What is Ensemble learning in Machine Learning?

1. ***An ensembled model is a machine learning model that combines the predictions from two or more models.***

2. Ensemble learning is primarily used to improve the model performance, such as classification, prediction and function approximation.

3. Basic idea is to learn a set of classifiers and make them to vote.

There are 3 most common ensemble learning methods in machine learning.

- Bagging
- Boosting
- Stacking

1. Bagging(Homogeneous)

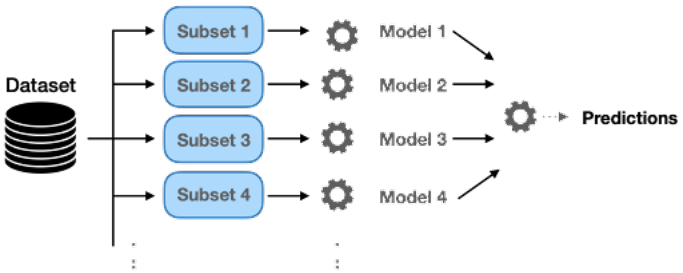
Bagging is a method of ensemble model, which is primarily used to solve supervised machine learning problems.

Here, Multiple models are trained independently on different subsets of the training data, and their outputs are combined by taking an average or majority vote.

Bagging increases bias but reduces variance.

It is generally completed in two steps as follows:

- **Bootstrapping:** It is a random sampling method that is used to derive sample subsets with equal tuples from the training data.
- **Aggregation:** This is a step that involves the process of combining the output of all base models and, based on their output, predicting an aggregate result with greater accuracy and reduced variance.



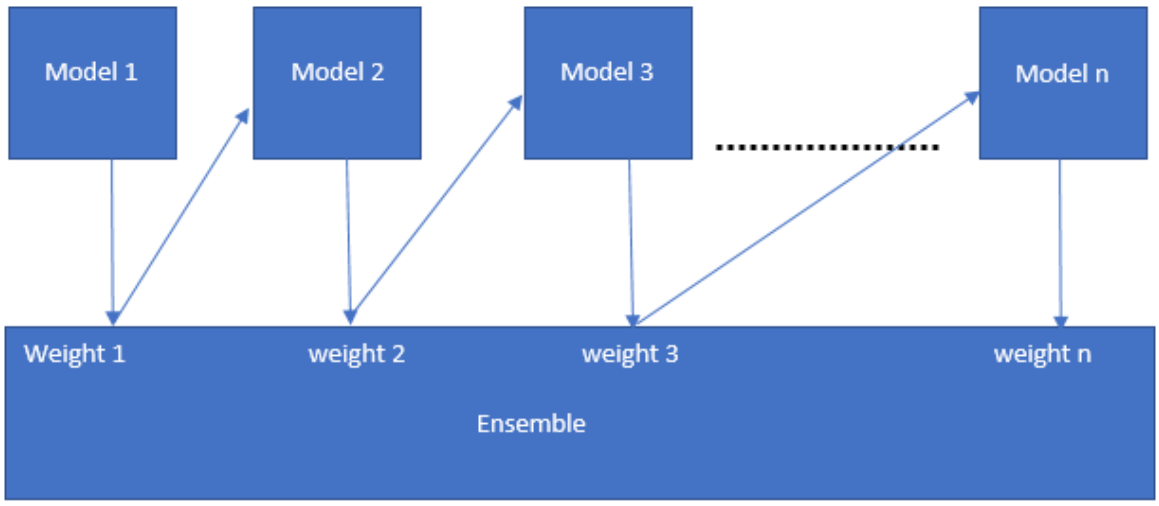
2. Boosting(Homogeneous)

Boosting is an ensemble method that enables each member to learn from the previous member's mistakes and make better predictions for the future.

In this approach, a sequence of models is trained, where each subsequent model focuses on the data that were previously misclassified by the previous models. The outputs of these models are combined in a weighted manner to produce the final prediction.

One of the most popular algorithms that uses boosting is the AdaBoost algorithm, which stands for Adaptive Boosting.

In AdaBoost, a series of weak classifiers are trained sequentially, where each subsequent classifier is trained to correct the mistakes of the previous classifiers. The output of each classifier is combined using a weighted sum to produce the final prediction.



3. Stacking(Heterogeneous)

Stacking is one of the most popular ensemble machine learning techniques used to predict multiple nodes to build a new model and improve model performance.

In this approach, multiple models are trained and their outputs are used as input to a meta-model, which then produces the final prediction.

- **Level-0 Predictions:** Each base model is trained on some training data and provides different predictions.
- **Meta Model:** It helps to combine the predictions of the base models and is trained on different predictions made by individual base models.

VOTING

Voting is a popular ensemble learning technique that involves combining the predictions of multiple models by taking the majority vote (for classification) or the average (for regression).

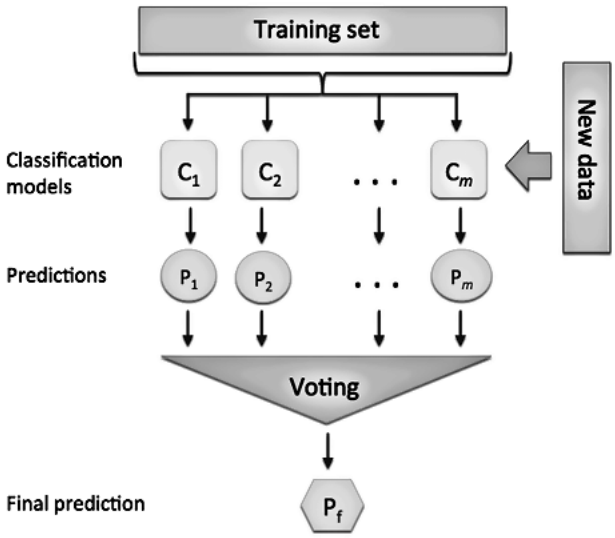
There are different types of voting methods, including:

Hard Voting: The predicted output class is a class with the highest majority of votes. Hard voting is typically used for classification problems.

Example : Suppose three classifiers predicted the *output class(A, A, B)*, so here the majority predicted *A* as output. Hence *A* will be the final prediction.

Soft Voting: The output class is the prediction based on the average of probability given to that class. Soft voting is typically used for regression problems.

Example: Suppose given some input to three models, the prediction probability for class *A = (0.30, 0.47, 0.53)* and *B = (0.20, 0.32, 0.40)*. So the average for class *A is 0.4333* and *B is 0.3067*, the winner is clearly class *A* because it had the highest probability averaged by each classifier.



Bagging & Boosting

RANDOM FOREST AND GRADIENT BOOSTING

Number of trees:

Random Forest creates a large number of decision trees independently and combines them by averaging their predictions,

While Gradient Boosting creates trees iteratively, with each tree attempting to correct the errors of the previous tree.

Sampling:

Random Forest uses bootstrap sampling to create subsets of the training data for each decision tree,

while Gradient Boosting samples the training data using a technique called gradient descent.

Variable selection:

Random Forest selects a random subset of features for each decision tree,

while Gradient Boosting selects features based on their importance in improving the performance of the model.

Output:

Random Forest outputs the average prediction of all the decision trees,

while Gradient Boosting outputs the sum of the predictions of all the decision trees.

Overfitting:

Random Forest tends to overfit less than Gradient Boosting, as each decision tree is constructed independently and the averaging process reduces the impact of individual trees.

Gradient Boosting, on the other hand, can overfit if the number of trees or the learning rate is too high.

Curse of Dimensionality

If there are more dimensions than observations, then there is a risk of massively overfitting the model.
As the number of dimensions (or variables) in a dataset increases, the amount of data required to cover the space increases exponentially, fuels the complexity and the curse of dimensionality occurs.
This can result in a variety of problems like data sparsity, where the number of observations becomes too small compared to the number of dimensions, making it difficult to identify meaningful patterns or relationship
Another problem is Distance concentration where the distance between points in high-dimensional spaces tend to become very similar, making it difficult to differentiate between them.

Dimensionality Reduction

To overcome Curse of Dimensionality we use Dimensionality Reduction
Dimensionality Reduction is the process of reducing the number of input variables(dimensions) in a dataset, by converting the high-dimensional variables into lower-dimensional variables without changing their attributes of the same.

Types of Dimensionality Reduction

Feature selection is a process that involves selecting the most important and relevant features from a dataset.
Feature extraction is a process that involves transforming a set of high-dimensional data into a lower-dimensional representation by creating new features that capture the most important information in the original data.

Principle Component Analysis

PCA, which stands for Principal Component Analysis, is a technique used for dimensionality reduction.
It helps to reduce the number of variables in a dataset while retaining the most important information.
PCA is process in which high dimensional correlated data is transformed to a lower dimensional representation by creating new set of feature components, referred to as principal components that capture as much of this information as possible.
PCA is to find a linear combination of the original features that captures the maximum amount of variation in the data rather than differences in mean.
PCA allows you to interpret the most important patterns and correlations in the data
These principal components are ordered by importance, with the first one capturing the most variation in the data, and the subsequent ones capturing progressively less.

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) is a supervised learning algorithm used for classification tasks in machine learning.
It is a technique used to find a linear combination of features that best separates the classes in a dataset.

LDA works by projecting the data onto a lower-dimensional space that maximizes the separation between the classes and minimises separation within classes.
It does this by finding a set of linear discriminants that maximize the ratio of between-class variance to within-class variance.
In other words, it finds the directions in the feature space that best separate the different classes of data.
When we classify them using a single feature, then it may show overlapping. If we have two classes with multiple features and can separate them efficiently.
LDA assumes that the data has a Gaussian distribution and that the covariance matrices of the different classes are equal.
It also assumes that the data is linearly separable, meaning that a linear decision boundary can accurately classify the different classes.
It can handle multicollinearity (correlation between features) in the data.

Singular value decomposition (SVD)

Singular value decomposition (SVD) is a matrix factorization technique used in linear algebra and machine learning. It decomposes a matrix into three smaller matrices that capture the most important information about the original matrix.
The decomposition can be written as follows: A = UΣV^T
The singular values(A) represent the magnitude of the importance of each singular vector in the matrix.
Given a matrix A, SVD decomposes it into three matrices: U, Σ, and V^T, where U and V^T are orthogonal matrices and Σ is a diagonal matrix containing the singular values of A
SVD has several applications, including image compression, data compression, and feature extraction.
Algorithm:

1. Calculate the matrix of interest that needs to be decomposed.
2. Calculate the transpose of the matrix by interchanging the rows and columns.
3. Multiply the matrix with its transpose to obtain a **square matrix AAT**.
4. Calculate the eigenvectors and eigenvalues of the square matrix
5. Calculate the left singular matrix, it is obtained by multiplying the original matrix with the eigenvectors of the square matrix.
6. Normalize the left and right singular matrices, so that their columns are unit vectors.
7. Combine the matrices: The three matrices obtained are combined to form the final decomposition of the original matrix.

1. Choose the matrix A needs to be decomposed.
2. Calculate the transpose of the matrix AT by interchanging the rows and columns.
3. Multiply the matrix with its transpose to obtain a square matrix AAT.
4. Calculate the eigenvalues of the square matrix AAT using the formula $|AAT - \lambda I| = 0$.
5. Calculate the eigenvectors for each eigenvalues using $AV = \lambda V$
6. They are normalised and combined into unit vectors correspond to the left singular matrix, and the root of eigenvalues correspond to the Σ diagonal matrix of singular values.
7. Multiply the transpose with the original matrix to obtain another square matrix AT A.
8. Calculate the eigenvalues of the square matrix AT A using the formula $|AT\ A - \lambda I| = 0$.

9. Calculate the eigenvectors for each eigenvalues using AV=λV
10. They are normalised and combined into unit vectors correspond to the right singular matrix V^T.
11. Combine the matrices to form the final decomposition of the original matrix.

Activation Functions

• Activation functions (transfer function) are an important component in deep learning models.

• The choice of activation function in the hidden layer will control how well the network model learns the training dataset.

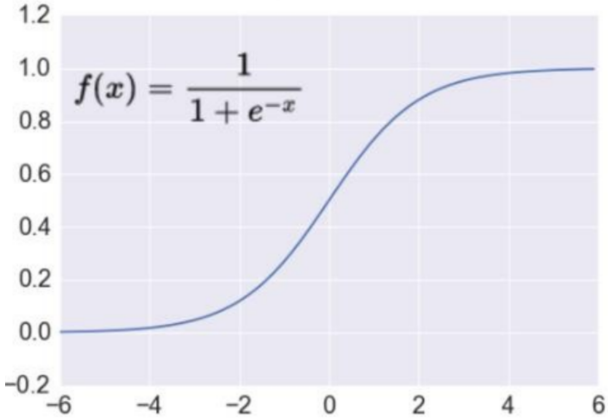
• If the **output range** of the activation function is limited, then it may be called a "squashing function".

• The output layer will typically use a different activation function from the hidden layers and is **dependent upon the type of prediction** required by the model.

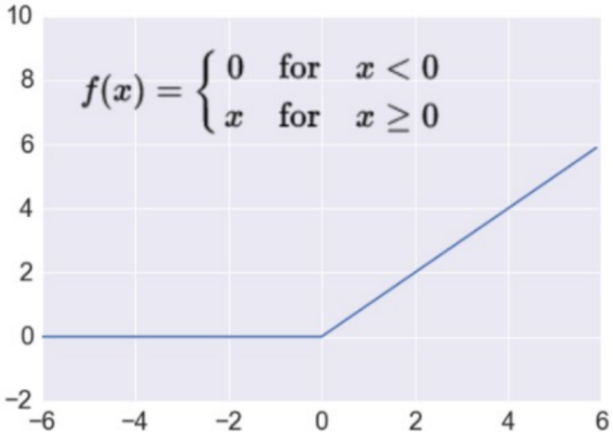
• They introduce non-linearity to the output of a neuron in a neural network.

• Activation functions are attached to each neuron and are mathematical equations that determine whether a neuron should be activated or not based on whether the neuron's input is relevant for the model's prediction or not.

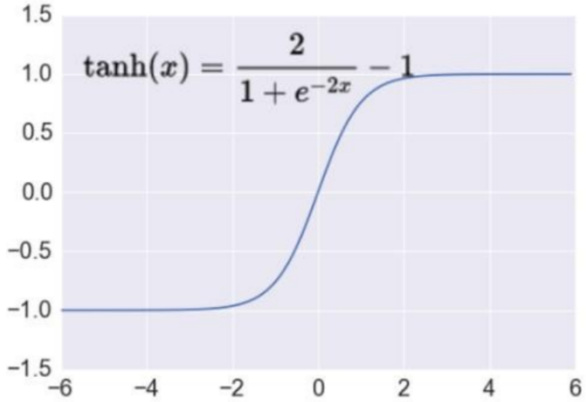
Some commonly used activation functions in deep learning include:
Sigmoid: This function is used in the output layer of a neural network for binary classification tasks. It **maps the input to a value between 0 and 1**, which can be interpreted as the probability of the input belonging to a certain class.



ReLU (Rectified Linear Unit): This function sets all negative values in the input to zero, and leaves positive values unchanged. ReLU is the most commonly used activation function in deep learning because it is computationally efficient and helps prevent the vanishing gradient problem.



Tanh (Hyperbolic Tangent): This function maps the input to a value between -1 and 1, and is commonly used in recurrent neural networks (RNNs)



XOR is where if one is 1 and other is 0 but not both.

Need:
0.w1 + 0.w2 doesn't fire, i.e. < t
1.w1 + 0.w2 cause a fire, i.e. >= t
0.w1 + 1.w2 >= t
1.w1 + 1.w2 also doesn't fire, < t

0 < t
w1 >= t -(1)
w2 >= t -(2)
w1+w2 < t -(3)
if(1) and (2) >=t then –(3) should also be >=t , but –(3) < t
So it is a contradiction.

Note: We need all 4 inequalities for the contradiction. If weights negative, e.g. weights = -4 and t = -5, then weights can be greater than t yet adding them is less than t, but t > 0 stops this.

A "single-layer" perceptron can't implement XOR. The reason is because the classes in XOR are not linearly separable. A perceptron can only converge on linearly separable data. You cannot draw a straight line to separate the points (0,0),(1,1) from the points (0,1),(1,0). Therefore, it isn't capable of imitating the XOR function.
Led to invention of multi-layer networks.

Why MP Neuron Model can't be used instead of perceptron?

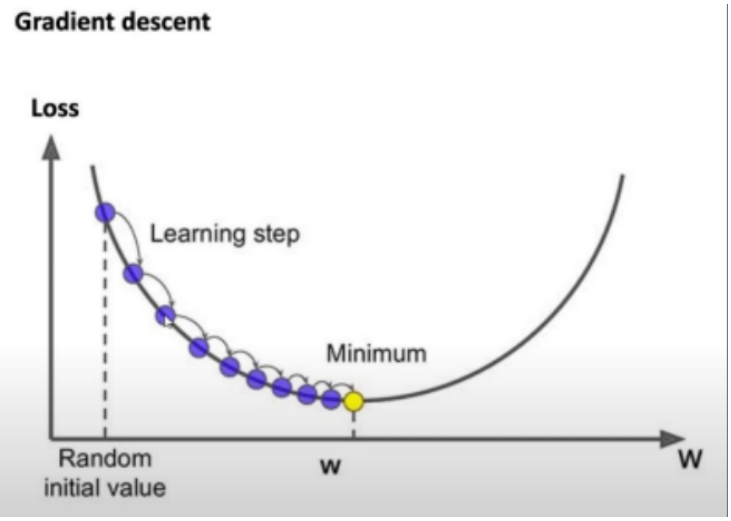
Both, MP Neuron Model as well as the Perceptron model work on linearly separable data.

MP Neuron Model only accepts boolean input whereas **Perceptron Model** can process any real input.

Inputs have fixed weight in MP Neuron Model, which makes this model less flexible. On the other hand, Perceptron model can take weights with respective to inputs provided.

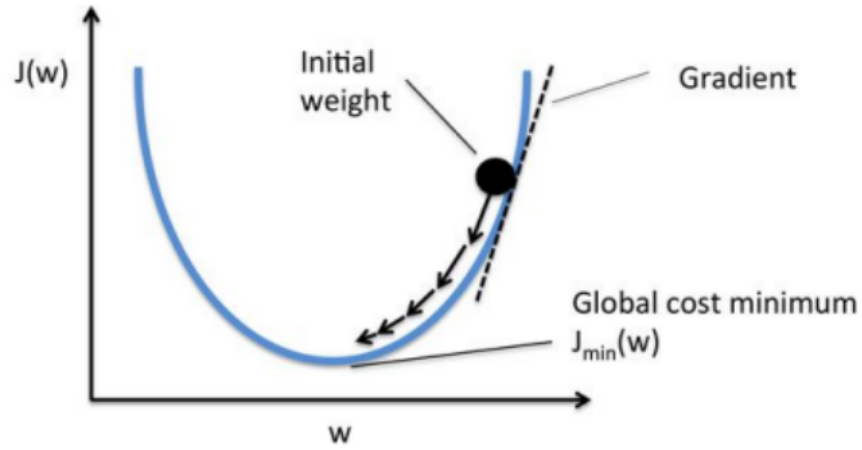
The MCP neuron has a binary output, which is either 0 or 1, based on whether its input exceeds a certain threshold.
In contrast, the Perceptron has a continuous output, which is a function of its weighted inputs passed through an activation function such as the sigmoid or ReLU.

The Perceptron uses a supervised learning algorithm called the Perceptron Learning Rule to adjust its weights during training.
This rule adjusts the weights based on the error between the predicted output and the target output.
In contrast, the MCP neuron does not have a learning algorithm and its weights are fixed.



Gradient descent is a popular optimization algorithm used in machine learning to find the minimum value of a function.
The algorithm works by iteratively adjusting the parameters of the function in the direction of the negative gradient until the minimum is reached.

What is Back Propagation?



Backpropagation is a commonly used algorithm for training neural networks using gradient descent.
Back propagation is the process of updating and finding the optimal values of weights which helps the model to minimize the error i.e difference between the actual and predicted values.
The backpropagation algorithm works by first forward propagating an input through the network, calculates the error or loss in forward propagation and feed this error backwards through the network, computing the gradient of the loss function with respect to the weights of each neuron to fine-tune the weights.

•The weights are updated with the help of optimizers.
•Optimizers are the methods/ mathematical formulations to change the attributes of neural networks i.e weights to minimize the error.