Subscriber Prediction

Classifying Potential Subscribers in a Bank Telemarketing Campaign

Milestone: Project Report

Group 17

Pingting Liu

Weikun Zhuang

Percentage of Effort Contributed by Student 1:_____50%_____

Percentage of Effort Contributed by Student 2:_____50%_____

Signature of Student 1:_____Pingting Liu_____

Signature of Student 2:_____Weikun Zhuang_____

Submission Date:_____12/13/2021_____

Table of Contents

Problem Setting

Despite the ubiquity of online campaigns, offline telemarketing that reaches directly to clients still remain one of the most common ways for companies to advertise products or services. It is effective because it can get instant reaction and feedback from customers. It also makes it easy to follow up clients that showed significant interests. Due to the sheer volume of client data, it's important to utilize data mining techniques to detect and predict customer behavior based on analyzing the relationships between parameters. It helps companies gain insights on how to direct marketing campaigns to clients more efficiently and to generate new market opportunities.

Problem Definition

We plan to create several classification models to predict if a client of a banking institution will subscribe a term deposit through telemarketing campaigns. In doing so, we hope to explore what are some of the main characteristics of the clients that will subscribe? What kind of models will better help identify those clients? What are some of the important features that contribute to deposit subscription? To answer these questions, we'll perform data exploration, visualization and train and test the model applying various data mining techniques.

Data Source & Description

The dataset we'll be using in this project comes from UCI machine learning repository. The original source of this dataset is a paper bank telemarketing. Both sources are cited below.

- Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. URL to dataset: https://archive.ics.uci.edu/ml/datasets/Bank+Marketing# , accessed Sep. 16th, 2021
- [Moro et al., 2014] S. Moro, P. Cortez and P. Rita. A Data-Driven Approach to Predict the Success of Bank Telemarketing. Decision Support Systems, Elsevier, 62:22-31, June 2014

We'll be using the file "bank-full.csv". The dataset contains 45211 observations and 17 variables (including 1 outcome variable – "y"). It contains information of clients that were contacted by the marketing campaign. There are 7 numerical variables – age, average yearly balance, the last contact day of the month, duration of the last contact, number of contacts performed on the client, number of days that passed by after the client was last contacted, and previous contacts. The remaining 10 variables are categorical variables – job, marital status, education level, whether the client has credit in default, housing and personal loan, last contact communication type, last contact month, outcome of the previous marketing campaign, and lastly our output variable – whether the client has subscribed or not. Here are some observations and preliminary processing for our data.

Data Exploration

Preliminary Exploration

Imbalanced dataset

In real business scenarios, most of the time, the marketing campaign couldn't succeed. Similarly, in our dataset, we found that the non-subscribers take up a large proportion. We counted the values of 'yes' and 'no' in our outcome variable and found out that there are 39922 no-subscribes and only 5289 subscribers, which is only 12%. The imbalanced dataset will affect the predictive performance of the model, specifically for the minority class. Therefore we will use oversampling technique to balance the dataset before building model. We will also encode the target variable as a binary 0/1 (No/Yes) variable for convenience.

Summary Statistics

Running the summary statistics on each numeric variables (partial output shown below), we found out that some of the variables have quite a wide range and we will further explore them in later stage.

	age								balance								
	count	mean	std	min	25%	50%	75%	max	count	mean	std	min	25%	50%	75%	max	count	mean
y																	
no	39922.0	40.838986	10.172662	18.0	33.0	39.0	48.0	95.0	39922.0	1303.714969	2974.195473	-8019.0	58.0	417.0	1345.0	102127.0	39922.0	15.89229
yes	5289.0	41.670070	13.497781	18.0	31.0	38.0	50.0	95.0	5289.0	1804.267915	3501.104777	-3058.0	210.0	733.0	2159.0	81204.0	5289.0	15.15825

Variables with an "unknown" category

There are some "unknown" records in the dataset and we checked these variables for the percentages of "unknown" so that we can decide to either impute or drop.

```
The percentages of "unknown" values in each categorical variables
job 0.64%
marital 0.00%
education 4.11%
default 0.00%
housing 0.00%
loan 0.00%
contact 28.80%
month 0.00%
poutcome 81.75%
```

Skewness of Data

Data is considered as skewed when the curve appears distorted or skewed either to the left or to the right, in a statistical distribution. But in a normal distribution, the graph appears symmetry meaning there are about as many data values on the left side of the median as on the right side. The skewed dataset can be misleading because the most common values in the distribution might not be near the mean. Additionally, skewed data can affect which types of analyses are valid to perform. We used "skew" from scipy.stats library and computed the skew number of 6 numerical variables. If the result equals to 1, it means the variable is not too skewed. For example, the skew of "age" is 0.68, the skew of "pdays" is 2.62, these variables only show a little skewness. But for variables like "previous", the skew is 41.85 and the skew of "balance" is 8.36. In the next phase, we should use some algorithm to change the skew.

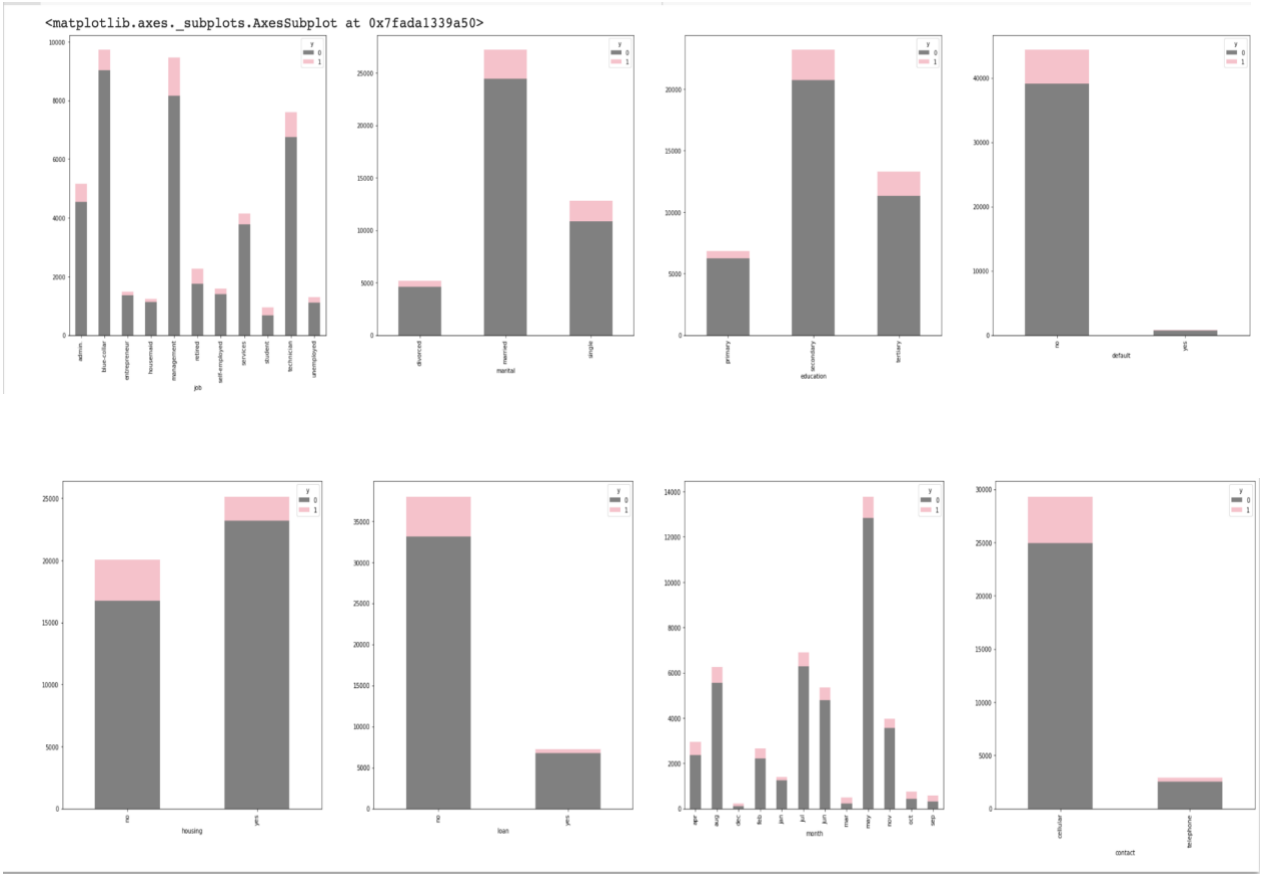
```
age          0.684818
balance      8.360308
day          0.093079
duration     3.144318
campaign     4.898650
pdays        2.615715
previous    41.846454
y            2.383480
dtype: float64
```

Correlation between variables

After creating correlation matrix between numerical variables and plot corresponding heatmap and pairplot, we didn't find any pairs of highly correlated variables that need extra attention.

Data Visualization

Part I: Categorical Variables



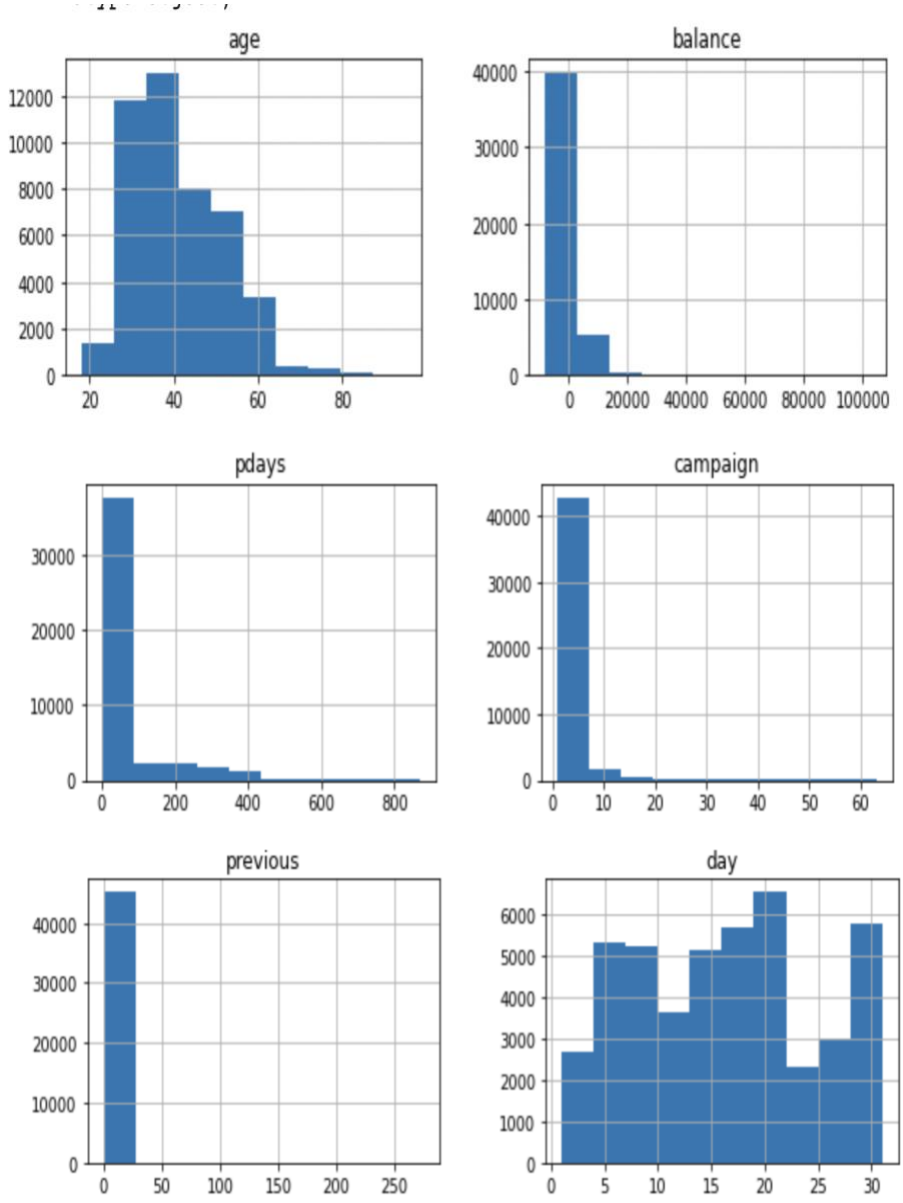
Stacked bar charts allow us to not only visualize how data is distributed in each category of every category variable in the dataset, but also add a layer to inspect how our target variable "y" (whether the client will subscribe to a term deposit or not) is distributed throughout all the categories. Above is a stacked bar chart for all 8 categorical variables in our data set. The variables, from left to right and top to bottom, are "job", "marital status", "education level", "credit in default (yes or no)", "housing loan (yes or no)", "personal loan (yes or no)", "contact month" and "contact type". Then, y = "yes" (people who subscribed) was shown in pink and "no" (people who didn't subscribe) was shown in grey.

After we drew the stacked bar chart, we had some insights as below:

1. For “job”, we could find that the most common jobs in this dataset are blue-collar, management and technician. Moreover, among these three categories, the proportions of clients who subscribed are the highest.
2. For “marital status”, the category “married” takes up the largest percentage and its proportion of subscribed clients seems to be the largest.
3. For “education”, clients with a secondary education are the most common and they seem to be more willing to subscribe to a term deposit through the campaign.
4. For the variable “default”, although it may be reasonable to think that clients with credits in default will not subscribe to a term deposit and it hardly shows in the bar chart, there are still 50 or so observations with “default” = “yes” that has y = “yes”.
5. There are two categories in the “housing” variable - meaning with or without housing loans. Clients WITH a housing loan take up a larger proportion in the dataset. However, clients without a housing loan have more subscribers, which is reasonable considering that people without housing loans probably have more money on their hands to do term deposit.
6. For the “loan” category, it’s very clear that customers without loans have more possibilities to subscribe to a term deposit with the bank - similar to the “housing” variable.
7. For the “month” category, May is the peak time that this bank successfully sells term deposits to clients. During winter and early spring, the number of clients who were contacted and purchased the subscription seems to decline.

Part II: Numerical Variables

A. Histograms



From the histogram, we wanted to explore the distributions of numerical variables in our dataset and find out if we need to take further steps to process them before fitting models. Below are some important insights:

1. For “age”, “campaign”, “previous” and “day”, we found out that they all have skewed distributions. For example, most of the customer’s ages are concentrated between 25-55 years old, they will give more weight in the model training step. Therefore, if we test the model, and most of customer’s age in test datasets are above 55 years old, we won’t get good performance for our model, so we should change the skewness in the next step.

2. For “campaign” and “pdays”, most of the clients in this dataset are first contacts, meaning they’ve never been contacted before or have only been recently contacted. This will give more realistic meaning to the model that we will build because it’s more useful to use it on first-time clients to predict if they will buy the bank’s product.
3. For the “balance” and “pdays” variables, there are some negative numbers in these two variables. For example, most of the customers’ balances are negative, but the negative number does not fit with some algorithm and the distribution is also unbalanced. So in the next step, we will try to normalize this category and make the distribution between 0~1.
4. “Day” variable is just every day (1-31) in a month to indicate the exact day that the client was contacted.

Data Preprocessing

Dropping Columns

1. The attribute "duration" highly affects the target variable. If duration = '0', then y = '0' and else means y= "1"). Drop this variable to have a realistic predictive model.
2. Drop column “contact” and “poutcome” because the “unknown” proportions are too high. If we use other records to replace, it will affect our model’s performance.

Imputing “unknown” values

For variables “job” and “education” with relatively low “unknown” value proportions, we used their respective majority class (“blue-collar” and “secondary”) to impute the “unknown” values. After this step, all the categories of categorical variables doesn’t contain “unknown” values which do not produce meaningful insights.

Changing skewness using “boxcox” transformation

In the scipy.stats library, there is a function called boxcox, it could help us to change the skewness for the variables which are skewed but without negative values. Therefore, we could use this function on “age”, “day”, “campaign” and “previous”.

Scaling numerical data

We have 6 numerical variables and we will scale them using the MinMaxScaler() function to scale them to the range (0,1). This is because a. it takes care of negative values and b. scaling variables between this range will help the logistic regression model (which we are going

to explore later) performs a bit better and since it won't affect models like KNN, Decision Trees, we are sticking with this method.

	age	balance	day	campaign	pdays	previous
0	0.731404	0.092259	0.195769	0.0	0.0	0.0
1	0.570729	0.073067	0.195769	0.0	0.0	0.0
2	0.395437	0.072822	0.195769	0.0	0.0	0.0
3	0.609772	0.086476	0.195769	0.0	0.0	0.0
4	0.395437	0.072812	0.195769	0.0	0.0	0.0

Creating dummy variables for categorical data

For some statistics models, like logistic regression, it couldn't train the dataset directly with categorical variable, so we use dummy function to change categorical variables to numerical variables, then combine with the numerical variables, we can get the final clean dataset which prepared for the feature selection and model training.

	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	job_services	job_student	job_technician	job_unemployed
0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0
2	0	0	1	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0
4	0	1	0	0	0	0	0	0	0	0	0

After scaling and creating dummies, we combine the data frame and the whole dataset show as below.

	age	balance	day	campaign	pdays	previous	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employed	j
0	0.731404	0.092259	0.195769	0.0	0.0	0.0	0	0	0	0	1	0	0	
1	0.570729	0.073067	0.195769	0.0	0.0	0.0	0	0	0	0	0	0	0	
2	0.395437	0.072822	0.195769	0.0	0.0	0.0	0	0	1	0	0	0	0	
3	0.609772	0.086476	0.195769	0.0	0.0	0.0	0	1	0	0	0	0	0	
4	0.395437	0.072812	0.195769	0.0	0.0	0.0	0	1	0	0	0	0	0	

Feature Selection and Dimension Reduction

Besides variables dropped in preprocessing stage, we can try further feature selection and dimension reduction. (In the appendix, we also included PCA implementation but since it's not significantly improving the performance and it loses explainability, we will not include this method formally.)

The previous preprocessing stage bumped up the dimension of the dataset to 41 variables. In the hope of reducing dimension, we tried feature selection using Backward Elimination based on p values of variables. If p-value is greater than 0.05, it will get filtered out. After feature selection, our variables decreased from 42 to 31. Below are the variables after selection.

```
['age', 'balance', 'campaign', 'pdays', 'previous', 'job_admin.', 'job_blue-collar', 'job_entrepreneur', 'job_housema
id', 'job_management', 'job_self-employed', 'job_services', 'job_technician', 'job_unemployed', 'marital_divorced',
'marital_single', 'education_primary', 'education_secondary', 'housing_no', 'loan_no', 'month_apr', 'month_aug', 'mon
th_dec', 'month_feb', 'month_jul', 'month_jun', 'month_mar', 'month_may', 'month_nov', 'month_oct', 'month_sep']
```

Model Exploration and Selection

Balancing the dataset

Before we train the model, we need to use oversampling technique to balance the dataset. After some research and consideration, we settled on a technique called SMOTETomek. This method provided by the *imblearn* library combines the ability of both SMOTE and Tomek Links. In SMOTE (Synthetic Minority Oversampling Technique), it generates synthetic data for minority class by randomly choosing one of the k-nearest-neighbors and using it to create similar, but slightly modified, new observations. After that, Tomek Links algorithm can remove data from the majority class that is the closest with the minority class data. This function not only oversamples, but also "cleans" the data a little bit. It keeps the volume of our original dataset without adding additional information. This will help the model to learn the minority class (subscribers) better. We will also balance the training dataset only so that the information from test data won't be leaked to training the models.

After SMOTETomek, the length of oversampled data is 54410. Number of Subscribers in oversampled data is 27205 and number of Non-subscribers in oversampled data is also 27205. Our dataset is balanced and can feed the model.

```
from imblearn.combine import SMOTETomek
os = SMOTETomek(random_state=0)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
columns = X_train.columns
os_data_X, os_data_y = os.fit_resample(X_train, y_train)

os_data_X = pd.DataFrame(data=os_data_X,columns=columns)
os_data_y = pd.DataFrame(data=os_data_y)
print("length of oversampled data is ",len(os_data_X))
print("Number of Subscribers in oversampled data",len(os_data_y[os_data_y['y']==1]))
print("Number of Non-Subscribers in oversampled data",len(os_data_y[os_data_y['y']==0]))
```

```
length of oversampled data is  54410
Number of Subscribers in oversampled data 27205
Number of Non-Subscribers in oversampled data 27205
```

Baseline models with default parameters

Since our outcome variable is categorical and our data mining task is classification, several models suitable for classification comes to mind. We will use Decision Tree, Logistic Regression, Naïve Bayes, Random Forest and KNN models on our data. To briefly summarize why we chose these models: KNN, Decision Tree and Random Forest (which is a bagged version of Decision Tree) works well on large datasets like this one and they are friendly to large dimension of the training data. Logistic Regression offers easily understandable explanation of predictors and Naïve Bayes works well with high proportions of categorical data like our dataset. Moreover, we can do hyperparameter tuning for Random Forest and KNN models to see if we can further increase the predictive performance.

Feeding our oversampled training data, we explored and compared the results of baseline models as follows:

Logistic Regression:

The accuracy of logistic regression classifier on testing set is 0.8 and AUROC value for testing set is 0.68.

	precision	recall	f1-score	support
0	0.93	0.83	0.88	11969
1	0.29	0.52	0.38	1595
accuracy			0.80	13564
macro avg	0.61	0.68	0.63	13564
weighted avg	0.85	0.80	0.82	13564

Decision Tree:

The accuracy of decision tree classifier on testing set is 0.78 and AUROC value for testing set 0.58.

Naïve Bayes:

The accuracy of Naïve Bayes is 0.82 and AUROC value for testing set is 0.65.

KNN:

The accuracy of decision tree on testing set is 0.76 and the AUROC value for testing set is 0.64.

Random Forest:

The accuracy of random forest on testing set is 0.84 and AUROC value for testing set is 0.64.

The confusion matrix of the random forest model is shown below:

```
[[10782  1187]
 [  995   600]]
              precision    recall  f1-score   support

           0       0.92      0.90      0.91     11969
           1       0.34      0.38      0.35      1595

    accuracy                           0.84     13564
   macro avg       0.63      0.64      0.63     13564
weighted avg       0.85      0.84      0.84     13564
```

Hyperparameter tuning and cross-validation

We can see that although the performance of models are not too bad, but the recall and precision still has some room to improve. We can use hyperparameter tunning to improve our model's performance. We used RandomizedSearchCV to help the model find the best parameters for our models, providing range of values for each parameter. Hyperparameter tuning works by running multiple trials in single training job. It optimizes a single target variable, also called the hyperparameter metric. To optimize the performance of the model in separating the success class (subscribe) and the alternative class (no-subscribe), we used AUROC as our scoring metrics. Here are the results after we tuned our random forest and KNN model:

Random Forest:

Max_depth: 33 (The maximum depth of the tree)

Max_features: 11 (The number of features to consider when looking for the best split)

Min_samples_leaf: 5 (The minimum number of samples required to be at a leaf node)

Min_samples_split: 13 (The minimum number of samples required to split an internal node)

N_estimators: 883 (The number of trees in the forest)

After tuning, we used these parameters and re-run the model on training data to see how the performance improved. After hyperparameter tunning, the accuracy of random forest model on testing set is 0.87 and the AUROC is 0.68. Compared with the un-tuned version, the accuracy improved 0.3 and AUROC improved 0.04.

KNN:

Here is the best parameters for KNN after using hyperparameter tunning:
N_neighbors: 2 (Number of neighbors to use by default for kneighbors queries.)
P :1 (Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2.)
After hyperparameter tunning, the accuracy of decision tree on testing set is 0.83 and the AUROC is 0.60. Compared with before result, the accuracy improved 0. 07and AUROC decreased 0.04.

Performance Evaluation

Right now, it seems like that the Random Forest model works best with our training data. However, to make sure that the performance of the models generalizes well on different combinations of training and testing data, we decided to run a 10-fold cross-validation on our balanced training dataset before we draw our conclusion based on the performance of models on our testing set. Moreover, we want to check if using hyperparameter tunning increased the risk of overfitting the training set. In our 10-fold cross-validation, we used multiple performance metrics (Accuracy, AUROC, F-1, Precision and Recall) to evaluate the model performance.

Below is a stacked chart for evaluating model performance.

	Model	Accuracy Score	ROC-AUC Score	F1 Score	Precision Score	Recall Score
0	Logistic	0.764069	0.837079	0.745288	0.809751	0.690391
1	Decision Tree	0.878111	0.878111	0.879113	0.871859	0.886525
2	Naive Bayes	0.669509	0.716423	0.636723	0.706702	0.579386
3	Random Forest	0.933964	0.982395	0.933905	0.934452	0.933385
4	Random Forest Tuned	0.905238	0.968197	0.902998	0.924717	0.882307
5	KNN	0.851994	0.924070	0.855837	0.834054	0.878855
6	KNN Tuned	0.902959	0.934029	0.901785	0.912715	0.891152

The data frame above represents the results of cross-validation for all the 5 models (along with tuned RF and KNN). We could observe that tree models seem to suit our oversampled training data more - all the performance metrics seem better compared to other models. Random Forest Classifier performs the best here, and it also performs the best on our test set, both before and after tuning. Here interestingly, the tuned RF performs worse in cross-validation. This might be that these tuned parameters have a higher risk of overfitting our training data. However, since it still performs better than un-tuned version on un-seen test data as shown in the previously section, we will stick to this model and further explore its performance.

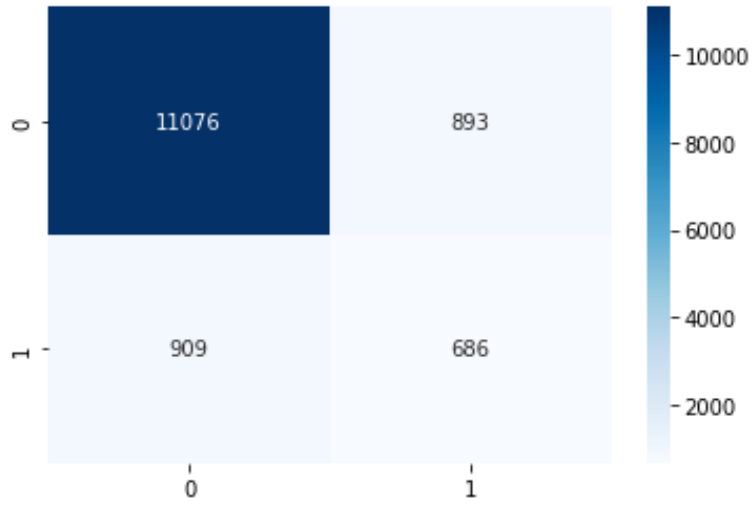
	precision	recall	f1-score	support
0	0.92	0.93	0.92	11969
1	0.43	0.43	0.43	1595
accuracy			0.87	13564
macro avg	0.68	0.68	0.68	13564
weighted avg	0.87	0.87	0.87	13564

Again, above is the classification report on testing set for **Random Forest tuned**. Here we noticed that the two classes are highly imbalanced. Since we don't want to use synthetic points in testing data and we'd like to prevent information leak from testing data (hence why we didn't do oversampling and cross-validation on testing set) We will consider using the weighted F-1, precision and recall as metrics and we are satisfied with 0.87 for class 1.

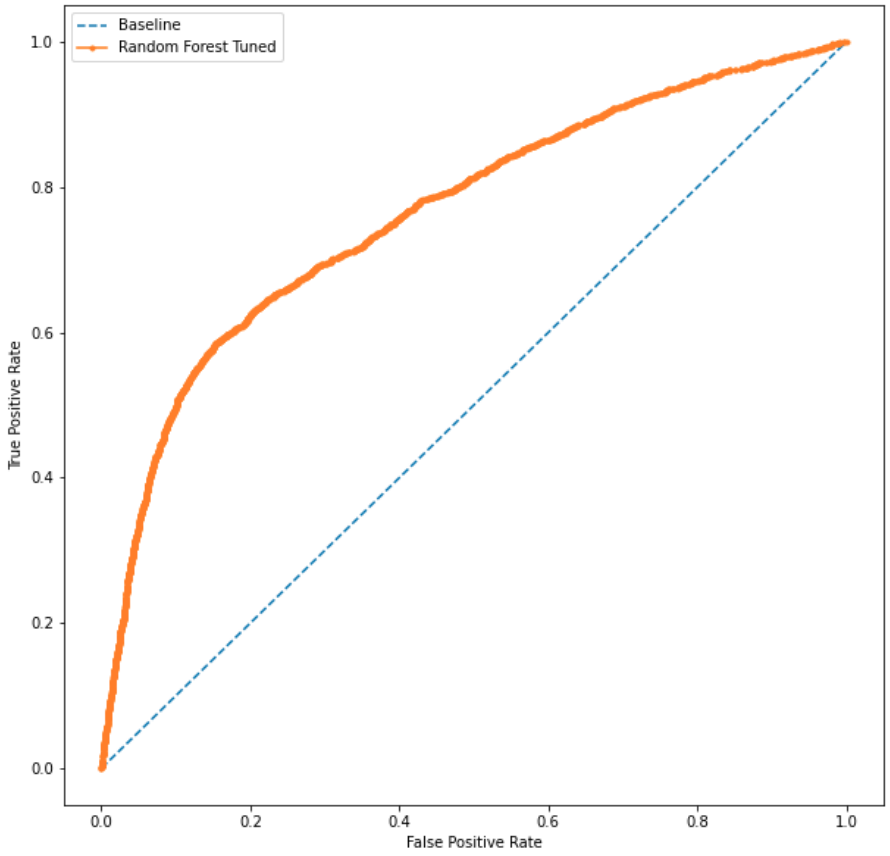
Performance Visualization:

Here is a heat map for the confusion matrix of our chosen model:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2b453696d0>
```



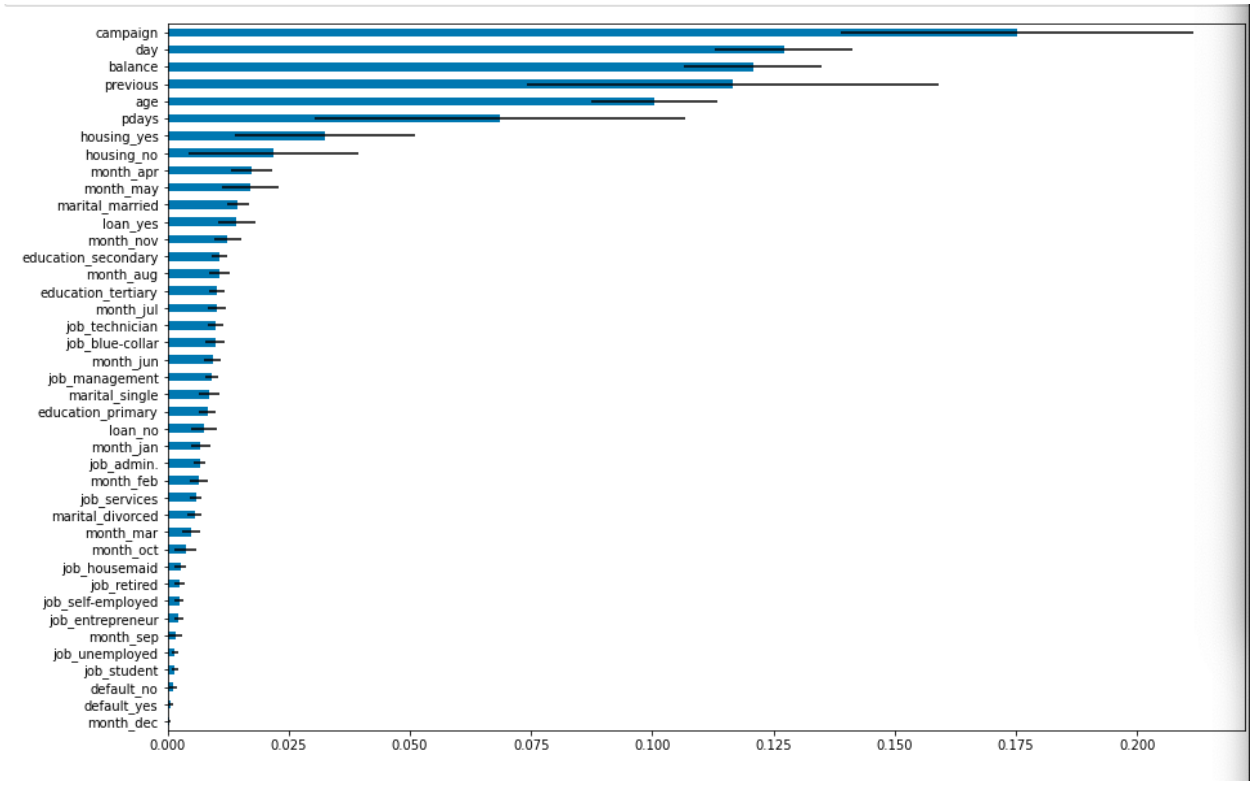
Here is the ROC Curve:



The Random Forest also could help us to select the best features, here is the importance ranking:

	feature	importance	std
31	month_dec	0.000313	0.000293
24	default_yes	0.000672	0.000620
23	default_no	0.001083	0.000801
14	job_student	0.001536	0.000715
16	job_unemployed	0.001567	0.000638
40	month_sep	0.001630	0.001516
8	job_entrepreneur	0.002344	0.000932
12	job_self-employed	0.002377	0.000832
11	job_retired	0.002530	0.000996
9	job_housemaid	0.002679	0.001147
39	month_oct	0.003717	0.002184
36	month_mar	0.004895	0.001967
17	marital_divorced	0.005613	0.001396
13	job_services	0.005849	0.001179
32	month_feb	0.006399	0.001901
6	job_admin.	0.006633	0.001175
33	month_jan	0.006857	0.002053
27	loan_no	0.007613	0.002689
20	education_primary	0.008241	0.001728
19	marital_single	0.008599	0.002190
10	job_management	0.009103	0.001392
35	month_jun	0.009289	0.001787
7	job_blue-collar	0.009830	0.001903
15	job_technician	0.009986	0.001590
34	month_jul	0.010110	0.001842
22	education_tertiary	0.010236	0.001668
30	month_aug	0.010650	0.002159
21	education_secondary	0.010819	0.001578
38	month_nov	0.012403	0.002741
28	loan_yes	0.014287	0.003913
18	marital_married	0.014541	0.002315
37	month_may	0.017040	0.005869
29	month_apr	0.017339	0.004194
25	housing_no	0.021818	0.017569
26	housing_yes	0.032522	0.018546
4	pdays	0.068601	0.038201
0	age	0.100527	0.012985
5	previous	0.116647	0.042455
1	balance	0.120728	0.014143
2	day	0.127127	0.014273
3	campaign	0.175249	0.036268

The five most important features are: campaign (how many times the client was contacted during this campaign), day(when the client was contacted), balance, previous(previous contacts), and age. We also included a feature importance plot below for the tuned random forest model.



All in all, we are choosing **Random Forest (tuned) classifier** as our final model.

Conclusion

Random Forest Model (tuned) has the best performance and we analyze the reasons as follows: Firstly, random forest intrinsically performs further feature selection automatically. Secondly, the model can do ensemble model training to achieve a more generalized performance. Thirdly, random forest is able to capture complex relationships (linear and non-linear) and performs well on large datasets like this one.

Moreover, using the feature importance summary above, the banking institution could focus on these five features for the client and the campaign. We also noticed that from both data exploration stage and the model result that this particular marketing campaign is time-sensitive. May has the highest volume of contacts made for the clients (from the bar chart) and although April doesn't have that many contacts made, it ranked higher than May in feature importance of our random forest classifier. This could also potentially help the bank to increase campaign efficiency.

Challenges and Future Improvement

Challenges:

Although the accuracy values are good across different models, we still have some challenges. For example, how to handle the "unknown" variables is still valuable to do more research. In our project, we use the imputation strategy and some variables were dropped. However, we can think about the way to make the "unknown" as an individual category, maybe there are new ways to handle the "unknown" variables and preprocess the data in general.

Future scopes for improvement:

In the future scopes, we still think about some ways to help us improve model's performance:
Firstly, the bank can collect more client data for subscribers from previous campaigns to improve models, meaning more data and higher accuracy.
Secondly, we can try more different ways to preprocess the dataset. For example, we can bin the age to 5 groups, then we can change the age to the categorical variable.
Thirdly, if possible, collect data for some social-economic factors for the year that the campaign was conducted. For example, unemployment rate, consumer price index (which measures the average change in prices over time that consumers pay for a basket of goods and service), etc.

Appendix: PCA

Except for the backward elimination, we also could use PCA to reduce dimension. Although PCA is designed for continuous variables and may not work well on binary data (which a lot of the variables in our dataset are), we'll try it anyways.

First step, we use the PCA function in Sklearn library and calculate the Principal Components Weights, which also called Eigenvectors for our dataset.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	0.025342	0.003625	0.010594	0.027324	-0.019901	-0.029692	-0.047572	-0.180803	0.002408	0.015890	0.194165	0.035295	0.013388	-0.053845	0.011
1	-0.056587	0.001367	-0.002634	-0.020738	0.018082	0.058268	-0.056647	-0.123248	0.004565	-0.018004	0.347406	-0.049544	0.009323	-0.052689	0.003
2	-0.086433	-0.001709	-0.006231	-0.033266	0.006078	0.022431	0.096390	-0.125961	-0.017467	-0.017269	-0.139854	-0.028261	-0.006175	0.053266	0.046
3	-0.012005	-0.003498	0.034662	0.014079	-0.014485	-0.046499	0.045292	-0.201053	0.020480	-0.012818	0.096549	-0.022483	0.003234	0.034912	-0.019
4	-0.009626	-0.003017	-0.013946	-0.000431	-0.021286	-0.095826	-0.086436	0.501698	-0.004759	0.026206	-0.176824	0.018825	-0.007335	-0.057255	0.011

The PCs are arranged in descending order of the variance(information) explained. We want to see how much of the total information is contributed by each PC, use explained_variance_ratio。

```
array([1.56805808e-01, 1.15638540e-01, 1.05503621e-01, 7.14288112e-02,
       6.37823253e-02, 4.51293880e-02, 4.40547367e-02, 4.07499664e-02,
       3.92390070e-02, 3.38545778e-02, 3.28680287e-02, 2.90014802e-02,
       2.59120692e-02, 2.45161967e-02, 2.09089070e-02, 2.01631900e-02,
       1.81525710e-02, 1.59827205e-02, 1.41826915e-02, 1.18809927e-02,
       1.03573229e-02, 8.87730863e-03, 8.51433096e-03, 7.34564349e-03,
       6.83648534e-03, 6.67891677e-03, 5.58276090e-03, 4.47890097e-03,
       3.40719997e-03, 3.18239180e-03, 2.80041199e-03, 1.28076356e-03,
       7.17890407e-04, 1.84042890e-04, 3.09244929e-32, 4.51651209e-33,
       4.04861255e-33, 2.44541014e-33, 1.74078301e-33, 1.02948805e-33,
       7.18493740e-35])
```

Then we print the Covariance Matrix:

```
Covariance Matrix
 [[ 6.19042678e-01  6.87596653e-18 -1.21134885e-16 ... -7.42010838e-33
   6.13981866e-33 -1.08373497e-34]
 [ 6.87596653e-18  4.56521301e-01  1.07333838e-16 ... -1.35446423e-33
   8.11369876e-33 -1.57559047e-34]
 [-1.21134885e-16  1.07333838e-16  4.16510363e-01 ...  3.98937154e-33
  -7.19000214e-33  7.55342823e-34]
 ...
 [-7.42010838e-33 -1.35446423e-33  3.98937154e-33 ...  6.86779908e-33
  -9.42890406e-36  1.32840941e-35]
 [ 6.13981866e-33  8.11369876e-33 -7.19000214e-33 ... -9.42890406e-36
   4.04455895e-33  2.77334370e-35]
 [-1.08373497e-34 -1.57559047e-34  7.55342823e-34 ...  1.32840941e-35
   2.77334370e-35  2.44576326e-34]]
```

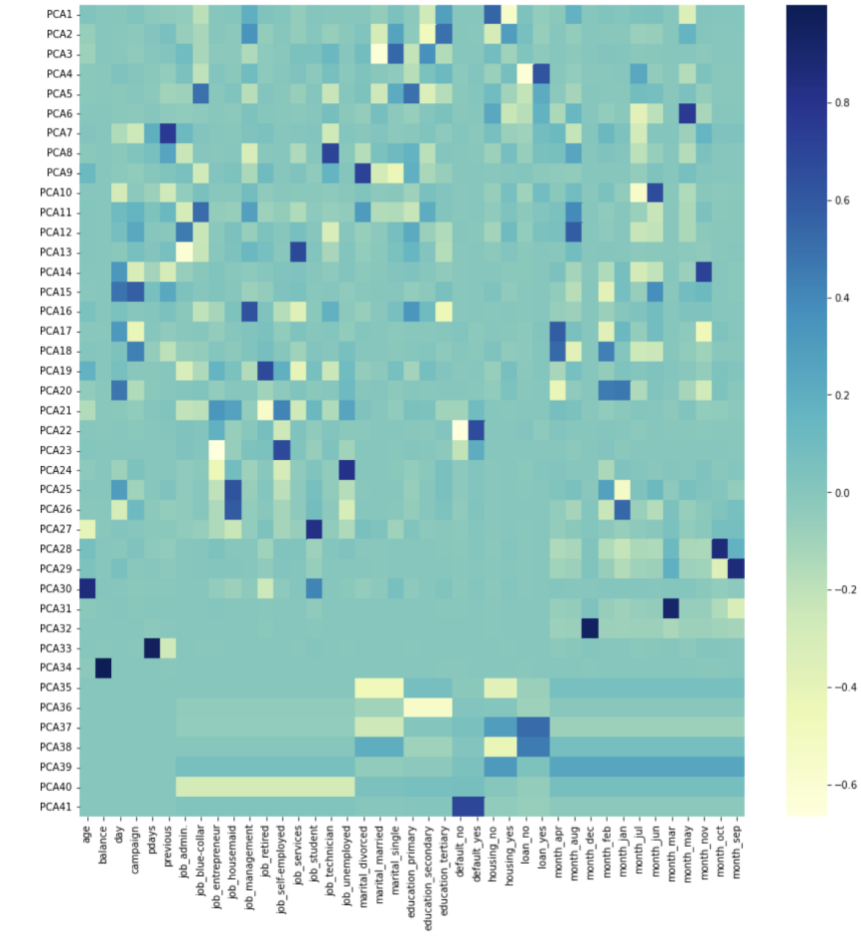
Print the Eigenvectors and Eigenvalues:

```
Eigenvectors
 [[ 1.00000000e+00 -4.23080747e-17 -5.98101517e-16 ...  2.02382660e-32
  -7.55698468e-33  1.29690536e-32]
 [ 0.00000000e+00  1.00000000e+00  1.17682901e-15 ...  2.89640667e-27
  -1.08157715e-27  1.85613351e-27]
 [ 0.00000000e+00  1.37097622e-15 -1.00000000e+00 ...  1.44103598e-27
  -5.38090462e-28  9.23490108e-28]
 ...
 [ 0.00000000e+00  1.65364980e-32 -3.22849286e-32 ...  1.64283241e-01
  -9.57300029e-01  2.05072461e-01]
 [ 0.00000000e+00  7.20240579e-33  3.30761428e-32 ... -1.32248195e-01
   9.59530577e-02 -8.24014971e-02]
 [ 0.00000000e+00 -2.84772440e-34 -2.28832352e-33 ...  2.65095905e-02
   9.50287139e-03 -1.03391584e-02]]

Eigenvalues
[6.19042678e-01 4.56521301e-01 4.16510363e-01 2.81988805e-01
2.51801779e-01 1.78163153e-01 1.73920612e-01 1.60873941e-01
1.54908930e-01 1.33652119e-01 1.29757390e-01 1.14492914e-01
1.02296445e-01 9.67857774e-02 8.25448108e-02 7.96008469e-02
7.16632650e-02 6.30970641e-02 5.59908552e-02 4.69041398e-02
4.08889503e-02 3.50461055e-02 3.36131314e-02 2.89993519e-02
2.69892821e-02 2.63672282e-02 2.20397911e-02 1.76819397e-02
1.34510464e-02 1.25635419e-02 1.10555505e-02 5.05623683e-03
2.83410930e-03 7.26570049e-04 1.12245301e-31 1.77781880e-32
2.30282767e-34 4.26307042e-33 1.32619503e-32 7.35162164e-33
1.07005093e-32]
```

Then, we plot a Heat map to check the effect of variables on each component.

We can see influence on each of the PC by the features.



For PCA we will not use SMOTE since it doesn't really make sense to us on synthetic points. Instead, we will do cross-validation on the entire dataset and use weighted F-1, recall, and precision as performance metrics. (Alternative strategy: use RandomOversampler from imblearn to create copies and then do PCA - we don't like creating copies so we will stick to the previous one)

Check the fundamental models first before do parameter tuning. Only check logistic regression, naive bayes and decision tree first.

Here is the result for the three models:

	Model	Accuracy Score	ROC-AUC Score	F1 Score	Precision Score	Recall Score
0	Logistic	0.882838	0.719493	0.832810	0.837940	0.882838
1	Decision Tree	0.826016	0.593671	0.828477	0.831092	0.826016
2	Naive Bayes	0.827741	0.692941	0.830111	0.832641	0.827741

PCA Conclusion:

It seems like the accuracy increased significantly for logistic and naive bayes model, but decreased for decision tree. For all three models. the AUROC scores all decreased significantly. This might suggest that the model doesn't separate 0 and 1 class well (although it's imbalanced dataset). Considering the fact that PCA takes away explainability and we only have a few continuous numerical variables in our predictors, we will stop here. For future improvement, we might consider doing other types of preprocessing to both use PCA and account for the imbalanced issue.