Bank Telemarketing Campaign Strategy

Introduction

This file provides documentation for Springboard's Capstone Two Project. This project consists of analysis and machine learning techniques applied to a historical tabular dataset of a bank's telemarketing campaign hosted on Kaggle.com.

Problem Statement

We want to build a model that classifies whether a new target of our marketing campaign will subscribe a term deposit. It is important to be precise in our marketing, without neglecting a sizeable marketing scope. Because we are looking to find the optimal balance between precision (correctly classifying true instances) and recall (capturing as many true cases as possible), we will use the f1-score metric which is the harmonic mean of precision and recall.

Methodology

Exploratory Data Analysis

We started on this project by first conducting exploratory data analysis to build an intuition and to gain an understanding of the data before delving further. Our first step within exploratory data analysis was to summarize the distributions of the feature columns using descriptive statistics. We followed this step by evaluating the data types of the feature columns and reviewing a count of the null values contained in each column.

We visualized the data using kernel density plots, histograms, boxplots, cumulative, and probability distribution functions, scatterplots, and correlation heatmaps. This process improved our understanding of the features' distributions and relationships.

Data Cleaning

From the processes above, we observed a few issues with the structure of the data which needed to be resolved before applying a machine learning pipeline to it. We started by splitting the dataset by independent and dependent variables. We then replaced 'yes'/'no' in the target variable (dependent variable) with 1/0. Using the independent variables from the split dataset, we applied a method called *One-Hot-Encoding* to convert the dataset's non-numeric data into numeric data. Our last step of data pre-processing was to standardize our data using the Z-score function.

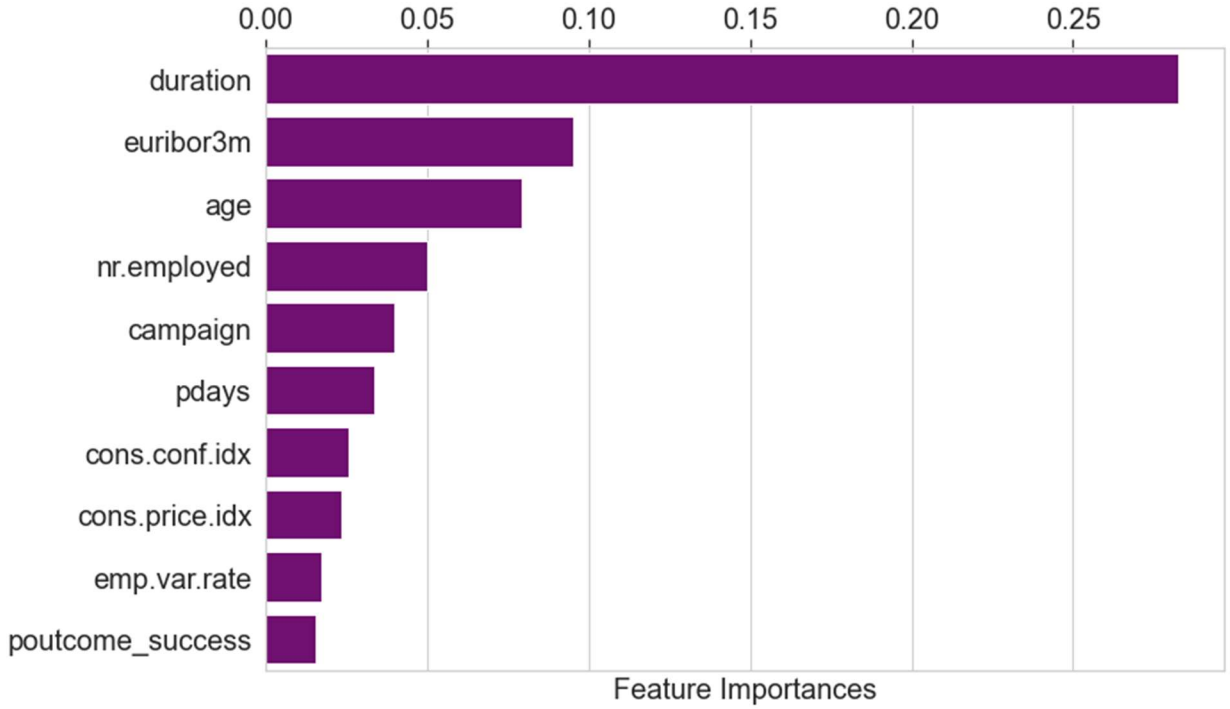
Modeling & Evaluation

The first step to modeling the data was to choose a model based on the type of problem at hand and the structure of the data. Understanding that this was a binary classification task, and that the data's dimension was neither too large nor too small, we devised a list of potential algorithms designed for binary classification with the given dimension of around 60.

We decided the following models were appropriate for the problem and that we would compare their default performances: Random Forest Classifier, Logistic Regression Classifier, K-Nearest Neighbors Classifier, Adaptive-Boost Classifier, Extreme Gradient Boosting Classifier,

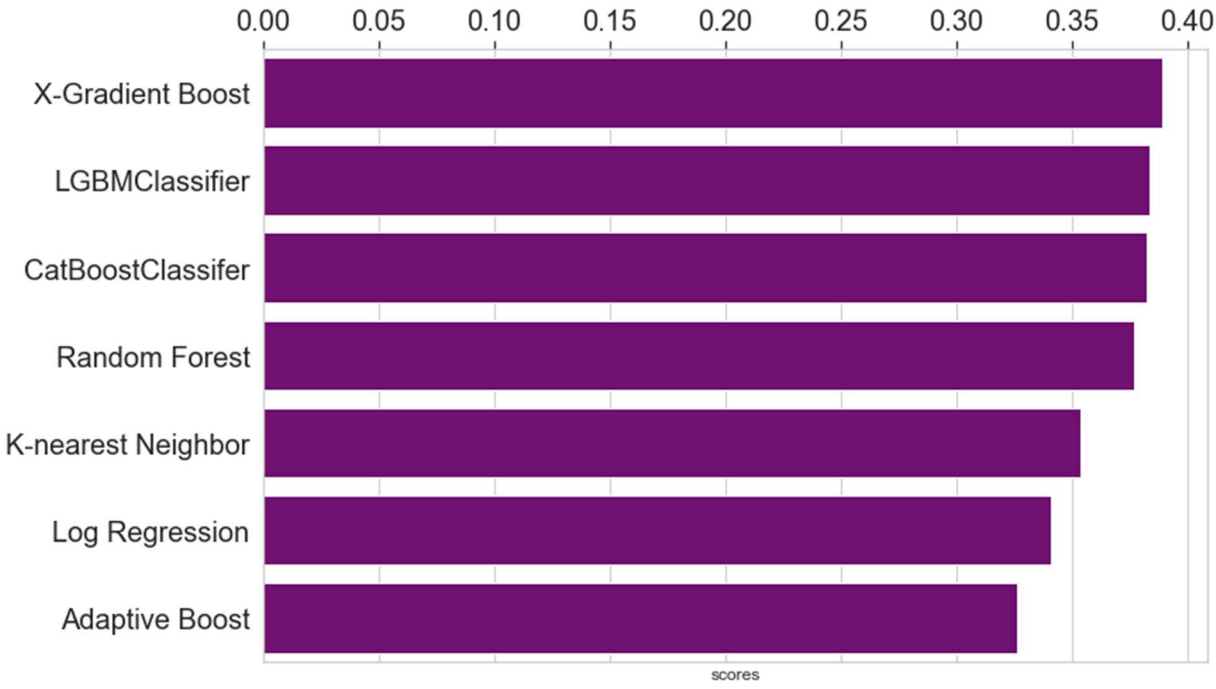
Categorical Boosting Classifier, Light Gradient Boosting Classifier, and a Stacking Classifier of some of the best performing models.

Before comparing all algorithms on the data, it is good practice to evaluate feature importance using one of several methods available. Random Forest Classifier has a 'Feature Importance' attribute which does a quick and easy job of revealing features most highly correlated with the target columns. This step is useful to gain a deeper understanding of the features' relationships with each other, and with the target column. This step can sometimes reveal data leakage and collinearity between features. The image below is the result from using Random Forest Classifier's 'Feature Importance' Attribute.



The 'duration' feature which represents the duration of the call, would cause data leakage in the model if included and thus we removed it from our dataset. In the process, we were able to gather valuable insight and we can communicate the importance of duration regarding whether a client will make a term deposit.

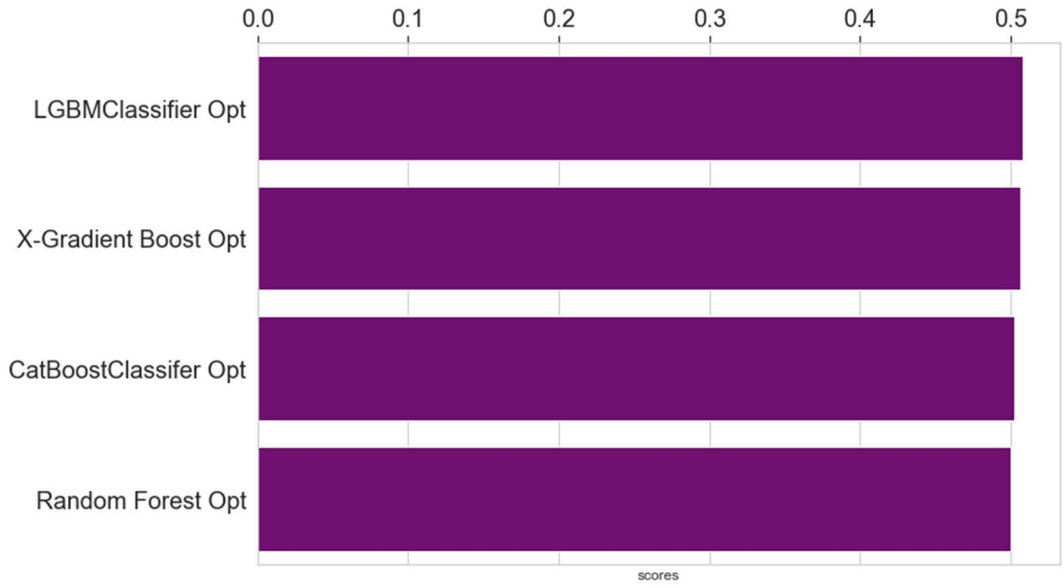
We instantiated the models with their default hyper-parameters and evaluated their performances using 5-fold cross-validation. The image below is a horizontal bar plot of the resulting f1 scores from the default algorithm comparison.



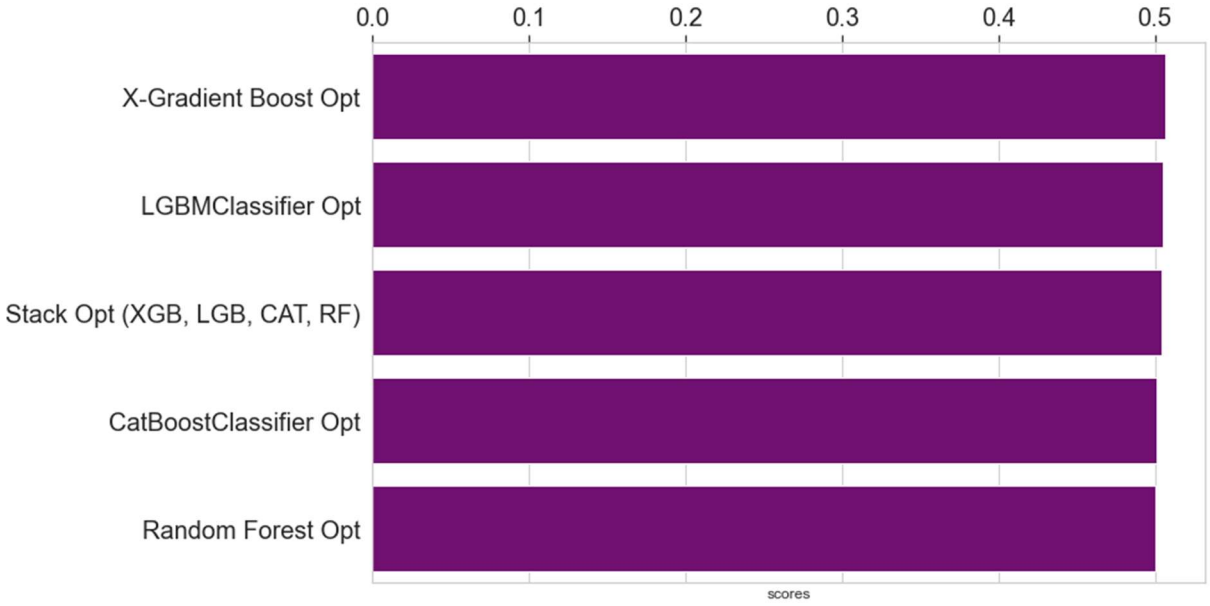
Using the elbow-method we selected the Extreme Gradient Boosting Classifier, Light Gradient Boosting Classifier, Categorical Boosting Classifier and Random Forest Classifier, as the best performing models would go on to be hyper-parameter optimized and compared again.

Results

After hyper-parameter optimizing each of these four models and scoring them using a standard five-fold cross validation. Below is a horizontal bar plot displaying the comparison of the better performing algorithms after being hyper-parameter optimized using a grid search.



The f1-scores are quite similar, it is difficult to say one algorithm has significantly better performance than the others. We will make one last comparison between the top two algorithms and a stacking classifier of the four algorithms pictured above.



Again, there is no obvious optimal algorithm for this problem based on the f1 score. Considering the algorithms' internal features, the Random Forest Classifier is the algorithm of choice for this problem due to its explicability and useful 'Feature Importance' attribute.

Further Exploration & Insight

We did not achieve a notable f1 score from our methods thus far, and as per recommendation by my mentor Dr. Yuxuan Xin in spirit of gaining experience and in hope of improving results, I was encouraged to stratify the dataset, and use an ensemble of models, each modeling only its assigned partition of the dataset. This stratification ensemble technique can sometimes improve results, however our attempt at partitioning the dataset on the age feature did not yield improved results for us. Unfortunately, due to time constraints, this project was put on hold until a later revisit. Upon return, the plan of action is to resume efforts to improve performance by stratifying on other features, a collection of features, gathering more relevant data, and possibly applying Dimensionality Reduction Techniques such as Principal Component Analysis. Although techniques such as PCA can be an effective method to improve algorithm performance, we generally avoid this method because of the tradeoff that is the loss of data interpretability.