Predicting Bank Marketing Campaign Success using Machine Learning

Chaitra Hegde Aakash Kaku Neelang Parghi

Abstract—**Data from a marketing campaign run by Banco de Portugal is examined. The campaign's aim was to increase customers' subscription rates to fixed-term deposit products, such as CDs. Using knowledge from the course, a number of machine learning algorithms are implemented to answer the question: How can banks successfully market these products in the most efficient way possible and with the highest possible rate of success?**

I. Introduction

With the startling rise over the last few decades of media and technology which increases the amount of information we have at our fingertips (cell phones, television, Internet, etc.), humans are now more connected than ever. One result of this is that marketing campaigns are growing evermore pervasive in our daily lives. This glut of advertising has forced businesses to compete for the attention of a populace that has an ever growing amount of distractions. Thus raising the question: How can businesses successfully advertise their products in the most efficient way possible with the highest possible rate of success? We will answer this question in the context of banks advertising fixed term deposit products to their customers. Using data collected from a previous bank marketing campaign, a number of features centered around the clients, the campaign itself, and general market conditions will be explored. Based on this data, machine learning models will predict which clients will subscribe and what banks can do to increase the rate of subscription.

II. Methodology

A. Programming in Python

Python provides a number of packages and libraries for the convenience of the programmer. The whole project is coded using *Python 3*. Packages/libraries used are *numpy* for array manipulation, *pandas* for dataframe operations, and *matplotlib* and *seaborn* for visualization. The *sklearn* libraries were also critical in providing packages for machine learning algorithms, tasks, and by giving the user the control to set important attributes of those algorithms as they wished. The dataset is stored in a dataframe and is intensively queried and manipulated using facilities provided by the Python 3 environment. Other data structures such as arrays, lists, and dictionaries are used as needed[1].

B. Data cleaning and exploratory analysis

The dataset was provided by the U. C. Irvine Machine Learning Repository and contained information on 41,188 clients across 20 different features, both categorial (marital status, job type, education, etc.) and numeric (age, number of days since previous contact, etc.). The target variable is a binary "Yes" (client subscribed) or "No" (client did not subscribe).

The first step is to load the dataset into a dataframe for easy manipulation and exploration using the *pandas* package. The 'duration' feature was dropped due to the risk of data leakage. This feature measures the length of the phone call between the bank's marketing representative and the customer. Since this time cannot be known until after the call has ended (when the outcome for that customer is already known), including it in a predictive model would not provide realistic results.

The next step was to explore and clean the categorical variables such as 'job type,' 'marital status,' 'education,' etc. Plots for each were produced that looked at their relative frequency as well as normalized relative frequency. In Python, these graphs are created using the *seaborn* package.

Many of these features contain unknown values so the next question is how to deal with this missing data. Simply discarding these rows would lead to a huge reduction in the amount of data and thus greatly interfere with the results. Instead, these missing values are imputed using other independent variables to infer the missing values. While this does not guarantee that all the missing data will be restored, a majority of it will be. For instance, cross-tabulation between 'job' and 'education' was used based on the hypothesis that a person's job will be influenced by their education. Thus, a person's job is used to predict their education level. The Python function *cross_tab* was created for this cross-tabulation step. A similar cross-tabulation process was carried out for the 'house ownership' and 'loan status' features. It's important to note that in making these imputations, care was taken to ensure the correlations made sense in the real world. If not, the values were not replaced. Throughout this process, dataframes using the *pandas* package were invaluable. Python provides quickness, ease of modifiability and ease of replacement of values throughout the dataset thanks to this tool.

The next task is to deal with missing data among the numerical features. In this particular dataset, all missing values were encoded as '999.' It's quickly noted that while only the 'pdays' (number of days since that customer had been contacted from the previous campaign) column contained such values, they made up the majority of the data for this feature. In other words, this column was missing more data than it contained. Further exploration showed that this missingness was due to customers who had not been contacted previously at all. To deal with this, the numerical feature ‘pdays’ was replaced with a categorical feature based on whether the customer had never been contacted, contacted 5 or less days ago, 6-15 days ago, etc.

Finally, a heatmap was created to show us whether there is strong correlation between the target variable and any independent variables. The heatmap is created using Spearman correlation, which measures the degree to which the rankings of each variable (as opposed to the actual values) align, thus minimizing the effect of outliers[2]. Once this is measured, those variables are expected to be significant during the modeling stage. This graphic was created using Python’s *seaborn* package and the specially written function *drawheatmap*, which takes a dataframe as an input. The code for this function can be seen in the Jupyter notebook for this project.

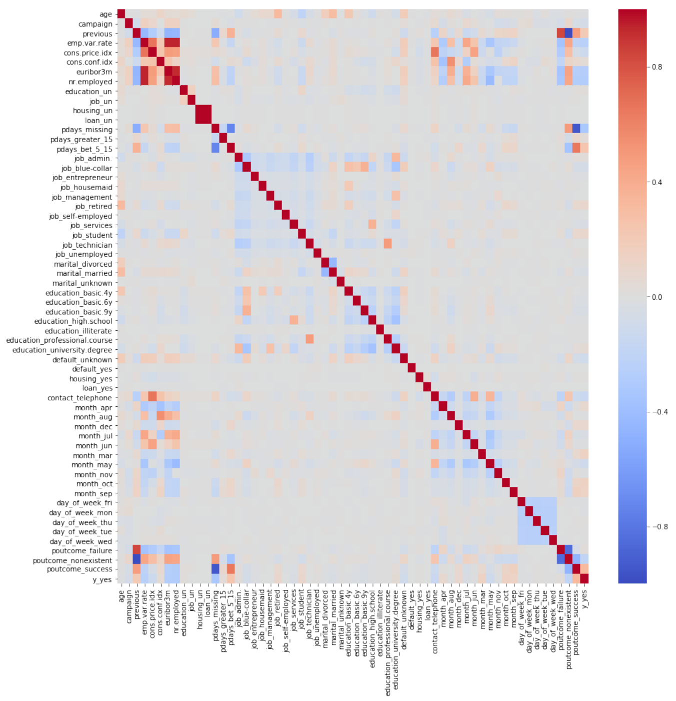

Fig. 1. Spearman correlation heatmap of rankings for each variable

For performing predictive analysis, many well known machine learning models should be fit on training data to learn parameters of the model and then they can be run on test set to get the prediction. Models used in our project are discussed below.

C. Model Building

The dataset is divided into training data and test data with the intention of using the training data to find the parameters of the particular model being used (fitting the model on the training data) and then applying this to the test data to determine the model’s performance and to draw conclusions about its predictive capability. This can be done with a *sklearn.cross_validation.train_test_split* function call by specifying split ratio.

Logistic Regression: Python provides the package *sklearn.linear_model.LogisticRegression* for Logistic Regression. LR a is well known classification model. The linear model fits the training data to the equation $y = w_0 + w_1x_1 + w_2x_2 + \dots$ (where y stands for the target variable, w_0 stands for the y intercept, $x_1, x_2, x_3, \dots$ are feature vectors, and $w_1, w_2, w_3, \dots$ are their corresponding weights) while the logistic regression algorithm uses the same decision boundary with bit modifications as shown: $P(X) = \frac{1}{1+e^{-y}}$.

Logistic regression is used because classification is not exactly a linear function and using linear regression produces an output within $[-\infty, +\infty]$ while the probability has to be within $[0, 1]$. The logistic function itself does output the probability of an instance belonging to the positive class. This output probability does indeed have a range of $[0, 1]$, hence overcoming the drawbacks of classification using a linear model.

Decision Trees: Python provides the package *sklearn.tree.DecisionTreeClassifier* for the decision tree classifier. Decision trees are a simple yet effective method for classification. Using a tree structure, this algorithm splits the data set based on one feature at every node until all the data in the leaf belongs to the same class. The criterion used for splitting is called information gain, which is based on a purity measure called entropy, a measure of disorder. The set with the highest impurity will have higher entropy whereas the set which has higher purity will have lower entropy. Information gain measures the change in entropy due to the amount of information added. The higher the information gain, the more information that feature provides about the target variable.

By default, the decision tree grows deep and complex until every leaf is pure and hence it is prone to overfitting.

Random Forest: Python provides the package *Sklearn.ensemble.RandomForestClassifier* for the Random Forest classifier. Random forest classifiers are one of the ensemble learning methods for classification. It constructs multiple decision trees (a “forest”) at the training time and the output prediction is the class which is the mode of the predictions made by the individual decision trees in the ensemble.

Formally, we can write this as $C_{\text{rf}}(x) =$ Majority vote $C_b(x)_1^B$ where $C_b(X)$ is the class prediction of the b^{th} decision tree.

In this method, the individual trees are intentionally overfit and the validation set is used to optimize tree level parameters.

Adaptive Boosting (AdaBoost): Python provides the *sklearn.ensemble.AdaBoostClassifier* package for AdaBoost classification. This is one of the most famous ensemble models that can be used for classification as well as regression. The idea here is to use a weak learning method several times to get a succession of hypotheses. The weak learner method here is decision trees with a single split. Here, those instances which are difficult to classify receive increasingly larger weights until the algorithm identifies the model that correctly classifies it. Predictions are made by a majority vote of the weak learners' predictions, weighted by their individual accuracy.

Gradient Boosting: Python provides the *sklearn.ensemble.GradientBoostingClassifier* package for Gradient Boosting classification. The Gradient Boosting model is a generalized version of AdaBoost. The objective is to minimize the loss of the model by adding weak learners using a gradient descent-like procedure. One new weak learner is added at a time and existing weak learners in the model are frozen and left unchanged.

D. Model Evaluation

For evaluating all the models built, AUC score is used. This is chosen as the scoring metric because it has been established that for cases where classes are unbalanced (such as this), AUC score is a better evaluation criterion than the accuracy score. For each model, five-fold cross-validation is performed over the training set. The *kfold* function from *sklearn* was used extensively for this step. The mean AUC score is calculated for each set of selected parameters. The final model (and hyper-parameters) are selected based on the highest out-of-sample mean AUC score.

E. Hyper-parameter Tuning

For all each model implemented, the hyper-parameters were tuned to obtain the optimal performance of the classifier.

Logistic Regression: For Logistic Regression, two hyper-parameters were tuned: the penalty type ('L1' or 'L2' penalty) and the regularization coefficient ('C': 10^{-4} to 10^{5} on the log scale). Below is the graph of mean AUC vs. C for the different penalty types. From the figure, it is clear that the classifier is quite robust to the C values and the penalty type. We obtain a maximum mean AUC of 0.7903 for C = 0.1 and penalty = 'L1'. This graph was created using Python's *matplotlib* package and a function we created called *plot_mean_auc_LR* which can be seen in the accompanying Jupyter notebook.

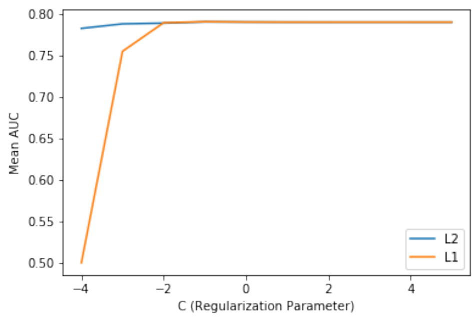

Fig. 2. Hyper-parameter tuning for Logistic Regression

Decision Trees, Random Forest Classifier and Gradient Boosted Trees: For Decision Trees, Random Forest, and Gradient Boosted Trees, two hyper-parameters were tuned: minimum samples split (the minimum number of samples required to split an internal node) and minimum samples leaf (the minimum number of samples required to be at a leaf node). These two parameters help control the depth of the trees and thus help to control the model's complexity. Below are the graphs of mean AUC vs. leaf values for different split values. From the figures, it is clear that the classifiers were sensitive to the hyper-parameter chosen. These figures were created using *matplotlib* and the function *plotAUCDTRF* in the Jupyter notebook.

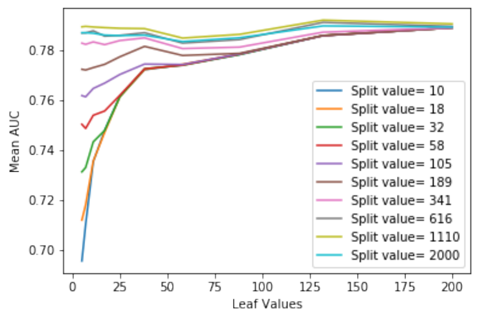

Fig. 3. Hyper-parameter tuning for Decision Trees

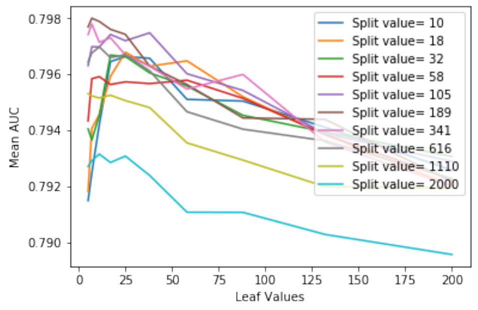

Fig. 4. Hyper-parameter tuning for Random Forest Classifier

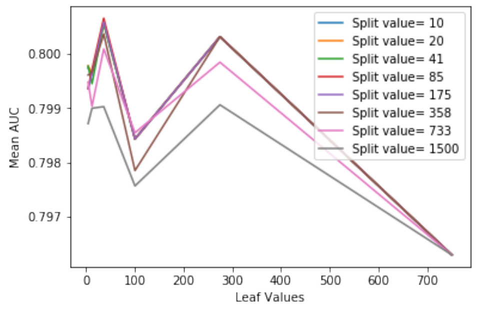

Fig. 5. Hyper-parameter tuning for Gradient Boosted Trees

Hyper-Parameter Tuning of Trees

Model	Best Leaf Value	Best Split Value	Mean AUC
Decision Tree Classifier	132	1110	0.7919
Random Forest Classifier	7	189	0.7979
Gradient Boosted Trees	37	85	0.8006

In the table, the best tree based models are summarized with the hyper-parameters and the AUC score obtained for the same.

AdaBoost Classifier: For the Ada-Boost classifier, only one hyper-parameter is tuned: the number of estimators. The higher the number of estimators, the more complex the model and the higher the chance of overfitting becomes. In Figure 6, we see a graph of mean AUC vs. number of estimators. As before, this graph was created with *matplotlib* and the function *plot_mean_auc_Ada_Boost*. From the figure, it is clear that the classifier is pretty sensitive to the estimator values. We obtain the maximum mean AUC of 0.8157 for $n_{\text{estimators}} = 1000$.

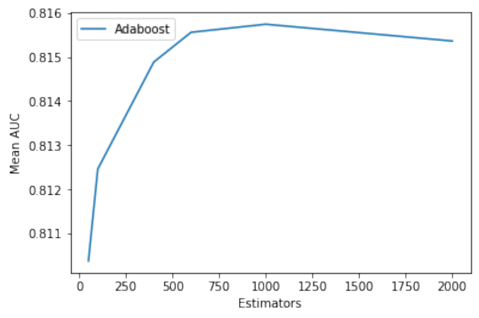

Fig. 6. Hyper-parameter tuning for Ada-Boosting Classifier

III. Results

From the above results, the best out of sample model performance was obtained for the AdaBoost Classifier with $n_{\text{estimators}} = 1000$. On the test data, the best AUC score achieved was 0.8036.

The importance of the features (in terms of how greatly they affected the coefficients) was also plotted. This provides valuable insight toward understanding which features contribute the most toward the models' performance.

From the feature importance plot, it can be inferred that Europe's Libor rate, age of the applicant, employment variation rate, campaign, consumer confidence index, consumer price index, mode of contact (= telephone), and number of employees are some of the most important features in predicting the outcome. The below graph was created using the *seaborn* package and the function *plotfeatureimportances*.

A. Discussion

Based on the feature importance plot, some recommendations can be made to the bank's marketing team:

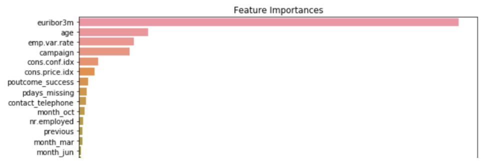

Fig. 7. Most important Features based on the AdaBoost model

- The marketing team should collaborate with economic experts so that as soon as they have some signals indicating the Libor going up (or the economic situation improving, i.e., consumer price index or consumer confidence index goes up), they can expect more customers to subscribe for the term deposit and should pro-actively reach out to them before the bank's competitors do.
- The marketing team should target relatively old age customers who would be looking for safe and profitable investment options. The marketers should ensure to convey the peace of mind and steady source of income these products provide as a value proposition to these customers.
- Although the 'duration' (length of marketing phone call) variable was not used in the prediction models for various reasons cited earlier, the correlation of the 'duration' variable with the target variable shows that the higher the duration, the more likely it is that the customer will subscribe to the term deposits (correlation = 0.405). This makes intuitive sense because longer duration shows that the customer is interested in the product. Hence, the marketers should try to make the call engaging and increase the duration of the call.
- The telephone seems to be the most preferred mode of communication.
- The marketing team should prioritize those customers to whom they previously reached out during previous campaigns. They are likely to subscribe for the term deposit.

IV. Conclusion

From this project, we learned how banks can improve their marketing campaigns by focusing their efforts on certain prime-grade clients and also how they can recognize market conditions which are favorable to increase client subscription for the fixed-term products they are offering. All of this was possible by implementing data science and machine learning methods in Python. Tools such as dataframes, arrays, for loops, etc. were all critical for the success of this project. A large number of other tools and techniques from the Python for Data Science course were used and these were invaluable for making our analyses and predictions. This project demonstrated how powerful Python can be for data science applications.

References

[1] Foster Provost and Tom Fawcett, *Data Science for Business*. O'Reilly Media, 2013.
[2] https://ababankmarketing.com/insights/network-effect-strong-ever/