My Learnings.

Technical Terms Used in the project:

1. Logistic Regression
2. GradientBoostingClassifier
3. Hyper Parameter Tuning (Params Grid)
4. GridSerachCV
5. ROC\_AUC\_Score
6. Average\_Precision\_Score
7. Confusion\_Matrix
8. Accuracy\_Score
9. ROC\_Curve
10. Precision\_Recall\_Curve
11. Flask Integration

# **Logistic Regression**

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of target or dependent variable is dichotomous, which means there would be only two possible classes.

In simple words, the dependent variable is binary in nature having data coded as either 1 (stands for success/yes) or 0 (stands for failure/no).

Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

# Types of Logistic Regression

Generally, logistic regression means binary logistic regression having binary target variables, but there can be two more categories of target variables that can be predicted by it. Based on those number of categories, Logistic regression can be divided into following types −

1. **Binary or Binomial**

In such a kind of classification, a dependent variable will have only two possible types either 1 and 0. For example, these variables may represent success or failure, yes or no, win or loss etc.

1. **Multinomial**

In such a kind of classification, dependent variable can have 3 or more possible ***unordered*** types or the types having no quantitative significance. For example, these variables may represent “Type A” or “Type B” or “Type C”.

1. **Ordinal**

In such a kind of classification, dependent variable can have 3 or more possible ***ordered*** types or the types having a quantitative significance. For example, these variables may represent “poor” or “good”, “very good”, “Excellent” and each category can have the scores like 0,1,2,3.

# Logistic Regression Assumptions

Before diving into the implementation of logistic regression, we must be aware of the following assumptions about the same −

* In case of binary logistic regression, the target variables must be binary always and the desired outcome is represented by the factor level 1.
* There should not be any multi-collinearity in the model, which means the independent variables must be independent of each other.
* We must include meaningful variables in our model.
* We should choose a large sample size for logistic regression.

## Regression Models

* [Binary Logistic Regression Model](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_binary_logistic_regression_model.htm) − The simplest form of logistic regression is binary or binomial logistic regression in which the target or dependent variable can have only 2 possible types either 1 or 0.
* [Multinomial Logistic Regression Model](https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_multinomial_logistic_regression_model.htm) − Another useful form of logistic regression is multinomial logistic regression in which the target or dependent variable can have 3 or more possible ***unordered*** types i.e. the types having no quantitative significance.

# GradientBoostingClassifier

One of the most powerful ways of training models is to train multiple models and aggregate their predictions. This is the main concept of Ensemble Learning. While many flavours of Ensemble Learning exist, some of the most powerful algorithms and Boosting Algorithms.

Boosting refers to any Ensemble Method that can combine several weak learners(a predictor with poor accuracy) to make a strong learner(a predictor with high accuracy). The idea behind boosting is to train models sequentially, each trying to correct its predecessor.

# **An Overview Of Adaptive Boosting**

In Adaptive Boosting, the main idea occurs with the model assigning a certain weight to each instance, and training a weak learner. Based on the predictor’s performance, it gets assigned its own separate weight based on a weighted error rate. The higher the accuracy of the predictor, the higher its weight, and the more “say” it will have on the final prediction.

Once the predictor has made predictions, AdaBoost looks at the misclassified instances, and boosts their instance weights. After normalising the instance weights so that they all equate to 1, a new predictor is trained and the process is repeated until a desirable output is reached, or a threshold is reached.

The final classification is done by taking a weighted vote. In other words, if we were predicting heart disease on a patient, and 60 stumps predicted 1 and 40 predicted 0, but the predictors in the 0 class had a higher cumulative weight(i.e the predictors had more “say”), then the final prediction would be 0.

# **Gradient Boosting**

In contrast to Adaptive Boosting, instead of sequentially boosting misclassified instance weights, Gradient Boosting actually make predictions on the predecessors residuals.

1. The first thing Gradient Boosting does is that is starts of with a *Dummy Estimator.* Basically, it calculates the mean value of the target values and makes initial predictions. Using the predictions, it calculates the difference between the predicted value and the actual value. This is called the *residuals.*
2. Next, instead of training a new estimator on the data to predict the target, it trains an estimator to predict the *residuals of the first predictor.*This predictor is usually a Decision Tree with certain limits, such as the maximum amount of leaf nodes allowed. If multiple instances’ residuals are in the same leaf node, it takes their average and uses that as the leaf node’s value.
3. Next, to make predictions, for each instance, it adds the base estimator’s value onto the Decision Tree’s predicted residual value of the instance to make a new prediction. It then calculates the residuals again between the predicted and actual value.
4. This process is repeated until a certain threshold is reached or the residual difference is very small.
5. To make a prediction for an unseen instance, it gives the instance to each and very decision tree made, sums their predictions and adds the base estimator’s value.

# **Learning Rate**

An important hyperparameter to take note of here is the learning rate. This actually scales the contribution of each tree, so essentially increasing bias in exchange for a lower variance. So at step 3 and 4, the predicted value is actually multiplied by a learning rate to achieve better generalisation on unseen data. Now, this model has a lot of parameters, so it is worth mentioning the most important ones:

**learning\_rate**: exactly the same parameter as we have discussed about above; it scales the contribution of each tree

**init:**the initial estimator, which equates to the DummyEstimator by default(aka predicts the mean for everything)

**max\_depth:** the maximum depth you want your trees to grow

**n\_estimators:**the amount of trees you want to create

**criterion**: what loss function you would like to minimise for the decision trees to use when it is searching for the best feature and threshold that splits the data.

**loss**: the loss to use for calculating residuals(the default is “ls”, or least squares)

**max\_leaf\_nodes:**the maximum number of leaf nodes you want to have for each tree. If this number is smaller then the number of training instances, and if two or more instances are in the same leaf, then the leaf’s value will be the average of all the training instance values in that leaf.

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

GB 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 binomial or multinomial deviance loss function. Binary classification is a special case where only a single regression tree is induced.

[**GradientBoostingClassifier**](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html#sklearn.ensemble.GradientBoostingClassifier) supports both binary and multi-class classification. The following example shows how to fit a gradient boosting classifier with 100 decision stumps as weak learners:

The number of weak learners (i.e. regression trees) is controlled by the parameter n\_estimators; [The size of each tree](https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting-tree-size) can be controlled either by setting the tree depth via max\_depth or by setting the number of leaf nodes via max\_leaf\_nodes. The learning\_rate is a hyper-parameter in the range (0.0, 1.0] that controls overfitting via [shrinkage](https://scikit-learn.org/stable/modules/ensemble.html#gradient-boosting-shrinkage) .

# Hyper Parameter Tuning (Params Grid)

In true machine learning fashion, we'll ideally ask the machine to perform this exploration and select the optimal model architecture automatically. Parameters which define the model architecture are referred to as **hyperparameters** and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.

***Hyperparameters are not model parameters*** and they cannot be directly trained from the data. Model parameters are learned during training when we optimize a loss function using something like [gradient descent](https://www.jeremyjordan.me/gradient-descent/).

Whereas the model parameters specify how to transform the input data into the desired output, the hyperparameters define how our model is actually structured. Unfortunately, there's no way to calculate “which way should I update my hyperparameter to reduce the loss?” (ie. gradients) in order to find the optimal model architecture; thus, we generally resort to experimentation to figure out what works best.

# ROC\_AUC\_Score

AUC-ROC curve helps us visualize how well our machine learning classifier is performing.

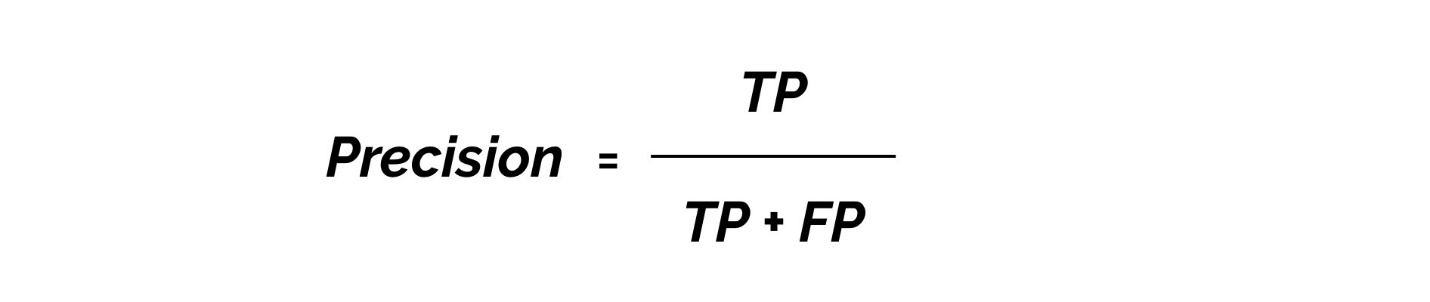
The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR**against **FPR**at various threshold values and essentially **separates the ‘signal’ from the ‘noise’**. The **Area Under the Curve (AUC)**is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

# Average\_Precision\_Score

**Precision**measures how accurate your predictions are. i.e. the percentage of your predictions are correct.

It measures how many of the predictions that your model made were actually correct.



*TP = True Positives (Predicted as positive as was correct)*

*FP = False Positives (Predicted as positive but was incorrect)*

Compute average precision (AP) from prediction scores

AP summarizes a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight.

The mean Average Precision or mAP score is calculated by taking the mean AP over all classes and/or overall IoU thresholds, depending on different detection challenges that exist.

# Confusion\_Matrix

There are plenty of ways to gauge the performance of your classification model but none have stood the test of time like the confusion matrix. It helps us evaluate how our model performed, where it went wrong and offers us guidance to correct our path.

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. The matrix compares the actual target values with those predicted by the machine learning model. This gives us a holistic view of how well our classification model is performing and what kinds of errors it is making.

For a binary classification problem, we would have a 2 x 2 matrix as shown below with 4 values:



* The target variable has two values: **Positive**or **Negative**
* The **columns**represent the **actual values** of the target variable
* The **rows**represent the **predicted values**of the target variable

**True Positive (TP)**

* The predicted value matches the actual value
* The actual value was positive and the model predicted a positive value

**True Negative (TN)**

* The predicted value matches the actual value
* The actual value was negative and the model predicted a negative value

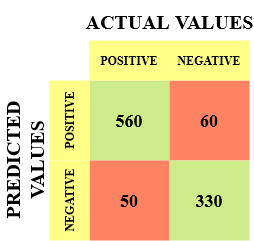
**False Positive (FP) – Type 1 error**

* The predicted value was falsely predicted
* The actual value was negative but the model predicted a positive value
* Also known as the **Type 1 error**

**False Negative (FN) – Type 2 error**

* The predicted value was falsely predicted
* The actual value was positive but the model predicted a negative value
* Also known as the **Type 2 error**

Let me give you an example to better understand this. Suppose we had a classification dataset with 1000 data points. We fit a classifier on it and get the below confusion matrix:

[](https://cdn.analyticsvidhya.com/wp-content/uploads/2020/04/Confusionmatrix-example.png)

The different values of the Confusion matrix would be as follows:

* True Positive (TP) = 560; meaning 560 positive class data points were correctly classified by the model
* True Negative (TN) = 330; meaning 330 negative class data points were correctly classified by the model
* False Positive (FP) = 60; meaning 60 negative class data points were incorrectly classified as belonging to the positive class by the model
* False Negative (FN) = 50; meaning 50 positive class data points were incorrectly classified as belonging to the negative class by the model

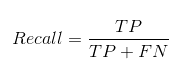
This turned out to be a pretty decent classifier for our dataset considering the relatively larger number of true positive and true negative values.

Equation_Accuracy

*Precision tells us how many of the correctly predicted cases actually turned out to be positive.*

Confusion Matrix Precision

*Recall tells us how many of the actual positive cases we were able to predict correctly with our model.*



Precision is important in music or video recommendation systems, e-commerce websites, etc. Wrong results could lead to customer churn and be harmful to the business.

Recall is important in medical cases where it doesn’t matter whether we raise a false alarm but the actual positive cases should not go undetected!

# Accuracy\_Score

Accuracy classification score.

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must *exactly* match the corresponding set of labels in y\_true.

# ROC\_Curve

ROC curves are VERY help with understanding the balance between true-positive rate and false positive rates. Sci-kit learn has built in functions for ROC curves and for analyzing them. The inputs to these functions (roc\_curve and roc\_auc\_score) are the actual labels and the predicted probabilities (not the predicted labels). Both roc\_curve and roc\_auc\_score are both complicated functions, so we will not have you write these functions from scratch. Instead, we will show you how to use sci-kit learn's functions and explain the key points.

# Precision\_Recall\_Curve

* ROC Curves summarize the trade-off between the true positive rate and false positive rate for a predictive model using different probability thresholds.
* Precision-Recall curves summarize the trade-off between the true positive rate and the positive predictive value for a predictive model using different probability thresholds.
* ROC curves are appropriate when the observations are balanced between each class, whereas precision-recall curves are appropriate for imbalanced datasets.

The precision-recall curve plot is then created showing the precision/recall for each threshold for a logistic regression model (orange) compared to a no skill model (blue).

Generally, the use of ROC curves and precision-recall curves are as follows:

* ROC curves should be used when there are roughly equal numbers of observations for each class.
* Precision-Recall curves should be used when there is a moderate to large class imbalance.

The reason for this recommendation is that ROC curves present an optimistic picture of the model on datasets with a class imbalance.

*However, ROC curves can present an overly optimistic view of an algorithm’s performance if there is a large skew in the class distribution. […] Precision-Recall (PR) curves, often used in Information Retrieval , have been cited as an alternative to ROC curves for tasks with a large skew in the class distribution.*