**Machine Learning Algorithms**

# **LR = LogisticRegression()**

Logistic regression is a type of regression analysis used for predicting the probability of an event occurring, especially in binary classification problems where the output can be one of two classes (e.g., spam or not spam, malignant or benign, fake or true).

**Logistic Regression Model:** Logistic regression models the probability that a given instance belongs to a particular category. The logistic function (also called the sigmoid function) is used to map any real-valued number into a value between 0 and 1, which can be interpreted as a probability.

**Usage**

Once you've created an instance of the logistic regression model using LogisticRegression(), you typically use it in the following steps:

* Data Preparation:
  + Load and preprocess your dataset. This may involve tasks such as handling missing values, scaling features, or encoding categorical variables.
* Splitting Data:
  + Split your dataset into training and testing sets. The training set is used to train the model, and the testing set is used to evaluate its performance on unseen data.
* Training:
  + Use the fit method to train the logistic regression model on your training data. This involves adjusting the model's parameters to minimize the difference between the predicted probabilities and the actual class labels.

#Logistic Regression model

from sklearn.linear\_model import LogisticRegression

LR = LogisticRegression()

**LR.fit(xvect\_train, y\_train)** #xvect\_train is the feature matrix of the training set and y\_train is the corresponding vector of labels

pred\_lr = LR.predict(xvect\_test)

#Accuracy score for Logistic Regression model

LR.score(xvect\_test, y\_test)

#Print classification report. The classification\_report function builds a text report showing the main classification metrics

print(classification\_report(y\_test, pred\_lr))

Here, xvect\_train is the feature matrix of the training set, and y\_train is the corresponding vector of labels.

pred\_lr = LR.predict(xvect\_test) make the model predict on testing set.

LR.score(xvect\_test, y\_test) here, we evaluate the performance of the trained model on the testing set to see how well it generalizes to new, unseen data. Here, xvect\_test is the feature matrix of the testing set and y\_train in the label matrix of the testing set.

# **Decision Tree Classifier model**

A Decision Tree Classifier is a type of machine learning algorithm that falls under the category of supervised learning. It is used for both classification and regression tasks. In the context of classification, the algorithm makes decisions based on the input features to assign a data point to one of the predefined classes.

**Decision Tree Structure:**A Decision Tree is a tree-like model where each internal node represents a decision based on the value of a specific feature. The branches leaving each node represent the possible outcomes of the decision, and the leaves represent the predicted class labels.

* Training Process :During the training process, the Decision Tree algorithm recursively splits the data based on feature values to create a tree that can make accurate predictions. The splitting process is guided by criteria such as Gini impurity or information gain, depending on the specific implementation.
* Decision Tree Structure: A Decision Tree is a tree-like model where each internal node represents a decision based on the value of a specific feature. The branches leaving each node represent the possible outcomes of the decision, and the leaves represent the predicted class labels.

In my code, Decision Tree Classifier is trained on a dataset, and its performance is evaluated on a testing set. The classification report includes metrics such as precision, recall, and F1-score for each class. The model is then used to make predictions on a user-input news article in the manual\_testing\_dt function.

# **Gradient Boosting Classifier model**

A Gradient Boosting Classifier is an ensemble machine learning algorithm that combines the predictions of multiple weak learners (typically decision trees) to create a strong predictive model. It belongs to the family of boosting algorithms, which sequentially builds a series of weak models and adjusts them to correct errors made by the previous ones. Gradient Boosting is particularly powerful and widely used for both regression and classification tasks.

GradientBoostingClassifier(random\_state=0). The random\_state parameter is set for reproducibility.

This algorithm builds an additive model in a forward stage-wise fashion; it allows for the optimization of arbitrary differentiable loss functions. In each stage n\_classes\_ regression trees are fit on the negative gradient of the loss function, e.g. binary or multiclass log loss. Binary classification is a special case where only a single regression tree is induced.

# **Random Forest classifier model**

Random forest (RF) is an advanced form of decision trees (DT) which is also a supervised learning model. RF consists of large number of decision trees working individually to predict an outcome of a class where the final prediction is based on a class that received majority votes. The error rate is low in random forest as compared to other models, due to low correlation among trees.

# Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification and regression tasks. In the context of classification, SVM is particularly powerful for tasks where the goal is to separate data into different classes based on their features. SVM finds the optimal hyperplane that maximally separates the instances of different classes in the feature space.

Key Concepts of SVM:

* Hyperplane:

In a two-dimensional space, a hyperplane is a line that separates two classes. In higher dimensions, it becomes a plane or a hyperplane.

* Support Vectors:

Support vectors are the data points that are closest to the hyperplane and have a significant impact on determining the optimal hyperplane.

* Margin:

The margin is the distance between the hyperplane and the nearest data point from either class. SVM aims to maximize this margin to achieve better generalization.

* Kernel Trick:

SVM can handle non-linear decision boundaries by mapping the input features into a higher-dimensional space using a kernel function.