Project 3: Classification Model

By: Neha Awasthi

Introduction

A time deposit or term deposit is a deposit in a financial institution with a specific maturity date or a period to maturity, commonly referred to as its "term". The money is invested for an agreed rate of interest over a fixed amount of time, or term. Term deposits are a major source of income for a bank. Banks have various outreach plans to sell term deposits to their customers such as email marketing, advertisements, telephonic marketing, and digital marketing.

Telephonic marketing campaigns remains one of the most effective ways to reach out to people. However, they require huge investment as large call centers are hired to execute these campaigns. Hence, it is crucial to identify the customers most likely to convert beforehand so that they can be specifically targeted via call.
In this project, we build a classification model to explain how a Portuguese bank can use predictive analytics to help prioritize customers which would subscribe to a bank term deposit.

Scope:

The dataset used here is obtained from https://www.kaggle.com . It can be accessed at https://www.kaggle.com/datasets/prakharrathi25/banking-dataset-marketing-targets.

The Dataset contains the client data such as: age of the client, their job type, their marital status, etc. along with the information of the calls made to the potential clients such as the duration of the call, day, and month of the call, etc. Given this information, we try to predict whether the client will subscribe to term deposit.

The data is related with over 40,000 direct telephone marketing campaigns of a Portuguese banking institution from May 2008 to November 2010. We use logistic regression model in R to build my model and then assess the stability and the accuracy of the model.

Analysis:

Regression Analysis:

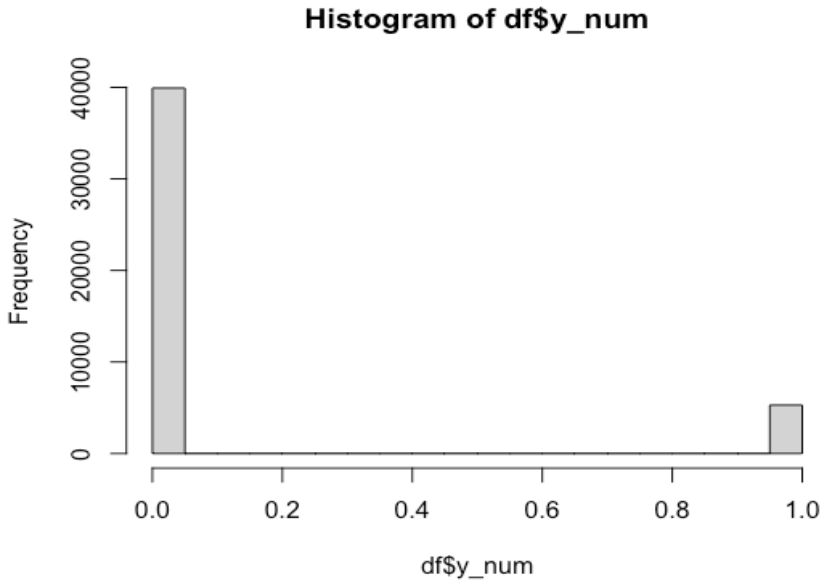
In this project, we use R to develop a predictive machine learning model. Exploring the data, we can see that there are various numerical and categorical columns.

Logistic regression is a traditional machine learning model that fits a linear decision boundary between the positive and negative samples. Logistic regression uses a line (Sigmoid function) in the form of an “S” to predict if the dependent variable is true or false based on the independent variables.

Exploratory Data Analysis and Feature Engineering:

Before we start building the model, let’s understand our data.

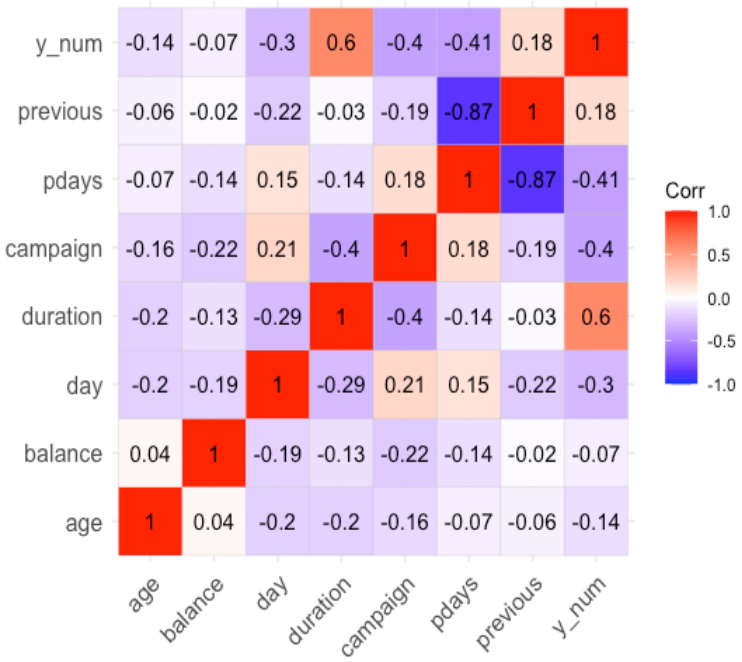
Let’s see the distribution of the output variable



We can see that there are nearly 40,000 observations for no and around 5,000 observations for yes.

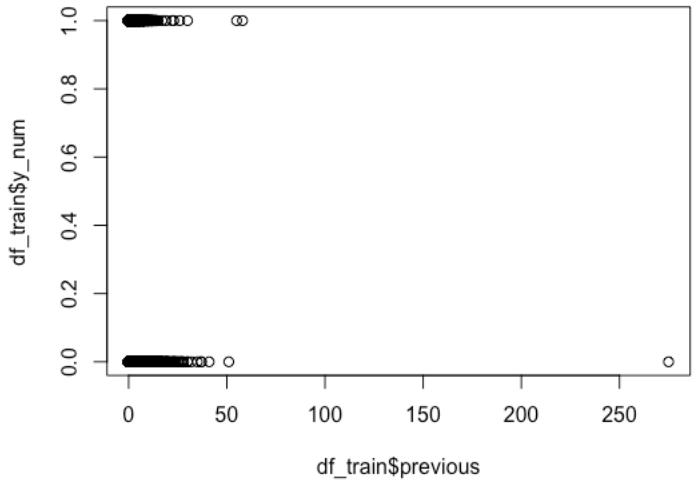
Majority of our variables are categorical variables with both 'job' and 'month' having more than 8 categories. Let's move onto basic feature engineering and categorize 'job' into 5 categories and 'month' into four quarters. We will also convert our output variable into numerical binary categories to facilitate the classification model.

Let's see the correlation matrix for our numerical variables:



We can see that ‘pdays’ and ‘previous’ are highly correlated so we will drop one of them from our model.

Let’s see the relationship of ‘previous’ with our output variable



Once the features are selected, because our data is so imbalanced, we first bifurcate the dataset with 0 and 1 values for the output variable. Next, we split both the sets into training and test randomly sampling 70% of both datasets into training and the rest into the test set. Now, we join the training and the test datasets respectively to maintain the integrity of the data.

Here, we build our initial logistic model with our modified features and run it on the training dataset. It is apparent that there are some variables whose coefficients are insignificant.

After various permutations tinkering with the features, we arrive at our final model.

Below is the confusion matrix of the model

```
prop.table(table(df_train$y_num, train_pred))
 train_pred
           0          1
0 0.86295699 0.02006509
1 0.07890163 0.03807628
prop.table(table(df_test$y_num, test_pred))
 test_pred
           0          1
0 0.86184017 0.02115895
1 0.08131820 0.03568269
```

As we can see, the training accuracy is 90% and the test accuracy is 89.68%. Looking at the confusion matrix, we can arrive at the conclusion that the model is stable (as both the confusion matrices for training and test contains similar values) and fairly accurate.

However, as we noticed in the start, this data is imbalanced and while we try to mitigate that with our data splitting, the guess work accuracy of this would have been 88% from the start. This is due to **accuracy paradox** where because of the data imbalance, accuracy of the model isn't really a great indicator of its performance.

Oversampling:

We can try an alternate method of compensating for data imbalance. Here, we use oversampling for the minority class, and we use ROSE package in R to oversample as our dataset contains categorical independent variables along with continuous independent variables.

If we populate the minority class with similar number of approximate observations as the majority class, since it will be approximately 7:1 ratio of dummy data, this might negatively affect our model too much considering most of our predictor variables are categorical. Below is the confusion matrix of the model with similar no. of observations for both the output values:

```
prop.table(table(df_train$y_num, train_pred))
 train_pred
           0          1
0 0.41630204 0.08377833
1 0.10940653 0.39051310
prop.table(table(df_test$y_num, test_pred))
 test_pred
           0          1
0 0.41490958 0.08184015
1 0.11242604 0.39082424
```

As we can see while the model is stable which we can observe from the similarity in confusion matrices of training and test data, the test accuracy has dropped to 80.4%.

So as a compromise, we took twice the observations in the minority class compared to the initial observations and kept our initial technique of data splitting in training and test. Below is the screenshot of the confusion matrix of the model:

```
prop.table(table(df_train$y_num, train_pred))
 train_pred
           0          1
0 0.76127873 0.03564707
1 0.11064849 0.09242571
prop.table(table(df_test$y_num, test_pred))
 test_pred
           0          1
0 0.76014637 0.03672655
1 0.10825017 0.09487691
```

Again , we can see that the model is stable and while the accuracy improved from the previous model, it still dropped from the initial model to 85.5%.

This might be our best bet to mitigate the effects of the imbalance and not overtly sully our data with dummy categorical observations which will hinder the learning of our model.

Conclusion:

Using our classification model, we can predict with a fair bit of accuracy which customers are likely to buy a term deposit. However, we need to account for the imbalance of data and take measures to mitigate it. We have tried two different methods here: First Data-splitting while maintaining the ratio of the initial dataset and second oversampling. Both have their pros and cons. Finally, we have tried a combination of the two to both account for the data imbalance and not sully our original dataset too much from which the model will be learning and predicting the behavior in future.