CS5228 Final Project Report

Pan Xiang(A0213836E) Wei Yihao(A0105593E) Zhu Qinglei(A0198751N)
Kaggle Team: **Exceed50K**

1. Introduction

This report summarizes our findings for CS5228 2020/sem1 Final Project - to predict if income exceeds 50k. It includes the details of data exploration, data preprocessing, model selection and training, and model evaluation using F1 Score.
Project details can be found on Kaggle at https://www.kaggle.com/c/cs5228/overview
Our codebase can be found in our Github: https://github.com/Xiang-Pan/NUS_CS5228_KDD_Project

2. Data Exploration

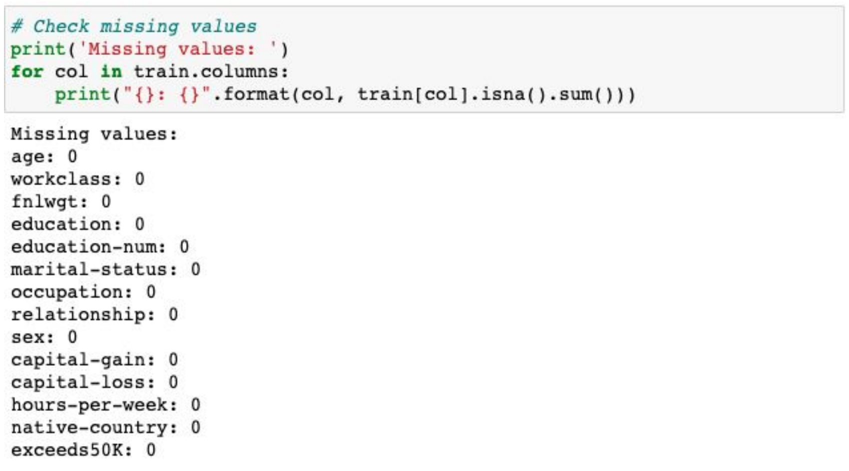
The given data is a slight modification of the existing Adult dataset. This dataset was extracted by Barry Becker from the 1994 Census database. The original dataset split is 70% for training and 30% for testing. In this project, the dataset is split by 50-50.

In this report, we will refer to non-label columns as feature columns. Both *train.csv* and *test.csv* have 13 feature columns, while *train.csv* has 1 more binary column (exceeds50K) to be used as training labels (0 means False, 1 means True).

Among the 13 feature columns, there are 6 continuous numerical features (age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week), and 7 categorical features (workclass, education, marital-status, occupation, relationship, sex, native-country).

2.1 Missing Values

We used pandas isna() and sum() functions to verify that there is no missing value for any column in both *train.csv* and *test.csv*. Therefore, there is no need to handle missing value in this project.



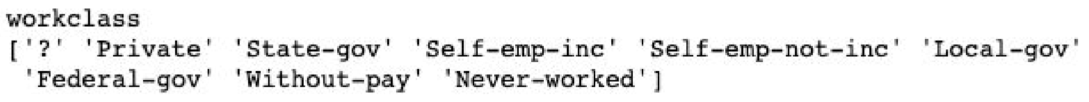
```python
# Check missing values
print('Missing values: ')
for col in train.columns:
    print("{}: {}".format(col, train[col].isna().sum()))
```

```
Missing values:
age: 0
workclass: 0
fnlwgt: 0
education: 0
education-num: 0
marital-status: 0
occupation: 0
relationship: 0
sex: 0
capital-gain: 0
capital-loss: 0
hours-per-week: 0
native-country: 0
exceeds50K: 0
```

Figure 1. Missing Value Check for train.csv

2.2 Invalid Values

Analyzing unique values using pandas *unique()*[1] function helps us to obtain all possible values for each column. Categorical columns like workclass, occupation, and native-country have invalid data represented as '?' as shown in *Figure 2*. Therefore, we need to find a way to represent these invalid values in preprocessing steps later.



```
workclass
['?' 'Private' 'State-gov' 'Self-emp-inc' 'Self-emp-not-inc' 'Local-gov'
 'Federal-gov' 'Without-pay' 'Never-worked']
```

2.3 Visualization on Data Distribution

In order to evaluate how each feature is related to the result, we can analyze the data distribution of each column and compare to the results. For numerical features, the data distribution can be visualized using *boxplot()*[2]. We used each of the 6 numerical features plots against the result column *exceeds50k* as shown in *Figure 3*. The difference can be compared in data distribution for people who have income exceeds 50k and not exceeds 50k side by side.

Observing from the boxplots below, we can conclude that:

1. Most of the people who have income exceed 50k are in the age of 37 to 51, studied 10 to 13 years, has no capital gain or loss, and works 40 to 50 hours per week
2. There is an overlapping range (from 37 to 46) of age for both people who have income exceeds 50k and not exceeds 50k, therefore it would relatively difficult to predict the result using age range in 37 to 46
3. Fnlwgt[3], capital gain, and capital loss shows the little difference in data distribution for people who have income exceeds 50k and not exceeds 50k, therefore fnlwgt, capital gain, and capital loss are not very relevant to the result
4. Education-num and hours-per-week are very good features for predicting result as there is limited overlapping in feature values for a different result

[1] Pandas unique(): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.unique.html
[2] Boxplot: https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.boxplot.html
[3] Fnlwgt: The continuous variable fnlwgt represents final weight, which is the number of units in the target population that the responding unit represents

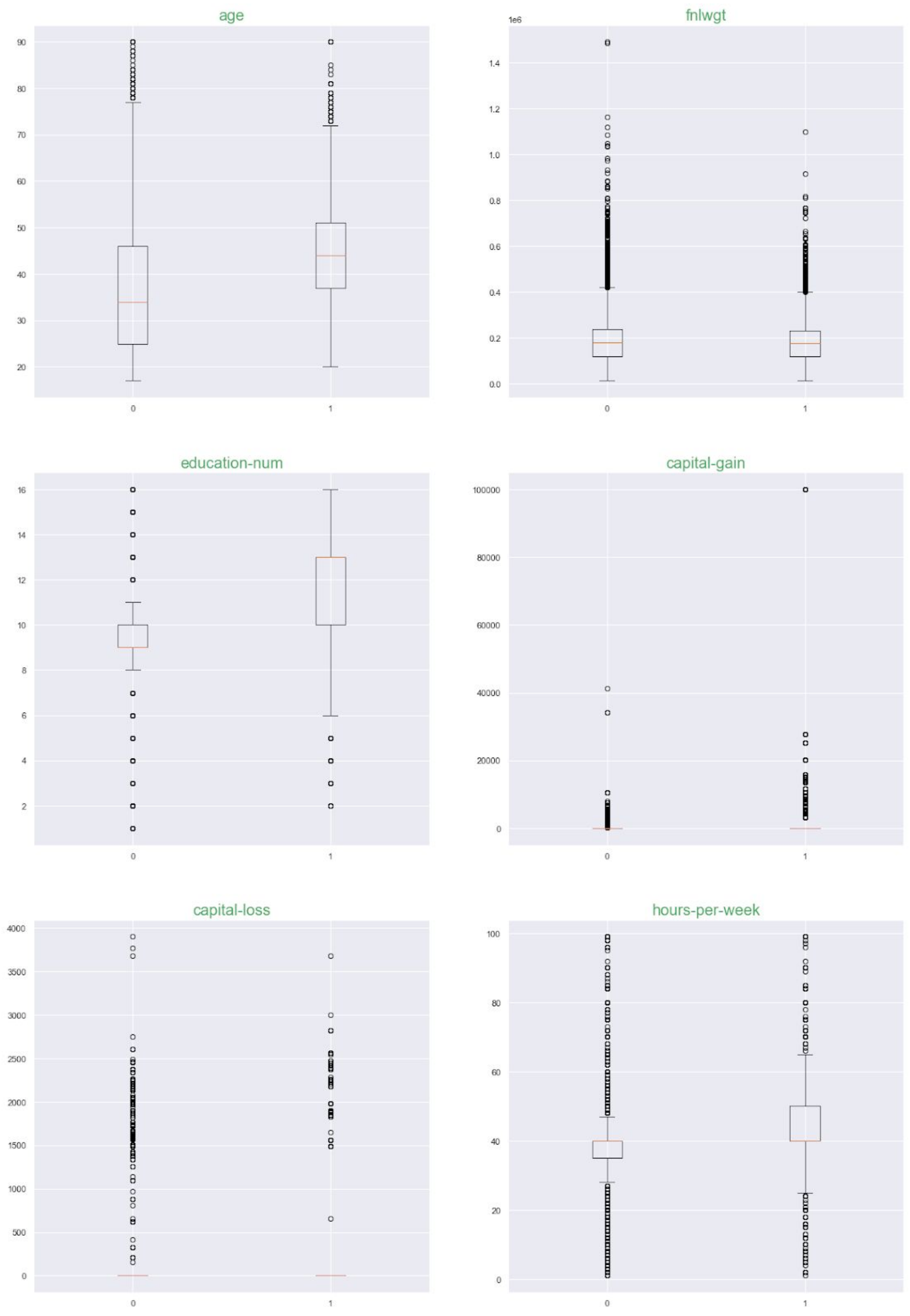


Figure 2. Boxplots for Numerical Features in relation to exceeds50k

For categorical features, we can visualize using barplot[4] as shown in *Figure 4* below:

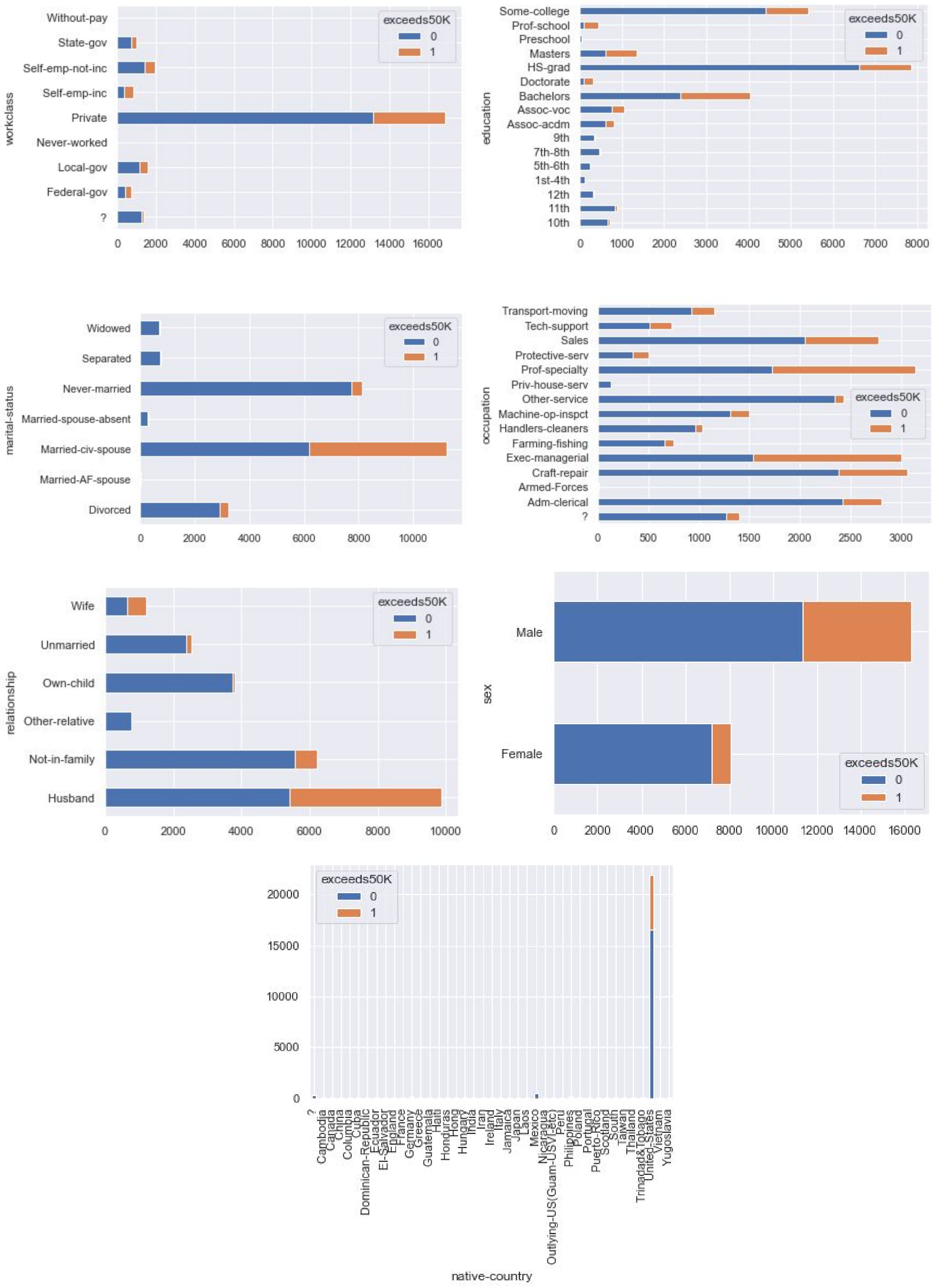


Figure 4. Barplot for Categorical Features

[4] Barplot: https://www.w3resource.com/pandas/dataframe/dataframe-plot-barh.php

Observed from the bar plots that most of the people who have income exceeds 50k are:

1. private working class
2. Bachelors or above in education level
3. married males
4. sales, professional or executional managers
5. mostly are in the US

In addition to the relationship between each feature and income exceeds 50k, we found that the dataset is unbalanced such that positive results (*exceeds50k*=1) are only around a quarter of the total dataset.

3. Data Preprocessing

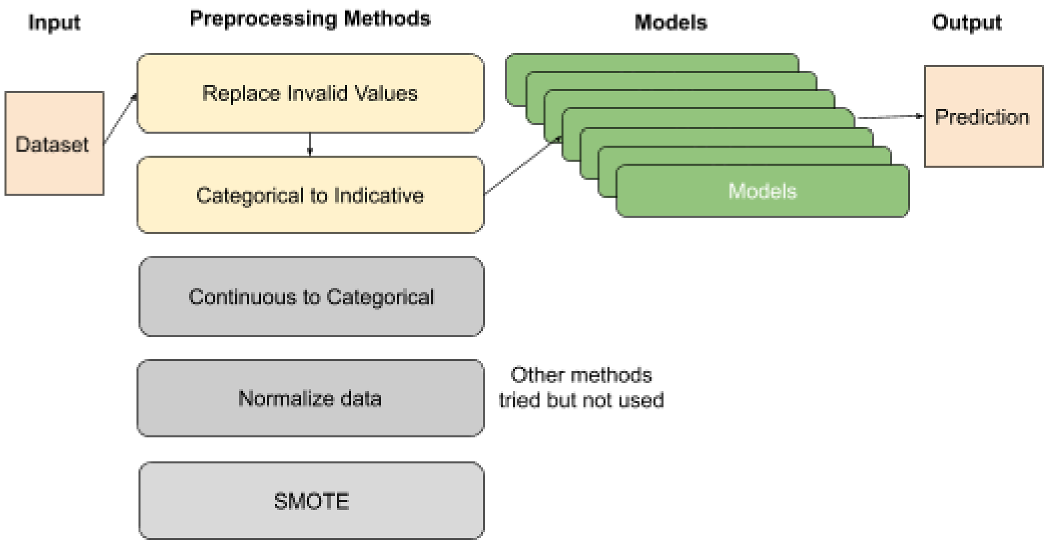


Figure 5. Data Preprocessing Steps

3.1 Handle Invalid Values

As observed from section 2.1 that there is no missing value in *train.csv* and invalid values represented by '?' are only present in categorical features, we can easily consider '?' as a category for that column. Therefore, no replacement for '?' is required.

3.2 Categorical to Indicative

To make the training dataset suitable for training calculation, we need to encode all discrete non-numerical values into numerical. This is achieved by using pandas get_dummies()[5] function to generate one-hot vector for each categorical value.

3.3 Other techniques we have explored

We have also explored the following preprocessing techniques. However, it is not included into our final procedure as it does not improve the performance.

[5] Pandas get_dummies(): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html

3.3.1 Continuous to Categorical

There are 6 continuous features (*age, fnlwgt, education-num, capital-gain, capital-loss, hours-per-week*), and we can easily convert them into discrete values using pandas *qcut()*[6] function.

3.3.2 Normalization

In order to make the training more accurate and do not dilute by large scale attributes, we did a normalization step on datasets. We first drop the result column in training dataset, then applied sklearn *StandardScaler()*[7] to the data.

3.3.3 SMOTE

Since we observed that the *exceeds50K* labels are highly imbalanced which may lead to overfitting on the majority class, we also tried to use an advanced oversampling algorithm called SMOTE to mitigate this issue.

Now we have the fully preprocessing training dataset for model training.

4. Approaches

4.1 Model Selection

There are many models for solving classification problems. In order to select best fit model for this project, we tried 11 different models from *scikit-learn* and other libraries: *KNeighborsClassifier, LogisticRegression, DecisionTreeClassifier, RandomForestClassifier, GradientBoostingClassifier, GaussianNB, SVC, XGBClassifier, HistGradientBoost, LGBMClassifier, CatBoost*. For model selection purposes, we used default parameters of each model to obtain validation scores (F1 Score) as the evaluation metric.

The validation scores obtained for all 11 models using default parameters is shown in *Table 1* below:

Model	**Validation Score**
GaussianNB	0.7669
SVC	0.7267
KNeighborsClassifier	0.7579
LogisticRegression	0.7581
DecisionTreeClassifier	0.815
RandomForestClassifier	0.85
GradientBoostingClassifier	0.8591
XGBoost	0.8723
HistGradientBoost	0.872

[6] Pandas qcut(): https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.qcut.html
[7] StandardScaler: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

LightGBM	**0.8742**
CatBoost	**0.8758**

Table 1. Validation Scores on 11 Models using GridSearchCV

We can observe from the result that Random Forest and Gradient Boosting algorithms generally have better performance than the other models as their validation score is equal or greater than 85%. Specifically speaking, some variations of Gradient Boosting models such as LightGBM and CatBoost have even better performance compared to the standard gradient boosting algorithm in sklearn. Therefore, we decided to zoom in to Gradient Boosting models for hyper-parameter optimization.

4.2 Hyper-parameters Optimization with GridSearchCV

The GridSearchCV parameters we used for model selection is as follows:

```
cv=5
scoring='f1_weighted’
refit=True
```

As there is no given validation dataset split for offline validation, we split the dataset to training and validation sets to evaluate model performance. Since the size of *train.csv* is not very big, default cv=5 would be adequate to cross validate the model performance. The goal of this project is to optimize the weighted f1 score, instead of using accuracy as our evaluation metric, we set scoring as *f1_weighted* to better reflect model performance. We also set refit to True in order to get the best model after the searching process is completed, and use it directly for predicting test dataset results.

Grid search with cross validation may take a long time if we consider a large set of hyper-parameters, we decided to employ a step-by-step ‘greedy’ grid search approach as follows:

1. Fix learning rate to a large number around 0.1~0.3 and *n_estimators* to a small number below 100
2. Select a few most important parameters such *max_depth*, *subsample* to optimize using grid search
3. Fix those optimized parameters and optimize the rest in the same way
4. Decrease learning rate and increase n-estimators until the cross validation score does not improve

In this way, we are able to tune all parameters in a fast manner instead of searching for all parameters at the same time.

Among all types of Gradient Boost Models, we selected 2 models for fine-tuning hyper-parameters and included test scores in Kaggle Leaderboard as another evaluation metrics for the model performance.

4.2.1 Gradient Boosting Classifier (GBM)[8]

After applying the step-by-step grid search described above, we obtained a few sets of parameters which can be regarded as ‘local maximum’ as shown in *Table 2* below.

[8] Gradient Boosting Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

Gradient Boosting Classifier	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**	**Trial 6**
loss	deviance	exponential	exponential	exponential	exponential	exponential
learning_rate	0.005	0.01	0.035	0.01	0.005	0.03
n_estimators	1600	2000	300	1000	2000	300
max_depth	13	10	100	100	100	120
min_samples_split	900	1400	4500	4500	4500	4500
max_features	16	12	90	90	90	None
subsample	1	0.75	1	1	0.95	1
min_samples_leaf	7	10	1	1	20	1
Validation Score	0.8692	0.8707	0.8757	0.8757	0.8751	0.8726
Kaggle Score	0.87354	0.87379	0.87428	0.87403	0.87469	**0.87477**

Table 2. Gradient Boosting Classifier Results

4.2.2 CatBoost[9]

Catboost is another implementation of Gradient Boosting algorithm developed by Yandex researchers. It has some special handling techniques on categorical features, thus it is more suitable for our dataset and performs better than the standard GBM. For the hyper-parameters optimization process, we used the same way as described above.

The *Figure 6* below shows that the F1 score for the training set goes up continuously as the number of iterations increase. However, the validation score reaches a bottleneck at around 1400 iteration. This is an indication of overfitting, so we should fix our iteration number to 1500.

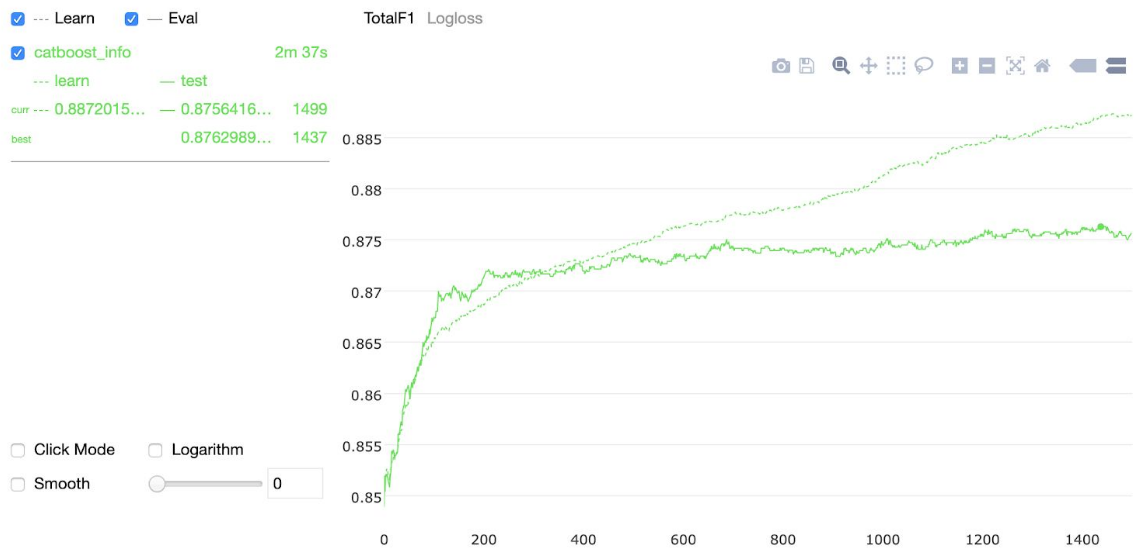


Figure 6. F1 Score versus iteration number

[9] Catboost: a machine learning algorithm that uses gradient boosting on decision trees. It is available as an open source library, https://catboost.ai/

Similarly, by doing grid search with cross validation, we found the best set of parameters by choosing the highest validation score. Note that it also has a relatively low training score, which indicates less overfitting.

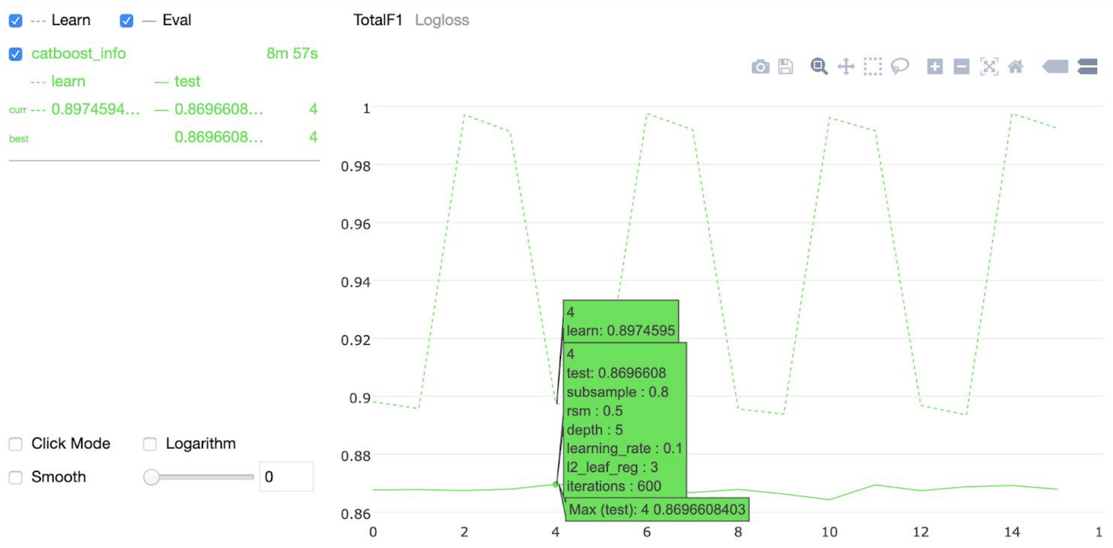


Figure 7. F1 Score for different sets of parameters

Below is the CatBoost model performance using different sets of parameters.

Catboost Classifier	**Trial 1**	**Trial 2**	**Trial 3**	**Trial 4**	**Trial 5**
cat_features	All	Non-numerical features	All	All	All
learning_rate	auto	auto	0.03	0.03	0.03
iterations	1000	1000	1500	1500	1500
depth	6	6	6	5	5
threshold	0.5	0.5	0.5	0.55	0.5
Validation Score	0.8758	0.8742	0.8771	0.8778	0.8775
Kaggle Score	0.87665	0.87624	0.87665	0.87657	**0.87796**

Table 3. Catboost Classifier Results

From the control experiment on Trial 1 and 2, we found that *cat_features=All* gives better performance than just using categorical features, therefore we continue to apply *cat_features=All* to all the following trials. From Trial 1 and 3, we can observe that changing the *learning_rate* from *auto* to 0.03 and increasing *iterations* to 1500 improves the validation score while Kaggle Score keeps the same. We also explored using a different threshold value (0.55 instead of 0.5), but this does not improve the actual performance.

4.2.3 Ensemble

To obtain a better result, we also tried to combine the advantages of different Gradient Boosting models. The approach we used is called majority voting.

There are 3 models we used for result ensembling: *CatBoost, LightGBM, and GBM*. We used each of the 3 models with the best configurations to predict *test.csv*, and then merged the obtained results together. We have explored different variants of the majority voting method and found the following method achieved both the best validation score and Kaggle score. We first combined the predicted results from *LightGBM* and *GBM* by using *AND* for each prediction and saving the result as *temp_1*. This helps us to obtain the double positive results predicted in both models. Since CatBoost has the best performance among all Gradient Boosting models, we then combined the predicted results from CatBoost with *temp_1* by using *OR* for each row, this is to add on any missing positive result predicted from CatBoost. The ensemble method can be concluded as:

CatBoost and (LightGBM or GBM).

The result obtained from the ensembled model is shown in *Table 4* below, it gives us both the best validation and testing score:

Ensembled Model	**Validation Score**	**Kaggle Score**
CatBoost		
LightGBM		
GBM	**0.8784**	**0.87895**

Table 4. Ensembled Model Result

5. Conclusion

In this competition, we explored the data distribution, and based on the data features and distribution, we used typical data preprocessing methods to format the data. Then we tried various models to find out which model can fit the data well. We chose the tree-based ensemble models and fine-tuned parameters for them using grid search with cross validation. Technically, using a weighted f1 score as an evaluation metric helped us to find the best parameters. Finally, we proposed an ensemble method to combine the best results from different models to make the prediction and achieved the top 1 ranking (at the time of writing) on Kaggle leaderboard .