NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ4041 Machine Learning

Kaggle Challenge - IEEE-CIS Fraud Detection

Academic Year 2019-20 Semester 2

Group 23
Goh Yong Wei (U1722195H)
Ng Chen Ee Kenneth (U1721316F)
Ng Huan Zhang (U1720445C)
Ngo Jun Hao Jason (U1721978B)

Table of Content

Team Members' Responsibilities

Huan Zhang:
- feature selection (for M and C)
- report writing (for feature selection)

Kenneth:
- project management
- technical leadership
- exploratory data analysis
- feature selection (for other transaction features)
- report writing (for various parts of the report)
- video creation (including powerpoint slides and script)

Jason:
- feature selection (for D and identity)
- model evaluation
- hyperparameter tuning
- ensemble learning
- feature engineering (for all engineered features)
- report writing (for various parts of the report)

Yong Wei:
- feature selection (for V)
- research on potential features for feature engineering
- feature engineering
- report writing (for feature selection)

Kaggle Result

public score: 0.943850
public leaderboard rank: 2744 out of 6381 (~43.00 percentile)

private score: 0.917212
private leaderboard rank: 2451 out of 6381 (~38.41 percentile)

Problem Statement

The aim of the challenge is to benchmark machine learning models on a challenging large-scale dataset to predict if a transaction made is fraudulent. The data comes from Vesta's real-world e-commerce transactions and contains a wide range of features from device type to product features.

Methodology

Exploratory Data Analysis

notebook: source_code/exploratory_data_analysis/eda_fraud_detection.ipynb

There are a total of 4 datasets provided, train_transaction.csv, train_identity.csv, test_transaction.csv and test_identity.csv. There are a total of 434 columns and 590540 rows for the training set. These columns can be divided into categories below.

Summary of features:

Column	Description
TransactionDT	timedelta from a given reference datetime (not an actual timestamp)
TransactionAmt	Transaction amount in USD
ProductCD	product code, the product for each transaction
card1-card6	Card used for payment
addr	address
dist	distance
P_ and (R_) email domains	purchaser and recipient email domain
C1 - C14	counting, address and other things, actual meaning is masked
D1 - D15	timedelta, such as days between previous transactions. etc.
M1 - M9	match, such as names on card and address, etc
V1 - V339	Vesta engineered rich features, including ranking, counter and other entity relations

Feature Selection

In the given dataset, there are 434 columns (including isFraud, the target variable). However, not all of the features are necessarily useful. These features may make it more difficult for the model to be trained, or worse still, reduce its performance. As such, We need to figure out which of these features should be kept, and which should be removed.

Techniques used for feature selection

Recursive Feature Elimination with Cross Validation

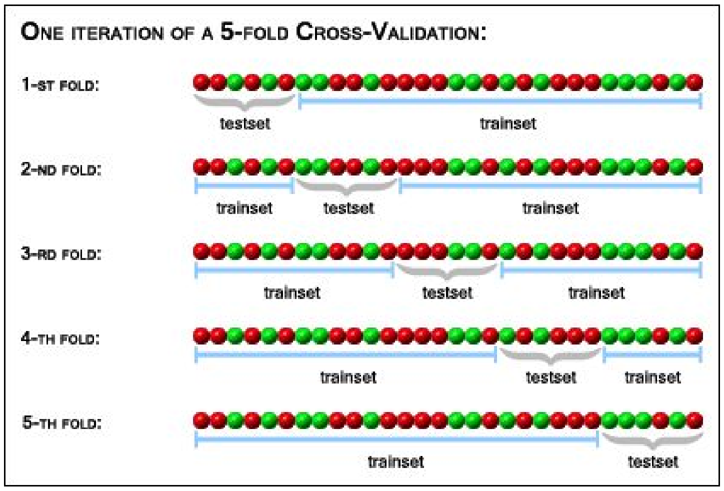
Recursive Feature Elimination (RFE) is a technique that, given a set of features and a classifier, returns the importance of the features in training the classifier [1].

RFE works by doing the following steps iteratively:

1. fit the given classifier using the given set of features
2. compute the ranking criterion for each of the features
3. remove the lowest-ranked feature, based on the computed ranking criterion

Although we can get the relative importance of each feature using RFE, we would not know what the optimal number of features to use is. For that, we need Recursive Feature Elimination with Cross Validation (RFECV), which is essentially performing RFE along with a cross validation technique such as k-fold cross validation.

Illustration of a 5-fold cross-validation (for 1 iteration) [2]:



In k-fold cross validation, the data set is first separated into *k* subsets of data (or folds), depending on what *k* was set to. Each of these *k* subsets would be used as the validation set once, during which the other subsets are used as the training set. A score is then generated based on the average of the validation scores for all the runs. For this challenge, the scoring method is the area-under-curve of the receiver operating characteristics curve (AUC).

By using k-fold cross validation with RFE, we can obtain the optimal number of features since the run with the optimal number of features would return the highest cross validation score.

Correlation Analysis

Any 2 features could have high positive or negative correlation (or collinearity). In the case of 2 highly positive collinear features, they follow a positive linear relationship: when one feature's value increases, the other feature's value increases as well. On the other hand, in the case of high negative collinearity, when one feature's value decreases, the other feature's value increases.

Highly collinear features are usually not good for machine learning as they lead to a high standard error. Also, it may be more difficult to determine which feature in the set of highly collinear features affected the machine learning model result.

For all of our correlation analyses, we used Pearson's correlation coefficient as our measurement of correlation.

Individual feature selection

We divided the dataset into V, identity, M, C, D, and other transaction features to distribute the workload among ourselves.

Feature selection for V features

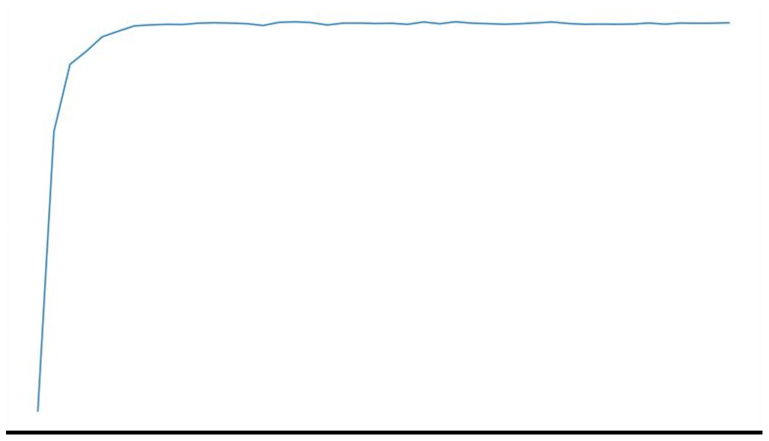
notebook: source_code/feature_selection/V_feature_selection.ipynb

V features have the most columns, so it is hard to decide which features to select among them. We decided to use RFECV to select the required features.

Why use RFECV for V features?
Since there are so many columns for V features and each feature is too vague to see any relationship between a single V feature and any other features, we decided that RFECV, an automatic method of selecting features, is the best approach to siphon out the best V features.

Approach to feature selection for V features
Preprocessing
Certain useless features could be removed for the selection. These features include features with more than 90% Not-a-Number (nan) values, or only one unique value. We removed these features before performing RFECV as we know that they are bad features to use and will just slow down the training process.

Plot of cross-validated AUC vs number of features selected



The graph above shows the change in cross validation score against the number of features selected. We see that as the number of features selected increases, the cross validation score also increases sharply and plateaus after a certain number of features are selected. We found out that the optimal number of features to be around 261 features.

How are the V features selected?
After running RFECV, the optimal features in a model are given a ranking of 1, which means that we can iterate through the columns to find the features with this ranking. These V features will then be selected for the training of our model.

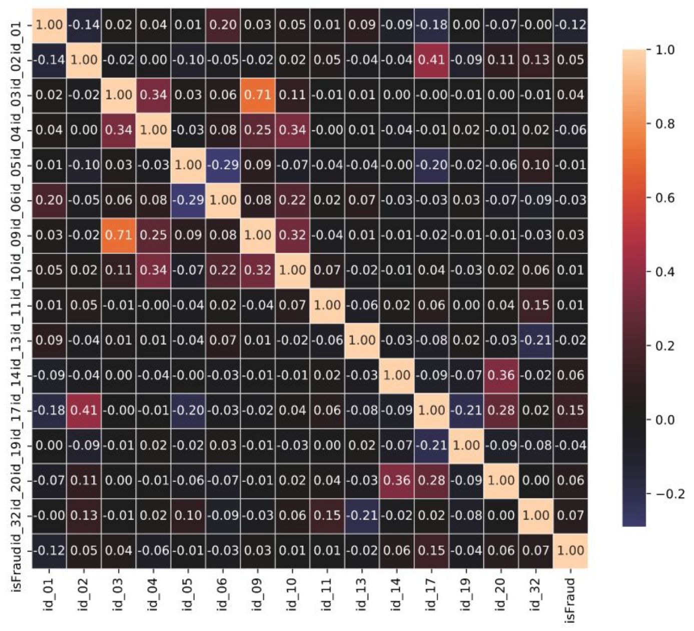
Feature selection for identity features

notebook: source_code/feature_selection/id_feature_selection.ipynb

Variables in this set of data are user identity information – network connection information (internet protocol, internet service provider, proxy, etc) and digital signature (internet browser, operating system, etc) – associated with transactions.

We start by removing features with more than or equal to 60% nan values as they have too many missing values to be useful.

Correlation Analysis for identity features



According to the correlation heatmap, since the identity features are not highly correlated to each other, there is no need to remove any of them.

RFECV for identity features
Prior to performing RFECV, we tried doing preliminary feature engineering for id_30 and id_31 (aggregating operating system and internet browser data based on name, dropping version number in the process). We performed RFECV for the identity features, with and without the aforementioned feature engineering, using Random Forest classifier and LightGBM classifier.

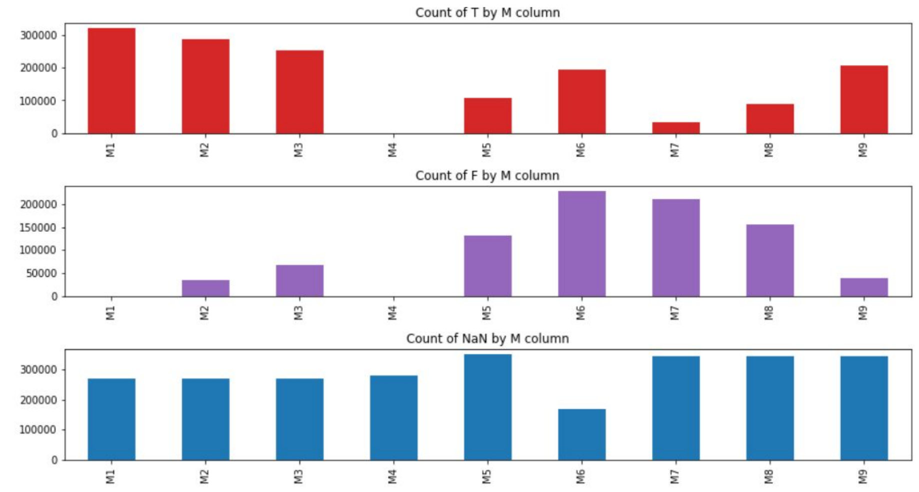
Based on the results of running RFECV, the optimal number of features to be used is 28 (id_28 and id_35 should be dropped) since this leads to the highest AUC. The combination that gave the highest AUC is the usage of feature engineering and LightGBM classifier.

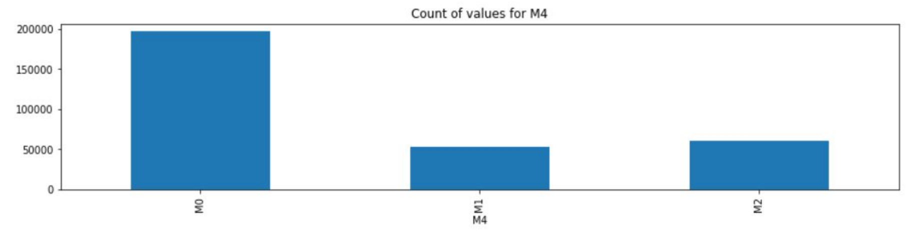
Feature selection for M features

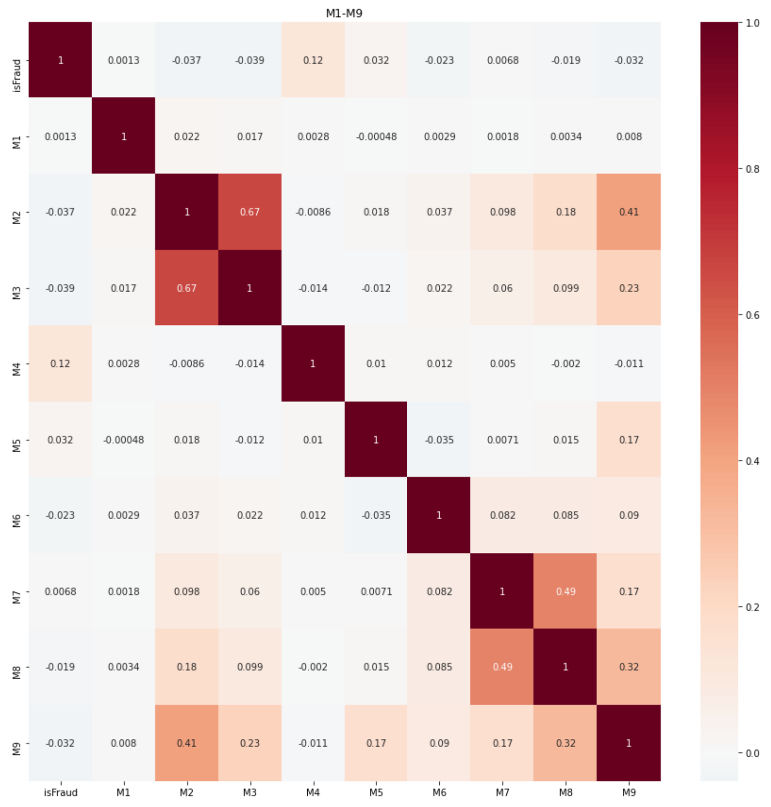
notebook: source_code/feature_selection/M_feature_selection.ipynb

By examining the data, we found out that M1 has close to 0 False values. Thus, it does not contribute significantly towards finding out if a transaction is fraud or legitimate. A correlation heatmap is used to confirm the relation between isFraud and M features.

Plots for M features:



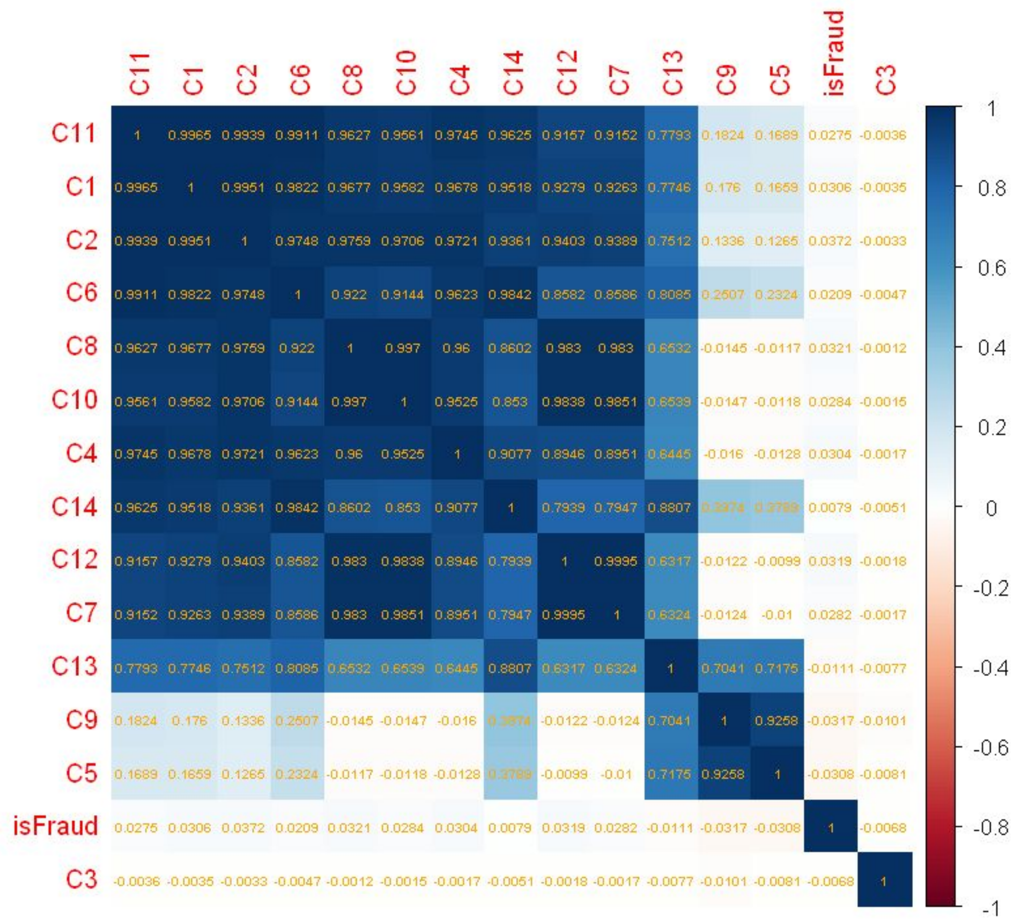




Feature selection for C features

notebook: source_code/feature_selection/C_feature_selection.ipynb

By examining the data, we found out that C3 has a correlation coefficient of -0.0068 when compared to isFraud. This would indicate that there is almost no correlation between C3 and isFraud which makes C3 not useful for predicting our target variable, isFraud.



	C11	C1	C2	C6	C8	C10	C4	C14	C12	C7	C13	C9	C5	isFraud	C3
C11	1	0.9965	0.9939	0.9911	0.9627	0.9561	0.9745	0.9625	0.9157	0.9152	0.7793	0.1824	0.1689	0.0275	-0.0036
C1	0.9965	1	0.9951	0.9822	0.9677	0.9582	0.9678	0.9518	0.9279	0.9263	0.7746	0.176	0.1659	0.0306	-0.0035
C2	0.9939	0.9951	1	0.9748	0.9759	0.9706	0.9721	0.9361	0.9403	0.9389	0.7512	0.1336	0.1265	0.0372	-0.0033
C6	0.9911	0.9822	0.9748	1	0.922	0.9144	0.9623	0.9842	0.8582	0.8586	0.8085	0.2507	0.2324	0.0209	-0.0047
C8	0.9627	0.9677	0.9759	0.922	1	0.997	0.96	0.8602	0.983	0.983	0.6532	-0.0145	-0.0117	0.0321	-0.0012
C10	0.9561	0.9582	0.9706	0.9144	0.997	1	0.9525	0.853	0.9838	0.9851	0.6539	-0.0147	-0.0118	0.0284	-0.0015
C4	0.9745	0.9678	0.9721	0.9623	0.96	0.9525	1	0.9077	0.8946	0.8951	0.6445	-0.016	-0.0128	0.0304	-0.0017
C14	0.9625	0.9518	0.9361	0.9842	0.8602	0.853	0.9077	1	0.7939	0.7947	0.8807	0.3874	0.3789	0.0079	-0.0051
C12	0.9157	0.9279	0.9403	0.8582	0.983	0.9838	0.8946	0.7939	1	0.9995	0.6317	-0.0122	-0.0099	0.0319	-0.0018
C7	0.9152	0.9263	0.9389	0.8586	0.983	0.9851	0.8951	0.7947	0.9995	1	0.6324	-0.0124	-0.01	0.0282	-0.0017
C13	0.7793	0.7746	0.7512	0.8085	0.6532	0.6539	0.6445	0.8807	0.6317	0.6324	1	0.7041	0.7175	-0.0111	-0.0077
C9	0.1824	0.176	0.1336	0.2507	-0.0145	-0.0147	-0.016	0.3874	-0.0122	-0.0124	0.7041	1	0.9258	-0.0317	-0.0101
C5	0.1689	0.1659	0.1265	0.2324	-0.0117	-0.0118	-0.0128	0.3789	-0.0099	-0.01	0.7175	0.9258	1	-0.0308	-0.0081
isFraud	0.0275	0.0306	0.0372	0.0209	0.0321	0.0284	0.0304	0.0079	0.0319	0.0282	-0.0111	-0.0317	-0.0308	1	-0.0068
C3	-0.0036	-0.0035	-0.0033	-0.0047	-0.0012	-0.0015	-0.0017	-0.0051	-0.0018	-0.0017	-0.0077	-0.0101	-0.0081	-0.0068	1

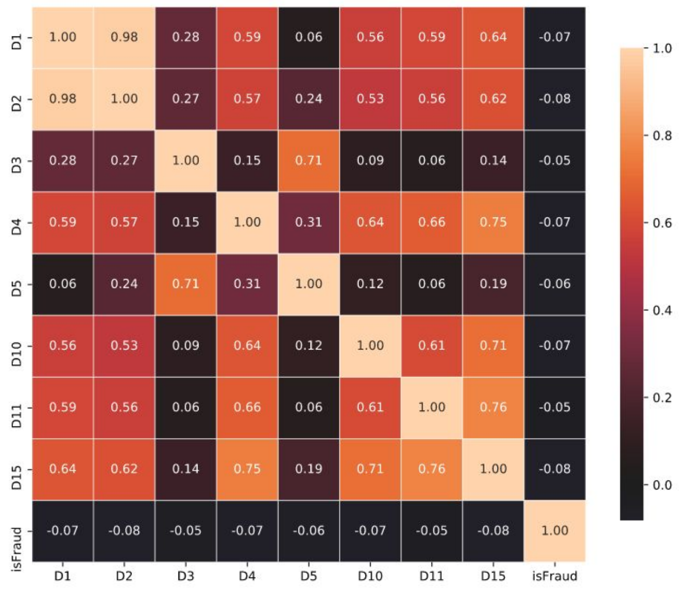
Feature selection for D features

notebook: source_code/feature_selection/D_feature_selection.ipynb

D features are time delta features. It signifies the number of days between transactions.

We start by removing features with more than or equal to 60% nan values as they have too many missing values to be useful.

Correlation Analysis for D features



Based on the correlation heatmap, there is a high correlation between D1 and D2, with a Pearson's correlation coefficient of 0.98. However, it is possible that this high correlation would not reduce our model's performance [3].

We performed RFECV for the identity features, with and without D2 in the training set, using Random Forest classifier and LightGBM classifier. We found that we can obtain the highest AUC by using all remaining features, including D2, with the LightGBM classifier.

Feature Selection for other transaction features

notebook: source_code/feature_selection/others_feature_selection.ipynb

For the remaining columns, TransactionID is removed as it is a feature that is completely distinct and used to merge identity and transaction csv files together. Dist2 has a disproportionately high percentage of NaN (93%) and was removed as a result.

Summary of features removed

Category	Features Removed
Transaction	'TransactionID' Dist2
D	'D6', 'D7', 'D8', 'D9', 'D12', 'D13', 'D14'
C	‘C3’
M	'M1'
V	'V6', 'V8', 'V9', 'V10', 'V11', 'V14', 'V15', 'V16', 'V18', 'V21', 'V22', 'V27', 'V28', 'V31', 'V32', 'V41', 'V42', 'V46', 'V50', 'V51', 'V59', 'V65', 'V68', 'V71', 'V72', 'V79', 'V80', 'V84', 'V85', 'V88', 'V89', 'V92', 'V93', 'V95', 'V98', 'V101', 'V104', 'V106', 'V107', 'V108', 'V109', 'V110', 'V111', 'V112', 'V113', 'V114', 'V116', 'V117', 'V118', 'V119', 'V120', 'V121', 'V122', 'V123', 'V125', 'V138', 'V141', 'V142', 'V144', 'V146', 'V147', 'V148', 'V151', 'V153', 'V154', 'V155', 'V157', 'V158', 'V159', 'V161', 'V163', 'V164', 'V166', 'V172', 'V173', 'V174', 'V175', 'V176', 'V177', 'V178', 'V179', 'V180', 'V181', 'V182', 'V183', 'V184', 'V185', 'V186', 'V190', 'V191', 'V192', 'V193', 'V194', 'V195', 'V196', 'V197', 'V198', 'V199', 'V214', 'V216', 'V220', 'V225', 'V226', 'V227', 'V230', 'V233', 'V235', 'V236', 'V237', 'V238', 'V239', 'V240', 'V241', 'V242', 'V244', 'V246', 'V247', 'V248', 'V249', 'V250', 'V252', 'V254', 'V255', 'V269', 'V276', 'V297', 'V300', 'V302', 'V304', 'V305', 'V325', 'V327', 'V328', 'V329', 'V330', 'V334', 'V335', 'V336', 'V337', 'V338', 'V339',
id	'id_07', 'id_08', 'id_18', 'id_21', 'id_22', 'id_23', 'id_24', 'id_25', 'id_26', 'id_27', 'id_28', 'id_35'

Modelling

notebooks: source_code/model_evaluation/try_lightgbm.ipynb, source_code/model_evaluation/try_xgboost.ipynb, source_code/model_evaluation/try_random_forest.ipynb

The models that we considered using are Light Gradient Boosting Machine (LightGBM), eXtreme Gradient Boost (XGBoost) and Random Forest - LightGBM and XGBoost for their known performance [4], and Random Forest due to our familiarity with it.

All 3 models are ensemble models - LightGBM and XGBoost make use of boosting, whereas Random Forest uses bagging.

Random Forest is an ensemble model that works by constructing multiple decision trees during training and outputs the class that has the majority vote among the trees. During training, it generates each decision tree by using bagging - training each tree on subsets of the training data, with replacement. Using multiple random decision trees instead of a single decision tree outperforms any of the individual constituent decision trees due to the increased diversity, leading to a better performance.

XGBoost is a scalable tree boosting system which has been used to win many machine learning competitions [5], [6]. It supports gradient-boosting decision trees (GBDT), which is the algorithm that we used within XGBoost [7]. The GBDT in XGBoost is a decision tree ensemble that consists of a set of classification and regression trees (CART), not too different from decision trees aside from the fact that leaves of decision trees contain decision values, whereas leaves of CART contain real scores.

During training, GBDT works by optimising an objective function (logistic regression, since we are using XGBClassifier) via gradient descent, adding a new decision tree which optimises the objective function at each step. It also learns the tree structure by adding splits one split at a time, provided the gain for adding the split is more than a regularisation term, γ (i.e. pruning). This approach tends to result in a final model with the best structure score, but has a possibility of failing due to edge cases.

LightGBM is a GBDT as well, but it includes 2 new techniques: Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) [8]. Although GOSS is supposed to be an improvement over GBDT, we still used GBDT within LightGBM since the notebook that we reference uses GBDT as their booster type [9].

LightGBM's EFB algorithm aims to reduce the number of features, since real-world features tend to be sparse. It consists of 3 main steps :
"

1. Construct a graph with all features, weighted by the edges which represents the total conflicts between the features.
2. Sort the features by their degrees in the graph in descending order.
3. Check each feature and either assign it to an existing bundle with a small conflict or create a new bundle.

" [10]

Other differences between LightGBM and other GBDT are its tree growth strategy and its support for categorical features.

Unlike other implementations such as XGBoost which grows their decision trees level-wise, LightGBM grows its trees leaf-wise. It splits the leaf that maximises its objective function and ignores the rest of the leaves in the same level. This results in asymmetrical trees which tend to have lower loss, but may overfit (however, overfitting is countered by the max_depth parameter).

For categorical features, LightGBM uses an algorithm by Fisher [11], [12] to find the optimal splits over categories. It works by sorting the categories according to the training objective at each split, then finding the best split.

We chose LightGBM as the final base classifier as it has the highest accuracy when we evaluate them (using reference parameters for LightGBM [9] and XGBoost [13], and default parameters for Random Forest). LightGBM is also commonly used in the winning ensembles in many Kaggle Competitions.

Results:

Model	LightGBM	XGBoost	Random Forest
Validation Score (AUC)	0.971931	0.975871	**0.999903**
Kaggle Public Score	**0.939755**	0.937507	0.898593
Kaggle Private Score	**0.911989**	0.908053	0.868237

Hyperparameter Tuning

notebook: source_code/hyperparameter_tuning/tune_params.ipynb

After selecting LightGBM as our base classifier, we attempted hyperparameter tuning to find parameters that will lead to higher AUC.

We tried to tune hyperparameters automatically by using Optuna. Optuna searches for the best parameters via trial and error after we input the parameters to be optimised, along a range of values to search within for each parameter [14].

Due to time constraint, the fact that hyperparameter tuning is time consuming, and our notebook crashing continuously over multiple attempts of tuning (probably due to memory leaks), we only manage to run 7 trials. The parameters yielded by the top 3 trials did not lead to a better model performance as compared to the usage of reference parameters we started off with, so we decided to continue using the reference parameters.

Reference parameters:

```
params = {
    'num_leaves': 491,
    'min_child_weight': 0.03454472573214212,
    'feature_fraction': 0.3797454081646243,
    'bagging_fraction': 0.4181193142567742,
    'min_data_in_leaf': 106,
    'objective': 'binary',
    'max_depth': -1,
    'learning_rate': 0.006883242363721497,
    'boosting_type': 'gbdt',
    'bagging_seed': 11,
    'metric': 'auc',
    'verbosity': -1,
    'reg_alpha': 0.3899927210061127,
    'reg_lambda': 0.6485237330340494,
    'random_state': 47,
    'num_boost_round': 10000,
    'early_stopping_rounds': 500,
}
```

Results:

	reference	best trial	2nd best trial	3rd best trial
Kaggle Public Score	**0.939755**	0.938188	0.934179	0.936401
Kaggle Private Score	**0.911989**	0.907923	0.905338	0.903841

Ensemble Learning

notebooks: source_code/ensemble/try_bag_avg_vote.ipynb, source_code/ensemble/try_skf_avg_vote.ipynb, source_code/ensemble/try_0_to_19.ipynb

Our next attempt at improving our model's performance is to combine the predictions of multiple LightGBM classifiers.

Since the output required is the probabilities of fraud for each transaction, we cannot use majority voting as that will skew our results. The base classifiers are all LightGBM classifiers, so they should theoretically have the same overall performance (though each of them may have a different focus on the feature they were trained on due to the random splitting of training and validation sets). As such, they should all be given equal weights when voting which means we should use a simple average voting (i.e. summing the predictions up and dividing by the total number of predictions to get the average prediction).

We tried bootstrapping, as well as sampling without replacement by doing a stratified k-folds split (where *k* = 10) to ensure an even distribution of the target variable across splits, but did not notice any significant difference in performance using either strategy. In both instances, 10 LightGBM classifiers were used.

When we used all 20 of these classifiers for average voting, the final prediction's private LB score is significantly higher. This aligns with what we learnt in lecture: assuming that the base classifiers are independent of each other (or slightly correlated), the ensemble's performance should improve if more base classifiers are used.

Results:

	Bootstrapping (10 classifiers)	Stratified k-Folds (10 classifiers)	Use all 20 base classifiers
Kaggle Public Score	0.941287	0.939414	**0.941296**
Kaggle Private Score	0.913408	0.915039	**0.916027**

Feature Engineering

Feature engineering is the method of converting raw data into features that better reflect the underlying issue for the predictive models, resulting in enhanced model accuracy on data that is not used.

We performed the following steps, sequentially, to augment selected features, based on a Kaggle discussion regarding feature engineering [15]:

Categorical feature representation

notebook: source_code/feature_engineering/categorical_features.ipynb

Although LightGBM has internal support for categorical features [11], we considered using label encoding as well in case it provides better results. Label encoding basically converts each category in a categorical feature to an integer to represent it. One problem with label encoding, however, is our model may wrongly learn an order between each category since the categories are now integers.

Nevertheless, we ran an experiment comparing letting LightGBM handle categorical features on its own, using label encoding, and a mixture of the previous 2 methods (let LightGBM handle the categories unless the number of categories is more than a threshold, *k*, in which case we switch to usage of label encoding). Based on the experiment's result, we cannot tell if any of these methods are better than the others, but decided to use the mixture and tune *k* along the way. Over the runs of several notebooks, we found that 30 is a reasonable value for *k* for optimal AUC.

Results:

Method	Let LightGBM handle	Label Encoding	Mixture of both (LE when *k* > 10)
Validation Score (AUC)	**0.976294**	0.974817	0.973303
Kaggle Public Score	0.936043	0.937378	**0.937547**
Kaggle Private Score	**0.909225**	0.905007	0.907447

Normalisation of numerical data

notebook: source_code/feature_engineering/normalisation.ipynb

The goal of normalisation is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the range of values. We carried out and tested out mean normalisation and the results show that not normalising the numerical columns gives a better score.

Results:

Preprocessing Method	No Normalisation	Normalisation
Validation Score (AUC)	**0.977915**	0.975334
Kaggle Public Score	**0.937887**	0.906302
Kaggle Private Score	**0.906541**	0.859245

Handling missing values

notebook: source_code/feature_engineering/nan_processing.ipynb

There are 3 methods that we tested to handle the remaining data that has missing values.

The first method is to let LightGBM handle the missing values, which is the default method for LightGBM. It does so by ignoring the nan values in a feature when computing a split, then allocating all the data with missing values to whichever side of the split reduces the loss more. This reduces the number of samples that have to be used when evaluating each split, speeding up the training process.

The second method is by replacing the missing value with a value outside the feature range (we used the minimum value of the feature minus 1000). This makes the model not overprocess the missing values and give it the same attention as the other numbers.

The third method is imputation, replacing the missing values of a feature with a representative value, such as the mean/median of the non-missing values of the feature.

After testing each method, we concluded that using imputation leads to the best performance.

Results:

Preprocessing	Let LightGBM handle	Replace with value outside feature range	Interquartile Mean Imputation
Validation Score (AUC)	0.977405	**0.978898**	0.978087
Kaggle Public Score	0.937467	0.936985	**0.938098**
Kaggle Private Score	0.907941	0.909055	**0.912961**

Augment operating system and internet browser data

notebook: source_code/feature_engineering/os_and_browser.ipynb

A closer look at features id_30 and id_31 reveals that they are features referring to the operating system (OS) version and the internet browser version of the transaction. As there are a high number of categories that belong to these 2 features (76 and 131 categories), it is possible to aggregate categories that belong to the same OS or browser type to improve our model's performance [16]. In addition, there exists some OS and browsers that do not belong to any known OS and browsers. These illegitimate data are categorized as 'other', for each of these features.

After testing whether to group categories of OS and internet browser, we found that grouping browsers by their names (and disregard their version numbers) only leads to the best performance.

Results:

Preprocessing Method	No Preprocessing	Group OS Type	Group Browser Type	Group OS Version and Browser Type
Validation Score (AUC)	**0.979721**	0.975518	0.975599	0.975977
Kaggle Public Score	0.938374	0.939401	**0.941506**	0.940318
Kaggle Private Score	0.912327	0.909783	**0.915348**	0.912921

Extracting transaction day and time

notebook: source_code/feature_engineering/day_and_time.ipynb

According to a notebook on Kaggle [17] which discusses the feature TransactionDT, extracting the time (hour) of transaction may boost our model's performance. We tested various combinations of preprocessing against a control group (without preprocessing of day and time) and found that the control group has the best performance.

Results:

Preprocessing Method	No preprocessing	Extract day	Extract time	Extract day and time
Validation Score (AUC)	**0.978945**	0.977912	0.972903	0.978910
Kaggle Public Score	**0.940459**	0.938442	0.937576	0.939177
Kaggle Private Score	**0.914248**	0.910745	0.911587	0.912920

Feature aggregation

notebook: source_code/feature_engineering/aggregations.ipynb

Feature aggregation combines input features into a smaller set of features. Providing our LightGBM classifier with group statistics allows it to determine if a value is common or rare for a particular group. This can be done by calculating group statistics by providing Pandas with the group, variable of interest and type of statistic. In our case, we used the publicly released aggregated features [15] to help us improve the model's performance.

Results:

Preprocessing Method	without aggregations	with aggregations
Validation Score (AUC)	0.977745	**0.978004**
Kaggle Public Score	0.940043	**0.942078**
Kaggle Private Score	0.912476	**0.915865**

Selecting representative values for imputation

notebook: source_code/feature_engineering/imputation.ipynb

After finding out that imputation is the best way to handle nan values, we considered the fact that median or mode may be more representative of a feature's values as compared to interquartile mean.

Trying each of these values shows that using the median led to a significant increase in private leaderboard score, at the expense of a less significant decrease in public leaderboard score, as compared to using interquartile mean of feature values. Using the mode simply led to a significant deterioration of performance. Therefore, we used the median of feature values for imputation.

Results:

Preprocessing	Interquartile Mean	Median	Mode
Validation Score (AUC)	**0.978386**	0.972760	0.978069
Kaggle Public Score	**0.942113**	0.942052	0.940725
Kaggle Private Score	0.915057	**0.916371**	0.911430

Final Ensemble

notebook: source_code/ensemble/final_ensemble.ipynb

To get our final prediction, we performed bagging by training 20 LightGBM base classifiers on preprocessed features, and using average voting to aggregate each of the classifiers prediction. This method is essentially Random Forest, except that individual trees are LightGBM classifiers, and we used average instead of majority voting. This gave us our best public score of 0.943850 and private score of 0.917212.

Conclusion

Through our project, our group has explored the data and selected features that are relevant and removed features that do not contribute to the prediction of fraudulent transactions. The discussion forum and notebooks contain valuable information that allowed us to get started on understanding the data and steps that we needed to take to predict the target variable. We had also experimented with different preprocessing methods such as label encoding and filling missing values using various imputation methods.

From this project, we learned that each dataset is different and must be preprocessed differently to get the best score. We are also exposed to using state of the art models such as LightGBM and XGBoost which are excellent models that are commonly used to win Kaggle competitions.

Bibliography

[1] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, B. Bioinformatics, and T. Labs, ‘Gene Selection for Cancer Classification using Support Vector Machines’, p. 39.

[2] ‘ProClassify User’s Guide - Cross-Validation Explained’. https://genome.tugraz.at/proclassify/help/pages/XV.html (accessed Apr. 30, 2020).

[3] ‘Multicollinearity in Regression Analysis: Problems, Detection, and Solutions’, *Statistics By Jim*, Apr. 02, 2017. http://statisticsbyjim.com/regression/multicollinearity-in-regression-analysis/ (accessed Apr. 30, 2020).

[4] A. Anghel, N. Papandreou, T. Parnell, A. De Palma, and H. Pozidis, ‘Benchmarking and Optimization of Gradient Boosting Decision Tree Algorithms’, *arXiv:1809.04559 [cs, stat]*, Jan. 2019, Accessed: May 01, 2020. [Online]. Available: http://arxiv.org/abs/1809.04559.

[5] T. Chen and C. Guestrin, ‘XGBoost: A Scalable Tree Boosting System’, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, Aug. 2016, doi: 10.1145/2939672.2939785.

[6] ‘XGBoost Documentation — xgboost 1.1.0-SNAPSHOT documentation’. https://xgboost.readthedocs.io/en/latest/index.html (accessed May 01, 2020).

[7] ‘Introduction to Boosted Trees — xgboost 1.1.0-SNAPSHOT documentation’. https://xgboost.readthedocs.io/en/latest/tutorials/model.html (accessed May 01, 2020).

[8] G. Ke *et al.*, ‘LightGBM: A Highly Efficient Gradient Boosting Decision Tree’, p. 9.

[9] ‘LGB Single model [LB 0.9419]’. https://kaggle.com/nroman/lgb-single-model-lb-0-9419 (accessed May 01, 2020).

[10] M. Joseph, ‘LightGBM’, *Medium*, Feb. 29, 2020. https://towardsdatascience.com/lightgbm-800340f21415 (accessed May 01, 2020).

[11] ‘Advanced Topics — LightGBM 2.3.2 documentation’. https://lightgbm.readthedocs.io/en/latest/Advanced-Topics.html#categorical-feature-support (accessed May 01, 2020).

[12] W. D. Fisher, ‘On Grouping for Maximum Homogeneity’, *Journal of the American Statistical Association*, vol. 53, no. 284, pp. 789–798, Dec. 1958, doi: 10.1080/01621459.1958.10501479.

[13] ‘XGB Fraud with Magic - [0.9600]’. https://kaggle.com/cdeotte/xgb-fraud-with-magic-0-9600 (accessed May 01, 2020).

[14] ‘Optuna: An Automatic Hyperparameter Optimization Framework’, *Preferred Networks Research & Development*, Dec. 03, 2018. https://tech.preferred.jp/en/blog/optuna-release/ (accessed Apr. 30, 2020).

[15] Chris Deotte, ‘Feature Engineering Techniques’. https://www.kaggle.com/c/ieee-fraud-detection/discussion/108575 (accessed Apr. 30, 2020).

[16] ‘Extensive EDA and Modeling XGB Hyperopt’. https://kaggle.com/kabure/extensive-eda-and-modeling-xgb-hyperopt (accessed Apr. 30, 2020).

[17] ‘Day and Time - powerful predictive feature?’ https://kaggle.com/fchmiel/day-and-time-powerful-predictive-feature (accessed Apr. 30, 2020).