Machine Learning

Machine learning is a branch of artificial intelligence (AI) that involves teaching computers to learn from data, identify patterns, and make decisions without being explicitly programmed to do so. In simple terms, it's like teaching a computer to learn from examples.

Arthur Samuel's Definition (1959)

"Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed."

Tom Mitchell's Definition (1997)

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Classical Programming vs Machine Learning

In classical programming, we provide logic and inputs to the machine which gives the outputs according to that logic.
In machine learning, we give only the input and output and the machine develops the logic based on the data.

Overfitting, Underfitting and Best Fit

Overfitting

Overfitting occurs when a machine learning model learns the training data too well, capturing noise or random fluctuations in the data rather than the underlying patterns.
This results in a model that performs well on the training data but poorly on unseen or new data.

Example:

Imagine you're trying to fit a curve to a set of data points. If you use a very high-degree polynomial function to fit the data, the curve might bend and twist to pass through every data point exactly, including outliers and noise. This curve might look great on the training data, but when you try to predict new data points, it might behave erratically and produce inaccurate results because it's overly sensitive to small variations in the training data.

Underfitting
Underfitting occurs when a machine learning model is too simple to capture the underlying structure of the data. This results in a model that performs poorly both on the training data and on new data. Underfitting often happens when the model is too simplistic relative to the complexity of the underlying data.

Example:
Continuing with the curve fitting analogy, if you use a linear function (a straight line) to fit data that actually follows a nonlinear pattern, the resulting line might not capture the true relationship between the variables. It might have a high error rate both on the training data and on new data because it's not flexible enough to represent the underlying data structure.

Best Fit
Best fit, also known as a good fit or optimal fit, occurs when a machine learning model generalizes well to new, unseen data. This means the model accurately captures the underlying patterns in the data without being overly complex or too simplistic. Achieving the best fit involves finding a balance between capturing enough complexity to represent the data accurately and avoiding overfitting.

Example:
Going back to the curve fitting example, if you use a polynomial function of moderate degree to fit the data, the resulting curve might not pass through every data point exactly, but it captures the overall trend of the data without being overly sensitive to noise or outliers. This curve would likely perform well not only on the training data but also on new data, making it a good fit for the dataset.

Concept of Epochs?

One Epoch is one complete pass through entire training dataset. An epoch might consist of multiple iterations.

An iteration is one complete pass through “batch” of training dataset.

Eg, say my dataset has 1000 datapoints and I chose a batch size of 50.

i.e., number of iterations = 1000/ 50 = 20.

Now in each epoch model will go through 20 iterations, each iteration consisting of 50 datapoints.

The number of epochs required for convergence depends on factors such as the complexity of the model, the size of the dataset, and the learning rate.
Increasing the number of epochs beyond a certain point may lead to overfitting, where the model performs well on the training data but poorly on new, unseen data.

Here's a simplified process for deciding these parameters

1. Start with a small number of epochs and gradually increase it until you observe diminishing returns in performance improvement or signs of overfitting.
2. Choose a batch size that balances computational efficiency and model convergence. Experiment with different batch sizes to find the optimal one for your model and hardware setup.
3. Monitor the model's performance on a validation set during training to determine the appropriate number of epochs and batch size.

Gradient Descent Algorithm

Initialization: Start with initial values for the model parameters (weights and biases).
Compute Gradient: Calculate the gradient of the loss function with respect to each parameter. The gradient indicates the direction of the steepest increase in the loss function.
Update Parameters: Adjust the parameters in the direction opposite to the gradient to minimize the loss function. This is done by subtracting a fraction of the gradient (learning rate times gradient) from the current parameter values.
Repeat: Repeat the process iteratively until convergence or until a stopping criterion is met (e.g., maximum number of iterations).
Convergence: The algorithm converges when the change in the loss function becomes small or negligible, indicating that further iterations are unlikely to significantly improve the model.

Types of Gradient Descent

Convergent Gradient Descent: In convergent gradient descent, the algorithm successfully reaches the optimal solution, where the loss function is minimized. This occurs when the learning rate is chosen appropriately, and the optimization process progresses smoothly towards convergence.

Oscillating Gradient Descent: Oscillating gradient descent occurs when the learning rate is too high, causing the optimization process to oscillate around the optimal solution without converging. This can lead to slow convergence or instability in the optimization process.

Divergent Gradient Descent: Divergent gradient descent occurs when the learning rate is too high or the optimization process is unstable, causing the loss function to increase indefinitely. In this case, the algorithm fails to converge and may even diverge away from the optimal solution.

Factors Affecting Gradient Descent

Learning Rate: The choice of learning rate plays a crucial role in determining the convergence behavior of gradient descent. A too high learning rate can cause oscillations or divergence, while a too low learning rate can result in slow convergence.

Initialization: The initial values of the model parameters can affect the convergence behavior of gradient descent. Random initialization or initialization close to the optimal solution may lead to faster convergence.

Batch Size: Gradient descent can be applied with different batch sizes, such as batch gradient descent (using the entire dataset), mini-batch gradient descent (using subsets of the dataset), or stochastic gradient descent (using individual examples). The choice of batch size can affect convergence speed and optimization stability.

Cases in Gradient Descent
Moderate, slow convergence, oscillating, diverging. All based on value of learning rate.

Supervised Learning Types

Regression
1. Simple Linear
2. Multiple Linear
3. Polynimial
4. Lasso
5. Ridge
6. Decision Tree Regressor
7. Support Vector Machine Regression

Classification
1. Logistic Regression
2. Decision Tree Classifier
3. Support Vector Machine Classifier
4. Naive Bayes Classifier
5. K-nearest neighbour

Unsupervised Learning Types

Clustering
1. K means
2. DBSCAN
3. Hierarchical

Associate Rule Mining

Dimensionality Reduction

Anomaly Detection

Bias-Variance tradeoff

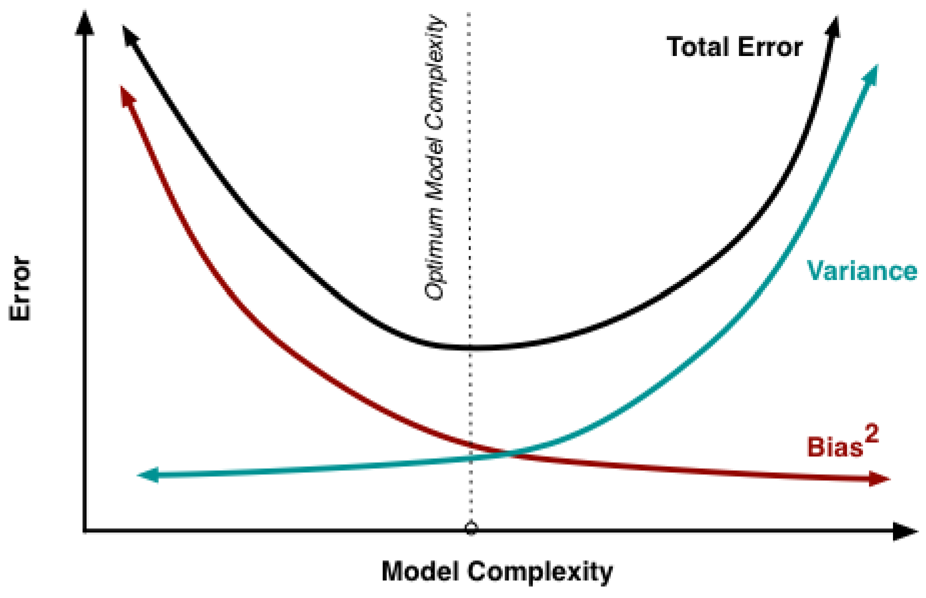
Bias: difference between the predicted and actual values of data is bias. High bias means there is large difference between predicted and actual values. High bias = high error on predicted values.

Variance: how much scattered the datapoints are with relation to each other. High variance means the datapoints are scattered.

Overfitting is low bias and high variance, means the predicted values are close to actual but those datapoints are spread from each other.

Underfitting is high bias and low variance, meant the predicted values are not correct, they differ largely from actual values but since the model is simpler thus the datapoints are close to each other.

What's desired is low bias and low variance of a model, which is called Best Fit Model.



Lasso & Ridge Regression

Also known as L1 & L2 regularizations. Lasso and Ridge are used to add penalty term to cost function so that the dependency on some features can be reduced. In Lasso and Ridge Regression our purpose is to reduce parameters (weights) as low as possible. This is done to resolve overfitting problem.

Lasso Regression equation,

$$Cost\ Function\ = \frac{1}{n}\sum_{i=1}^{n}\left(h_\theta(x)^i - y^i\right)^2 + \lambda\sum_{i=1}^{n}|slope|$$
$$\lambda = Hyperparameter$$

Ridge Regression equation,

$$SSE_{L_2} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\sum_{j=1}^{P}\beta_j{}^2$$

The difference between **Lasso** and Ridge is that Lasso regression also helps in **Feature Selection** be eliminating some or few parameters with low significance.

Cost Function

Hypothesis:	$h_\theta(x) = \theta_0 + \theta_1 x$
Parameters:	θ_0, θ_1
Cost Function:	$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
Goal:	$\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Parameters vs Hyperparameters

Parameters	Hyperparameters
Variables learned by model during training	Settings or configurations set before training
Directly affects the predictions made by model	Indirectly affects the learning process of model
These can be adjusted automatically during training	These can’t be adjusted during training, they are set before training
Eg, weights and biases	Eg, learning rate α, number of neurons, number of hidden layers, regularization strength λ of Lasso & Ridge regression, branches in decision tree
These are optimized using Gradient Descent optimization	These are optimized using, grid search, random search, bayesian optimization and hand tuning
These are crucial for the model to predict correct values	These are also crucial to prepare best learning settings for the model

Grid Search Optimization

In Grid search the best setting is found from predefined lists of settings.
This is a brute force optimization technique, it checks for each pairs of hyperparameters.
Eg, we have a = [10, 20, 30, 40] and b = [1, 2, 3, 4] as 2 hyperparameters.
Then, it will consider a grid of a and b and checks for each value of ‘a’ with ‘b’ and selects the most optimum setting.

Advantage:
Grid search finds the most optimal setting.
It is straight forward and easy to implement.

Disadvantage:
Grid search is computationally very expensive.
For large dimensional datasets, it might take very very large time.

Random Search Optimization

In random search, a grid is selected as well like grid search. But instead of searching each cell, it randomly picks cells and calculate accuracy for those settings. And finally gives the setting it found optimal.

Advantage:
Computationally very less expensive than grid search.
It can be used for datasets with higher dimensions.

Disadvantages:
It might not explore the most optimal setting, and can give less optimal setting as answer as well.
It doesn’t learn from priorly checked experiments.
And it becomes difficult to predict next set of experiment

Bayesian Optimization

Bayesian optimization is a sequential model-based optimization technique used to find the optimal set of hyperparameters for a machine learning model. It leverages probabilistic models to approximate the objective function and intelligently select the next set of hyperparameters to evaluate based on past observations.

Advantages:
Efficiently explores the hyperparameter space and converges to the optimal solution with fewer evaluations compared to grid search or random search.
Automatically adapts to the structure of the objective function and identifies promising regions of the search space, making it suitable for complex and high-dimensional optimization problems.

Disadvantages:
Requires careful tuning of its own hyperparameters, such as the choice of surrogate model and acquisition function.
Can be computationally expensive and time-consuming for large-scale optimization problems or when evaluating complex objective functions.

Hand Tuning Optimization

Hand tuning, also known as manual tuning or trial and error, involves manually selecting hyperparameter values based on the practitioner's intuition, domain knowledge, and experimentation. It relies on human judgment and expertise to iteratively adjust hyperparameters until satisfactory performance is achieved.

Advantages:
Allows for fine-grained control and customization of hyperparameter values based on domain-specific knowledge and insights.
Can be straightforward to implement and interpret, especially for practitioners with expertise in the domain or familiarity with the model architecture.

Disadvantages:
Relies heavily on the practitioner's intuition and may lead to suboptimal or biased hyperparameter choices.
Can be time-consuming and resource-intensive, as it often involves manual experimentation and evaluation of multiple hyperparameter configurations.

Capacity of ML model

Capacity refers to the ability of a machine learning model to capture a wide range of patterns and relationships present in the data. Models with higher capacity have more flexibility and expressiveness, allowing them to learn complex mappings between input and output variables.

Characteristics:
High-capacity models have a greater number of parameters and are capable of fitting more intricate and detailed patterns in the data.
Low-capacity models have fewer parameters and are limited in their ability to capture complex relationships, often resulting in simpler, more generalized models.

Examples:
Neural networks with many layers and neurons **have high capacity** and can learn intricate features in images or text data.
Linear models like linear regression or logistic regression **have lower capacity** and can only capture linear relationships between variables.

Capacity can also be considered as a measure if the model might overfit or underfit.

Ways to reduce High Variance?

1. Using a more simpler model.
2. Using regularization techniques: Lasso and Ridge
3. Cross-Validation techniques: k-fold method
4. Feature Selection methods and filtering out non relevant features
5. Early Stopping when the performance of model starts to degrade
6. Ensemble methods

Coefficient of Determination/ R^2

The coefficient of determination, often denoted as R^2, is a statistical measure that indicates the proportion of the variance in the dependent variable that is predictable from the independent variables in a regression model. In simple terms, R^2 tells us how well the independent variables explain the variability of the dependent variable.

$$R^2 = 1 - \frac{SS_{RES}}{SS_{TOT}} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$$

SSR: sum squared regression
SST: sum squared total

Classification

Classification is another supervised learning technique used to separate datapoints into separate classes. Eg, classifying flowers from IRIS dataset into 3 categories.

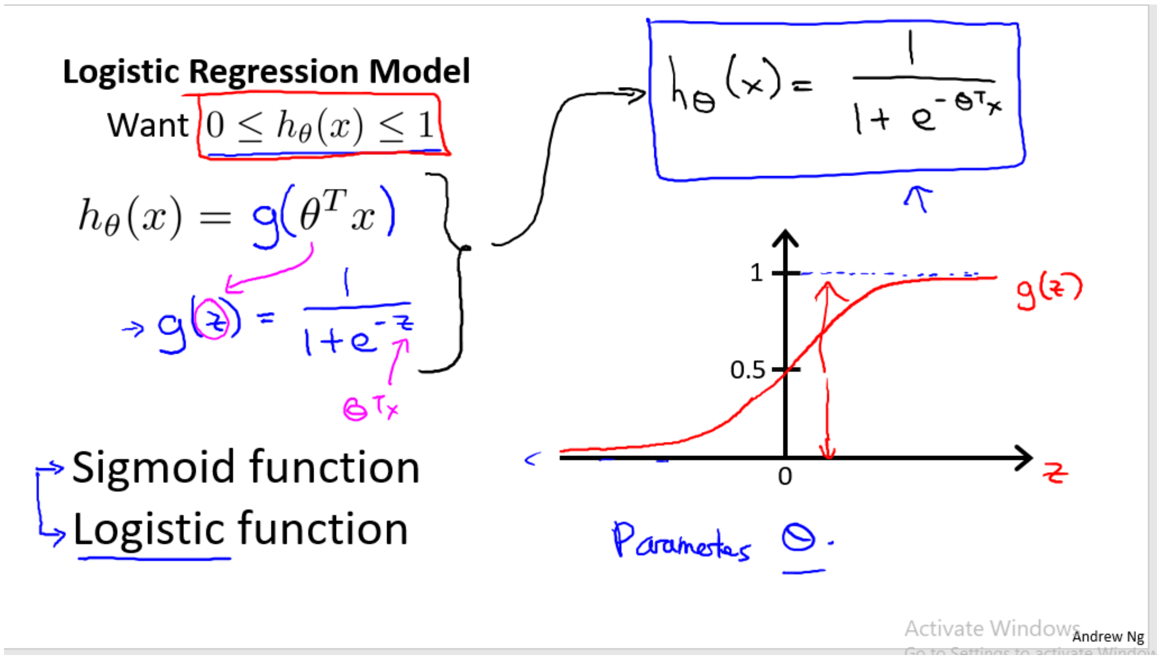
Classification is done using Logistic Regression and is further divided into 3 categories:

1. Binary LR	2. Multinomial LR	3. Ordinal LR
(classifying in 2 classes) Eg, spam/ non spam, Rumour classification, Comment analysis, churn/ no churn, Student pass/ fail	(classifying in more than 2 classes) Eg, IRIS classification, Sentiment analysis,	(classifying ordinal datapoints) Eg, movie rating, Grades classification

Why can't use Linear Regression instead of Logistic?

Linear regression predicts values in numerical-continuous range whereas we want classes in classification problems. Even if we used Linear regression, then also in case of outliers it might fail. Linear regression gives values which could be < 0 or > 1. Linear regression cost functions (MSE, MAE) may give **J** value negative, and for mis-classification they can't assign higher values to **J** as well.

For Binary Classification we use Sigmoid function to squash data in range from 0 to 1.


Logistic Regression Model

Want $\boxed{0 \le h_\theta(x) \le 1}$

$$h_\theta(x) = g(\theta^T x)$$

$$g(z) = \frac{1}{1+e^{-z}}, \quad z = \theta^T x$$

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

Sigmoid function

Logistic function

Cost Functions Used in Logistic Regression

Binary Cross-Entropy Loss (Log Loss): Used in binary logistic regression.
Categorical Cross-Entropy Loss: Used in multinomial logistic regression.

Desired qualities of cost function in Classification?

1. $J_W = 0$ if $\hat{y} = y$
2. Value of J should be very high in case of mis-classification
3. For consistency, J should always be >= 0, never yield negative values

Softmax Function

$$s\left(x_i\right) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

The softmax function is commonly used in the output layer of neural networks for multiclass classification tasks. It converts the raw output scores of the network into probabilities, allowing for intuitive interpretation and comparison of class likelihoods. The predicted class is usually the one with the highest probability after applying the softmax function.

As the scores, zi's increases, the corresponding probabilities ŷi's also increase. However, the softmax function emphasizes the differences between the scores, leading to a more pronounced probability distribution.

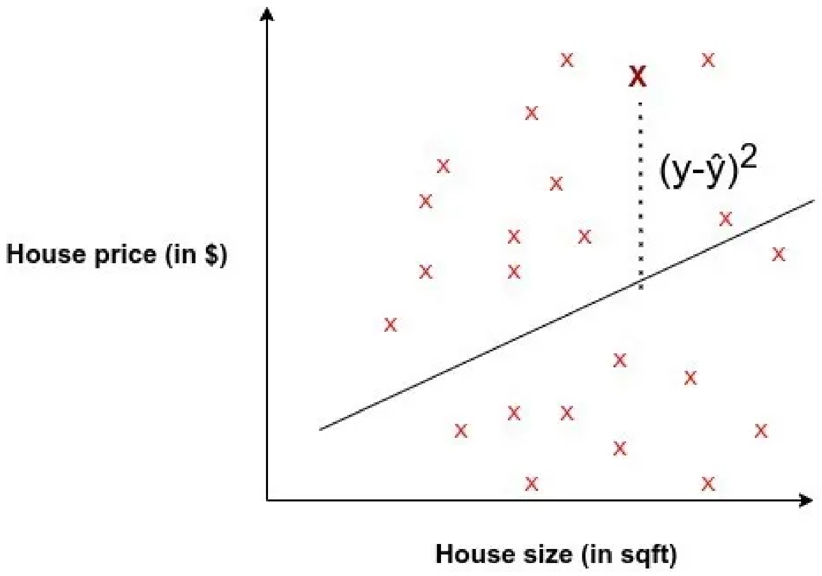
<u>**Performance Metrics: Regression**</u>

1. Mean Squared Error (MSE)
Mean squared error is perhaps the most popular metric used for regression problems. It essentially finds the average of the squared difference between the target value and the value predicted by the regression model.

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \breve{y}_j)^2$$

Where:
y_j: ground-truth value
y_hat: predicted value from the regression model
N: number of datums
Performance metrics - MSE

Few key points related to MSE:
1. It's differentiable, so it can be optimized better.
2. It penalizes even small errors by squaring them, which essentially leads to an overestimation of how bad the model is.
3. Error interpretation has to be done with squaring factor(scale) in mind. For example in our Boston Housing regression problem, we got MSE=21.89 which primarily corresponds to (Prices)^2.
4. Due to the squaring factor, it's fundamentally more prone to outliers than other metrics.



2. Mean Absolute Error (MAE)

Mean Absolute Error is the average of the difference between the ground truth and the predicted values. Mathematically, its represented as :

$$MAE = \frac{1}{N}\sum_{j=1}^{N}|y_j - \check{y}_j|$$

Where:

y_j: ground-truth value
y_hat: predicted value from the regression model
N: number of datums

Few key points for MAE:

1. It's more robust towards outliers than MAE, since it doesn't exaggerate errors.
2. It gives us a measure of how far the predictions were from the actual output. However, since MAE uses absolute value of the residual, it doesn't give us an idea of the direction of the error, i.e. whether we're under-predicting or over-predicting the data.
3. Error interpretation needs no second thoughts, as it perfectly aligns with the original degree of the variable.
4. MAE is non-differentiable as opposed to MSE, which is differentiable.

3. Root Mean Squared Error (RMSE)

Root Mean Squared Error corresponds to the square root of the average of the squared difference between the target value and the value predicted by the regression model. Basically, sqrt(MSE). Mathematically it can be represented as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(y_j - \check{y}_j)^2}$$

It addresses a few downsides in MSE.

Few key points related to RMSE:

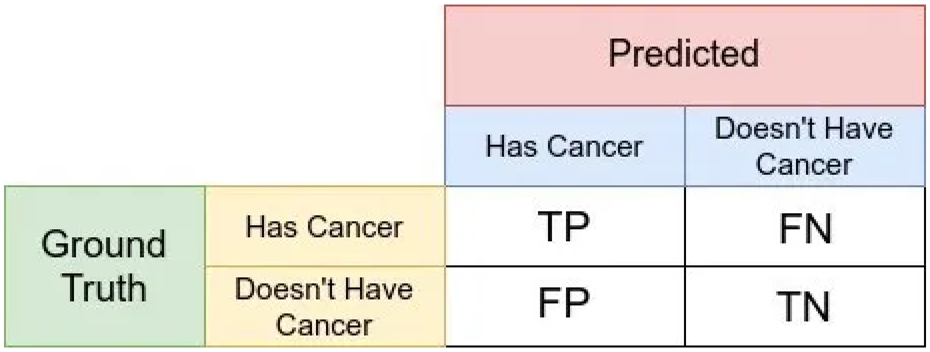
1. It retains the differentiable property of MSE.
2. It handles the penalization of smaller errors done by MSE by square rooting it.

3. Error interpretation can be done smoothly, since the scale is now the same as the random variable.
4. Since scale factors are essentially normalized, it’s less prone to struggle in the case of outliers.

4. Coefficient of Determination: discussed above.

<u>**Performance Metrics: Classification**</u>

Confusion Matrix



		Predicted	
		Has Cancer	Doesn't Have Cancer
Ground Truth	Has Cancer	TP	FN
	Doesn't Have Cancer	FP	TN

Each cell in the confusion matrix represents an evaluation factor. Let’s understand these factors one by one:

1. **True Positive(TP)** signifies how many positive class samples your model predicted correctly.
2. **True Negative(TN)** signifies how many negative class samples your model predicted correctly.
3. **False Positive(FP)** signifies how many negative class samples your model predicted incorrectly. This factor represents Type-I error in statistical nomenclature. This error positioning in the confusion matrix depends on the choice of the null hypothesis.
4. **False Negative(FN)** signifies how many positive class samples your model predicted incorrectly. This factor represents Type-II error in statistical nomenclature. This error positioning in the confusion matrix also depends on the choice of the null hypothesis.

Difference between Supervised and Unsupervised Learning

Last Updated : 11 Jan, 2024

Navigating the realm of machine learning, many grapple with understanding the key disparities between supervised and unsupervised learning. This article aims to elucidate these differences, addressing questions on input data, computational complexities, real-time analysis, and the reliability of results.

Supervised learning

When an algorithm is trained on a labelled dataset—that is, when the input data used for training is paired with corresponding output labels—it is referred to as supervised learning. Supervised learning aims to find a mapping or relationship between the input variables and the desired output, which enables the algorithm to produce precise predictions or classifications when faced with fresh, unobserved data.

An input-output pair training set is given to the algorithm during a supervised learning process. For every example in the training set, the algorithm iteratively modifies its parameters to minimize the discrepancy between its predicted output and the actual output (the ground truth). This procedure keeps going until the algorithm performs at an acceptable level.

Supervised learning can be divided into two main types:

1. **Regression:** In regression problems, the goal is to predict a continuous output or value. For example, predicting the price of a house based on its features, such as the number of bedrooms, square footage, and location.
2. **Classification:** In classification problems, the goal is to assign input data to one of several predefined categories or classes. Examples include spam email detection, image classification (e.g., identifying whether an image contains a cat or a dog), and sentiment analysis.

Why supervised learning?

The basic aim is to approximate the mapping function(mentioned above) so well that when there is a new input data (x) then the corresponding output variable can be predicted. It is called supervised learning because the process of learning(from the training dataset) can be thought of as a teacher who is supervising the entire learning process. Thus, the “learning algorithm” iteratively makes predictions on the training data and is corrected by the “teacher”, and the learning stops when the algorithm achieves an acceptable level of performance (or the desired accuracy).

Supervised Learning Example

Suppose there is a basket which is filled with some fresh fruits, the task is to arrange the same type of fruits in one place. Also, suppose that the fruits are apple, banana, cherry, and grape. Suppose one already knows from their *previous work* (or experience) that, the shape of every fruit present in the basket so, it is easy for them to arrange the same type of fruits in one place. Here, the previous work is called **training data** in Data Mining terminology. So, it learns things from the training data. This is because it has a response variable that says y that if some fruit has so and so features then it is grape, and similarly for every fruit. This type of information is deciphered from the data that is used to train the model. This type of learning is called **Supervised Learning**. Such problems are listed under classical *Classification Tasks*.

Unsupervised Learning

Unsupervised learning is a type of machine learning where the algorithm is given input data without explicit instructions on what to do with it. In unsupervised learning, the algorithm tries to find patterns, structures, or relationships in the data without the guidance of labelled output.

The main goal of unsupervised learning is often to explore the inherent structure within a set of data points. This can involve identifying clusters of similar data points, detecting outliers, reducing the dimensionality of the data, or discovering patterns and associations.

There are several common types of unsupervised learning techniques:

1. **Clustering:** Clustering algorithms aim to group similar data points into clusters based on some similarity metric. K-means clustering and

hierarchical clustering are examples of unsupervised clustering techniques.
2. **Dimensionality Reduction:** These techniques aim to reduce the number of features (or dimensions) in the data while preserving its essential information. Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE) are examples of dimensionality reduction methods.
3. **Association:** Association rule learning is used to discover interesting relationships or associations between variables in large datasets. The Apriori algorithm is a well-known example used for association rule learning.

Why Unsupervised Learning?

The main aim of Unsupervised learning is to model the distribution of the data to learn more about the data. It is called unsupervised learning because there is no correct answer and there is no such teacher(unlike supervised learning). Algorithms are left to their own devices to discover and present an interesting structure in the data.

Unsupervised Learning example

Again, Suppose there is a basket and it is filled with some fresh fruits. The task is to arrange the same type of fruits in one place. This time there is no information about those fruits beforehand, it's the first time that the fruits are being seen or discovered So how to group similar fruits without any prior knowledge about them? First, any physical characteristic of a particular fruit is selected. Suppose *colour*. Then the fruits are arranged based on the color. The groups will be something as shown below:

- **RED COLOR GROUP**: apples & cherry fruits.
- **GREEN COLOR GROUP**: bananas & grapes. So now, take another physical character say, *size*, so now the groups will be something like this.
- **RED COLOR** AND **BIG SIZE**: apple.
- **RED COLOR** AND **SMALL SIZE**: cherry fruits.
- **GREEN COLOR** AND **BIG SIZE**: bananas.
- **GREEN COLOR** AND **SMALL SIZE**: grapes.

The job is done! Here, there is no need to know or learn anything beforehand. That means, no train data and no response variable. This type of learning is known as Unsupervised Learning.

Difference between Supervised and Unsupervised Learning

The distinction between supervised and unsupervised learning depends on whether the learning algorithm uses pattern-class information. Supervised learning assumes the availability of a teacher or supervisor who classifies the training examples, whereas unsupervised learning must identify the pattern-class information as a part of the learning process.

Supervised learning algorithms utilize the information on the class membership of each training instance. This information allows supervised learning algorithms to detect pattern misclassifications as feedback to themselves. In unsupervised learning algorithms, unlabeled instances are used. They blindly or heuristically process them. Unsupervised learning algorithms often have less computational complexity and less accuracy than supervised learning algorithms.

	Supervised Learning	**Unsupervised Learning**
Input Data	Uses Known and Labeled Data as input	Uses Unknown Data as input
Computational Complexity	Less Computational Complexity	More Computational Complex
Real-Time	Uses off-line analysis	Uses Real-Time Analysis of Data
Number of Classes	The number of Classes is known	The number of Classes is not known
Accuracy of Results	Accurate and Reliable Results	Moderate Accurate and Reliable Results

	Supervised Learning	**Unsupervised Learning**
Output data	The desired output is given.	The desired, output is not given.
Model	In supervised learning it is not possible to learn larger and more complex models than in, supervised learning	In unsupervised learning it is possible to learn larger and more complex models than with unsupervised learning
Training data	In supervised learning training data is used to infer model	In unsupervised learning training data is not used.
Another name	Supervised learning is also called classification.	Unsupervised learning is also called clustering.
Test of model	We can test our model.	We can not test our model.
Example	Optical Character Recognition	Find a face in an image.

Conclusion

In conclusion, the article unravels the intricate tapestry of supervised and unsupervised learning, shedding light on their roles in data analysis. Whether classifying known data or exploring uncharted territories, these methodologies play crucial roles in shaping the landscape of artificial intelligence.

Get 90% Course fee refund in just 90 Days! Also get 1:1 Mock Interview, Job assistance and more additional benefits on selected courses. Take up

Cross Validation in Machine Learning

Last Updated : 21 Dec, 2023

In **machine learning**, we couldn't fit the model on the training data and can't say that the model will work accurately for the real data. For this, we must assure that our model got the correct patterns from the data, and it is not getting up too much noise. For this purpose, we use the **cross-validation technique**. In this article, we'll delve into the process of cross-validation in machine learning.

What is Cross-Validation?

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance. Cross validation is an important step in the machine learning process and helps to ensure that the model selected for deployment is robust and generalizes well to new data.

What is cross-validation used for?

The main purpose of cross validation is to prevent overfitting, which occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model's generalization performance, i.e., its ability to perform well on new, unseen data.

Types of Cross-Validation

There are several types of cross validation techniques, including **k-fold cross validation, leave-one-out cross validation, and Holdout validation, Stratified Cross-Validation.** The choice of technique depends on the size and nature of the data, as well as the specific requirements of the modeling problem.

1. Holdout Validation

In Holdout Validation, we perform training on the 50% of the given dataset and rest 50% is used for the testing purpose. It's a simple and quick way to evaluate a model. The major drawback of this method is that we perform training on the 50% of the dataset, it may possible that the remaining 50% of the data contains some important information which we are leaving while training our model i.e. higher bias.

2. LOOCV (Leave One Out Cross Validation)

In this method, we perform training on the whole dataset but leaves only one data-point of the available dataset and then iterates for each data-point. In LOOCV, the model is trained on $n-1$ samples and tested on the one omitted sample, repeating this process for each data point in the dataset. It has some advantages as well as disadvantages also.

An advantage of using this method is that we make use of all data points and hence it is low bias.

The major **drawback** of this method is that it leads to **higher variation** in the testing model as we are testing against one data point. If the data point is an outlier it can lead to higher variation. Another drawback is it **takes a lot of execution time** as it iterates over 'the number of data points' times.

3. Stratified Cross-Validation

It is a technique used in machine learning to ensure that each fold of the cross-validation process maintains the same class distribution as the entire dataset. This is particularly important when dealing with imbalanced datasets, where certain classes may be underrepresented. In this method,

1. The dataset is divided into k folds while maintaining the proportion of classes in each fold.
2. During each iteration, one-fold is used for testing, and the remaining folds are used for training.
3. The process is repeated k times, with each fold serving as the test set exactly once.

Stratified Cross-Validation is essential when dealing with classification problems where maintaining the balance of class distribution is crucial for the model to generalize well to unseen data.

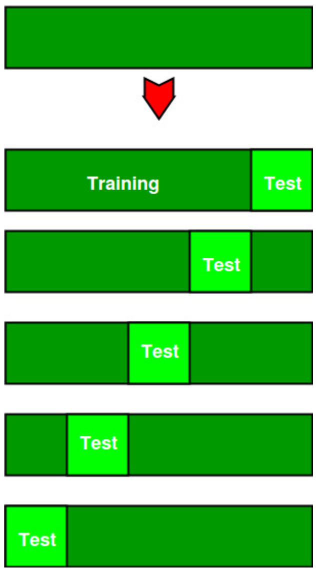
4. K-Fold Cross Validation

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

> ***Note:*** *It is always suggested that the value of k should be 10 as the lower value of k is takes towards validation and higher value of k leads to LOOCV method.*

Example of K Fold Cross Validation

The diagram below shows an example of the training subsets and evaluation subsets generated in k-fold cross-validation. Here, we have total 25 instances. In first iteration we use the first 20 percent of data for evaluation, and the remaining 80 percent for training ([1-5] testing and [5-25] training) while in the second iteration we use the second subset of 20 percent for evaluation, and the remaining three subsets of the data for training ([5-10] testing and [1-5 and 10-25] training), and so on.



```
Total instances: 25
Value of k     : 5
```

```
No. Iteration               Training set observations
Testing set observations
 1      [ 5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24]   [0 1 2 3 4]
 2      [ 0  1  2  3  4 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24]   [5 6 7 8 9]
 3      [ 0  1  2  3  4  5  6  7  8  9 15 16 17 18 19 20 21 22
23 24]   [10 11 12 13 14]
 4      [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 20 21 22
23 24]   [15 16 17 18 19]
 5      [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19]   [20 21 22 23 24]
```

Comparison between cross-validation and hold out method

Advantages of train/test split:

1. This runs K times faster than Leave One Out cross-validation because K-fold cross-validation repeats the train/test split K-times.
2. Simpler to examine the detailed results of the testing process.

Advantages of cross-validation:

1. More accurate estimate of out-of-sample accuracy.
2. More “efficient” use of data as every observation is used for both training and testing.

Advantages and Disadvantages of Cross Validation

Advantages:

1. Overcoming Overfitting: Cross validation helps to prevent overfitting by providing a more robust estimate of the model’s performance on unseen data.
2. Model Selection: Cross validation can be used to compare different models and select the one that performs the best on average.
3. Hyperparameter tuning: Cross validation can be used to optimize the hyperparameters of a model, such as the regularization parameter, by

selecting the values that result in the best performance on the validation set.
4. Data Efficient: Cross validation allows the use of all the available data for both training and validation, making it a more data-efficient method compared to traditional validation techniques.

Disadvantages:

1. Computationally Expensive: Cross validation can be computationally expensive, especially when the number of folds is large or when the model is complex and requires a long time to train.
2. Time-Consuming: Cross validation can be time-consuming, especially when there are many hyperparameters to tune or when multiple models need to be compared.
3. Bias-Variance Tradeoff: The choice of the number of folds in cross validation can impact the bias-variance tradeoff, i.e., too few folds may result in high variance, while too many folds may result in high bias.

Python implementation for k fold cross-validation

Step 1: Import necessary libraries.

Python3

```python
from sklearn.model_selection import cross_val_score, KFold
from sklearn.svm import SVC
from sklearn.datasets import load_iris
```

Step 2: Load the dataset

let’s use the iris dataset, a multi-class classification in-built dataset.

Python3

```python
iris = load_iris()
X, y = iris.data, iris.target
```

Step 3: Create SVM classifier

`SVC` is a Support Vector Classification model from scikit-learn.

Python3

```
svm_classifier = SVC(kernel='linear')
```

Step 4:Define the number of folds for cross-validation

Python3

```
num_folds = 5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
```

Step 5: Perform k-fold cross-validation

Python3

```
cross_val_results = cross_val_score(svm_classifier, X, y, cv=kf)
```

Step 6: Evaluation metrics

Python3

```
print(f'Cross-Validation Results (Accuracy): {cross_val_results}')
print(f'Mean Accuracy: {cross_val_results.mean()}')
```

Output:

```
Cross-Validation Results (Accuracy): [1.         1.
0.96666667 0.93333333 0.96666667]
Mean Accuracy: 0.9733333333333334
```

Frequently Asked Questions(FAQs)

1.What is K in K fold cross validation?

It represents the number of folds or subsets into which the dataset is divided for cross-validation. Common values are 5 or 10.

2.How many folds for cross-validation?

The number of folds is a parameter in K-fold cross-validation, typically set to 5 or 10. It determines how many subsets the dataset is divided into.

3.What is cross-validation example?

Split the dataset into five folds. For each fold, train the model on four folds and evaluate it on the remaining fold. The average performance across all five folds is the estimated out-of-sample accuracy.

4.What is the purpose of validation?

Validation assesses a model's performance on unseen data, helping detect overfitting. It ensures the model generalizes well and is not just memorizing the training data.

5. Why use 10-fold cross-validation?

10-fold cross-validation provides a balance between robust evaluation and computational efficiency. It offers a good trade-off by dividing the data into 10 subsets for comprehensive assessment.

32

Suggest improvement

Previous

Expert Systems

Next

ML | Underfitting and Overfitting

Share your thoughts in the comments

Add Your Comment

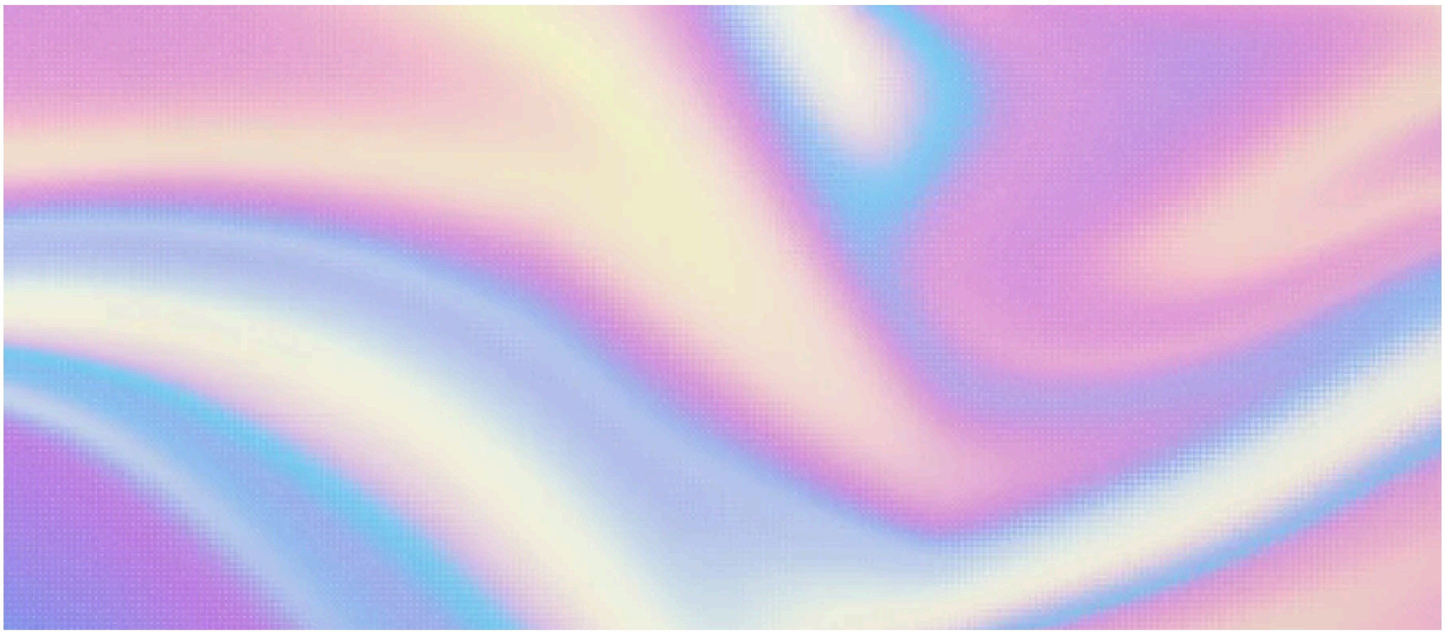
Similar Reads

Understanding Cross Decomposition in Machine Learning

ML | Kaggle Breast Cancer Wisconsin Diagnosis using KNN and Cross Validation

G Gianluca Turcatel Dec 23, 2021 1 min read

Derivation of the Binary Cross Entropy Loss Gradient



The binary cross entropy loss function is the preferred loss function in binary classification tasks, and is utilized to estimate the value of the model's parameters through gradient descent. In order to apply gradient descent we must calculate the derivative (gradient) of the loss function w.r.t. the model's parameters. Deriving the gradient is usually the most tedious part of training a machine learning model.
In this article we will derive the derivative of the binary cross entropy loss function w.r.t. W, step by step.

The binary cross entropy loss is given by

$$L(\boldsymbol{W}) = -[\boldsymbol{y} * \log(\hat{\boldsymbol{y}}) + (1 - \mathbf{y}) * \log(1 - \hat{\boldsymbol{y}})] \quad (1)$$

y is the observed class, y_hat the prediction, W the model's parameters. Predictions are given by:

$$\hat{\boldsymbol{y}} = \sigma(z) = \frac{1}{1 + e^{-z}} \quad (2)$$

z is equal to:

$$\boldsymbol{z} = \boldsymbol{w}_0 + \boldsymbol{x}_1 * \boldsymbol{w}_1 + \boldsymbol{x}_2 * \boldsymbol{w}_2 + \cdots + \boldsymbol{x}_k * \boldsymbol{w}_k = \boldsymbol{W}^T \boldsymbol{X} \quad (3)$$

To calculate the gradient of L(W) w.r.t. W we will use the chain rule:

$$\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{\partial L(\boldsymbol{W})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} \quad (4)$$

Let's derive the first term:

$$\frac{\partial L(\boldsymbol{W})}{\partial \hat{\boldsymbol{y}}} = -\left(\frac{\boldsymbol{y}}{\hat{\boldsymbol{y}}} - \frac{1-\boldsymbol{y}}{1-\hat{\boldsymbol{y}}}\right) = \frac{\hat{\boldsymbol{y}} - \boldsymbol{y}}{\hat{\boldsymbol{y}}(1-\hat{\boldsymbol{y}})} \quad (5$$

The second term is a little more complicated:

$$\frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{z}} = \frac{\partial}{\partial \boldsymbol{z}}\left[\frac{1}{1+e^{-\boldsymbol{z}}}\right] = \frac{\partial}{\partial \boldsymbol{z}}(1+e^{-\boldsymbol{z}})^{-1} = \quad (6$$

$$= -(1+e^{-\boldsymbol{z}})^{-2}(-e^{-\boldsymbol{z}}) = \frac{e^{-\boldsymbol{z}}}{(1+e^{-\boldsymbol{z}})^2} = \quad (7$$

$$= \frac{1}{(1+e^{-\boldsymbol{z}})} \frac{e^{-\boldsymbol{z}}}{(1+e^{-\boldsymbol{z}})} = \frac{1}{1+e^{-\boldsymbol{z}}} \frac{(1+e^{-\boldsymbol{z}})-1}{(1+e^{-\boldsymbol{z}})} = (8$$

$$= \frac{1}{1+e^{-\boldsymbol{z}}}\left(\frac{1+e^{-\boldsymbol{z}}}{1+e^{-\boldsymbol{z}}} - \frac{1}{1+e^{-\boldsymbol{z}}}\right) = \quad (9$$

$$= \frac{1}{1+e^{-\boldsymbol{z}}}\left(1 - \frac{1}{1+e^{-\boldsymbol{z}}}\right) = \hat{\boldsymbol{y}}(1-\hat{\boldsymbol{y}}) \quad (10$$

Done with the second term. The derivative of the third term is straight forward:

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{W}} = \boldsymbol{X} \tag{11}$$

Now let's put everything together:

$$\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{\widehat{\boldsymbol{y}} - \boldsymbol{y}}{\widehat{\boldsymbol{y}}(1 - \widehat{\boldsymbol{y}})} \widehat{\boldsymbol{y}}(1 - \widehat{\boldsymbol{y}}) \boldsymbol{X} = (\widehat{\boldsymbol{y}} - \boldsymbol{y}) \boldsymbol{X} \tag{12}$$

And there you have it: the derivative of the binary cross entropy loss function w.r.t. the model's parameters.

Follow me on Twitter and Facebook to stay updated.

#Python #MachineLearning #CrossEntropy #BinaryClassification

Performance Metrics: Classification Problems

Confusion Matrix

Confusion matrix is a tabular structure which denote the performance measure of a classification algorithm. It provides the comparision of model's predicted values and actual values based on test data.

The matrix displays four instances of possible conditions:

1. **True positives (TP):** occur when the model accurately predicts a positive data point.
2. **True negatives (TN):** occur when the model accurately predicts a negative data point.
3. **False positives (FP):** occur when the model predicts a positive data point incorrectly.
4. **False negatives (FN):** occur when the model mispredicts a negative data point.

		Actual	
		Dog	Not Dog
Predicted	Dog	**TP**	**FP**
	Not Dog	**FN**	**TN**

Precision

Precision measures the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives and false positives). It indicates the accuracy of positive predictions.

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

Recall

Recall measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives and false negatives). It indicates the ability of the classifier to find all positive instances.

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

Specificity
Specificity measures the proportion of correctly predicted negative instances (true negatives) out of all actual negative instances (true negatives and false positives). It indicates the ability of the classifier to correctly identify negative instances.

Sensitivity
Sensitivity measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives and false negatives). It indicates the ability of the classifier to find all positive instances.

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$\text{Specificity} = \frac{\text{True Negatives}}{\text{True Negatives} + \text{False Positives}}$$

Accuracy
Accuracy measures the proportion of correctly classified instances (true positives and true negatives) out of the total number of instances. It indicates the overall correctness of the classifier's predictions.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

F1 Score
The F1 score is calculated as the harmonic mean of precision and recall. It combines both precision and recall into a single metric, providing a balance between the two measures.

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

F1 Beta score

The F1 Beta score is a variant of the F1 score that allows for adjusting the relative importance of precision and recall using a parameter beta. It provides a way to emphasize either precision or recall based on the specific requirements of the classification task.

$$F\beta = \frac{(1 + \beta^2) \times Precision \times Recall}{\beta^2 \times Precision + Recall}$$

Where:

1. If β = 1, it is equivalent to the regular F1 score.
2. If β < 1, it emphasizes precision over recall.
3. If β >1, it emphasizes recall over precision.

Mis classification rate

Misclassification rate measures the proportion of incorrectly classified instances (false positives and false negatives) out of the total number of instances. It indicates the overall accuracy of the classifier.

MCR = 1 - Accuracy

Encoding Techniques for Classification Problems

Encoding techniques are used to convert categorical data to numeric, so that it becomes easier to deal with and make proper model.

One Hot Encoding

In this encoding type additional columns are added into the dataset, each column for a class type. Then, if the object belonged to a class it's class is set as 1 or hot and other set to 0 or cold.
Eg, there are three color categories R, G and B.

Object	**Red**	**Green**	**Blue**
Apples	1	0	0
Grapes	0	1	0
Blueberries	0	0	1

Advantages

1. It allows the use of categorical variables in models that require numerical input.
2. It can improve model performance by providing more information to the model about the categorical variable.
3. It can help to avoid the problem of ordinality, which can occur when a categorical variable has a natural ordering (e.g. "small", "medium", "large").

Disadvantages

1. It can lead to increased dimensionality, as a separate column is created for each category in the variable. This can make the model more complex and slow to train.
2. It can lead to sparse data, as most observations will have a value of 0 in most of the one-hot encoded columns.
3. It can lead to overfitting, especially if there are many categories in the variable and the sample size is relatively small.
4. One-hot-encoding is a powerful technique to treat categorical data, but it can lead to increased dimensionality, sparsity, and overfitting. It is important to use it cautiously and consider other methods such as ordinal encoding or binary encoding.

Label Encoding
In label encoding each class is assigned a label of unique integer value and no extra columns are created.

Object	**Color**	**Label**
Apples	Red	0
Grapes	Green	1
Blueberries	Blue	2

Advantages

1. **Simplicity:** It is straightforward to implement and understand. It involves converting categories into numerical labels, making it easy to work with for beginners.
2. **Compatibility with Algorithms:** Many machine learning algorithms, especially traditional ones like linear regression, decision trees, and support vector machines, require numerical input features. Label encoding provides a way to represent categorical data in a format compatible with these algorithms.
3. **Efficient Memory Usage:** It typically uses less memory than other encoding methods like one-hot encoding. This can be important when working with large datasets.

Disadvantages

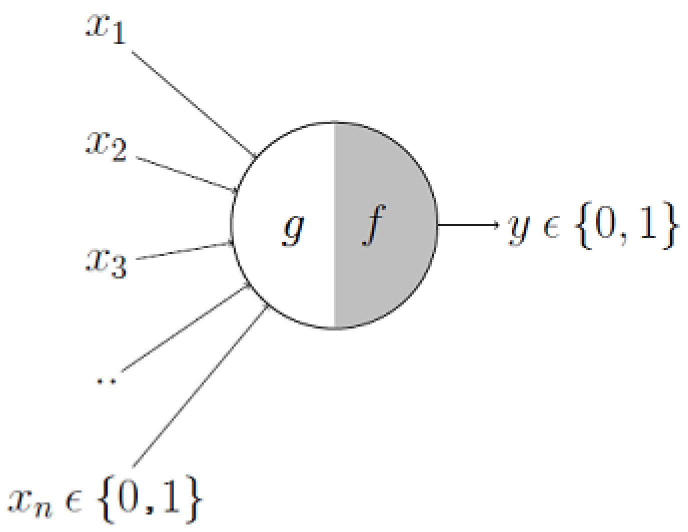
1. **Ordinal Misinterpretation:** It assumes an ordinal relationship between the categories, which may not always be true. For nominal categorical data (where there's no inherent order among categories), label encoding can mislead the algorithm into treating the encoded values as ordinal, potentially leading to incorrect model results.
2. **Arbitrary Numerical Values:** The integers assigned during label encoding are random and do not convey meaningful category information. Machine learning algorithms might misinterpret these values as having numerical significance or relationships when there aren't any.
3. **Impact on Model Performance:** It can introduce unintended relationships between categories, especially in algorithms that rely on distance metrics (e.g., k-means clustering). The algorithm may interpret smaller numerical differences as more significant than they should be.
4. **Not Suitable for High Cardinality:** When dealing with categorical features with a high number of unique categories, label encoding can lead to a significant increase in dimensionality. This can negatively impact model performance and increase the risk of overfitting.

5. **Loss of Information:** It can result in a loss of information about the original categorical data, especially if the encoded values do not represent the true nature of the categories.

Ordinal Encoding
In this encoding type, integer values are assigned maintaining the ordinal structure of data. It is very similar to label encoding. Eg, I have {low, medium, high} then these can be represented as {0, 1, 2}.

<u>McCulloch and Pitts Neuron Model</u>

Proposed by Warren McCulloch and Walter Pitts in 1943 based on real world neuron structure.
It consisted of binary inputs {x1, x2, ... , xn} values in {0, 1} with binary output {y} values in {0, 1}.



Inputs can be excitatory or inhibitory, means if the input is excitatory means they have more likelihood of firing an action and inhibitory inputs don't.
Output 'y' is 0 if any input Xi is 0, i.e., inhibitory.

$$g(x_1, x_2, x_3, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 \quad if \quad g(\mathbf{x}) \geq \theta \\ &= 0 \quad if \quad g(\mathbf{x}) < \theta \end{aligned}$$

To implement AND using mc pitts neuron, the threshold value would be 3 considering two variable input size.
Similarly to implement OR using mc pitts, the threshold value would be 1 for two variable system.

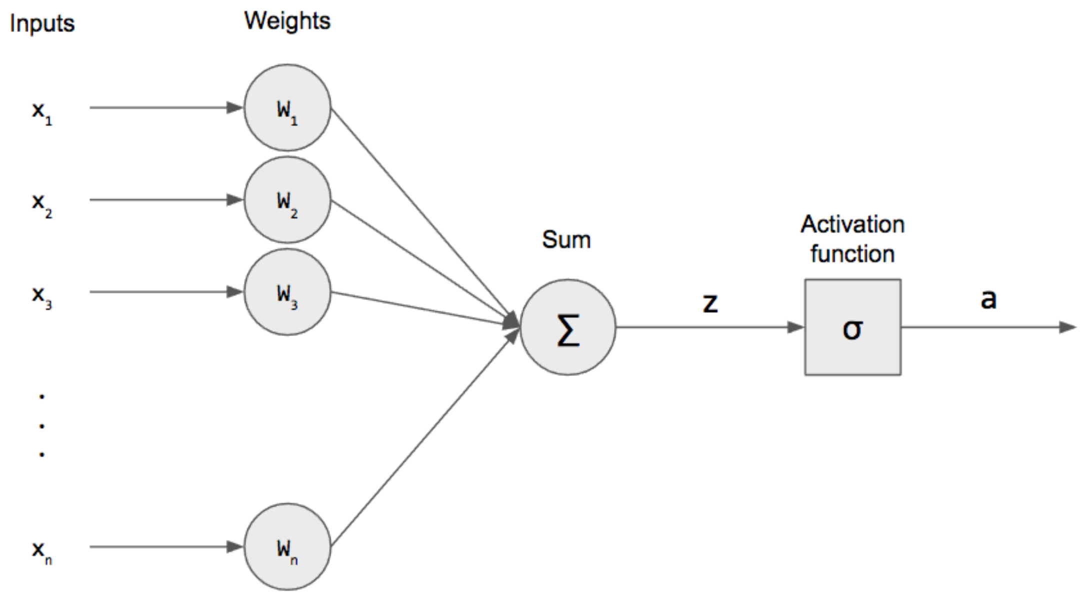
Implement OR, AND, NOR, NAND, XOR, XNOR using mc pitts here:

Linearly Separable
This means that the two classes can be separated linearly using a line or some hyperplane in higher dimension. Eg, during implementing AND using mc pitts, the inputs that produced output 1 and 0 are linearly separable as (1, 1) produced y = 1 and (0, 0), (0, 1), (1, 0) produced y = 0.

Non Linearly Separable
This means opposite of linearly separable. Eg, implementing XOR using mc pitts neuron, the input classes that produce outputs 0 and 1 can't be separated linearly.
(0, 0) and (1, 1) produced output 0 and others 1.

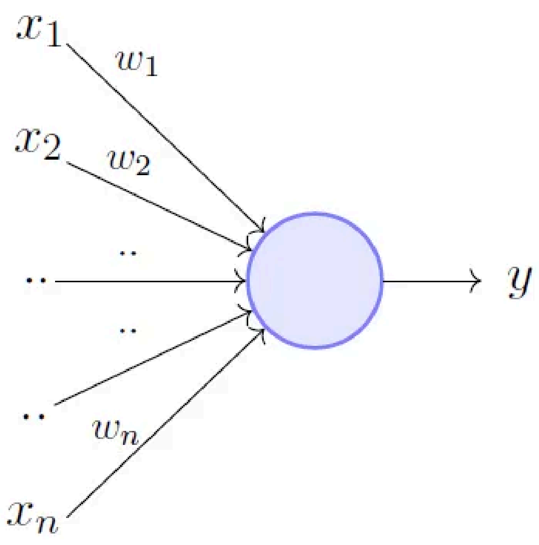
<u>Perceptron Neuron Model</u>

Frank Rosenblatt proposed the first perceptron neuron model in 1958, which was more complex that simple mc pitts neuron model. Later refined by Minsky and Papert in 1969.



Main difference than mc pitts
Introduction of weights, considering importance of some or more inputs.
Inputs not limited to boolean values.

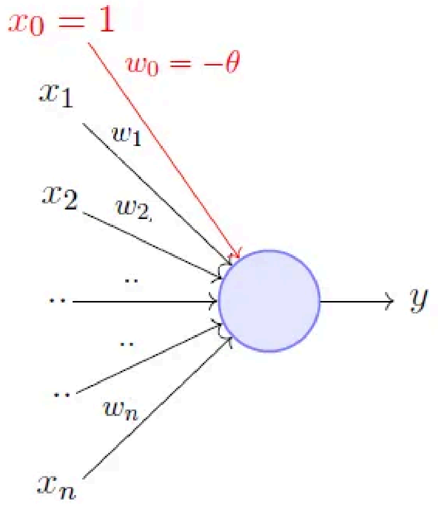
Types of Perceptron Neurons
Single layer: Single layer perceptron can learn only linearly separable patterns.
Multilayer: Multilayer perceptrons can learn about two or more layers having a greater processing power.



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$

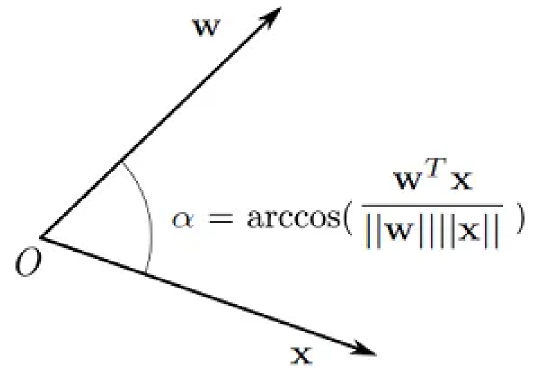
Algorithm: Perceptron Learning Algorithm

```
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ do
    Pick random $\mathbf{x} \in P \cup N$ ;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w.x} < 0$ then
        $\mathbf{w = w + x}$ ;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w.x} \geq 0$ then
        $\mathbf{w = w - x}$ ;
    end
end
//the algorithm converges when all the
 inputs are classified correctly
```

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_n]$$
$$\mathbf{x} = [1, x_1, x_2, ..., x_n]$$
$$\mathbf{w} \cdot \mathbf{x} = \mathbf{w^T x} = \sum_{i=0}^{n} w_i * x_i$$

$$\mathbf{w}^T\mathbf{x} = \|\mathbf{w}\|\|\mathbf{x}\|\, cos\alpha$$

$$cos\alpha = \frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||}$$


$\mathbf{w}$
$\alpha = \text{arccos}(\frac{\mathbf{w}^T\mathbf{x}}{||\mathbf{w}||||\mathbf{x}||})$
O
$\mathbf{x}$

Implementing OR, AND, XOR using perceptron here:

A network of $2^n + 1$ perceptron is not necessary but sufficient?
Any boolean function with ‘n’ inputs can be represented using 2^n neurons in single hidden layer and 1 extra neuron for output layer, thus completing $2^n + 1$ neurons.
However, this also means that there exists some another method which uses less than $2^n + 1$ neurons for a boolean function with ‘n’ inputs.
Thus, a network of $2^n + 1$ perceptron neurons is not necessary but sufficient.

Activation Functions

An activation function in a neural network is a mathematical function that determines the output of a neuron or a layer of neurons based on the weighted sum of inputs. The activation function introduces non-linearity to the network, allowing it to learn complex patterns and relationships in the data.

Purpose of Activation Functions

Introduce Non-linearity: Activation functions introduce non-linearity to the network, enabling it to approximate complex functions and learn non-linear patterns in the data.
Without activation functions, the network would be limited to representing linear transformations of the input data, making it unable to capture complex relationships.

Enable Learning Hierarchical Representations: Activation functions enable neural networks to learn hierarchical representations of data by transforming the input data into higher-dimensional feature spaces.
Each layer of the network applies a series of non-linear transformations to the input data, allowing the network to learn increasingly abstract and complex features.

Ensure Network Expressiveness: Activation functions play a crucial role in determining the expressiveness and representational power of the neural network.
By introducing non-linearity, activation functions enable neural networks to approximate arbitrary functions, making them powerful universal function approximators.

Stabilize Gradient Descent: Activation functions help stabilize the training process by controlling the flow of gradients during backpropagation.
Well-designed activation functions prevent issues such as vanishing or exploding gradients, which can hinder the convergence of the training algorithm.

Common Uses of Activation Functions

Classification: In classification tasks, activation functions such as softmax are used in the output layer to convert the raw output of the network into probabilities representing the likelihood of each class.

Non-linear Regression: Activation functions such as ReLU (Rectified Linear Unit) are commonly used in regression tasks to introduce non-linearity and capture complex relationships between input and output variables.

Feature Extraction: Activation functions in hidden layers help extract meaningful features from the input data by transforming it into higher-dimensional representations.

Regularization: Activation functions such as dropout or sigmoid help regularize the network by preventing overfitting and improving its generalization ability.

In summary, activation functions play a crucial role in neural network architectures by introducing non-linearity, enabling learning of complex patterns, and ensuring the network's expressiveness and stability during training. They are essential components of neural networks and are used in various tasks such as classification, regression, and feature extraction.

How a neural network of non - linear activation function behaves?
A neural network with a non-linear activation function exhibits behavior characterized by the ability to learn and represent complex patterns, extract hierarchical features from the input data, and solve a wide range of tasks that require non-linear mappings between input and output variables.

Linear Activation Function

A linear activation function computes a simple linear transformation of the input data and is suitable for regression tasks and specific architectures where linearity is desired. However, its lack of non-linearity limits its applicability to more complex tasks that require learning non-linear relationships in the data.

Advantages

1. **Simple:** Linear activation functions are straightforward and easy to understand, making them suitable for simple regression tasks.

2. **Stability:** The linear activation function provides stable gradients during training, making it less prone to issues such as vanishing or exploding gradients.

Disadvantages

1. **Limited Expressiveness:** Linear activation functions are not capable of learning complex patterns or non-linear relationships in the data, limiting the types of tasks they can be applied to.
2. **Unsuitable for Classification:** Linear activation functions are not suitable for classification tasks where non-linear decision boundaries are required, as they cannot capture non-linear relationships between input and output variables.
3. It is not possible to back propagate as the derivative of this function is constant.
4. All layers of neural network will collapse into 1.

1.

$$\textit{Binary step}$$

$$f(x) = \begin{cases} 0 & for\ x < 0 \\ 1 & for\ x \geqslant 0 \end{cases}$$

Range (f(x)) = {0,1}

2.

$$\textit{Linear}$$

$$f(x) = x$$

Range (f(x)) = (−∞,+∞)

Non Linear Activation Functions

Non-linear activation functions are mathematical functions used in neural networks to introduce non-linearity into the network's output. These functions are applied to the weighted sum of inputs and biases of neurons in the network, allowing the network to approximate complex non-linear relationships between input and output data. Non-linear activation functions are crucial for enabling neural networks to learn and represent intricate patterns and relationships in the data.

1. Sigmoid

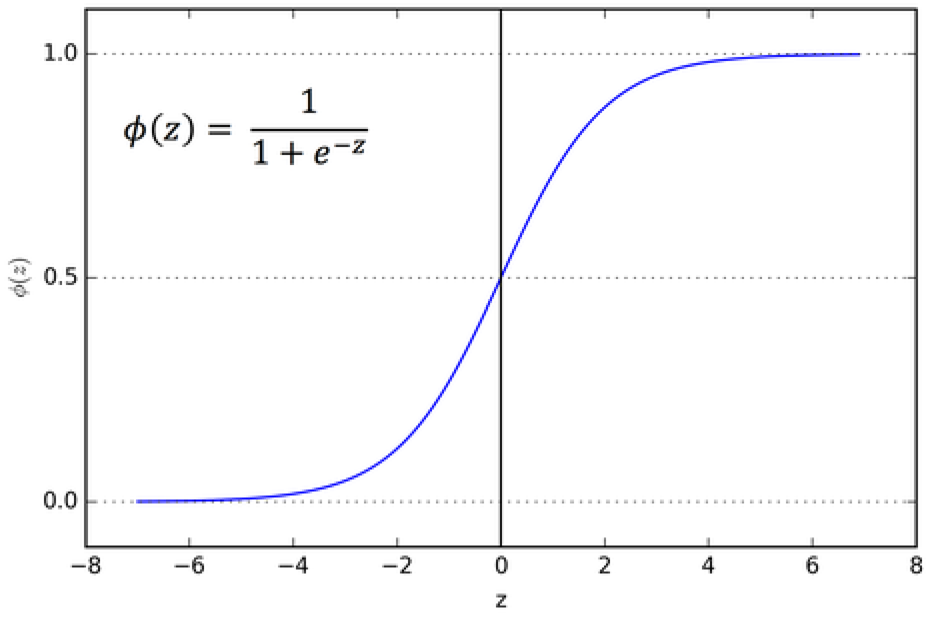
Normally used as the output of a binary probabilistic function.

Advantages

-> Gives you a smooth gradient while converging.
-> One of the best Normalised functions.
-> Gives a clear prediction(classification) with 1 & 0.

Disadvantages

-> Prone to Vanishing Gradient problem.
-> Not a zero-centric function(Always gives a positive values).
-> Computationally expensive function(exponential in nature).



2. Tanh

Normally used as the input of a binary probabilistic function.

Advantages

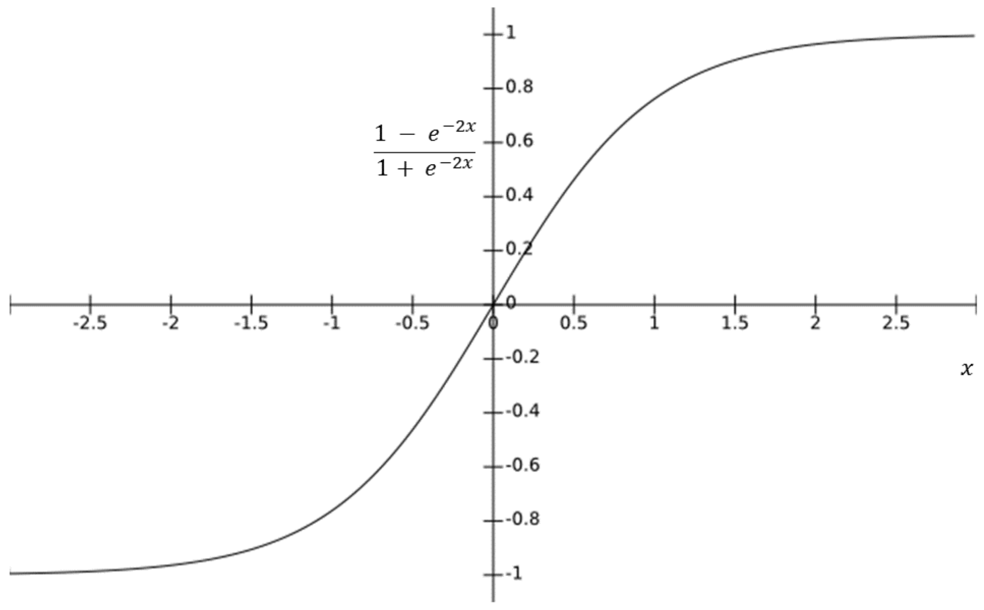
-> Zero-centric function unlike Sigmoid.

-> It is a smooth gradient converging function.

Disadvantages

-> Prone to Vanishing Gradient function.

-> Computationally expensive function(exponential in nature).



3. RELU (Rectified Linear Unit)

Advantages

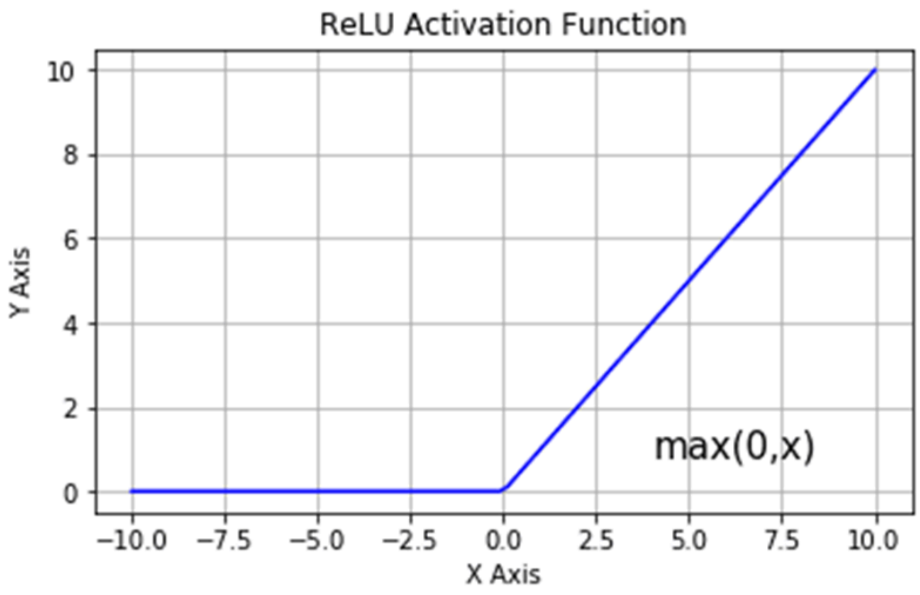
-> Can deal with Vanishing Gradient problem.

-> Computationally inexpensive function(linear in nature).

Disadvantages

-> Not a zero-centric function.

-> Gives zero value as inactive in the negative axis.



4. Leaky RELU

It is the same as of RELU function except it gives some partial value(0.01 instead zero as of RELU) in the negative axis.

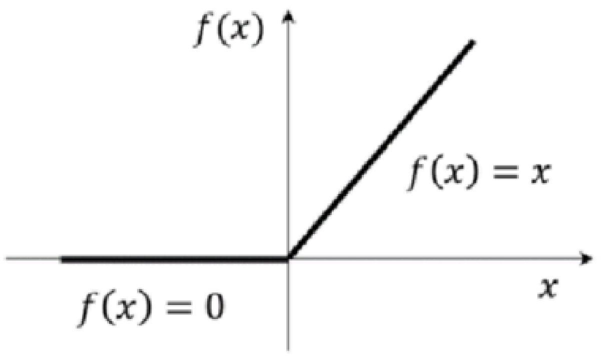
Advantages

-> Addresses the dying ReLU problem by allowing a small, non-zero gradient for negative inputs, preventing neurons from becoming inactive.

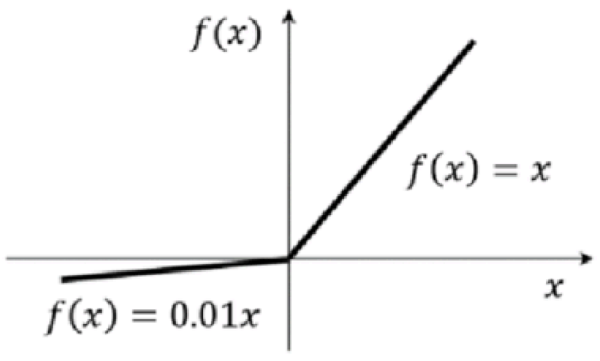
-> Provides better performance than ReLU in scenarios where dead neurons are prevalent.

Limitations

-> Introduces an additional hyperparameter (α) that needs to be tuned, which may require additional computational resources during hyperparameter optimization.



ReLU activation function



LeakyReLU activation function

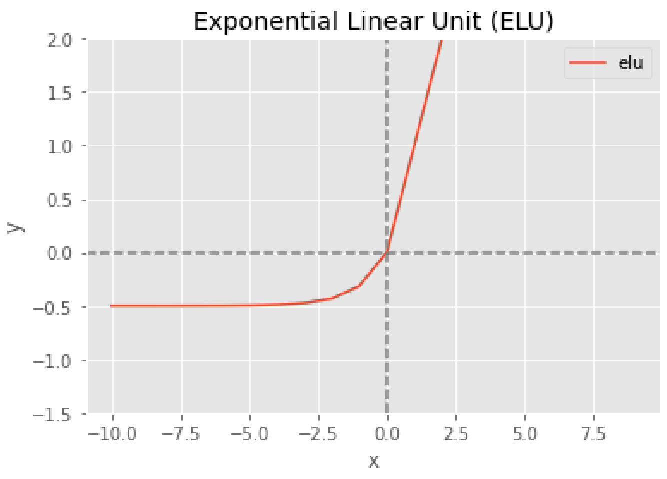
5. ELU (Exponential Linear Unit)

Advantages

-> Gives smoother convergence for any negative axis value.

-> For any positive output, it behaves like a step function and gives a constant output.

$$ELU(\alpha, x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha(e^x - 1), & \text{otherwise} \end{cases}$$

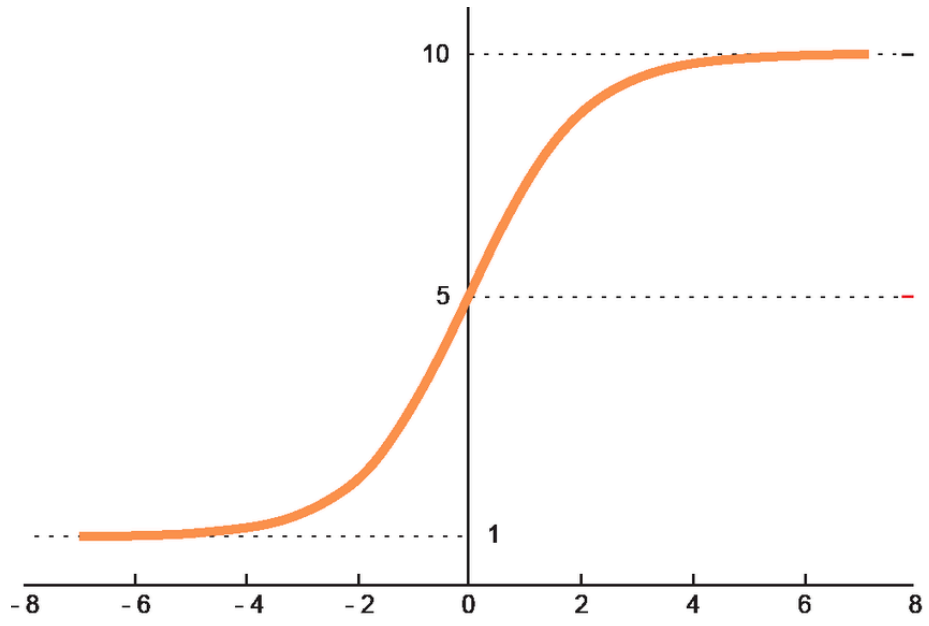


6. SoftMax
Normally used as the output in multi-class classification problems to find out different probabilities for different classes(Unlike Sigmoid which is prefered for a binary-class classification).

Advantages
-> Suitable for multi-class classification tasks as it converts raw scores into probabilities, making it easy to interpret the output as class probabilities.
-> Encourages competition between classes, making it beneficial for tasks where only one class should be activated.

Limitations
-> Sensitive to outliers and large input values, which can result in numerical instability during training.
-> Not suitable for regression or binary classification tasks where a single output is desired.

$$f(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$



7. PRELU (Parametric RELU)

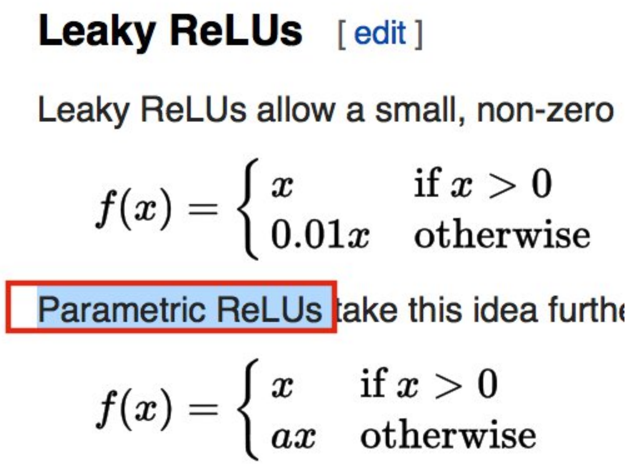
The advantage of PRELU is it has the learning parameter function which fine-tunes the activation function based on its learning rate(unlike zero in the case of RELU and 0.01 in the case of Leaky RELU).

Advantages

-> Similar to Leaky ReLU, Parametric ReLU introduces a learnable parameter (α) that allows the network to adaptively adjust the slope of the negative part of the activation function.
-> Offers more flexibility compared to Leaky ReLU, as the slope can be learned from data.

Limitations

-> Requires additional parameters to be learned, increasing the model's complexity and the risk of overfitting.



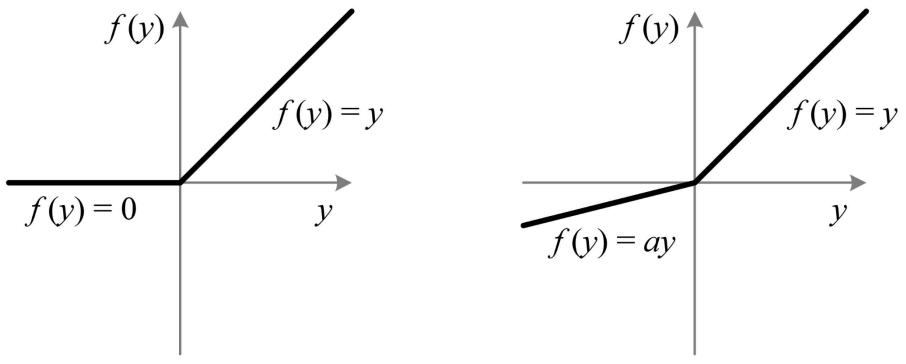
Leaky ReLUs [edit]

Leaky ReLUs allow a small, non-zero

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0.01x & \text{otherwise} \end{cases}$$

Parametric ReLUs take this idea furth

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ ax & \text{otherwise} \end{cases}$$



8. SWISH

Also known as self gated function. This activation function is one of the kinds that is being inspired by the use of the Sigmoid function inside an LSTM(Long Short Term Memory) based network.

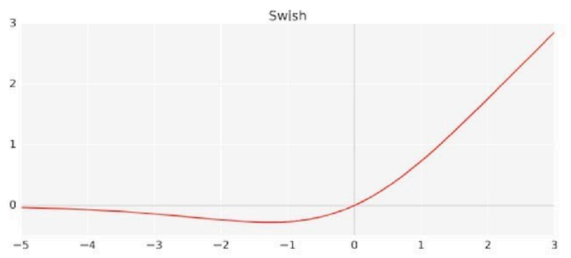
Advantages

-> Can deal with Vanishing Gradient problem.

-> The output is a workaround between RELU and Sigmoid function which helps in normalising the output.

Disadvantage

-> Computationally expensive function (as of Sigmoid).



Swish Activation Function

f(x) = x * sigmoid(x)

Self Gated Activation Function

New Activation By Google Mind

9. MaxOut

Also known as the Learnable Activation Function.

It has all the advantages of a RELU function but at the same time do not have its disadvantages.

$$h(x) = \max\left(Z_1, Z_2, \dots, Zn\right)$$

$$h(x) = \max\left(W_1 \cdot x + b_1, W_2 \cdot x + b_2, \dots, Wn \cdot x + bn\right)$$

10. SoftPlus

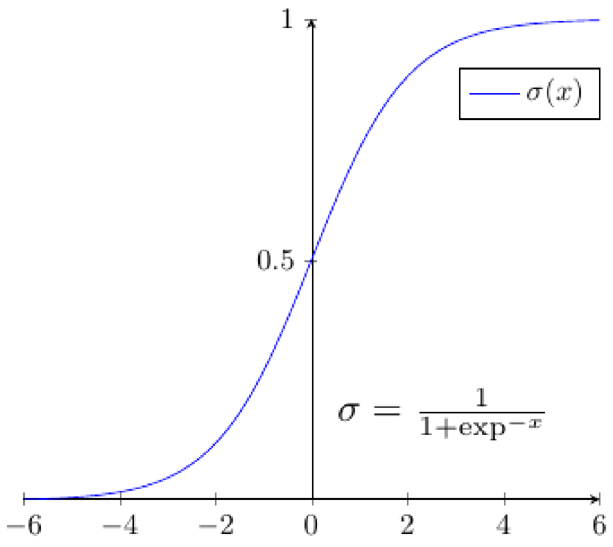
Advantages

-> Convergence of gradient is smoother than RELU function.

-> It can handle the Vanishing Gradient problem.

Disadvantage

-> Computationally expensive than RELU(as of exponential in nature).



11. Elish

Advantages

-> The Elish activation function introduces a learnable parameter (β) that controls the shape of the activation function, allowing the network to adaptively adjust the slope and curvature.

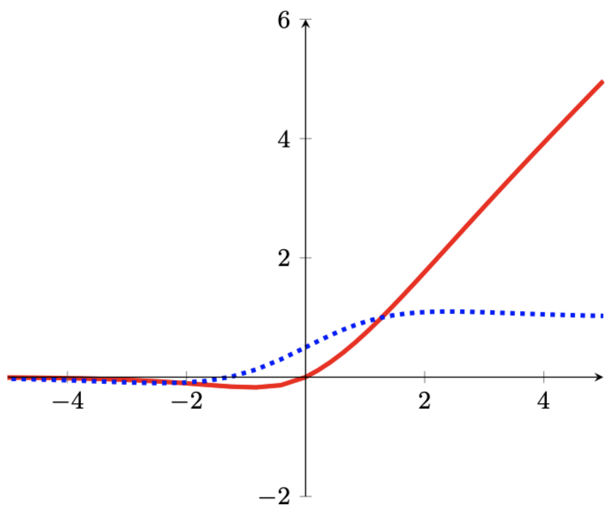
-> Offers flexibility in modeling complex relationships between input and output.

Limitations

-> Requires careful initialization and tuning of the β parameter, which may increase the complexity of model training and hyperparameter optimization.

$$f(x) = \frac{x}{1+e^{-x}} \text{ if } x \geq 0$$

$$f(x) = \frac{e^{x}-1}{1+e^{-x}} \text{ if } x < 0$$



$y(x) = \texttt{ELiSH}$

COST FUNCTIONS

The key difference between loss and cost functions lies in their scope: the loss function measures the performance of the model on individual data points, while the cost function measures the overall performance of the model across the entire dataset.

Cost Functions for Regression Models

1. **Mean Squared Error (MSE)**
   Advantages
   -> **Sensitive to large errors:** It penalizes larger errors more heavily than smaller errors.
   -> **Differentiable:** The MSE loss function is smooth and differentiable, making it suitable for gradient-based optimization algorithms.

   Disadvantages
   -> **Sensitive to outliers:** Outliers in the data can significantly impact the MSE, leading to suboptimal model performance.
   -> **Bias towards large errors:** Due to squaring the errors, MSE may overweight large errors compared to smaller errors, which may not be desired in some scenarios.

$$\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

2. **Mean Absolute Error (MAE)**
   Advantages
   -> **Robust to outliers:** MAE is less sensitive to outliers compared to MSE, making it suitable for datasets with noisy or skewed distributions.
   -> **Intuitive interpretation:** MAE directly measures the average magnitude of errors, providing an intuitive understanding of model performance.

   Disadvantages
   -> **Less sensitive to small errors:** MAE treats all errors equally regardless of their magnitude, which may not be desirable in scenarios where small errors are more critical.
   -> **Not differentiable at zero:** MAE is not differentiable at zero, which can pose challenges for gradient-based optimization algorithms.

$$\mathrm{MAE} = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

3. **Huber Loss**
   A combination of MSE and MAE, which is less sensitive to outliers.

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{2}(y_i - \hat{y}_i)^2 \qquad |y_i - \hat{y}_i| \le \delta$$

$$Huber = \frac{1}{n}\sum_{i=1}^{n}\delta\left(|y_i - \hat{y}_i| - \frac{1}{2}\delta\right) \qquad |y_i - \hat{y}_i| > \delta$$

   Advantages
   -> **Robust to outliers:** Huber loss combines the best of MSE and MAE by providing a compromise between sensitivity to outliers and robustness.
   -> **Differentiable:** Huber loss is differentiable everywhere, making it suitable for gradient-based optimization.

Disadvantages
-> **Requires tuning of hyperparameter:** Huber loss introduces a hyperparameter (delta) that controls the threshold for switching between MSE and MAE, which may require tuning.

4. **Quantile Loss**
   Used for quantile regression, where different quantiles of the conditional distribution of the target variable are estimated.

Loss Functions for Classification Models

1. **Binary Cross Entropy Loss**
   Advantages
   -> **Suitable for binary classification:** Binary cross entropy is specifically designed for binary classification tasks and measures the difference between predicted probabilities and actual binary labels.
   -> **Encourages well-calibrated probabilities:** It encourages the model to output well-calibrated probabilities for binary classification.

   Disadvantages
   -> **Not suitable for multi-class problems:** Binary cross entropy is designed for binary classification tasks and cannot be directly applied to multi-class classification problems without modification.
   -> **Sensitive to class imbalance:** Binary cross entropy may produce biased predictions in scenarios with severe class imbalance.

$$L_{BCE} = -\frac{1}{n}\sum_{i=1}^{n}(Y_i \cdot \log \hat{Y}_i + (1 - Y_i) \cdot \log(1 - \hat{Y}_i))$$

2. **Categorical Cross Entropy Loss**
   Advantages
   -> **Suitable for multi-class classification:** Categorical cross entropy extends binary cross entropy to multi-class classification tasks and measures the difference between predicted probabilities and actual categorical labels.
   -> **Encourages well-calibrated probabilities:** Similar to binary cross entropy, it encourages the model to output well-calibrated probabilities for multi-class classification.

Disadvantages
-> **Sensitive to class imbalance:** Categorical cross entropy may produce biased predictions in scenarios with severe class imbalance, similar to binary cross entropy.
-> **Computationally expensive:** Computing categorical cross entropy involves calculating the logarithm of predicted probabilities, which can be computationally expensive for large datasets or high-dimensional output spaces.

$$H(y, \hat{y}) = -\sum_{i}^{n} y_i log(\hat{y}_i)$$

3. **Hinge Loss Function**
The hinge loss function is commonly used in binary classification tasks, particularly in support vector machines (SVMs). It measures the loss incurred by a model for misclassifying examples and encourages maximizing the margin between classes.

Advantages
-> **Encourages large margin classification:** The hinge loss penalizes misclassifications more heavily as they move further away from the decision boundary, encouraging the model to maximize the margin between classes.
-> **Robust to outliers:** Hinge loss is less sensitive to outliers compared to other loss functions like squared error loss, making it suitable for datasets with noisy or skewed distributions.

Disadvantages
-> **Not differentiable at zero:** The hinge loss function is not differentiable at zero, which can pose challenges for gradient-based optimization algorithms. However, subgradient methods can be used to address this issue.
-> **Lack of probabilistic interpretation:** Unlike probabilistic loss functions such as binary cross-entropy, the hinge loss does not provide a probabilistic interpretation of model predictions. It focuses solely on margin maximization and does not directly encourage well-calibrated probabilities.

$$\ell(y) = \max(0, 1 - \overset{\substack{\text{intended output } \boldsymbol{t} = \pm 1 \\ \text{class labels: +1 or -1}}}{t} \cdot \underset{\textbf{raw}\text{ classification output}}{y})$$

Back Propagation Algorithm

Backpropagation is an algorithm used to train neural networks by adjusting the weights and biases of the neurons to minimize the error between the predicted output and the actual output of the network.

Forward Pass
Input data is passed forward through the network, and calculations are performed at each layer to generate predictions.
The input data is multiplied by weights, and biases are added to produce an output at each neuron. Activation functions are applied to these outputs to introduce non-linearity.

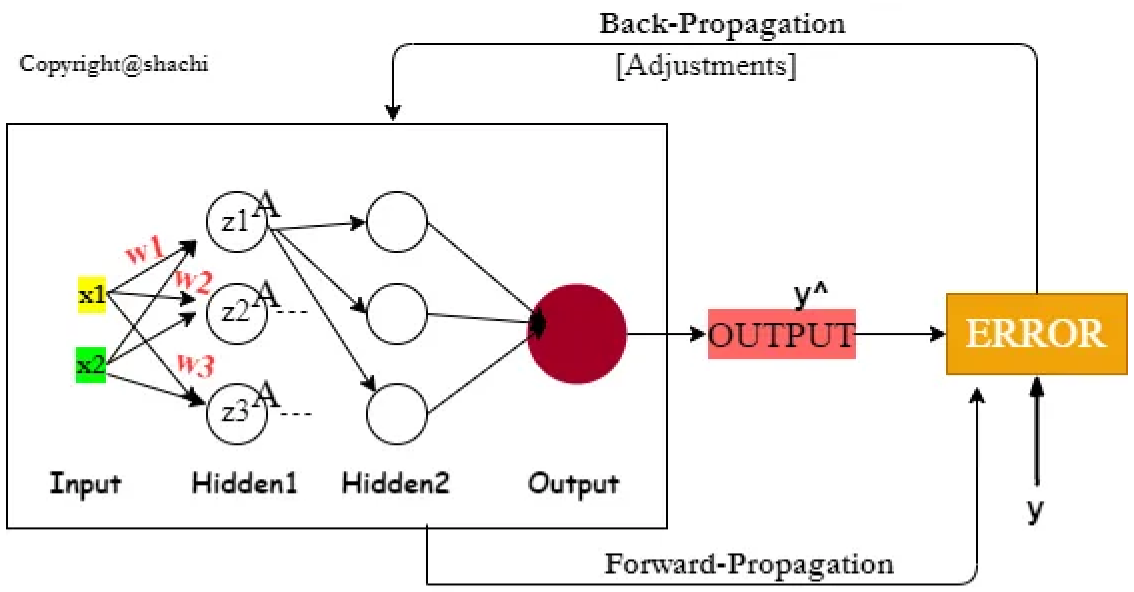
Calculate Error
The error between the predicted output and the actual output (target) is calculated using a chosen loss function.

Backward Pass
Starting from the output layer, the algorithm works backward through the network to compute the gradient of the loss function with respect to each parameter (weight and bias).
This is done using the chain rule of calculus, which allows the algorithm to propagate the error backward layer by layer.

Update Weights and Biases
Using the gradients computed during the backward pass, the algorithm updates the weights and biases of the neurons to minimize the error.
This update is performed using an optimization algorithm such as gradient descent, which adjusts the parameters in the direction that reduces the error.

Repeat
Steps 1 to 4 are repeated iteratively for multiple epochs (passes through the entire dataset) until the error converges to a minimum or until a stopping criterion is met.

In summary, backpropagation is a key algorithm for training neural networks. It involves propagating the error backward through the network, calculating gradients, and updating parameters to minimize the error. This process is repeated iteratively until the network learns to make accurate predictions.



Why can't use Gradient Descent in neural networks?
Since, the total number of trainable parameters with or without bias can be very large in case of neural networks. So, the cost or computation cost becomes very expensive in case of using Gradient Descent.
This was one thing, other is that we can't tell the amount of error or proportion of error due to one parameter in case of Gradient Descent.
This is why we use Back Propogation Algorithm.

Advantages of Backpropagation
-> Efficient parameter updates for neural networks.
-> Handles complex architectures effectively.
-> Flexible with various activation functions and network structures.
-> Scales well to large datasets and high-dimensional inputs.

Disadvantages of Backpropagation
-> Requires differentiability of activation functions and loss functions.
-> Prone to getting stuck in local minima during optimization.
-> Sensitivity to initialization of model parameters.
-> Vulnerable to issues like vanishing or exploding gradients in deep networks.

Unsupervised Learning

Clustering is the task of partitioning a dataset into groups or clusters such that data points within the same cluster are more similar to each other than to those in other clusters, with the goal of discovering inherent structures or patterns in the data.

Distance Measures

Some properties:

1. $d(a, b) \geq 0$, (non-negativity)
2. $d(a, b) = 0 \Leftrightarrow a = b$ (definiteness)
3. $d(a, b) = d(b, a)$ (symmetry)
4. $d(a, c) \leq d(a, b) + d(b, c)$, for any $b \in X$ (triangle inequality)

Centroid/ Partitioning Based Clustering
Divides the dataset into k clusters where each data point belongs to the cluster with the nearest mean.
Eg, K-means

Connectivity Based Clustering
Builds a hierarchy of clusters by iteratively merging or splitting clusters based on their proximity to each other.
Eg, Hierarchical Clustering

Density-Based Clustering
Identifies clusters as dense regions of data points separated by low-density regions.
Eg, DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

Graph-Based Clustering
Utilizes the graph representation of data, where data points are nodes and edges represent relationships or similarities between them, to partition the dataset into clusters.
Eg, Affinity Propagation

Compression-Based Clustering
Constructs a compact summary of the data using lossless compression techniques, where clusters are formed based on the similarity of the compressed representations of data points.
Eg, Spectral Clustering, BIRCH (Balanced Iterative Reducing and Clustering using Hierarchies)

Distance Measures

1. Manhattan/ Cityblock/ L1 distance

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

2. Euclidean/ L2 distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3. Cosine distance

Cosine similarity = $\cos(\theta) = \dfrac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \dfrac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}}$

Cosine distance = 1 - CS

4. Jaccard distance

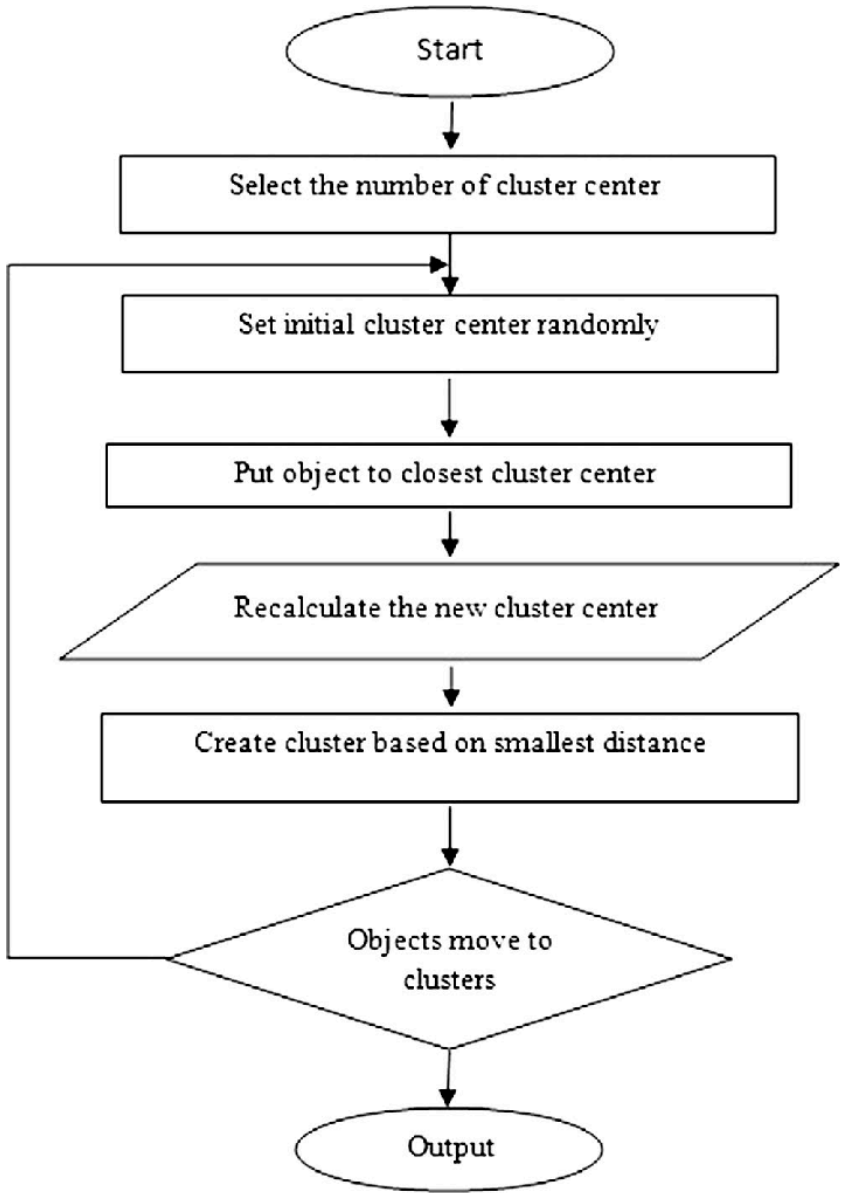
$$\begin{aligned}\text{Jaccard Distance} J_D(A, B) &= 1 - \text{Jaccard Similarity} J(A, B) \\ &= 1 - \frac{|A \cap B|}{|A \cup B|} \\ &= \frac{|A \cup B| - |A \cap B|}{|A \cup B|} \\ &= \frac{|A \triangle B|}{|A \cup B|}\end{aligned}$$

Clustering Algorithm

1. K-means Clustering

K-means clustering is an unsupervised learning algorithm that groups an unlabeled dataset into a predefined number of clusters, where each observation belongs to the cluster with the nearest mean. It works by minimizing the sum of distances between the data points and their corresponding cluster centroids.

Algorithm


Start
Select the number of cluster center
Set initial cluster center randomly
Put object to closest cluster center
Recalculate the new cluster center
Create cluster based on smallest distance
Objects move to clusters
Output

Advantages
-> K-means is simple and easy to implement.
-> It is scalable for large datasets.
-> The algorithm guarantees convergence.

Disadvantages
-> The number of clusters needs to be pre-determined.
-> It is sensitive to outliers.
-> There's a risk of getting stuck in local minima.

Effect of K-means of crescent shaped dataset?
K-means clustering, which is a distance-based method, may not perform well on datasets with complex geometric shapes like crescent shapes. This is because K-means tends to create clusters based on the mean distance of points, leading to spherical clusters. Therefore, it might not correctly identify the separate crescent-shaped clusters, and instead, it might group them as a single cluster or divide them inappropriately. For complex shapes, density-based clustering methods might be more effective

2. Heirarchial Clustering
Hierarchical clustering is an unsupervised machine learning method that groups data into a tree-like cluster hierarchy based on similarity or distance. It doesn't require pre-determination of the number of clusters, unlike K-means.

There are two main types of hierarchical clustering:
1. **Agglomerative Clustering**: Also known as the bottom-up approach, it starts by considering each data point as a single cluster and then merges the closest pair of clusters iteratively until only one cluster is left.
2. **Divisive Clustering**: This is the reverse of agglomerative clustering, also known as the top-down approach. It starts with all data points in one cluster and splits the most heterogeneous cluster at each step until only singleton clusters of individual data points remain.

Dendrogram
A dendrogram is a tree-like diagram that shows the hierarchical relationship between objects. It is most commonly created as an output from hierarchical clustering.

Linkage Criteria in hierarchical clustering
- Single Linkage: The distance between two clusters is defined as the minimum distance between any single data point in the first cluster and any single data point in the second cluster.
- Complete (Maximum) Linkage: The distance between two clusters is defined as the maximum distance between any single data point in the first cluster and any single data point in the second cluster.
- Average Linkage: The distance between two clusters is defined as the average distance between all pairs of data points in the first and second clusters.
- Ward's Method: This method merges the two clusters that provide the smallest increase in the combined error sum of squares. It's an ANOVA-based approach.

**** solve numerical**