Springboard - DSC
Capstone Project II
Detecting Potential Candidates Who are Looking for a New Job

Final Report

Yang Liu Kunz

Nikoloz Skhirtladze

Data Science Career Track

November 2020

Table of Contents

1 Introduction **3**

1.1 Objective **3**

1.2 Significance **3**

2 Dataset **4**

2.1 Data Description **4**

2.2 Dataset Characteristic: Class Imbalance and Feature Space Overlap **5**

3 Package Introduction **6**

4 Data Wrangling **7**

4.1 Dataset Information **7**

4.2 Data Processing **8**

5 Exploratory Data Analysis **9**

5.1 Summary Statistic **9**

5.2 Overall Distribution **9**

5.3 Experience vs. Target **11**

5.4 City vs. Target **12**

5.5 Training Hours vs. Target **13**

5.6 Variable Correlation Coefficient **13**

6 Machine Learning **14**

6.1 Data Preprocessing and Feature Selection **14**

6.2 Model Selection **15**

6.3 Baseline Model Evaluation **16**

6.4 Oversampling (SMOTE) **20**

6.4.1 Model Comparison after SMOTE 20

6.4.2 Model and key Metric Selection 22

6.5 Experiment with Model Xgboost Classifier **25**

6.6 Experiment with Model CatBoost Classifier **27**

7 Final Model Selection and Hyperparameters Tuning **29**

8 Conclusion **32**

8.1 Dataset **32**

8.2 Models **32**

8.3 Feature Work **33**

References **34**

***Appendix*......35**

1 Introduction

Company XYZ is a training institute which conducts training for analytics/data science. They want to expand their business to include manpower recruitment (data science only). They aim to do so by connecting their enrollees with their clients who are looking to hire employees working in the same domain. Before that they want to know among the large number of signups, which of these candidates are looking for a new employment. To understand the factors that lead the enrollees to look for a job change, Company XYZ wants to build a model based on the current credentials, demographics and experience data they collected from the enrollees to predict the probability of them to look for a new job. By identifying as many target enrollees (potential candidates) as possible from all enrollees registered on the training platform, the recruiting team in company XYZ could approach to their targeted potential candidates more efficiently and effectively.

1.1 Objective

The objectives of this project are to:

- Explore and analyze enrollee data for the XYZ training institute
- Identify the key features that lead enrollees to look for new employment
- Develop machine learning models that predict the probability of enrollees looking for new jobs
- Identify the final model that captures the most target enrollees within the top 20% and top 50% of the test dataset in descending order by their prediction scores

This report is divided into the following sections:

- Section 2: Dataset
- Section 3: Package Introduction
- Section 4: Data Wrangling
- Section 5: Exploratory Data Analysis
- Section 6: Machine Learning
- Section 7: Final Model Selection and Hyperparameters Tuning
- Section 8: Conclusion
- The programming codes used for this report can be found in this Github Repository Detecting_Potential_Candidate_Springboard_Capstone2.

1.2 Significance

By thoroughly explore the dataset, we will identify the important features that affect the enrollee's decision of career change. We will also develop machine learning models that can be used by the recruiting team of company XYZ to filter out the potential candidates from the user data base and approach them with better efficiency and accuracy.

2 Dataset

2.1 Data Description

The datasets are sourced from the website kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners that allows users to find and publish datasets, explore and build models in a web-based data-science environment. The dataset train.csv used in this project, was collected on August 12, 2020. It consists of 18,359 rows and 14 columns, 4 of them are numerical columns and 10 of them are categorical columns. Each row contains credentials/demographics/experience data for each unique enrollee. A description of each of the 14 columns is provided in *Table 1.1*

Table 1.1 Description of dataset

No.	Variable Name	Variable Description	Data Summary
1	enrollee_id	unique ID for enrollees	integer, 18359 unique values
2	city	city code	object, 123 unique values
3	city_development_index	development index of the city (scaled)	float, 93 unique values
4	gender	gender	object, 3 unique values
5	relevent[1]_experience	relevant experience in analytics, data science	object, 2 unique values
6	enrolled_university	type of University course enrolled if any	object, 3 unique values
7	education_level	education level	object, 5 unique values
8	major_discipline	major discipline	object, 6 unique values
9	experience	total working experience in years	object, 22 unique values
10	company_size	number of employees in current employer's company	object, 8 unique values
11	company_type	type of current employer	object, 6 unique values
12	last_new_job	difference in years between previous job and current job	object, 6 unique values
13	training_hours	training hours completed	integer, 241 unique values
14	target	looking for job change or not	Integer, 0=not looking for job change, 1=Looking for a job change

[1] The original dataset mis-spelled word 'relevant' as 'relevent'. We used the mis-spelled one in this study.

2.2 Dataset Characteristic: Class Imbalance and Feature Space Overlap

A distinctive characteristic of this dataset is its class imbalance. In the original dataset train.csv, there are 15,934 negative classes and 2425 positive classes in column 'target'. Value 0 represents the negative class which means the enrollee is not open for career change, we will define this class as 'non-Target' enrollee in this study. While value 1 represents the positive class, means the enrollee is open for career change, we will define this class 'Target' enrollee in this study, using capitalized 'Target' to distinguish it from column 'target'. A breakdown of non-Target enrollee and Target enrollee is provided in *Table 2.1*.

Table 2.1 Number of non-Target and Target enrollee

	Label	Counts	Percentage
0	non-Target	15934	86.79
1	Target	2425	13.21

We can see there is an approximately 6:1 ratio of non-Target enrollees to Target enrollees. The much smaller proportion of Target enrollees would cause the baseline models tend to only predict the majority class (non-Target enrollee) as the features of the minority class (Target enrollee) are treated as noise. As a matter of fact, most of the base line models we selected for this study misclassified almost all the Target enrollee in the test set (the baseline models' Recall on Target enrollee is very low or was zero). Therefore, we will address this issue in the further section (Section 5) with introduction of oversampling technique.

Another issue of this dataset is high degree of feature overlap between classes. Many Target enrollee have feature values that are very identical to feature values of non-Target enrollee. This could bring difficulty for the models to predict Target enrollee without increasing False Positive (non-Target enrollee misclassified as Target enrollee).

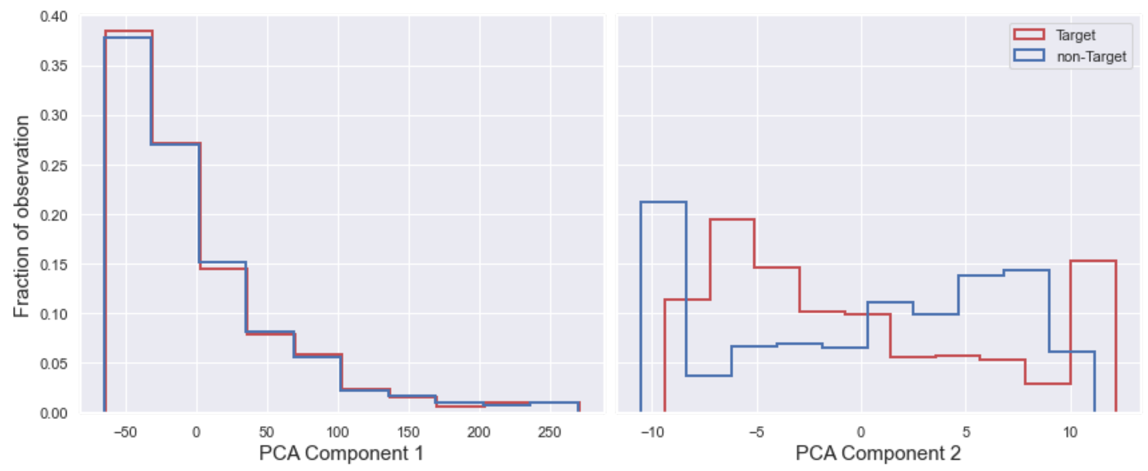

Figure 2.2 Normalized histogram for Target and non-Target enrollee after PCA reduction of features to 2 components **categorical variables were converted to numerical in this plot.*

By applying PCA to observe variables onto two dimensional spaces, we observed a lack of distinction between Target enrollee and non-Target enrollee. In *Figure 2.2*, the 2 classes are seen to have significant amount of overlap and indistinguishable distribution across all features.

3 Package Introduction

In this study we used JupyterLab (2.1.5) to run all the code. Numpy (1.18.5), Pandas (1.0.5), Matplotlib (3.2.2), Seaborn (0.11.0) were installed as basic package. Scikit-learn (0.23.1) was installed as the machine learning library. Imblearn (0.7.0) was installed for data oversampling. Scikitplot (0.3.7) was installed for plotting Cumulative Lift.

4 Data Wrangling

4.1 Dataset Information

Please find the general information of the original dataset from *Table 4.1,* sorted by the count of unique value in each variable in descending order. It contains columns of original variable index, variable name, counts of unique values, percentage of unique value, percentage of missing value and data type of each variable.

Table 4.1 General information of dataset

Index	Variable Name	Counts	Unique Value Percentage	Missing Value Percentage	Data Type
0	enrollee_id	18359	100.00	0.00	int64
12	training_hours	241	1.31	0.00	int64
1	city	123	0.67	0.00	object
2	city_development_index	93	0.51	0.00	float64
8	experience	22	0.12	0.32	object
9	company_size	8	0.04	26.03	object
7	major_discipline	6	0.03	15.46	object
10	company_type	6	0.03	27.45	object
11	last_new_job	6	0.03	2.00	object
6	education_level	5	0.03	2.49	object
3	gender	3	0.02	22.32	object
5	enrolled_university	3	0.02	1.86	object
4	relevent_experience	2	0.01	0.00	object
13	target	2	0.01	0.00	int64

*variable that has missing value percentage higher than 20% is highlighted in yellow

According to the information we summarized in *Table 4.1*, 10 out of 14 variables are categorical variables, 8 out of 14 variables have missing values, and the missing value ratios in variable ***company_type, company_size, gender*** are all higher than 20%.

4.2 Data Processing

After examining all the variables, we decided to group variables to 6 groups based on their features (we will exclude 'enrollee_id' and 'target' from the grouping process). Then we chose the optimal data processing method for each group based on their variables' characteristics. The detailed explanation of process we took for each group will be find in *Table 4.2*.

Tabel 4.2 Data processing method on each variable

Group	Variable Feature	Variable Names	Missing Value Percentage	Process Method
Group 1	experience related variables	relevent experience	0	data type correction
		experience	0.32	data type correction, missing value imputation (fill with mode)
		last_new_job	2	data type correction, missing value imputation (fill with mode)
Group 2	employer related variables	company_type,	27.45	data grouping, missing value imputation (fill with KNN imputation) *KNN imputation will automatically convert the variable's data type to float
		company_size	26.03	data grouping, missing value imputation (fill with KNN imputation)
Group 3	education related variables	education_level,	2.49	data grouping, data type correction, missing value imputation (fill with 0)
		major_discipline	15.46	data grouping, data type correction, missing value imputation (fill with 0 or mode) *when education_level equals to 0 (None) or 1 (high school or primary school) fill missing value with 0, then fill the rest with 0
		enrolled_university	1.86	fill with mode
Group 4	gender	gender	22.32	data type correction, missing value imputation (fill with mode)
Group 5	city related variables	city	-	data grouping, data type correction (dummy encoding)
		city_development_index	-	-
Group 6	training related variables	training_hours	-	-

*This grouping is purely used for data processing, not for analysis or other purpose.

In the cases that the variable has a missing value percentage higher than 25%, we chose KNN imputation over filling the missing value with 0 or with mode value, in order to avoid introducing a certain type of bias to the dataset. The cleaned data contains 18,359 rows and 23 columns. You can find more details about the cleaning process from this jupyter notebook Detecting Potential Candidate_Part1(01.01-06.04.03).ipynb. (part 04.06 Variable Grouping)

5 Exploratory Data Analysis

5.1 Summary Statistic

The statistic including mean, standard deviation, minimum values, maximum values and percentile for each variable were summarized in *Table 5.1*

Table 5.1 Summary statistics

variable name	count	mean	std	min	25%	50%	75%	max
city_development_index	18359.0	0.847140	0.110189	0.448	0.796	0.91	0.92	0.949
gender	18359.0	1.085299	0.314034	1.000	1.000	1.00	1.00	3.000
relevent_experience	18359.0	0.740563	0.438338	0.000	0.000	1.00	1.00	1.000
enrolled_university	18359.0	0.237377	0.425487	0.000	0.000	0.00	0.00	1.000
education_level	18359.0	2.082194	0.693807	0.000	2.000	2.00	3.00	3.000
major_discipline	18359.0	1.585326	0.755994	0.000	1.000	2.00	2.00	2.000
experience	18359.0	10.647040	6.769905	0.000	5.000	9.00	16.00	21.000
company_size	18359.0	1.895419	0.730158	1.000	1.000	2.00	2.00	3.000
company_type	18359.0	2.700147	0.621645	1.000	3.000	3.00	3.00	3.000
last_new_job	18359.0	2.044338	1.680945	0.000	1.000	1.00	3.00	5.000
training_hours	18359.0	65.899014	60.885300	1.000	23.000	47.00	89.00	336.000
target	18359.0	0.132088	0.338595	0.000	0.000	0.00	0.00	1.000

5.2 Overall Distribution

The overall distribution of numerical variables was visualized in *Figure 5.1*. Unique values and their counts breakdown by non-Target and Target enrollee for each categorical variable were visualized in *Figure 5.2*

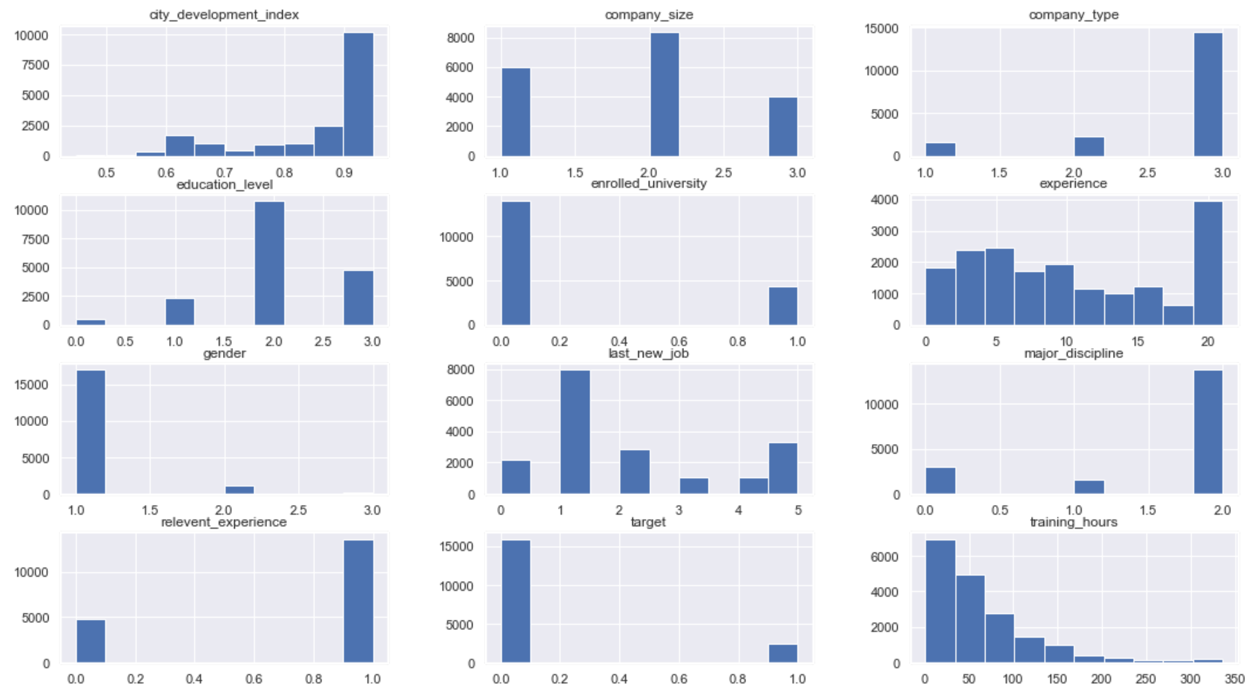

Figure 5.1 Distributions of numerical variables

please note that this was visualized after data grouping and data type correction for categorical variables

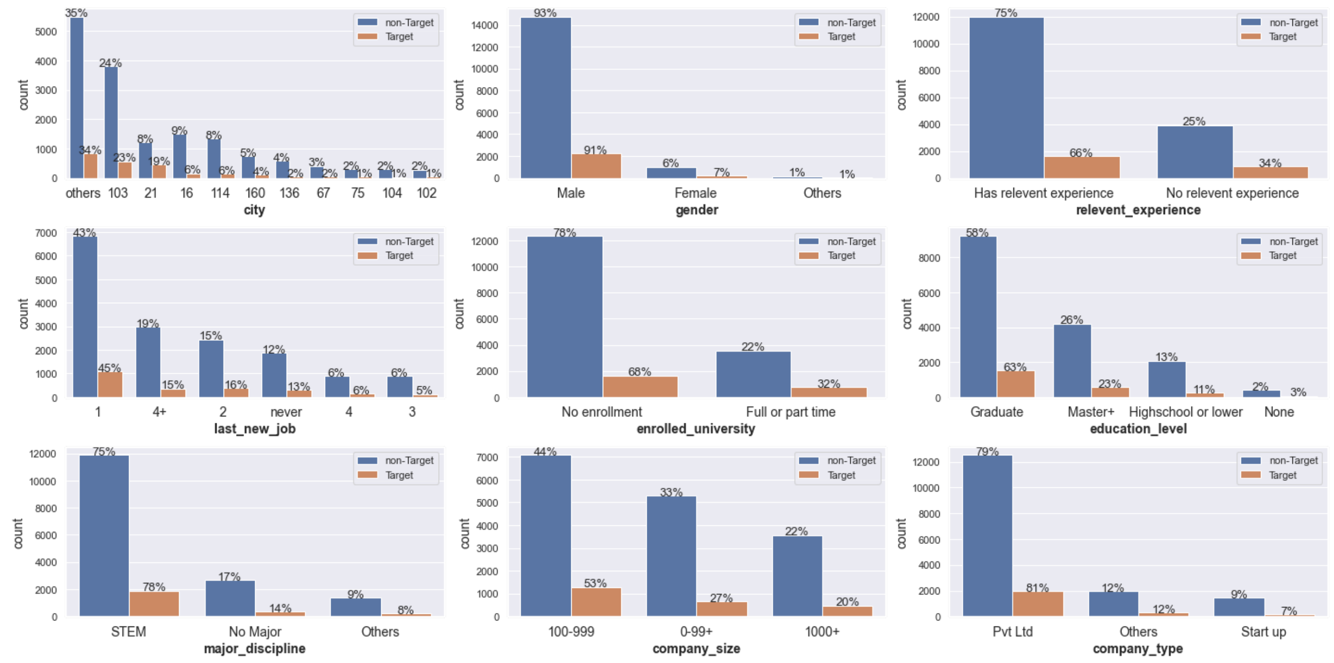

Figure 5.2 Barplots of categorical variables break down by non-Target and Target enrollee

please note that this was visualized after data grouping

From *Figure 5.2*, we observed the relation between target and most of the variables (expect variable *experience, city_development_index, training_hours*). We noticed that about 42% of the enrollees in the dataset are from city_103, city_21, city_16, and Target enrollees from these 3 cities contributed approximately 48% of all the Target enrollees in the dataset. We also noticed that Male is very dominant in the gender variable, Graduate is the most common education level among all the enrollees and the Target enrollees. Most Target enrollee have majored in STEM, worked for medium large size companies with comparatively shorter period of time (1 year) between their current job and last job.

5.3 Experience vs. Target

You may have noticed that variable 'experience' is excluded from *Figure 5.2*, as we consider *Table 5.2* and *Figure 5.3* (boxplot) to be a better option to visualize this variable.

Table 5.2 Experience summary statistic break down by variable ***target***

target	count	mean	std	min	25%	50%	75%	max
non-Target	15934.0	10.872725	6.763342	0.0	5.0	10.0	17.0	21.0
Target	2425.0	9.164124	6.625679	0.0	4.0	7.0	14.0	21.0

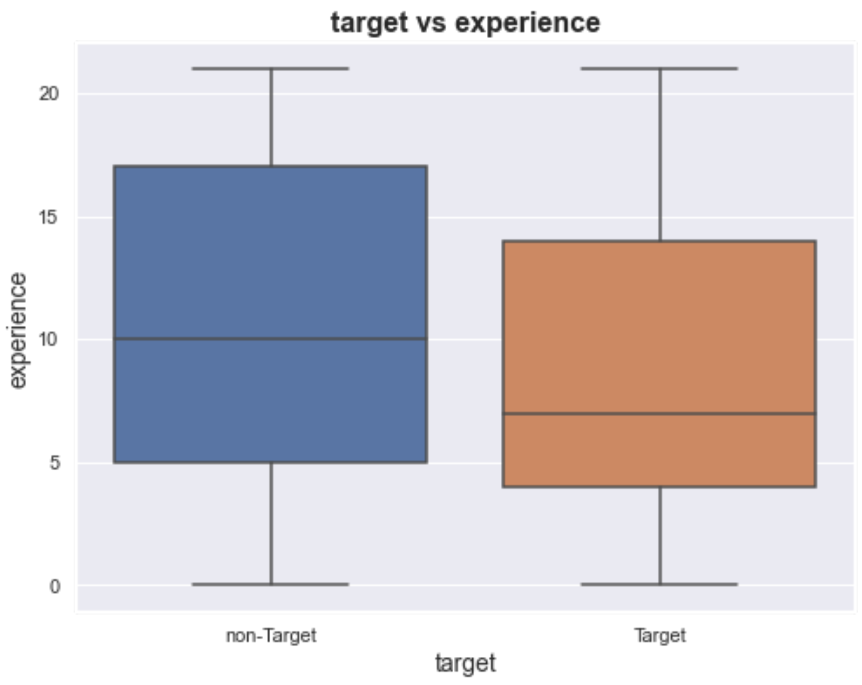

Figure 5.3 boxplot for variable target and experience

We can see that non-Target enrollee compares to Target enrollee, has slightly longer average working years and wider range of working experience in years.

5.4 City vs. Target

By plotting the distribution of two classes in *Figure 5.4,* we noticed, regardless of Target or non-Target enrollee, the peak in the distribution located at city_development_index that ranges 0.910 ~ 0.920 (city_16, city_103, city_160). For Target enrollee the second peak located at city_development_index equals to 0.624 (city_21). This would explain why out of 2425 Target enrollee in our dataset, 1170 of them were from city_103, city_21 and city_16, which shares about 48% of the total number.

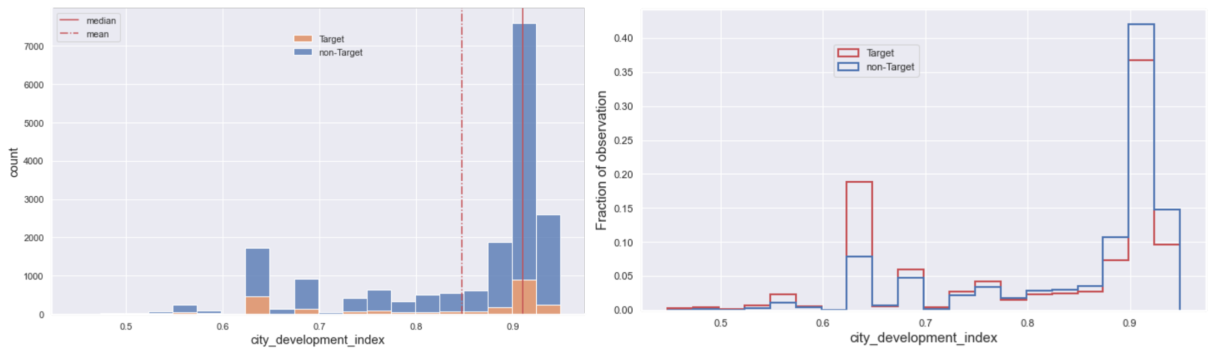

Figure 5.4 city_development_index distribution breakdown by non-Target and Target enrollee before (left) and after normalization (right)

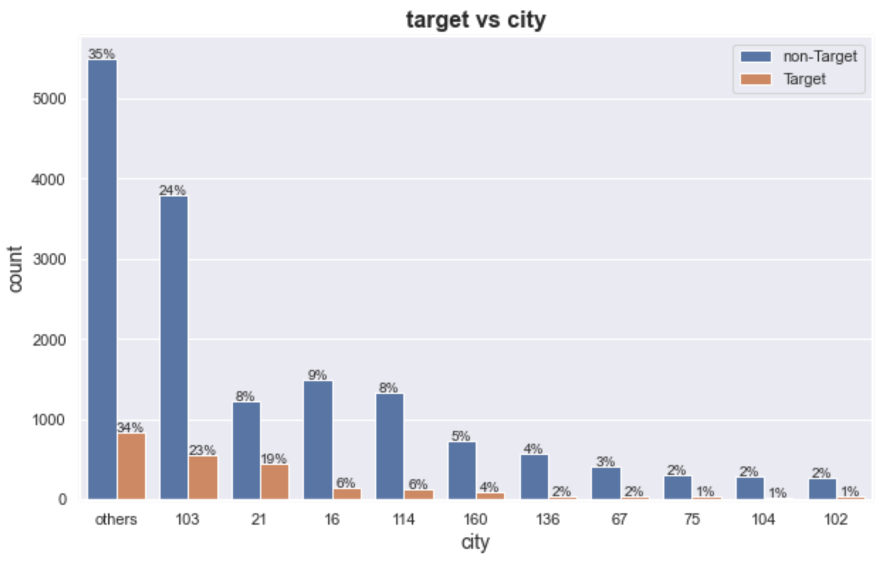

Figure 5.5 city barplot breakdown by non-target and Target enrollee

5.5 Training Hours vs. Target

*Table 5.3 Training hours summary statistic break down by variable **target***

target	count	mean	std	min	0.25	0.50	0.75	max
non-Target	15934	66.01	60.91	1.00	23.00	47.00	89.00	336.00
Target	2425	65.16	60.74	1.00	23.00	47.00	87.00	336.00

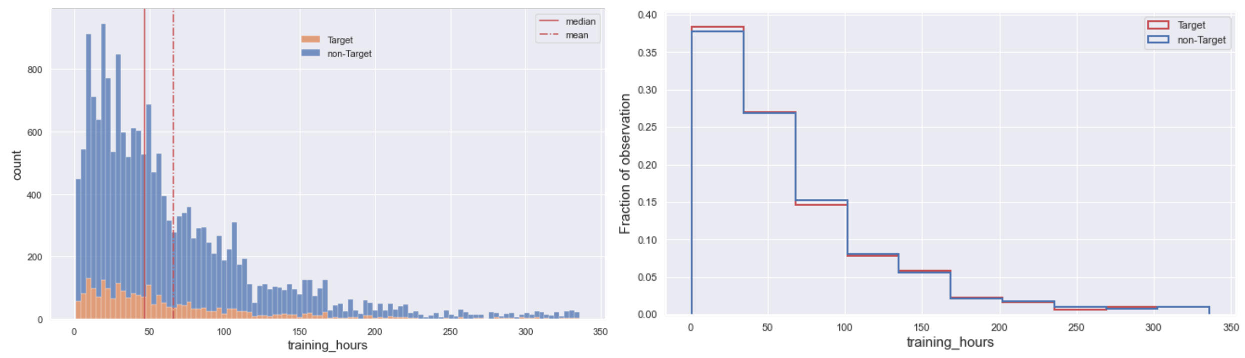

Figure 5.6 Training hours distribution breakdown by non-Target and Target enrollee before(left) and after normalization (right)

From *Figure 5.6*, we observed regardless of the class, the training time distribution is skewed right with a wide range of training time from 1 hour to 336 hours. From *Table 5.3* we noticed the two classes share lots of similarities in terms of summary statistic. The average training hours for Target and non-Target enrollee only has less than 1hour difference.

5.6 Variable Correlation Coefficient

After plotting heatmap *Figure 5.7*, we can see no significant correlation coefficient[2] was found among variables.

[2] As we dummy encoded variable city, therefore the correlation coefficient may not be statistically robust due to these binary values

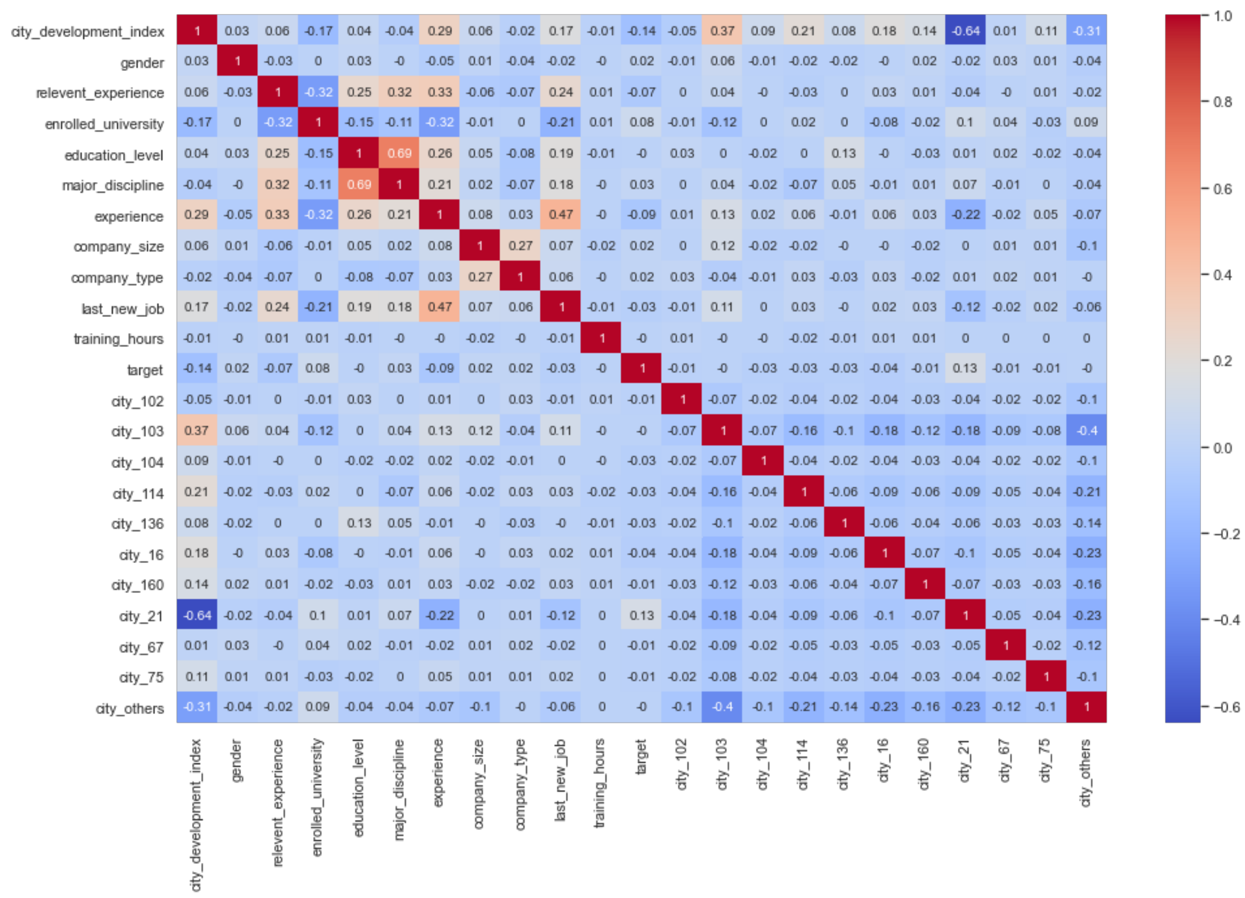

Figure 5.7 Heatmap of correlation coefficients between variables

6 Machine Learning

6.1 Data Preprocessing and Feature Selection

Dummy features were created for categorical variable city. As we already corrected the data type for all the other categorical variables in Section 4 Data Wrangling, so we only needed to dummy encode variable city in this step. This process increased the number of variables from 14 to 24. Then we dropped variable enrollee_id and trained our models with the rest 23 features. The data was split into 70%/30% training/testing sets and stratified on the 'target' variable to ensure an equal percentage of Target enrollee samples in each set, respectively. More details could be found in this jupyter notebook Detecting Potential Candidate_Part1(01.01-06.04.03).ipynb.(part 06.01 Data Preprocessing and Feature Selection)

Tabel 6.1 Stratified training and testing sets

	non-Target Count	Target Count	Target %
Full Dataset	15,934	2,425	13.209
Training set	11,154	1,697	13.205
Testing set	4,780	728	13.217

6.2 Model Selection

In this section we studied the performance of 10 classification models: Logistic Regression, Gaussian Naïve Bayes, k-Nearest Neighbors, Support Vector Machine, Decision Tree, Random Forest, Gradient Boosting, XGBoost, LightGBM, CatBoost. The pros and cons of each models are summarized in *Table 6.2*

Table 6.2 Overview of classifier (need to summarize this table)

No.	Binary Classifier	Advantages	Disadvantages
1	Logistic Regression	• East to interpret • Small number of hyperparameters • Overfitting can be addressed though regularization	• May overfit when provided with large numbers of features • Can only learn linear hypothesis functions • Input data might need scaling • May not handle irrelevant features well
2	Gaussian Naïve Bayes	• No training involved • Very little parameter tuning is required • Features do not need scaling	• Assumes that the features are independent, which is rarely true
3	k-Nearest Neighbors	• No training involved, easy to implement • Only one hyperparameter	• Need to find optimal number of K • Slow to predict • Outlier sensitivity
4	Support Vector Machine	• Accuracy • Works well on smaller cleaner datasets • Is effective when umber of dimensions is greater than the number of samples.	• Isn't suited to larger datasets as the training time can be high • Less effective on noisier datasets with overlapping classes
5	Decision Tree	• Easy to interpret • Work with numerical and categorical features. • Requires little data preprocessing • Performs well on large datasets • Doesn't require normalization	• Overfitting • Unable to predict continuous values • Doesn't work well with lots of features and complex large dataset
6	Random Forest	• Excellent predictive power • Requires little data preprocessing • Doesn't require normalization • Suitable for large dataset • Plenty of optimization options	• Overfitting risk • Parameter complexity • Limited with regression

No.	Binary Classifier	Advantages	Disadvantages
7	Gradient Boosting	• Excellent predictive accuracy • Can optimize on different loss functions • Provides several hyperparameter tuning options that make the function fit very flexible.	• Overfitting risk • Computationally expensive • Parameter complexity
8	XGBoost	• Regularization to prevent overfitting • Computational efficiency and often better model performance • Can handle missing value	• Difficult to interpret and visualize • Parameter complexity • Time consuming when dataset is large
9	LightGBM	• High speed, high accuracy • Can handle missing value and categorical value • Low memory usage • Compatibility with large dataset	• Parameter complexity • Overfitting risk
10	CatBoost	• Can handle missing value and categorical value • Work well with both small and large dataset • Can monitor loss function	• Prevent overfitting • Normally doesn't need to tune hyperparameters to gain better result

6.3 Baseline Model Evaluation

Step1: Firstly, we trained all the model with the baseline implementation, meaning all the hyperparameters of the models were left as the default value in the scikit-learn APIs. The evaluations are all conducted over the same Training set (70%) with 5Folds Cross Validation. The average performance on the test folds for each model can be found from *Table 6.3* and *Figure 6.1.* (Top 3 best score for metrics accuracy is highlighted in yellow*)*

Table 6.3 Average of baseline model performance on 5 test folds (dataset: Training set)

	model	fit_time	score_time	test_accuracy	test_precision	test_recall	test_f1_score	test_roc_auc
0	LR	0.077	0.007	0.868	0.000	0.000	0.000	0.637
1	GNB	0.004	0.008	0.754	0.233	0.373	0.286	0.629
2	KNN	0.009	0.096	0.853	0.204	0.038	0.064	0.535
3	SVC	6.286	0.379	0.868	0.000	0.000	0.000	0.499
4	DT	0.029	0.006	0.767	0.178	0.210	0.193	0.531
5	RF	0.630	0.090	0.858	0.278	0.047	0.080	0.603
6	GB	0.730	0.013	0.868	0.434	0.006	0.012	0.642
7	XGB	0.398	0.016	0.861	0.320	0.049	0.085	0.611
8	LightGB	0.097	0.014	0.867	0.358	0.019	0.036	0.626
9	CatBoost	6.673	0.022	0.868	0.449	0.025	0.047	0.638

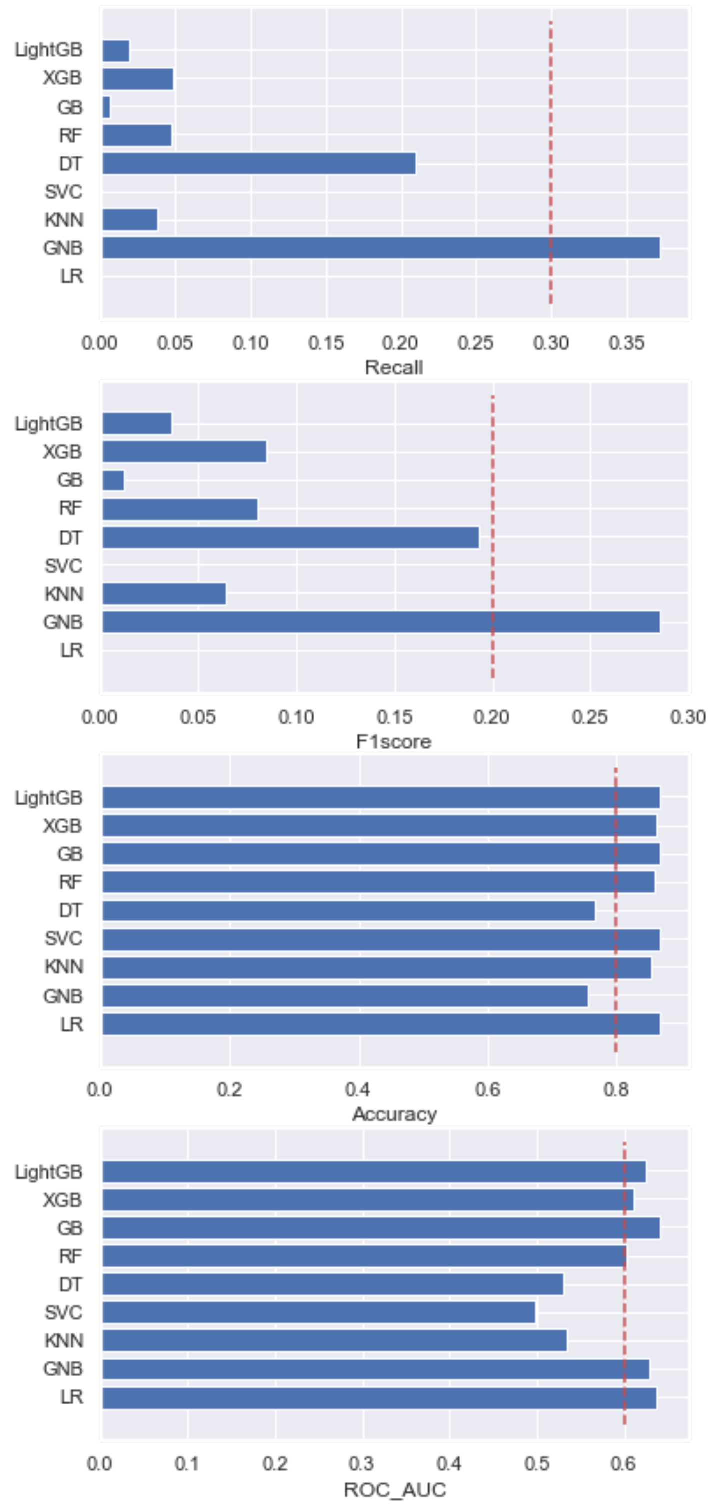

Figure 6.1 Bar plot for Average of baseline model performance on 5 test folds (dataset: Training set)

As we can see from above, most of the models have very good accuracy score (0.868 as the highest, 0.754 as the lowest). But only focusing on accuracy could be misleading in the case of this dataset due to the imbalance class problem. For instance, Logistic Regression results the top in accuracy, but Precision and Recall are all zero, meaning no positive class was predicted from the model. However, in this business case, we need to capture the Target enrollees (positive class) as many as possible. Therefore, instead of accuracy, we need to focus on Recall, as Recall measures the percentage of the positive class accurately predicted by the model out of all the positive classes in the dataset. Below you can find the detailed explanation of the rest of the 3 steps we took to define our model evaluation metrics, test model with Training and Testing set, record the performance result.

Step 2: On the top of the common metrics provided by scikit-learn, we created 2 new metrics specifically for this business case:

1) **Target in top 20%**: the number of positive classes the model is able to catch within the top 20% of the test dataset when you sort the test dataset by their prediction scores (predicted probabilities) in descending order.
2) **Target in top 50%**: the number of the positive classes the model is able to catch within the top 50% of the dataset when you sort the test dataset by their prediction scores (predicted probabilities) in descending order.
   *Instead of the predicted class, we use the prediction scores here.
   *There are 728 positive classes (Target) in the Testing set

Step 3: In this step, we fed the Training (70%) set to each model that we trained previously in Step 1, recorded the scores for Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target in top 50% and Cumulative Lift for each model. Additionally, we plotted the prediction scores distribution breakdown by Target and non-Target classes.

Step 4: Last, we fed the Testing (30%) set to each model that we trained previously in Step 1, recorded the scores for Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target in top 50% and Cumulative Lift for each model. Additionally, we have plotted the prediction scores distribution breakdown by Target and non-Target classes.

The performance result in Step 2 and Step 3 for each model could be found from this jupyter notebook Detecting Potential Candidate_Part1(01.01-06.04.03).ipynb. (part 06.03.02 Model Performance on Training and Testing set)

Here in this report, we will show you the result of Testing set. *Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target in top 50%*, these 6 metrics for the 10 models on Testing set could be found from *Table 6.4.*

Table 6.4 Model performance on Testing set (30%) (sorted by Target in top 20%)

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target in top 50%
LR	0	0	0	0.629	262	483
CatBoost	0.244	0.014	0.026	0.630	261	482
GB	0.286	0.003	0.005	0.645	259	493
GaussianNB	0.230	0.337	0.274	0.608	251	458
LightGB	0.333	0.016	0.031	0.618	237	473
XGB	0.220	0.037	0.063	0.599	225	456
RF	0.270	0.051	0.086	0.594	211	456
KNN	0.210	0.041	0.069	0.550	189	413
DT	0.179	0.220	0.197	0.533	186	391
SVC	0	0	0	0.507	152	364

As we can see despite the poor Recall and Precision score, Linear Regression (LR) performed the best in terms of capturing Target enrollee in the top 20% in the Testing set. CatBoost, Gradient Boosting (GB) are the second and third best performed models in terms of capturing Target enrollee in the top 20%. Gradient Boosting outperformed Linear Regression and CatBoost in terms of capturing Target enrollee in the top 50% from the Testing set. Overall, all models scored poorly in terms of Recall. This is due to the class imbalance issue that we have mentioned in previous section. In the case of this dataset, the negative class is roughly about 6 times more than the positive class. While models are fed with this type of imbalanced data, to ensure model accuracy, the models are going to predict the negative class as much as possible. Which could cause the increase of False Negative or the lack of positive class being predicted.

To address this issue, we will introduce oversampling method in the next section and see if the model performance will be improved after we train models with oversampled data.

6.4 Oversampling (SMOTE)

A problem with imbalanced classification is that there are too few examples of the minority class for a model to effectively learn. One way to solve this problem is to oversample the datapoints in the minority class. There are different approaches to achieve this, in this study we used **Synthetic Minority Oversampling Technique**, or SMOTE for short. This technique was described by Nitesh Chawla, et al. in their 2002 paper named for the technique titled "SMOTE: Synthetic Minority Over-sampling Technique." SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then *k* of the nearest neighbors for that example are found (typically *k=5*). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

In this section we applied SMOTE only to the train folds in the 5 Folds Cross Validation using the Training set (70%), because we want the test folds to approximate the data distribution in practice. After applying SMOTE to each train fold, the ratio of the majority class (non-Target enrollee) and minority class (Target enrollee) decreased from 6:1 to 1:1.

Then we tested the SMOTE – 5 Folds CV models on the Training set (70%) and Testing set (30%), you can find more details of the test results from this jupyter notebook Detecting Potential Candidate_Part1(01.01-06.04.03).ipynb. (part 06.04.02 Model Performance on Training and Testing set After Oversampling)

6.4.1 Model Comparison after SMOTE

In this report we will show you the result of Testing set.
Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target in top 50%, these 6 metrics for the 10 models on Testing set could be found from *Table 6.5*

Table 6.5 Model performance after SMOTE on Testing (30%) set (sorted by Target in top 20%)

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target in top 50%
GB	0.381	0.033	0.061	0.633	260	475
LR	0.192	0.521	0.281	0.623	253	480
GaussianNB	0.151	0.674	0.246	0.598	251	441
CatBoost	0.315	0.040	0.071	0.623	247	481
LightGB	0.309	0.029	0.053	0.628	236	484
XGB	0.257	0.038	0.067	0.600	215	450
SVC	0.153	0.647	0.247	0.580	212	440
RF	0.237	0.065	0.102	0.588	206	450
KNN	0.144	0.455	0.218	0.541	182	391
DT	0.162	0.195	0.177	0.520	171	387

As we can see from above, after we trained the models by applying SMOTE to each train fold split in the Training set, and test the model on the Testing set, Recalls of some of the models drastically improved. For instance, Logistic Regression (LR) and Support Vector Classifier (SVC) before oversampling, predicted zero Target enrollee (Recall was zero). However, after oversampling, their Recall became the second (SVC) and the third (LR) highest comparing to other models. Gaussian Naïve Bayes (GNB) has the best Recall, scored 0.674. What about other metrics? How much did they improve after oversampling? Please find the answer *in Table 6.6* below.

Table 6.6 Metrics percentage change after SMOTE

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target in top 50%
LR	∞	∞	∞	-0.95	-3.44	-0.62
GaussianNB	-34.35	100.00	-10.22	-1.64	0.00	-3.71
KNN	-31.43	1009.76	215.94	-1.64	-3.70	-5.33
SVC	∞	∞	∞	14.40	39.47	20.88
DT	-9.50	-11.36	-10.15	-2.44	-8.06	-1.02
RF	-12.22	27.45	18.60	-1.01	-2.37	-1.32
GB	33.22	1000.00	1120.00	-1.86	0.39	-3.65
XGB	16.82	2.70	6.35	0.17	-4.44	-1.32
LightGB	-7.21	81.25	70.97	1.62	-0.42	2.33
CatBoost	29.10	185.71	173.08	-1.11	-5.36	-0.21

*∞ represents where the previous metric (before oversampling) was zero
* red text represents where metrics decreased after oversampling

From *Table 6.6* we can see despite the drastic improvement in Recall, oversampling is more likely to cause a decline in other metrics. The Top 3 highest number of *Target in top 20%* and *Target in top 50%* both dropped after oversampling. Previously, before applying SMOTE, Logistic Regression was able to capture 262 Target enrollees out from 1101 data samples of Testing set, Gradient Boosting was able to capture 493 Target enrollees out from 2754 data samples of Testing set. After applied SMOTE, the highest number for the 2 metrics decreased to 260 (GB) and 480 (LR) respectively. Let's take a closer look at how did Top 3 highest number of *Target in top 20%* and *Target in top 50%* dropped in *Table 6.7*.
(*There are 5508 data samples in Testing set and the top 20% from Testing set contains 1101 data samples, top 50% from Testing set contains 2754 data samples)

Table 6.7 Top 3 best result of capturing Target in top 20% and 50% before vs. after SMOTE

Before SMOTE Target in top 20%	After SMOTE Target in top 20%	Before SMOTE Target in top 50%	After SMOTE Target in top 50%
262 (LR)	260 (GB)	493 (GB)	480 (LR)
261 (CatBoost)	253 (LR)	483 (LR)	475 (GB)
259 (GB)	251 (GNB)	482 (CatBoost)	441 (GNB)

6.4.2 Model and key Metric Selection

So, now we want to ask two questions.
First one: which model is the best for this business case? Shall we choose the model has the highest Recall after oversampling? Or shall we go with the model that can capture the most Target enrollee in Top 50% from the Testing set?

Let's break this question down by doing a comparison calculation.
If we choose the model that provides the best Recall after oversampling, then it should be model Gaussian Naïve Bayes as it has the best Recall 0.674. Let's assume we use this model and predicted the result for our Testing set and get the confusion matrix result:

Confusion matrix of model Gaussian Naïve Bayes after applying SMOTE

	Predicted Target	Predicted non-Target
Target	TP: 491	FN:237
non-Target	FP: 2770	TN: 2010

In this case, we would provide the final Predicted Target list to the recruiting team for further action, for instance, sending out email to this Target list. From the confusion matrix, we know this Target list would be a list that contains 3261 (491 + 2770) potential enrollees while 491 of them are the real Target enrollee. If we assume that the cost per email sent is $1 then we would eventually spend $3261 and reach out to 491 real Target enrollees. The cost per Target is $6.6.

Now let's take a look at the model that captured the most Target enrollees in the top 50% from the Testing data. The model achieved this was Gradient Boosting before oversampling, which captured 493 Target enrollees in the top 50% of Testing set. In this case what we would do is provide the top 50% list to the recruiting team. This list contains, as we explained previously, 2754 enrollees. Which means the recruiting team would only need to send out 2754 emails and they would be able to reach out to 493 real Target enrollees. The cost per Target is $5.6.
Let's say Company XYZ's recruiting team has a KPI of aiming to reach out to no less than 300 Target enrollees with minimum costs, which model is more efficient and economical in this case? The answer is obvious. Therefore, we could come to 3 conclusions after the study in this section:

1. Oversampling (SMOTE) is very effective in terms of improving models' ability to predict Positive classes. But when the dataset has high degree of feature overlap

between classes, applying oversampling method may consequently cause an increase in False Positive classes (Precision will decrease).

2. Depends on the approach we take after the prediction the increase of False Positive classes may cause negative impact in ROI.
3. In this business case, metrics Target in top 50% is our key metrics in terms of model comparison.
4. Model Gradient Boosting before oversampling would be the best solution to this business case.

Lastly, in this section let's take a look at how SMOTE method affected performance of the best model Gradient Boosting. The result[3] of *Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target% in top 20%, Target in top 50%, Target% in top 50%*, before and after SMOTE oversampling are provided in *Table 6.8.*

Cumulative Lift plot before and after SMOTE oversampling are provided *in Figure 6.2.* Prediction scores distribution before and after SMOTE oversampling are provided in *Figure 6.3.*

Table 6.8 Gradient Boosting model performance on Testing set before and after SMOTE

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target% in top 20%	Target in top 50%	Target% in top 50%
GB before	0.286	0.003	0.005	0.645	259	35.6%	493	67.7%
GB after	0.381	0.033	0.061	0.633	260	35.7%	475	65.2%

[3] Customized metrics (Target% in top N%), Lift, Cumulative Lift's calculation formula could be found in Appendix

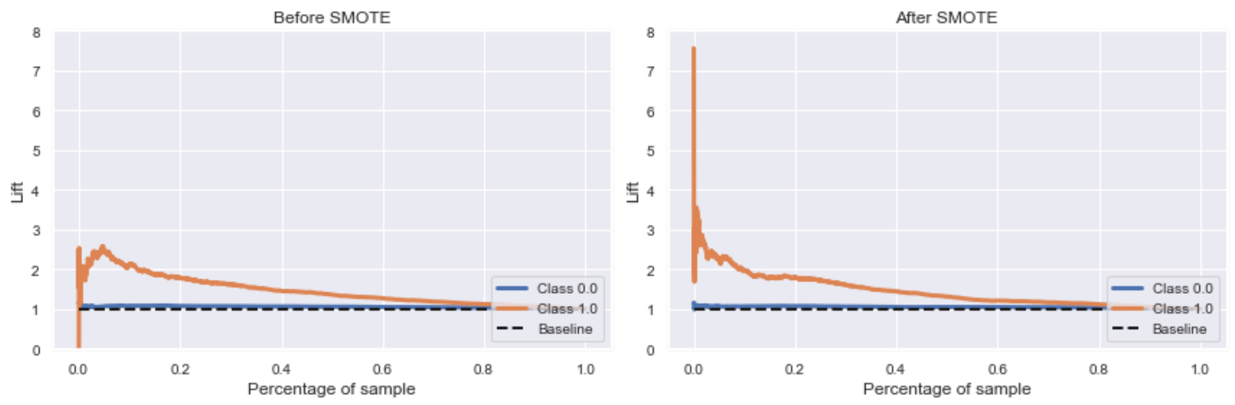

Figure 6.2 Gradient Boosting model Cumulative Lift before and after SMOTE oversampling

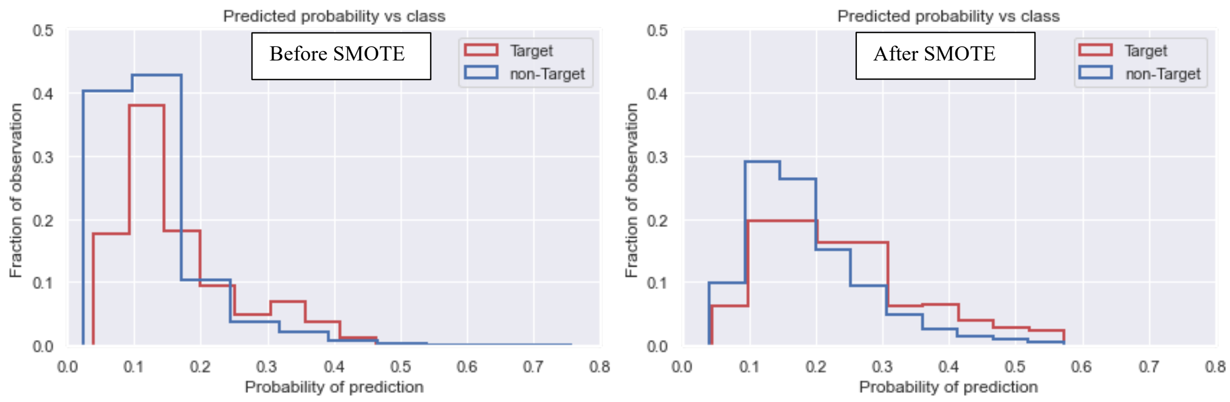

Figure 6.3 Gradient Boosting model prediction scores distribution plot before and after SMOTE oversampling

As we can see from above, oversampling did improve our model's ability to predict Target enrollee as the count of probabilities (predicted by model) over 0.5 slightly increased comparing to the distribution before oversampling, this explained the increase in Recall. Also, we observed a decline in the number of False Positive after oversampling, this explained the increase in Precision. However, it's still obvious that our model is having difficulties in terms of separating the two classes as the two classes can be seen to have almost indistinguishable distributions across prediction scores. Furthermore, our key metric *Target in top 50%* dropped from 493 (67.7%) to 475 (65.2%) after oversampling.

After achieved the best result from model Gradient Boosting (before SMOTE), our study didn't end here. As we are still wondering if that is the best model for us. If we take a look back at the models we selected in our list, some of them as a matter of fact don't require us to fill in missing value or encode categorical variables. For instance, Xgboost Classifier is able to handle missing value and CatBoost Classifier is able to handle both missing value and categorical variables. So here we want to ask the second question: will the 2 models' performance get improved if we train Xgboost Classifier without filling in missing values, or train CatBoost Classifier with our original dataset (with missing values and categorical variables)? If they do get improved, is it possible that we build a model that will outperform the best model we have so far? We will explore more about this in the next 2 sections.

6.5 Experiment with Model Xgboost Classifier

In this section:

1. We used the same data set which contains 18,359 rows and 14 columns, and grouped the variables, corrected their data type. The process of grouping and data type correction is exactly the same with the one we explained in Section 4.2 Data Processing, the only difference is we didn't fill in the missing values.
2. Then we preprocessed the data taking the same steps that we took in Section 6.1 Data Preprocessing and Feature Selection.
3. Trained our model Xgboost Classifier with default hyperparameter settings with 5 Folds Cross Validation on Training set (70%) and tested the model on the Testing set (30%).
4. Recorded *Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target% in top 20%, Target in top 50%, Target% in top 50%.*
5. Compared the recorded result with the result we obtained from Xgboost Classifier in Section 6.3 Baseline Model Evaluation.

The more details of step1~ step 5 are provided in this jupyter notebook: Detecting Potential Candidate_Part2_Model XGB.ipynb. Please find the performance comparison in step 5 from *Table 6.9*, *Figure 6.4* and *Figure 6.5*

Table 6.9 Xgboost model performance on Testing set

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target% in top 20%	Target in top 50%	Target% in top 50%
Null imputed	0.220	0.037	0.063	0.599	225	30.9%	456	62.6%
Null not imputed	0.263	0.060	0.098	0.611	244	33.5%	454	62.4%

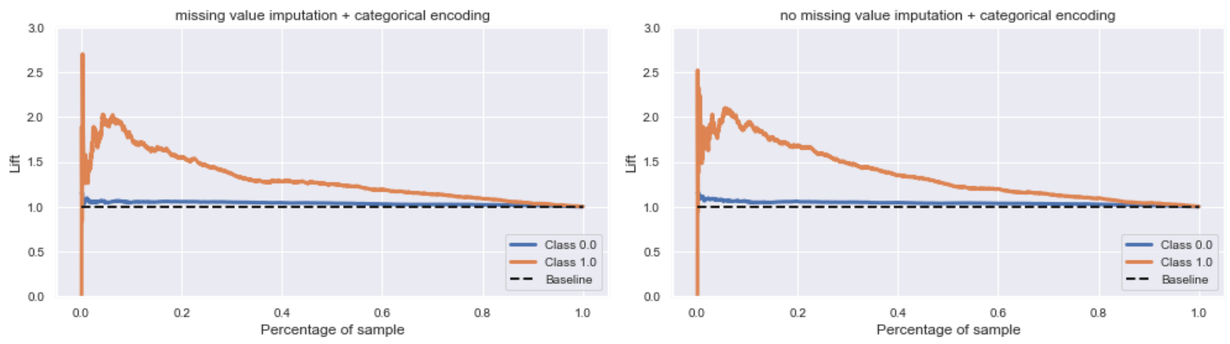

Figure 6.4 Xgboost model Cumulative Lift with and without missing value imputation

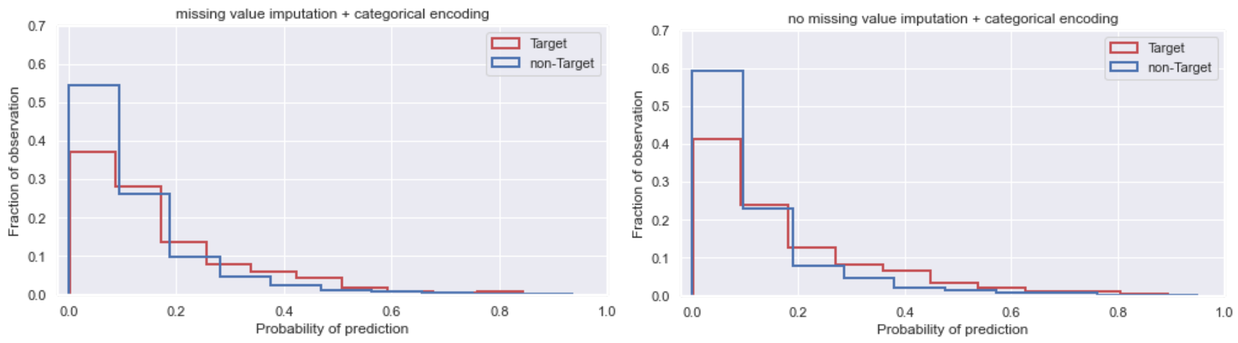

Figure 6.5 Xgboost model prediction scores distribution with missing value imputation and no missing value imputation

The results above show that train Xgboost Classifier without missing value imputation did enhance the model performance in multiple ways. Besides metric *Target in top 50%*, we observed the improvement in all the other metrics. However, our conclusion after this experiment is that this model does not perform better than Gradient Boosting model before SMOTE, which could capture 259 (35.6%) and 493 (67.7%) Target enrollees out of the top 20% and top 50% of the Testing set respectively.

6.6 Experiment with Model CatBoost Classifier

In this section:

1. We used the same data set which contains 18,359 rows and 14 columns. We replaced the missing value in the categorical variables with value ‘None’.
   *Although CatBoost Classifier is supposed to handle null values, but categorical variables cannot contain null values.
2. Then we preprocessed the data taking the same steps that we took in Section 6.1 Data Preprocessing and Feature Selection.
3. Trained our model CatBoost Classifier with default hyperparameter settings with 5 Folds Cross Validation on Training set (70%) and tested the model on the Testing set (30%).
4. Recorded *Precision, Recall, F1score, ROC_AUC, Target in top 20%, Target% in top 20%, Target in top 50%, Target% in top 50%.*
5. Compared the recorded result with the result we obtained from CatBoost Classifier in Section 6.3 Baseline Model Evaluation.

The more details are provided in this jupyter notebook: Detecting Potential Candidate_Part3_Model CatBoost.ipynb. Please find the performance comparison in step 5 from *Table 6.10*, *Figure 6.6* and *Figure 6.7.*

Table 6.10 CatBoost model performance on Testing set

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target% in top 20%	Target in top 50%	Target% in top 50%
Data processed	0.244	0.014	0.026	0.630	261	35.9%	482	66.2%
Data Not processed	0.500	0.005	0.011	0.662	275	37.8%	514	70.6%

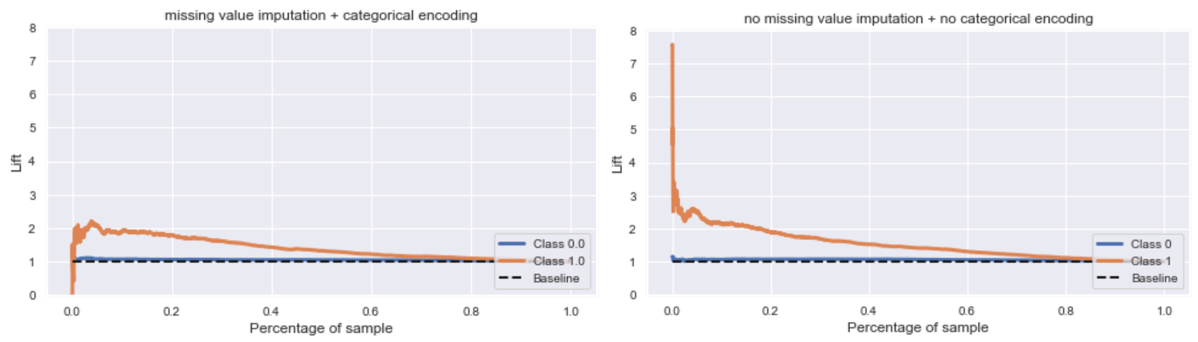

Figure 6.6 CatBoost model Cumulative Lift with data processing and no data processing

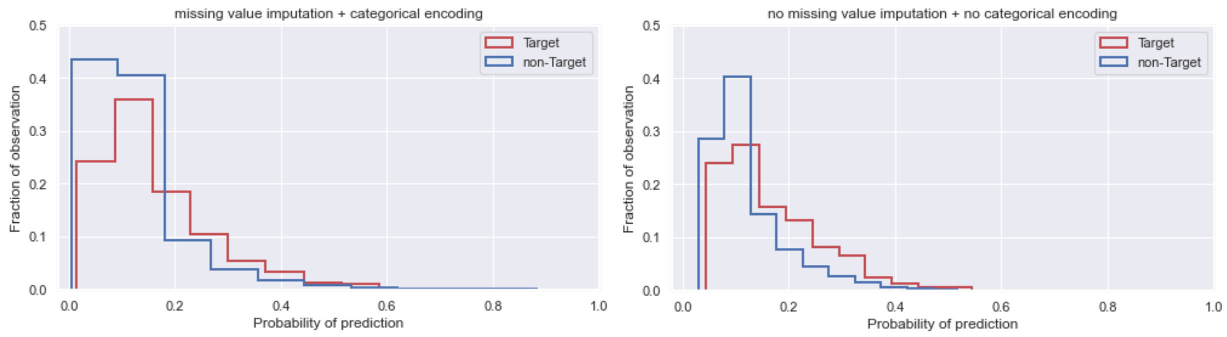

Figure 6.7 CatBoost model prediction scores distribution with data processing and no data processing

The result above show that train CatBoost Classifier without imputing missing value or encoding categorical variable, did enhance the model performance in terms of reducing False Positive. Besides metric *F1score and Recall,* we can see a significant improvement in Cumulative Lift plot and all the other metrics. The most significant improvement is, this model enabled us to capture **275**, which is 37.8% of Target enrollees and **514**, which is 70.6% of Target enrollees, out of the top 20% and top 50% of the Testing set respectively. This is the best achievement we had so far.

7 Final Model Selection and Hyperparameters Tuning

Through Section 6.6 Experiment with Model CatBoost Classifier, we should be able to identify the best solution for our business case this time is

1. Firstly, feed Catboost Classifier with the original dataset without any missing value imputation or categorical variable encoding.
2. We then train the model with 5 Folds Cross Validation on Training set (70%).

As this solution outperformed all the other solution we tested in this study, furthermore it would save us lots of time by cutting off all the data processing part.

After identifying CatBoost Classifier as our best model, we applied hyperparameter tuning via random search Cross Validation (3 Folds) to the model and obtained our optimized model by adjusting 3 hyperparameters: iterations, depth and learning_rate to [iterations=988, depth=2, learning_rate=0.072]. A comparison of model performance with and without hyperparameter tuning is provided in *Table 7.1, Figure 7.1 and Figure 7.2.*

Table 7.1 Model performance on Testing set with and without hyperparameter tuning

Model	Precision	Recall	F1score	ROC_AUC	Target in top 20%	Target% in top 20%	Target in top 50%	Target% in top 50%
CatBoost	0.500	0.005	0.011	0.662	275	37.8%	514	70.6%
CatBoost_tuned	0.571	0.005	0.011	0.664	283	38.9%	515	70.7%

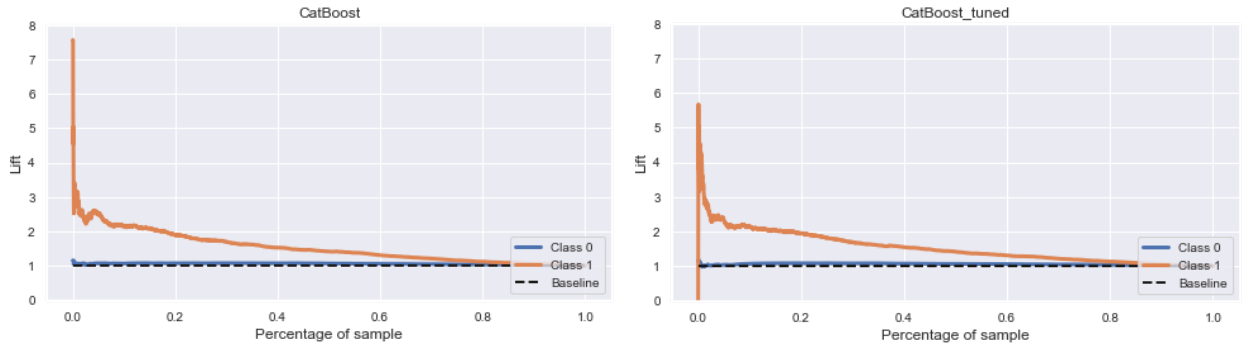

Figure 7.1 CatBoost model Cumulative Lift before and after hyperparameter tuning

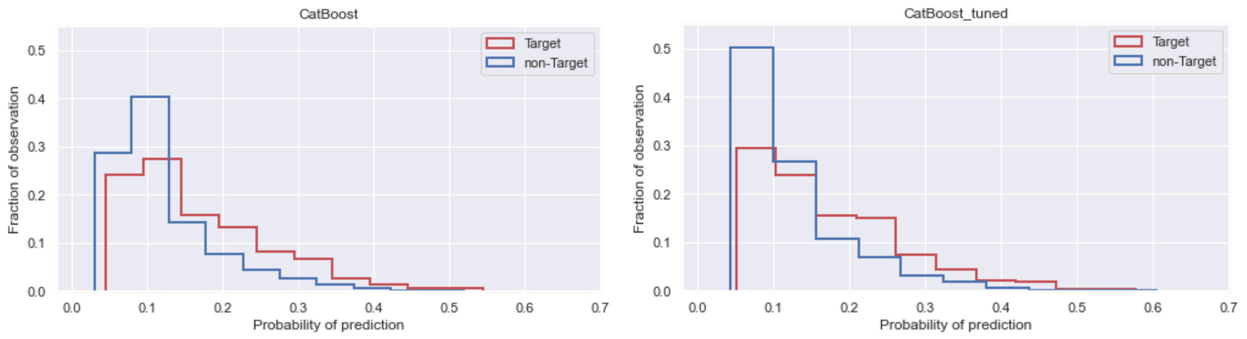

Figure 7.2 CatBoost Model prediction scores distribution before and after hyperparameter tuning

We can see from *Figure 7.2*, after tuning hyperparameters there was no significant change in prediction scores distribution. On the other hand, the result of *Table 7.1* is showing after hyperparameter tuning, Target enrollee captured in top 20% and top 50% from the Testing set **increased** by 2% and 0.1% due to the improvement in Lift and Cumulative Lift (*Figure 7.1*). And with tuned CatBoost model, we could capture almost 71% of Target enrollees from only 50% of the Testing set. Thus, we identify the tuned CatBoost Classifier as our final model. The Lift and Cumulative Lift table before and after tuned was provided in *Table 7.2*. ROC curve of the final model was provided in *Figure 7.3*, followed by *Figure 7.4* Feature importance based on tuned CatBoost Classifier.

Table 7.2 Lift, Cum.Lift table before (left) and after tuned (right)

Decile	Target count	Non Target count	Lift	Cum_Lift
1st	156	395	2.992	2.159
2nd	118	433	2.065	1.891
3rd	95	455	1.582	1.697
4th	74	477	1.175	1.527
5th	70	481	1.103	1.414
6th	56	495	0.857	1.307
7th	45	506	0.674	1.208
8th	36	514	0.531	1.119
9th	34	517	0.498	1.047
10th	43	508	0.641	1.001

Decile	Target count	Non Target count	Lift	Cum_Lift
1st	156	395	2.992	2.145
2nd	127	424	2.269	1.945
3rd	86	464	1.404	1.692
4th	80	471	1.287	1.544
5th	66	485	1.031	1.417
6th	55	496	0.840	1.307
7th	48	503	0.723	1.214
8th	40	510	0.594	1.131
9th	28	523	0.406	1.048
10th	42	509	0.625	1.001

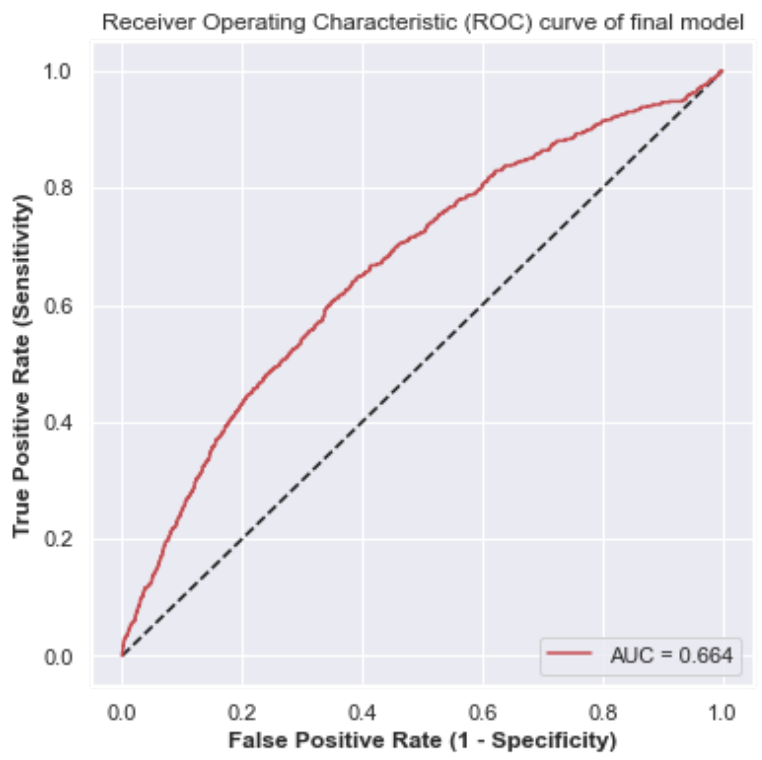

Figure 7.1 Receiver Operating Characteristic (ROC) curve of final model

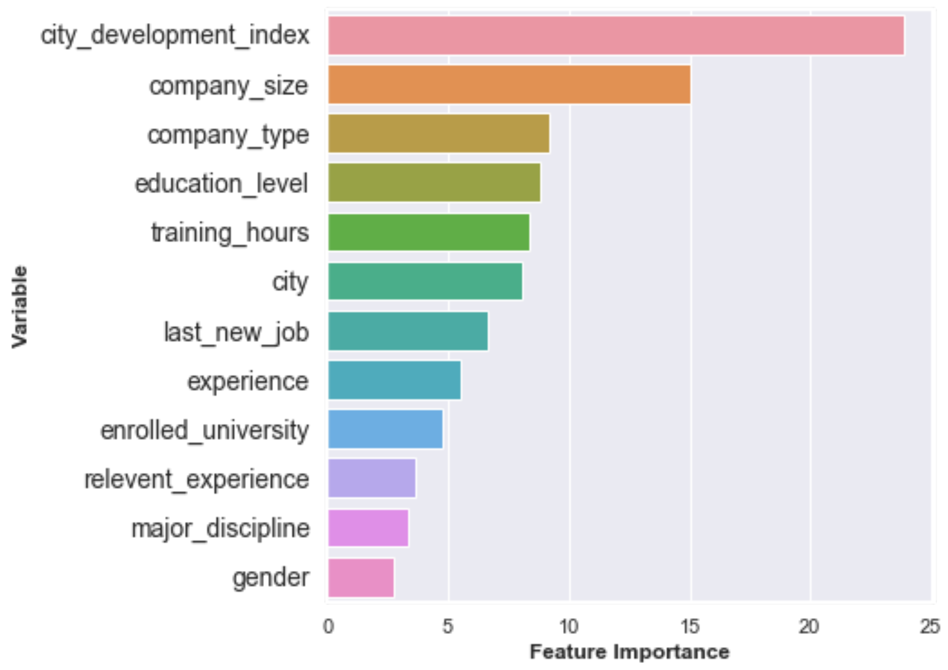

Figure 7.2 Feature importance for tuned Catboost Classifier

8 Conclusion

8.1 Dataset

After exploring the dataset provided by Company XYZ, we discovered there are a lot of feature similarities in Target and non-Target enrollees. This added challenges to the machine learning process as highly similar instances in datasets that have different target classes normally lead to poor model performance. On top of that, the dataset's classes are imbalanced at a ratio of 6:1. This posed another challenge for predictive modeling as most of the machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class, which is the Target enrollee in this case.

8.2 Models

1. In this study we selected 10 classification algorithms that contain probabilistic classifier (Logistic Regression, Gaussian Naïve Bayes), Instance-based learning algorithm (K-Nearest Neighbor), Support Vector Classifier, Tree-based models (Decision Tree, Random Forest, Gradient Boosting) and Optimized gradient boosting algorithms (XGBoost Classifier, LightGBM Classifier, CatBoost Classifier). Firstly, trained them on the Training set that contains 70% of the entire dataset, applied 5 Folds Cross Validation to the training process. While averaging the test folds performance, we noticed that almost all of the models

performed reasonably good in terms of accuracy (0.754 ~ 0.868) but extremely poor in terms of Recall (0 ~ 0.373). Recall remained low when tested models on Testing set (contains the rest 30% instances of the dataset). Gaussian Naïve Bayes generated the best result with this dataset in terms of Recall (0.337).

2. To enhance the model performance, we applied data oversampling method **Synthetic Minority Oversampling Technique** to each train fold when conducting 5Fold Cross Validation, decreased the imbalance class ration from 6:1 to 1:1. This method did significantly improve the models' ability to predict Target enrollee (minority class, positive class), however it caused a drop in other metrics such as Precision, as the number of non-Target enrollees being misclassified as Target enrollee (False Positive) also increased together with the increase of correctly classified Target enrollee.
3. Depends on the approach we take after the prediction the increase of False Positive classes may cause negative impact in ROI.
4. After comparing metrics Recall with our customized metrics *Target in top 20%* and *Target in top 50%*, we came to the conclusion that in this business case, metrics Target in top 50% is our key metrics as it helps us to identify the model with the best ROI (Reach to the most true Target enrollees with the least financial and human resource investment).
5. In terms of our customized metrics, models that enable training with missing values or categorical variables performed significantly better without data processing (missing value imputation, categorical variable encoding). CatBoost Classifier trained without data processing outperformed all the other models and was identified as our final model.

8.3 Feature Work

To solve the high degree of feature overlap between the two classes in this dataset, a good solution we would like to exam with is to add more features/variables in the dataset if possible. Features that will distance the distribution of the two classes. Currently, age, income and job title/level information are not included in the dataset but are worthy of further study in this business case.

The approach we took for oversampling was SMOTE method, there are some other resampling techniques we have not explored yet would like to dive deeper in the future. Such as ADASYN oversampling method, undersampling method (decrease the number of majority class to balance the classes), adjusting parameters of these resampling techniques we are able to set the target ratio of minority class to majority class (can make it to 2:1 as instance).

References

1. https://www.kaggle.com/aswathrao/hr-analysis?select=train.csv (assessed Aug. 12, 2020)

2. John D. Kelleher, Brian Mac Namee, Aoife D'Arcy. (2015) *Fundamentals of Machine Learning for Predictive Data Analytics*: *Algorithms, Worked Examples, and Case Studies.* The MIT Press.

3. SMOTE for Imbalanced Classification with Python. URL: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ (assessed Sep. 20, 2020)

4. Pablo Guevara *Load Default Predictive Modeling A Case Study of Classification Algorithms*. URL: https://github.com/pandrewg/Lending-Club/tree/master/analysis (assessed Oct. 22, 2020)

5. https://en.wikipedia.org/wiki/Kaggle (assessed Nov. 11, 2020)

6. https://pixabay.com/vectors/choice-rh-hr-candidate-best-eq-4518660/(assessed Nov. 1, 2020)

Appendix

1. Target% in top 20%: rank the predictions made for the Testing set in descending order by prediction scores and then divide them into deciles. A decile is a group containing 10% of a dataset. Top 20% means the top 2 deciles.
   Target% in top 20% is defined as:

$$\text{Target\% in top 20\%} = \frac{\text{Targets in top 2 deciles}}{\text{total Targets in Testing set}} = \frac{\text{Targets in top 2 deciles}}{728}$$

2. Target% in top 50% is defined as:

$$\text{Target\% in top 50\%} = \frac{\text{Targets in top 5 deciles}}{\text{total Targets in Testing set}} = \frac{\text{Targets in top 5 deciles}}{728}$$

3. The Lift at each decile dec is defined as:

$$\text{lift (dec)} = \frac{\text{\% of Targets in decile dec}}{\text{\% of Targets in Testing set}} = \frac{\text{Targets in decile dec}/728}{\text{\% of Targets in Testing set}}$$

4. Cumulative Lift at each decile dec is defined as:

$$\text{cumulative lift (dec)} = \frac{\text{\% of Targets in all deciles up to dec}}{\text{\% of Targets in Testing set}} = \frac{\text{Targets in all deciles up to dec}/728}{\text{\% of Targets in Testing set}}$$