Technical Report

1. Data Loading:

In the first step, we load the data from data/machine_learning_challenge_order_data.csv.gz by using the get_csv_gz function in code_/data_code/data_utils.py. The README.md file contains all the data about this dataset.

2. Data Grouping and Feature Engineering:

We grouped the data by the customer_id column in the second step, and we created features based on the grouped data.

The list of created features are as follows:

- customer_id: unique customer ID.
- number_return: the number of returns of each customer that happened within a six months period.
- avg_order_hour, sin_order_hour, cos_order_hour: the average order hour and its trigonometric functions for each customer to capture the periodic nature of this feature.
- number_order: the number of orders for each customer.
- failure_percentage: the percentage of failure for each order of each customer.
- min_delivery_fee, max_delivery_fee, total_delivery_fee, avg_delivery_fee: the statistics of the delivery fee for each customer.
- min_amount_paid, max_amount_paid, total_amount_paid, avg_amount_paid: the statistics of the amount paid by each customer.
- min_voucher_amount, max_voucher_amount, total_voucher_amount, avg_voucher_amount: the statistics of voucher amount for each customer.
- most_frequent_restaurant, restaurant_frequency, number_unique_restaurant: the most frequent restaurant, the number of ordering from this restaurant, and the number of unique restaurants each customer ordered.
- most_frequent_city, city_frequency, number_unique_city: the most frequent city, the number of ordering from this city, and the number of unique cities each customer ordered.
- most_frequent_payment, payment_frequency, number_unique_payment: the most frequent payment method, the number of ordering from this payment method, and the number of unique payment methods each customer ordered.
- most_frequent_transmission, transmission_frequency, number_unique_transmission: the most frequent transmission method, the number of ordering from this transmission method, and the number of unique transmission methods each customer ordered.

- most_frequent_platform, platform_frequency, number_unique_platform: the most frequent platform, the number of ordering from this platform, and the number of unique platforms each customer ordered.
- amount_payment_1491,amount_payment_1523,amount_payment_1619,amount_payment_1779, amount_payment_1811: the amount paid by each payment_id and each customer which are featured by 'amount_payment_{payment_id}'.
- amount_transmission_212,amount_transmission_1988,amount_transmission_2020, amount_transmission_4196,amount_transmission_4228,amount_transmission_4260, amount_transmission_4324,amount_transmission_4356,amount_transmission_4996, amount_transmission_21124: the amount paid by each transmission_id and each customer which are featured by 'amount_payment_{trnasmission_id}'.
- amount_platform_525,amount_platform_22167,amount_platform_22263,
- amount_platform_22295,amount_platform_29463,amount_platform_29495, amount_platform_2975: the amount paid by each platform_id and each customer which are featured by 'amount_payment_{platform_id}'.

This procedure produces 63 unique features for 245455 unique customers. The resulting file is saved in data/feature_engineered_data.csv to be exploited in the final steps.

3. EDA:

The goal in this step is to examine some exploratory data analysis to get some insights about created features and to remove nonessential, unnecessary, or irrelevant features. There are many variables to be used in the functions in this section, and they are all contained in config_files/eda_config.json.

- **Missing Values:**
  Asa the first step of our EDA, we develop the method remove_missing_value in the file code_/data_code/data_analysis.py to get comprehensive information about the missing value and its percentage for each column in our dataframe and saved it in the data/missing_value_df.csv. Furthermore, we plan to remove all columns with more than the specific threshold of percent missing values.

- **Correlation Analysis**:
  In this step, we get information regarding the correlation of numerical features of a dataframe and remove the numerical features with high correlations by implementing two methods corr_plot and remove_correlated_features in the file code_/data_code/data_analysis.py.

  The correlation plot of the numerical features before and after feature removals are available in the folder figures. The correlation matrix before and after feature removals are available in the folder data. The dataframe with the information about correlated features is saved in the data/correlated_features.csv.

In this step, three features avg_delivery_fee, avg_amount_paid, avg_voucher_amount are removed.

- **Remove Almost Zero Numerical Features:**
  In this step, we get information regarding the numerical features of a dataframe which have a number of zeros more than a specific threshold and remove them by implementing the method remove_almost_zero_numerical_features in the file code_/data_code/data_analysis.py.

  In this step, all the numerical features **except** the following features are removed from the features.
  avg_order_hour, sin_order_hour, cos_order_hour,
  min_delivery_fee, max_delivery_fee, total_delivery_fee,
  min_amount_paid, max_amount_paid, total_amount_paid,
  amount_payment_1619, amount_payment_1779,
  amount_transmission_4228, amount_transmission_4324, amount_transmission_4356,
  amount_platform_29463, amount_platform_29815, amount_platform_30231,
  amount_platform_30359

- **Handling Categorical Variables with High Cardinal Categories:**
  In this step, we get insight regarding the number of categories each categorical columns have by implementing the method remove_highly_variable_categorical_features in the file code_/data_code/data_analysis.py. To do so, our solutions involve reducing the number of categories they have or removing them all when they have more than a specific number.

  The information about the initial and final number of categories, which is generated after running the method, are both available in data/initial_num_categories_features.csv and data/final_num_categories_features.csv.

There is an implementation of each of these techniques in the file code_/data_code/data_analysis.py in the class EDA. It is important to keep in mind that the thresholds can be changed to change the number of features removed by each technique. The following features will be obtained when all the above-mentioned techniques are applied:

Main Feature:
customer_id

Categorical Features:
number_return,
number_order,
restaurant_frequency, number_unique_restaurant,
most_frequent_city, city_frequency, number_unique_city,
most_frequent_payment, payment_frequency, number_unique_payment,
most_frequent_transmission, transmission_frequency, number_unique_transmission,
most_frequent_platform, platform_frequency, number_unique_platform

Numerical Features:
avg_order_hour, sin_order_hour, cos_order_hour,
min_delivery_fee, max_delivery_fee, total_delivery_fee,
min_amount_paid, max_amount_paid, total_amount_paid,
amount_payment_1619, amount_payment_1779,
amount_transmission_4228, amount_transmission_4324, amount_transmission_4356,
amount_platform_29463, amount_platform_29815, amount_platform_30231,
amount_platform_30359

The final dataframe which will be used in our training procedure has been saved in data/eda_data.csv.

4. Training:

This section describes the training procedure for our models. Choosing the models, we are primarily looking for those that can handle categorical variables without any extra effort, such as One-Hot Encoding or Label Encoding. Subsequently, five different models are used in this project, two of which are machine learning and three of them are deep learning models.

4.1. ML Models:

1. **LightGBM**: LightGBM can handle categorical features by taking the input of feature names. It offers good accuracy with integer-encoded categorical features. It does not convert to one-hot encoding and it is much faster than one-hot encoding. LightGBM applies Fisher (1958) to find the optimal split over categories as described here. This technique often performs better than one-hot encoding.

2. **CatBoost**: CatBoost has the flexibility of giving indices of categorical columns so that it can be encoded as one-hot encoding using one_hot_max_size (OHMS, Use one-hot encoding for all features with the number of different values less than or equal to the given parameter value). CatBoost cal also solves the exponential growth of the combination of the feature by using the greedy method at each new split of the current tree.

   If If you don't pass anything in the cat_features argument, CatBoost will consider all columns as numerical variables. In addition, although CatBoost took a long time compared to LightGBM, it proved to be more accurate.

In addition to this, these ML models can also determine the importance of each feature, which will be reported for the final models. The models are implemented in the file run_ml_models.py.

4.2. DL Models:

Three state-of-the-art deep learning models are used here to handle categorical variables naturally. The models are implemented in the file run_dl_models.py.

1. **TabNet:** TabNet selects just the features which are relevant to the example at hand using a kind of soft feature selection. TabNet accomplishes this by using a sequential multi-step decision process. In other words, information is processed from top to bottom in several steps.

   The building blocks for performing this sequential attention are called transformer blocks even though they are a bit different from the transformers used in popular NLP models such as BERT. Those transformers use self-attention and seek to model the dependencies between the different words in a sentence. The type of transformer used here is trying to eliminate, step by

step, those features that are not relevant for the example at hand using a "soft" feature selection, which is accomplished by using the sparsemax function.
A comprehensive explanation of TabNet and its architecture can be found here.

2. **NODE**: What makes CatBoost special is its use of Oblivious Trees. Unlike other tree-based methods that consider every leaf at every depth, Oblivious trees ensure a feature is reused across all trees of a certain depth.

   In Neural Oblivious Decision Ensemble (NODE) authors borrow attention breakthroughs from EntMax, to make soft oblivious decision trees more closely resemble the traditional greedy learners. By imagining trees as a set of attention mechanisms and allowing differentiable compute, these models steal performance from embeddings, shared feature representation, and advances in pre-training, to achieve promising results on tabular data problems.

   One major advantage of NODE is that leaves effectively serve as embedding layers, allowing leaves to easily learn multiclass classification problems. To ensemble models, authors recommend using summing across trees at a particular layer and using that vector as input to later tree ensembles. This is an interesting proposition of this approach which sets it apart from the traditional ensembles.

   A comprehensive explanation of NODE and its architecture can be found here.

   3. **CategoryEmbedding**: This is the most basic deep learning model used in this project. In this architecture, the categorical features are passed through a learnable layer embedded in a feed-forward network.

The implementation of all these three DL models is based on the pytorch-tabular package, and all information about the parameters of these models is available here. Despite the fact that TabNet and NODE can also provide feature importance, there is no approach implemented in the package for evaluating it.

4.3. Hyperparameter Tuning:

To We use the optuna python package, which is one of the best hyperparameter tuning packages, to determine the best parameters for each model. This procedure optimizes the "Area Under the Receiver Operating Characteristic Curve (ROC AUC)" from prediction scores, which are evaluated by the scikit-learn python package.

As running DL models is very time-consuming, I have just tuned hyperparameters on ML models, which are LightGBM and Catboost. The required methods for hyperparameter tuning of ML models are all implemented in the file model_hp.py.

The information about hyperparameter tuning and the best parameters of each ML model are all saved in the folder hp_files.

4.4. Training:

As the final step, we train the five available models on the final data created by joining the dataframe available in data/eda_data.csv resulting from the EDA section.

We have utilized 5-fold cross-validation techniques and reported all the precision, recall, accuracy score of each fold by the method classification_report of scikit-learn package. In addition, we evaluate the ROC-AUC score of each fold using the method roc_auc_score of this package. The reported CSV files for each fold are all available in the folder model_files.

Due to the time-consuming nature of training DL models, we have modified their default parameters; however, ML models are trained with the parameters resulting from hyperparameter tuning.

5. Results:

Based on the files generated by the training of each model, which are all available in the folder model_files, the following insight can be derived.

1. In general, **ML models outperform DL models** because the latter do not take advantage of their best hyperparameters for the dataset.

2. CatBoost and LightGBM provide almost the same AUC, precision, recall, and F1 performance; however, they provide different scores for feature importance. NODE is more accurate in its AUC score than other DL models, but each model has different Precision, Recall, and F1 scores for each fold.

5. What should do Next:

1. It would be worth working on creating more informative features which can significantly enhance the performance of our models.

2. playing with thresholds applied in the EDA section and checking the effect of those changes might be insightful to select the optimal thresholds.

3. Since I do not have GPUs, hyperparameter tuning of DL models was infeasible since they are very time-consuming. It would be very beneficial if we could get the best hyperparameters of each DL model and also train all the resulting models on more epochs.