Applied Machine Learning (COMS 4995)
Final Project Report (Group 19)

I. Background

The advent of e-commerce is closely intertwined with the end of the era of many brick-and-mortar stores, thanks to its ability to offer exponentially more products and experiences to users without physical limitations. However, a critical issue plaguing this business is delay in deliveries which leads to negative user experience. Companies that do not act to mitigate delivery problems often experience a decline in sales. We intend to perform a delivery delay prediction for an **e-commerce dataset** that can tremendously help a business gain insight into problems that may cause this delay by looking at trends and potentially mitigate the causes. We begin by exploring our data thoroughly, followed by selecting and applying ML techniques to the problem of predicting sales volume based on a fairly high dimensional dataset that spans multiple years. Finally, we evaluate the performance of our models & select the best one.

II. Exploratory Data Analysis

EDA contains several parts:

1. INITIAL EDA: We get basic insights from the data distribution (categorical & numerical data), find correlations between features & target label, and an overall insight of target label in time scale.
2. DATA CLEANING
   Missing Value: We first drop *review_comment_title*, *review_comment_message, review_answer_timestamp, product_category_name_english* as we do not plan to handle text data in this project. Since there are very few missing values for all attributes (maximum being 3240 for *order_delivered_customer_date*), we'll drop these rows. Dropping such a miniscule number of rows will have negligible effect on the quality of the data or our predictions.
   Drop Columns: *customer_unique_id, order_id, customer_id, product_id, seller_id, review_id,* are ID values, unique for almost every row, hence, not correlated to our prediction, therefore, we drop them; *order_estimated_delivery_date, order_delivered_customer_date, review_creation_date, order_approved_at, order_delivered_carrier_date, shipping_limit_date* are also dropped while *order_purchase_timestamp* is kept to denote the date of purchase. The reason is that time before the order is delivered has no utility for our prediction.
3. DATA PREPROCESSING
   Target Generation: 'is_delayed' (0-1 value) is the target label we want to predict: 1 meaning the order is delayed. If *order_delivered_customer_date* is later than *order_estimated_delivery_date*, then we denote this order as delayed order with is_delayed=1, else is_delayed=0.
   Datetime Processing: *order_purchase_timestamp* is split into year, month, day, and daypart (based on hour), and is_holiday (weekend and black friday (2017-11-24)).
   Compute Distance: Using *geolocation_lat_x, geolocation_lat_y, geolocation_lng_x, geolocation_lng_y*, we can compute the distance between seller and customer using geopy.distance, then all columns mentioned before is dropped.
   Compute Volume: *product_length_cm, product_height_cm, product_width_cm* are multiplied to compute the product volume. Then all aforementioned columns are dropped.
4. ENCODING
   One-hot encoding: *payment_type* (only 4 types).
   Target encoding: *order_status, product_category_name, customer_city, customer_state, seller_city, seller_state, daypart.*

III. Model Building & Experiments

Since our dataset is highly imbalanced, we use some sampling methods to process our data. By investigating several sampling methods, our team found that the stratified and the downsampling method give the best performance on our data. Thus, in the following models, we experimented on both sampling methods before fitting the model to the data. For each model, we split the dataset into dev/test data in 80/20 fashion. We used 10-fold cross validation and picked the best model with the highest F-1 scores.

Baseline Model: Dummy Classifier

We use different strategies, including *most_frequent*, *prior*, *stratified* and *uniform*, to find the model with best performance as our baseline model. As stated above, we construct two models for the dummy classifier. The first one is the dummy classifier trained with stratified preprocessing, resulting in the best f-1 score to be 0.1279 by using the *uniform* strategy. For the second, we downsampled the training data before feeding it to the dummy classifier. It turns out that the model with *stratified* strategy has the higher f-1 score of 0.1297. The *stratified* strategy here has the same meaning as *uniform* strategy. The result indicates that randomly choosing a prediction will get higher f-1 scores performance than using the *most_frequency*/major label. The result also reflects the fact that our dataset is highly imbalanced.

Logistic Regression

Feeding unscaled data to our logistic regression (LR) model resulted in our model not converging. Further, because of the high imbalance in the labels of our dataset, we had to employ *class_weight = 'balanced'* in our LR model. When we scaled our data using StandardScaler and fed this data to our model, we were able to get an f-1 score of 0.364 with best hyperparameters as *C = 0.1, penalty = 'l2',* and *solver* = '*newton-cg*' (found using GridSearchCV). Area under the AUC curve is found to be 0.86. Downsampling our data did not lead to much difference in the f-1 score or the best hyperparameters - just that the penalty term changed to *'none'*. Area under the curve also remained the same.

Support Vector Machine (SVM)

To make the SVM models converge, we used the scaled/standardized data as input. Even though SVM can use different types of kernels to separate the data regardless of the linear separability of the data, the training time would be over 8 hours for each model, which isn't feasible. Thus, we use only linear SVM in our experiments. As recommended in the official documentation, we turn off the *dual* option as the number of data samples is larger than the number of features. By tuning the hyperparameters of loss functions, regularization parameters and penalty types, we achieved the highest f-1 score of 0.1066 on test data with C as 0.5, *squared_hinge* loss and L1 penalty. The corresponding scores of recall and precision are 0.0573 and 0.7519. As for the best model trained with downsampled data, the scores achieved on test data are 0.3676 on f-1 score, 0.739 on recall and 0.2447 on precision. Since the data is highly imbalanced and nonlinearly separable, the performance on the stratified linear SVM model is even poorer than the baseline model. However, when we perform the downsampled methods to the data before feeding it to the linear SVM model, the performance gets hugely improved and the corresponding f-1 score is much higher.

Random Forest

We have focussed on finding the best values of two important hyperparameters: 'n_estimators' and *'max_depth'* for building our Random Forest (RF) model. We initially began with giving values near the default value of *n_estimators* (100) for GridSearch. However, it soon became clear that our model was performing better on the maximum value of *n_estimators* till we finally found the optimal value to be 300 for *n_estimators* and 27 for *max_depth* after repeatedly performing GridSearch on a wide range of values. Our f-1 score on the model with best hyperparameters is 0.495. Accuracy for this model is very high ≈ 95%, meaning that our RF model clearly overfit. AUC estimates to be 0.9. Using undersampling, our f-1 score worsens to 0.42, changing the best hyperparameters to 250 for *n_estimators* and 23 for *max_depth*.

XGBoost

In the XGBoost (XGB) training process, we tuned several hyperparameters including *learning_rate*, *n_estimators* and *depth* of the model. For the models trained with stratified sampling data, we achieved the best model with recall 0.4349, precision 0.7904 and the f-1 score 0.5611 on test data. The best model has a *learning_rate* of 0.15, *max_depth* of 10 and *n_estimator* = 300. The best model trained with downsampling preprocess has *learning_rate* of 0.1, *max_depth* of 10 and 300 *n_estimators.* Feeding the test data into the best model, we get the recall score of 0.82, precision score of 0.2843 and f-1 score of 0.42. By comparing the two models, the model trained with stratified sampling has better performance. And yet, the XGB model with stratified sampling method also achieved the highest f-1 score among all models.

LightGBM

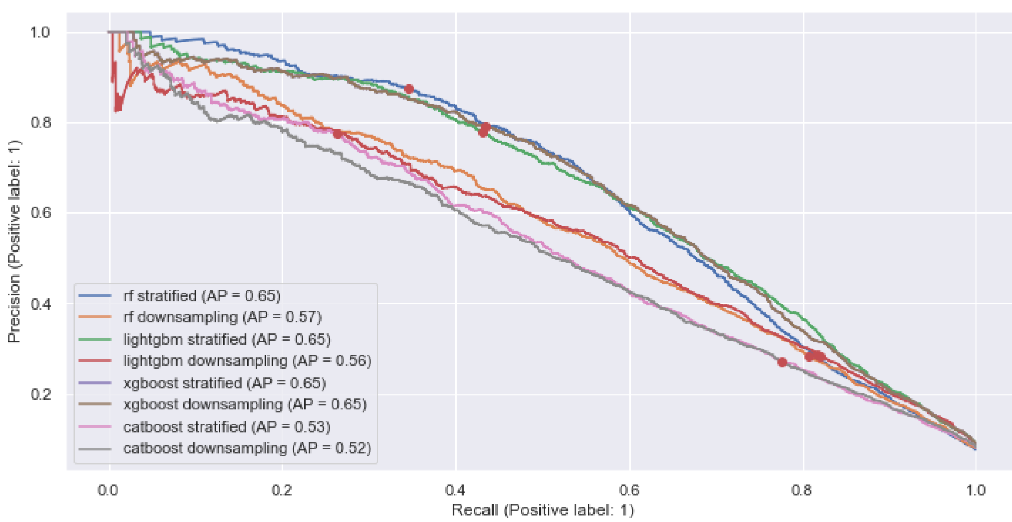
By training LightGBM (another type of boosted tree-based model), we aimed to achieve a relevantly "good" performance with less training time. We looked for the best hyperparameters of *num_leaves, max_depth* and *n_estimators.* For the stratified preprocessed model, the best performance has recall=0.43, precision=0.78 and f-1 score=0.55 with *max_depth* of 20, *n_estimators* of 300 and *num_leaves* of 127. Meanwhile, the downsampled preprocess model with the same *max_depth* and *num_leaves* with 200 *n_estimators* achieved the recall of 0.82, precision of 0.29 and f-1 score of 0.42.
LightGBM and XGB models achieved similar results with the downsampled sampling method. But for the models trained with stratified sampling, XGB performed much better than LightGBM. However, since the training time of LightGBM is about half that of XGB, we prefer LightGBM over XGB to save significant training time.

CatBoost

In order to train the CatBoost (CB) model, we use the encoded data and not scaled since it is a tree based model. The dataset is first split using a stratified split in order to ensure the minority class is properly represented across development & test sets. Then *learning_rate*, *n_estimators*, and *max_depth* hyperparameters are tuned using GridSearch. We found optimal parameters of *learning_rate*=0.05, *max_depth*=6, and *n_estimators*=100. This tuned model yielded an f1 score = 0.393, precision score=0.777, recall score=0.263, and accuracy score=0.939. We trained another CB model on downsampled data. The optimal hyperparameters found via GridSearch were *learning_rate*=0.05, *max_depth*=6, and *n_estimators*=100. The tuned model yielded an f1 score = 0.400, precision score = 0.270, recall score = 0.263, and accuracy score = 0.825.

IV. Choice of Model

The best model is chosen based on the PR curve. Given *recall=0.8*, Lightgbm with stratified sampling achieves better results. The reason why we adopt this strategy is that recall is more important in our prediction as FN (the order is delayed but we predict on-time) is crucial while FP (the order is on-time but we predict delayed) is more acceptable compared with FN.



V. Conclusion & Future Scope

This project helped us apply the skills & tools we learned during class. Most important takeaways are:

1. We employed different data analysis techniques to assess the quality of our data & then worked on improving its quality.
2. We brainstormed to understand which metric suited our use case the best. Despite getting tremendously high accuracy, we realized it's not the best measure to assess our model.
3. Since our data is highly imbalanced, we used stratified splitting and undersampling on data & built ML models using the best set of hyperparameters found for each model by GridSearch & evaluated the performance of the models.

In the future, we can extend the scope of this project by training neural networks on our dataset. In addition, we can attempt to get more samples belonging to the class with positive label, if possible.