

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

SINGAPORE

CZ4041 Machine Learning

Project: Elo Merchant Category Recommendation

Group 2 Members

Name	Matriculation Number
Darryl See Wei Shen	U2020488B
Dinh Phuc Hung	U1921644A
Goh Peng Aik	U2022363E
Lau Zhen Jie	U1920833C
Oong Jie Xiang	U1920153J

Table of Contents

1 Introduction.....	4
1.1 Background.....	4
1.2 Problem Statement.....	4
1.3 Challenges	4
1.3.1 Anonymity of Features	5
1.3.2 Inconsistency due to Stochasticity of Splitting the Dataset.....	5
1.3.3 Rare Data Points	6
1.3.4 Huge Raw Feature List	7
2 Methodology for Pre-Machine Learning Phase.....	8
2.1 Preprocessing	8
2.1.1 Imputation.....	8
2.1.2 Data Correlation	9
2.2 Feature Engineering.....	10
2.2.1 One-Hot Encoding.....	10
2.2.2 Feature Generation	10
2.2.3 Feature Selection	11
3 Methodology for Machine Learning Phase.....	12
3.1 Model Architecture	12
3.1.1 Binary Classifier	14
3.1.2 Sampling	15
3.1.3 Upweighting	16
3.2 Model Evaluation	16
3.3 Model Selection.....	16
3.4 Hyperparameter Tuning.....	17
3.4.1 Random Search and Grid Search.....	17
3.4.2 Bayesian Optimisation.....	17
4 Results.....	18
4.1 Leaderboard	18

5 Project Management	19
5.1 Timeline.....	19
5.2 Contributions.....	19
6 Conclusion	20
7 References.....	20

1 Introduction

1.1 Background

Identifying and targeting the correct audience is of paramount importance in any business. This report discusses the approach to help Elo, one of the largest payment brands in Brazil, identify and serve the most relevant opportunities to individuals by uncovering signals in customer loyalty. There are a total of five datasets provided by Kaggle, namely, `train`, `test`, `historical_transactions`, `new_merchant_transactions`, and `merchants`.

In this project, Python's Scikit-Learn library (called `sklearn` hereafter) is used to perform machine learning tasks, as this library comprises many machine learning algorithms and is regularly maintained.

1.2 Problem Statement

Elo is one of the largest payment brands in Brazil. They can provide restaurant recommendations and discounts based on customers' personal preferences and their credit card providers. They have built partnerships with various merchants to offer discounts and promotions for their customers.

However, the effectiveness of these promotions remains uncertain as each customer has his own preference and each merchant may offer different products. To provide the best experience for the customers and to encourage repeat business for merchants, personalization needs to be done based on customers' preferences.

In this project, various machine learning algorithms are explored to predict the loyalty score of the customers based on their transactions and some anonymized features that are obtained from the [Elo Merchant Category Recommendation competition in Kaggle](#).

The submission score is based on the root mean squared error (RMSE). RMSE is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2},$$

where \hat{y} is the predicted loyalty score for each card identifier and y is the actual loyalty score assigned to each card identifier.

1.3 Challenges

In this project, 4 challenges are encountered. This section details each challenge.

1.3.1 Anonymity of Features

In all the five datasets provided, each of them has some anonymized attributes, which makes the relationship between these attributes and the target value hard to be determined. The only way to identify their correlation with the target value is by running various machine learning algorithms to predict. Table 1 shows the anonymized attributes in each dataset.

Table 1 Anonymized Attribute(s) in Each Dataset

Dataset	Anonymized Attribute(s)
train	feature_1, feature_3, feature_3
test	feature_1, feature_3, feature_3
historical_transactions	category_3, category_1, merchant_category_id, subsector_id, city_id, state_id, category_2
new_merchant_transactions	category_3, category_1, merchant_category_id, subsector_id, city_id, state_id, category_2
merchants	merchant_group_id, merchant_category_id, subsector_id, numerical_1, numerical_2, category_1, category_4, city_id, state_id, category_2

1.3.2 Inconsistency due to Stochasticity of Splitting the Dataset

The stochastic nature of sklearn train_test_split() module also poses a challenge in selecting the model. Figure 1 shows the distribution of scores (RMSE) over 100 iterations of training and testing using the K-Fold Cross-Validation technique with a model used in this project. The spread of the data suggests that the training of the chosen model is significantly impacted by the splitting of the train and test dataset, since some training and testing splits yield much better results than the others.

There are two main ways to improve the reproducibility of the results, namely, (1) setting a random seed or (2) improving the generalization of the models. The former is less ideal in a real-world application. Therefore, the approach taken for the latter is to explore ways to improve the model by reducing overfitting. This serves as a motivating factor to perform model selection and hyperparameter tuning further downstream in the machine learning pipeline.

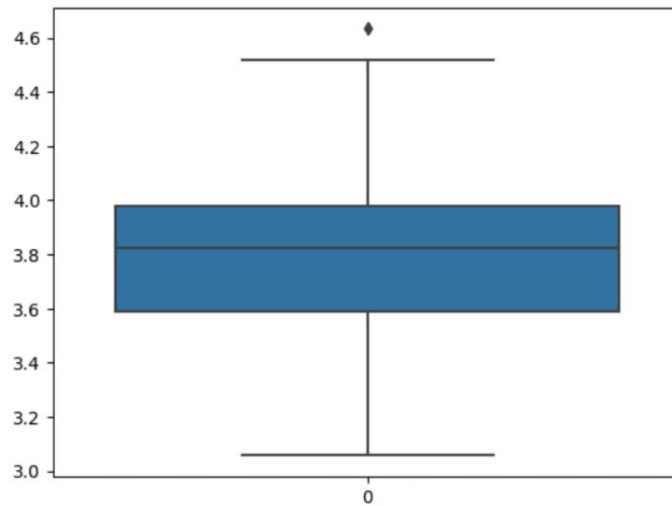


Figure 1: Distribution of RMSE Over 100 Iterations of Training using K-Fold Cross-Validation

1.3.3 Rare Data Points

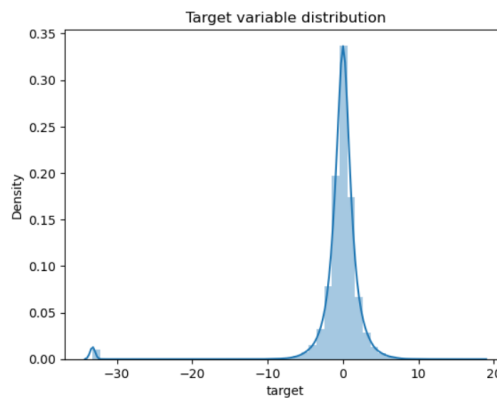


Figure 2: Distribution of Target Values in Training Dataset

Figure 2 shows the distribution of the target value in the training dataset. Most of the target values are between -1 to 1, which means that the target values may be normalized. Some values are less than -30 which may potentially be outliers.

This poses an issue for the regression model. As the mean of the target value is 0, the regression model will incur a huge error cost while predicting the target values of the rare points. Additionally, through exploratory analysis, there is no clear distinguishing feature that separates the rare points from the others.

1.3.4 Huge Raw Feature List

In the Train dataset, there are several rows that have the same value for each feature in the dataset but different target values. For example, as illustrated in Figure 3, `card_id` C_ID_4082ad893d and C_ID_028d432536 both have the same value for `first_active_month`, `feature_1`, `feature_2` and `feature_3` but have a huge difference in the target value. This suggests that more features are needed from other datasets to predict the target value.

1	first_active_month	card_id	feature_1	feature_2	feature_3	target
90670	2013-12	C_ID_4082ad893d	5	1	1	-33.2193
132131	2013-12	C_ID_028d432536	5	1	1	2.033015

Figure 3: Difference in Target Value of Two card_ids Even Though They Have the Same Value for All Attributes

Of all the five datasets, `train` and `test` datasets have the same number of features except for the missing target value in the test dataset, which is to be predicted. There are a total of 4 features that can be used in this train and test dataset to predict the target value. The `historical_transactions` and the `new_merchant_transactions` both also have the same feature list, each comprising 13 features (excluding `card_id`) that can be used to predict the target value. On the other hand, the merchant dataset has a total of 22 features that can be selected to predict the target value.

With this huge number of features to select from the other datasets, it is challenging to choose the feature(s) that is highly correlated to the target value. Some of the features may be noise to reduce the accuracy of the model's prediction.

2 Methodology for Pre-Machine Learning Phase

Many explorations and processing take place before machine learning models are created. This section details the preprocessing and feature engineering phases.

2.1 Preprocessing

2.1.1 Imputation

In each dataset, there are some columns that have null value(s) (except the train dataset). The table below shows the column(s) that contain at least one null value for test, historical_transactions, and new_merchant_transactions.

Table 2 Columns with Null Value

Dataset	Column(s) with at least one null value
test	first_active_month
historical_transactions	category_3, merchant_id, category_2
new_merchant_transactions	category_3, merchant_id, category_2

A few ways to handle these null values include dropping these rows or imputing them with a reasonable value such as the mean, median, or mode of the column. Dropping rows risk losing data which might impact the model's prediction. Hence, whenever possible, the null values are imputed with a certain reasonable value instead of being dropped.

In the test dataset, there is only a single row with a null value on the first_active_month column. This row should not be dropped as this row consists of one of the card_ids whose target value should be predicted. Hence, this null value is imputed with the mode value of first_active_month column.

For the historical_transactions and new_merchant_transactions datasets, merchant_id is not imputed because this key field is unpredictable. As for the other two fields, they are categorical data. Hence, the best reasonable value to impute for these two columns is by mode value. The data is first imputed with the mode of the aggregated value based on the card_id. If a specific card_id does not have any value for all the rows, then the data will be imputed with the mode value of that column.

2.1.2 Data Correlation

The train and transactions datasets are then merged, and a correlation matrix is plotted to test for the correlation of variables believed to be numerical in relation to their correlation with the target. From the correlation matrix in Figure 4, it can be seen that all the features have very low or no correlation with the target, indicating that some of these features may be categorical data instead.

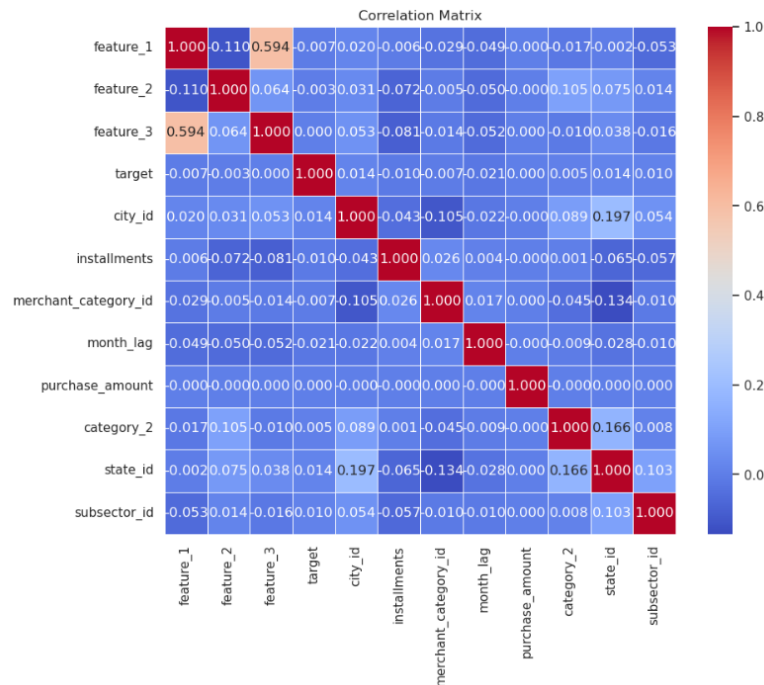


Figure 4 Correlation Matrix between the Features

The feature_1, feature_2 and feature_3 attributes were additionally tested with the target using Cramér's V correlation, as illustrated in Figure 5.

```
Cramér's V for feature_1: 0.0
Cramér's V for feature_2: 0.021144306534647047
Cramér's V for feature_3: 0.039342001566294194
```

Figure 5: Cramér's V Correlation for Features in train.csv

The results being very close to 0 show that there is little to no association between the anonymised features and the target.

As such, generation of new features has been considered necessary to better predict the target.

2.2 Feature Engineering

2.2.1 One-Hot Encoding

There are several categorical data in the datasets. One of the techniques to derive new features is by applying one-hot encoding on categorical data. One-hot encoding is a technique used to represent categorical variables as binary vectors. One-hot encoding is useful for machine learning algorithms because it transforms categorical data into a format that can be easily interpreted and used by models.

In one-hot encoding, each categorical value is mapped to a binary vector of zeros and ones. The length of the binary vector is equal to the number of possible categories, and each category is represented by a unique combination of zeros and ones. Specifically, the index corresponding to the category is set to one, and all other indices are set to zero. By doing so, issues of ordinality and magnitude that may arise can be avoided if categorical variables are treated as numerical.

This technique is applied on the `feature_1` and `feature_2` columns in the train dataset to derive new columns. After deriving the new columns from one-hot encoding, the first column of the resulting binary vector is dropped. This is to avoid the "dummy variable trap" as the first column can be predicted based on the values of other columns. In other words, this first column is redundant and may lead to multicollinearity of the training model.

2.2.2 Feature Generation

There are two parts to feature generation. Firstly, the data-centrality based features and re-creating low-level features from the given features.

The data-centrality based approach serves as a method to group individual rows of transaction data into a single row for each unique `card_id`. This approach groups transactions based on `card_id`, aggregates the mean, min and max values for each of the numerical columns in the existing dataset.

Inspiration for the creation of low-level features was from a Kaggle notebook titled "Target-True meaning revealed" [1]. From their findings, it has been derived that the target is related to the ratio of money spent in the future divided by money spent in the past. This implies that the target is likely related to the ratio of `purchase_amount` in positive `month_lag` over negative `month_lag`. While the exact calculation may not be known, it provides some direction on how to proceed with feature generation.

The raw purchase amount has also been reverse-engineered by the author, and this has been used to better assist with feature generation. A column named "`ratio_between_ave_monthly_purchase_raw_for_positive_and_negative`" is

generated, which is the average monthly raw purchase amount of positive `month_lag` over that of the negative for each `card_id`. Additionally, a column named `ratio` has been generated, which is the average ratio of all ratios of `month_lag[i]/month_lag[i-1]` for each `card_id`.

A number of card ids do not have transactions for positive `month_lag`. This is likely due to them being intentionally hidden by the organisers rather than actually not having any transactions in those months so as to prevent participants from getting an RMSE of 0.000 by figuring out exactly how the target value is calculated, as mentioned in Raddar's Kaggle notebook [1]. As a result, Autoregressive Integrated Moving Average (ARIMA) was attempted on these card ids to predict the missing total raw purchase amounts for those `month_lag` to better generate the ratios. However, the non-positive `month_lag` only range from -13 to 0, and many of the card ids do not have their first transactions dated back as far and only has a small number of transactions made during negative `month_lag`. This may not provide enough information for the ARIMA model to capture the underlying patterns, trends, and seasonality in the time series, possibly leading to inaccurate predictions.

2.2.3 Feature Selection

There are commonly 2 methods for feature selection, namely filter and wrapper methods. Both are tested in this project.

The wrapper method creates many models with different subsets of input features and selects those features that result in the best-performing model according to a performance metric. These methods are unconcerned with the variable types, although they can be computationally expensive. In our case, the Recursive Feature Elimination method is used due to the large number of features in the dataset, with Linear Regression being used as the estimator.

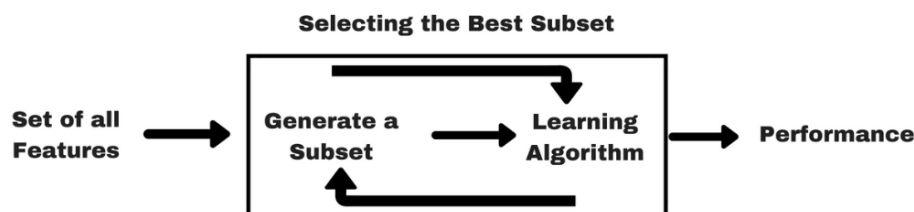


Figure 6: Wrapper Method for Feature Selection

The filter method uses statistical techniques to evaluate the relationship between each input variable and the target variable, and these scores are used as the basis to filter those input variables that will be used in the model. Pearson correlation coefficient is used to find the best features due

to the target output being a numerical value. After that, the `SelectKBest` function in Sckit-learn is made use of to identify the top features. This function selects features according to the k highest scores. By trial and error, we found that choosing 35 features gives the best result.

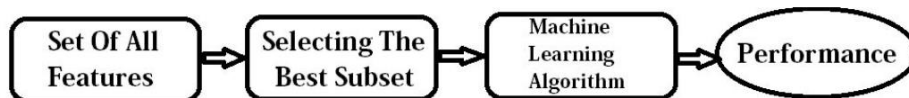


Figure 7: Filter Method for Feature Selection

Besides, we tested a tree-based method, namely Light Gradient Boosting Machine Regressor (LGBM Regressor) to see its effectiveness. LGBM Regressor is a type of gradient-boosting algorithm that uses a set of decision trees to make predictions. It is called "Light" because it is designed to be faster and more memory-efficient than other gradient-boosting algorithms. The algorithm works by iteratively adding decision trees to the model, where each new tree is trained to correct the errors made by the previous trees. In this way, the model gradually learns to make more accurate predictions.

The results for different feature selection methods are shown in Figure 8.




 LGBRegressor.csv Complete (after deadline) · now	3.64913	3.74078	<input type="checkbox"/>
 SelectKBest.csv Complete (after deadline) · 32m ago · SelectKBest	3.64782	3.73978	<input type="checkbox"/>
 LinearRegressionRFE.csv Complete (after deadline) · 2h ago · LinearRegressionRFE	3.68561	3.77844	<input type="checkbox"/>

Figure 8: RMSE Scores for Models Trained on Different Feature Selection Methods on Kaggle

3 Methodology for Machine Learning Phase

This section details the methods used to train models to predict the `test.csv` file.

3.1 Model Architecture

To tackle the challenge of having rare points whose loyalty scores are very far from the median value, Patekha E. created a model architecture with two regression models [2]. The sample is first predicted to be an outlier by a binary classifier. Next, the author created two regression models – one trained on low concentration of outliers, and the other on high concentration of outliers. With a third regression model that trains on all the training data, the predictions from the three models are blended to form the final prediction.

Inspired by this approach, the model architecture is redesigned, as depicted in **Error! Reference source not found.**. This architecture is similar to stacking, where the meta-model receives input data from three regressors. To lower the correlation between first level regressors and allow the model to predict the loyalty scores of rare points more accurately, each regressor is trained on different data. The regressors are trained on training subsets with low concentration of rare points, training subsets with high concentration of rare points, and the entire training set. The first dataset consists of all non-rare points and 20% of rare points, whereas the second dataset consists of all rare points and non-rare points that amount to 1/5 of the number of rare points.

Contrary to Patekha's architecture, instead of predicting a sample as a rare point, this binary classifier outputs the probability that this sample is a rare point as input to the meta-model.

The predicted loyalty scores from the regression models and probability from the binary classifier will then be the training data for the meta-model, which will output the final loyalty score for that sample. The rationale of creating the meta-model is to allow the meta-model to learn a more accurate weighting mechanism on the suggested loyalty score from each regression model. Therefore, with more information about the probability that this sample is rare, the meta-model can make more intelligent guesses about which loyalty scores to believe. Figure 9 summarises the approach detailed above in a model architecture diagram.



Figure 9: Model Architecture Used in this Project

Since Patekha's architecture does not feed any output from the binary classifier to the meta-model, an alternative model is also attempted. This second model architecture does not feed `rare_prob` to the meta-model, effectively supplying only the 3 predicted regression values to the meta-model. In short, two model architectures are experimented.

3.1.1 Binary Classifier

The training for binary classification has two challenges: (1) choosing a classifier (2) training on imbalanced data. This subsection discusses the first approach – choosing a classifier.

Exploratory analysis shows that the loyalty scores of the rare points are less than -30. Therefore, a Boolean ‘outlier’ column is added to the training data to indicate if such points are rare.

8 binary classifiers were attempted: K Neighbours, Light GBM, Random Forest, Bagging with LGBM classifiers, Bagging with Decision Tree classifiers, Gradient Boosting and Adaptive Boosting.

To select the best classifier, 5-fold cross validation is performed on the dataset to predict the rare points. Since the rare data points occupy 1.09% of the train set, it is extremely-to-moderately imbalanced [3], risking the binary classifier to predict all samples as non-rare for accuracy. Therefore, F_1 score is used. The formula is given by:

$$F_1 = \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

F_1 is chosen because precision or accuracy alone is insufficient to ensure that the model does not simply classify all samples as non-rare. To ensure that the model learns the features of a rare point, recall should be factored into the metric. Table 3 shows that Bagging of LGBM classifiers achieves the best performance, after using the best sampling strategy, which will be detailed in the next section.

Table 3: F1 Score for Classification of Rare Points for Each Classification Algorithm

	F1 Score
K-Neighbor	0.030787
LightGB	0.158998
RF	0.152092
LogReg	0.022671
Bag-LGBM	0.164485
Bag-DecTree	0.106386
GradBoost	0.119687
AdaBoost	0.119836

Light GBM is the next best performing model, followed by Random Forest. Although both are ensemble models, Light GBM performs better possibly because it builds new trees sequentially to correct errors made in the previous tree, instead of building independent trees that do not learn from previous errors like in Random Forest. Therefore, by additionally perform Bagging on Light

GBM classifiers, the correlation between trees is further reduced, lowering the overfitting problem Light GBM often has than Random Forest. This also improves the generalization error and F1 score.

3.1.2 Sampling

To train the binary classifier on imbalanced data, resampling is required. In general, resampling has two categories: oversampling data of the minority class, or undersampling data of the majority class. The oversampling techniques used include Random Oversampling, Synthetic Minority Oversampling Technique (SMOTE), whereas the undersampling techniques include Random Undersampling, Tomek Links and Near Miss.

Resampling can be performed with multiple samplers. For instance, after increasing the number of minority samples with random oversampling, random undersampling of the majority samples can be applied. This strategy is also experimented to explore its contribution to higher F1 scores.

To identify the best set of samplers, LightGBM binary classifier is used as the benchmark algorithm, and the mean of F1 scores from cross-validations are used as the evaluation metric.

Table 4 shows that SMOTE technique performs the best and is selected as the sampling strategy. From the table, R-Over is random oversampling, whereas R-Under(1:4) indicates random undersampling to achieve a ratio of 1:4 for minority-to-majority class samples. The sampling sequence of SMOTE, undersampling with 1:4 ratio, and random oversampling of rare points is the best set.

Table 4: F1 Score for Binary Classification with Different Sampling Strategies

	F1 Score
R-Over	0.116932
R-Under(1:1)	0.083522
R-Under(1:4)	0.112947
Tomek	0.115873
SMOTE	0.159081
NearMiss	0.022505
SMOTE,R-Under(1:4),R-Over	0.164989
SMOTE,R-Under(1:1),R-Over	0.157395
SMOTE,Tomek	0.157731

This is probably because SMOTE generates points that exist in the similar subspace as the rare points, introducing variety to the rare points pool. Additionally, the random samplers balanced the dataset to 1:1 ratio. Random undersampling is performed before oversampling because the dataset is heavily imbalanced. However, as the majority points may have wide variations, random

undersampling is performed at 1:4 ratio instead of 1:1 ratio, before random oversampling takes place.

3.1.3 Upweighting

During undersampling, upweighting should be performed. Upweighting is the addition of example weights to the downsampled class, thereby helping in calibrating the model and allowing the outputs to be interpreted as probabilities [3].

In tree classifiers, upweighting is done by setting the `class_weight` parameter to “balanced”, setting the classifier to compute the weights to be inversely proportional to the class frequencies of the input data. For a Random Forest classifier that produces multiple trees, another option is “balanced_subsample”, which computes the weights based on the bootstrap sample for each tree created [4].

3.2 Model Evaluation

The candidate models, namely, Random Forest Regressor, LGBM Regressor, Linear Regression and Filter Method (`f_regression`), were evaluated using K-Fold Cross-Validation with 100 iteration each. The purpose of this is to examine the effects of using different models on the same training data, with different training and testing splits. This serves two purposes. On one hand, this provides a basis of comparison of performance across the candidate models we selected. On the other hand, the model’s learning ability from the training data can be determined with some level of certainty. The next sub-section details the process of selecting the model from a pool of candidates.

3.3 Model Selection

This sub-section covers the selection of the best models from the pool of candidate models.

There are many regression models that are available to use for similar problems. In our initial iterations, we started by building a linear regression model, labeled the simplest regression model available. The results were unsurprisingly undesirable. Through several iterations and re-evaluations, the team decided to use LGBM Regressor as the main model as it performed the best consistently over multiple iterations and testing.

The approach adopted for model selection involved rounds of training and testing using an End-to-End machine learning pipeline that is refined through the duration of the project. Figure 10 illustrates the model selection process.

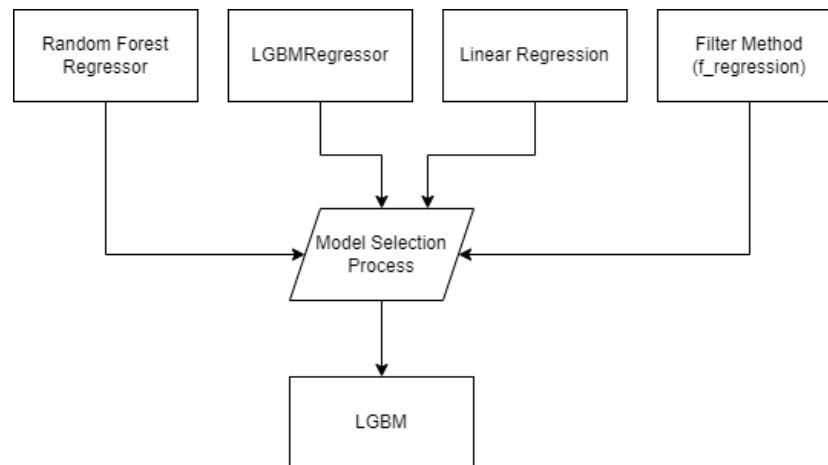


Figure 10: Model Selection Process

3.4 Hyperparameter Tuning

Although the classifier and regression models are selected after conducting validation for possible machine learning algorithms, these models are evaluated with their default hyperparameters. For instance, the default number of estimators in a Light BGM regressor is 100, but this regressor may perform at its peak with 60 estimators.

With many continuous hyperparameters to tune for each model, pure grid search may be impractical, whereas random search does not deterministically yield the best hyperparameter combination. Therefore, two approaches are utilised.

3.4.1 Random Search and Grid Search

This method initially sets random values for hyperparameters in the valid space. After a few random combinations, the results are manually inspected to identify the best hyperparameter set. For each hyperparameter in this set, a few possible values are considered for grid search.

By combining both search methods, the random search reduces the need to explore every possible hyperparameter value, whereas grid search systematically tests a few sets that potentially promise a more performing model.

3.4.2 Bayesian Optimisation

The first search strategy aims to balance exploration and exploitation. Similarly, Bayesian Optimisation exists for the same aim and can additionally solve the constrained global optimisation problem. In this technique, given an objective function that trains the model and outputs the score, the posterior distribution of functions that best describes this objective function is constructed by initially using random combinations of hyperparameters. As the number of score observations

increase, this posterior distribution improves, allowing the technique to better estimate the hyperparameter spaces that are worth exploring more [5].

To tune with Bayesian Optimisation algorithm, a minimum of 10 initial random combinations are tested for a given model. For linear regression models, 60 explorations after the random exploration are performed.

4 Results

4.1 Leaderboard

The best performing models built on different methodologies are used to predict the competition's `test.csv` dataset. Figure 11 shows the scores attained in the Kaggle leaderboard. In this module, the Public Leaderboard ranking is used as the metric.







Overview	Data	Code	Discussion	Leaderboard	Rules	Team	Submissions	Late Submission	...
	Best_CZ4041-2 Model_SelectKBest.csv						3.64782	3.73978	<input type="checkbox"/>
Complete (after deadline) · 7h ago · Custom Model: 3 regressors with Top 35 features chosen by SelectKBest + 1 classifier									
	Best_CZ4041-2 Model_Stack All Regressors.csv						3.65686	3.73247	<input type="checkbox"/>
Complete (after deadline) · 7h ago · Custom Model with 3 regressors trained on Top 35 features by RFE + classifier on rare point									
	Best_CZ4041-2 Model_Stack 3 Regressors Only.csv						3.6679	3.74101	<input type="checkbox"/>
Complete (after deadline) · 8h ago · Custom Model with 3 regressors trained on Top 35 features by RFE									
	Best_CZ4041-Pure LGBM.csv						3.64817	3.73665	<input type="checkbox"/>
Complete (after deadline) · 8h ago · Pure LGBM Regressor trained on all features									
	Best_CZ4041-Pure Random Forest.csv						3.76372	3.86741	<input type="checkbox"/>
Complete (after deadline) · 8h ago · Pure Random Forest Regressor trained on all features									
	Best_CZ4041-Pure Decision Tree Regressor.csv						3.68666	3.78564	<input type="checkbox"/>
Complete (after deadline) · 8h ago · Pure Decision Tree Regressor trained on all features									

Figure 11: Scores of Predictions of Performing Models on Kaggle (in reverse chronological order)

The first performing models in this project are the pure models of Decision Tree Regressor, Random Forest Regressor and Light GBM Regressor. Light GBM Regressor performs best possibly because of its model complexity that iteratively grows a tree regressor after penalising the errors made in the previous tree regressor.

Next, the model architecture presented in “3.1 Model Architecture” is used, where the first level models are trained on the Top 35 features selected by RFE. By stacking the predictions from 3 regressors of which 2 trained on different concentrations of rare points, the meta-model could have more prediction references. This ensemble model performs better when a binary classifier's prediction of whether a data point is rare is fed to the meta-model, helping the meta-model to

determine whether to allocate more weight to the prediction from a regressor trained on concentrated rare points. Its score of 3.73247 is the best score achieved by this group.

To experiment if a different feature selection method can improve the model performance, SelectKBest algorithm is also used, yielding a similar best score of 3.73978.

Figure 12 shows the ranking in Kaggle's Public Leaderboard with the score of 3.73247, which shows the final standings. The attained rank is 2744 out of 4110 (66.8th percentile).




2743	hejsa32k		3.73233	11	4y
2744	dbraun31		3.73273	2	4y
2745	mrroeo		3.73283	12	4y

Figure 12: Ranking on Kaggle's Public Leaderboard

5 Project Management

5.1 Timeline

Figure 13 shows the project timeline in a Gantt chart. In general, the team started early for the exploration, development and experimentation phases to attempt several models.

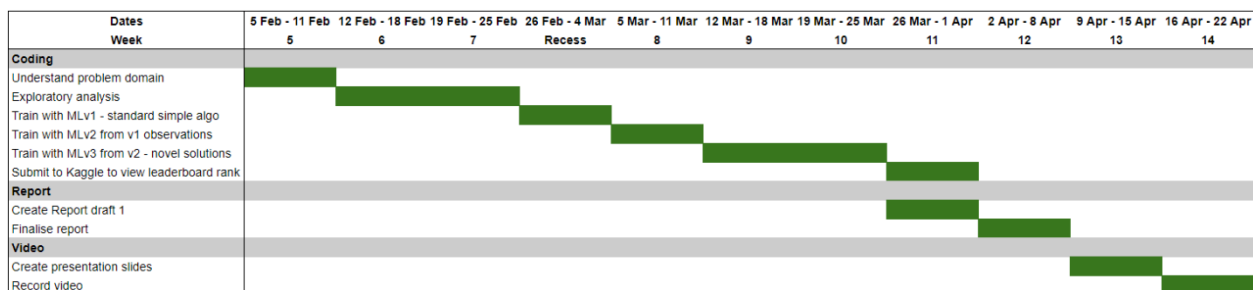


Figure 13: Project Timeline

5.2 Contributions

Table 5 shows the breakdown of tasks by each team member. To ensure high concurrency in experimenting, the tasks are broken down according to the phases in machine learning pipeline. This project would not be successful without the collaboration of each team member, taking care of his assigned task, sharing experience in other pipeline phases, and maintaining the codebase that contains reusable modular functions for this project.

Table 5: Team Contributions

Name	Tasks
Darryl See Wei Shen	Model training, evaluation and experimenting
Dinh Phuc Hung	Feature engineering (statistical analysis), model exploration
Goh Peng Aik	Data cleaning (data correlation, feature generation), exploratory analysis
Lau Zhen Jie	Data cleaning (imputation, one-hot encoding), hyperparameter tuning
Oong Jie Xiang	Model architecture, hyperparameter tuning

6 Conclusion

The Elo competition is a regression problem that requires careful feature handling due to the anonymity of features, huge raw feature list that influences the stochasticity of splitting the data for machine learning, and the presence of rare data points. To address this, extensive preprocessing and feature engineering are conducted before the machine learning task. Additionally, a novel solution detailing an ensemble model architecture is explored and presented.

By experimenting with various sampling and model algorithms before applying hyperparameter tuning to improve the performance of the models and stabilising the learning of the model by applying K-Fold cross validation method, the team achieved 3.73247 in the Kaggle competition. Nevertheless, a future improvement is to submit the results of candidate models to Kaggle early so that the performance of these best models on Kaggle’s test dataset can be observed quickly, thereby allowing more iterations in experimentation and development with a direction.

The key lessons learnt from this project are (1) the pre-machine learning phase is as important as the machine learning phase (2) each model should be adjusted, or even pieced with other models to achieve optimal performance and (3) the result in the training phase may not be a strong indicator of the final test result.

7 References

- [1] “target - true meaning revealed!” <https://kaggle.com/code/raddar/target-true-meaning-revealed> (accessed Apr. 19, 2023).
- [2] “Elo Merchant Category Recommendation.” <https://kaggle.com/competitions/elo-merchant-category-recommendation> (accessed Mar. 26, 2023).
- [3] “Imbalanced Data | Machine Learning,” *Google Developers*. <https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data> (accessed Mar. 26, 2023).
- [4] “sklearn.ensemble.RandomForestClassifier,” *scikit-learn*. <https://scikit-learn/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> (accessed Mar. 26, 2023).
- [5] fernando, “Bayesian Optimization.” Apr. 08, 2023. Accessed: Apr. 09, 2023. [Online]. Available: <https://github.com/fmfn/BayesianOptimization>