# Multi-Level Classification ML Project to Predict Churn in Telecom

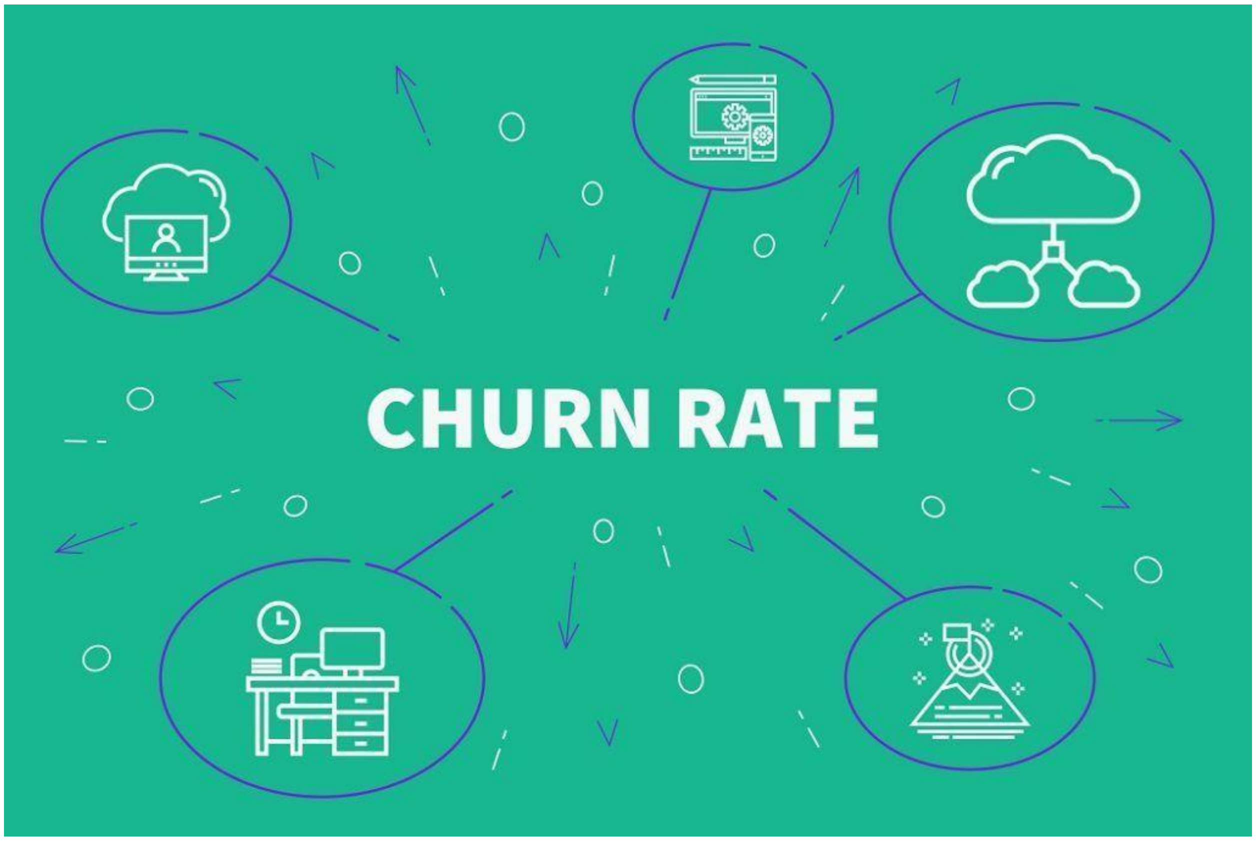
## Introduction

An organization loses its customers to its competition for various reasons. What mainly attracts customers to go ahead with a shift is attributed to the price of the product and its quality. This is true for the Telecom industry as well. A churn of 10 to 60 percent in any company is a big number that can affect the company's overall growth.

Churn is the percentage of customers who stop using a company's product or service during a particular period of time. Churn category refers to the reason why a customer stops using a company's service, while churn reason refers to the specific reason for a customer to churn. In the telecom industry, multiclass classification is used to predict a customer's churn category and churn reason.

This Project is part of the "telecom data" cluster of projects, targeting the telecom industry. It is recommended to go through the "Exploratory Data Analysis" notebook before executing this project to understand the Data better.

The main goal of multiclass and multilevel classification in telecom is to predict the churn category and churn reason of a customer, which can help a telecom company to take proactive measures to reduce customer churn, improve customer retention and enhance customer satisfaction. By understanding the reasons behind churn, a telecom company can develop targeted retention strategies to retain customers.



Credit: Retently

## Business Impact of Churn Classification

**Increase revenue with customer retention**: Reduced churn means company is not losing out against it competitors and a happy customer would keep on spending money on the platform

**Improve customer acquisition cost**: If the company is able to stop old customers from leaving then it doesn't need to spend extra money on getting new customers or in an extreme case throw offers to get the old customer back. This impacts cost of acquisition per customer

**Improve customer satisfaction**: If the company is able to identify which customer will churn and reason behind that, it can fix the problem and in the end improve customer satisfaction

## Why Data Science?

Companies are typically blind to future churn and only come to know once a customer leaves the platform. In better cases, even if they know past reasons of customers leaving they can only combines few of these reasons to come up with few heuristics which will them if a customer is going to churn.

Often they have vast majority of data along with signals of the churned customers and the reason. A machine learning model if supplied with such rich and crucial data can do wonders for the company.

Data science enables these companies to build a model which can help them understand the likelihood of a customer churning and the reason why the customer churned. Model understands the historical nuances and builds the internal model parameters so as to tell the company about possibility of a churn which a heuristics based approach cannot do owing to hundreds of parameters which a human cannot make sense of.

# Possible Solutions

## Methods for multi-level churn classification

There are various ways to build a model for multi-level churn classification, and the choice depends on factors such as the data available, the complexity of the system, and the computational resources available. Some approaches are:

### 1. Rule-based systems:

In this approach, we typically look at churn classification based on predefined rules. Rule-based systems are easy to implement and interpret, but they may not be very accurate or adaptive. It makes sense to use them when you have a specific business mandate (example - one ro two reasons leading to churn).

### 2. Statistical techniques:

Statistical techniques such as correlation, probability are very helpful to predict churn but they don't scale well as the number of signals increase. In this approach, we use historical data to come up with priors and correlations to do churn classification.

### 3. Machine learning:

Machine learning systems combines strength of historical data and statistical techniques to explain the right churn reason for individual customers of the company. For example, a ML system can tell with a high confidence if a potentail customer will churn or not and what is the reason for the churn?

### Assumptions

- We assume that **Churn Category & Churn Reason** are our target variables.
- We assume that whenever the churn reason is attributed to wrong churn category, we do a switch
- Churn category identification is a multi-class classification and we do that first. Post that we incorporate churn reason and formulate the problem as a multi-label classification problem.
- *Churn Category = Not Applicable* denotes that the customer didn't churn and it is our negative label.
- Columns with a lot of null values are not meaningful and imputation also won't be helpful.
- Most of the univariate EDA is already done over this data.
- Most of the missing values are imputed(some are remaining as they can be imputed according to the problem).

## Approach

We are treating this problem as a supervised learning problem doing multi-label classification. So every data point will have multiple target variables(in this case it would be 2, i.e., churn category and churn reason) for the model to learn the dependencies and predict on the unknown.

In real life, this model would tell the business that which category of churn does a user lie in and the reason behind the churn. It would in turn help the company to proactively prevent customers from leaving the platform.

Given our assumptions about the data, we will build a prediction model based on the historical data. Simplifying, here's the logic of what we'll build:

1. We will try to understand the churn data first and do a detailed problem specific EDA;
2. We'll build a model to predict the churn category using multi-class classification;
3. We'll then formulate the problem as a multi-label classification problem and predict both churn category and churn reason

**Supervised Machine Learning:**

In supervised machine learning, the algorithm is trained on an labeled dataset with a predefined target variable. The goal is to identify patterns, relationships, and structures of the data with the target variable, such as logistic regression, decision tree or boosting trees

**Multi-Class Classification:**

Multiclass classification is a type of machine learning task where the goal is to classify instances into one of three or more classes. In other words, given a set of input data, the task is to predict the class label of the instance, where the instance can belong to any one of the several predefined classes.

For example, in a medical diagnosis application, the goal may be to predict whether a patient has a certain disease, where the disease can be one of several possibilities. In this case, the output of the classification model would be a probability distribution over the possible classes, with the highest probability indicating the predicted class.

There are various algorithms that can be used for multiclass classification, including logistic regression, decision trees, random forests, support vector machines, and neural networks. The choice of algorithm depends on various factors, such as the nature of the data, the number of classes, and the available computing resources.

**Multi-Label Classification:**

Multilabel classification is a type of machine learning task where the goal is to assign one or more class labels to an instance. Unlike multiclass classification, where an instance can belong to only one class, in multilabel classification, an instance can belong to more than one class simultaneously.

For example, in a news article categorization task, an article may belong to multiple categories, such as "politics", "world news", and "entertainment". In this case, the task would be to predict a set of labels that best describe the content of the article.

Multilabel classification is used in a variety of applications, such as text classification, image annotation, and recommendation systems, where multiple labels can be assigned to an instance based on its content or characteristics.

# Exploratory Data Analysis

## Data Exploration

Data exploration is a critical step in the data analysis process, where you examine the dataset to gain a preliminary understanding of the data, detect patterns, and identify potential issues that may need further investigation. Data exploration is important because it helps to provide a solid foundation for subsequent data analysis tasks, hypothesis testing and data visualization.

Data exploration is also important because it can help you to identify an appropriate approach for analyzing the data.

Here are the various functions that help us explore and understand the data.

- Shape: Shape is used to identify the dimensions of the dataset. It gives the number of rows and columns present in the dataset. Knowing the dimensions of the dataset is important to understand the amount of data available for analysis and to determine the feasibility of different methods of analysis.
- Head: The head function is used to display the top five rows of the dataset. It helps us to understand the structure and organization of the dataset. This function gives an idea of what data is present in the dataset, what the column headers are, and how the data is organized.
- Tail: The tail function is used to display the bottom five rows of the dataset. It provides the same information as the head function but for the bottom rows. The tail function is particularly useful when dealing with large datasets, as it can be time-consuming to scroll through all the rows.
- Describe: The describe function provides a summary of the numerical columns in the dataset. It includes the count, mean, standard deviation, minimum, and maximum values, as well as the quartiles. It helps to understand the distribution of the data, the presence of any outliers, and potential issues that can affect the model's accuracy.
- Isnull: The isnull function is used to identify missing values in the dataset. It returns a Boolean value for each cell, indicating whether it is null or not. This function is useful to identify the presence of missing data, which can be problematic for regression analysis.
- Dropna: The dropna function is used to remove rows or columns with missing data. It is used to remove any observations or variables with missing data, which can lead to biased results in the regression analysis. The dropna function is used after identifying the missing data with the isnull function.
- Columns: The .columns method is a built-in function that is used to display the column names of a pandas DataFrame or Series. It returns an array-like object that contains the

names of the columns in the order in which they appear in the original DataFrame or Series. It can be used to obtain a quick overview of the variables in a dataset and their names.

## Boxplot

A box plot, also known as a box-and-whisker plot, is a type of data visualization that displays the distribution of a set of numerical data. It provides a summary of the data's central tendency, variability, and any potential outliers.

A box plot consists of a rectangle, which represents the middle 50% of the data, and "whiskers" that extend from the rectangle to represent the range of the data that falls within 1.5 times the interquartile range (IQR) from the edges of the rectangle. The IQR is the distance between the first and third quartiles of the data. Any data points that fall outside the whiskers are considered outliers and are plotted as individual points.

In a standard box plot, the rectangle is positioned vertically, with the bottom edge representing the first quartile (Q1) of the data, the top edge representing the third quartile (Q3), and the line inside the rectangle representing the median of the data. However, variations of box plots exist, such as horizontal box plots, notched box plots, and violin plots.

Box plots are useful for comparing the distributions of multiple data sets, identifying potential outliers, and visualizing the spread and skewness of the data. They are commonly used in statistical analysis, data exploration, and reporting.

# Data Processing & Feature engineering

### Data Preprocessing and Leakage

Data leakage is a situation where information from the test or prediction data is inadvertently used during the training process of a machine learning model. This can occur when information from the test or prediction data is leaked into the training data, and the model uses this information to improve its performance during the training process.

Data leakage can occur during the preprocessing phase of machine learning when information from the test or prediction data is used to preprocess the training data, inadvertently leaking information from the test or prediction data into the training data.

For example, consider a scenario where the preprocessing step involves imputing missing values in the dataset. If the missing values are imputed using the mean or median values of the entire dataset, including the test and prediction data, then the imputed values in the training data may be influenced by the values in the test and prediction data. This can lead to data leakage, as the model may learn to recognize patterns in the test and prediction data during the training process, leading to overfitting and poor generalization performance.

To avoid data leakage, it's important to perform the data preprocessing steps on the training data only, and then apply the same preprocessing steps to the test and prediction data separately. This ensures that the test and prediction data remain unseen by the model during the training process, and helps to prevent overfitting and improve the accuracy of the model.

In the context of this problem, we will perform data preprocessing steps together for the sake of simplicity, which could potentially lead to data leakage. However, in real-world scenarios, it's important to treat the test and prediction data separately and apply the necessary preprocessing steps separately, based on the characteristics of the data.

## Dropping Irrelevant Features and IDs

Irrelevant features in a dataset are those that do not contribute to the predictive power of the model, and including them in the feature set can negatively affect the model's performance. Therefore, dropping irrelevant features during feature engineering is important for several reasons:

1. Reduce overfitting: Including irrelevant features can cause overfitting, where the model fits the noise in the data instead of the underlying patterns. This can result in a model that performs well on the training data but poorly on the test data. By dropping irrelevant features, we can reduce the complexity of the model and prevent overfitting.
2. Improve interpretability: Including irrelevant features can make the model harder to interpret, as it becomes difficult to understand which features are truly important in making predictions. Dropping irrelevant features can make the model simpler and more interpretable.
3. Reduce computational complexity: Including irrelevant features can increase the computational complexity of the model, which can result in longer training times and higher memory usage. Dropping irrelevant features can reduce the computational complexity of the model and make it more efficient.
4. Improve model performance: Dropping irrelevant features can improve the performance of the model by reducing noise and focusing on the features that are truly important in making predictions.

Therefore, it is important to carefully select the features that are relevant to the problem at hand and drop the irrelevant ones during feature engineering to build a better-performing and more interpretable model.

## Label Encoding

### Transforming Categorical Variables

Transforming variables is an important step in the data preprocessing pipeline of machine learning, as it helps to convert the data into a format that is suitable for analysis and modeling. There are several ways to transform variables, depending on the type and nature of the data.

Categorical variables, for example, are variables that take on discrete values from a finite set of categories, such as colors, gender, or occupation. One common way to transform categorical variables is through one-hot encoding. One-hot encoding involves creating a new binary variable for each category in the original variable, where the value is 1 if the observation belongs to that category and 0 otherwise. This approach is useful when the categories have no natural order or ranking.

Another way to transform categorical variables is through label encoding. Label encoding involves assigning a unique integer value to each category in the variable. This approach is useful when the categories have a natural order or ranking, such as low, medium, and high. Transforming categorical features into numerical labels:

**Note:** We are NOT using dummies here to minimize the explosion of columns because of the distance methods we are using.

## SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) is a technique used in machine learning to address class imbalance in a dataset. Class imbalance occurs when the number of instances in one class is much lower than the number of instances in another class, making it difficult for machine learning algorithms to learn from the data and predict the minority class accurately.

SMOTE works by creating synthetic samples from the minority class by interpolating new instances between existing instances. The new instances are created by selecting pairs of instances that are close to each other in the feature space and generating new instances along the line that connects them. The number of new instances to be generated is determined by a user-defined parameter that specifies the desired ratio of minority to majority class instances.

The synthetic instances generated by SMOTE are used to balance the classes in the dataset, allowing the machine learning algorithm to learn from a more balanced dataset and make better predictions on the minority class.

**Applications**

SMOTE is commonly used in machine learning applications such as fraud detection, medical diagnosis, and anomaly detection, where the minority class is often of greater interest than the majority class.

## Undersampling

Undersampling is a technique used in machine learning to address class imbalance in a dataset. Class imbalance occurs when the number of instances in one class is much lower than the number of instances in another class, making it difficult for machine learning algorithms to learn from the data and predict the minority class accurately.

Undersampling works by randomly selecting a subset of instances from the majority class so that the number of instances in the majority class is reduced to a level comparable to the number of instances in the minority class. This creates a more balanced dataset and allows the machine learning algorithm to learn from a more representative sample of the data.

Undersampling can be effective in reducing the computational cost and training time of machine learning models, as well as reducing the risk of overfitting to the majority class.

**Applications**

Undersampling is commonly used in machine learning applications such as credit scoring, fraud detection, and medical diagnosis, where the minority class is often of greater interest than the majority class.

## 1. Multi-class classification to predict Churn Category

## Supervised learning

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Supervised learning can be separated into two types of problems when data mining—classification and regression:

1. Classification uses an algorithm to accurately assign test data into specific categories. It recognizes specific entities within the dataset and attempts to draw some conclusions on how those entities should be labeled or defined. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor, and random forest, which are described in more detail below.
2. Regression is used to understand the relationship between dependent and independent variables. It is commonly used to make projections, such as for sales revenue for a given business. Linear regression, logistical regression, and polynomial regression are popular regression algorithms.

## Decision Trees

### Decision Trees in Classification

Decision trees are a type of supervised learning algorithm that can be used for classification as well as regression problems. They are widely used in machine learning because they are easy to understand and interpret, and can handle both categorical and numerical data. The idea behind decision trees is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

The decision tree starts with a single node, called the root node, which represents the entire dataset. The root node is then split into several child nodes based on the value of a chosen feature. The process of selecting the best feature and splitting the nodes is repeated recursively for each child node until a stopping criterion is reached. This results in a tree-like structure that represents the decision rules learned from the data.

Each node in the decision tree represents a decision or a test of a feature value, and each branch represents the possible outcomes of that decision. The leaves of the tree represent the final decision or the class label assigned to the input data.

**Splitting Criteria**

To build a decision tree, we need a measure that determines how to split the data at each node. The splitting criterion is chosen based on the type of data and the nature of the problem. The most common splitting criteria are:

- Gini index: measures the impurity of a set of labels. It calculates the probability of misclassifying a randomly chosen element from the set, and is used to minimize misclassification errors.
- Information gain: measures the reduction in entropy (uncertainty) after a split. It is used to maximize the information gain in each split.
- Chi-square: measures the difference between observed and expected frequencies of the classes. It is used to minimize the deviation between the observed and expected class distribution.

**Overfitting in Decision Trees**

One of the main challenges in building decision trees is overfitting. Overfitting occurs when the tree is too complex and fits the training data too well, resulting in poor performance on new and unseen data. This can be addressed by pruning the tree or limiting its depth, or by using ensemble methods such as bagging and boosting.

**Ensemble Methods**

Ensemble methods are techniques that combine multiple models to improve performance and reduce overfitting. The two most common ensemble methods used with decision trees are:

- Bagging (Bootstrap Aggregating): involves training multiple decision trees on different subsets of the training data and then combining their predictions by averaging or voting. This reduces the variance and improves the stability of the model.
- Boosting: involves training multiple decision trees sequentially, where each subsequent tree focuses on the misclassified examples of the previous tree. This reduces the bias and improves the accuracy of the model.

Deecision trees are powerful tools for classification problems that provide a clear and interpretable representation of the decision rules learned from the data. The choice of splitting criterion, stopping criterion, and ensemble method can have a significant impact on the performance and generalization of the model.

## Bagging

Bagging is an ensemble learning technique that aims to decrease the variance of a single estimator by combining the predictions from multiple learners. The basic idea behind bagging is to generate multiple versions of the training dataset through random sampling with replacement, and then train a separate classifier for each sampled dataset. The predictions from these individual classifiers are then combined using averaging or voting to obtain a final prediction.

**Algorithm:**

Suppose we have a training set D of size n, and we want to train a classifier using bagging. Here are the steps involved:

- Create k different bootstrap samples from D, each of size n.
- Train a classifier on each bootstrap sample.
- When making predictions on a new data point, take the average or majority vote of the predictions from each of the k classifiers.

**Mathematical Explanation:**

Suppose we have a binary classification problem with classes -1 and 1. Let's also assume that we have a training set D of size n, and we want to train a decision tree classifier using bagging.

**Bootstrap Sample**: For each of the k classifiers, we create a bootstrap sample of size n by sampling with replacement from D. This means that each bootstrap sample may contain duplicates of some instances and may also miss some instances from the original dataset. Let's denote the i-th bootstrap sample as D_i.

**Train a Classifier**: We train a decision tree classifier T_i on each bootstrap sample D_i. This gives us k classifiers T_1, T_2, ..., T_k.

**Combine Predictions**: To make a prediction on a new data point x, we take the majority vote of the predictions from each of the k classifiers.

The idea behind bagging is that the variance of the prediction error decreases as k increases. This is because each classifier has a chance to explore a different part of the feature space due to the random sampling with replacement, and the final prediction is a combination of these diverse classifiers.

## Random Forest

Random Forest is an ensemble learning algorithm that builds a large number of decision trees and combines them to make a final prediction. It is a type of bagging method, where multiple decision trees are trained on random subsets of the training data and features. The algorithm then averages the predictions of these individual trees to produce a final prediction. Random Forest is particularly useful for handling high-dimensional data and for avoiding overfitting.

### Algorithm of Random Forest

The algorithm of Random Forest can be summarized in the following steps:

- Start by randomly selecting a subset of the training data, with replacement. This subset is called the bootstrap sample.
- Next, randomly select a subset of features from the full feature set.
- Build a decision tree using the bootstrap sample and the selected subset of features. At each node of the tree, select the best feature and split the data based on the selected feature.
- Repeat steps 1-3 to build multiple trees.
- Finally, combine the predictions of all trees to make a final prediction. For classification, this is usually done by taking a majority vote of the predicted classes. For regression, this is usually done by taking the average of the predicted values.

### Mathematics Behind Random Forest

The mathematics behind Random Forest involves the use of decision trees and the bootstrap sampling technique. Decision trees are constructed using a recursive binary partitioning algorithm that splits the data based on the values of the selected features. At each node, the algorithm chooses the feature and the split point that maximizes the information gain. Information gain measures the reduction in entropy or impurity of the target variable after the split. The goal is to minimize the impurity of the subsets after each split.

Bootstrap sampling is a statistical technique that involves randomly sampling the data with replacement to create multiple subsets. These subsets are used to train individual decision trees. By using bootstrap samples, the algorithm can generate multiple versions of the same dataset with slightly different distributions. This introduces randomness into the training process, which helps to reduce overfitting.

### Difference between Bagging and Random Forest

Bagging and Random Forest are both ensemble learning algorithms that involve training multiple models on random subsets of the data. The main difference between the two is the way the individual models are trained.

Bagging involves training multiple models using the bootstrap sampling technique, but each model uses the same set of features. This can lead to correlated predictions, which reduces the variance but not necessarily the bias of the model.

Random Forest, on the other hand, involves training multiple models using the bootstrap sampling technique, but each model uses a randomly selected subset of features. This introduces additional randomness into the model and helps to reduce the correlation between individual predictions. Random Forest can achieve better performance than Bagging, especially when dealing with high-dimensional data or noisy features. In simpler terms it uses subsets of observations as well as features.

## Boosting

Boosting is a machine learning algorithm that works by combining several weak models (also known as base learners) into a strong model. The goal of boosting is to reduce the bias and variance of the base learners by iteratively adding new models to the ensemble that focus on correcting the errors made by the previous models. In other words, the boosting algorithm tries to learn from the mistakes of the previous models and improve the overall accuracy of the ensemble.

Boosting works by assigning higher weights to the data points that the previous models misclassified, and lower weights to the ones that were classified correctly. This ensures that the new model focuses more on the difficult data points that the previous models struggled with, and less on the ones that were already well-classified. As a result, the new model is more specialized and can improve the accuracy of the ensemble.

There are several types of boosting algorithms, including AdaBoost (Adaptive Boosting), Gradient Boosting, and XGBoost (Extreme Gradient Boosting). Each of these algorithms has its own approach to assigning weights to the data points and building the new models, but they all share the fundamental idea of iteratively improving the accuracy of the ensemble by combining weak models into a strong one. Boosting is a powerful algorithm that has been shown to achieve state-of-the-art results in many machine learning tasks, such as image classification, natural language processing, and recommender systems.

### Difference between Bagging and Boosting

It's important to remember that boosting is a generic method, not a specific model, in order to comprehend it. Boosting involves specifying a weak model, such as regression or decision trees, and then improving it. In Ensemble Learning, the primary difference between Bagging and Boosting is that in bagging, weak learners are trained in simultaneously, but in boosting, they are trained sequentially. This means that each new model iteration increases the weights of the prior model's misclassified data. This redistribution of weights aids the algorithm in determining which parameters it should focus on in order to increase its performance.

Both the Ensemble techniques are used in a different way as well. Bagging methods, for example, are often used on poor learners who have large variance and low bias such as decision trees because they tend to overfit, whereas boosting methods are employed when there is low variance and high bias. While bagging can help prevent overfitting, boosting methods are more vulnerable to it because of a simple fact they continue to build on weak learners and continue to minimise error. This can lead to overfitting on the training data but specifying a decent number of models to be generated or hyperparameter tuning, regularization can help in this case, if overfitting encountered.

## Gradient Boosting

The primary idea behind this technique is to develop models in a sequential manner, with each model attempting to reduce the mistakes of the previous model.The additive model, loss function, and a weak learner are the three fundamental components of Gradient Boosting.

The method provides a direct interpretation of boosting in terms of numerical optimization of the loss function using Gradient Descent. We employ Gradient Boosting Regressor when the target column is continuous, and Gradient Boosting Classifier when the task is a classification problem. The "Loss function" is the only difference between the two. The goal is to use gradient descent to reduce this loss function by adding weak learners. Because it is based on loss functions, for regression problems, Mean squared error (MSE) will be used, and for classification problems, log-likelihood.

## XG Boost

XGBoost is a variant of gradient boosting, which is a popular ensemble learning technique that works by iteratively adding new models to an ensemble, each model attempting to correct the errors made by the previous models. In each iteration, the algorithm calculates the negative gradient of the loss function with respect to the current prediction, and fits a new model to the residual errors. The new model is then added to the ensemble, and the algorithm repeats this process until the desired number of models is reached.

In XGBoost, the objective function is used to measure the difference between the predicted values and the true labels. The objective function is a sum of the loss function and the regularization term, where the latter prevents overfitting and encourages the model to be simple.

Suppose we have a dataset with three features, x1, x2, and x3, and we want to predict a binary outcome, y. We decide to use decision trees as our weak learners. We start by training a decision tree on the entire dataset. However, this decision tree may not be able to capture the complex relationships between the features and the outcome, and it may be overfitting the training data.

To improve upon the first decision tree, we can use XGBoost. Here's how:

- Initialize the model: We start by initializing the XGBoost model with default hyperparameters. This model will be a simple decision tree with a single split.
- Make predictions: We use this model to make predictions on the training data. We compare these predictions to the true labels and calculate the residuals, which are the differences between the predicted values and the true labels.
- Fit a new tree: We then fit a new decision tree to the residuals. This tree will be a weak learner, as it is only modeling the errors of the previous model.
- Combine the models: We add the new tree to the previous model to create a new ensemble. This new ensemble consists of the previous model plus the new tree.
- Repeat: We repeat steps 2-4 for a specified number of iterations, adding a new tree to the ensemble each time.
- Predictions: To make predictions on new data, we combine the predictions of all the trees in the ensemble.

The key idea behind XGBoost is that it improves upon the predictions of the weak learners by focusing on the misclassified data points. By fitting a new tree to the residuals, XGBoost can correct the errors of the previous model and improve its overall accuracy. Additionally, XGBoost uses regularization to prevent overfitting and to improve generalization performance.

## Classification Evaluation Metrics

Classification evaluation metrics are used to evaluate the performance of a machine learning model that is trained for classification tasks. Some of the commonly used classification evaluation metrics are F1 score, recall score, confusion matrix, and ROC AUC score. Here's an overview of each of these metrics:

**F1 score**: The F1 score is a metric that combines the precision and recall of a model into a single value. It is calculated as the harmonic mean of precision and recall, and is expressed as a value between 0 and 1, where 1 indicates perfect precision and recall. F1 score is the harmonic mean of precision and recall. It is calculated as follows:

$$F1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

where precision is the number of true positives divided by the sum of true positives and false positives, and recall is the number of true positives divided by the sum of true positives and false negatives.

**Recall**: Use the recall score when the cost of false negatives (i.e., missing instances of a class) is high. For example, in a medical diagnosis problem, the cost of missing a positive case may be high, so recall would be a more appropriate metric. Recall score (also known as sensitivity) is the number of true positives divided by the sum of true positives and false negatives. It is given by the following formula:

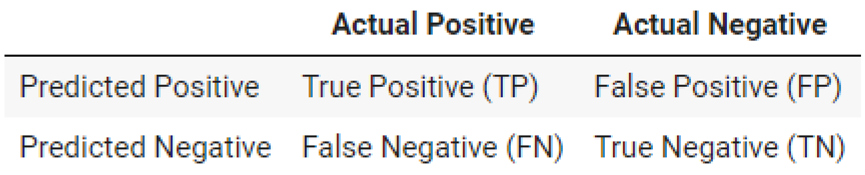
$$Recall = \frac{TP}{TP + FN}$$

**Precision**: Precision is another important classification evaluation metric, which is defined as the ratio of true positives to the total predicted positives. It measures the accuracy of positive predictions made by the classifier, i.e., the proportion of positive identifications that were actually correct. The formula for precision is:

$$precision = \frac{true\ positive}{true\ positive + false\ positive}$$

where true positive refers to the cases where the model correctly predicted the positive class, and false positive refers to the cases where the model incorrectly predicted the positive class. Precision is useful when the cost of false positives is high, such as in medical diagnosis or fraud detection, where a false positive can have serious consequences. In such cases, a higher precision indicates that the model is better at identifying true positives and minimizing false positives.

**Confusion Matrix**: A confusion matrix is a table that is often used to describe the performance of a classification model. It compares the predicted labels with the true labels and counts the number of true positives, false positives, true negatives, and false negatives. Here is an example of a confusion matrix:



| | Actual Positive | Actual Negative |
|---|---|---|
| Predicted Positive | True Positive (TP) | False Positive (FP) |
| Predicted Negative | False Negative (FN) | True Negative (TN) |

**ROC AUC Score**: ROC AUC (Receiver Operating Characteristic Area Under the Curve) score is a measure of how well a classifier is able to distinguish between positive and negative classes. It is calculated as the area under the ROC curve. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. TPR is the number of true positives divided by the sum of true positives and false negatives, and FPR is the number of false positives divided by the sum of false positives and true negatives.

$$ROC\ AUC\ Score = \int_0^1 TPR(FPR^{-1}(t))dt$$

where $FPR^{-1}$ is the inverse of the FPR function.

**When to use which**:

The choice of evaluation metric depends on the specific requirements of the business problem. Here are some general guidelines:

- F1 score: Use the F1 score when the class distribution is imbalanced, and when both precision and recall are equally important.
- Recall score: Use the recall score when the cost of false negatives (i.e., missing instances of a class) is high. For example, in a medical diagnosis problem, the cost of missing a positive case may be high, so recall would be a more appropriate metric.
- Precision: Precision is useful when the cost of false positives is high, such as in medical diagnosis or fraud detection, where a false positive can have serious consequences. In such cases, a higher precision indicates that the model is better at identifying true positives and minimizing false positives.
- Confusion matrix: The confusion matrix is a versatile tool that can be used to visualize the performance of a model across different classes. It can be useful for identifying specific areas of the model that need improvement.

- ROC AUC score: Use the ROC AUC score when the ability to distinguish between positive and negative classes is important. For example, in a credit scoring problem, the ability to distinguish between good and bad credit risks is crucial.

Importance with respect to the business problem:

The importance of each evaluation metric varies depending on the business problem. For example, in a spam detection problem, precision may be more important than recall, since false positives (i.e., classifying a non-spam email as spam) may annoy users, while false negatives (i.e., missing a spam email) may not be as harmful. On the other hand, in a disease diagnosis problem, recall may be more important than precision, since missing a positive case (i.e., a false negative) could have serious consequences. Therefore, it is important to choose the evaluation metric that is most relevant to the specific business problem at hand.

## Importance of Evaluation Metrics in Churn

Churn is a critical business problem for many companies because losing customers can have a significant impact on revenue and profitability. Therefore, it is essential to have accurate and reliable churn prediction models that can identify customers who are at risk of leaving.

Evaluation metrics such as F1-score, recall, and ROC-AUC score provide insight into the performance of churn prediction models. By analyzing these metrics, businesses can determine how well their models are performing and make informed decisions about how to improve them. For example, if the F1-score is low, it may indicate that the model is not accurately identifying customers who are likely to churn. This may prompt businesses to re-evaluate their feature selection, hyperparameter tuning, or even their data collection processes.

Moreover, evaluation metrics also help businesses make trade-offs between different prediction models. For example, in some cases, a company may prioritize recall over precision because they want to identify as many at-risk customers as possible, even if it means some false positives. On the other hand, in other cases, a company may prioritize precision over recall because they want to avoid incorrectly flagging customers as at-risk and taking unnecessary retention actions.

Therefore, evaluation metrics play a crucial role in helping businesses optimize their churn prediction models to minimize churn and retain valuable customers.

### Deep Neural Network a.k.a Multi-layer Perceptron

A deep neural network (DNN) model is a type of artificial neural network (ANN) that has multiple layers between the input and output layers. Each layer in a DNN consists of a set of neurons that are connected to the neurons in the previous layer. The neurons in each layer transform the inputs from the previous layer into a new set of outputs that are fed to the next layer.

DNNs are capable of learning complex patterns and representations in data, which makes them a powerful tool for solving a wide range of machine learning problems such as image recognition, natural language processing, speech recognition, and many others.

The layers in a DNN can be of different types, including fully connected layers, convolutional layers, recurrent layers, and others. The number of layers and the number of neurons in each layer can vary depending on the complexity of the problem and the size of the dataset.

DNNs are trained using a process called backpropagation, where the error between the predicted output and the true output is propagated backwards through the network to adjust the weights of the connections between the neurons. This process is repeated for multiple epochs until the network achieves a satisfactory level of accuracy on the training data.

## What is the purpose of using Keras library?

Keras is an open-source neural network library written in Python. It is designed to enable fast experimentation with deep neural networks, and has become one of the most popular high-level neural network APIs due to its simplicity, modularity, and ease of use.

Keras provides a user-friendly interface for building and training neural networks, and supports both convolutional networks for image recognition and processing, as well as recurrent networks for natural language processing and time-series analysis.

Keras allows for the construction of complex neural network architectures with minimal code, using building blocks such as layers, activations, and optimizers. It also provides a range of pre-trained models and utilities for data preprocessing, data augmentation, and model evaluation.

Keras is built on top of other machine learning libraries such as TensorFlow, Microsoft Cognitive Toolkit (CNTK), and Theano, and provides a simplified and streamlined interface for using these lower-level libraries.

Keras is widely used in industry and academia for developing state-of-the-art deep learning models, and has been adopted by many companies including Google, Microsoft, and IBM. It has also been integrated into other machine learning libraries and frameworks such as TensorFlow 2.0, making it a popular choice for building deep learning models.

## Categorical Accuracy

Categorical accuracy, also known as categorical accuracy score or simply accuracy, is a performance metric used to evaluate the performance of classification models. It measures the proportion of correctly classified samples out of the total number of samples in a dataset.

For example, suppose we have a dataset of 100 images, each of which belongs to one of five possible classes. A classification model is trained to predict the correct class label for each image. After evaluating the model on the test set, we find that it has correctly classified 80 of the images. The categorical accuracy of the model would then be 80/100 = 0.8, or 80%.