Bertelsmann/Arvato Project Report

by/ Noureldin Yosri

1. Project Definition

1.1 Project Overview

given the demographic data of people we want to classify them into one of two categories, those who will respond positively to an Ad and those who won't.

1.2 Problem Statement

Essentially we want to answer the question: How can the mail order company acquire new clients more efficiently?

using the provided demographics dataset and the dataset of existing customers we want to know which individuals are more likely to be customers of the mail company if targets with ads

examples of this and similar problems include google or facebook[1] ads which decide on whether you as a customer should see a specific ad or not, a similar but more general problem is Amazon/Netflix recommender systems.

1.3 Metrics

since this is a binary classification problem, several metrics can be used like accuracy, recall, F1-score and for training binary cross entropy loss, however the associated kaggle competition uses AUC[2] for the ROC curve metric defined as

"A ROC, or receiver operating characteristic, is a graphic used to plot the true positive rate (TPR, proportion of actual customers that are labeled as so) against the false positive rate (FPR, proportion of non-customers labeled as customers)."

for that reason instead of outputting class labels I output probabilities which I submit to kaggle.

2. Analysis

2.1 Data Exploration

in the first notebook "1. EDA" I explore the data, the first thing to note is that the data is has a large number of dimensions (366 to be exact), the column names are abbreviations of german words which made it more challenging to make sense of the data.

after examining the statistics of the columns I decided to:

---

[1] Recommending items to more than a billion people

[2] https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve

1. drop the columns which have small variation, the justification for this is that in those columns most entries are approximately the same value, thus have a low chance of explaining the data.
2. drop columns with large percentage of nulls/NaNs

of the 366 columns 6 columns were obviously categorical and are **CAMEO_DEU_2015, CAMEO_DEUG_2015, CAMEO_INTL_2015, D19_LETZTER_KAUF_BRANCHE, EINGEFUEGT_AM, OST_WEST_KZ** by exploring their values and using google translate to translate from german I could make these conclusions:

- **EINGEFUEGT_AM:** the data/time of creation of that entry in the table
- **OST_WEST_KZ:** east or west which most likely indicates whether the individual is from east or west germany
- **D19_LETZTER_KAUF_BRANCHE:** is likely the work sector of the individual
- **CAMEO_DEU_2015, CAMEO_DEUG_2015, CAMEO_INTL_2015:** if I'm not very much mistaken these are the results of running the data by CAMEO International: Customer Segmentation & Analysis where the codes have the meaning conveyed by this pdf

and for these columns I decided to:

- **EINGEFUEGT_AM:** drop it
- **OST_WEST_KZ:** replace each value with 1 if it's west and 0 otherwise
- **D19_LETZTER_KAUF_BRANCHE:** one hot encode this column because there is no ordering relation that can be defined on the enteries (so we can't replace them with numbers)
- for the CAMEO columns I consulted the pdf given above and concluded that the codes follow and order hence I just needed to convert the codes into numbers
- **CAMEO_DEUG_2015**: is the first char (number) of CAMEO_DEU_2015 #where 1 is the richest, 9 is the poorest and X is unknown so I cast the values into ints and replaced Xs with a special value
- **CAMEO_DEU_2015**: is code of two characters the first is CAMEO_DEUG the second is a character between A and E (and X for unknown) where E is the wealthiest and E is the poorest so I replaced each entry with ascii of the second character - ascii of 'A'
- **CAMEO_INTL_2015**: also classify from richest (1) to poorest (55) where XX is unknown which I dealt with in the same way as CAMEO_DEUG

the rest of the columns were either numeric or categorical but with all values numerically coded so it was almost impossible to distinguish between them except for clear ones (like a column whose values are either 19xx and 20xx which was obviously a year).

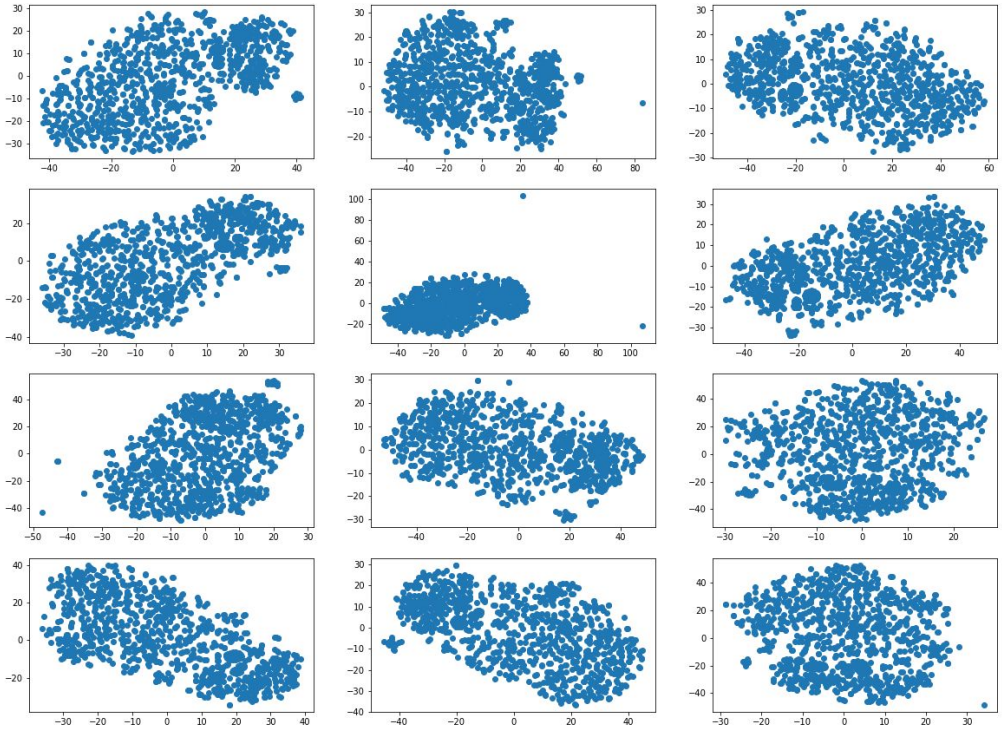
for the remaining columns those of which that still had NaNs had it in small percentages so it made sense to replace those NaNs with averages.

the last step was to record the average of each column and fit a min_max scalar to each column for normalization.

a programming note: in this notebook each step is mentioned twice in each cell the first the actual execution step, the second is nothing more than saving that step in a list, all those lists are saved in a dictionary 'transforms' that maps a column name to a list of transforms to be done on this column in order, those transformations can be replacing NaN with a const, dropping the column, normalizing it using the scalar fitted to the azdias dataset, one hot encoding it, the way this is done is that I wrap all these transformations into classes that inherit from a base class and implement the same interface, mainly an initialization (__init__) function and an apply function

2.2 Exploratory Visualization

in the second notebook '2. FeatureExtraction' I do most of the visualization, first I load and transform the data using the transforms dictionary from the first notebook, then I run tSNE to visualize the data in 2d, now what tSNE does is that it tries to project the data points from high dimensions to low dimensions which preserving topology (so that close points remain close and distant points remain distant), the problem with this is that it takes a long time since the data is large the complexity of tSNE is O(d*n^2) where n is the number of points and d is the original dimension, so it's impossible to run tSNE on the whole dataset, thus I ran it on several random subsets of the data producing the following visualization, which reveals that the points aren't clustered in well defined clusters but that there is a continuation from one dense area to another so that a clustering algorithm like k-means that assumes spherical distributions and that each dimension is as important as the other would likely fail.



the next step was to run principal component analysis PCA, which revealed that out of the ~370 dimensions resulting from the first transformation only 83 are enough to explain 98% of the variance of the data, thus a PCA that maps to 83 components was fitted to the data and saved.

the third notebook '3. prepare data' applies the transformations obtained from the previous two notebooks to the data and creates train/val/test csv files ready for training and testing

the way it does that is by merging the `Udacity_MAILOUT_052018_TRAIN.csv` and `Udacity_CUSTOMERS_052018.csv` files and assuming that customers have a high chance of reacting positively to our Ads they were given a score of 95% to reflect that confidence, the merged data set was then split randomly into train-val parts (with val data = 15% of the data)

2.4 Benchmark

in the fourth notebook '4. baseline model (xgb)' I fit an xgboost model to the training data, generate the predictions to the test data and submit to kaggle which gave a score of 0.63847 on kaggle

3. the Model

the problem can be restated as: given several hundred thousand points in 83 dimensional space we want to learn a function that maps those points to the real line, specifically to the range [0, 1] and minimize the binary cross entropy loss (since this is a classification problem after all)

so in the fifth and final notebook '5. train model' I train a multilayer perceptron network to learn the mapping with the following characteristics:

- each hidden layer has between 50 and 150 nodes
- the non-linearity applied is ReLU except on the last layer where it's Sigmoid
- a dropout of 0.3 is applied after each hidden layer to prevent overfitting
- an Adam optimizer with learning rate between 8e-4 and 3e-3 is used

a hyperparameter tuning job is created to learn the best parameters and the result of which is deployed to compute the predictions for the test data, which when submitted obtained a highest score of 0.70511 on kaggle (where the best score of any submission on kaggle is 0.80917