PROJECT REPORT

Customer Predictor

By - Mihir Ahlawat
Data Science Intern
1st June 2019 - 25th July 2019

Acknowledgement

I, Mihir Ahlawat would like to convey my gratitude to Mr. Sujit Bhattacharyya , Chief Digital Officer of Career Launcher, New Delhi for emphasizing on the 2-month Summer Internship Program and giving me the platform to interact with industry professionals. He enlightened me whenever required and helped me tremendously in the successful completion of this project. Moreover, we worked as an efficient team and learnt interpersonal skills as well.

Index

Introduction

Customer Predictor is a machine learning driven project which aims at predicting whether a user is a probable future customer or not. Customer prediction systems are a huge asset to a company. A firm may be able to take up decisions at a much more finer and profitable manner once it gets to know how its users think and how they respond to an action taken by them.
Customer predictor is able to read and understand, how a user spends his/her time on a website generating leads for us to gain insights about the user activity. The leads captured from each user are in the form of clicks, links of web pages visited, forms submitted, logins and logouts etc. We aim to use this data and apply data cleaning rigorously to be able to feed it into a machine learning model.

Dataset

The dataset used in this project is timestamped data of leads generated from the website for all user ids.

Following is the description of columns in given in the dataset -

id = uniquely identifies each website user

dt = timestamp of visit

event = page_view or struct (page_view is a hit on the site, struct is additional signals collected about the user from that page

If the value in event is struct then we have data in the following columns -

se_catgeory = Category of event (e.g Purchase is the event category for all events in the shopping zone

se_action = Action taken (e.g Product Detail means user has viewed the details of a product

se_label = More details about the event e.g prod title or category

se_property = more details e.g we use it sometimes to store the email of visitor

se_value = a numeric field (user_id or price) , not used every time

page = url visited

Data Exploration

We begin data cleaning by first exploring it. All the entries in the data are in a categorical format with time-step increments.
We use data.describe(include='O') to describe the object variables.

Column ‘event’ captures an entry as ‘page_view’ if a website is opened and captures an entry as ‘struct’ if a click has been made.
The value ‘Purchase’ in ‘se_category’ marks a visit to a product purchase page.
For this column ‘se_category’==’Purchase’, we have 4 unique values for ‘se_action’.
The 4 unique values are 'Product Detail', 'Init', 'Add to Cart' and 'Success'.
The occurrence of ‘Purchase’ and ‘Success’ together marks a successful purchase.
We counted the total number of unique ids that existed in our data(whether a customer or not). This value came out to be 7007.

The column ‘se_property’ contains mixed values of user emails and other details regarding the course product a user viewed.

Data Cleaning

Initially we are considering only the data of ids who have made a successful purchase.

On the first step of data cleaning, we need to fetch the Course names from ‘se_property’ column. So, we take the unique values other than the email ids in this column.
To do so, we maintain a list and insert all the values which do not have ‘@’ in them.
The unique categories in 'se_property' still have lots of dirty values, even after removing all the email-ids. So, we created a list of categories from the previous list by manual inspection and name this list as ‘Course_specific’.

Next up, we speculated that cookie-ids with multiple registered email-ids are possible cyber cafes or library computers. So, we need to remove such ids from our data as they would hamper the working of our model.
so, we counted the number of email ids registered per each unique id. To do so, we counted the number of values containing ‘@’ in them for each unique id. This count was stored in a dictionary mapped to their id number.

We find some ids with multiple associated email-ids. Now we count the number of purchase ids that exist when we set a limit to the maximum number of email-ids a cookie id can be registered with.
We select the number of email ids permissible per user id as less than equal to 3. We arrived at this number to keep it as low as possible without losing the generality of the model.

So finally we are left with 680 unique purchase ids.

We realise that multiple ids exist in our data which do not have many instances. We do not want our model to learn from such ids as they won't be depicting an accurate time series data.
So, we need to reduce the noise in data by removing ids with few instances.
For this we must come up with a cut-off value for the least number of instances that an id must have to be a part of the training data.
We make a list of customer ids which exist in our data by introducing multiple thresholds for the instance counts.
We discover only 2 unique ids exist with instances less than equal to 6 and made a purchase. So we consider only those ids who have at least 6 instances in the data.

Next step is to extract dates out of the date-time format and create a new column for the purchases(whether a person is a customer yet or not).
The given timestamps are in the format 'Y-m-d H:M:S'. But for our model, we would be needing only the date of the visit. So we will just keep the date part of timestamp in 'dt' variable.

In the given data, a successful purchase is captured by the instances where 'se_category'=='Purchase' and 'se_action'=='Success'. So we will create a new variable 'id2' and mark all the instances with a successful purchase.

Now we clean the columns and group multiple categories together.
To do so, we replace the categorical values from each column with an appropriate category name.

- The 'na' values are replaced by a 0,
- The list of course specific categories are labelled as one variable 'Course_specific',
- Label the variations of 'Submit' and 'Signup' categories as one variable.
- The categories which won't be needed in our model are labelled as 'Other'.

Feature Extraction

Now we one-hot encode the required categorical variables and drop the previous columns.
After this, we group the data on the basis of 'id' and 'dt' and sum them up. On summing up, we get the number of times a signal was recorded by an id on a day, for all the days that id generated a lead.
Then the grouped data is sorted on the basis of id first and then on date. Doing this ensures us that none of the instances of an id will overlap with that of other ids and the data will remain in chronological order as well.

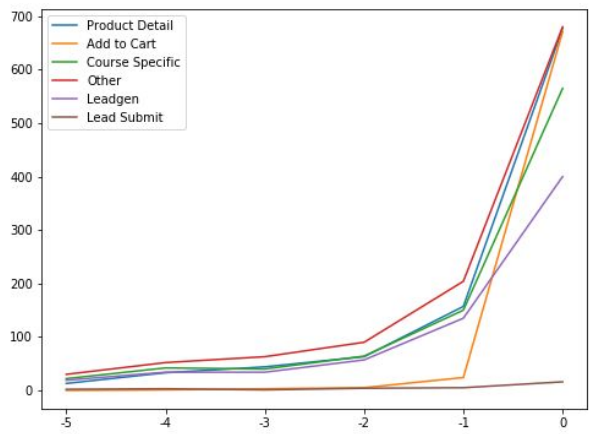
Our next step is to map the ids with their dates of purchase along with mapping each instance of an id with the relative date of their purchase.
First off, we Label Encode all the possible dates in our data so as to get a sequential numeric value for each date relative to the first date in our data which will be encoded as 0.

Now we map the ids with their date of purchase. Also we create a column for relative date of an instance with respect to the date of purchase of that id. The value of relative date would be 0,-1,-2,-3... for prior to a purchase and would be 1,2,3,4... for days after a purchase.
For our model, we would be using the data for only 5 days prior to a purchase. So, we remove the data with dates before 5 days prior to purchase date and after the date of purchase.

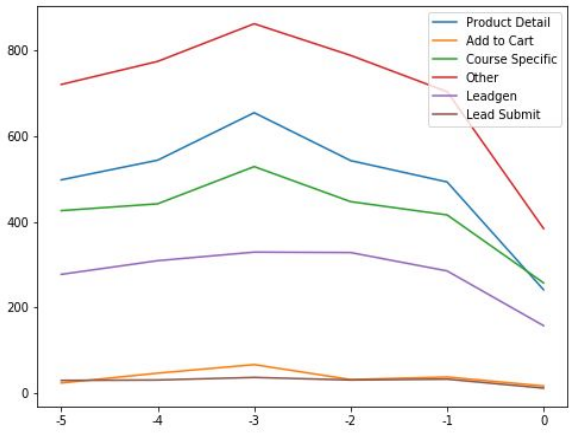
Now, we create a pivot table using our dataframe with index as ids and columns as relative dates. Pivoting a table would result in making columns out of a categorical value in a column. Thus on pivoting on columns as relative date, we get the count of a signal for each id on 5 days prior to a purchase.

Now as an exploratory step, we count the activity of each signal on the last 5 days prior to purchase. The count is taken as a sum for all ids.

We can see that the distribution of all these signals is distributed normally about the date of purchase, as expected. The activity for all these signals is maximum on the date of purchase and decreases subsequently. Though the same is not true for the ids who didn't make a purchase, as anticipated. The count of these signals does not follow a pattern.



IDs who made a purchase



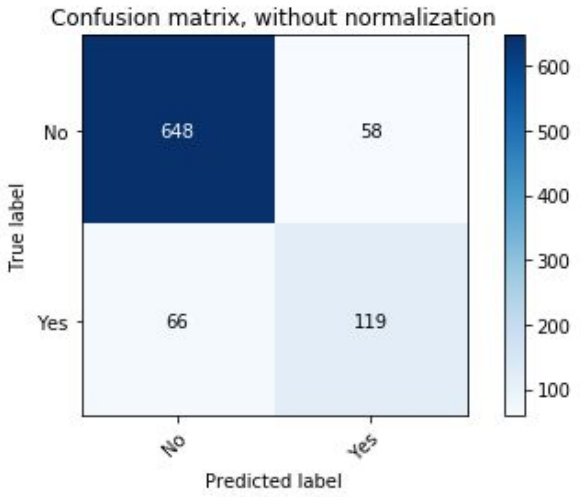
IDs who didn't make a purchase

Now we follow the above given steps for the data of ids who didn't make a purchase. We assume, the last date of the given data as the predicting purchase date for these ids.

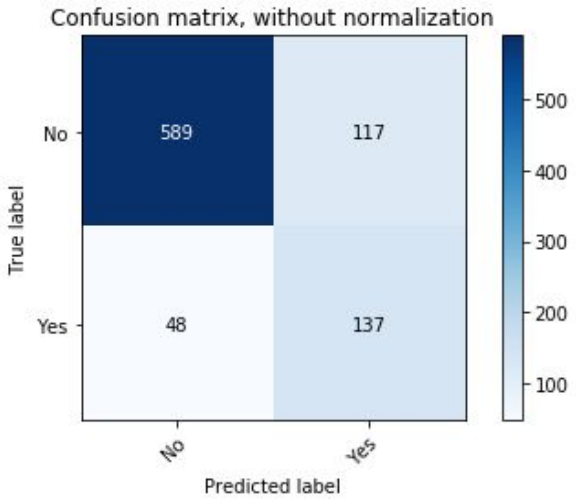
And finally we flatten the multi index columns we obtained after pivoting the dataframe to get one indexed column names and concatenate both the dataframes obtained from purchase and non purchase ids.

Building the Model

We split the data into feature vectors and target variable i.e. x and y.
Then we split the resulting data into training and testing data using train_test_split from sklearn. Finally we fit a RandomForestClassifier model from sklearn and tune the parameters.
We save our predictions into the 'preds' variable.Now we reduce the threshold for predicting values as positive using predicted_proba and set the probability to 0.25. Doing this allows the model to predict a person as a customer at a much lower threshold and thus reducing the False Negatives.
Finally we plot the confusion matrices for the models.



Confusion Matrix for the model



Confusion matrix for the model with lowered threshold

Results

While tuning the parameters we were able to reduce the False Negative values. Now to predict our probable customers we use the False Positives as a measure. These False Positives are the users whose leads are similar to a user who actually made a purchase. These False Positives are set to false as they haven't made a purchase yet and were labelled as a 'No' by us.
So, for our final output, we return a dataframe of ids who didn't make a purchase in our dataset but were classified as a buyer by our model i.e. the False Positives.