inference is to establish whether or not the intervention will be effective. An ML model can only tell you whether there is a relationship between two variables.

The message here is that if all you need to do is make accurate predictions, causal inference is irrelevant. However, if you wish to act on those projections or another component of the model, you will need some sort of causal model.

**Statistical Inference vs Machine Learning**

The definition of learning and inference varies according to the topic of research. Confusion frequently occurs when the terms are used carelessly without regard for a specific profession.

At the broader level, we are all aware of the term “inference.” We notice some data and wish to get knowledge from it.

* **The inference is the process of examining facts and drawing information from them**.

When statisticians discuss inference, they generally refer to statistical inference. During statistical inference, we see some data and want to say something about the process that created that data. As a result, the statistical inference would include predictions, predicting error margins, testing hypotheses, and parameter estimation.

Traditional ML experts from a technical background, on the other hand, frequently distinguish between inference and learning. Learning is connected with model parameters and is not considered an inference issue directly. As a result, a statistician’s understanding of the term “inference” is limited. The inference is commonly believed to as making a prediction.

Making a separation across learning and inference has the benefit of automatically separating machine learning algorithms from inference algorithms. Although certain parameters may be determined analytically for particular cases, most require a learning technique.

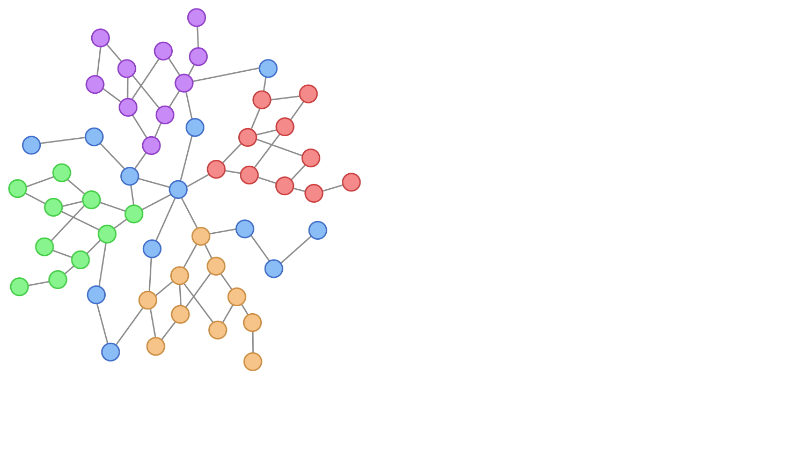
Similarly, in other inference issues, the prediction is typically not a plug-and-chug operation and must be computed using an inference process. Things get much more fascinating in latent variable models, where an inference method is frequently layered within a learning process.

To recap, the distinction between learning and inference is determined by the modeler’s sight. Parameter and learning estimation is a sort of inference if you think like a statistician. Learning is generally parameter estimate and inference is usually prediction if you think like a classic machine learning researcher. Different points of view are important in various situations

3.Independent and Identically Distributed (IID) in Machine Learning

**Introduction**

In machine learning, the concept of independent and identically distributed (IID) plays a critical role in various aspects of data analysis, model training, and evaluation. IID assumptions are fundamental in ensuring the reliability and validity of many machine learning algorithms and statistical techniques. This essay explores the significance of IID in machine learning, its assumptions, and its implications on model development and performance.



**Understanding IID in Machine Learning**

In the context of machine learning, IID refers to the assumption that the training data used to build a model are independently and randomly sampled from the same underlying distribution. Each data point is assumed to be independent of others and follows the same distributional characteristics. This assumption enables the application of powerful statistical methods and learning algorithms that rely on the absence of systematic dependencies or biases within the data.

**Assumptions of IID in Machine Learning**

1. Independence: The independence assumption implies that the **occurrence**or value of one data point does not provide any information about the occurrence or value of another data point. It assumes that the data points are not influenced by each other and that there is no hidden structure or correlation among them. Violations of this assumption can lead to biased or unreliable model predictions.
2. **Identical Distribution**: The identical distribution assumption assumes that the data points are drawn from the same underlying distribution. It implies that the statistical properties, such as mean, variance, and other distributional characteristics, remain consistent across the entire dataset. Deviations from this assumption can introduce sampling bias, causing models to generalize poorly to new, unseen data.

**Implications of IID in Machine Learning**

1. **Training and Evaluation**: IID assumptions are crucial during model training and evaluation. When the training data satisfies the IID assumption, machine learning algorithms can effectively learn the underlying patterns and make accurate predictions. Additionally, during model evaluation, IID allows for the use of cross-validation techniques and statistical tests, ensuring that the performance estimates are reliable and representative of the model’s true performance.
2. **Feature Selection and Engineering**: The IID assumption influences feature selection and engineering processes. If the independence assumption is violated, it is essential to identify and handle correlated or dependent features properly. Feature selection methods can help identify redundant or highly correlated features, while feature engineering techniques can transform or combine features to mitigate the impact of dependencies within the data.
3. **Regularization and Overfitting**: IID assumptions are closely tied to the problem of overfitting. When the data violates the IID assumption, models may tend to memorize or overfit to the specific patterns present in the training data, failing to generalize well to unseen data. Regularization techniques, such as L1 or L2 regularization, can help mitigate overfitting and improve the generalization performance of models.
4. **Statistical Inference and Hypothesis Testing**: IID assumptions are critical in statistical inference and hypothesis testing within machine learning. Statistical tests, such as t-tests or chi-square tests, assume that the data points are independently and identically distributed. Violations of the IID assumption can lead to inaccurate p-values, affecting the validity of statistical inferences and hypothesis testing results.

**Challenges and Considerations**

It is essential to recognize that the IID assumption may not hold in all real-world scenarios. Real-world datasets often exhibit complex dependencies, temporal correlations, or imbalanced distributions. When dealing with non-IID data, specialized techniques, such as time series analysis, sequence modeling, or techniques for handling imbalanced data, need to be employed to address these challenges appropriately.

In machine learning, the concept of independent and identically distributed (IID) is often assumed for the training and evaluation of models. While the data may not always strictly adhere to the IID assumption, it is a common starting point for many algorithms. Here’s an example of how you can create an IID dataset and train a simple machine learning model using Python:

# **4.Bayes Theorem in Machine learning**

Machine Learning is one of the most emerging technology of Artificial Intelligence. We are living in the 21th century which is completely driven by new technologies and gadgets in which some are yet to be used and few are on its full potential. Similarly, Machine Learning is also a technology that is still in its developing phase. There are lots of concepts that make machine learning a better technology such as supervised learning, unsupervised learning, reinforcement learning, perceptron models, Neural networks, etc. In this article "Bayes Theorem in Machine Learning", we will discuss another most important concept of Machine Learning theorem i.e., **Bayes Theorem**. But before starting this topic you should have essential understanding of this theorem such as what exactly is Bayes theorem, why it is used in Machine Learning, examples of Bayes theorem in Machine Learning and much more. So, let's start the brief introduction of Bayes theorem.

## Introduction to Bayes Theorem in Machine Learning

Bayes theorem is given by an English statistician, philosopher, and Presbyterian minister named **Mr. Thomas Bayes** in 17th century. Bayes provides their thoughts in decision theory which is extensively used in important mathematics concepts as Probability. Bayes theorem is also widely used in Machine Learning where we need to predict classes precisely and accurately. An important concept of Bayes theorem named **Bayesian method** is used to calculate conditional probability in Machine Learning application that includes classification tasks. Further, a simplified version of Bayes theorem (Naïve Bayes classification) is also used to reduce computation time and average cost of the projects.

Bayes theorem is also known with some other name such as **Bayes rule or Bayes Law. Bayes theorem helps to determine the probability of an event with random knowledge**. It is used to calculate the probability of occurring one event while other one already occurred. It is a best method to relate the condition probability and marginal probability.

In simple words, we can say that Bayes theorem helps to contribute more accurate results.

Bayes Theorem is used to estimate the precision of values and provides a method for calculating the conditional probability. However, it is hypocritically a simple calculation but it is used to easily calculate the conditional probability of events where intuition often fails. Some of the data scientist assumes that Bayes theorem is most widely used in financial industries but it is not like that. Other than financial, Bayes theorem is also extensively applied in health and medical, research and survey industry, aeronautical sector, etc.

## What is Bayes Theorem?

Bayes theorem is one of the most popular machine learning concepts that helps to calculate the probability of occurring one event with uncertain knowledge while other one has already occurred.

Bayes' theorem can be derived using product rule and conditional probability of event X with known event Y:

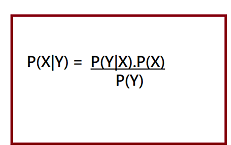
* According to the product rule we can express as the probability of event X with known event Y as follows;

1. P(X ? Y)= P(X|Y) P(Y)       {equation 1}

* Further, the probability of event Y with known event X:

1. P(X ? Y)= P(Y|X) P(X)       {equation 2}

Mathematically, Bayes theorem can be expressed by combining both equations on right hand side. We will get:



Here, both events X and Y are independent events which means probability of outcome of both events does not depends one another.

The above equation is called as Bayes Rule or Bayes Theorem.

* P(X|Y) is called as **posterior**, which we need to calculate. It is defined as updated probability after considering the evidence.
* P(Y|X) is called the likelihood. It is the probability of evidence when hypothesis is true.
* P(X) is called the **prior probability**, probability of hypothesis before considering the evidence
* P(Y) is called marginal probability. It is defined as the probability of evidence under any consideration.

Hence, Bayes Theorem can be written as:

**posterior = likelihood \* prior / evidence**

## Prerequisites for Bayes Theorem

While studying the Bayes theorem, we need to understand few important concepts. These are as follows:

**1. Experiment**

An experiment is defined as the planned operation carried out under controlled condition such as tossing a coin, drawing a card and rolling a dice, etc.

**2. Sample Space**

During an experiment what we get as a result is called as possible outcomes and the set of all possible outcome of an event is known as sample space. For example, if we are rolling a dice, sample space will be:

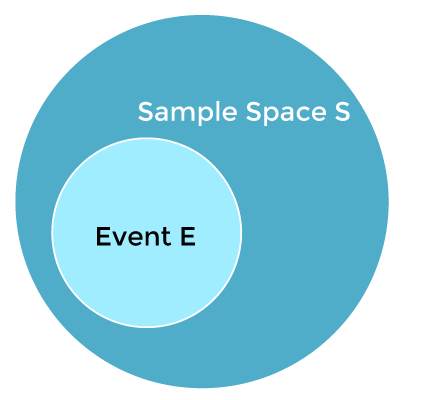
S1 = {1, 2, 3, 4, 5, 6}

Similarly, if our experiment is related to toss a coin and recording its outcomes, then sample space will be:

S2 = {Head, Tail}

**3. Event**

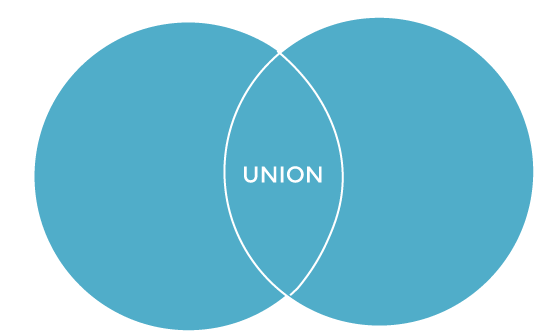
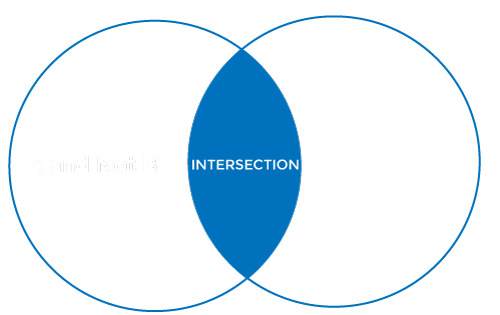
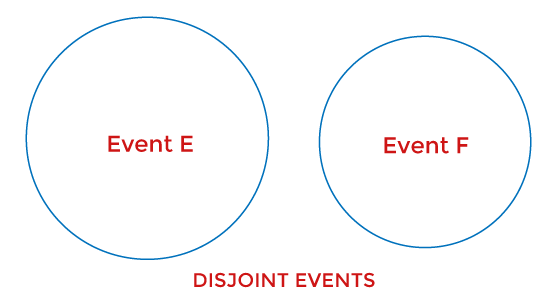
Event is defined as subset of sample space in an experiment. Further, it is also called as set of outcomes.



Assume in our experiment of rolling a dice, there are two event A and B such that;

A = Event when an even number is obtained = {2, 4, 6}

B = Event when a number is greater than 4 = {5, 6}

* **Probability of the event A ''P(A)''**= Number of favourable outcomes / Total number of possible outcomes  
  P(E) = 3/6 =1/2 =0.5
* Similarly, **Probability of the event B ''P(B)''**= Number of favourable outcomes / Total number of possible outcomes  
  =2/6  
  =1/3  
  =0.333
* **Union of event A and B:**  
  A∪B = {2, 4, 5, 6}  
  
* **Intersection of event A and B:**  
  A∩B= {6}  
  
* **Disjoint Event:** If the intersection of the event A and B is an empty set or null then such events are known as **disjoint event** or **mutually exclusive events** also.  
  

**4. Random Variable:**

It is a real value function which helps mapping between sample space and a real line of an experiment. A random variable is taken on some random values and each value having some probability. However, it is neither random nor a variable but it behaves as a function which can either be discrete, continuous or combination of both.

**5. Exhaustive Event:**

As per the name suggests, a set of events where at least one event occurs at a time, called exhaustive event of an experiment.

Thus, two events A and B are said to be exhaustive if either A or B definitely occur at a time and both are mutually exclusive for e.g., while tossing a coin, either it will be a Head or may be a Tail.

**6. Independent Event:**

Two events are said to be independent when occurrence of one event does not affect the occurrence of another event. In simple words we can say that the probability of outcome of both events does not depends one another.

Mathematically, two events A and B are said to be independent if:

P(A ∩ B) = P(AB) = P(A)\*P(B)

**7. Conditional Probability:**

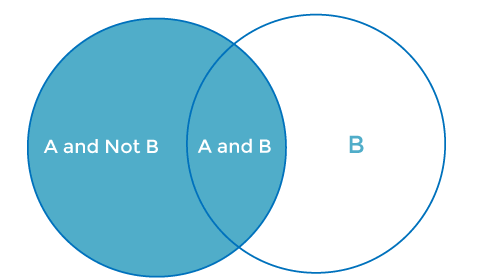
Conditional probability is defined as the probability of an event A, given that another event B has already occurred (i.e. A conditional B). This is represented by P(A|B) and we can define it as:

P(A|B) = P(A ∩ B) / P(B)

**8. Marginal Probability:**

Marginal probability is defined as the probability of an event A occurring independent of any other event B. Further, it is considered as the probability of evidence under any consideration.

P(A) = P(A|B)\*P(B) + P(A|~B)\*P(~B)



Here ~B represents the event that B does not occur.

## How to apply Bayes Theorem or Bayes rule in Machine Learning?

Bayes theorem helps us to calculate the single term P(B|A) in terms of P(A|B), P(B), and P(A). This rule is very helpful in such scenarios where we have a good probability of P(A|B), P(B), and P(A) and need to determine the fourth term.

Naïve Bayes classifier is one of the simplest applications of Bayes theorem which is used in classification algorithms to isolate data as per accuracy, speed and classes.

Let's understand the use of Bayes theorem in machine learning with below example.

Suppose, we have a vector A with I attributes. It means

A = A1, A2, A3, A4……………Ai

Further, we have n classes represented as C1, C2, C3, C4…………Cn.

These are two conditions given to us, and our classifier that works on Machine Language has to predict A and the first thing that our classifier has to choose will be the best possible class. So, with the help of Bayes theorem, we can write it as:

P(Ci/A)= [ P(A/Ci) \* P(Ci)] / P(A)

Here;

P(A) is the condition-independent entity.

P(A) will remain constant throughout the class means it does not change its value with respect to change in class. To maximize the P(Ci/A), we have to maximize the value of term P(A/Ci) \* P(Ci).

With n number classes on the probability list let's assume that the possibility of any class being the right answer is equally likely. Considering this factor, we can say that:

P(C1)=P(C2)-P(C3)=P(C4)=…..=P(Cn).

This process helps us to reduce the computation cost as well as time. This is how Bayes theorem plays a significant role in Machine Learning and Naïve Bayes theorem has simplified the conditional probability tasks without affecting the precision. Hence, we can conclude that:

P(Ai/C)= P(A1/C)\* P(A2/C)\* P(A3/C)\*……\*P(An/C)

Hence, by using Bayes theorem in Machine Learning we can easily describe the possibilities of smaller events.

## What is Naïve Bayes Classifier in Machine Learning

Naïve Bayes theorem is also a supervised algorithm, which is based on Bayes theorem and used to solve classification problems. It is one of the most simple and effective classification algorithms in Machine Learning which enables us to build various ML models for quick predictions. It is a probabilistic classifier that means it predicts on the basis of probability of an object. Some popular Naïve Bayes algorithms are **spam filtration, Sentimental analysis, and classifying articles.**

### **Advantages of Naïve Bayes Classifier in Machine Learning:**

* It is one of the simplest and effective methods for calculating the conditional probability and text classification problems.
* A Naïve-Bayes classifier algorithm is better than all other models where assumption of independent predictors holds true.
* It is easy to implement than other models.
* It requires small amount of training data to estimate the test data which minimize the training time period.
* It can be used for Binary as well as Multi-class Classifications.

### **Disadvantages of Naïve Bayes Classifier in Machine Learning:**

The main disadvantage of using Naïve Bayes classifier algorithms is, it limits the assumption of independent predictors because it implicitly assumes that all attributes are independent or unrelated but in real life it is not feasible to get mutually independent attributes.

**Unit-4**

# Logistic Regression in Machine Learning

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class or not. It is a kind of statistical algorithm, which analyze the relationship between a set of independent variables and the dependent binary variables. It is a powerful tool for decision-making. For example email spam or not.

## Logistic Regression

Logistic regression is a [supervised machine learning](https://www.geeksforgeeks.org/supervised-unsupervised-learning/) algorithm mainly used for [classification](https://www.geeksforgeeks.org/getting-started-with-classification/) tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. it’s referred to as regression because it takes the output of the [linear regression](https://www.geeksforgeeks.org/ml-linear-regression/)function as input and uses a sigmoid function to estimate the probability for the given class. The [difference between linear regression and logistic regression](https://www.geeksforgeeks.org/ml-linear-regression-vs-logistic-regression/) is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

**Logistic Regression:**

It is used for predicting the categorical dependent variable using a given set of independent variables.

* Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value.
* It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
* Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems.
* In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts two maximum values (0 or 1).
* The curve from the logistic function indicates the likelihood of something such as whether the cells are cancerous or not, a mouse is obese or not based on its weight, etc.
* Logistic Regression is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.
* Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification.

**Logistic Function (Sigmoid Function):**

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

**Type of Logistic Regression:**

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

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

| **Sr.No** | **Linear Regresssion** | **Logistic Regression** |
| --- | --- | --- |
| **1** | Linear regression is used to predict the continuous dependent variable using a given set of independent variables. | Logistic regression is used to predict the categorical dependent variable using a given set of independent variables. |
| **2** | Linear regression is used for solving Regression problem. | It is used for solving classification problems. |
| **3** | In this we predict the value of continuous variables | In this we predict values of categorical varibles |
| **4** | In this we find best fit line. | In this we find S-Curve . |
| **5** | Least square estimation method is used for estimation of accuracy. | Maximum likelihood estimation method is used for Estimation of accuracy. |
| **6** | The output must be continuous value,such as price,age,etc. | Output is must be categorical value such as 0 or 1, Yes or no, etc. |
| **7** | It required linear relationship between dependent and independent variables. | It not required linear relationship. |
| **8** | There may be collinearity between the independent variables. | There should not be collinearity between independent varible. |

**Terminologies involved in Logistic Regression:**

Here are some common terms involved in logistic regression:

* **Independent variables:** The input characteristics or predictor factors applied to the dependent variable’s predictions.
* **Dependent variable:** The target variable in a logistic regression model, which we are trying to predict.
* **Logistic function:** The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.
* **Odds:**It is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur.
* **Log-odds:**The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.
* **Coefficient:**The logistic regression model’s estimated parameters, show how the independent and dependent variables relate to one another.
* **Intercept:**A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.
* [**Maximum likelihood estimation**](https://www.geeksforgeeks.org/probability-density-estimation-maximum-likelihood-estimation/)**:** The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model.

## How does Logistic Regression work?

The logistic regression model transforms the [linear regression](https://www.geeksforgeeks.org/ml-linear-regression/) function continuous value output into categorical value output using a sigmoid function, which maps any real-valued set of independent variables input into a value between 0 and 1. This function is known as the logistic function.

Let the independent input features be

 and the dependent variable is Y having only binary value i.e. 0 or 1.

then apply the multi-linear function to the input variables X

Here  is the ith observation of X,  is the weights or Coefficient, and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias.

whatever we discussed above is the [linear regression](https://www.geeksforgeeks.org/ml-linear-regression/).

### Sigmoid Function

Now we use the [sigmoid function](https://www.geeksforgeeks.org/derivative-of-the-sigmoid-function/) where the input will be z and we find the probability between 0 and 1. i.e predicted y.



*Sigmoid function*

As shown above, the figure sigmoid function converts the continuous variable data into the [probability](https://www.geeksforgeeks.org/probability-gq/) i.e. between 0 and 1.

* tends towards 1 as
* tends towards 0 as
* is always bounded between 0 and 1

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

## Logistic Regression Equation

The odd is the ratio of something occurring to something not occurring. it is different from probability as the probability is the ratio of something occurring to everything that could possibly occur. so odd will be

Applying natural log on odd. then log odd will be

then the final logistic regression equation will be:

## Likelihood function for Logistic Regression

The predicted probabilities will p(X;b,w) = p(x) for y=1 and for y = 0 predicted probabilities will 1-p(X;b,w) = 1-p(x)

Taking natural logs on both sides

## Gradient of the log-likelihood function

To find the maximum likelihood estimates, we differentiate w.r.t w,

## Assumptions for Logistic Regression

The assumptions for Logistic regression are as follows:

* **Independent observations:**Each observation is independent of the other. meaning there is no correlation between any input variables.
* **Binary dependent variables:**It takes the assumption that the dependent variable must be binary or dichotomous, meaning it can take only two values. For more than two categories softmax functions are used.
* **Linearity relationship between independent variables and log odds:** The relationship between the independent variables and the log odds of the dependent variable should be linear.
* **No outliers:** There should be no outliers in the dataset.
* **Large sample size:**The sample size is sufficiently large

## Types of Logistic Regression

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

### **Binomial Logistic regression:**

target variable can have only 2 possible types: “0” or “1” which may represent “win” vs “loss”, “pass” vs “fail”, “dead” vs “alive”, etc., in this case, sigmoid functions are used, which is already discussed above.

* Python3

|  |
| --- |
| # import the necessary libraries  **from** sklearn.datasets **import** load\_breast\_cancer  **from** sklearn.linear\_model **import** LogisticRegression  **from** sklearn.model\_selection **import** train\_test\_split  **from** sklearn.metrics **import** accuracy\_score  # load the breast cancer dataset  X, y **=** load\_breast\_cancer(return\_X\_y**=**True)  # split the train and test dataset  X\_train, X\_test,\      y\_train, y\_test **=** train\_test\_split(X, y,                                         test\_size**=**0.20,                                         random\_state**=**23)  # LogisticRegression  clf **=** LogisticRegression(random\_state**=**0)  clf.fit(X\_train, y\_train)  # Prediction  y\_pred **=** clf.predict(X\_test)    acc **=** accuracy\_score(y\_test, y\_pred)  **print**("Logistic Regression model accuracy (in %):", acc**\***100) |

**Output**:

Logistic Regression model accuracy (in %): 95.6140350877193

### **Multinomial Logistic Regression**

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

In this case, the softmax function is used in place of the sigmoid function. [Softmax function](https://www.geeksforgeeks.org/understanding-activation-functions-in-depth/)for K classes will be:

Then the probability will be:

In Multinomial Logistic Regression, the output variable can have **more than two possible discrete outputs**. Consider the Digit Dataset.

* Python3

|  |
| --- |
| **from** sklearn.model\_selection **import** train\_test\_split  **from** sklearn **import** datasets, linear\_model, metrics    # load the digit dataset  digits **=** datasets.load\_digits()    # defining feature matrix(X) and response vector(y)  X **=** digits.data  y **=** digits.target    # splitting X and y into training and testing sets  X\_train, X\_test,\      y\_train, y\_test **=** train\_test\_split(X, y,                                         test\_size**=**0.4,                                         random\_state**=**1)    # create logistic regression object  reg **=** linear\_model.LogisticRegression()    # train the model using the training sets  reg.fit(X\_train, y\_train)    # making predictions on the testing set  y\_pred **=** reg.predict(X\_test)    # comparing actual response values (y\_test)  # with predicted response values (y\_pred)  **print**("Logistic Regression model accuracy(in %):",        metrics.accuracy\_score(y\_test, y\_pred)**\***100) |

**Output:**

Logistic Regression model accuracy(in %): 96.52294853963839

### **Ordinal Logistic Regression**

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

## Applying steps in logistic regression modeling:

The following are the steps involved in logistic regression modeling:

* **Define the problem:** Identify the dependent variable and independent variables and determine if the problem is a binary classification problem.
* **Data preparation:** Clean and preprocess the data, and make sure the data is suitable for logistic regression modeling.
* [**Exploratory Data Analysis (EDA)**](https://www.geeksforgeeks.org/what-is-exploratory-data-analysis/)**:**Visualize the relationships between the dependent and independent variables, and identify any outliers or anomalies in the data.
* [**Feature Selection**](https://www.geeksforgeeks.org/feature-selection-techniques-in-machine-learning/)**:** Choose the independent variables that have a significant relationship with the dependent variable, and remove any redundant or irrelevant features.
* [**Model Building**](https://www.geeksforgeeks.org/learning-model-building-scikit-learn-python-machine-learning-library/)**:**Train the logistic regression model on the selected independent variables and estimate the coefficients of the model.
* [**Model Evaluation**](https://www.geeksforgeeks.org/machine-learning-model-evaluation/)**:** Evaluate the performance of the logistic regression model using appropriate metrics such as [accuracy](https://www.geeksforgeeks.org/techniques-to-evaluate-accuracy-of-classifier-in-data-mining/), [precision, recall, F1-score](https://www.geeksforgeeks.org/precision-recall-and-f1-score-using-r/), or [AUC-ROC](https://www.geeksforgeeks.org/auc-roc-curve/).
* **Model improvement:** Based on the results of the evaluation, fine-tune the model by adjusting the independent variables, adding new features, or using regularization techniques to reduce overfitting.
* [**Model Deployment**](https://www.geeksforgeeks.org/python-model-deployment-using-tensorflow-serving/)**:** Deploy the logistic regression model in a real-world scenario and make predictions on new data.

## Logistic Regression Model Thresholding

Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.

The decision for the value of the threshold value is majorly affected by the values of [precision and recall.](https://www.geeksforgeeks.org/confusion-matrix-machine-learning/) Ideally, we want both precision and recall to be 1, but this seldom is the case.

In the case of a Precision-Recall tradeoff, we use the following arguments to decide upon the threshold:

1. **Low Precision/High Recall:** In applications where we want to reduce the number of false negatives without necessarily reducing the number of false positives, we choose a decision value that has a low value of Precision or a high value of Recall. For example, in a cancer diagnosis application, we do not want any affected patient to be classified as not affected without giving much heed to if the patient is being wrongfully diagnosed with cancer. This is because the absence of cancer can be detected by further medical diseases but the presence of the disease cannot be detected in an already rejected candidate.
2. **High Precision/Low Recall:** In applications where we want to reduce the number of false positives without necessarily reducing the number of false negatives, we choose a decision value that has a high value of Precision or a low value of Recall. For example, if we are classifying customers whether they will react positively or negatively to a personalized advertisement, we want to be absolutely sure that the customer will react positively to the advertisement because otherwise, a negative reaction can cause a loss of potential sales from the customer.

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses)

# **Classification Algorithm in Machine Learning**

As we know, the Supervised Machine Learning algorithm can be broadly classified into Regression and Classification Algorithms. In Regression algorithms, we have predicted the output for continuous values, but to predict the categorical values, we need Classification algorithms.

## What is the Classification Algorithm?

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, **Yes or No, 0 or 1, Spam or Not Spam, cat or dog,** etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.

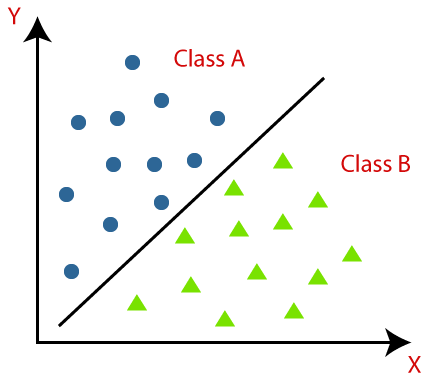
In classification algorithm, a discrete output function(y) is mapped to input variable(x).

1. y=f(x), where y = categorical output

The best example of an ML classification algorithm is **Email Spam Detector**.

The main goal of the Classification algorithm is to identify the category of a given dataset, and these algorithms are mainly used to predict the output for the categorical data.

Classification algorithms can be better understood using the below diagram. In the below diagram, there are two classes, class A and Class B. These classes have features that are similar to each other and dissimilar to other classes.



The algorithm which implements the classification on a dataset is known as a classifier. There are two types of Classifications:

* **Binary Classifier:** If the classification problem has only two possible outcomes, then it is called as Binary Classifier.  
  **Examples:** YES or NO, MALE or FEMALE, SPAM or NOT SPAM, CAT or DOG, etc.
* **Multi-class Classifier:** If a classification problem has more than two outcomes, then it is called as Multi-class Classifier.  
  **Example:** Classifications of types of crops, Classification of types of music.

## Learners in Classification Problems:

In the classification problems, there are two types of learners:

1. **Lazy Learners:** Lazy Learner firstly stores the training dataset and wait until it receives the test dataset. In Lazy learner case, classification is done on the basis of the most related data stored in the training dataset. It takes less time in training but more time for predictions.  
   **Example:** K-NN algorithm, Case-based reasoning
2. **Eager Learners:**Eager Learners develop a classification model based on a training dataset before receiving a test dataset. Opposite to Lazy learners, Eager Learner takes more time in learning, and less time in prediction. **Example:** Decision Trees, Naïve Bayes, ANN.

## Types of ML Classification Algorithms:

Classification Algorithms can be further divided into the Mainly two category:

* **Linear Models**
  + Logistic Regression
  + Support Vector Machines
* **Non-linear Models**
  + K-Nearest Neighbours
  + Kernel SVM
  + Naïve Bayes
  + Decision Tree Classification
  + Random Forest Classification

#### **Note: We will learn the above algorithms in later chapters.**

## Evaluating a Classification model:

Once our model is completed, it is necessary to evaluate its performance; either it is a Classification or Regression model. So for evaluating a Classification model, we have the following ways:

**1. Log Loss or Cross-Entropy Loss:**

* It is used for evaluating the performance of a classifier, whose output is a probability value between the 0 and 1.
* For a good binary Classification model, the value of log loss should be near to 0.
* The value of log loss increases if the predicted value deviates from the actual value.
* The lower log loss represents the higher accuracy of the model.
* For Binary classification, cross-entropy can be calculated as:

1. ?(ylog(p)+(1?y)log(1?p))

Where y= Actual output, p= predicted output.

**2. Confusion Matrix:**

* The confusion matrix provides us a matrix/table as output and describes the performance of the model.
* It is also known as the error matrix.
* The matrix consists of predictions result in a summarized form, which has a total number of correct predictions and incorrect predictions. The matrix looks like as below table:

|  |  |  |
| --- | --- | --- |
|  | **Actual Positive** | **Actual Negative** |
| Predicted Positive | True Positive | False Positive |
| Predicted Negative | False Negative | True Negative |

Classification Algorithm in Machine Learning

**3. AUC-ROC curve:**

* ROC curve stands for **Receiver Operating Characteristics Curve** and AUC stands for **Area Under the Curve**.
* It is a graph that shows the performance of the classification model at different thresholds.
* To visualize the performance of the multi-class classification model, we use the AUC-ROC Curve.
* The ROC curve is plotted with TPR and FPR, where TPR (True Positive Rate) on Y-axis and FPR(False Positive Rate) on X-axis.

## Use cases of Classification Algorithms

Classification algorithms can be used in different places. Below are some popular use cases of Classification Algorithms:

* Email Spam Detection
* Speech Recognition
* Identifications of Cancer tumor cells.
* Drugs Classification
* Biometric Identification, etc.

Representation: Feature Engineering

## Mapping Raw Data to Features

The left side of Figure 1 illustrates raw data from an input data source; the right side illustrates a **feature vector**, which is the set of floating-point values comprising the examples in your data set. **Feature engineering** means transforming raw data into a feature vector. Expect to spend significant time doing feature engineering.

Many machine learning models must represent the features as real-numbered vectors since the feature values must be multiplied by the model weights.

**Figure 1. Feature engineering maps raw data to ML features.**

### Mapping numeric values

Integer and floating-point data don't need a special encoding because they can be multiplied by a numeric weight. As suggested in Figure 2, converting the raw integer value 6 to the feature value 6.0 is trivial:

**Figure 2. Mapping integer values to floating-point values.**

### Mapping categorical values

[Categorical features](https://developers.google.com/machine-learning/glossary#categorical_data) have a discrete set of possible values. For example, there might be a feature called street\_name with options that include:

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Since models cannot multiply strings by the learned weights, we use feature engineering to convert strings to numeric values.

We can accomplish this by defining a mapping from the feature values, which we'll refer to as the **vocabulary** of possible values, to integers. Since not every street in the world will appear in our dataset, we can group all other streets into a catch-all "other" category, known as an **OOV (out-of-vocabulary) bucket**.

Using this approach, here's how we can map our street names to numbers:

* map Charleston Road to 0
* map North Shoreline Boulevard to 1
* map Shorebird Way to 2
* map Rengstorff Avenue to 3
* map everything else (OOV) to 4

However, if we incorporate these index numbers directly into our model, it will impose some constraints that might be problematic:

* We'll be learning a single weight that applies to all streets. For example, if we learn a weight of 6 for street\_name, then we will multiply it by 0 for Charleston Road, by 1 for North Shoreline Boulevard, 2 for Shorebird Way and so on. Consider a model that predicts house prices using street\_name as a feature. It is unlikely that there is a linear adjustment of price based on the street name, and furthermore this would assume you have ordered the streets based on their average house price. Our model needs the flexibility of learning different weights for each street that will be added to the price estimated using the other features.
* We aren't accounting for cases where street\_name may take multiple values. For example, many houses are located at the corner of two streets, and there's no way to encode that information in the street\_name value if it contains a single index.

To remove both these constraints, we can instead create a binary vector for each categorical feature in our model that represents values as follows:

* For values that apply to the example, set corresponding vector elements to 1.
* Set all other elements to 0.

The length of this vector is equal to the number of elements in the vocabulary. This representation is called a **one-hot encoding** when a single value is 1, and a **multi-hot encoding** when multiple values are 1.

Figure 3 illustrates a one-hot encoding of a particular street: Shorebird Way. The element in the binary vector for Shorebird Way has a value of 1, while the elements for all other streets have values of 0.

**Figure 3. Mapping street address via one-hot encoding.**

This approach effectively creates a Boolean variable for every feature value (e.g., street name). Here, if a house is on Shorebird Way then the binary value is 1 only for Shorebird Way. Thus, the model uses only the weight for Shorebird Way.

Similarly, if a house is at the corner of two streets, then two binary values are set to 1, and the model uses both their respective weights.

One-hot encoding extends to numeric data that you do not want to directly multiply by a weight, such as a postal code.

### Sparse Representation

Suppose that you had 1,000,000 different street names in your data set that you wanted to include as values for street\_name. Explicitly creating a binary vector of 1,000,000 elements where only 1 or 2 elements are true is a very inefficient representation in terms of both storage and computation time when processing these vectors. In this situation, a common approach is to use a [sparse representation](https://developers.google.com/machine-learning/glossary#sparse_representation) in which only nonzero values are stored. In sparse representations, an independent model weight is still learned for each feature value, as described above.

## Mapping Raw Data to Features

The left side of Figure 1 illustrates raw data from an input data source; the right side illustrates a **feature vector**, which is the set of floating-point values comprising the examples in your data set. **Feature engineering** means transforming raw data into a feature vector. Expect to spend significant time doing feature engineering.

Many machine learning models must represent the features as real-numbered vectors since the feature values must be multiplied by the model weights.

**Figure 1. Feature engineering maps raw data to ML features.**

### Mapping numeric values

Integer and floating-point data don't need a special encoding because they can be multiplied by a numeric weight. As suggested in Figure 2, converting the raw integer value 6 to the feature value 6.0 is trivial:

**Figure 2. Mapping integer values to floating-point values.**

### Mapping categorical values

[Categorical features](https://developers.google.com/machine-learning/glossary#categorical_data) have a discrete set of possible values. For example, there might be a feature called street\_name with options that include:

{'Charleston Road', 'North Shoreline Boulevard', 'Shorebird Way', 'Rengstorff Avenue'}

Since models cannot multiply strings by the learned weights, we use feature engineering to convert strings to numeric values.

We can accomplish this by defining a mapping from the feature values, which we'll refer to as the **vocabulary** of possible values, to integers. Since not every street in the world will appear in our dataset, we can group all other streets into a catch-all "other" category, known as an **OOV (out-of-vocabulary) bucket**.

Using this approach, here's how we can map our street names to numbers:

* map Charleston Road to 0
* map North Shoreline Boulevard to 1
* map Shorebird Way to 2
* map Rengstorff Avenue to 3
* map everything else (OOV) to 4

However, if we incorporate these index numbers directly into our model, it will impose some constraints that might be problematic:

* We'll be learning a single weight that applies to all streets. For example, if we learn a weight of 6 for street\_name, then we will multiply it by 0 for Charleston Road, by 1 for North Shoreline Boulevard, 2 for Shorebird Way and so on. Consider a model that predicts house prices using street\_name as a feature. It is unlikely that there is a linear adjustment of price based on the street name, and furthermore this would assume you have ordered the streets based on their average house price. Our model needs the flexibility of learning different weights for each street that will be added to the price estimated using the other features.
* We aren't accounting for cases where street\_name may take multiple values. For example, many houses are located at the corner of two streets, and there's no way to encode that information in the street\_name value if it contains a single index.

To remove both these constraints, we can instead create a binary vector for each categorical feature in our model that represents values as follows:

* For values that apply to the example, set corresponding vector elements to 1.
* Set all other elements to 0.

The length of this vector is equal to the number of elements in the vocabulary. This representation is called a **one-hot encoding** when a single value is 1, and a **multi-hot encoding** when multiple values are 1.

Figure 3 illustrates a one-hot encoding of a particular street: Shorebird Way. The element in the binary vector for Shorebird Way has a value of 1, while the elements for all other streets have values of 0.

**Figure 3. Mapping street address via one-hot encoding.**

This approach effectively creates a Boolean variable for every feature value (e.g., street name). Here, if a house is on Shorebird Way then the binary value is 1 only for Shorebird Way. Thus, the model uses only the weight for Shorebird Way.

Similarly, if a house is at the corner of two streets, then two binary values are set to 1, and the model uses both their respective weights.

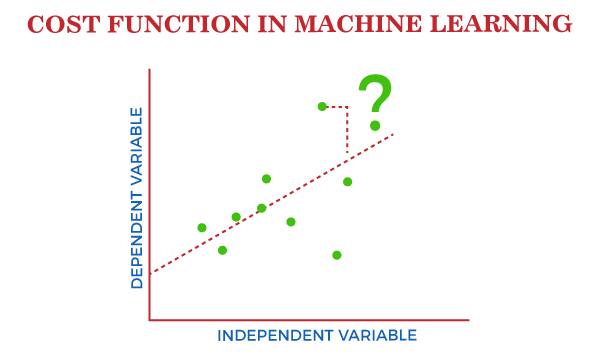
One-hot encoding extends to numeric data that you do not want to directly multiply by a weight, such as a postal code.

### Sparse Representation

Suppose that you had 1,000,000 different street names in your data set that you wanted to include as values for street\_name. Explicitly creating a binary vector of 1,000,000 elements where only 1 or 2 elements are true is a very inefficient representation in terms of both storage and computation time when processing these vectors. In this situation, a common approach is to use a [sparse representation](https://developers.google.com/machine-learning/glossary#sparse_representation) in which only nonzero values are stored. In sparse representations, an independent model weight is still learned for each feature value, as described above.

# **Cost Function in Machine Learning**

A Machine Learning model should have a very high level of accuracy in order to perform well with real-world applications. But how to calculate the accuracy of the model, i.e., how good or poor our model will perform in the real world? In such a case, the Cost function comes into existence. It is an important machine learning parameter to correctly estimate the model.



Cost function also plays a crucial role in understanding that how well your model estimates the relationship between the input and output parameters.

In this topic, we will explain the cost function in Machine Learning, Gradient descent, and types of cost functions.

## What is Cost Function?

**A cost function is an important parameter that determines how well a machine learning model performs for a given dataset.** It calculates the difference between the expected value and predicted value and represents it as a single real number.

PlayNext

Unmute

Current TimeÂ 0:00

/

DurationÂ 18:10

Loaded: 0.37%

Â

Fullscreen

Backward Skip 10sPlay VideoForward Skip 10s

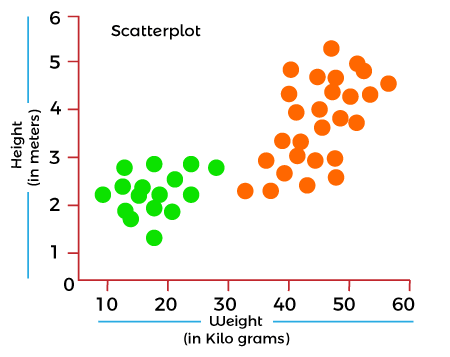
In machine learning, once we train our model, then we want to see how well our model is performing. Although there are various accuracy functions that tell you how your model is performing, but will not give insights to improve them. So, we need a function that can find when the model is most accurate by finding the spot between the undertrained and overtrained model.

In simple, "**Cost function is a measure of how wrong the model is in estimating the relationship between X(input) and Y(output) Parameter**." A cost function is sometimes also referred to as Loss function, and it can be estimated by iteratively running the model to compare estimated predictions against the known values of Y.

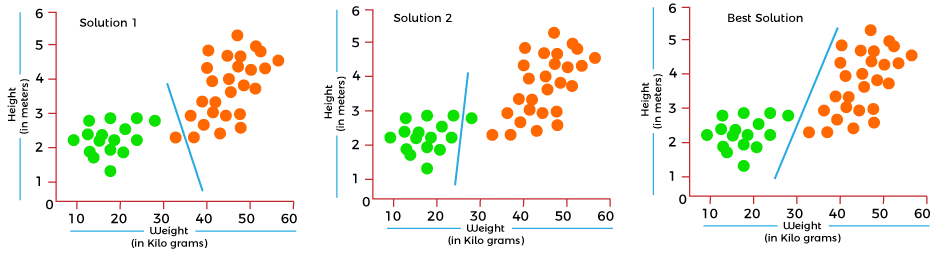
The main aim of each ML model is to determine parameters or weights that can minimize the cost function.

## Why use Cost Function?

While there are different accuracy parameters, then why do we need a Cost function for the Machine learning model. So, we can understand it with an example of the classification of data. Suppose we have a dataset that contains the height and weights of cats & dogs, and we need to classify them accordingly. If we plot the records using these two features, we will get a scatter plot as below:



In the above image, the green dots are cats, and the yellow dots are dogs. Below are the three possible solutions for this classification problem.



In the above solutions, all three classifiers have high accuracy, but the third solution is the best because it correctly classifies each datapoint. The reason behind the best classification is that it is in mid between both the classes, not close or not far to any of them.

To get such results, we need a Cost function. It means for getting the optimal solution; we need a Cost function. It calculated the difference between the actual values and predicted values and measured how wrong was our model in the prediction. By minimizing the value of the cost function, we can get the optimal solution.

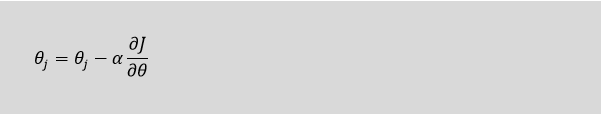
## Gradient Descent: Minimizing the cost function

As we discussed in the above section, the cost function tells how wrong your model is? And each machine learning model tries to minimize the cost function in order to give the best results. Here comes the role of Gradient descent.

"**Gradient Descent is an optimization algorithm which is used for optimizing the cost function or error in the model."** It enables the models to take the gradient or direction to reduce the errors by reaching to least possible error. Here direction refers to how model parameters should be corrected to further reduce the cost function. The error in your model can be different at different points, and you have to find the quickest way to minimize it, to prevent resource wastage.

Gradient descent is an iterative process where the model gradually converges towards a minimum value, and if the model iterates further than this point, it produces little or zero changes in the loss. This point is known as convergence, and at this point, the error is least, and the cost function is optimized.

Below is the equation for gradient descent in linear regression:



In the gradient descent equation, alpha is known as the learning rate. This parameter decides how fast you should move down to the slope. For large alpha, take big steps, and for small alpha value, you need to take small steps.

## Types of Cost Function

Cost functions can be of various types depending on the problem. However, mainly it is of three types, which are as follows:

1. Regression Cost Function
2. Binary Classification cost Functions
3. Multi-class Classification Cost Function.

### **1. Regression Cost Function**

Regression models are used to make a prediction for the continuous variables such as the price of houses, weather prediction, loan predictions, etc. When a cost function is used with Regression, it is known as the "Regression Cost Function." In this, the cost function is calculated as the error based on the distance, such as:

1. Error= Actual Output-Predicted output

There are three commonly used Regression cost functions, which are as follows:

**a. Means Error**

In this type of cost function, the error is calculated for each training data, and then the mean of all error values is taken.

It is one of the simplest ways possible.

The errors that occurred from the training data can be either negative or positive. While finding mean, they can cancel out each other and result in the zero-mean error for the model, so it is not recommended cost function for a model.

However, it provides a base for other cost functions of regression models.

**b. Mean Squared Error (MSE)**

Means Square error is one of the most commonly used Cost function methods. It improves the drawbacks of the Mean error cost function, as it calculates the square of the difference between the actual value and predicted value. Because of the square of the difference, it avoids any possibility of negative error.

The formula for calculating MSE is given below:

Cost Function in Machine Learning

Mean squared error is also known as L2 Loss.

In MSE, each error is squared, and it helps in reducing a small deviation in prediction as compared to MAE. But if the dataset has outliers that generate more prediction errors, then squaring of this error will further increase the error multiple times. Hence, we can say MSE is less robust to outliers.

**c. Mean Absolute Error (MAE)**

Mean Absolute error also overcome the issue of the Mean error cost function by taking the absolute difference between the actual value and predicted value.

The formula for calculating Mean Absolute Error is given below:

Cost Function in Machine Learning

This means the Absolute error cost function is also known as **L1 Loss**. It is not affected by noise or outliers, hence giving better results if the dataset has noise or outlier.

### **2. Binary Classification Cost Functions**

Classification models are used to make predictions of categorical variables, such as predictions for 0 or 1, Cat or dog, etc. The cost function used in the classification problem is known as the Classification cost function. However, the classification cost function is different from the Regression cost function.

One of the commonly used loss functions for classification is cross-entropy loss.

The binary Cost function is a special case of Categorical cross-entropy, where there is only one output class. For example, classification between red and blue.

To better understand it, let's suppose there is only a single output variable Y

1. Cross-entropy(D) = - y\*log(p) when y = 1
3. Cross-entropy(D) = - (1-y)\*log(1-p) when y = 0

The error in binary classification is calculated as the mean of cross-entropy for all N training data. Which means:

1. Binary Cross-Entropy = (Sum of Cross-Entropy for N data)/N

### **3. Multi-class Classification Cost Function**

A multi-class classification cost function is used in the classification problems for which instances are allocated to one of more than two classes. Here also, similar to binary class classification cost function, cross-entropy or categorical cross-entropy is commonly used cost function.

It is designed in a way that it can be used with multi-class classification with the target values ranging from 0 to 1, 3, ….,n classes.

In a multi-class classification problem, cross-entropy will generate a score that summarizes the mean difference between actual and anticipated probability distribution.

# **Gradient Descent in Machine Learning**

Gradient Descent is known as one of the most commonly used optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results. Further, gradient descent is also used to train Neural Networks.

In mathematical terminology, Optimization algorithm refers to the task of minimizing/maximizing an objective function f(x) parameterized by x. Similarly, in machine learning, optimization is the task of minimizing the cost function parameterized by the model's parameters. The main objective of gradient descent is to minimize the convex function using iteration of parameter updates. Once these machine learning models are optimized, these models can be used as powerful tools for Artificial Intelligence and various computer science applications.

In this tutorial on Gradient Descent in Machine Learning, we will learn in detail about gradient descent, the role of cost functions specifically as a barometer within Machine Learning, types of gradient descents, learning rates, etc.

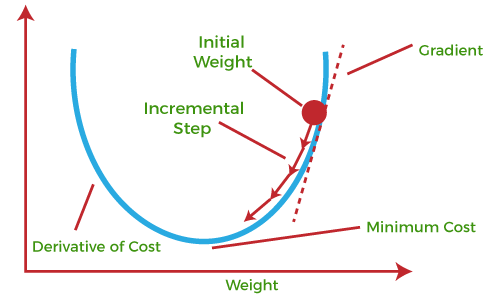
## What is Gradient Descent or Steepest Descent?

Gradient descent was initially discovered by **"Augustin-Louis Cauchy"** in mid of 18th century. **Gradient Descent is defined as one of the most commonly used iterative optimization algorithms of machine learning to train the machine learning and deep learning models. It helps in finding the local minimum of a function.**

Backward Skip 10sPlay VideoForward Skip 10s

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.

Hypothesis:

Parameters:

Cost function:

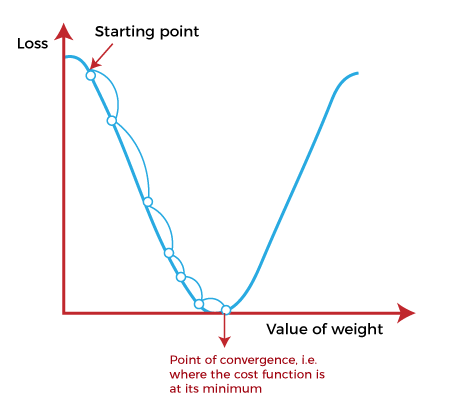
Goal:

### **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:

1. 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.**

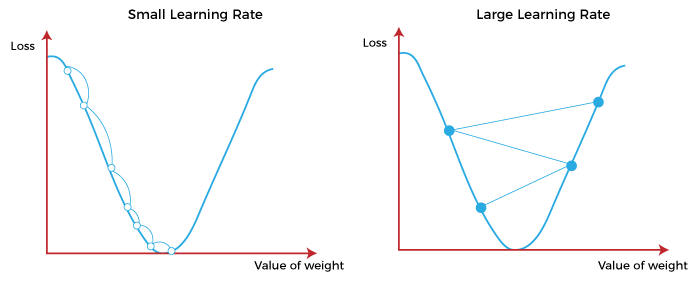
The main objective of gradient descent is to minimize the cost function or the error between expected and actual. To minimize the cost function, two data points are required:

* **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:**

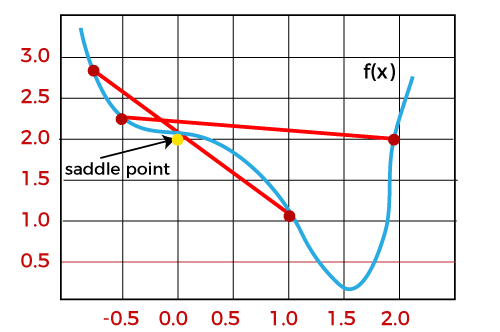
* It is easier to fit in allocated memory.
* It is computationally efficient.
* It produces stable gradient descent convergence.

## 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.

# **Regularization in Machine Learning**

## What is Regularization?

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it.

Sometimes the [machine learning](https://www.javatpoint.com/machine-learning) model performs well with the training data but does not perform well with the test data. It means the model is not able to predict the output when deals with unseen data by introducing noise in the output, and hence the model is called overfitted. This problem can be deal with the help of a regularization technique.

This technique can be used in such a way that it will allow to maintain all variables or features in the model by reducing the magnitude of the variables. Hence, it maintains accuracy as well as a generalization of the model.

It mainly regularizes or reduces the coefficient of features toward zero. In simple words, "In regularization technique, we reduce the magnitude of the features by keeping the same number of features."

### **How does Regularization Work?**

Regularization works by adding a penalty or complexity term to the complex model. Let's consider the simple linear regression equation:

y= β0+β1x1+β2x2+β3x3+⋯+βnxn +b

In the above equation, Y represents the value to be predicted

X1, X2, …Xn are the features for Y.

β0,β1,…..βn are the weights or magnitude attached to the features, respectively. Here represents the bias of the model, and b represents the intercept.

Linear regression models try to optimize the β0 and b to minimize the cost function. The equation for the cost function for the linear model is given below:

Regularization in Machine Learning

Now, we will add a loss function and optimize parameter to make the model that can predict the accurate value of Y. The loss function for the linear regression is called as **RSS or Residual sum of squares.**

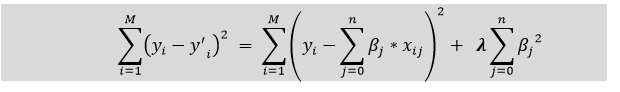
## Techniques of Regularization

There are mainly two types of regularization techniques, which are given below:

* **Ridge Regression**
* **Lasso Regression**

### **Ridge Regression**

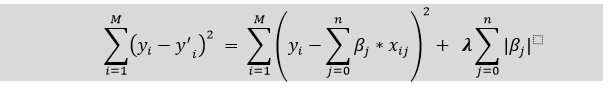
* Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.
* Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as **L2 regularization**.
* In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called **Ridge Regression penalty**. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
* The equation for the cost function in ridge regression will be:



* In the above equation, the penalty term regularizes the coefficients of the model, and hence ridge regression reduces the amplitudes of the coefficients that decreases the complexity of the model.
* As we can see from the above equation, if the values of **λ tend to zero, the equation becomes the cost function of the linear regression model.** Hence, for the minimum value of λ, the model will resemble the linear regression model.
* A general linear or polynomial regression will fail if there is high collinearity between the independent variables, so to solve such problems, Ridge regression can be used.
* It helps to solve the problems if we have more parameters than samples.

### **Lasso Regression:**

* Lasso regression is another regularization technique to reduce the complexity of the model. It stands for **Least Absolute and Selection Operator.**
* It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
* Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.
* It is also called as **L1 regularization.** The equation for the cost function of Lasso regression will be:



* Some of the features in this technique are completely neglected for model evaluation.
* Hence, the Lasso regression can help us to reduce the overfitting in the model as well as the feature selection.

### **Key Difference between Ridge Regression and Lasso Regression**

* **Ridge regression** is mostly used to reduce the overfitting in the model, and it includes all the features present in the model. It reduces the complexity of the model by shrinking the coefficients.
* **Lasso regression** helps to reduce the overfitting in the model as well as feature selection.

# **Overfitting in Machine Learning**

In the real world, the dataset present will never be clean and perfect. It means each dataset contains impurities, noisy data, outliers, missing data, or imbalanced data. Due to these impurities, different problems occur that affect the accuracy and the performance of the model. One of such problems is Overfitting in Machine Learning. Overfitting is a problem that a model can exhibit.

A statistical model is said to be overfitted if it can’t generalize well with unseen data.

Before understanding overfitting, we need to know some basic terms, which are:

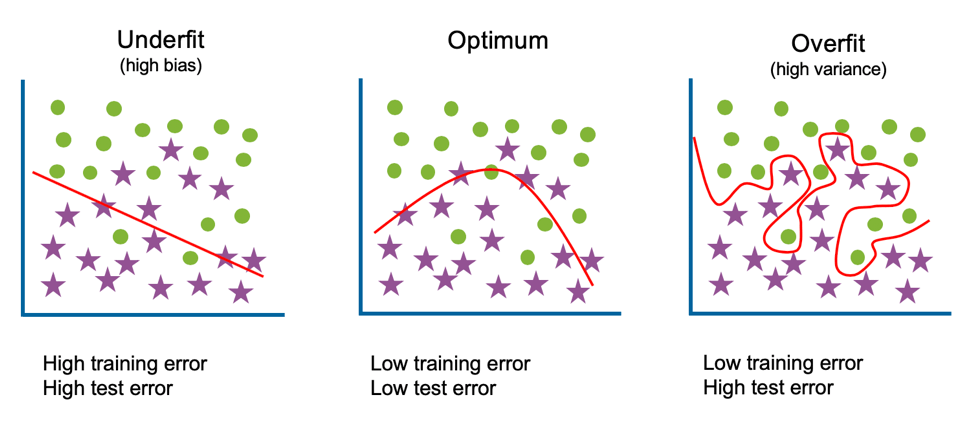
**Noise**: Noise is meaningless or irrelevant data present in the dataset. It affects the performance of the model if it is not removed.

**Bias**: Bias is a prediction error that is introduced in the model due to oversimplifying the machine learning algorithms. Or it is the difference between the predicted values and the actual values.

**Variance**: If the machine learning model performs well with the training dataset, but does not perform well with the test dataset, then variance occurs.

**Generalization**: It shows how well a model is trained to predict unseen data.

## What is Overfitting?



* Overfitting & underfitting are the two main errors/problems in the machine learning model, which cause poor performance in Machine Learning.
* Overfitting occurs when the model fits more data than required, and it tries to capture each and every datapoint fed to it. Hence it starts capturing noise and inaccurate data from the dataset, which degrades the performance of the model.
* An overfitted model doesn't perform accurately with the test/unseen dataset and can’t generalize well.
* An overfitted model is said to have low bias and high variance.

## Example to Understand Overfitting

We can understand overfitting with a general example. Suppose there are three students, X, Y, and Z, and all three are preparing for an exam. X has studied only three sections of the book and left all other sections. Y has a good memory, hence memorized the whole book. And the third student, Z, has studied and practiced all the questions. So, in the exam, X will only be able to solve the questions if the exam has questions related to section 3. Student Y will only be able to solve questions if they appear exactly the same as given in the book. Student Z will be able to solve all the exam questions in a proper way.

The same happens with machine learning; if the algorithm learns from a small part of the data, it is unable to capture the required data points and hence under fitted.

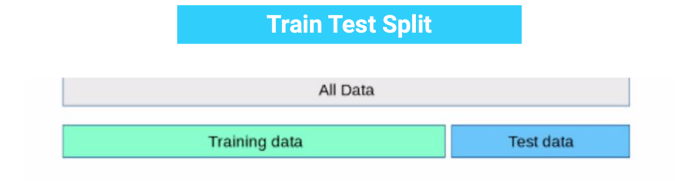
Suppose the model learns the training dataset, like the Y student. They perform very well on the seen dataset but perform badly on unseen data or unknown instances. In such cases, the model is said to be Overfitting.

And if the model performs well with the training dataset and also with the test/unseen dataset, similar to student Z, it is said to be a good fit.

## How to detect Overfitting?

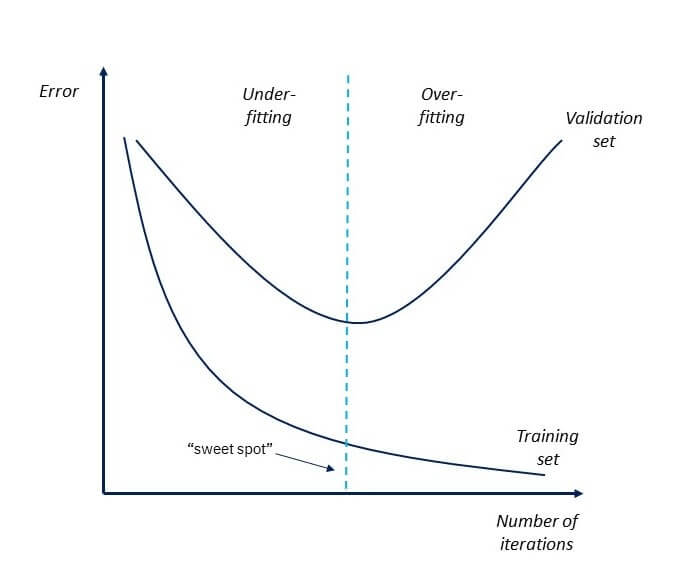
Overfitting in the model can only be detected once you test the data. To detect the issue, we can perform **Train/test split.**

In the train-test split of the dataset, we can divide our dataset into random test and training datasets. We train the model with a training dataset which is about 80% of the total dataset. After training the model, we test it with the test dataset, which is 20 % of the total dataset.



Now, if the model performs well with the training dataset but not with the test dataset, then it is likely to have an overfitting issue.

For example, if the model shows 85% accuracy with training data and 50% accuracy with the test dataset, it means the model is not performing well.



## Ways to prevent the Overfitting

Although overfitting is an error in Machine learning which reduces the performance of the model, however, we can prevent it in several ways. With the use of the linear model, we can avoid overfitting; however, many real-world problems are non-linear ones. It is important to prevent overfitting from the models. Below are several ways that can be used to prevent overfitting:

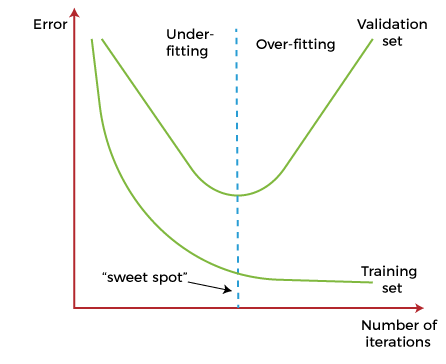
1. **Early Stopping**
2. **Train with more data**
3. **Feature Selection**
4. **Cross-Validation**
5. **Data Augmentation**
6. **Regularization**

### **Early Stopping**

In this technique, the training is paused before the model starts learning the noise within the model. In this process, while training the model iteratively, measure the performance of the model after each iteration. Continue up to a certain number of iterations until a new iteration improves the performance of the model.

After that point, the model begins to overfit the training data; hence we need to stop the process before the learner passes that point.

Stopping the training process before the model starts capturing noise from the data is known as **early stopping.**



However, this technique may lead to the underfitting problem if training is paused too early. So, it is very important to find that "sweet spot" between underfitting and overfitting.

### **Train with More data**

Increasing the training set by including more data can enhance the accuracy of the model, as it provides more chances to discover the relationship between input and output variables.

It may not always work to prevent overfitting, but this way helps the algorithm to detect the signal better to minimize the errors.

When a model is fed with more training data, it will be unable to overfit all the samples of data and forced to generalize well.

But in some cases, the additional data may add more noise to the model; hence we need to be sure that data is clean and free from in-consistencies before feeding it to the model.

### **Feature Selection**

While building the ML model, we have a number of parameters or features that are used to predict the outcome. However, sometimes some of these features are redundant or less important for the prediction, and for this feature selection process is applied. In the feature selection process, we identify the most important features within training data, and other features are removed. Further, this process helps to simplify the model and reduces noise from the data. Some algorithms have the auto-feature selection, and if not, then we can manually perform this process.

### **Cross-Validation**

Cross-validation is one of the powerful techniques to prevent overfitting.

In the general k-fold cross-validation technique, we divided the dataset into k-equal-sized subsets of data; these subsets are known as folds.

### **Data Augmentation**

Data Augmentation is a data analysis technique, which is an alternative to adding more data to prevent overfitting. In this technique, instead of adding more training data, slightly modified copies of already existing data are added to the dataset.

The data augmentation technique makes it possible to appear data sample slightly different every time it is processed by the model. Hence each data set appears unique to the model and prevents overfitting.

### **Regularization**

If overfitting occurs when a model is complex, we can reduce the number of features. However, overfitting may also occur with a simpler model, more specifically the Linear model, and for such cases, regularization techniques are much helpful.

Regularization is the most popular technique to prevent overfitting. It is a group of methods that forces the learning algorithms to make a model simpler. Applying the regularization technique may slightly increase the bias but slightly reduces the variance. In this technique, we modify the objective function by adding the penalizing term, which has a higher value with a more complex model.

The two commonly used regularization techniques are L1 Regularization and L2 Regularization.

### **Ensemble Methods**

In ensemble methods, prediction from different machine learning models is combined to identify the most popular result.

The most commonly used ensemble methods are **Bagging and Boosting.**

In bagging, individual data points can be selected more than once. After the collection of several sample datasets, these models are trained independently, and depending on the type of task-i.e., regression or classification-the average of those predictions is used to predict a more accurate result. Moreover, bagging reduces the chances of overfitting in complex models.

In boosting, a large number of weak learners arranged in a sequence are trained in such a way that each learner in the sequence learns from the mistakes of the learner before it. It combines all the weak learners to come out with one strong learner. In addition, it improves the predictive flexibility of simple models.

# What is Machine Perception?

Machine perception refers to the capability of machines to interpret and make sense of sensory information from the environment.

Machine perception refers to the capability of machines to interpret and make sense of sensory information from the environment. This information can include data obtained from sensors such as cameras, microphones, or other sensors. The machine then processes this data, analyzes it, and draws conclusions from it.

Machine perception plays a significant role in enabling machines to interact with the physical world, understand human behavior and communication, and make decisions based on sensory information. In essence, machine perception is the foundation of many technologies such as autonomous driving, computer vision, speech recognition, and natural language processing.

## Types of Machine Perception

There are various types of machine perception, including computer vision, speech recognition, natural language processing, and sensor fusion.

* [**Computer vision**](https://www.datacamp.com/tutorial/seeing-like-a-machine-a-beginners-guide-to-image-analysis-in-machine-learning) involves the use of computers to interpret and understand visual data from digital images or videos. This technology has several applications such as facial recognition, object detection, and tracking.
* **Speech recognition** involves the ability of a machine to understand and interpret spoken language. Speech recognition technology has various applications such as virtual assistants, dictation software, and customer service bots.
* [**Natural Language Processing**](https://www.datacamp.com/tutorial/tutorial-natural-language-processing)**(NLP)** enables computers to understand and interpret human language in a more nuanced way. NLP technology has several applications, including chatbots, automated customer service systems, and sentiment analysis.
* **Sensor fusion** involves the integration of data from multiple sensors, such as cameras and LIDAR, to create a more comprehensive understanding of the environment. This technology is particularly useful for autonomous vehicles, robotics, and drones.

## Examples of Real-World Machine Perception Applications

One of the earliest applications of machine perception was optical character recognition, developed by [**Emanuel Goldberg**](https://books.google.com.pk/books?id=InFxDgAAQBAJ&q=1914+Emanuel+Goldberg&pg=PA91&redir_esc=y#v=snippet&q=1914%20Emanuel%20Goldberg&f=false) in 1914 . His character recognition machine could read characters and convert them into standard telegraph code, demonstrating the potential for machines to perceive symbols and text. Since Goldberg's initial work, the field has advanced rapidly. Today, machine perception is used extensively in:

* **Autonomous Vehicles:** Machine perception is a critical technology for enabling autonomous vehicles to operate safely and efficiently. Autonomous vehicles use a combination of computer vision, LIDAR, and radar to perceive their surroundings and make decisions in real-time. For example, Tesla's Autopilot system uses machine perception to detect objects, lanes, and signs, and to make decisions based on this information.
* **Healthcare:** Machine perception technology is being used to diagnose diseases and conditions by analyzing medical images such as X-rays, CT scans, and MRIs. For example, [**Google's DeepMind**](https://www.deepmind.com/) AI system can diagnose eye diseases by analyzing retinal images with a high degree of accuracy.
* **Robotics:** Machine perception is essential for robots to understand their environment and interact with it effectively. For example, [**Boston Dynamics' Spot robot**](https://robots.ieee.org/robots/spotmini/) uses computer vision and sensor fusion to navigate through environments, avoid obstacles, and perform tasks such as inspecting and monitoring.
* **Security:** Machine perception is being used to improve security systems by analyzing video footage and detecting unusual behavior or objects. For example, AI-powered security cameras can recognize faces and identify individuals, detect intruders, and alert authorities to potential threats.

## How Machine Perception Works

Machine perception works by processing and analyzing sensory data using [**machine learning algorithms**](https://www.datacamp.com/blog/top-machine-learning-use-cases-and-algorithms). The process begins with the collection of data from various sensors, such as cameras, microphones, or other sensors. The data is then preprocessed to remove noise and enhance its quality.

Next, the preprocessed data is fed into machine learning algorithms, such as convolutional neural networks (CNNs), [**recurrent neural networks**](https://www.datacamp.com/courses/recurrent-neural-networks-rnn-for-language-modeling-in-python) (RNNs), or [**support vector machines**](https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python) (SVMs), which analyze the data and extract relevant features. These features are then used to make predictions or decisions based on the specific application of the machine perception technology.

For example, in computer vision applications, the machine learning algorithms analyze the visual data to detect objects, recognize faces, or track movement. In speech recognition applications, the algorithms analyze the audio data to transcribe speech, identify individual speakers, or perform voice commands.

## Limitations and Challenges of Machine Perception

While machine perception has the potential to revolutionize various industries and applications, there are still several limitations and challenges that need to be addressed. Here are a few examples:

### Limited Understanding of Context

Machine perception systems often struggle to understand the context in which they operate. For example, an image recognition system may identify an object in a photo, but it may not be able to understand the scene or the significance of the object in the context of the overall image.

### **Limited Data Availability**

Machine perception algorithms require large amounts of high-quality data to function effectively. However, in some cases, such data may not be available or may be difficult to collect. An example of this is within the development of autonomous vehicles. While there is a significant amount of data available on driving scenarios and road conditions, there may be limited data on rare or unusual situations such as extreme weather conditions or unexpected road obstacles. This can make it difficult for autonomous vehicles to accurately perceive and respond to these situations, potentially leading to safety concerns.

### **Biases in Data and Algorithms**

Machine perception systems can be biased due to the biases present in the data used to train them or in the algorithms themselves. This can lead to inaccurate or unfair predictions and decisions. An example of bias in algorithms is when facial recognition systems have been shown to have higher error rates for people with darker skin tones, due to the lack of diversity in the training data.

### **Security and Privacy Concerns**

Machine perception systems often collect and process sensitive data, which can raise concerns about security and privacy. Hackers or malicious actors could potentially access or misuse this data, leading to serious consequences. For instance, a machine perception system used in a hospital to monitor patient vitals could potentially be hacked, allowing unauthorized access to sensitive medical information and compromising patient privacy.

## What is the Future of Machine Perception?

Currently we have an excellent speech recognition model in [**OpenAI Whisper**](https://www.datacamp.com/tutorial/converting-speech-to-text-with-the-openAI-whisper-API), the best object detection algorithm in [**YOLOv7**](https://www.datacamp.com/blog/yolo-object-detection-explained), and the NLP platform [**HuggingFace**](https://www.datacamp.com/tutorial/an-introduction-to-using-transformers-and-hugging-face) which provides high-quality datasets and state-of-the-art models. So I believe the future of machine perception lies in multimodality, where advanced systems can process image, speech, and text inputs to provide a complete understanding of our surroundings.

We already have multimodal systems like DALLE-2, an image generation model that generates images from text prompts, and [**GPT-4**](https://www.datacamp.com/blog/what-we-know-gpt4), which can generate text from both images and text prompts. Expect significant developments in this space in the near future, with [**Google and OpenAI researching in the field of multimodal models.**](https://www.datacamp.com/blog/openai-google-ai-data-science)

In the future, these systems will process video and audio in real-time to enable enhanced analysis and pattern recognition. Furthermore, progress in [**agent-based**](https://www.datacamp.com/tutorial/introduction-to-ai-agents-autogpt-agentgpt-babyagi) systems could enable artificial general intelligence (AGI).

AGI systems will have human-level intelligence and the ability to perform any intellectual task from generating art to writing books using multiple sensory information.

# Introduction to Multi-Task Learning(MTL) for Deep Learning

Multi-Task Learning (MTL) is a type of machine learning technique where a model is trained to perform multiple tasks simultaneously. In deep learning, MTL refers to training a neural network to perform multiple tasks by sharing some of the network’s layers and parameters across tasks.

In MTL, the goal is to improve the generalization performance of the model by leveraging the information shared across tasks. By sharing some of the network’s parameters, the model can learn a more efficient and compact representation of the data, which can be beneficial when the tasks are related or have some commonalities.

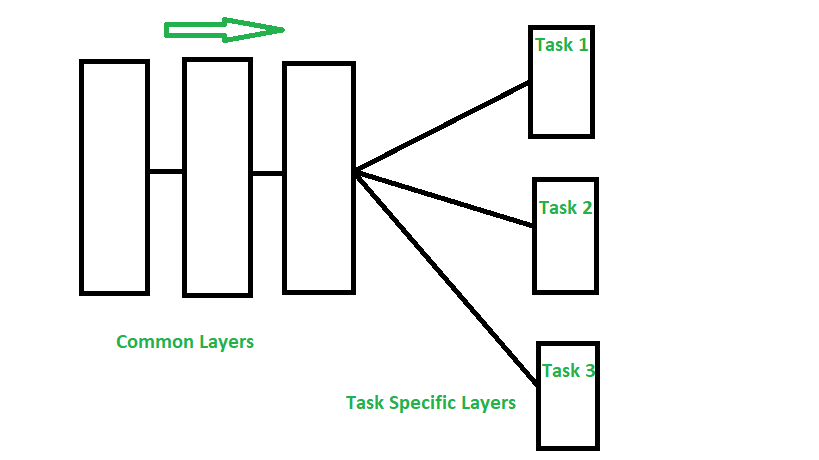
There are different ways to implement MTL in deep learning, but the most common approach is to use a shared feature extractor and multiple task-specific heads. The shared feature extractor is a part of the network that is shared across tasks and is used to extract features from the input data. The task-specific heads are used to make predictions for each task and are typically connected to the shared feature extractor.

Another approach is to use a shared decision-making layer, where the decision-making layer is shared across tasks, and the task-specific layers are connected to the shared decision-making layer.

MTL can be useful in many applications such as natural language processing, computer vision, and healthcare, where multiple tasks are related or have some commonalities. It is also useful when the data is limited, MTL can help to improve the generalization performance of the model by leveraging the information shared across tasks.

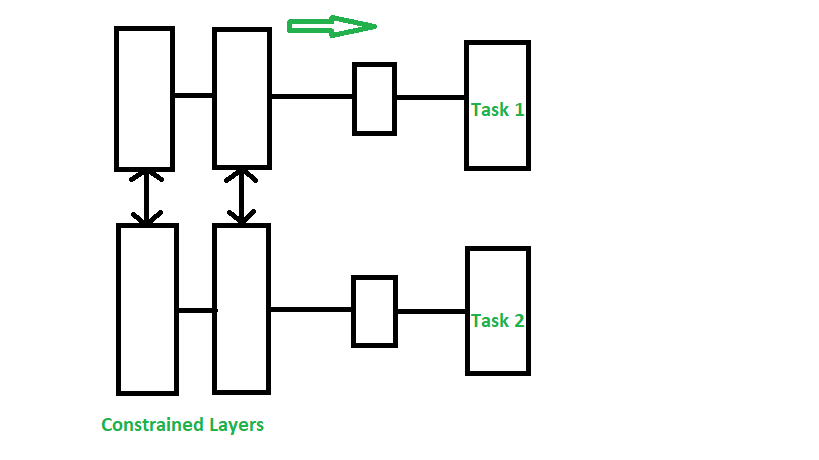
However, MTL also has its own limitations, such as when the tasks are very different

Multi-Task Learning is a sub-field of Deep Learning. It is recommended that you familiarize yourself with the concepts of [neural networks](https://www.geeksforgeeks.org/implementing-ann-training-process-in-python/) to understand what multi-task learning means. **What is Multi-Task Learning?** Multi-Task learning is a sub-field of Machine Learning that aims to solve multiple different tasks at the same time, by taking advantage of the similarities between different tasks. This can improve the learning efficiency and also act as a regularizer which we will discuss in a while. Formally, if there are **n** tasks (conventional deep learning approaches aim to solve just 1 task using 1 particular model), where these **n** tasks or a subset of them are related to each other but not exactly identical, Multi-Task Learning **(MTL)** will help in improving the learning of a particular model by using the knowledge contained in all the n tasks. **Intuition behind Multi-Task Learning (MTL):**By using Deep learning models, we usually aim to learn a good representation of the features or attributes of the input data to predict a specific value. Formally, we aim to optimize for a particular function by training a model and fine-tuning the hyperparameters till the performance can’t be increased further. By using MTL, it might be possible to increase performance even further by forcing the model to learn a more generalized representation as it learns (updates its weights) not just for one specific task but a bunch of tasks. Biologically, humans learn in the same way. We learn better if we learn multiple related tasks instead of focusing on one specific task for a long time. **MTL as a regularizer:** In the lingo of Machine Learning, MTL can also be looked at as a way of inducing bias. It is a form of inductive transfer, using multiple tasks induces a bias that prefers hypotheses that can explain all the **n** tasks. MTL acts as a regularizer by introducing inductive bias as stated above. It significantly reduces the risk of overfitting and also reduces the model’s ability to accommodate random noise during training. Now, let’s discuss the major and prevalent techniques to use MTL. **Hard Parameter Sharing –**A common hidden layer is used for all tasks but several task specific layers are kept intact towards the end of the model. This technique is very useful as by learning a representation for various tasks by a common hidden layer, we reduce the risk of overfitting.



*Hard Parameter Sharing*

**Soft Parameter Sharing –**Each model has their own sets of weights and biases and the distance between these parameters in different models is regularized so that the parameters become similar and can represent all the tasks.



*Soft Parameter Sharing*

**Assumptions and Considerations –**Using MTL to share knowledge among tasks are very useful only when the tasks are very similar, but when this assumption is violated, the performance will significantly decline. **Applications:** MTL techniques have found various uses, some of the major applications are-

* Object detection and Facial recognition
* Self Driving Cars: Pedestrians, stop signs and other obstacles can be detected together
* Multi-domain collaborative filtering for web applications
* Stock Prediction
* Language Modelling and other NLP applications

### Important points:

Here are some important points to consider when implementing Multi-Task Learning (MTL) for deep learning:

1. Task relatedness: MTL is most effective when the tasks are related or have some commonalities, such as natural language processing, computer vision, and healthcare.
2. Data limitation: MTL can be useful when the data is limited, as it allows the model to leverage the information shared across tasks to improve the generalization performance.
3. Shared feature extractor: A common approach in MTL is to use a shared feature extractor, which is a part of the network that is shared across tasks and is used to extract features from the input data.
4. Task-specific heads: Task-specific heads are used to make predictions for each task and are typically connected to the shared feature extractor.
5. Shared decision-making layer: another approach is to use a shared decision-making layer, where the decision-making layer is shared across tasks, and the task-specific layers are connected to the shared decision-making layer.
6. Careful architecture design: The architecture of MTL should be carefully designed to accommodate the different tasks and to make sure that the shared features are useful for all tasks.
7. Overfitting: MTL models can be prone to overfitting if the model is not regularized properly.
8. Avoiding negative transfer: when the tasks are very different or independent, MTL can lead to suboptimal performance compared to training a single-task model. Therefore, it is important to make sure that the shared features are useful for all tasks to avoid negative transfer.

**Reference:** [An overview of multi-task learning](https://academic.oup.com/nsr/article/5/1/30/4101432#112515844)

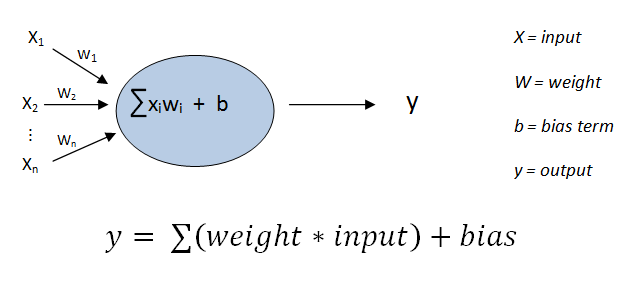
Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses)

# 5 Must-Know Activation Functions Used in Neural Networks

The universal approximation theorem implies that a neural network can approximate any continuous function that maps inputs (X) to outputs (y). The ability to represent any function is what makes the neural networks so powerful and widely-used.

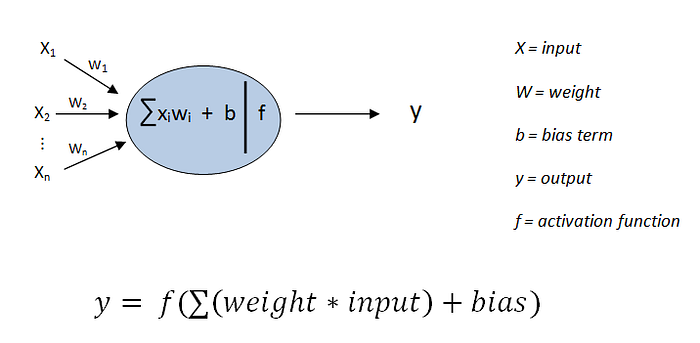
To be able to approximate any function, we need non-linearity. That’s where the activation functions come into play. They are used to add non-linearity to neural networks. Without activation functions, neural networks can be considered as a collection of linear models.

Neural networks are combinations of layers that contain many nodes. Thus, the building process starts with a node. The following represents a node without an activation function.



A neuron without an activation function (image by author)

The output y is a linear combination of inputs and a bias. We need to somehow add an element of non-linearity. Consider the following node structure.



A neuron with an activation function (image by author)

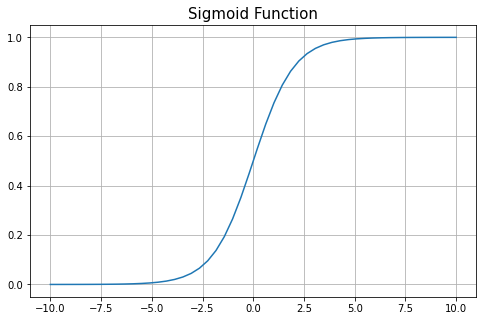
Non-linearity is achieved by applying an activation function to the sum of the linear combination of inputs and bias. The added non-linearity depends on the activation function.

In this post, we will talk about 5 commonly used activations in neural networks.

# 1. Sigmoid

The sigmoid function bounds a range of values between 0 and 1. It is also used in logistic regression models.

Whatever the input values to a sigmoid function are, the output values will be between 0 and 1. Thus, the output of each neuron is normalized into the range 0–1.



(image by author)

The output (y) is more sensitive to the changes on the input (x) for x values close to 0. As the input values move away from zero, the output value becomes less sensitive. After some point…

# Activation functions in Neural Networks

It is recommended to understand [Neural Networks](https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/) before reading this article.

In the process of building a neural network, one of the choices you get to make is what [Activation Function](https://www.geeksforgeeks.org/activation-functions-neural-networks/) to use in the hidden layer as well as at the output layer of the network. This article discusses some of the choices.

## ****Elements of a Neural Network****

**Input Layer:**This layer accepts input features. It provides information from the outside world to the network, no computation is performed at this layer, nodes here just pass on the information(features) to the hidden layer.

**Hidden Layer*:***Nodes of this layer are not exposed to the outer world, they are part of the abstraction provided by any neural network. The hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the output layer.

**Output Layer:**This layer bring up the information learned by the network to the outer world.

## ****What is an activation function and why use them?****

The activation function decides whether a neuron should be activated or not by calculating the weighted sum and further adding bias to it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

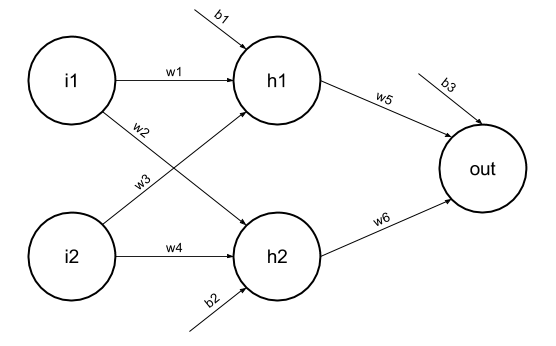
**Explanation:** We know, the neural network has neurons that work in correspondence with *weight, bias,* and their respective activation function. In a neural network, we would update the weights and biases of the neurons on the basis of the error at the output. This process is known as [***back-propagation***](https://www.geeksforgeeks.org/backpropagation-in-data-mining/). Activation functions make the back-propagation possible since the gradients are supplied along with the error to update the weights and biases.

## ****Why do we need Non-linear activation function?****

A neural network without an activation function is essentially just a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

### **Mathematical proof**

*Suppose we have a Neural net like this :-*



Elements of the diagram are as follows:

**Hidden layer i.e. layer 1:**

*z(1) = W(1)X + b(1) a(1)*

*Here,*

* *z(1) is the vectorized output of layer 1*
* *W(1) be the vectorized weights assigned to neurons of hidden layer i.e. w1, w2, w3 and w4*
* *X be the vectorized input features i.e. i1 and i2*
* *b is the vectorized bias assigned to neurons in hidden layer i.e. b1 and b2*
* *a(1) is the vectorized form of any linear function.*

*(****Note:****We are not considering activation function here)*

**Layer 2 i.e. output layer :-**

***Note :****Input for layer 2 is output from layer 1*

*z(2) = W(2)a(1) + b(2)*

*a(2) = z(2)*

#### **Calculation at Output layer**

*z(2) = (W(2) \* [W(1)X + b(1)]) + b(2)*

*z(2) = [W(2) \* W(1)] \* X + [W(2)\*b(1) + b(2)]*

*Let,*

*[W(2) \* W(1)] = W*

*[W(2)\*b(1) + b(2)] = b*

*Final output : z(2) = W\*X + b*

*which is again a linear function*

This observation results again in a linear function even after applying a hidden layer, hence we can conclude that, doesn’t matter how many hidden layer we attach in neural net, all layers will behave same way because ***the composition of two linear function is a linear function itself***. Neuron can not learn with just a linear function attached to it. A non-linear activation function will let it learn as per the difference w.r.t error. **Hence we need an activation function.**

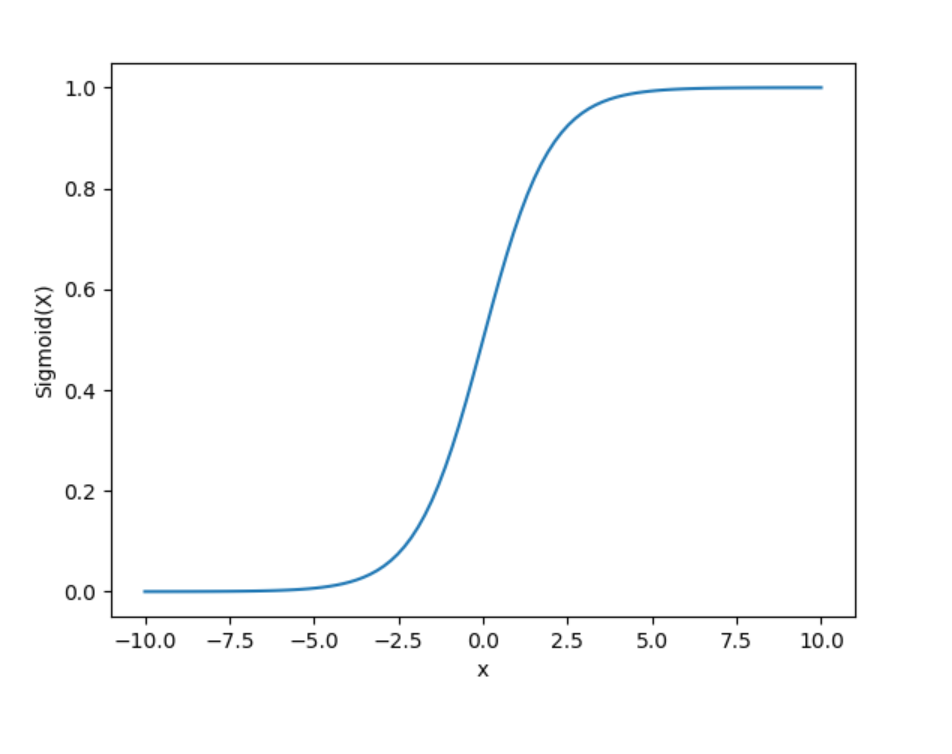
## ****Variants of Activation Function****

### **Linear Function**

* **Equation :**Linear function has the equation similar to as of a straight line i.e. **y = x**
* No matter how many layers we have, if all are linear in nature, the final activation function of last layer is nothing but just a linear function of the input of first layer.
* **Range :** -inf to +inf
* **Uses : Linear activation function** is used at just one place i.e. output layer.
* **Issues :**If we will differentiate linear function to bring non-linearity, result will no more depend on *input “x”* and function will become constant, it won’t introduce any ground-breaking behavior to our algorithm.

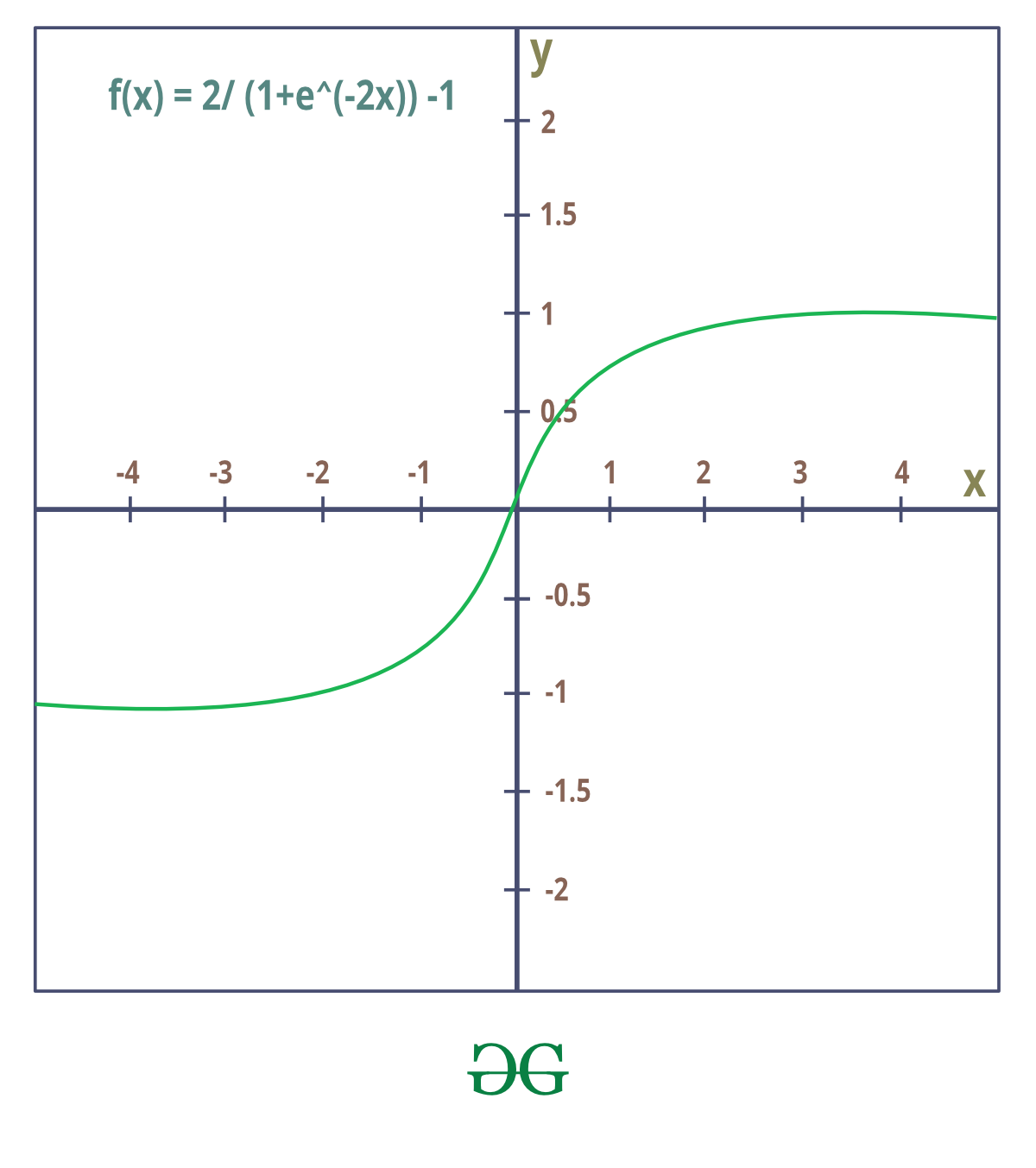
**For example :** Calculation of price of a house is a regression problem. House price may have any big/small value, so we can apply linear activation at output layer. Even in this case neural net must have any non-linear function at hidden layers.

### **Sigmoid Function**

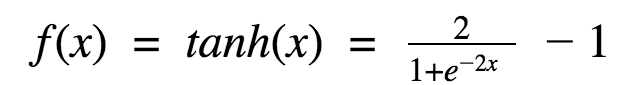


* It is a function which is plotted as **‘S’** shaped graph.
* **Equation :**A = 1/(1 + e-x)
* **Nature :** Non-linear. Notice that X values lies between -2 to 2, Y values are very steep. This means, small changes in x would also bring about large changes in the value of Y.
* **Value Range :**0 to 1
* **Uses :**Usually used in output layer of a binary classification, where result is either 0 or 1, as value for sigmoid function lies between 0 and 1 only so, result can be predicted easily to be ***1*** if value is greater than **0.5** and ***0*** otherwise.

### **Tanh Function**

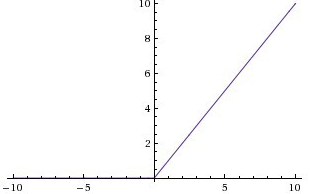


* The activation that works almost always better than sigmoid function is Tanh function also known as **Tangent Hyperbolic function**. It’s actually mathematically shifted version of the sigmoid function. Both are similar and can be derived from each other.
* **Equation :-**



* **Value Range :-**-1 to +1
* **Nature :-**non-linear
* **Uses :-**Usually used in hidden layers of a neural network as it’s values lies between **-1 to 1**hence the mean for the hidden layer comes out be 0 or very close to it, hence helps in *centering the data* by bringing mean close to 0. This makes learning for the next layer much easier.

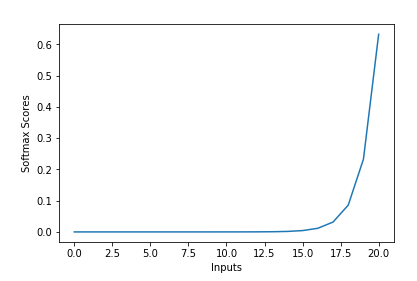
### **RELU Function**



* It Stands for *Rectified linear unit*. It is the most widely used activation function. Chiefly implemented in *hidden layers* of Neural network.
* **Equation :- *A(x) = max(0,x)***. It gives an output x if x is positive and 0 otherwise.
* **Value Range :-**[0, inf)
* **Nature :-**non-linear, which means we can easily backpropagate the errors and have multiple layers of neurons being activated by the ReLU function.
* **Uses :-**ReLu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. At a time only a few neurons are activated making the network sparse making it efficient and easy for computation.

In simple words, RELU learns *much faster* than sigmoid and Tanh function.

### **Softmax Function**



The softmax function is also a type of sigmoid function but is handy when we are trying to handle multi- class classification problems.

* **Nature :-**non-linear
* **Uses :-**Usually used when trying to handle multiple classes. the softmax function was commonly found in the output layer of image classification problems.The softmax function would squeeze the outputs for each class between 0 and 1 and would also divide by the sum of the outputs.
* **Output:-**The softmax function is ideally used in the output layer of the classifier where we are actually trying to attain the probabilities to define the class of each input.
* The basic rule of thumb is if you really don’t know what activation function to use, then simply use *RELU* as it is a general activation function in hidden layers and is used in most cases these days.
* If your output is for binary classification then, *sigmoid function* is very natural choice for output layer.
* If your output is for multi-class classification then, Softmax is very useful to predict the probabilities of each classes.

Whether you're preparing for your first job interview or aiming to upskill in this ever-evolving tech landscape, [GeeksforGeeks Courses](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses) are your key to success. We provide top-quality content at affordable prices, all geared towards accelerating your growth in a time-bound manner. Join the millions we've already empowered, and we're here to do the same for you. Don't miss out - [check it out now!](https://www.geeksforgeeks.org/courses?utm_source=geeksforgeeks&utm_medium=article_bottom_text&utm_campaign=courses)