# **Kesavan – Analytics Vidhya - Jobathon**

The primary objectives of this jobathon are:

* To predict user will buy or not in next three months using baseline models and score the models.
* To preprocess the data by cleansing it and then transforming it dataset ready for applying machine model.
* Carrying out analysis of weights to balance the imbalance data and then split the data into train and test data.
* Choosing the right hyper tuning parameters for classification model.
* To implement baseline models and classification models on this oversampled data and score the models.
* Evaluate & interpreting models to see which model works the best for this dataset.

# Understanding the Data for Logistic Regression

Before applying the models, it necessary to find out the features which are impactful in predicting the target variable (buy). However, in order to find out the features, lets visualize the features pertaining to the feature buy.

# Exploratory Data Analysis before creating a Logistic Regression Model

Target variable is compare with other variables to check if any impact or not.

Heatmap is produced to check if any correlation is present between target variable and other variables.

Result shows – high correlated values are only Campaign 1 and 2.

# Data Engineering & pre-processing:

NA values in the purchase is replaced with Zero values.

Date variable and id variable is dropped.

when a dataset has more data points or observations belonging to one category and very few for another, we call it a class imbalance problem since the distribution of class labels is not balanced and skewed. Let’s see whether we have a class imbalance problem.

There are a variety of approaches to dealing with class imbalance, such as increasing minority class samples or decreasing majority class samples to ensure that both classes have the same distribution.

Because we’re using the Scikit-learn machine library to create the model, it has a logistic regression implementation that supports class weighting. We will use the inbuilt parameter “class\_weight” while creating an instance of the Logistic Regression model.

Both the majority and minority classes will be given separate weights. During the training phase, the weight differences will influence the classification of the classes.

The purpose of adding class weights is to penalize the minority class for misclassification by setting a higher class weight while decreasing the weight for the majority class.

# Build and Train Logistic Regression model in Python

To implement Logistic Regression, we will use the Scikit-learn library. We’ll start by building a base model with default parameters, then look at how to improve it with Hyperparameter Tuning.

As previously stated, we will use the “class\_weight” parameter to address the problem of class imbalance. Let’s start by creating our base model with the code below.

After training our model on the training dataset, we used our model to predict values for the test dataset and recorded them in the y\_pred\_basemodel variable.

Let’s look at which metrics to use and how to evaluate our base model.

# Model Evaluation Metrics

To evaluate performance or our model we will be using “f1 score” as this is a class imbalance problem using accuracy as a performance metrics is not good also, we can say that f1 score is the go-to metric when we have a class imbalance problem. The formula for calculating the F1 score is as follows:

F1 Score = 2\*(Recall \* Precision) / (Recall + Precision)

Precision is the ratio of accurately predicted positive observations to the total predicted positive observations.

Precision = TP/TP+FP

Recall is the ratio of accurately predicted positive observations to all observations in actual class – yes.

Recall = TP/TP+FN

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.

# Creating hyper tuned model

Model parameters (such as weight, bias, and so on) are learned from data, whereas hyperparameters specify how our model should be organized. The process of finding the optimum fit or ideal model architecture is known as hyperparameter tuning. Hyperparameters control the overfitting or underfitting of the model. Hyperparameter tuning can be done using algorithms like Grid Search or Random Search.

We will use Grid Search which is the most basic method of searching optimal values for hyperparameters. To tune hyperparameters, follow the steps below:

* Create a model instance of the Logistic Regression class
* Specify hyperparameters with all possible values
* Define performance evaluation metrics
* Apply cross-validation
* Train the model using the training dataset
* Determine the best values for the hyperparameters given.
* We can use the below code to implement hyperparameter tuning in python using the Grid Search method.

Model Evaluation

We will evaluate our model on Test Dataset. First, we will predict values on the Test dataset.

We chose “f1 score” as our performance metric above, but let’s look at the scores for all of the metrics, including confusion metrics, precision, recall, ROC-AUC score, and ultimately f1 score, for learning purposes.

Then, we’ll compare our final model’s f1 score to our base model to see if it’s improved.

We’ll use the code below to calculate the score for various metrics

# Conclusion

We can see that by tuning hyperparameters, we were able to improve the performance of our model since our F1 Score for the final model (0.71) is higher than that of the base model (0.57). After the hyperparameter tuning model got a 0.88 ROC-AUC score.

With this, we were able to construct our logistic regression model and test it on the Test dataset. More feature engineering, hyperparameter optimization, and cross-validation techniques can improve its performance even more.