

Pattern Recognition Project Report

final report

Group 3

Minjeong Kim (2391007)

Suahn Lee (2391016)

Yoonji Lee (2391017)

Supervisor : Junhyug Noh

Department of Artificial Intelligence
Ewha Womans University

Table of Contents

1. Abstract	3
2. Data Exploration	3
a. Data Visualization	3
b. Data Examination	5
3. Comparison of Data Preprocessing Changes with Proposal	6
4. Exploring various approaches and processes for sampling and model selection	8
a. Sampling	8
• Oversampling	8
• Oversampling & Undersampling	8
• Undersampling	9
b. Model Selection	10
5. Finalized model (includes model training and validation)	12
6. Troubleshooting Processes & Remaining Problems	13
7. References	14

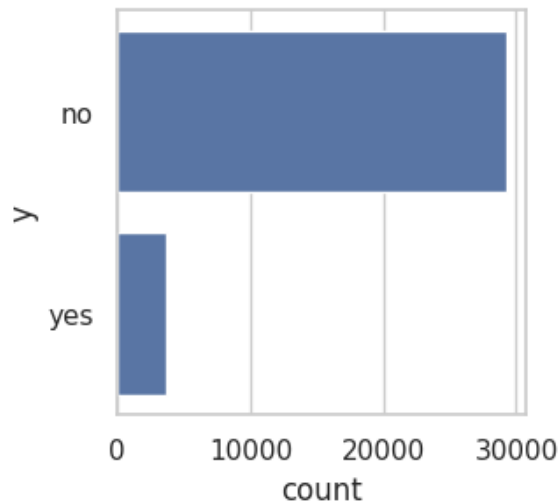
1. Abstract

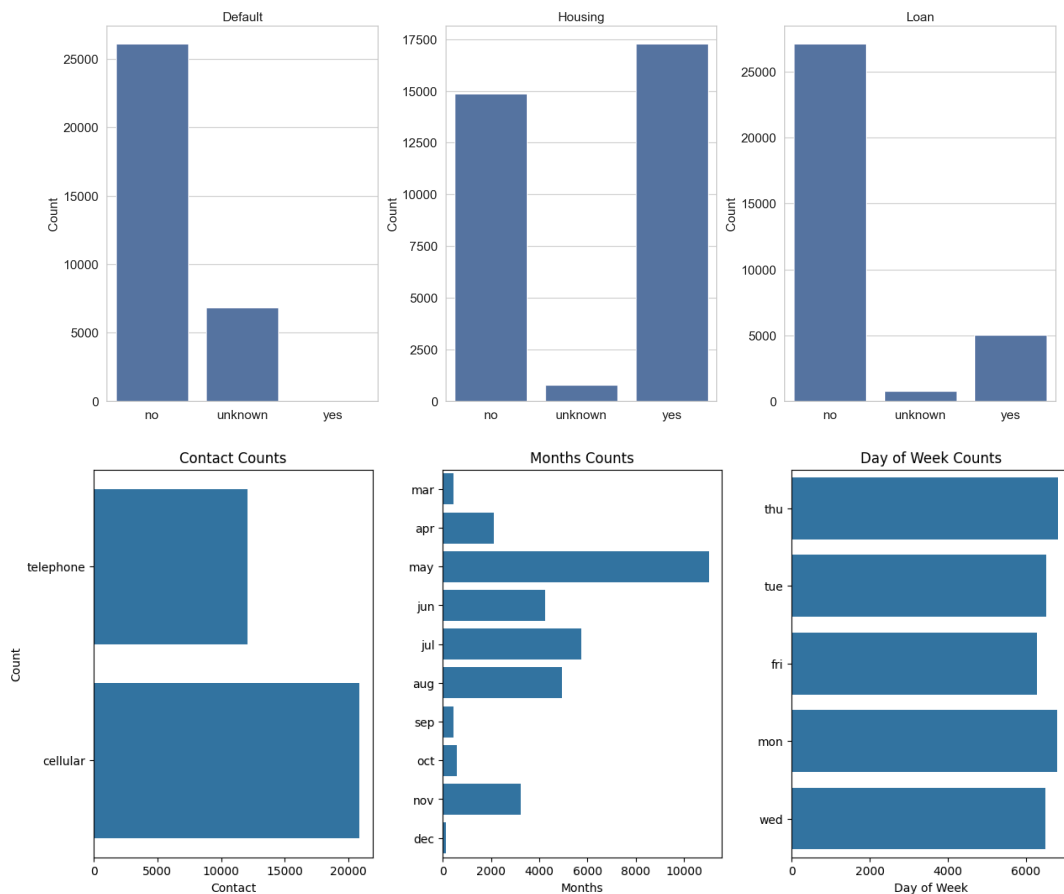
This report details our application of CatBoost and various preprocessing techniques to improve our predictive model's F1 score, achieving 0.648 on the training set (along with other criteria - Accuracy: 0.894, ROC AUC: 0.857). Initially, we modified the 'job' column based on age and incorporated economic indicators but faced multicollinearity issues, leading to the removal of the 'euribor3m' and 'nr.employed' columns. To address the class imbalance, we tested various sampling methods and found NCR undersampling to be the most effective. Using PyCaret for preliminary model selection, we evaluated multiple algorithms and chose CatBoost for its handling of categorical data and consistent performance. Our final model included comprehensive preprocessing, NCR undersampling, grid search for parameter optimization, and k-fold cross-validation.

2. Data Exploration

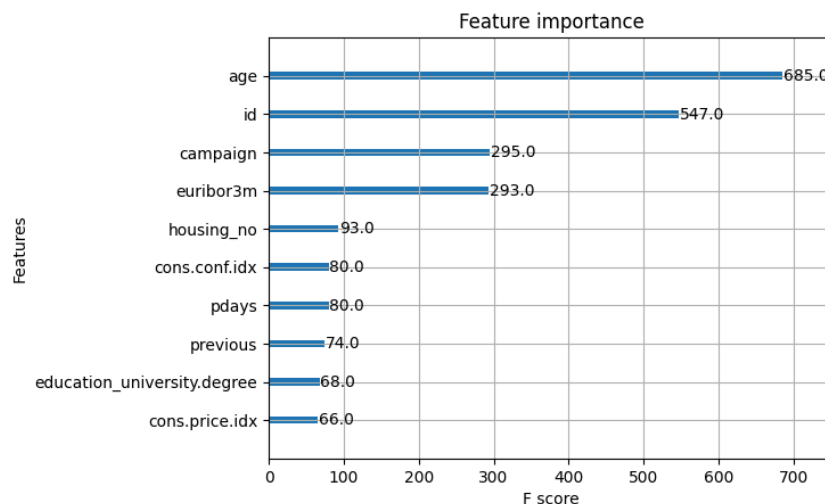
a. Data Visualization

We noticed that the given data set was highly imbalanced from the beginning. As shown in the graph below, a high proportion (around 90%) of the 'y' values are 'no'. When training a model with such imbalanced data, there is a high risk that the model will be unable to properly classify values belonging to the minority class. Therefore, we put a lot of effort into EDA and data preprocessing.





In the figures above, the 'default' column has very few 'yes' values (less than 100) and a relatively high percentage of 'unknown' values. The 'housing' column has about 2000 more 'yes' values than 'no' values, and again, there are missing values. The 'loan' column appears to have more than 5 times as many 'no' values as 'yes' values. In the 'contact' column, cellular phones are about 7,000 times more frequent than telephones. In the 'month' column, 'may' is much more frequent than others. In the 'day of week' column, all values are about equal.



The above graph illustrates the top 10 most important features, using XGBoost's built-in `plot_importance`. It is consistent with the conventional wisdom that age is important. Furthermore, it shows the importance of 'pdays', with all missing values being 999.

b. Data Examination

We used `df.describe()` for data examination, both train and test datasets. This was done on the advice of a professor to compare the distributions of the test and train sets.

```
[12] df_train.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id	32950.0	20569.615569	11895.520420	1.000	10258.250	20571.000	30846.750	41188.000
age	32950.0	40.023703	10.401749	17.000	32.000	38.000	47.000	98.000
campaign	32950.0	2.567830	2.766994	1.000	1.000	2.000	3.000	56.000
pdays	32950.0	962.415964	187.054556	0.000	999.000	999.000	999.000	999.000
previous	32950.0	0.172838	0.498098	0.000	0.000	0.000	0.000	7.000
emp.var.rate	32950.0	0.083129	1.571951	-3.400	-1.800	1.100	1.400	1.400
cons.price.idx	32950.0	93.576610	0.578725	92.201	93.075	93.749	93.994	94.767
cons.conf.idx	32950.0	-40.500091	4.632363	-50.800	-42.700	-41.800	-36.400	-26.900
euribor3m	32950.0	3.622516	1.734791	0.634	1.344	4.857	4.961	5.045
nr.employed	32950.0	5167.036495	72.250873	4963.600	5099.100	5191.000	5228.100	5228.100

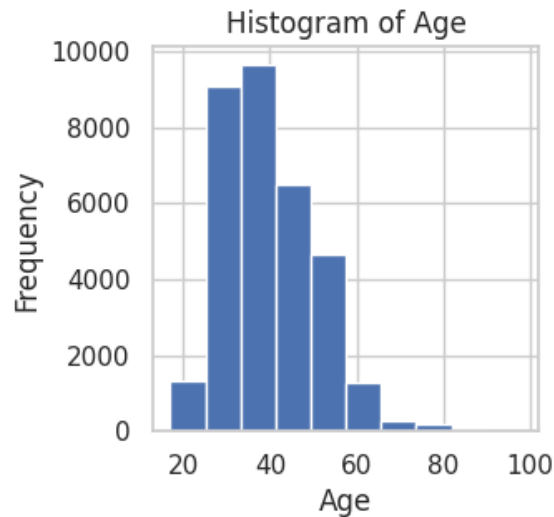
```
[13] df_test.describe().T
```

	count	mean	std	min	25%	50%	75%	max
id	8238.0	20694.031682	11868.573076	4.000	10457.750	20694.000	31056.250	41181.000
age	8238.0	40.025492	10.499522	17.000	32.000	38.000	47.000	94.000
campaign	8238.0	2.566642	2.782228	1.000	1.000	2.000	3.000	43.000
pdays	8238.0	962.713401	186.346378	0.000	999.000	999.000	999.000	999.000
previous	8238.0	0.173464	0.481933	0.000	0.000	0.000	0.000	5.000
emp.var.rate	8238.0	0.076912	1.567072	-3.400	-1.800	1.100	1.400	1.400
cons.price.idx	8238.0	93.571882	0.579322	92.201	93.075	93.444	93.994	94.767
cons.conf.idx	8238.0	-40.512637	4.611767	-50.800	-42.700	-41.800	-36.400	-26.900
euribor3m	8238.0	3.616390	1.733168	0.635	1.334	4.857	4.961	5.045
nr.employed	8238.0	5167.033576	72.258533	4963.600	5099.100	5191.000	5228.100	5228.100

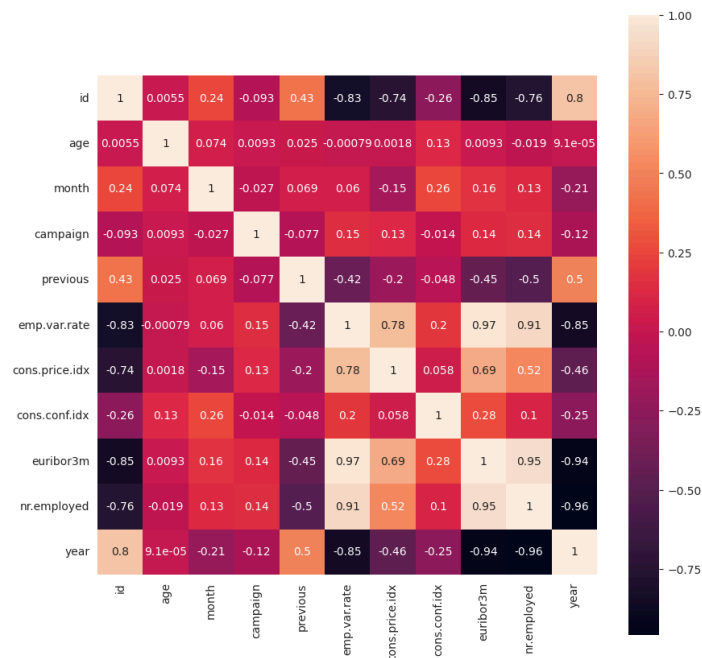
All of the figures show very similar statistics, which leads us to believe that data providers probably split the train and test sets in a stratified way.

3. Comparison of Data Preprocessing Changes with Proposal

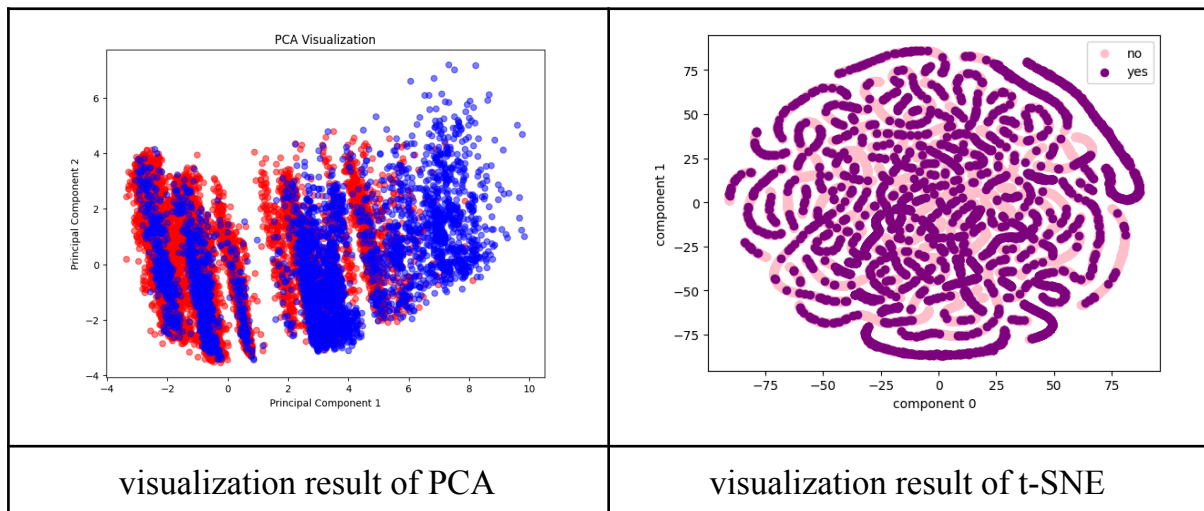
In the proposal stage, we changed the 'job' column to 'retired' if the 'age' column was over 65. Unfortunately, it did not contribute to increasing the F1 score, so we decided to preserve the original data.



Also done during the proposal stage, we used the information found by matching the economic indicator (consumer confidence index) of Portugal with the cons.conf.idx column one by one. By doing this, we could create the year derivative variable and delete the 'euribor3m' and 'nr.employed' columns as we checked the multicollinearity in the heatmap.



We figured out that dimensionality reduction techniques such as Principal Component Analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE) are another way to solve the multicollinearity problem. So, we tried both PCA and t-SNE, and then visualized the results.



As the visualization shows, we couldn't find any meaningful results from dimensionality reduction. Instead of using these techniques, we decided to delete the 'euribor3m' and 'nr.employed' columns as we initially did. It was less computationally expensive but still able to cope with the multicollinearity problem.

We compared every row with all the values of all columns except the 'id' and 'y' columns. We found there were some rows where only their 'y' values were different. So, we assumed that in some cases, the consumer's intent to subscribe to the bank campaign changed from 'no' to 'yes'. If the dataset included the time series, we might have been able to analyze it in more detail, but we couldn't as there were no time series. Thus, we preprocessed the duplicate columns, excluding the 'id' and 'y' columns, leaving only rows with a y value of 'yes'. Through the aforementioned processing, we also aimed to mitigate data bias.

We also replaced all unknown missing values with their mode values in columns 'job', 'marital' and 'loan'. In addition, the aforementioned factor, the 'pdays' column, was originally planned to replace all missing values with zeros, but during the model training process, it was found that the performance of the model decreased when the column was processed or removed, so we proceeded without processing missing values and deleting the column. Also, after creating the dummy variable, there were no rows in the test set with a 'default' column value of 'yes', so we deleted it from the train set as well.

Lastly, we scaled the dataset with dummy variables in it using MinMaxScaler, and moved on to the next data sampling steps.

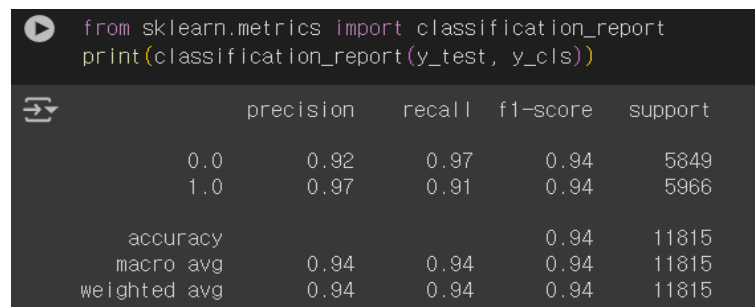
4. Exploring various approaches and processes for sampling and model selection

a. Sampling

We tried various sampling methods, including oversampling, undersampling, and a combination of oversampling & undersampling. Every sampling method was paired with the CatBoostClassifier. Details follow:

- Oversampling

- ADASYN (Adaptive Synthetic Sampling) is a technique used to address class imbalance in datasets by generating synthetic samples of the minority class. It focuses on generating samples in areas where the minority class is underrepresented, thereby adapting to the distribution of the data.



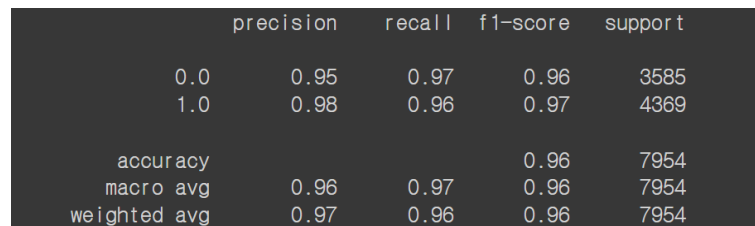
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_cls))
```

	precision	recall	f1-score	support
0.0	0.92	0.97	0.94	5849
1.0	0.97	0.91	0.94	5966
accuracy			0.94	11815
macro avg	0.94	0.94	0.94	11815
weighted avg	0.94	0.94	0.94	11815

- The model sampled with ADASYN achieved an F1 score of 0.94. However, such a highly elevated F1 score could be the result of overfitting, so we decided to explore other sampling methods.

- Oversampling & Undersampling

- SMOTEENN (SMOTE Edited Nearest Neighbors) is a hybrid sampling technique designed to address class imbalance. It combines SMOTE and ENN. It also helps synthesize minority oversampling technology combined with edited nearest neighbors.



	precision	recall	f1-score	support
0.0	0.95	0.97	0.96	3585
1.0	0.98	0.96	0.97	4369
accuracy			0.96	7954
macro avg	0.96	0.97	0.96	7954
weighted avg	0.97	0.96	0.96	7954

- SMOTETomek is a combination of two different techniques: SMOTE and Tomek Links. This combination addresses the class imbalance problem in datasets by simultaneously increasing the number of minority class instances and cleaning the dataset by removing noisy or borderline majority class instances.

	precision	recall	f1-score	support
0.0	0.36	0.49	0.42	743
1.0	0.93	0.89	0.91	5847
accuracy			0.84	6590
macro avg	0.65	0.69	0.66	6590
weighted avg	0.87	0.84	0.85	6590

- Introducing both oversampling and undersampling led to similar results as using only the oversampling method. SMOTEENN's F1 score was still significantly high, which may indicate overfitting, and the SMOTETomek's F1 score was lower than the other models and not useful.

- Undersampling

- NCR (Neighborhood Cleaning Rule) combines sampling techniques, which are CNN and ENN. It operates based on the neighbors of each data point. By examining the distribution of neighbor classes, it removes unnecessary noise.

	precision	recall	f1-score	support
0.0	0.91	0.97	0.94	3162
1.0	0.81	0.57	0.67	713
accuracy			0.90	3875
macro avg	0.86	0.77	0.81	3875
weighted avg	0.89	0.90	0.89	3875

- Tomek Links helps in cleaning the dataset by removing specific instances to improve the quality of the dataset for machine learning models.

	precision	recall	f1-score	support
0.0	0.91	0.98	0.94	5557
1.0	0.70	0.32	0.44	781
accuracy			0.90	6338
macro avg	0.81	0.65	0.69	6338
weighted avg	0.89	0.90	0.88	6338

- One-sided selection is a method to tackle class imbalance in classification. It selects a subset of the majority class while preserving minority class samples, enhancing classification performance by reducing noise.

	precision	recall	f1-score	support
0.0	0.92	0.98	0.95	5589
1.0	0.71	0.33	0.45	715
accuracy			0.91	6304
macro avg	0.81	0.65	0.70	6304
weighted avg	0.90	0.91	0.89	6304

Considering each result of every method we tried in the above, we concluded that NCR reliably increases F1 scores. Therefore, we only applied downsampling in the model selection process.

b. Model Selection

To establish a temporary guide for model selection, we compared several models using the PyCaret library. PyCaret is one of the autoML libraries that makes machine learning easy, including preprocessing data, training models, and finding the right parameters. So, we trained the models with functions supported by the above libraries.

```
model = compare_models(sort = 'F1', fold = 3, n_select = 5)
```

Initiated	22:47:45
Status	Loading Dependencies
Estimator	Compiling Library

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
lda	Linear Discriminant Analysis	0.8896	0.7961	0.8896	0.8799	0.8839	0.3925	0.3960	0.2433
lightgbm	Light Gradient Boosting Machine	0.9004	0.0000	0.9004	0.8827	0.8826	0.3394	0.3745	0.5267
catboost	CatBoost Classifier	0.8992	0.0000	0.8992	0.8810	0.8823	0.3414	0.3717	4.1900
gbc	Gradient Boosting Classifier	0.9017	0.8059	0.9017	0.8845	0.8821	0.3316	0.3742	1.2633
rf	Random Forest Classifier	0.8959	0.0000	0.8959	0.8770	0.8804	0.3358	0.3595	0.4633
knn	K Neighbors Classifier	0.8920	0.0000	0.8920	0.8735	0.8785	0.3341	0.3503	0.5833
lr	Logistic Regression	0.9002	0.7961	0.9002	0.8823	0.8771	0.2947	0.3481	0.8600
ada	Ada Boost Classifier	0.8984	0.8015	0.8984	0.8787	0.8767	0.2970	0.3419	0.5533
ridge	Ridge Classifier	0.8994	0.7961	0.8994	0.8808	0.8763	0.2904	0.3426	0.2133
et	Extra Trees Classifier	0.8857	0.0000	0.8857	0.8674	0.8736	0.3144	0.3253	0.4767
nb	Naive Bayes	0.8376	0.0000	0.8376	0.8760	0.8529	0.3535	0.3664	0.2167
dt	Decision Tree Classifier	0.8419	0.0000	0.8419	0.8505	0.8460	0.2513	0.2519	0.2700
dummy	Dummy Classifier	0.8873	0.0000	0.8873	0.7874	0.8344	0.0000	0.0000	0.2400
svm	SVM - Linear Kernel	0.6908	0.6495	0.6908	0.8770	0.6858	0.1969	0.2580	0.3867
qda	Quadratic Discriminant Analysis	0.4256	0.7881	0.4256	0.8708	0.4233	0.1490	0.1890	0.2500

```
save_model(tuned_lda, './lda')
```

Transformation Pipeline and Model Successfully Saved

```
(Pipeline(memory=Memory(location=None),
  steps=[('label_encoding',
    TransformerWrapperWithInverse(exclude=None, include=None,
      transformer=LabelEncoder())),
    ('numerical_imputer',
      TransformerWrapper(exclude=None,
        include=['id', 'age', 'campaign', 'pdays',
          'previous', 'emp.var.rate',
          'cons.price.idx', 'cons.conf.idx',
          'euribor3m', 'nr.employed'],
        transformer=SimpleImpute...,
          return_df=True,
          use_cat_names=True,
          verbose=0))),
    ('clean_column_names',
      TransformerWrapper(exclude=None, include=None,
        transformer=CleanColumnNames(match='[\\]\\[\\],\\{\\}\\|\\\"\\:;]+'))),
    ('trained_model',
      LinearDiscriminantAnalysis(covariance_estimator=None,
        n_components=None, priors=None,
        shrinkage=None, solver='svd',
        store_covariance=False,
        tol=0.0001))),
  verbose=False),
'./lda.pkl')
```

However, several issues existed. First, when we tuned the model and printed the pipeline, we saw that it was a complex shape that could be overfitted. Then we looked at other metrics, including F1 score(0.89) and decision boundary, and we were confident that this indicated overfitting. Additionally, the results from PyCaret did not align with those of the model we coded manually. After careful consideration, we decided to base our model selection on the results from our own code, and use the PyCaret results only to narrow down a few models for model training. In order to be more time-efficient, at the beginning of the project, we tried to train and compare different models without cross-validation, using only preprocessing. The following is a list of the models we used.

- Logistic regression
- decision tree
- randomforest
- Naive Bayes
- SGD (Stochastic Gradient Descent)
- Gradient boosting classifier
- lightGBM
- BBC (balanced bagging classifier)
- XGboost
- Catboost

Results for each model follow below;

<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.90</td><td>0.98</td><td>0.94</td><td>5354</td></tr><tr><td>1.0</td><td>0.65</td><td>0.23</td><td>0.34</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.89</td><td>6132</td></tr><tr><td>macro avg</td><td>0.77</td><td>0.60</td><td>0.64</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.89</td><td>0.86</td><td>6132</td></tr></table> <p>Logistic Regression</p>		precision	recall	f1-score	support	0.0	0.90	0.98	0.94	5354	1.0	0.65	0.23	0.34	778	accuracy			0.89	6132	macro avg	0.77	0.60	0.64	6132	weighted avg	0.87	0.89	0.86	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.91</td><td>0.96</td><td>0.93</td><td>5354</td></tr><tr><td>1.0</td><td>0.55</td><td>0.31</td><td>0.40</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.88</td><td>6132</td></tr><tr><td>macro avg</td><td>0.73</td><td>0.64</td><td>0.67</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.88</td><td>0.87</td><td>6132</td></tr></table> <p>Decision Tree</p>		precision	recall	f1-score	support	0.0	0.91	0.96	0.93	5354	1.0	0.55	0.31	0.40	778	accuracy			0.88	6132	macro avg	0.73	0.64	0.67	6132	weighted avg	0.86	0.88	0.87	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.89</td><td>0.99</td><td>0.94</td><td>5354</td></tr><tr><td>1.0</td><td>0.70</td><td>0.18</td><td>0.29</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.89</td><td>6132</td></tr><tr><td>macro avg</td><td>0.80</td><td>0.59</td><td>0.61</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.89</td><td>0.86</td><td>6132</td></tr></table> <p>Random Forest</p>		precision	recall	f1-score	support	0.0	0.89	0.99	0.94	5354	1.0	0.70	0.18	0.29	778	accuracy			0.89	6132	macro avg	0.80	0.59	0.61	6132	weighted avg	0.87	0.89	0.86	6132
	precision	recall	f1-score	support																																																																																								
0.0	0.90	0.98	0.94	5354																																																																																								
1.0	0.65	0.23	0.34	778																																																																																								
accuracy			0.89	6132																																																																																								
macro avg	0.77	0.60	0.64	6132																																																																																								
weighted avg	0.87	0.89	0.86	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.91	0.96	0.93	5354																																																																																								
1.0	0.55	0.31	0.40	778																																																																																								
accuracy			0.88	6132																																																																																								
macro avg	0.73	0.64	0.67	6132																																																																																								
weighted avg	0.86	0.88	0.87	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.89	0.99	0.94	5354																																																																																								
1.0	0.70	0.18	0.29	778																																																																																								
accuracy			0.89	6132																																																																																								
macro avg	0.80	0.59	0.61	6132																																																																																								
weighted avg	0.87	0.89	0.86	6132																																																																																								
<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.92</td><td>0.87</td><td>0.89</td><td>5354</td></tr><tr><td>1.0</td><td>0.34</td><td>0.48</td><td>0.40</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.82</td><td>6132</td></tr><tr><td>macro avg</td><td>0.63</td><td>0.67</td><td>0.65</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.85</td><td>0.82</td><td>0.83</td><td>6132</td></tr></table> <p>Naive Bayes</p>		precision	recall	f1-score	support	0.0	0.92	0.87	0.89	5354	1.0	0.34	0.48	0.40	778	accuracy			0.82	6132	macro avg	0.63	0.67	0.65	6132	weighted avg	0.85	0.82	0.83	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.87</td><td>1.00</td><td>0.93</td><td>5354</td></tr><tr><td>1.0</td><td>0.00</td><td>0.00</td><td>0.00</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.87</td><td>6132</td></tr><tr><td>macro avg</td><td>0.44</td><td>0.50</td><td>0.47</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.76</td><td>0.87</td><td>0.81</td><td>6132</td></tr></table> <p>Stochastic Gradient Descent</p>		precision	recall	f1-score	support	0.0	0.87	1.00	0.93	5354	1.0	0.00	0.00	0.00	778	accuracy			0.87	6132	macro avg	0.44	0.50	0.47	6132	weighted avg	0.76	0.87	0.81	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.91</td><td>0.97</td><td>0.94</td><td>5354</td></tr><tr><td>1.0</td><td>0.60</td><td>0.30</td><td>0.40</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.89</td><td>6132</td></tr><tr><td>macro avg</td><td>0.75</td><td>0.64</td><td>0.67</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.89</td><td>0.87</td><td>6132</td></tr></table> <p>Gradient Boosting Classifier</p>		precision	recall	f1-score	support	0.0	0.91	0.97	0.94	5354	1.0	0.60	0.30	0.40	778	accuracy			0.89	6132	macro avg	0.75	0.64	0.67	6132	weighted avg	0.87	0.89	0.87	6132
	precision	recall	f1-score	support																																																																																								
0.0	0.92	0.87	0.89	5354																																																																																								
1.0	0.34	0.48	0.40	778																																																																																								
accuracy			0.82	6132																																																																																								
macro avg	0.63	0.67	0.65	6132																																																																																								
weighted avg	0.85	0.82	0.83	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.87	1.00	0.93	5354																																																																																								
1.0	0.00	0.00	0.00	778																																																																																								
accuracy			0.87	6132																																																																																								
macro avg	0.44	0.50	0.47	6132																																																																																								
weighted avg	0.76	0.87	0.81	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.91	0.97	0.94	5354																																																																																								
1.0	0.60	0.30	0.40	778																																																																																								
accuracy			0.89	6132																																																																																								
macro avg	0.75	0.64	0.67	6132																																																																																								
weighted avg	0.87	0.89	0.87	6132																																																																																								
<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.90</td><td>0.97</td><td>0.94</td><td>5354</td></tr><tr><td>1.0</td><td>0.63</td><td>0.29</td><td>0.40</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.89</td><td>6132</td></tr><tr><td>macro avg</td><td>0.77</td><td>0.63</td><td>0.67</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.89</td><td>0.87</td><td>6132</td></tr></table> <p>LightGBM</p>		precision	recall	f1-score	support	0.0	0.90	0.97	0.94	5354	1.0	0.63	0.29	0.40	778	accuracy			0.89	6132	macro avg	0.77	0.63	0.67	6132	weighted avg	0.87	0.89	0.87	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.91</td><td>0.97</td><td>0.94</td><td>5354</td></tr><tr><td>1.0</td><td>0.57</td><td>0.31</td><td>0.40</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.89</td><td>6132</td></tr><tr><td>macro avg</td><td>0.74</td><td>0.64</td><td>0.67</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.88</td><td>0.87</td><td>6132</td></tr></table> <p>XGBoost</p>		precision	recall	f1-score	support	0.0	0.91	0.97	0.94	5354	1.0	0.57	0.31	0.40	778	accuracy			0.89	6132	macro avg	0.74	0.64	0.67	6132	weighted avg	0.86	0.88	0.87	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.91</td><td>0.97</td><td>0.94</td><td>5354</td></tr><tr><td>1.0</td><td>0.57</td><td>0.31</td><td>0.40</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.88</td><td>6132</td></tr><tr><td>macro avg</td><td>0.74</td><td>0.64</td><td>0.67</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.86</td><td>0.88</td><td>0.87</td><td>6132</td></tr></table> <p>CatBoost</p>		precision	recall	f1-score	support	0.0	0.91	0.97	0.94	5354	1.0	0.57	0.31	0.40	778	accuracy			0.88	6132	macro avg	0.74	0.64	0.67	6132	weighted avg	0.86	0.88	0.87	6132
	precision	recall	f1-score	support																																																																																								
0.0	0.90	0.97	0.94	5354																																																																																								
1.0	0.63	0.29	0.40	778																																																																																								
accuracy			0.89	6132																																																																																								
macro avg	0.77	0.63	0.67	6132																																																																																								
weighted avg	0.87	0.89	0.87	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.91	0.97	0.94	5354																																																																																								
1.0	0.57	0.31	0.40	778																																																																																								
accuracy			0.89	6132																																																																																								
macro avg	0.74	0.64	0.67	6132																																																																																								
weighted avg	0.86	0.88	0.87	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.91	0.97	0.94	5354																																																																																								
1.0	0.57	0.31	0.40	778																																																																																								
accuracy			0.88	6132																																																																																								
macro avg	0.74	0.64	0.67	6132																																																																																								
weighted avg	0.86	0.88	0.87	6132																																																																																								
<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.93</td><td>0.87</td><td>0.90</td><td>5354</td></tr><tr><td>1.0</td><td>0.39</td><td>0.57</td><td>0.47</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.83</td><td>6132</td></tr><tr><td>macro avg</td><td>0.66</td><td>0.72</td><td>0.68</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.83</td><td>0.85</td><td>6132</td></tr></table> <p>Decision Tree + BBC</p>		precision	recall	f1-score	support	0.0	0.93	0.87	0.90	5354	1.0	0.39	0.57	0.47	778	accuracy			0.83	6132	macro avg	0.66	0.72	0.68	6132	weighted avg	0.87	0.83	0.85	6132	<table><tr><th></th><th>precision</th><th>recall</th><th>f1-score</th><th>support</th></tr><tr><td>0.0</td><td>0.94</td><td>0.87</td><td>0.90</td><td>5354</td></tr><tr><td>1.0</td><td>0.40</td><td>0.62</td><td>0.48</td><td>778</td></tr><tr><td>accuracy</td><td></td><td></td><td>0.83</td><td>6132</td></tr><tr><td>macro avg</td><td>0.67</td><td>0.74</td><td>0.69</td><td>6132</td></tr><tr><td>weighted avg</td><td>0.87</td><td>0.83</td><td>0.85</td><td>6132</td></tr></table> <p>Random Forest + BBC</p>		precision	recall	f1-score	support	0.0	0.94	0.87	0.90	5354	1.0	0.40	0.62	0.48	778	accuracy			0.83	6132	macro avg	0.67	0.74	0.69	6132	weighted avg	0.87	0.83	0.85	6132																															
	precision	recall	f1-score	support																																																																																								
0.0	0.93	0.87	0.90	5354																																																																																								
1.0	0.39	0.57	0.47	778																																																																																								
accuracy			0.83	6132																																																																																								
macro avg	0.66	0.72	0.68	6132																																																																																								
weighted avg	0.87	0.83	0.85	6132																																																																																								
	precision	recall	f1-score	support																																																																																								
0.0	0.94	0.87	0.90	5354																																																																																								
1.0	0.40	0.62	0.48	778																																																																																								
accuracy			0.83	6132																																																																																								
macro avg	0.67	0.74	0.69	6132																																																																																								
weighted avg	0.87	0.83	0.85	6132																																																																																								

In conclusion, the different models did not significantly change the F1 score. Among the Decision Tree, Gradient Boosting Classifier, and Catboost models that had relatively high f1 scores and other metrics, we chose Catboost as our baseline because it handles categorical data relatively well.

5. Finalized model (includes model training and validation)

We did all of the preprocessing mentioned above and tried to solve the target imbalance problem by undersampling using NCR. In addition, we trained the catboost model by finding the appropriate parameters through grid search. We introduced k-fold cross-validation (in the final model, k=5) because we believed that it would improve the general performance of the model on the test set.

	precision	recall	f1-score	support
0.0	0.91	0.97	0.94	3498
1.0	0.80	0.54	0.65	759
accuracy			0.89	4257
macro avg	0.86	0.76	0.79	4257
weighted avg	0.89	0.89	0.89	4257

6. Troubleshooting Processes & Remaining Problems

We wanted to try LOOCV, a type of cross-validation but it wasn't possible due to repeated runtime initialization, so we expect performance to improve once the computation issue is resolved. In addition, since the Colab Notebook has a structure that shuts down if the user does not make any movements for 90 minutes, we improved the inconvenience by entering the JavaScript code in the console of the devtool so that the code continues to work after a certain period of time.

```

LOOCV

# CatBoost 모델 정의
model = CatBoostClassifier()

param_grid = {'iterations': [200, 300], 'depth': [5], 'learning_rate': [0.01, 0.05, 0.1], 'l2_leaf_reg': [3, 5, 7]}

from sklearn.model_selection import LeaveOneOut
loo = LeaveOneOut()
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='f1', cv=loo, verbose=1, n_jobs=-1)

#model training
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
|
print(best_params)
print(best_score)

best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

print(classification_report(y_test, y_pred))

```

Fitting 17024 folds for each of 18 candidates, totalling 306432 fits

6시간 39분 13초 오전 5:50에 완료됨

Except for the above code, we tried several runs excluding possible parameters to reduce runtime.

4시간 54분 19초 오전 8:07에 완료됨	5시간 11분 32초 오전 8:07에 완료됨	6시간 39분 13초 오전 5:50에 완료됨
<pre> function ClickConnect(){ console.log("Working"); document.querySelector("colab-toolbar-button#connect").click() }setInterval(ClickConnect, 1800000) // the code we used </pre>		

It would have been nice to use a collaborative tool (like Jira, Github, etc.) to version control the code or to be able to view performance more comfortably. We just used Notion pages and the shared drives to manage them.

As mentioned earlier about the data, when the user changes their own mind between yes and no and vice versa, there is no temporal data included, so when we want to include it in the business model, we think time-related information should be included. Also, we couldn't be sure if it was the same person because it was duplicated except for the 'id' and 'y' values.

Running in multiple environments would have been confusing. such as VS Code, PyCharm, and Colab notebooks. Except for running PyCaret locally, which cannot be run in colab, we have unified the code execution environment for colab notebooks.

7. References

- scikit-learn, "GridSearchCV",
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#gridsearchcv
- scikit-learn, "Ensembles: Gradient boosting, random forests, bagging, voting, stacking", <https://scikit-learn.org/stable/modules/ensemble>
- scikit-learn, "Cross-validation: evaluating estimator performance",
https://scikit-learn.org/stable/modules/cross_validation
- matplotlib, "Statistical distributions",
https://matplotlib.org/stable/plot_types/stats/index.html