Click to Cart: A Predictive Modelling Task

Introduction:

The task requires us to predict whether the user made a purchase or not using a session-level customer data. The dataset includes include features like ‘Time spent on site’, ’Pages viewed’ ,etc which have a pretty high correlation with the person making a purchase. We can use any supervised learning algorithm to tackle the problem. I have chosen Random Forest Classifier and XGBoost Classifier.

1) Exploratory Data analysis:

I have divided the features into categorical and numerical features. I have made histograms (No. of training instances vs Given feature) of each numerical feature. For the categorical features, I have used bar plots for better visualisation.

A correlation matrix helps us better interpret the correlation between the features and the target variable.

To analyse feature distributions and detect outliers, boxplots can be very helpful.

(**Description of a boxplot :** We can get the following five statistical measures using a boxplot:

- The minimum
- The first quartile(Q1)
- Median(Q2)
- The third quartile(Q3)
- The maximum

The interquartile range (ie from Q1 to Q3) is represented as a box with a line at the median, and whiskers that extend to the smallest and largest values within 1.5 * IQR from the quartiles)

Now, the data points that extend beyond these whiskers can be safely assumed as outliers in the dataset.

But here , we observe that the number of outliers found using the boxplot of Cart_value are relatively high compared to other features.

This is because of the skewness of the data. To address this, we apply a log transformation to the cart values to reduce skewness and make the distribution more symmetric. Thus, we thereby replace the feature 'Cart_value' with 'Cart_value_log'. After making the boxplot again, we can see that there are no outliers.

Since, the correlation between the feature 'Browser_Refresh_Rate' and the target variable is very low, we can say that the feature is irrelevant to our analysis. Thus, I have dropped that column from our dataset.

2) Data preprocessing:

Secondly, I have used the One hot encoder to encode the categorical columns. Then we use the StandardScaler to complete the preprocessing step.

3) Model Selection:

➢ Random forest Classifier:

Random forest classifier is an ensemble of Decision Trees trained on different subsets of features and training instances. It makes predictions by aggregating the outputs of these trees.

<u>Training:</u>

- Each decision tree draws a bootstrap sample from the training data (ie a random subset of features and training instances)

- At each node, split the data according to the best feature and the corresponding best threshold. We get the best split by minimising the impurity (gini impurity, entropy, etc)
- Gini impurity:

$$I_G = 1 - \sum_{j=1}^{c} p_j^2$$

p_j: proportion of the samples that belongs to class c for a particular node

<u>Assumptions behind the model:</u>

- Each decision tree has low bias but high variance. Due to the randomness in the features and data points in the trees, we are able to reduce the overall variance of the outputs.
- The trees are almost independent, ie not perfectly correlated

<u>Making Predictions:</u>

- The final prediction is made by taking majority vote across all the trees.

This model is suitable for this task because:

It captures non-linear and interaction effects and reduces overfitting naturally.We can extract feature importance scores to understand what drives purchase decisions, making the model interpretable.

➢ XGBoost Classifier:

While Random forest uses the bagging technique to ensemble, XGBoost uses Boosting. Bagging involves training multiple independent models (here Decision Trees) parallelly on different bootstrap samples of the data, and aggregate their predictions. This ensures that the final output has less variance.
Whereas, boosting mainly focuses on reducing the bias. In boosting, we train models sequentially , where each new model tries to correct the errors of the previous ones. Each instance is weighed based on previous mistakes.

<u>Training:</u>
XGBoost starts with an initial prediction, often 0.5 for binary classification.

- Similarity Scores: They measure the homogeneity of residuals within a node. Residuals are nothing but the difference between predicted probabilities and actual class labels.
- Gain Calculation : Gain is computed as the difference between the similarity scores of parent node and child nodes.
- Pruning : To prevent overfitting , XGBoost prunes branches where the gain is less than a specified threshold – Gamma.
- XGBoost incorporation logistic regression principles, transforming outputs into probabilities using the sigmoid function. This allows for probabilistic interpretation of predictions.

<u>Mathematical Details:</u>

The goal is to find the output value to minimise the objective function which is :

$$[\sum_{i=1}^{n} L(y_i, p_i^0 + O_{value})] + \frac{1}{2}\lambda O_{value}^2$$

Since the loss functions for classification and regression are different, the final similarity scores for classification and regression tasks are different.

Loss function for regression:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

Loss function for classification:

$$L = y_i \log(p_i) + (1 - y_i)\log(1 - p_i)$$

To simplify the optimisation problem, we first apply taylor’s expansion on the objective function (upto second degree derivatives).

To minimise the function, we equate the derivative of the function to zero.

On simplification, we get:

$$O_{value} = \frac{-(g_1 + g_2 + \cdots + g_n)}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

Where g_i’s and h_i’s are gradients(first derivative) and hessians (second derivative) of the loss function respectively.

For regression:

$$O_{value} = \frac{\text{Sum of Residuals}}{\text{Number of Residuals} + \lambda}$$

For classification:

$$O_{value} = \frac{\left(\sum \text{Residual}_i\right)}{\sum \left[\text{Previous Probability}_i \times (1 - \text{Previous Probability}_i)\right] + \lambda}$$

Similarity scores are calculated as follows:

$$-(g_1 + g_2 + \cdots + g_n)O_{value} - \frac{1}{2}(h_1 + h_2 + \cdots + h_n + \lambda)O_{value}^2$$

If this parabola is plotted with O_value on the X-axis, the maximum value of y-value is the similarity score.

On solving:

$$\text{Similarity Score} = \frac{1}{2}\frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

But in the XGBoost implementations, the half is omitted since similarity score is a relative measure. So,

$$\text{Similarity Score} = \frac{(g_1 + g_2 + \cdots + g_n)^2}{(h_1 + h_2 + \cdots + h_n + \lambda)}$$

Then, we calculate the gain of a split using the formula:

$$\text{Gain} = \text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$$

<u>Assumptions behind the model:</u>

XGBoost assumes that your rows are independently and identically distributed(IID) meaning no time dependence unless manually engineered.

Other than this, there are no major assumptions behind the xgboost model implying you can use the model on any dataset.

Making Predictions:

- XGBoost starts with a random prediction initially(mostly 0.5)
- Computing residuals – difference between actual and predicted values
- Train a decision tree to minimise the loss function(as described above)
- New prediction = Old Prediction + (learning rate * output value)
- Repeat for the number of boosting rounds(n_estimators)
- The final output value is equal to log(odds). To get the probability, pass the value through sigmoid(for binary) or softmax(for multi-class).

This model is suitable for this task because:

XGBoost learns complex patterns through boosting, this allows it to focus on hard to classify users, often leading to higher accuracy. XGBoost gives high control over tree depth, learning rate and regularisation parameters.

4) Comparative Analysis:

Both of the chosen models have performed almost equally on the data set giving approximately equal accuracy scores and f1 scores. But , RandomForestClassifier performed sightly better with higher accuracy.

However, RandomForest has outperformed XGBoost possibly due to the following reasons:

- XGBoost is more sensitive to noise and overfitting. Even we tried our best to reduce the noise by removing the outliers, there can still be some noise in the dataset.
- RandomForest reduces overfitting better due to bootstrap sampling and feature bagging.

- XGBoost usually shines when the target depends on complex feature interactions .But in our data, the features are mostly independent(evident from the correlation heat map). In this case, XGBoost’s iterative boosting does not help much.