ISOM674: Machine Learning I
Final Project Description: Team 12

Roch Jia, Lessly Rocha, Ayah Assaadi, J.R. Noh, and Kaya Aslanoba

Data and Task Understanding

Dataset Overview

Two dataset files were provided for application in our project: ProjectTrainData.csv and ProjectTestData.csv. The training dataset contains 24 columns and more than 30 million rows. The test dataset contains 23 columns and more than 10 million rows. The data structure of the two datasets is basically the same, except that the test dataset does not contain the target variable "click." After checking the data summary for both datasets, we found that:

- Both, train and test datasets, are extremely large. In fact, using the full dataset for data manipulation and modeling led to RAM shortage and the program crashed many times during our coding process. Therefore, we decided to randomly sample a fractional dataset to avoid running out of RAM.
- The feature variables were categorical. In fact, there is no truly numerical variable in this dataset, and many categorical variables are encoded by both numbers and letters. Therefore, we knew that we would need to re-encode the categorical variables carefully to build our model. We approached this process carefully because we know that the approaches of encoding categorical variables can significantly impact the performance of models.
- There are many anonymized categorical variables, which complicates our model building process because we lack practical understanding of what they represent and therefore our understanding of their importance. So, we opted to not delete them without careful consideration.

Task Understanding

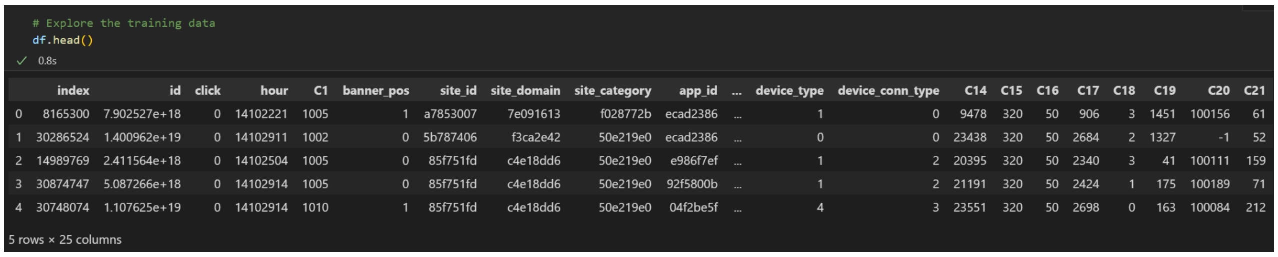
The goal of this project is to build and optimize a machine learning model to predict a variable called 'click'. This is a typical classification machine learning modeling task. No restriction was given for model selection, feature engineering, and parameter tuning but the evaluation criteria is log loss—representing the error on likelihood. Therefore, the aim of our project is to build a model that decreases the log loss. Lastly, the language that we used for our code was using Python 3.

Data Preparation

We approached the data preparation process in the following three steps:

1) Import the training data and take a random sample of 10% of the full dataset
2) Import the test data and align its data structure with training data (i.e. add a ‘click’ column for test data and reorder the columns; later on, we deleted the new column after the feature engineering)
3) Merge the two datasets in order to do feature engineering for both simultaneously. Then, reset the index for further data synthesis.

After doing so, we checked the merged dataset for overview of data (shown below):



```python
# Explore the training data
df.head()
```

✓ 0.8s

	index	id	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id	...	device_type	device_conn_type	C14	C15	C16	C17	C18	C19	C20	C21
0	8165300	7.902527e+18	0	14102221	1005	1	a7853007	7e091613	f028772b	ecad2386	...	1	0	9478	320	50	906	3	1451	100156	61
1	30286524	1.400962e+19	0	14102911	1002	0	5b787406	f3ca2e42	50e219e0	ecad2386	...	0	0	23438	320	50	2684	2	1327	-1	52
2	14989769	2.411564e+18	0	14102504	1005	0	85f751fd	c4e18dd6	50e219e0	e986f7ef	...	1	2	20395	320	50	2340	3	41	100111	159
3	30874747	5.087266e+18	0	14102914	1005	0	85f751fd	c4e18dd6	50e219e0	92f5800b	...	1	2	21191	320	50	2424	1	175	100189	71
4	30748074	1.107625e+19	0	14102914	1010	1	85f751fd	c4e18dd6	50e219e0	04f2be5f	...	4	3	23551	320	50	2698	0	163	100084	212

5 rows × 25 columns

After checking the missing values, we found that the dataset contains no missing value at all. So, we knew we were ready for the feature engineering process.

Feature Exploration, Selection and Engineering

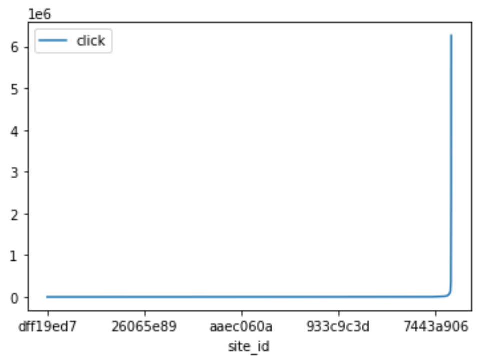
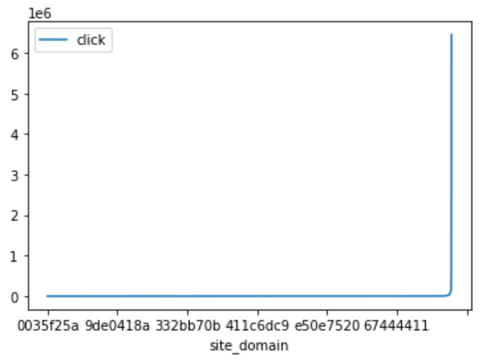
Our manipulation structure for each categorical variable follows the following steps:

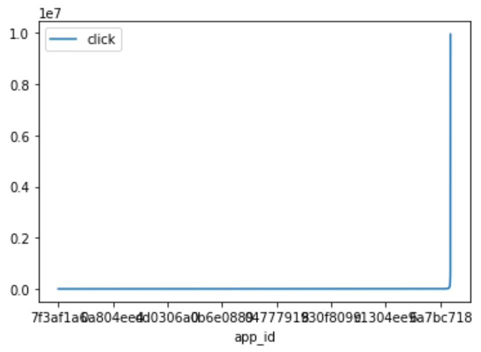
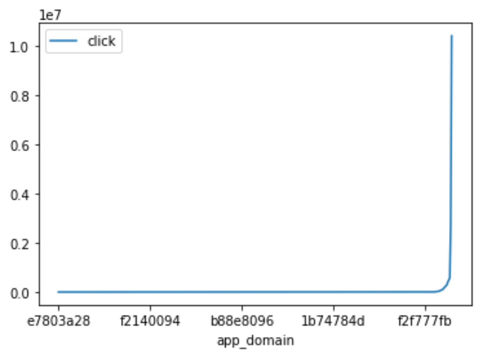
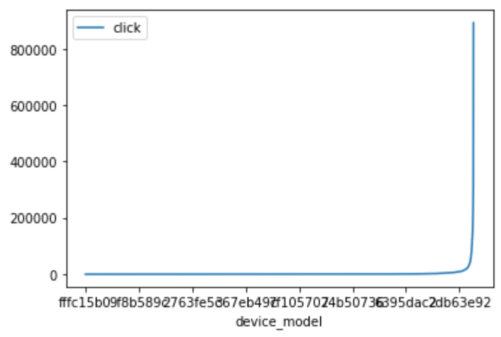
1) Directly delete it: Based on our domain knowledge. For example, the webpage view id (the first column) should be directly deleted since it does not contain any informative value and may be misleading in the model training process. So, we just delete this column.
2) Check the number of categories: We used pandas.DataFrame.nunique() to check the number of categories for each feature. If it is smaller than 30, we do label encoding. However, if it is larger than 30, do manipulation before trying to encode it.
3) Check distributions: With so many categories, we checked the frequency distribution of the features. We use groupby, count, and sort_values to get the frequency of different categories ordered by the frequency amount. From this, we found that most of the distributions are highly imbalanced; that means that only a small number of categories occupies values for most rows. Then, we set all the “minority categories” as *“others”* while keeping the “majority categories” (with a specific threshold; for

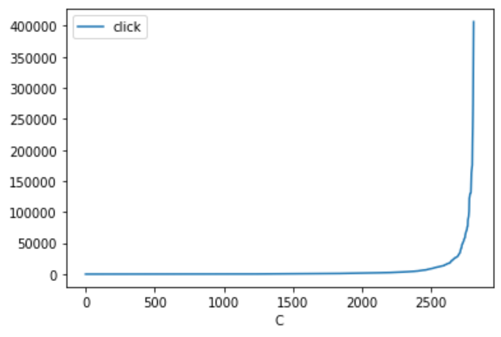
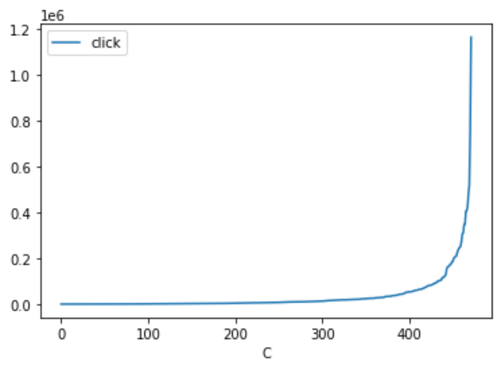
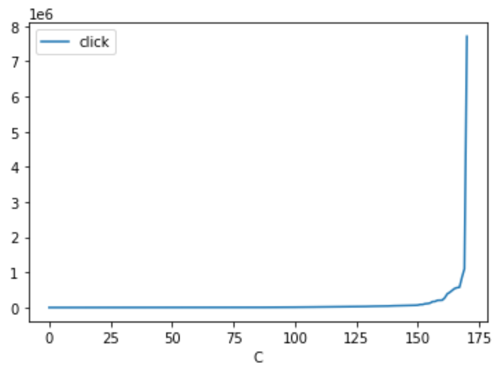
example, set the categories with 90/80/50 total view/rows as “major” and keep them, and set the other categories as “others”), and do encoding in the following step, to significantly reduce the number of categories.

4) For feature engineering part, we used two encoding approaches: label encoding (for # of categories <= 30) and baseN encoding (for # of categories > 30). For *‘hour’*, we separate the value of day and hour to make the column more informative (since this modeling is basically not a time series-oriented task, we did not make further effort on this column).

Please see the following table to explore how we did data manipulation for each feature in detail:

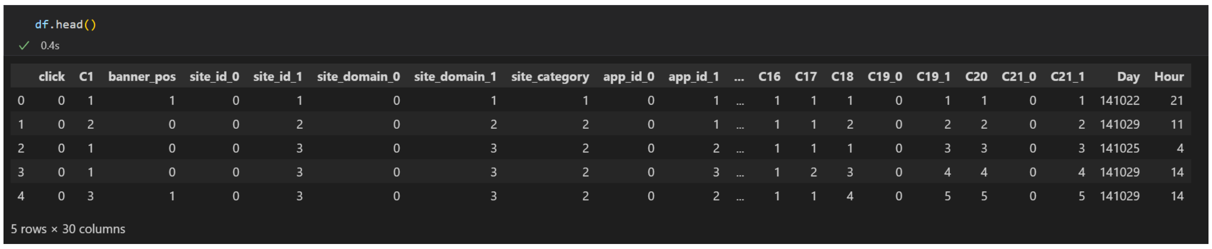
Feature Name	**Number of Categories**	**Data Distribution**	**Manipulation & Engineering Approaches**
id	/	The index is not informative at all	Delete the column
hour	/	The data itself is a string-like timestamp and not informative enough	Split it into two columns: Day: YYYYMMDD Hour: 1~24
C1	7	/	Label Encoding
site_id	4168		Threshold: 90% # of categories after manipulation: 75 Base 10 Encoding
site_domain	5785		Threshold: 90% # of categories after manipulation: 48 Base 10 Encoding
site_cate	25	/	Label Encoding

gory			
app_id	7187		Threshold: 0.9 # of categories after manipulation: 33 Base 10 Encoding
app_domain	430		Threshold: 0.9 # of categories after manipulation: 6 Label Encoding
app_category	33	/	Base 10 Encoding
device_id	1296455	The data has too many categories and is not informative	Delete the column
device_ip	3912984	The data has too many categories and is not informative	Delete the column
device_model	7265		Threshold: 0.5 # of categories after manipulation: 46 Base 10 Encoding
device_type	/	Already a well-defined numerically encoded column	/
device_conn_type	/	Already a well-defined numerically encoded column	/

C14	2808		Threshold: 0.5 # of categories after manipulation: 59 Base 10 Encoding
C15	8	/	Label Encoding
C16	9	/	Label Encoding
C17	472		Threshold: 0.5 # of categories after manipulation: 27 Label Encoding
C18	4	/	Label Encoding
C19	68	/	Base 10 Encoding
C20	171		Threshold: 0.9 # of categories after manipulation: 24 Label Encoding
C21	62	/	Base 10 Encoding

After doing the feature engineering process, the DataFrame contains 30 columns, with 29 features and 1 target (*Figure 1*). We used all the columns to draw a correlation heatmap to have a basic idea about the relationship between the target and each feature variable, respectively (*Figure 2*). We observed that every feature variable shows a weak relationship with *click*, so the heatmap does not give us a lot of information.

Figure 1.

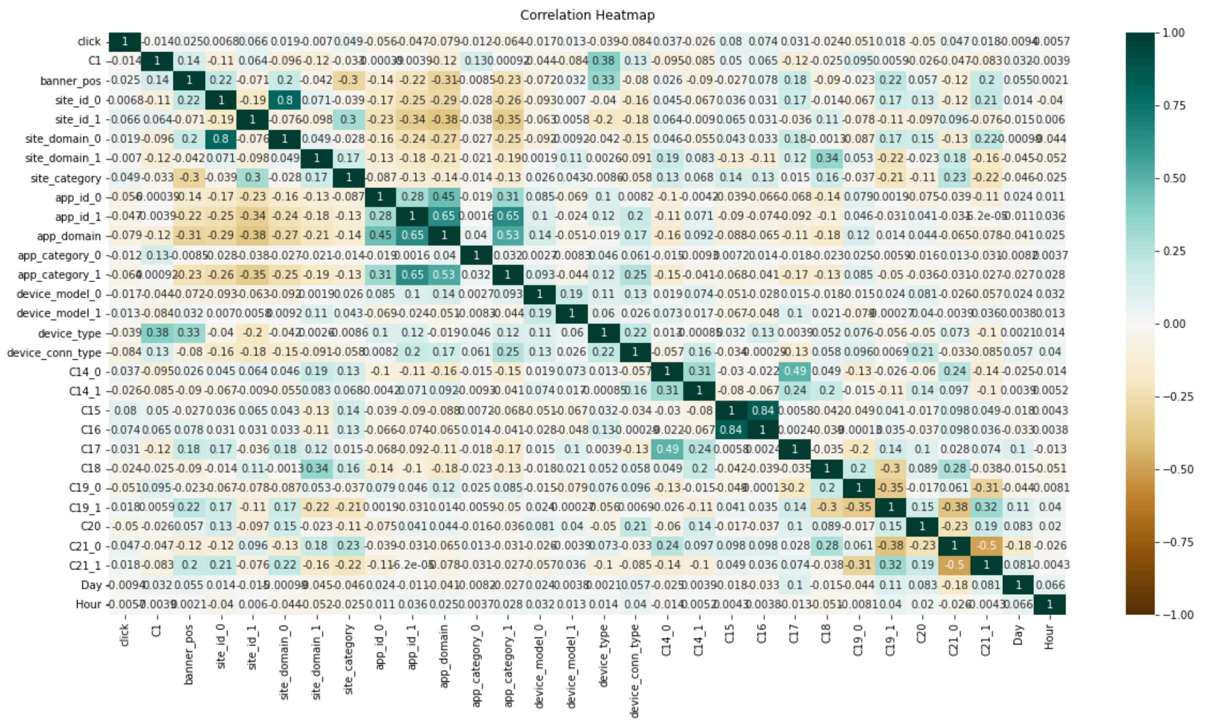


```
df.head()
✓ 0.4s

   click  C1  banner_pos  site_id_0  site_id_1  site_domain_0  site_domain_1  site_category  app_id_0  app_id_1  ...  C16  C17  C18  C19_0  C19_1  C20  C21_0  C21_1     Day  Hour
0      0   1           1          0          1              0              1              1         0         1  ...    1    1    1      0      1    1      0      1  141022    21
1      0   2           0          0          2              0              2              2         0         1  ...    1    1    2      0      2    2      0      2  141029    11
2      0   1           0          0          3              0              3              2         0         2  ...    1    1    1      0      3    3      0      3  141025     4
3      0   1           0          0          3              0              3              2         0         3  ...    1    2    3      0      4    4      0      4  141029    14
4      0   3           1          0          3              0              3              2         0         2  ...    1    1    4      0      5    5      0      5  141029    14

5 rows × 30 columns
```

Figure 2.



Modeling

In this part, we try to select five "candidate" classification models to then later on select the best performing model. The model we considered were the following:

1) Logistic Regression (baseline)
2) Random Forest
3) XGBoost
4) LightGBM
5) Neural Network

Logistic Regression is a numerical-like model predicting probability of the target; since it doesn't work well for large datasets, we just set it as a baseline model. Random Forest, XGBoost, and LightGBM are all tree-based models using bagging and bootstrapping to stabilize the model performance. Neural Network is another advanced deep learning approach, which was covered in detail in our machine learning class.

Initially Test the Models

After splitting the data back to training and test dataset, we then split the training dataset to training and validation dataset (validation dataset is referred to as X_test and y_test in our code) by 80%/20%. Then we created classifiers and set default hyperparameters, fit the training model and use the validation dataset to generate the log loss score to get an initial idea about the performance. The performance is shown below:

Model	**Parameters**	**Log_Loss Score**

Logistic Regression (baseline)	random_state=42
	penalty=”L2”
		0.4547

Random Forest		0.4061

XGBoost	objective=”binary:logistic”
	nthread=4
	seed=42
		0.4011

LightGBM	/	0.4049

Keras Neural Network	Layers=3
	units=512, 512, 1
	activation=’relu’, ‘relu’, ‘sigmoid’
	epoch=9
	batch_size=4096
		0.4024

The table indicates that XGBoost is the best model fitting the validation dataset, while RF, LightGBM and NN perform slightly worse than it. All the 4 models perform significantly better than the benchmark model, logistic regression.
Obviously, we cannot decide which model we should pick at this stage. Parameter tuning is required for better performance of the model.

Hyperparameter Tuning

For hyperparameter tuning, we use TuneGridSearchCV (a faster grid search method using bayesian optimization) to do hyperparameter tuning for all the four models. We did this in hopes that it would bring down the time spent to get to the optimal set of parameters and that it brings better generalization performance on the test set. There are three main steps we followed for this process:

1) Import the TuneGridSearchCV function from tune_sklearn package
2) Get the parameter value list with best performing log loss score
3) Apply the optimal parameter values to the models and get the new log_loss score for validation dataset, choose the best performing one and use it to predict the data from test dataset

The result table of this process is shown below:

Model	**Parameters**	**Log_Loss Score**

Random Forest	random_state=42
	n_estimator=100
	criterion=”entropy”
	max_features=”sqrt”
	min_samples_split=300
	max_depth=10
		0.3990

XGBoost	n_estimator=200
	max_depth=9
	learning_rate=0.01
		0.3998

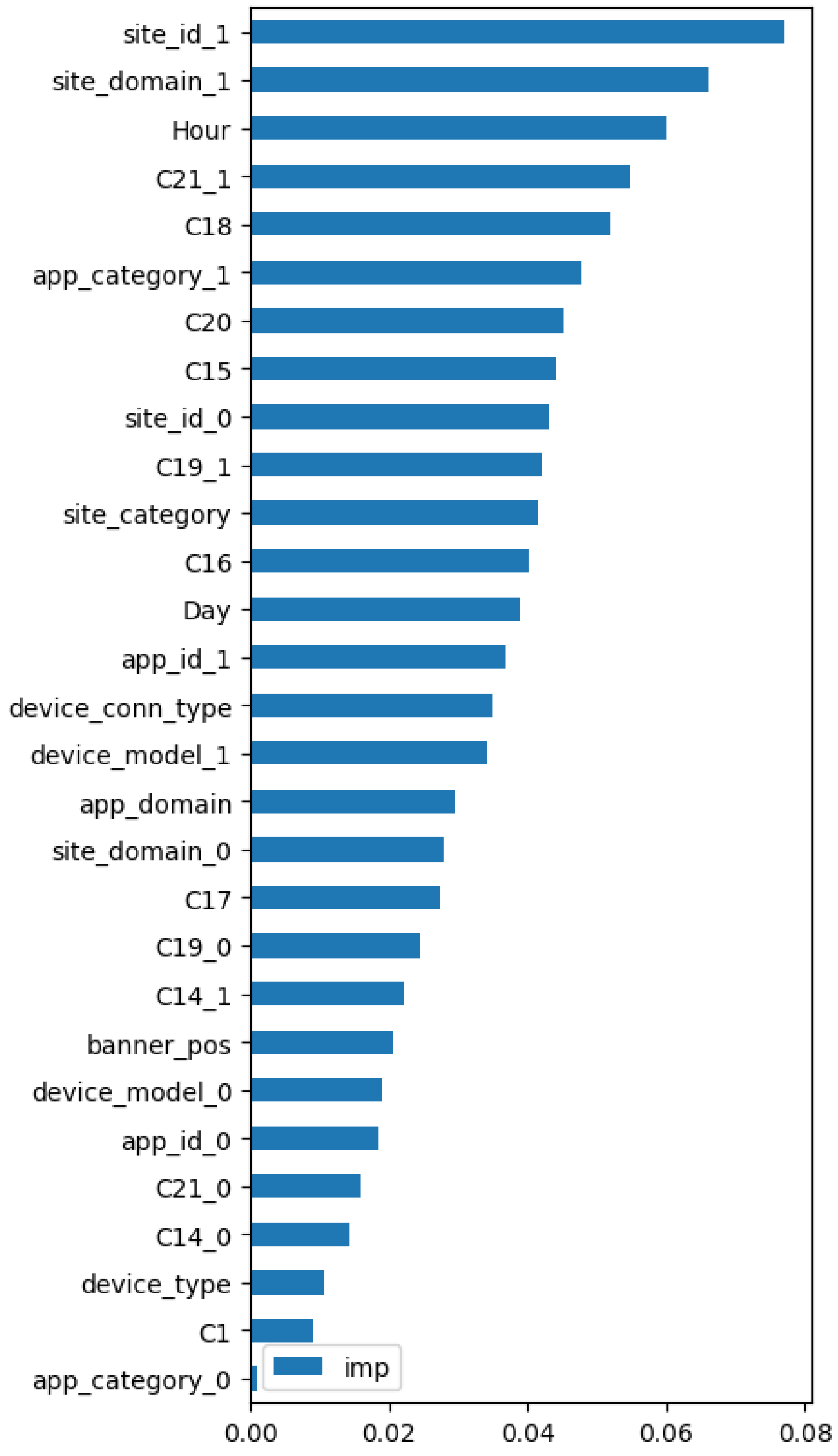
LightGBM	num_leaves=100
	min_child_samples=10
	max_depth=10
	learning_rate=0.05
	reg_alpha=0
		0.4012

Based on the score results, we decide to choose the random forest model as our optimal model, applying it to our test dataset and come up with the prediction result, which is stored in “ProjectSubmission-Team12.csv”.

Reflections on the Final Project

Feature Importance - Which factors influence the prediction result more?

Although not required for the project, we were especially curious about the feature importance. The motivation comes from two perspectives: on the one hand, in the “real business world” feature importance can provide the strategy decision makers a statistical-based view on which factors they should focus on to most improve their products and services. On the other hand, the advanced tree-based modeling approaches, Random Forest and XGBoost, directly provide you functions to do so. To satisfy our curiosity, we decided to use Random Forest feature importance, and the importance ranking plot from this process is shown below:



As we can see from the plot, *site_id, site_domain, Hour* and *C21* are the most influential factors, and *C18, app_category* and *C20* are also decisive for predicting the model. In real business cases, we could use these observations to make targeting suggestions on business development.

Highlights of the Project

We used a clever method on feature engineering. After understanding that basically all the features are categorical and that many variables have a lot of categories, we realized that we should explore the frequency distribution (i.e. the skewness) of the data. Intuitively, we believe that the distribution should be highly skewed since most web page views should concentrate on several major websites/apps - Pareto Principles. The distribution plots confirmed our assumption. Then, we tried to craft the data structure by separating the "major" and "minor" categories and help us to do encoding.

Secondly, we applied many different models with advanced algorithms to help our predictions perform better. XGBoost and LightGBM are both gradient boosting models with optimized algorithms. Consequently, the results also show those advanced models perform much better than the benchmark model.

Thirdly, we put a lot of effort into crafting our models by hyperparameter tuning. Several different approaches (GridSearchCV, TuneGridSearchCV, Bayesian Optimization, and many self-defined functions) were applied, and compared with the initial default models, the tuned models exactly provide significantly better performance in log loss.

Restrictions and Further Improvements of the Project

In this project, our major difficulties come from the large data size; 40 million data requires a tremendous amount of calculation and RAM for our computers, so we spent a significant amount of time on model sampling. We believe that if we have equipment with better capacities, we could apply more data to give a better performance.
Moreover, we did not focus on commissioning the Neural Network model, since the default model does not show outperforming ability.

To conclude, this project helped us reflect on what we have learned during this semester. It helped us acquire a deeper understanding of different models and their performances, as well as how to deal with messy data and align it to a machine learning format.