Conversion Rate Prediction

Deepak Rao Nadipelly—Pareekshit Reddy Gaddam— Ziyue Wang

1 Abstract

The archetypical example of conversion rate in the field of e-commerce is the percentage of website visitors who buy something on the site. The goal of this project is to predict the conversion rate of the visitors on the e-commerce website, MercadoLibre, considering the varying factors like position on the page, available quantity, whether the product is fulfilled by the website, free shipping, sold quantity, etc. Estimated conversion rate can be used by an online retailer to redesign their business model based on the customer's needs. We initially tried to understand the relationship between the parameters using the Exploratory Data Analysis. We then used machine learning models like Logistic Regression, Support Vector Classifier, Random Forest, Light Gradient Boosting Machine, Extreme Gradient Boosting Algorithm, Neural network for this prediction. Since the cost of False positives is high in our problem, we have used Recall and AUC Score as the metrics to evaluate our model.

2 Statement of Contribution

- **Ziyue Wang:** Data Preprocessing, Feature extraction, Implemented LightGBM and Random Forest algorithms, worked on presentation and report.
- **Deepak Rao Nadipelly:** Data Preprocessing, Feature extraction, Implemented XGBoost and Support Vector Machine algorithms, worked on presentation and report.
- **Pareekshit Reddy Gaddam:** Data Preprocessing, Feature extraction, Implemented Neural Network and Logistic Regression algorithms, worked on presentation and report.

3 Introduction

The aim of this project is to predict the conversion rate for an online retailer based on various item features and other details regarding sales and logistics of the item. Conversions in e-commerce are defined as the number of times that a user completes a desired action on the platform. In this case, the desired action is to purchase a product. Estimating conversion rate is very critical for an online retailer as it impacts many aspects of their business. For example, the quality of the recommendations, search results and product advertisements they display to their users. This dataset was taken from a kaggle competition.

Source of the dataset: https://www.kaggle.com/c/conversion-rate-prediction/data

4 Feature Extraction and Preprocessing:

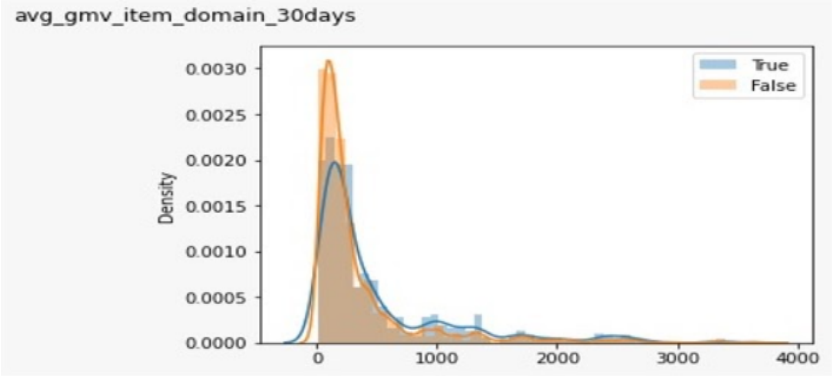
The dataset had over 180k records. It consisted of a combination of numerical,categorical and boolean features. The target variable (conversion) is highly imbalanced. It has 91% of the samples

belonging to the negative class and only 9% of the values belonging to the positive class. So, there is a need to handle this imbalance.

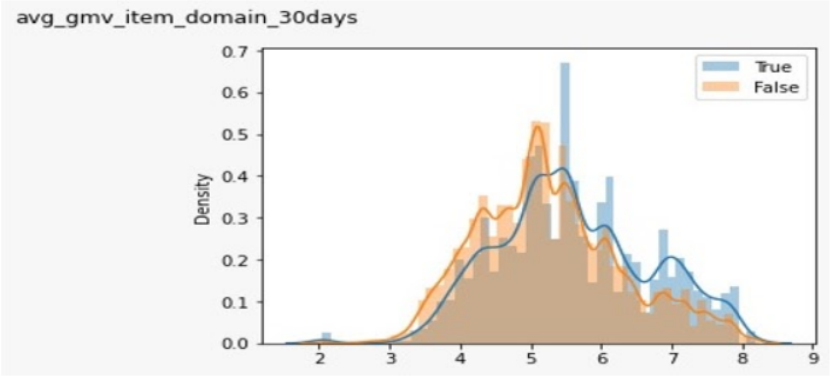
We are now going to look at various pre-processing steps taken to prepare the data for modeling and visualization purposes.

A few of the numeric columns in the dataset have a right-skewed distribution. So, we will to apply a log transformation to ensure that the values are normally distributed.

Before applying log



After applying log

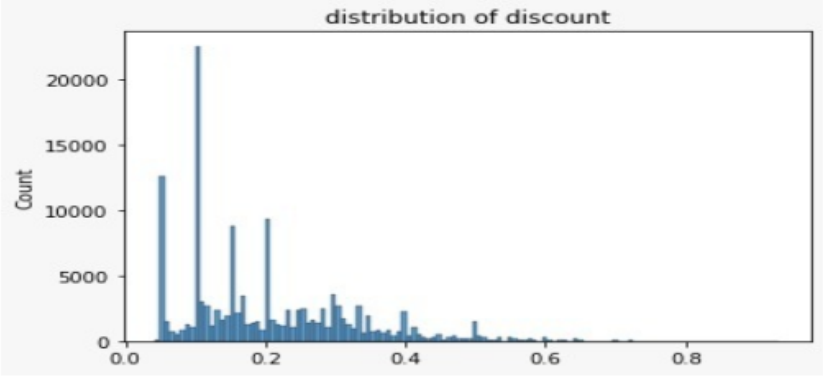


Also, some of the values in these numeric columns were null. We imputed these null values with mean.

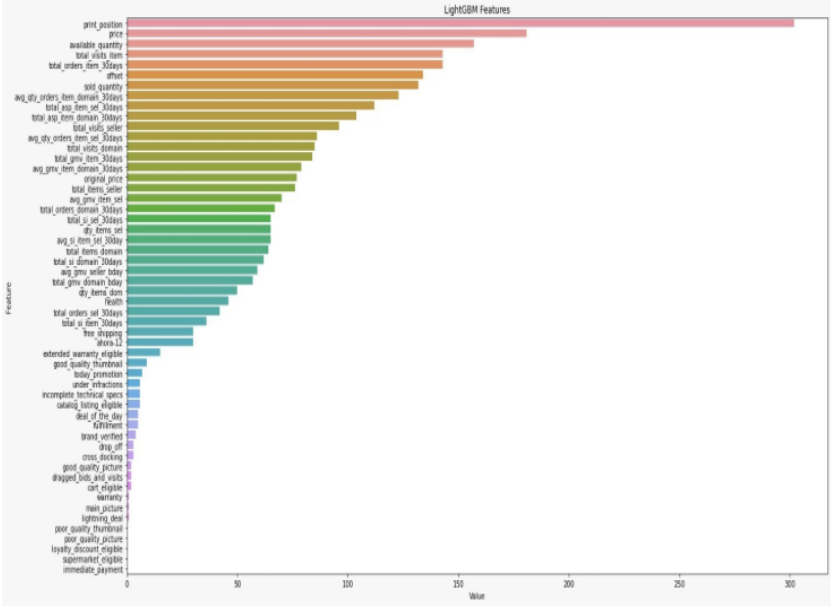
After we have dealt with the numeric columns, we converted the categorical columns to numeric columns using one hot encoding technique. We would like to cite the example of a tags column in which each entry was a list of strings of tags like deal of the day, extended warranty eligible etc. Some of these tags might be of interest to the customer. For example: If a customer is searching for a product on amazon and if the product had a 'Amazon's choice' tag it might give a good impression to the customer and may lead to the customer buying that product. So, we tried to extract these tags and created 18 new features for each individual tag using one hot encoding. We then converted warranty and main_picture columns to boolean. Warranty column had text in spanish. So we extracted information and set the value to true or false accordingly. For the main_picture column, we set the value to True if there was a URL to the product picture. If the URL wasn't present or if it was null, we set the value to False.

We created new features by combining values from the existing features. One such feature was discount which was computed from the original price and price fields. We have then fitted lasso to extract important features. We also plotted the feature importances graph using the LightGBM method. We also plotted correlation plots but we did not find a strong correlation between the predictors and the target variable. So we couldn't quite figure out much from the correlation plot.

Plotting the distribution of discount:

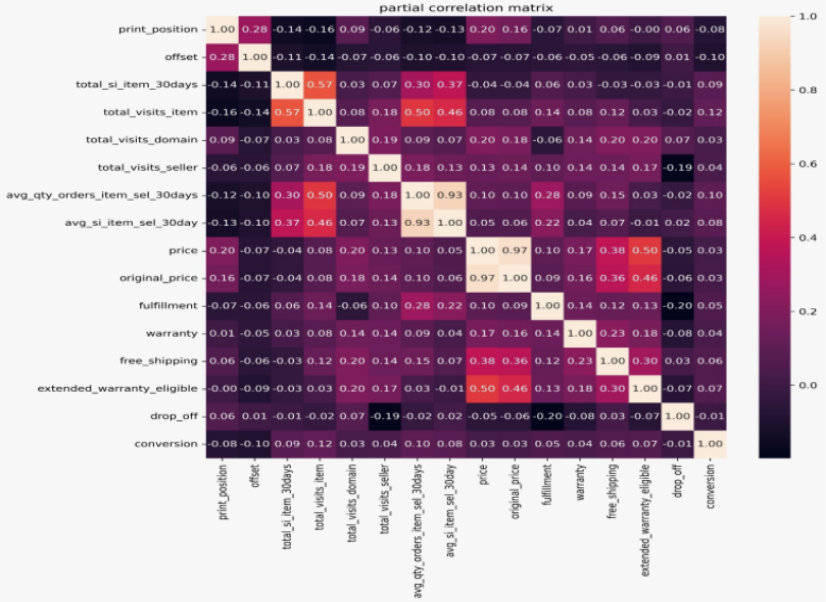


Feature Importances plot:



From the feature importances graph, we can see that print position(Position of the product on the page), price, available quantity, total orders of the item and total orders of the item over the last 30 days were the top five important features.

Correlation plot:



As we can see in the plot above, there is no strong correlation between any of the predictor and target variables. As we mentioned earlier the dataset was highly imbalanced. So, we had to apply sampling techniques to deal with the imbalance.

Some of the sampling techniques we applied are:

1. **Smote(Synthetic Minority oversampling technique):** Unlike random oversampling that only duplicate random samples from the minority class, SMOTE generates examples based on the distance(usually Euclidean distance) of each data and the minority class nearest neighbors, so the generated examples are different from the original minority class.These samples are also known as synthetic samples.

2. **Smote + Tomek Links:** Another way to handle imbalance would be to combine both oversampling and undersampling techniques. Tomek Links is an undersampling technique in which we find desired samples of data from the majority class that are having the lowest Euclidean distance with the minority class data. If the samples from the majority class are at a closer proximity to the samples from the majority class, it would be difficult to distinguish them. We identify such samples and then remove them.

5 Models

Since the task at hand was a binary classification, we have used the following algorithms.

1. Logistic Regression:

   In a binary logistic regression model, the dependent variable has two levels. The logistic regression model itself simply models probability of output in terms of input and does not perform classification, though it can be used to make a classifier, for instance by choosing a

cutoff value and classifying inputs with probability greater than the cutoff as one class and the below the cutoff as the other.

Pros:

(a) Logistic regression is easy to implement,interpret and very efficient to train and it is very fast in classifying unknown records.

(b) It can be easily extended to multiple classes(multinomial regression)

Cons:

(a) One major limitation with logistic regression is the assumption of linearity between the dependent and independent variables.

(b) Logistic regression is very sensitive to outliers. So the presence of data values that deviate from the expected range may lead to incorrect results.

2. Support Vector Machine:

   The goal of the support vector machine is to find a hyperplane in an N dimensional space that linearly separates the data points. To classify the points from the two classes, there can be many hyperplanes. The goal is to find the hyperplane that has the maximum margin(the maximum distance between the data points of both classes). Maximizing the margin helps us classify new data points more confidently. Support vectors play a crucial role in maximizing the margin. In addition, support vectors are the data points that are close to the hyperplane and influence the positioning of the hyperplane.

   Pros:

   (a) It uses a subset of the training points(called support vectors) in the decision function, so it is also memory efficient.

   (b) It works really well with a clear margin of separation.

   Cons:

   (a) The training time is very high.

   (b) It doesn't perform well when the data has noise.

3. Random Forest:

   Random forest belongs to the class of supervised learning algorithms. It builds a forest which is an ensemble of decision trees, usually trained with the bagging method. Bagging involves bootstrapping the data and learning multiple methods with each randomly sampled subset.

   When a random forest is used for a classification task and is presented with a new sample, the final prediction is made by taking most of the predictions coming from each individual decision tree.

   Pros:

   (a) Random Forests can handle both linear and non-linear relationships.

(b) Random Forests generally provide high accuracy and balance the bias-variance trade-off well.

Cons:

(a) Random Forest can be computationally intensive when the dataset is very large.

(b) Random Forests are not easily interpretable.

4. Light Gradient Boosting Machine:

   LightGBM is a gradient boosting framework based on decision trees to increase the efficiency of the model and also reduces memory usage. LightGBM adopts two novel techniques Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB). With GOSS, LightGBM can train each tree with only a small fraction of the entire dataset. With EFB, LightGBM handles high-dimensional sparse features in an efficient manner. It also supports distributed training with a very low communication cost and fast training on GPUs.

   Pros:

   (a) LightGBM models have a high training speed and high efficiency.

   (b) They support parallel and GPU learning.

   Cons:

   (a) Light GBM splits the trees leaf-wise which may lead to overfitting.

5. Extreme Gradient Boosting:

   XGBoost is a decision-tree based ensemble model that uses a gradient boosting framework. Gradient boosting attempts to accurately predict a target variable by combining the estimates of a set of simpler and weaker models.Some of the features of XGBoost are:

   (a) Parallelized tree building.

   (b) Tree pruning using a depth-first approach.

   (c) Regularization to avoid overfitting.

   Pros:

   (a) It is highly flexible and uses the power of parallel processing.

   (b) It handles missing values by default.

   Cons:

   (a) XGBoost does not perform well on sparse and unstructured data.

6. Neural Network:

   Neural networks learn by processing examples, each of which contains a known input and a result in turn forming probability-weighted associations between the input and the output. The training of a neural network from a given example is usually conducted by determining the difference between the predicted value and a true value. This difference is known as the error. The network then adjusts its weights according to a learning rate and using this error value. Successive adjustments will cause the neural network to predict values which are increasingly similar to the true value.

   Pros:

   (a) Neural networks work well against non-linear data with a large number of inputs.

   (b) Neural networks are quite robust to noise in training data.

   Cons:

   (a) It is computationally very expensive.

   (b) Neural networks depend a lot on training data and this may be lead to overfitting.

6 Experiments and Evaluation

Some of the techniques we used are:

(a) Stratified K-fold:
   The Stratified k fold cross-validation is an extension to cross-validation technique. As the dataset was imbalanced, we wanted to maintain the same ratio of samples in the train and validation splits.

(b) Model Stacking:
   Model Stacking is a way to improve model predictions by combining the outputs of multiple models and running them through another one. In our case, after we get the prediction of LGB model, we add the prediction array as an extra column of the input of XGB model. This increases the performance of our model.

(c) Bayesian Optimization:
   Bayesian Optimization is usually used to optimize functions that are expensive to evaluate. It builds a surrogate for the objective function and quantifies the uncertainty in that surrogate using a Bayesian machine learning technique, Gaussian process regression, and then uses an acquisition function defined from this surrogate to decide where to sample.

(d) Optimal Threshold:
   Since we are using model stacking as well as model ensembling which would output probability values instead of class predictions, we built a function to find the optimal threshold for deciding which class the sample belongs to so that it would reach the highest F1 score. For example, values greater than the optimal threshold will belong to one class and the values below the optimal threshold will belong to another class.

(e) Stepwise selection:

Stepwise selection is a method of fitting in which the choice of predictive variables is carried out by an automatic procedure. In each step, a variable is considered for addition to or subtraction from the model based on some specified criterion. Usually, this can be done in a forward or backward manner. In forward, we start with one predictor and keep on adding predictors one after the other until we get the best model. In backward, we start with all predictors and keep dropping features that are of less importance.

6.1 Evaluation Metrics Used

We have used AUC score and recall as our metrics to evaluate our model. We haven't used accuracy as the target class was imbalanced. We have chosen recall as a metric as it was important for us to reduce the number of False negatives. Recall is the value of True Positives over the sum of True Positives and False Negatives. False negatives in this scenario would be that the customer actually bought the product but we predicted that the customer hasn't bought the product.

7 Results

In this section, we will list out the hyper-parameters used in the models and also list the performance metrics of each of these algorithms.

(a) **Logistic Regression:**

Hyperparameters:

Penalty = 'L1',Solver = 'liblinear',Class_weight = 'balanced'

Metrics	Recall Score	ROC_AUC Score
Train	0.7937	0.7321
Test	0.7026	0.7128

(b) **Support Vector Machine:**

Hyperparameters:

Penalty='l1',dual=False,class_weight='balanced',C=1.0,max_iter=10000

Metrics	Recall Score	ROC_AUC Score
Train	0.8286	0.7318
Test	0.7141	0.7367

(c) **Random Forest:**

Hyperparameters:

n_estimators=200,max_depth=5,class_weight="balanced"

Metrics	Recall Score	ROC_AUC Score
Train	0.7952	0.8112
Test	0.7445	0.7427

(d) **LightGBM**

Hyperparamters:

'boost': 'gbdt','bagging_fraction': 0.48,'bagging_freq': 7,'feature_fraction': 0.11,'lambda_l1': 0.07,'lambda_l2': 0.03, 'learning_rate': 0.005, 'max_depth': 21, 'min_data_in_leaf': 66, 'num_leaves': 650

Metrics	Recall Score	ROC_AUC Score
Train	0.9234	0.8218
Test	0.8928	0.7754

(e) **XGBoost**

Hyperparameters:

objective="binary:logistic" , n_estimators=1000 scale_pos_weight=10 subsample = 0.8 colsample_bytree= 0.8 learning_rate= 0.09 max_depth= 5 early_stopping_rounds=40

Metrics	Recall Score	ROC_AUC Score
Train	0.8527	0.8668
Test	0.7245	0.8052

(f) **Neural Networks:**
Hyperparameters:
Optimizers: Adam, Decay_rate: 0.0000001 Activation functions: relu, sigmoid L2 regularization penalty: 0.001 Drop out: 0.2 Initializer: random_normal

Metrics	Recall Score	ROC_AUC Score
Train	0.8021	0.8067
Test	0.7168	0.7613

Our Working repository is on GitHub: https://github.com/JohnnyWang1998/Conversion-rate-prediction

8 Conclusion:

From the results, we can see that XGBoost performed the best among all the models. Logistic regression did not perform well relative to other models. Ensembling LightGBM and XGBoost gave us a higher AUC and recall score. Model stacking (LGB + XGBoost) gave the highest AUC and Recall score (AUC: 0.8199 Recall: 0.7832).

In the future we plan to implement CatBoost classifier. We would implement GridsearchCV for tuning the hyperparameters of the models.

9 References:

[1] https://arxiv.org/abs/1807.02811

[2] https://towardsdatascience.com/imbalanced-classification-in-python-smote-tomek-links-method-6e48dfe69bbc

[3] https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/

[4] https://scikit-learn.org/stable/