Using Machine Learning for Increased Efficiency in Marketing Reach-Outs Exemplified through publicly available bank data

Elijah Seeley

Abstract

There is a lot of time to be saved by prioritizing customers based on their susceptibility to a marketing campaign. This is why some companies contribute large portions of their budget just to make sure that the demographic is correct for a particular service or product. If the possible customer base is wide there is a loss of a lot of working hours contacting the incorrect customers. The goal of this project is to implore the idea of creating customer tiers derived from machine learning models in order to focus the marketing team that is doing the direct marketing onto customers that are likely to return value. This project is going off of Peruvian banking data, where the customers had recently been contacted for a marketing campaign to subscribe to term deposits with this bank. This documentation will be going over the basis of the steps that were taken to arrive at an appropriate tiering system based on the data as well as the development of a prototype model that can be iterated over in the future for new customers and an evolving dataframe.

Introduction

There are many ways to increase the efficiency of a direct marketing campaign. A simple straightforward way of doing this is to limit the amount of customers that will be contacted. Now, the idea of limiting the amount of customers may seem counterintuitive due to the nature of sales where more contacts lead to higher sales. However, this project shows that with machine learning models we are able to focus on the customers that have the highest likelihood of providing a successful result. We will do this by first identifying any key features if available, then cleaning and preprocessing the data, creating the model, adjusting the model, and developing a tier system for employees to work from. At the end of this documentation, there will also be an example of an api version of this model, for employees to run new data against, update, etc. The api version of this model is not completed, but we will go over the basics of what a completed version could accomplish and it’s benefit for the company and its campaigns.

The Data

This data is publicly available at
https://www.kaggle.com/janiobachmann/bank-marketing-dataset.
The data contain all types of information relevant to our goals.

age	job	marital	education	in_default	avg_yearly_balance	housing_loan	personal_loan	contact_method	day	month	duration	campaign_contacts
58	management	married	tertiary	no	2143	yes	no	unknown	5	5	261	1
44	technician	single	secondary	no	29	yes	no	unknown	5	5	151	1
33	entrepreneur	married	secondary	no	2	yes	yes	unknown	5	5	76	1
47	blue-collar	married	unknown	no	1506	yes	no	unknown	5	5	92	1
33	unknown	single	unknown	no	1	no	no	unknown	5	5	198	1

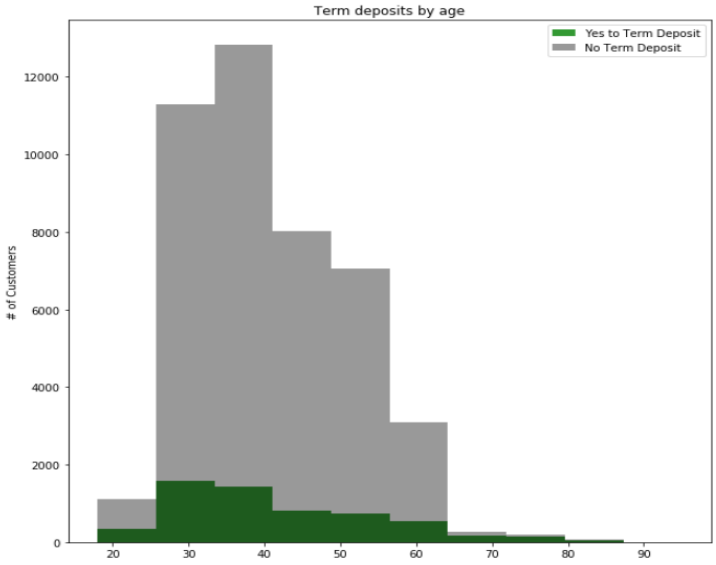
This includes demographic information for all of the customers such as education, marital status, age. Also, bank data, such as whether or not the customer has a personal or housing loan, their average yearly balance in their account, if they are in default or not. Importantly, the data contain marketing points of interest such as whether the previous campaign was successful, how long the customers were on the phone for during this reach-out and how many times the customers were contacted. With all of this information we are able to make a few assumptions based on the likelihood of a customer to subscribe to a term deposit, however, mainly we are not able to determine exact causes. There are some correlations, such as students and the retired being most apt to subscribe to a term deposit, although the most signifying feature we have available to us is whether or not the previous campaign was successful. We saw the greatest correlation between term deposit subscriptions and previous campaign successes. This makes sense as customers who have built trust and agreed to products and services before are most likely to do repeat.

The data provided didn't need any cleaning. It came packaged neatly without extraordinarily incorrect values, missing values, misspellings, etc. However, the features of the dataframe were renamed, for ease of reading and comprehension.

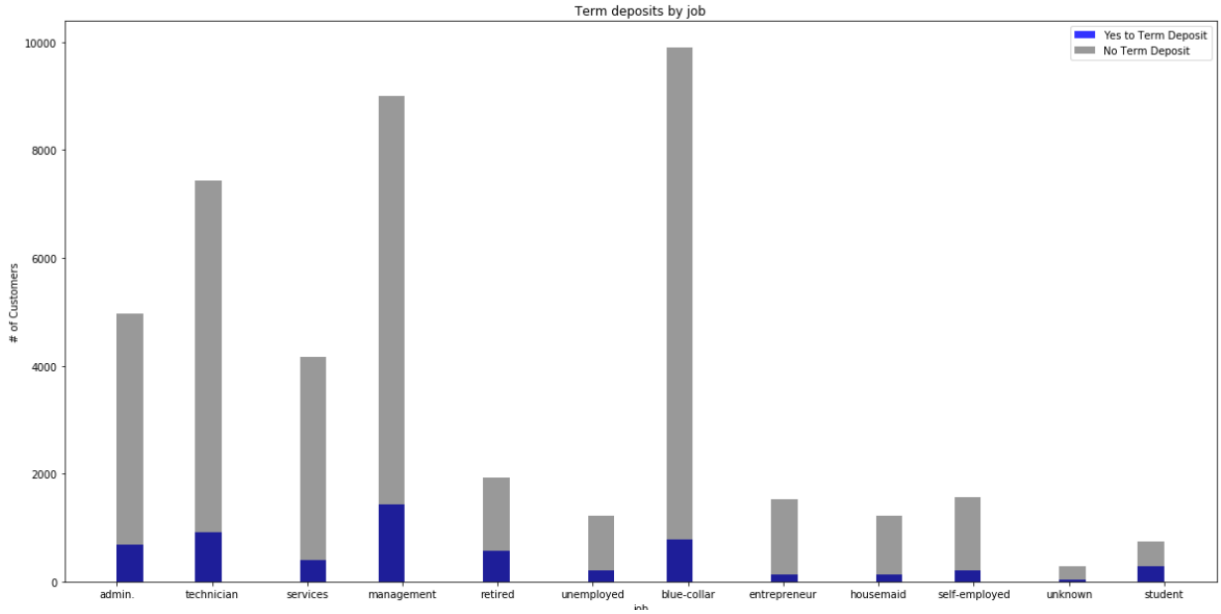
```
df.rename(columns = {'y':'term_deposit', 'pdays':'prev_days', 'poutcome': 'prev_outcome', 'contact': 'contact_method', 'default'
          'campaign':'campaign_contacts', 'housing':'housing_loan', 'loan': 'personal_loan'}, inplace = True, errors = 'raise')
```

Exploration

Unfortunately initial exploration of the data did not derive anything spectacular. There was not enough trend in any particular to make any assumptions on how the model would approach determining term deposit predictability. However, some interesting things did surface.



We see here that the older the customer is, the larger percentage of those customers subscribe to a term deposit. At some point around 65 years of age, nearly all of the customers begin to subscribe to term deposits.



Here we see term deposits by occupation. Coupled with the following matrix, we see a small trend of retirees and students subscribing to term deposits, but everywhere else the numbers don’t follow any particular order.

	Coefficient	Abs_Coefficient
age	0.010840	0.010840
avg_yearly_balance	0.029405	0.029405
day	-0.005246	0.005246
month	-0.017539	0.017539
duration	0.859012	0.859012
campaign_contacts	-4.809073	4.809073
prev_days	-0.000106	0.000106
previous_contacts	0.051427	0.051427
job_admin.	0.193715	0.193715
job_blue-collar	-0.148794	0.148794
job_entrepreneur	-0.309368	0.309368
job_housemaid	-0.371037	0.371037
job_management	-0.000304	0.000304
job_retired	0.582960	0.582960
job_self-employed	-0.252542	0.252542
job_services	-0.113171	0.113171
job_student	0.684426	0.684426
job_technician	-0.018214	0.018214
job_unemployed	-0.019383	0.019383
job_unknown	-0.195707	0.195707
marital_divorced	0.033654	0.033654
marital_married	-0.165645	0.165645
marital_single	0.164571	0.164571
education_primary	-0.240654	0.240654
education_secondary	-0.012608	0.012608
education_tertiary	0.235825	0.235825
education_unknown	0.050017	0.050017
in_default_no	0.053313	0.053313
in_default_yes	-0.020733	0.020733
housing_loan_no	0.390091	0.390091
housing_loan_yes	-0.357510	0.357510
personal_loan_no	0.275839	0.275839
personal_loan_yes	-0.243259	0.243259
contact_method_cellular	0.417475	0.417475
contact_method_telephone	0.361502	0.361502
contact_method_unknown	-0.746397	0.746397
prev_outcome_failure	-0.545485	0.545485
prev_outcome_other	-0.243383	0.243383
prev_outcome_success	1.673286	1.673286
prev_outcome_unknown	-0.851837	0.851837

Here we also take note that one of the strongest indicators that a customer will subscribe to a term deposit is whether or not a campaign has had success on them previously. This will surely be of great use to the model in its predictions.

Preparing the Data for Modelling

All of the categorical data had one-hot encoding applied to it, while the numerical data had various forms of preprocessing. Because of the type of data we are dealing with, the numerical data ranged from columns that had numbers from 0 - 1, and numbers that went from negative thousands to tens of thousands. Because of this, a combination of scaling such as RobustScaler and MinMaxScaler were used depending on the feature in order for the model to appropriately understand what it is comparing.

Model Development

The initial model was built using logistic regression, this is because logistic regression is easy to get off of the ground fairly quickly and offers the opportunity to get an idea of how the model is going to respond to the data and what adjustments need to be made. The initial model was fairly responsive, we were able to get decent precision on the term

deposit predictability, however, this is not what the goal for this project is. It was at this point that the realization came where the model would have to emphasize recall. This is because the intentions of the model are not to limit the amount of success, just focus the employees to conjure the same amount of success with less time.
Below is the initial logistic regression model and its classification report.

```
#Logistic Regression CV
for i in range(10,21,2):
    clf = LogisticRegressionCV(cv=3, Cs = i, random_state=44, max_iter = 1500).fit(X_train, y_train)
    clf_predictions = clf.predict(X_test)
    clf_train_score = clf.score(X_train, y_train)
    clf_test_score = clf.score(X_test, y_test)
    print("Train Score: {}, Test Score: {} \n".format(clf_train_score, clf_test_score))
    print("Cs: {}. \n{}".format(i, classification_report(y_test, clf_predictions)))
```

```
Train Score: 0.8998621165115478, Test Score: 0.9012064343163538

Cs: 10.
              precision    recall  f1-score   support

          no       0.91      0.98      0.95     13107
         yes       0.70      0.33      0.45      1813

    accuracy                           0.90     14920
   macro avg       0.81      0.65      0.70     14920
weighted avg       0.89      0.90      0.89     14920

Train Score: 0.8998333907847869, Test Score: 0.9012734584450403

Cs: 12.
              precision    recall  f1-score   support

          no       0.91      0.98      0.95     13107
         yes       0.70      0.33      0.45      1813

    accuracy                           0.90     14920
   macro avg       0.81      0.66      0.70     14920
weighted avg       0.89      0.90      0.89     14920

Train Score: 0.8998046650580259, Test Score: 0.9015415549597855
```

The model was pivoted to random forest classification because, with the about 40 features that we ended up with, having decision trees seemed like the most likely avenue to produce the desired results of 100% recall.

```
from sklearn.ensemble import RandomForestClassifier

#To make later parameter tuning easier, let's make sure we check multiple depths.

for i in range(5,40,5):
    rfc = RandomForestClassifier(max_depth=i, random_state=44).fit(X_train, y_train)
    rfc_predictions = rfc.predict(X_test)
    rfc_train_score = rfc.score(X_train, y_train)
    rfc_test_score = rfc.score(X_test, y_test)
    print("Train Score: {}, Test Score: {} \n".format(rfc_train_score, rfc_test_score))
    print("Depth: {}. \n{}".format(i, classification_report(y_test, rfc_predictions)))
```

```
Depth: 35.
              precision    recall  f1-score   support

          no       0.93      0.98      0.95     13107
         yes       0.75      0.46      0.57      1813

    accuracy                           0.92     14920
   macro avg       0.84      0.72      0.76     14920
weighted avg       0.91      0.92      0.91     14920
```

```
#Grid Search
parameter_grid = {'n_estimators' : [100,300, 500], 'max_depth' : [10, 20, 30], 'max_features' : ['auto']}
rfc = RandomForestClassifier()
grid = GridSearchCV(estimator = rfc, param_grid = parameter_grid, scoring = 'accuracy').fit(X_train, y_train)
grid.best_params_
grid.cv_results_
```

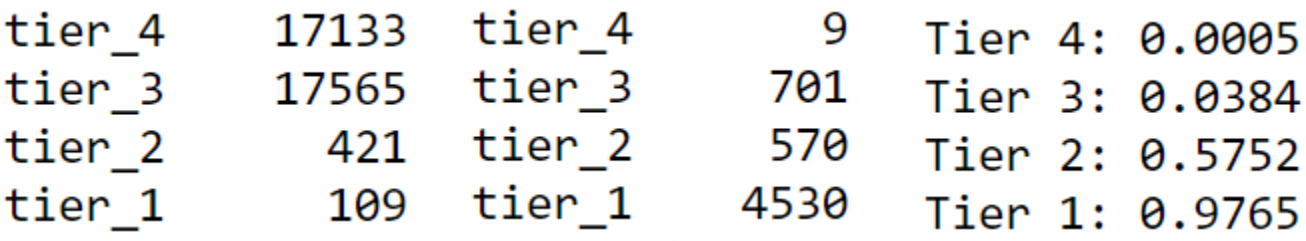
After performing a few grid searches to procure the best model, it was time to adjust again. The model as it was, was not going to be able to produce a perfect recall on the term deposits at the threshold that it was currently running off of. So, with some testing and lowering of the threshold, it was found that near perfect recall sat at the threshold of 1.3% for our data.

```
proba_thresh_list = [0.01, 0.025, 0.05,0.075,0.1,0.15,0.2,0.25,0.3]
rfc_best_proba = rfc_best.predict_proba(X_test)
for i in proba_thresh_list:
    rfc_best_lower_thresh = (rfc_best_proba[:,1] >i).astype('int')
    print("Probability Threshold: {}. \n{}".format(i, classification_report((y_test), rfc_best_lower_thresh)))
```

This meant that any customer that had at least a 1.3% probability of subscribing to a term deposit was considered a ‘yes’. Obviously, this is an absurdly low threshold, however, in the context of a marketing campaign, we can use multiple thresholds in order to prioritize the customers based on their likelihood. This is achieved by gathering all of the customers probabilities based on established thresholds and then tiering these customers in order to give priority to the ones that are most likely to respond well.

Customer Tiering

The customers' probability and assigned tier were then concatenated to a copy of the original dataframe. This would serve as material for the employees to use when beginning the campaign. In this example, the customers were tiered as follows:
Tier 1 (Highest Chance) 60% >
Tier 2 40% - 60%
Tier 3 1.3% - 40%
Tier 4 0% - 1.3%



No to term deposit / Yes to term deposit / percentage Yes of tier.

As we see, the tiering of the customer base was able to focus on the customers that are most likely to subscribe to a term deposit. Now, ideally, we would have been able to get all of the successful term deposits out of tier 4, however, in this scenario we were unable to secure everyone. We can see that if the campaign had simply ignored the tier 4 customers, that only 9 sales would have been lost, but over 17000 reach-outs could have been avoided. This is a significant amount of working hours that can be better used elsewhere than calling customers that are not going to be interested. Here we would recommend that the company focus first on tier 1 customers and then the rest as the campaign continued, but securing those profits first rather than later gives the company the opportunity to better understand it's customer base and allow for faster campaigns if for instance, tier 4 was ignored.

Adjusting Thresholds for Speed

There may come a situation where speed is the most important thing to a company, say for instance a company is in dire need of making sales before sinking, or generating enough capital for a new project as soon as possible. While we are fantasizing the extreme, it happens all of the time in various degrees. This model can be adjusted for speed simply by tuning the threshold to a higher cutoff. This will reduce the amount of subscriptions in this example, but the amount of time is greatly reduced in tandem.

```
Threshold: 1.3%
No Model Conversions: 5840.0
Model Conversions: 5840.0
No Model Minutes used: 215000.0
Model Minutes Used: 125560.0
    No Model Conversion Interval: 36.82
Model Conversion Interval: 21.5

Threshold: 5%
No Model Conversions: 5840.0
Model Conversions: 5723.2
No Model Minutes used: 215000.0
Model Minutes Used: 82032.53
    No Model Conversion Interval: 36.82
Model Conversion Interval: 14.33

Threshold: 10%
No Model Conversions: 5840.0
Model Conversions: 5489.6
No Model Minutes used: 215000.0
Model Minutes Used: 60526.36
    No Model Conversion Interval: 36.82
Model Conversion Interval: 11.03

Threshold: 15%
No Model Conversions: 5840.0
Model Conversions: 5256.0
No Model Minutes used: 215000.0
Model Minutes Used: 50224.0
    No Model Conversion Interval: 36.82
Model Conversion Interval: 9.56

Threshold: 20%
No Model Conversions: 5840.0
Model Conversions: 4964.0
No Model Minutes used: 215000.0
Model Minutes Used: 41853.33
    No Model Conversion Interval: 36.82
Model Conversion Interval: 8.43

Threshold: 25%
No Model Conversions: 5840.0
Model Conversions: 4672.0
No Model Minutes used: 215000.0
Model Minutes Used: 35874.29
    No Model Conversion Interval: 36.82
Model Conversion Interval: 7.68

Threshold: 30%
No Model Conversions: 5840.0
Model Conversions: 4321.6
No Model Minutes used: 215000.0
Model Minutes Used: 30971.47
    No Model Conversion Interval: 36.82
Model Conversion Interval: 7.17
```

To the left, we see that with our recommended threshold of 1.3% we are able to reduce campaign minutes used from 215000 minutes to a mere 125560 minutes by eliminating tier 4 (this report was generated through rounded numbers, therefore the 9 customers in tier 4 from before did not show as missed) while maintaining near all term deposits. The conversion interval represents the amount of minutes that pass in between term deposits (this number is derived by average call duration divided by number of contacts made). Simply by using the recommended 1.3% model, we are able to reduce conversion intervals from 36.82 minutes, to 21.5 minutes. If speed of sales is first priority, the campaign could in fact increase the model's threshold to 30% in this example and reduce total campaign minutes to an estimated 30971, compared to using no model, this is a savings of 3067 man-hours at the cost of 1518 term deposits. There is nothing that says that the company can not later on, reduce this threshold to pick up the remaining customers later. Or that two models cannot be run at different thresholds in order to tier the customers based on time frames either. The goal of this model has been fulfilled, we have theoretically increased the efficiency of the marketing campaign and developed a flexible model that can adapt to the company's needs.

Future Integrations - Developing an api

The launch of an application or api of this model would be of great use to the employees and the company. The ideal application would be simple enough for a (assumingly) non-technical marketing team to apply to their efforts without needing to refer to the developers of the application for assistance. This would go as far as adding new customers to the model and evaluating those customers without the need of additional code.

```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import RobustScaler, MinMaxScaler
import pickle
from flask import Flask
from sklearn.ensemble import RandomForestClassifier

df_load = pd.read_csv(r"C:\GitClonesEvol\Springboard\Capstone_Term_Deposit_Predictability\Peruvian_Bank_Data\Test_For_EXE.csv")

df = pd.get_dummies(data = df_load, columns = ['in_default', 'job', 'marital', 'education', 'contact_method', 'prev_outcome', 'housing_loan', 'personal_loan'])
df.drop(columns = ['term_deposit'], inplace = True)

def preproc(dataframe, column, scalertype):
    X = pd.DataFrame(dataframe[column])
    scaler = scalertype.fit_transform(X)
    dataframe[column] = scaler

#Scaling the incoming data
preproc(df, 'avg_yearly_balance', RobustScaler())
preproc(df, 'duration', RobustScaler())
preproc(df, 'prev_days', RobustScaler())
preproc(df, 'campaign_contacts', MinMaxScaler())
preproc(df, 'previous_contacts', MinMaxScaler())
preproc(df, 'age', MinMaxScaler())
preproc(df, 'campaign_contacts', MinMaxScaler())

preproc(df, 'avg_yearly_balance', RobustScaler())
preproc(df, 'duration', RobustScaler())
preproc(df, 'prev_days', RobustScaler())
preproc(df, 'campaign_contacts', MinMaxScaler())
preproc(df, 'previous_contacts', MinMaxScaler())
preproc(df, 'age', MinMaxScaler())
preproc(df, 'campaign_contacts', MinMaxScaler())

#loading model

loaded_rfc = pickle.load(open(r"C:\GitClonesEvol\Springboard\Capstone_Term_Deposit_Predictability\013_TermDeposit_rfc.sav", 'rb'))

predictions_probability = loaded_rfc.predict_proba(df)
predictions = loaded_rfc.predict(df)

df_load['probability'] = predictions_probability[:,1]
print(df_load.probability.head(30))
df_load['probability'].fillna(0)
df_load['probability'] = df_load['probability'].round(3)


df_load['tier'] = pd.cut(df_load['probability'],[-1, 0.013, 0.4, 0.6, 1.0],labels=['tier_4', 'tier_3', 'tier_2', 'tier_1'])
df_load = df_load.copy()
print(df_load.loc[df_load['tier'] == 'tier_1'])
print(df_load.tier.loc[df_load['term_deposit'] == 'yes'].value_counts().sort_index())
print(df_load.tier.loc[df_load['term_deposit'] == 'no'].value_counts().sort_index())
df_load.to_csv(r"C:\GitClonesEvol\Springboard\Capstone_Term_Deposit_Predictability\Exported_DataFrame.csv")
```

The transmutation of this model from its current state to an api was, at the time, a bit out of my wheelhouse. However, I was able to generate sample code of what the internals of this script would look like for loading a dataframe, processing that dataframe, and running it against the pickled model that was developed beforehand. My idea was to use flask and postman to deliver a finished product that would be able for anyone to try and attempt, this may be available in the future in which case I will update this document. Ideally, the api would allow for the addition of new data to be added to the

model in order to increase the effectiveness of this model for new and evolving customers. As the customers change, so will the campaigns, so the ability for this model to be flexible is imperative.

Conclusion

The goal of the model was fulfilled, increasing the efficiency of a marketing campaign by tiering the customers based on their threshold for success. To have saved nearly 89440 minutes with a threshold as low as 1.3% is more than enough to confirm that this model could be used in even the smallest capacity. When it comes to the use of working hours, I feel very strongly that we can use machine learning in almost all capacities to reduce wasted time, freeing up working hours for other important activities.