Udacity Machine Learning Nanodegree

Bank Marketing Campaign Predictive Analysis

Loay Zakaria

August 30th, 2019

I. Definition

Project Overview

In banks, huge data records information about their customers. This data can be used to create and keep clear relationship and connection with the customers in order to target them individually for definite products or banking offers. Usually, the selected customers are contacted directly through: personal contact, telephone cellular, mail, and email or any other contacts to advertise the new product/service or give an offer, this kind of marketing is called direct marketing. In fact, direct marketing is in the main a strategy of many of the banks and insurance companies for interacting with their customers .

Historically, the name and identification of the term direct marketing suggested first time in 1967 by Lester Wunderman, which he is considered to be the father of direct marketing . In addition, some of the banks and financial-services companies may depend only on strategy of mass marketing for promoting a new service or product to their customers. In this strategy, a single communication message is broadcasted to all customers through media such as television, radio or advertising firm, etc.In this approach, companies do not set up a direct relationship to their customers for new-product offers. In fact, many of the customers are not interesting or respond to this kind of sales promotion .

Accordingly, banks, financial-services companies and other companies are shifting away from mass marketing strategy because its ineffectiveness, and they are now targeting most of their customers by direct marketing for specific product and service offers [1, 4]. Due to the positive results clearly measured; many marketers attractive to the direct marketing. For example, if a marketer sends out 1,000 offers by mail and 100 respond to the promotion, the marketer can say with confidence that the campaign led immediately to 10% direct responses. This metric is known as the 'Response Rate', and it is one of many clear quantifiable success metrics employed by direct marketers.

From the literature, the direct marketing is becoming a very important application in data mining these days. The data mining has been used widely in direct marketing to identify prospective customers for new products, by using purchasing data, a predictive model to measure that a customer is going to respond to the promotion or an offer [5]. Data mining has gained popularity for illustrative and predictive applications in banking processes.

Personally it is an interesting field to dig in and explore the data trying to figure out a model to fit in real work this is why I choose it because it is nearly a real live problem to be solve beside this domain is so interesting as it hold a lot of various data to manipulate with it

Problem Statement

All bank marketing campaigns are dependent on customers' huge electronic data. The size of these data sources is impossible for a human analyst to come up with interesting information that will help in the decision-making process. Data mining models are completely helping in the performance of these campaigns.

The purpose is increasing the campaign effectiveness by identifying the main characteristics that affect a success (the deposit subscribed by the client) based on a handful of algorithms that we will test (e.g. Logistic Regression, Random Forests, Decision Trees and others). With the experimental results we will demonstrate the performance of the models by statistical metrics like accuracy, sensitivity, precision, recall, etc. We the higher scoring of these metrics, we will be able to judge the success of these models in predicting the best campaign contact with the clients for subscribing deposit. The aim of the marketing campaign was to get customers to subscribe to a bank term deposit product. Whether they did this or not is variable 'y' in the data set. The bank in question is considering how to optimize this campaign in future. What would your recommendations to the marketing manager be?

In this problem I will use a supervised learning and will use many algorithms trying to pick the best of it varying between regression and classification

Metrics

The evaluation metrics proposed are appropriate given the context of the data, the problem statement, and the intended solution. The performance of each classification model is evaluated using three statistical measures; classification accuracy, sensitivity and specificity.

We use the classification accuracy metrics to give us the first hint about the accuracy of our work to just have a small prove that we go in the right direction

Then we use the sensitivity which gives us more specific results as for higher value of sensitivity mean higher value of true positive and lower value of false negative and vice versa. Also it measure is used to determine the proportion of actual positive cases, which got predicted correctly.

And we also use specificity as for higher value of specificity would mean higher value of true negative and lower false positive rate and vice versa to determine the proportion of actual negative cases, which got predicted correctly.

And we use them both beside the classification accuracy to determine the model performance For healthcare and financial domain, and in our case we use it in a financial domain, a domain with high sensitivity.

It is using true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The percentage of Correct/Incorrect classification is the difference between the actual and predicted values of variables. True Positive (TP) is the number of correct predictions that an instance is true, or in other words;

it is occurring when the positive prediction of the classifier coincided with a positive prediction of target attribute.

True Negative (TN) is presenting a number of correct predictions that an instance is false, (i.e.) it occurs when both the classifier, and the target attribute suggests the absence of a positive prediction. The False Positive (FP) is the number of incorrect predictions that an instance is true. Finally, False Negative (FN) is the number of incorrect predictions that an instance is false. Table below shows the confusion matrix for a two-class classifier.

	Predicted No	**Predicted Yes**
Actual No	TN	FN
Actual Yes	FP	TP

Classification accuracy is defined as the ratio of the number of correctly classified cases and is equal to the sum of TP and TN divided by the total number of cases (TN + FN + TP + FP).

$$Accuracy = \frac{TP + TN}{TN + FN + TP + FP}$$

Precision is defined as the number of true positives (TP) over the number of true positives plus the number of false positives (FP).

$$Precision = \frac{TP}{TP + FP}$$

Recall is defined as the number of true positives (TP) over the number of true positives plus the number of false negatives (FN).

$$Recall = \frac{TP}{TP + FN}$$

Sensitivity refers to the rate of correctly classified positive and is equal to TP divided by the sum of TP and FN. Sensitivity may be referred as a True Positive Rate.

$$Sensitivity = \frac{TP}{FN + TP}$$

Specificity refers to the rate of correctly classified negative and is equal to the ratio of TN to the sum of TN and FP

$$Specificity = \frac{TN}{TN + FP}$$

II. Analysis

Data Exploration

The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be ('yes') or not ('no') subscribed. The classification goal is to predict if the client will subscribe (yes/no) a term deposit (variable y).

Input variables:

1. **age** (numeric)
2. **job** : type of job (categorical: 'admin.','blue-collar','entrepreneur','housemaid','management','retired','self-employed','services','student','technician','unemployed','unknown')
3. **marital** : marital status (categorical: 'divorced','married','single','unknown'; note: 'divorced' means divorced orwidowed)
4. **education** (categorical: 'basic.4y','basic.6y','basic.9y','high.school','illiterate','professional.course','university.degree','unknown')
5. **default**: has credit in default? (categorical: 'no','yes','unknown')
6. **housing**: has housing loan? (categorical: 'no','yes','unknown')
7. **loan**: has personal loan? (categorical: 'no','yes','unknown')
8. **contact**: contact communication type (categorical: 'cellular','telephone')
9. **month**: last contact month of year (categorical: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
10. **day_of_week**: last contact day of the week (categorical: 'mon','tue','wed','thu','fri')
11. duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.
12. **campaign**: number of contacts performed during this campaign and for this client (numeric, includes last contact)
13. **pdays**: number of days that passed by after the client was last contacted from a previous campaign (numeric; 999 means client was not previously contacted)
14. **previous**: number of contacts performed before this campaign and for this client (numeric)
15. **poutcome**: outcome of the previous marketing campaign (categorical: 'failure','nonexistent','success')
16. **emp.var.rate**: employment variation rate - quarterly indicator (numeric)
17. **cons.price.idx**: consumer price index - monthly indicator (numeric)
18. **cons.conf.idx**: consumer confidence index - monthly indicator (numeric)
19. **euribor3m**: euribor 3 month rate - daily indicator (numeric)
20. **nr.employed**: number of employees - quarterly indicator (numeric)

Output variable (desired target):

1. **y** - has the client subscribed a term deposit? (binary: 'yes','no')

The dataset has 21 columns and 41188 rows, with 20 features, and one response variable. The number of customers who subscribed is 4640 and the number of customers who did not subscribe is 36548. Based on this the response rate of customers is about 11.27%, this makes the dataset very imbalanced. Fig.1 shows age distribution in the data and is a normal distribution with slight left skewness. This hints that majority of the responses are in 25-40 age group.

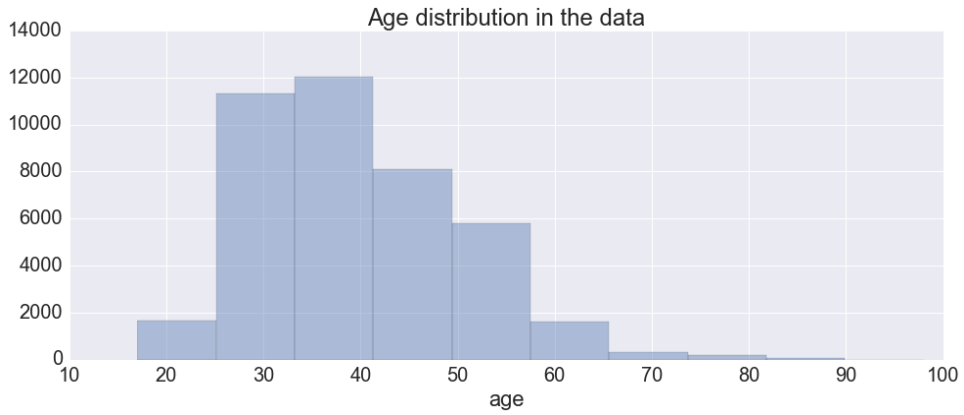


Fig. 1 Age distribution

Below table shows descriptive statistics on the data. `duration` seems to have a lot more variance, it could be a good predictor.

	age	duration	campaign	pdays	previous	emp.var.rate	cons.price.idx
count	41188.00000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000	41188.000000
mean	40.02406	258.285010	2.567593	962.475454	0.172963	0.081886	93.575664
std	10.42125	259.279249	2.770014	186.910907	0.494901	1.570960	0.578840
min	17.00000	0.000000	1.000000	0.000000	0.000000	-3.400000	92.201000
25%	32.00000	102.000000	1.000000	999.000000	0.000000	-1.800000	93.075000
50%	38.00000	180.000000	2.000000	999.000000	0.000000	1.100000	93.749000
75%	47.00000	319.000000	3.000000	999.000000	0.000000	1.400000	93.994000
max	98.00000	4918.000000	56.000000	999.000000	7.000000	1.400000	94.767000

Exploratory Visualization

Fig. 2 shows data distribution of `*job*` factors with `*duration*` and split by `*marital*` status. It is hard to spot relevances between factors from this plot. We can pick a few numeric variables and plot them to see if we can find any different patterns.

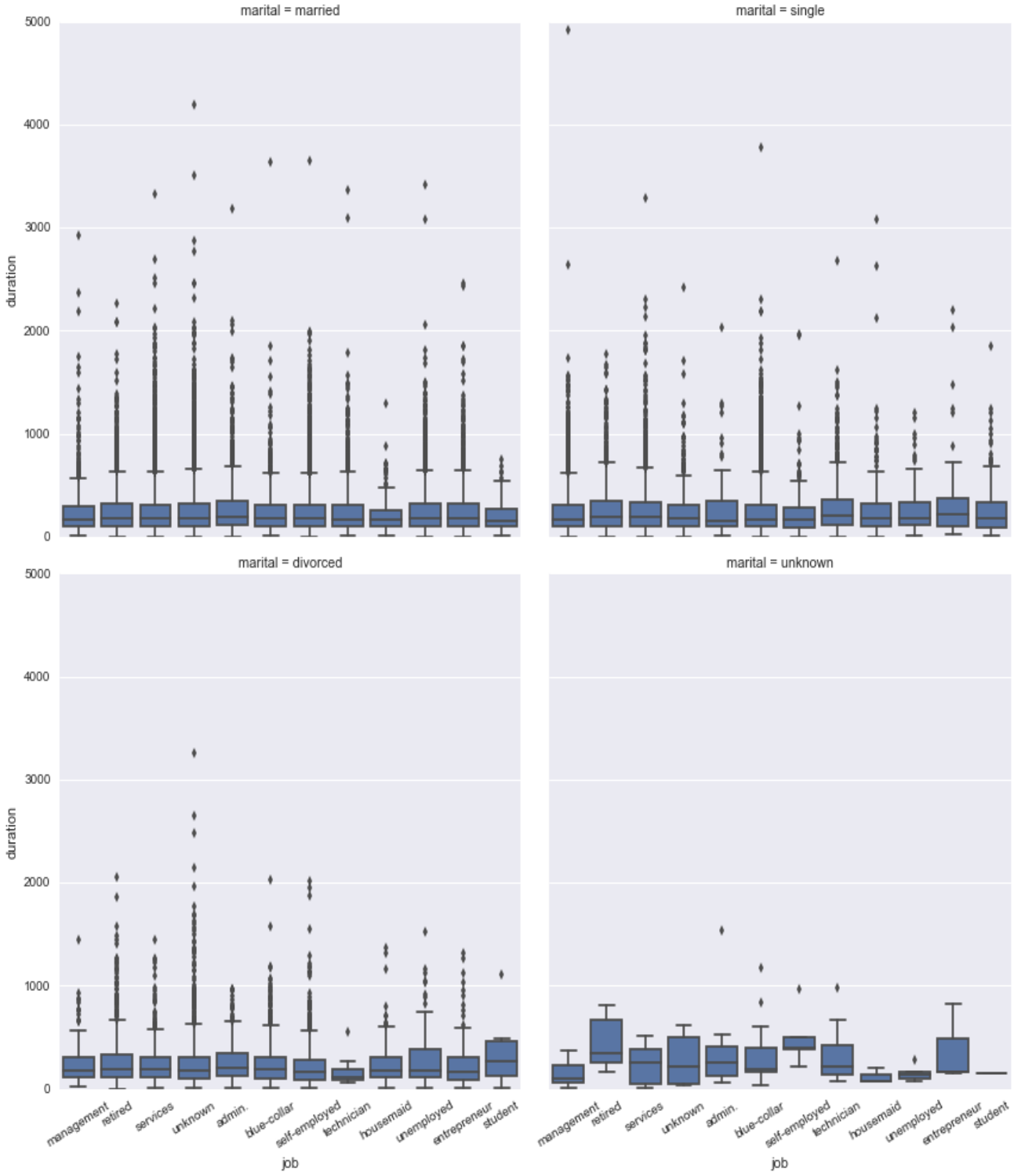


Fig. 2 Job vs duration split by marital status

Fig. 3 Scatterplot matrix of `*age*`, `*duration*`, `*campaign*` and `*euribor3m*`

We can see some interesting patterns in Fig. 3, there are splits in data for `*euribor3m*` and also note that the durations are smaller for age > 60 and other such relevant information. We could also check data distributions of each variable but I will leave it out for now and move ahead. We also applied scatterplot matrix on whole dataset to visualize inherent relationships between variables in the data shown in Fig. 4. But concluded that there was not much of a strong relationship.

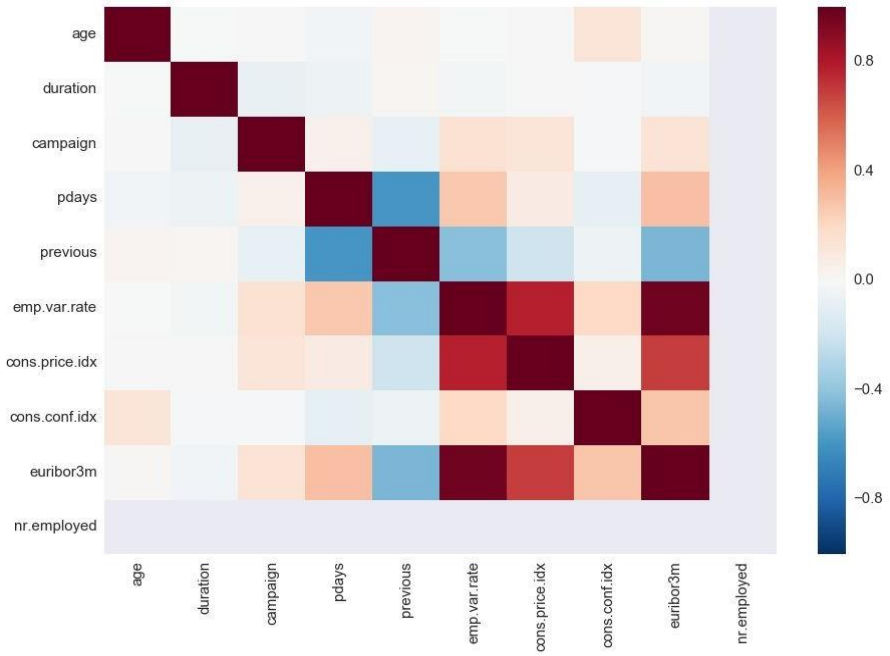


Fig. 4 Scatterplot matrix

Algorithms and Techniques

A Decision Tree is a robust and transparent Machine Learning model. The tree starts with a single node and then branches out, with a decision being made at every branch point. It can be used to predict whether a particular variable would have mattered in the customer's decision to subscribe or not to the bank's term deposit. The given dataset is a typical supervised learning problem for which tree type models perform a lot better than the rest. We don't know which algorithms would be fit for this problem or what configurations to use. So let's pick a few algorithms to evaluate.

- **Logistic Regression (LR):** is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). it is a classification algorithm used to assign observations to a discrete set of classes. It has two types binary and multi-linear function failsclass. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

  Below is an example logistic regression equation:

  $$y = e^{(b0 + b1*x)} / (1 + e^{(b0 + b1*x)})$$

  Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data. The actual representation of the model that y

- **Simplicity of results.** In most cases, the interpretation of results summarized in a tree is very simple. This simplicity is useful not only for purposes of rapid classification of new observations (it is much easier to evaluate just one or two logical conditions, than to compute classification scores for each possible group, or predicted values, based on all predictors and using possibly some complex nonlinear model equations), but can also often yield a much simpler "model" for explaining why observations are classified or predicted in a particular manner
- **Tree methods are nonparametric and nonlinear.** The final results of using tree methods for classification or regression can be summarized in a series of (usually few) logical if-then conditions (tree nodes). Therefore, there is no implicit assumption that the underlying relationships between the predictor variables and the dependent variable are linear, follow some specific non-linear link function

- **Advantage of Decision Tree**

  Easy to use and understand.

  Can handle both categorical and numerical data.

  Resistant to outliers, hence require little data preprocessing.

- **Classification and Regression Trees (CART)**: The CART algorithm is structured as a sequence of questions, the answers to which determine what the next question, if any should be. The result of these questions is a tree like structure where the ends are terminal nodes at which point there are no more questions.

Classification vs Regression	
A tree model where the target variable can take a discrete set of values.	A tree model where the target variable can take continuous values typically real numbers.
Dependent Variable	
For classification tree, the dependent variables are categorical.	For regression tree, the dependent variables are numerical.
Values	
Has a set amount of unordered values.	Has either discrete yet ordered values or indiscrete values.
Purpose of Construction	
Purpose of constructing the regression tree is to fit a regression system to each determinant branch in a way that the expected output value comes up.	A classification tree branches out as determined by a dependent variable derived from the previous node.

- **Classification Trees:** where the target variable is categorical and the tree is used to identify the "class" within which a target variable would likely fall into.

  The classification tree starts with the independent variable, which branches out into two groups as determined by the existing dependent variables. It is meant to elucidate the responses in the form of categorization brought about by the dependent variables.

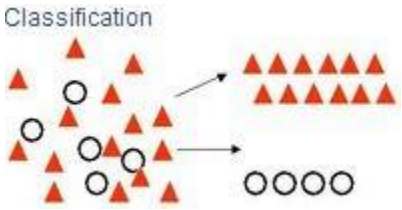


Fig. 5 Classification Tree

- **Regression Trees:** where the target variable is continuous and tree is used to predict it's value.

  The regression tree starts with one or more precursor variables and terminates with one final output variable. The dependent variables are either continuous or discrete numerical variables.

Fig. 6 Regression Tree

- **Random Forests (RF):**

  Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set,Random forests are bagged decision tree models that split on a subset of features on each split. This is a huge mouthful, so let's break this down by first looking at a single decision tree, then discussing bagged decision trees and finally introduce splitting on a random subset of features.
- **It has some advantage such as :**

  Impressive in Versatility, Parallelizable, Great with High dimensionality, Quick Prediction/Training Speed, Robust to Outliers and Non-linear Data, Low Bias, Moderate Variance
- **Adaptive Boosting (AB):**

  It can be used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In some problems it can be less susceptible to the overfitting problem than other learning algorithms. The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model can be proven to converge to a strong learner, the Ada Boost error function E (f) = e ^− y (x) f (x) takes into account the fact that only the

sign of the final result is used, thus | F (x) | can be far larger than 1 without increasing error. Ada Boost is a type of "Ensemble Learning" where multiple learners are employed to build a stronger learning algorithm. Ada Boost works by choosing a base algorithm (e. decision trees) and iteratively improving it by accounting for the incorrectly classified examples in the training set.

- **Extreme Gradient Boost (XGB**): The idea behind GBM is a more sophisticated. Roughly, the idea is to again, combine weak predictors. The trick is to find areas of misclassification and then “boost” the importance of those incorrectly predicted data points. And repeat. The result is a single Tree unlike RF.

  XGBoost is a model based on tree ensemble which is a set of classification and regression trees (CART). XGBoost classifies the members of a family into different leaves and assigns scores on corresponding leaf. Usually, a single tree is a weak leaner which is not strong enough to use in practice. Boosted trees are typically shallow which are high in bias and low in variance. A tree ensemble model sums prediction of multiple tree. Below is an example of tree ensemble of two trees that classifies whether someone likes play computer game (Fig.7).

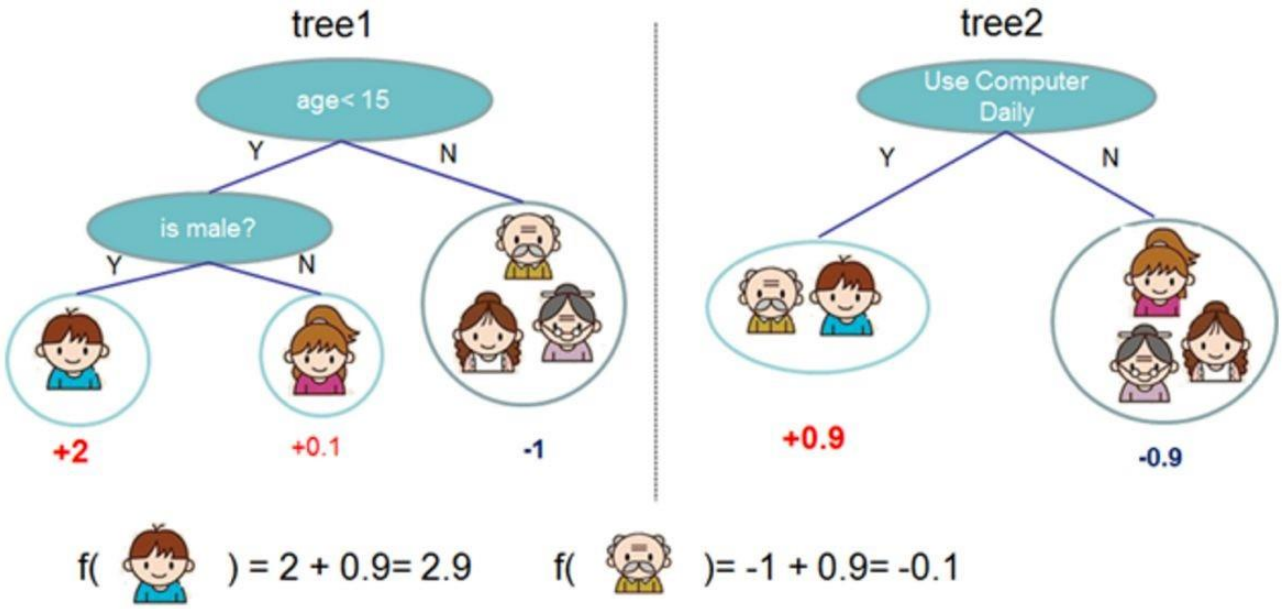


Fig. 7 Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree

We are using 5-fold cross validation to estimate accuracy. This will split our dataset 5 parts, train on 4 and test on 1 and repeat for all combinations of train-test splits. Also, we are using

the metric of accuracy to evaluate models. This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the scoring variable when we run build and evaluate each model next.

Benchmark Model

Below table highlights performances of various models that were tried with their accuracies and errors. The standard Extreme Gradient Boosting (XGB) model with default parameters yields 91.4% accuracy on training data. So will consider it as benchmark and try to beat the benchmark with hyper parameter turning.

Algorithms	**Accuracy**	**Std. Error**
Logistic Regression	0.909611	0.005315
CART	0.888800	0.005325
Random Forest	0.912525	0.006937
AdaBoost	0.913218	0.007852
XGBoost	0.914640	0.007852

I. Methodology

Data Preprocessing

We will prepare the data by splitting feature and target/label columns and also check for quality of given data and perform data cleaning. To check if the model I created is any good, I will split the data into `training` and `validation` sets to check the accuracy of the best model. We will split the given `training` data in two ,70% of which will be used to train our models and 30% we will hold back as a `validation` set.

There are several non-numeric columns that need to be converted. Many of them are simply yes/no, e.g. housing. These can be reasonably converted into 1/0 (binary) values. Other columns, like profession and marital, have more than two values, and are known as categorical variables. The recommended way to handle such a column is to create as many columns as possible values (e.g. profession_admin, profession_blue-collar, etc.), and assign a 1 to one of them and 0 to all others. These generated columns are sometimes called dummy variables, and we will use the pandas.get_dummies() function to perform this transformation.

Several Data preprocessing steps like preprocessing feature columns, identifying feature and target columns, data cleaning and creating training and validation data splits were followed here and can also be referenced for details in attached jupyter notebook.

Data Preprocessing/Cleaning

In this section, I will prepare the data by splitting feature and target/label columns and also check for quality of given data and perform data cleaning. To check if the model I created is any good, I will split the data into training and validation sets to check the accuracy of the best model. We will split the given training data in two ,70% of which will be used to train our models and 30% we will hold back as a validation set.

Preprocess Feature Columns

As you can see, there are several non-numeric columns that need to be converted! Many of them are simply yes/no, e.g. housing. These can be reasonably converted into 1/0 (binary) values. Other columns, like profession and marital, have more than two values, and are known as categorical variables. The recommended way to handle such a column is to create as many columns as possible values (e.g. profession_admin, profession_blue-collar, etc.), and assign a 1 to one of them and 0 to all others. These generated columns are sometimes called dummy variables, and we will use the pandas.get_dummies() function to perform this transformation. The code cell below performs the preprocessing routine.

```
def preprocess_features(X):
    ''' Preprocesses the student data and converts non-numeric binary variables into
        binary (0/1) variables. Converts categorical variables into dummy variables. '''

    # Initialize new output DataFrame
    output = pd.DataFrame(index = X.index)

    # Investigate each feature column for the data
    for col, col_data in X.iteritems():

        # If data type is non-numeric, replace all yes/no values with 1/0
        if col_data.dtype == object:
            col_data = col_data.replace(['yes', 'no','unknown'], [1, 0, np.nan])

        # If data type is categorical, convert to dummy variables
        if col_data.dtype == object:
            col_data = pd.get_dummies(col_data, prefix = col)

        # Collect the revised columns
        output = output.join(col_data)

    return output

full_data = preprocess_features(full_data)
print("Processed feature columns ({} total features):\n{}".format(len(full_data.columns), list(full_data.columns)))
```

Identify feature and target columns

```
# Extract feature columns
feature_cols = list(full_data.columns[:-1])

# Extract target column 'responded'
target_col = full_data.columns[-1]

# Show the list of columns
print("Feature columns:\n{}".format(feature_cols))
print("\nTarget column: {}".format(target_col))

# Separate the data into feature data and target data (X_all and y_all, respectively)
X_all = full_data[feature_cols]
y_all = full_data[target_col]

# Show the feature information by printing the first five rows
print("\nFeature values:")
print(X_all.head())
```

Data cleaning/washing

Let's check how many missing values our dataset now has...

```
X_all.isnull().sum()
```

```
Thos are the missing values in the dataset the others are zero
default                         8597
housing                          990
loan                             990
nr.employed                    33425
```

So looks like we have 4 columns that have missing values.

Default, housing and loan are the four columns that has the most missing values but instead of dropping these altogether I will try to impute them as they may hold relevant information for the dataset. nr.employed has too many missing so we will go ahead and drop it. To fill missing values, We will use median statistic.

```
X_all.drop('nr.employed',  axis=1, inplace=True)
```

The following code here is dropping the nr.employed column .

Training and Validation Data Split

So far, I have converted all categorical features into numeric values. For the next step, I will split the data (both features and corresponding labels) into training and test sets. In the code cell below, we will implement the following:

- Randomly shuffle and split the data (X_all, y_all) into training and validation subsets.
- Split training and validation into 70% and 30%.
- Set a random_state for the function(s).
- Store the results in X_train, X_validation, y_train and y_validation.

```
validation_size = 0.30
from sklearn.model_selection import train_test_split

X_train, X_validation, y_train, y_validation = train_test_split(X_all, y_all, stratify = y_all,
                                                    test_size = validation_size, random_state = 123)
print("Train set 'yes' pct = {:.2f}%".format(100 * (y_train == 1).mean()))
print("Validation  set 'yes' pct = {:.2f}%".format(100 * (y_validation == 1).mean()))

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Validation set has {} samples.".format(X_validation.shape[0]))
```

```
Train set 'yes' pct = 11.27%
Validation  set 'yes' pct = 11.26%
Training set has 28831 samples.
Validation set has 12357 samples.
```

Implementation

The project follows typical predictive analytics hierarchy as shown in Fig. 9:

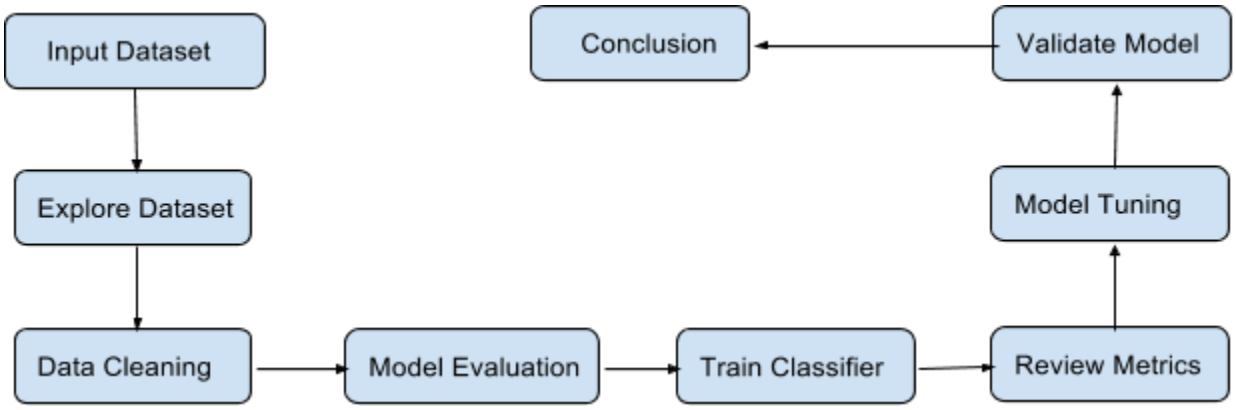


Fig. 9 Project Flow

Following the direction of the arrow as shown, with the dataset we chose, we first performed exploratory analysis by seeing distribution of some of the feature that seems to be relevant for the problem we plan to solve. We also did a check of correlations using scatterplot matrix in this process. We then performed data cleaning steps to identify columns with highest amount of null values and make a decision if we want to fill the missing values or completely drop the columns from the analysis. We processed the data through a pre-processing function to identify features that have values like 'yes', 'no','unknown' and convert them to 1, 0, np.nan. For categorical features we converted them to dummy variables so that the dataset becomes fully numeric and it then is easy to process it through our chosen algorithm for meaningful outcome.

Before we jump into the model evaluation, we split our full dataset into training and validation splits with a 70:30 ratio and then perform another check to make sure the splits have equal percent of responses to avoid deviations in the classification iterations. We used tree based algorithms to perform a check on accuracy and standard error metrics to pick a model that performs best. Typically Random Forest and XGBoost are well known to perform at best in usual supervised machine learning applications and we did confirm this by observing that XGBoost indeed has highest accuracy.One key observation we made was the accuracy of XGBoost sklearn wrapper was better compared to the XGBoost from dmlc library.

Once we benchmarked the accuracy of untuned XGBoost model, we then performed tuning of XGBoost model's hyperparameters in batches using GridSearchCV. This way we could keep track of best outcomes of each parameter and apply it to the next batch.

3. Evaluating Models

We don't know which algorithms would be good on this problem or what configurations to use. So let's pick a few algorithms to evaluate.

- Logistic Regression (LR)
- Classification and Regression Trees (CART)
- Random Forests (RF)
- Adaptive Boosting (AB)
- Extreme Gradient Boosting (XGB)

We are using 10-fold cross validation to estimate accuracy. This will split our dataset 10 parts, train on 9 and test on 1 and repeat for all combinations of train-test splits.

Also, we are using the metric of `accuracy` to evaluate models. This is a ratio of the number of correctly predicted instances in divided by the total number of instances in the dataset multiplied by 100 to give a percentage (e.g. 95% accurate). We will be using the `scoring` variable when we run build and evaluate each model next.

Test to check how XGBoost compares with SKLearn wrapper

```
# Test to check how XGBoost compares with SKLearn wrapper

from xgboost.sklearn import XGBClassifier
clf = XGBClassifier()
param = clf.get_xgb_params()
clf.fit(X_train, y_train)
preds_sk = clf.predict(X_validation)

import xgboost as xgb
dtrain = xgb.DMatrix(X_train, label=y_train)
dvalid = xgb.DMatrix(X_validation)
bst = xgb.train(param, dtrain)
preds = bst.predict(dvalid).round()

print("Accurary of sklearn wrapper: ",accuracy_score(y_validation, preds_sk))
print("Accuracy of XGBoost library: ",accuracy_score(y_validation, preds.round()))
print("-----------------------------------------")
print("sklearn predictions: ", preds_sk)
print("XGB library predictions: ",preds)
```

```
Accurary of sklearn wrapper:  0.9184268026219956
Accuracy of XGBoost library:  0.9117099619648782
-----------------------------------------
sklearn predictions:  [1 0 0 ... 0 0 0]
XGB library predictions:  [1. 0. 0. ... 0. 0. 0.]
```

Accuracy of sklearn wrapper is much better so we would use that going forward.

Build Models

```
scoring = 'accuracy'

#Spot check algorithms
models = []
models.append(('1. LR', LogisticRegression()))
models.append(('2. CART', DecisionTreeClassifier()))
models.append(('3. RF', RandomForestClassifier(n_estimators=100)))
models.append(('4. AB', AdaBoostClassifier(RandomForestClassifier(n_estimators=100),
                                           algorithm='SAMME',n_estimators=100, learning_rate=1.0)))
#models.append(('5. XGB', XGBClassifier()))
# Evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = model_selection.KFold(n_splits = 10, random_state = 123)
    cv_results = model_selection.cross_val_score(model, X_train, y_train,
                                                 cv = kfold,
                                                  scoring = scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

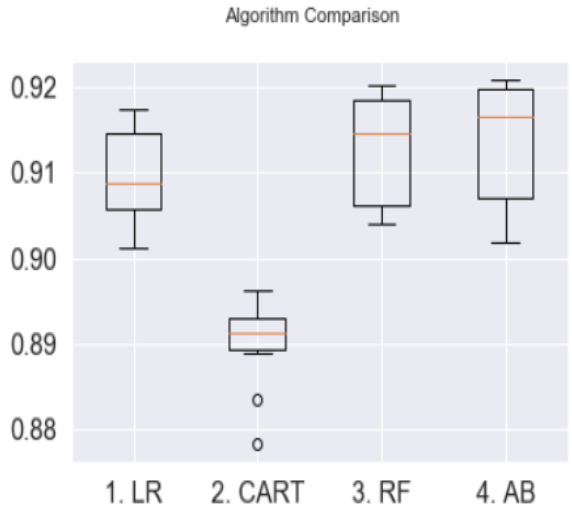
```
1. LR: 0.909681 (0.005114)
2. CART: 0.890014 (0.005160)
3. RF: 0.912802 (0.006164)
4. AB: 0.913843 (0.007151)
```

Select Best Model

We now have 5 models and accuracy estimations for each. We need to compare the models to each other and select the most accurate. The output of above code cell shows XGB being the winner with highest estimated accuracy score.

The plot below shows model evaluation results and compare the spread and the mean accuracy of each model. There is a population of accuracy measures for each algorithm beacuse each algortihm was evaluated 10 times (10 fold cross validation)

```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



The box and whisker plots are aligned at the top of the range near ~91% mark but appears `CART` is low performer and hovering in the ~89% range.

Refinement

We will prepare the data by splitting feature and target/label columns and also check for quality of given data and perform data cleaning. To check if the model we created is any good, we will split the data into training and validation sets to check the accuracy of the best model. We will split the given training data in two ,70% of which will be used to train our models and 30% we will hold back as a validation set.

We performed hyper tuning of parameters of XGBoost wrapper from scikit-learn library and the parameters tuned were shown in below table:

Parameter	Description	Values Tested	Best Value
max_depth	maximum depth of a tree to control overfitting	(3,4,5,6,7,8,9)	5
min_child_weight	minimum sum of weights of all observations required in a child	(2,3,4,5,6,7)	3
gamma	minimum loss reduction required to make a split	(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.9,1.0)	0.1
reg_alpha	L1 regularization term on weights	(0,2,4)	2
reg_lambda	L2 regularization term on weights	(1,3,5	5
subsample	Subsample ratio of the training instance	(0.5,1.0)	1
colsample_bytree	Subsample ratio of columns when constructing each tree	(0.5,1.0)	1
colsample_bylevel	Subsample ratio of columns for each split, in each level	(0.5,1.0)	1

Complete code of the simulation can be found here:

One refinement we could do was to improve recall rate for minority class labels (1 in our case) as the dataset was imbalanced. This may reduce the overall accuracy but the overall performance of model in predicting the outcome would be more robust.

II. Results

Model Evaluation and Validation

We applied and compared XGB model out of the box vs the rest of tree based models. The metrics we used are calculated using sklearn wrapper so can be trusted for the model performance. Our end goal was to have a tuned model that could beat the untuned benchmark which it did by a very fine margin. So the solution described below is satisfactory to our initial expectations. We generated a final model with above list of tuned parameters. The output of this tuned model came just about 0.2% higher in accuracy vs the untuned model. Code snippet of final model shown below:

```
# Select best model
# Make predictions on validation dataset using tuned parameters
tuned_model = XGBClassifier(silent=True, nthread=-1, max_delta_step=0.7, seed=0, objective='reg:linear',
                    max_depth=5, min_child_weight=3, gamma=0.1, reg_alpha=2, reg_lambda=5, subsample=1,
                          colsample_bytree=1, colsample_bylevel=1)
tuned_fit = tuned_model.fit(X_train, y_train)
tuned_pred = tuned_model.predict(X_validation)

print("Accuracy Score: ",accuracy_score(y_validation, tuned_pred))
print("-----------------------------------------------------------")
print("Confusion Matrix: \n",confusion_matrix(y_validation, tuned_pred))
print("-----------------------------------------------------------")
print("Classification Report: \n",classification_report(y_validation, tuned_pred))
```

```
Accuracy Score:  0.920288095816
-----------------------------------------------------------
Confusion Matrix:
 [[10615   350]
 [  635   757]]
-----------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

          0       0.94      0.97      0.96     10965
          1       0.68      0.54      0.61      1392

avg / total       0.91      0.92      0.92     12357
```

The recall rate is quite low as we discussed in above section. This can be improved by providing more weight to the positive labels `1`, this in turn decreases the overall accuracy but the model performance becomes robust to classify the outcome labels.

I split the dataset into just two sets train and validation despite that it could be divided into three as the dataset I had is not big enough to do that beside I use 4 different algorithms to pick the best of them which give more accuracy to the final result I obtain

Justification

There is room for improvement on the final results, the tuned final model made no significant improvement over the untuned model. There could be more ways we could improve the score, particularly by selecting a subset of features using the ranking obtained from observing feature importance ranking and then performing the same exercise we did as describe above. But this will be out of scope of this report for now.

	Benchmark Model (Untuned XGBClassifier Model)	**Final Model (Tuned XGBClassifier Model)**
Accuracy Score	0.9184	0.9202

III. Conclusion

Free-Form Visualization

The importance of each feature was visualized with the following plot:

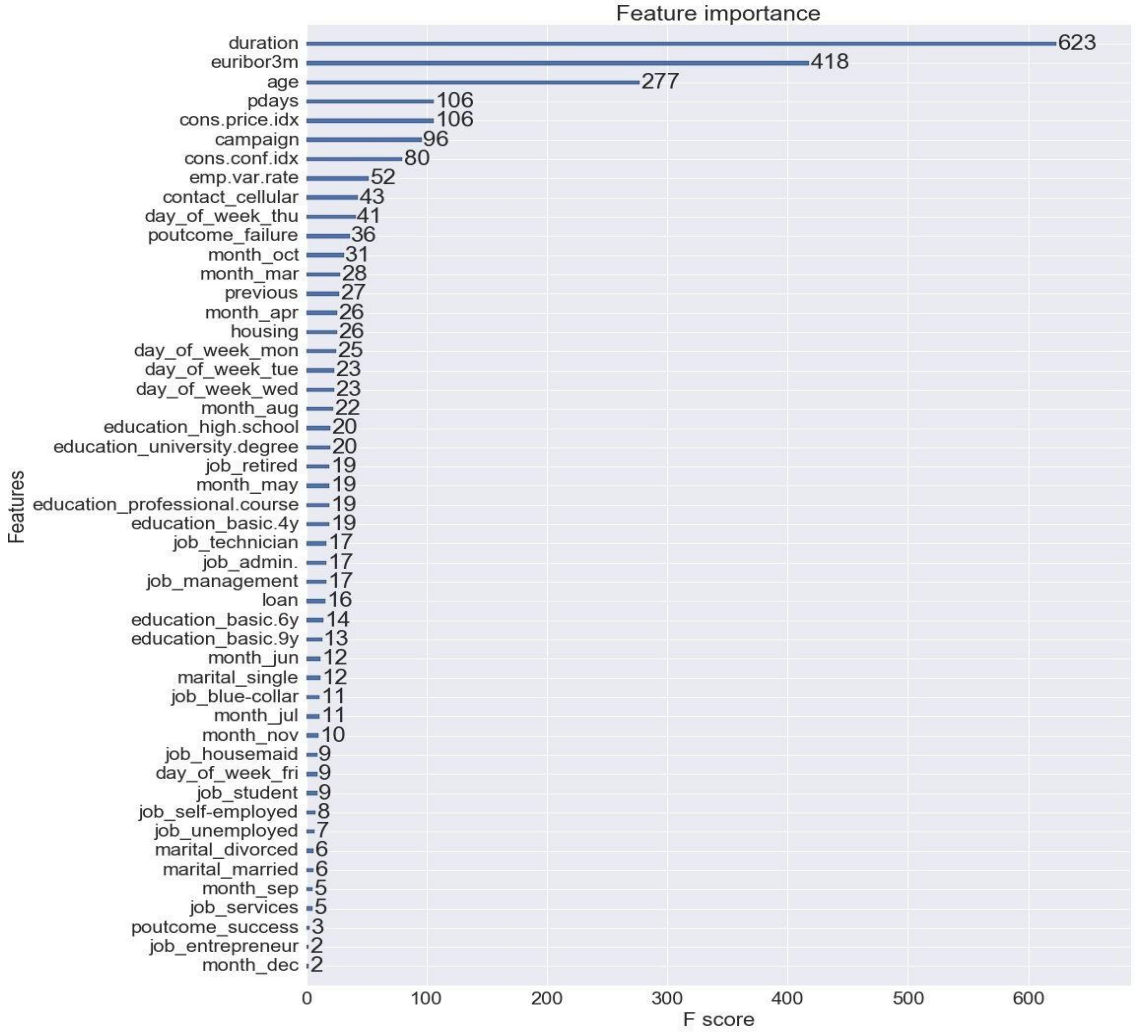


Fig.10 Important Feature ranking

Reflection

The most important and time consuming part of the problem was data cleansing and processing as there were multiple columns with null values. Once the data was prepared and ready, the next challenge was to pick an algorithm that could be best suited for the problem we choose to solve. With experience and

prior knowledge and also the outcome of accuracy on training data, we observed that XGBoost performed the best out of others.

If we look at the important predictors in their order of importance, as per the tree shown in Fig. 10, they are: duration, euribor3m, pdays, age, cons.price.idx, campaign, emp.var.rate, contact_cellular, and so on. We know characteristics of an entity are not in control of bank or any company manager as they are individual characteristics. Removing them from important predictors leaves us with duration, pdays, previous, housing, contact, and loan as option. Among these duration can be enhanced and in this case it's the most important predictor as well. Based on data analyzed and domain following are few suggestions for improving duration.

- Outbound call creates negative attitude towards bank due to the intrusion of privacy. Bank should decrease the outbound call rate and use inbound calls for cross-selling intelligently to increase the duration of the call. Agents may pitch about profits of term deposit for a particular client during inbound calls.
- Bank should have a clear promotion and they should put the value proposition up front – waiving fees, offering something free, or promoting a bundled service at a discounted price increases response rates versus just informing customers about a product. The value proposition should be right up front, with a clear call to action.
- “Duration” has positive effect on people saying “yes”. This is because the longer the conversations on the phone, the higher interest the customer will show to the term deposit. The Bank ought to focus on the potential clients who have significant call duration and moreover who have reacted emphatically amid the past campaign.

Improvements

One of the improvements we could do was to perform tuning to improve recall rate to improve overall prediction performance of the model. The other improvement could be to work with a subset of features that are high in variable importance pareto. Another key aspect would be to review metrics like log-loss to understand how quick the model is able to tune.

References

[1] Ou, C., Liu, C., Huang, J. and Zhong, N. ‘One Data mining for direct marketing’, Springer-Verlag Berlin Heidelberg, pp. 491–498., 2003.

[2] http://en.wikipedia.org/wiki/Direct_marketing. Wikipedia has a tool to generate citations for particular articles related to direct marketing.

[3] O'guinn, Thomas.” Advertising and Integrated Brand Promotion”. Oxford Oxfordshire: Oxford University Press. p. 625. ISBN 978-0-324-56862-2. , 2008.

[4] Petrison, L. A., Blattberg, R. C. and Wang, P. ‘Database marketing: Past present, and future’, Journal of Direct Marketing, 11, 4, 109–125, 1997.

[5] Eniafe Festus Ayetiran, “A Data Mining-Based Response Model for Target Selection in Direct Marketing”, I.J.Information Technology and Computer Science, 2012, 1, 9-18.

https://archive.ics.uci.edu/ml/datasets/bank+market