UDACITY MACHINE LEARNING NANODEGREE (MLND)

Capstone Project Report

Customer Segmentation – Arvato Financial Solutions

arvato
BERTELSMANN

by Beatriz Sousa

December 3rd, 2019

Table of Contents

Table of Figures

Project Overview

This project was one of the proposed capstone options for the Udacity Machine Learning Nanodegree. The goal for the project is to determine how one of Arvato's clients can acquire new customers for their mail-order organic products.

To help with this problem statement Arvato provides data on general population demographics, on their customers and on client response to previous campaign. This data is protected under terms and conditions and not shareable.

Using these datasets, we are proposed to predict which characteristics from individuals from the general population can be used to selectively target as good responders to this marketing campaign.

The project is broken down into 2 major subsections:

- Unsupervised Learning to identify segments of German population that match the existing customer segments;
- Supervised Learning to identify the likelihood of customer conversion from the general population

Completion of this project requires:

1. Creation of a Customers Segmentation model
2. Creation of a Supervised Learning model to qualify the performance of the predictions
3. Kaggle submission of the results obtained

All the supporting analysis and documentation (with the exception of the datasets) is available at Github.

Domain Background

Bertelsmann found its origins as a publishing house in 1835 (Schuler, 2010), and through steady growth and development made its way to the software and hardware distribution market in the 80's (Computerwoche, 1983). By 1999 the company received its current name Arvato Bertelsmann (Name, 1999) and over the next decade fully entered the domain of high-tech, information technology, and e-commerce services (Paperlein, 2012).

Arvato offers financial solutions in the form of diverse segments, from payment processing to risk management activities. It is in this domain that that this capstone project will be developed. Arvato is looking to use its available datasets to support a client (mail-order company selling organic products) in identifying the best data founded way to acquire new client base. To achieve this goal, I will explore Arvato's existing datasets to identify attributes and demographic features that can help segment customers of interest for this particular client.

Customer centric marketing is a growing field that benefits greatly from accurate segmentation, with the help of machine learning hidden patterns can be found in volumes that could easily be missed without computational help, requiring very little maintenance or human intervention, leading to an improved experience from customer seekers and customers alike.

Problem Statement

The problem statement for this project is "How can a client – mail order company selling organic products – acquire new clients in a more efficient way?".

The solution I propose for this problem is divided in 3 subproblems.

I will use an unsupervised learning approach to identify customer segments of value based on demographics data of existing customers versus general population data, and will follow-up on the discovered customer segments with a supervised learning approach using a dataset with demographics information for the target customers for the advertising campaign and predict which individuals would be more likely to convert to company customers.

Datasets and Inputs

All the datasets were provided by Arvato in the context of the Udacity Machine Learning Engineer Nanodegree, on the subject of Customer Acquisition / Targeted Advertising prediction models.

There are 4 datasets to be explored in this project:

- Udacity_AZDIAS_052018.csv: Demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns)
- Udacity_CUSTOMERS_052018.csv: Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns)
- Udacity_MAILOUT_052018_TRAIN.csv: Demographics data for individuals who were targets of a marketing campaign; 42 982 persons (rows) x 367 (columns).
- Udacity_MAILOUT_052018_TEST.csv: Demographics data for individuals who were targets of a marketing campaign; 42 833 persons (rows) x 366 (columns).

And 2 metadata files associated with these datasets:

- DIAS Information Levels — Attributes 2017.xlsx: a top-level list of attributes and descriptions, organized by informational category
- DIAS Attributes — Values 2017.xlsx: a detailed mapping of data values for each feature in alphabetical order

Which can help mapping the attributes to its type or missing value encoding.

Evaluation Metrics

This problem is a multi-class classification problem, and one the most valuable metrics to measure model performance is the Area Under the Curve Receiver Operating Characteristics (ROC-AUC). The curve represents a degree or measure of separability and, the higher the score the better the model is performing.

A great advantage of using RIOC-AUC is the immunity to class imbalance, which is the case for this problem. The number of people that are positive responders to an ad campaign are on average far lesser than those that respond negatively.

This is also the required evaluation metric for the Kaggle submission.

EDA and Preprocessing

Early Error Warnings

The first think we see when loading the data is that there is a mixed type warning and that there is an extra unnamed column that seems to be an index duplication:

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3058: DtypeWarning: Columns (19,20) have mixed t
ypes. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Figure 1-Mixed-type warning loading the data

I decided to deal with some of the most striking issues as they appear, so for the mixed-type error I decided to determine what were the names of the offending features.

```
# I will now check what is the problem with the columns 19 and 20
# getting the name of these columns
print(azdias.iloc[:,19:21].columns)
print(customers.iloc[:,19:21].columns)
```

```
Index(['CAMEO_DEUG_2015', 'CAMEO_INTL_2015'], dtype='object')
Index(['CAMEO_DEUG_2015', 'CAMEO_INTL_2015'], dtype='object')
```

Figure 2-Mixed-type error feature names

Unfortunately these columns were a mixture of strings, floats and a missing values marker compose by [‘X’] or [‘XX’], so I created a cleaning function (available in the utils.py file) to convert the strings to floats and the ‘X’ marker to np.nan.

Once this first warning was dealt with I decided to do some dataframe harmonization before I went over the identification of missing values and unknows. The customers dataframe had 3 columns that were overtly announced as not existing in the azdias dataframe so I dropped them. This brought the number of columns in azdias and customers to the same number:

Azdias Shape

```
# checking how the azdias dataframe looks like
print('Printing dataframe shape')
print(azdias.shape)
print('________________________________________________________________')

azdias.head()
```

```
Printing dataframe shape
(891221, 366)
```

Customers Shape

```
# checking how the customer dataframe looks like
print('Printing dataframe shape')
print(customers.shape)
print('________________________________________________________________')

customers.head()
```

```
Printing dataframe shape
(191652, 366)
```

Column Harmony between dfs

Figure 3-Comparison of the number of features between azdias and customers

Once the obvious different columns were removed from the customers dataframe I proceeded with checking how many of the attributes in the azdias dataframe had information in the DIAs metadata files. My reasoning is that the azdias dataframe might contain features that lack a description (but not importance) due to deficient data entries or evolving nomenclature of the attributes.

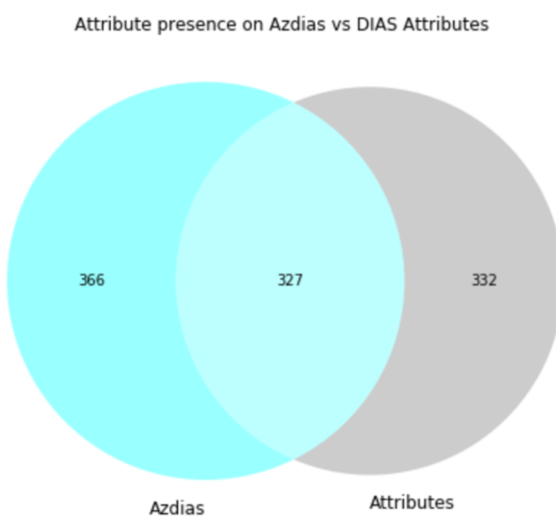

Figure 4-Venn Diagram of the overlap of column names between the azdias dataframe and the attributes in the DIAS file

Even though gut feeling would ideally have no place in data science or machine learning instinct showed me that there are in fact quite a few features in azdias whose meaning is not described in the metadata files.

It was at this point that it started to make sense to have some sort on function that checked if the dataframes were composed of the same column names as I processed them (and in this way guarantee that the dataframes were always shape compatible). The balance_checker() function is available for scrutiny and criticism at utils.py.

Missing Values

Moving on to treating missing data I identified 33493669 cells in the azdias dataframe that were missing data and 13864774 cells missing data in the customers dataframe. I created a function that identifies missing and unknown data based on the information provided in the DIAs file and replaces these tags with nans.

The results of running unknowns_to_NANs() are as follows:

```
print('Identified missing data in Azdias: ')
print('Pre-cleanup: ' + str(azdias_pre_cleanup.isnull().sum().sum()) +
      ' Post_cleanup: ' + str(azdias.isnull().sum().sum()))

print('Identified missing data in Customers: ')
print('Pre-cleanup: ' + str(customers_pre_cleanup.isnull().sum().sum()) +
      ' Post_cleanup: ' + str(customers.isnull().sum().sum()))
```

```
Identified missing data in Azdias:
Pre-cleanup: 33493669 Post_cleanup: 37088263
Identified missing data in Customers:
Pre-cleanup: 13864774 Post_cleanup: 14488721
```

Figure 5-Comparison on identified missing values before and after treatment with unknowns_to_NANs()

As and added step of exploration I also noticed that there were 93 columns after the cleanup step that were not missing values which means that just under a third of the columns are complete.

For the next step of cleanup, I dropped all the columns that had more than 30% of their data missing (I tried other percentages, but this was the one that lead to the best results further down in the prediction steps).

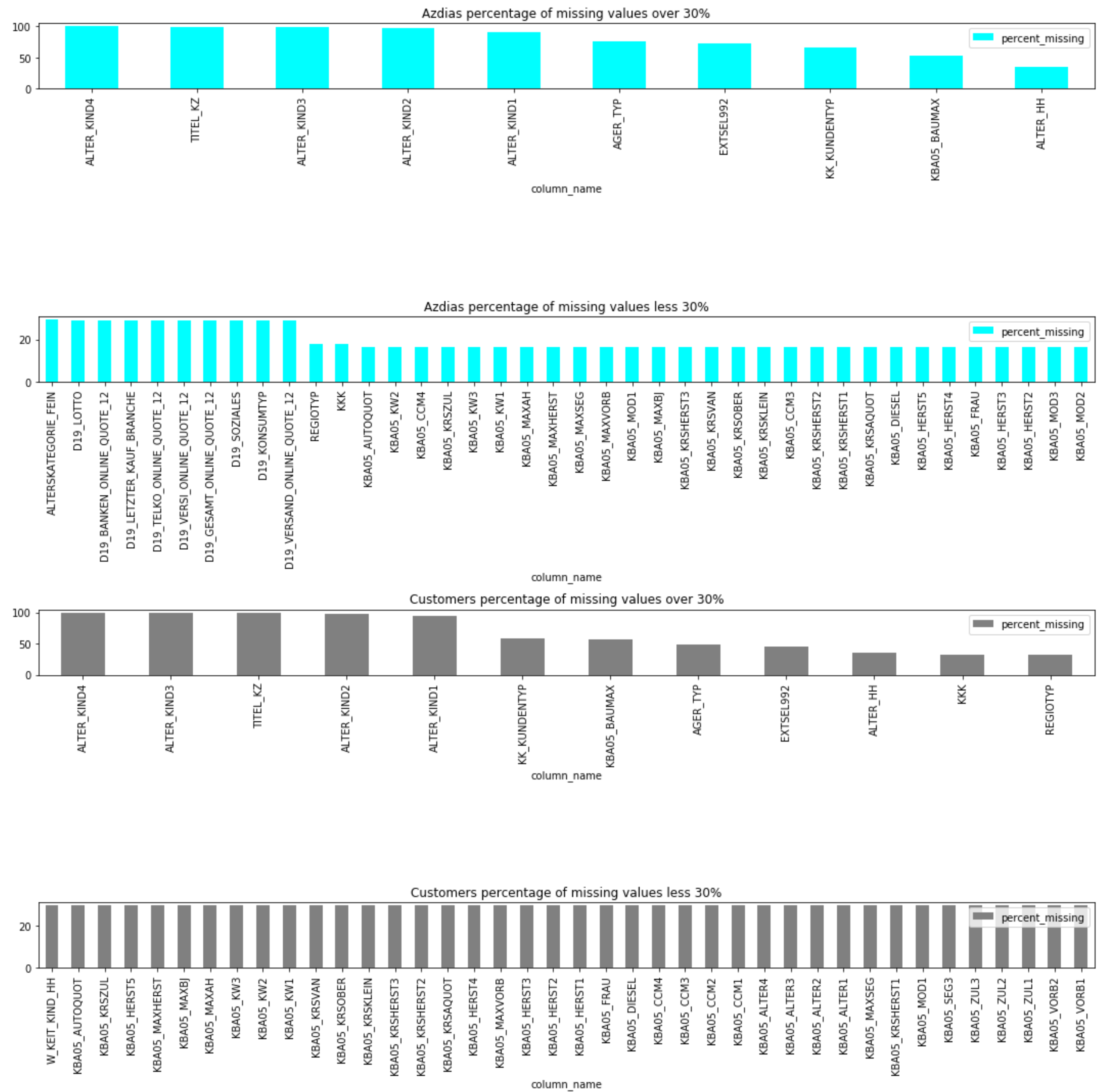

Figure 6-bar-plots demonstrating the distribution of columns missing over or under 30% of their data. For the columns missing less than 30% I am only displaying the closest to the 30% boundary.

Based on this information I dropped all the columns missing more than 30% of their entries, which lead to an obvious inconsistency between the azdias and the customers dataframe:

```
#since I just dropped several columns I will do another balance check
balance_checker(azdias, customers)

Feature balance between dfs?:  False
Your first argument df differs from the second on the following columns:
{'REGIOTYP', 'KKK'}
Your second argument df differs from the first on the following columns:
set()
```

Figure 7-Dropping the columns with more than 30% missing entries led to azdias retaining two more columns than customers

To correct this discrepancy, I dropped these 2 columns from azdias.

I also considered if it would be a good strategy to remove rows that were missing more than a certain number of entries and this exploration is available in my notebook, it was a strategy that I abandoned because it was very deleterious to the performance of my models.

Feature Encoding and Engineering

Finally, as a last step in the preprocessing portion I performed some Feature Encoding and Engineering.

- 'CAMEO_DEU_2015' is a categorical feature that ranged from 1 to 9 and A to F I used a LabelEncoder() to encode the categories to ints.
- ‘ANREDE_KZ’ refers to lifestyle characteristics and I used One Hot Encoding to encoded it.
- ‘OST_WEST_KZ’ is a binary feature that had the values array ['W', 'O'], which I mapped to: {'W':0, 'O':1}
- ‘EINGEFUEGT_AM’ is a time related feature, so I converted it to a datetime object and extracted the year.
- I created 2 different features from 'PRAEGENDE_JUGENDJAHRE', a ‘DECADE’ feature and a type of ‘MOVEMENT’ feature (avant-garde or not)
- ‘WOHNLAGE’ refers to neighborhood area, from very good to poor; rural so I created 2 different features from this one ‘QUALITY’ related to the quality of the borough and ‘AREA’ to identify if it is rural or not
- 'LP_LEBENSPHASE_FEIN' was used to create two new features, one related to life stage, 'LP_LEBENSPHASE_FEIN_life_stage' and one related to the wealth scale 'LP_LEBENSPHASE_FEIN_fine_scale'

Once I had created and encoded all these features I used SimpleImputer() to impute the nans with the most-frequent values.

Dimensionality Reduction

It as said by someone else much better than I probably can: “As the number of features increase, the number of samples also increases proportionally. The more features we have, the greater number of samples we will need to have all combinations of feature values well represented in our sample.” (Raj, 2019) and this can make datasets incredibly complex and prone to issues like overfitting (a model becomes so complex that it becomes hard to generalize it to anything else). Dimensionality reduction leads to the creation of models that are more accurate due to the reduction of misleading data, it leads to the need of less computing power and storage and a reduction in feature noise.

I tested quite a few methods to scale my data (standard scaler, min max scaler and robust scaler), for this problem the performances of the standard scaler and min max scaler were identical. Once I had my data scaled, I used principal component analysis (PCA) (Brems, 2017) which is a method of dimensionality reduction through feature extraction. To decide how many components, I should keep that accounted for over 80% of the variance observed in the results, I used a scree plot (line plot of the eigenvalues of factors or principal components in an analysis).

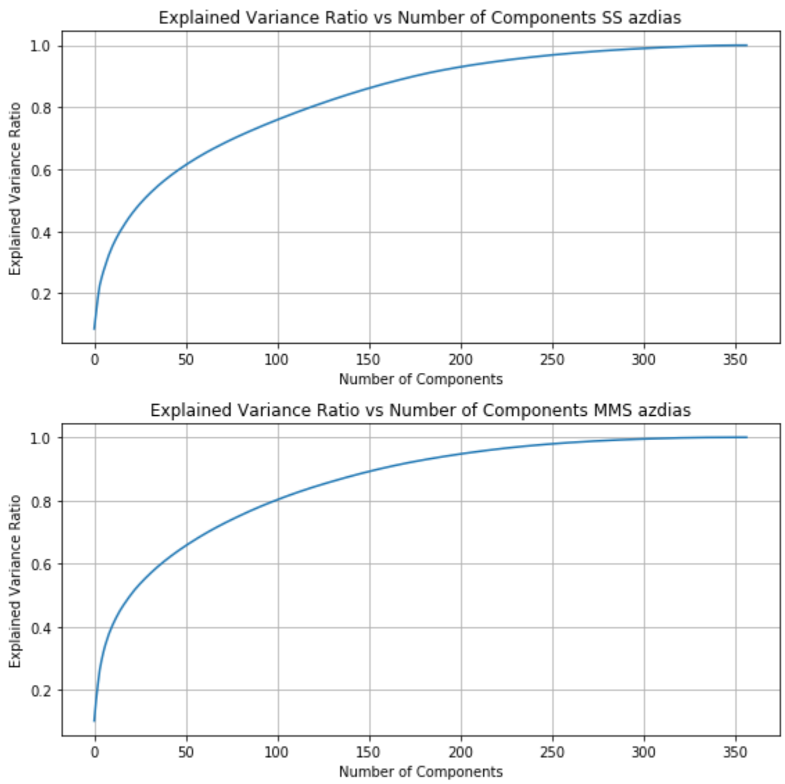

Figure 8-Scree-plots using standardscaler or minmaxscaler.

The highest the weight of an attribute the more relevant it is, let's look at the most important features for a few dimensions considering that positive weights might relate to a positive relationship and negative weights a negative one:

- dimension 1(0) using standard scaler:

  These are some of features related to positive weights:

  - MOBI_RASTER: refers to the individual's mobility
  - KBA13_ANTG1: lower share of car owners
  - PLZ8_ANTG1: lower number of 1-2 family houses

  And these are some of the features related to negative weights:

  - KBA13_ANTG4: refers to possession of higher number of cars
  - PLZ-ANTG3: number of 6-10 family houses in the PLZ8

- CAMEO_DEU_2015: detailed classification of cultural and living status

So overall the first dimension refers to the social status and living conditions of the individuals present in the dataset.

Interesting to see that even though different features were selected for dimensionality reduction, but they overall have the same meaning.

Also, based on the scree-plots:

- using standard scaler with 150 principal components 90% of the original variance can be represented
- using minmax scaler with 150 components we represent 90% of the original variance

Algorithms and Techniques

For the Unsupervised Learning version of the project I will be using Principal Component Analysis algorithm for dimensionality reduction. At its core, PCA allows the decomposition and transformation of data into the most relevant component though the principal that the more variance a feature has, the greater is the power of understanding of the data (Ding, 2004). Once the data is in its principal components I will implement K-means as a clustering approach.

Customer Segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics, for this purpose K-means is a simple and fast approach that fits well this problem since it scales quite well to large datasets. The principle of K-means implementation is quite elegant:

- Specify number of clusters K (that in this case are obtained through optimization using the elbow method)
- Initialize centroids by first shuffling the dataset and select random K points
- Iterate until the centroids don't change anymore.

It is important to note that in the case of K-means there is no ground truth to evaluate model's performance there is no single right answer to evaluation since for instance the number of k clusters is a hyperparameter input.

I used the Elbow Method to optimize the number of clusters that were ideal for this dataset. The concept behind the elbow method is to run the k-means clustering on the dataset on ranges of k while calculating the sum of squared errors for each k tested. Once plotted the like looks like (you probably guessed it by now!) an elbow. The elbow point represents the point of diminishing returns when the k increases (since there is nothing better than a demo check figure 9 for the elbow method implemented on this problem, it truly encapsulates how it is supposed to look like) (Gove, 2017).

For the Supervised portion of this analysis I decided to use a parallel testing approach. Customer segmentation is an extremely diverse field, which means that there are many models that have the potential to be the ideal on.

Customer Segmentation can be defined as: "Dividing the target market or customers on the basis of some significant features which could help a company sell more products in less marketing expenses.", and there are 2 ways to approach segmentation. Objective, which is supervised and non-objective which is unsupervised. Our case is Objective since we have a well stated base for the segmentation.

- Logistic Regression: simply put it uses a 1 or 0 indicator in historical campaign data (the responses we have in our mailout datasets) to indicate if a customer responded or not to the campaign and in this way create segments
- RandomForestClassifier: the workflow to random forest includes generating a random target vector of 0s and 1s, building a classifier fitted to the random target vectors and counting how many times observations fall in the same terminal node. It then measures the "proximity" between the observations. This will be a curiosity-based implementation as sometimes RFC implementation in imbalanced datasets does not return the best results
- XGBClassifier: I had a lot of high hopes to test this model. Looking at Kaggle it is impressive how many problem solutions are won with this model. It is an implementation of gradient boosted decision trees which is documented to deal better with some of the issues raised for the random forest classifier since Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. New models are created that predict the residuals or errors of prior models and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. It is touted everywhere as incredibly fast and having great performance but, I will spoil you here, it took me whopping 24 hours to run it. The results are incredible, there are no doubts there, but I quite literally can't use my machine for anything else when using the Bayesian approach for hyperparameter tuning due to how intensive computation is.
- LGBMClassifier: like XGB Light GBM is a gradient boosting framework jus like the one I just described for XGB that uses tree-based learning algorithm, but it grows trees leaf-wise while other algorithm grow level-wise

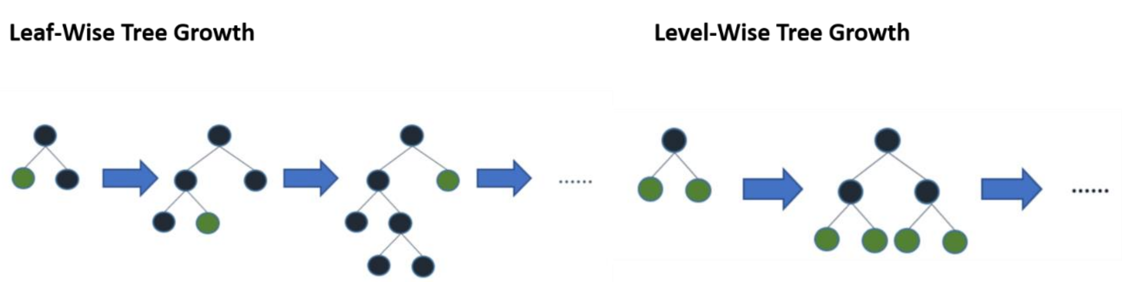

Figure 9-Comparison of Gradient Boosted models, LGBM with leaf-wise growth versus level-wise growth

- GradientBoostingClassifier: fundamentally the same as XGB as it is another gradient boosting model, except not as fast, it can't be parallelized (or at least not by default due to how sklearn handles memory), also, it does not perform tree regularization like XGB to avoid data overfitting.
- MLPClassifier: I thought it would be important to include at least one neural net-based model. By my own experience sometimes it overcomplicates solutions and they do not lead necessarily to better results, (I recall that in NLP for example simplistic approaches lead to better results) but I though it was worth to give it a try. MLP trains using back propagation (the weights of different neurons are updated in a way that the difference between the desired and predicted output is as small as possible)

Benchmark

To determine how changes in data processing and model hyperparameter tuning were affecting my models I decided to first create base scores of the different models on cleaned but unscaled data. Out of all the models used in this solution LR is the simplest and more straightforward model to use as a benchmark.

	Unscaled		StandardScaler		MinMaxScaler	
	Model	Score	Model	Score	Model	Score
0	LR	0.6660	standardLR	0.6613	minmaxLR	0.6745
1	RF	0.6351	standardRF	0.6376	minmaxRF	0.6352
2	XGB	0.7607	standardXGB	0.7607	minmaxXGB	0.7607
3	LGBM	0.7117	standardLGBM	0.7191	minmaxLGBM	0.7117
4	GB	0.7571	standardGB	0.7571	minmaxGB	0.7571
5	MLP	0.6026	standardMLP	0.5915	minmaxMLP	0.5914

Figure 10-Model performance output depending on the scaling method.

I created a reference table to the non-optimized models to be compared against the LR results.

Customer Segmentation Report

Well, the base notebook provided by Udacity opens this section with "The main bulk of your analysis will come in this part of the project." Which although true, it does not truly represent the time investment you have in each section, this was possibly the least time-consuming portion of the project.

At this point I had chosen how many components to keep from my PCA (150) and I had decided to use the standard scaler to scale my data, for the clustering portion I decided to use K-means (Garbade, 2018) since it is one the most popular algorithms used for this type of clustering and there were copious amounts of resources to help me on the way if there were any hiccups.

One of the first steps with K-means is to define a k (number of centroids) that are the imaginary centers for each cluster. To help me select an optimal k I will use the Elbow method. The Elbow method compares the within cluster sum of squares until there is no more visible improvement. This is quite visually striking when using plots:

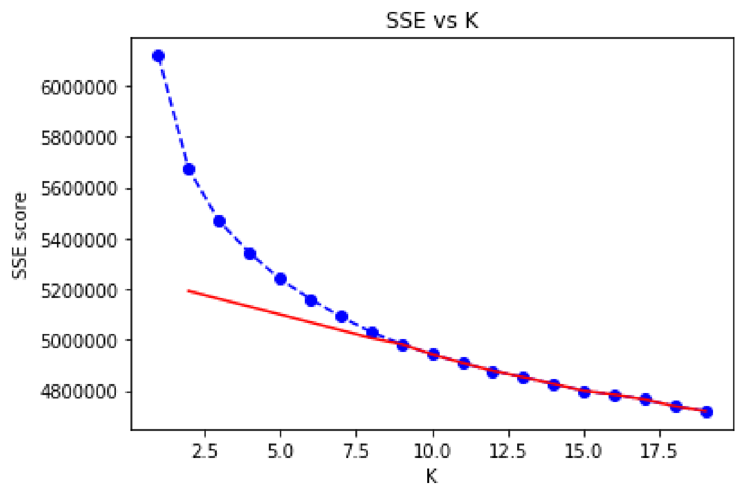

Figure 11-Elbow plot, it is visible that after the 9th cluster the gradient on the SSE score becomes lesser as you move to the next cluster

Based on the Elbow method we know that for these datasets and the data cleaning I implemented k = 9 seems to be the ideal.

After fitting kmeans to 9 clusters and my data I found that in fact there are a few clusters that are more proportionally populated with customers when compared to the general population:

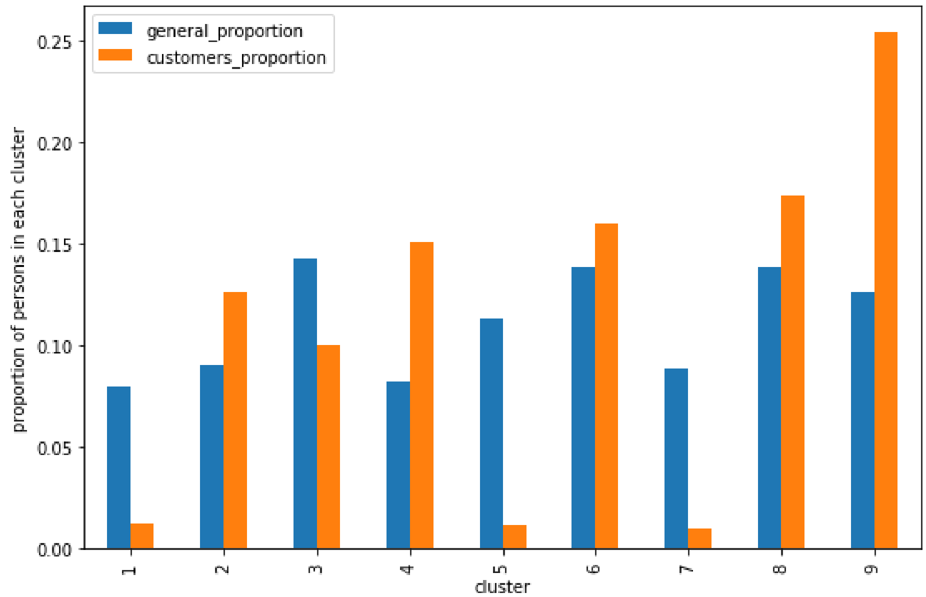

Figure 12-bar-plot with the distribution of general population and customers across clusters

Using 9 clusters for my analysis I find that cluster 4 and cluster 9 seem to have the features that coin what a core customer is and cluster 1, 5 and 7 round up what excludes who the core customers are.

I don't consider the differences on the other clusters to be striking enough to draw conclusions, overall, they seem neutral.

Supervised Learning Model

We are reaching the closing portion, and for me my favorite part (because I can compare my results to others and have an idea of how well my work performed).

Comparing the positive vs negative responses truly shows how imbalanced the data is:

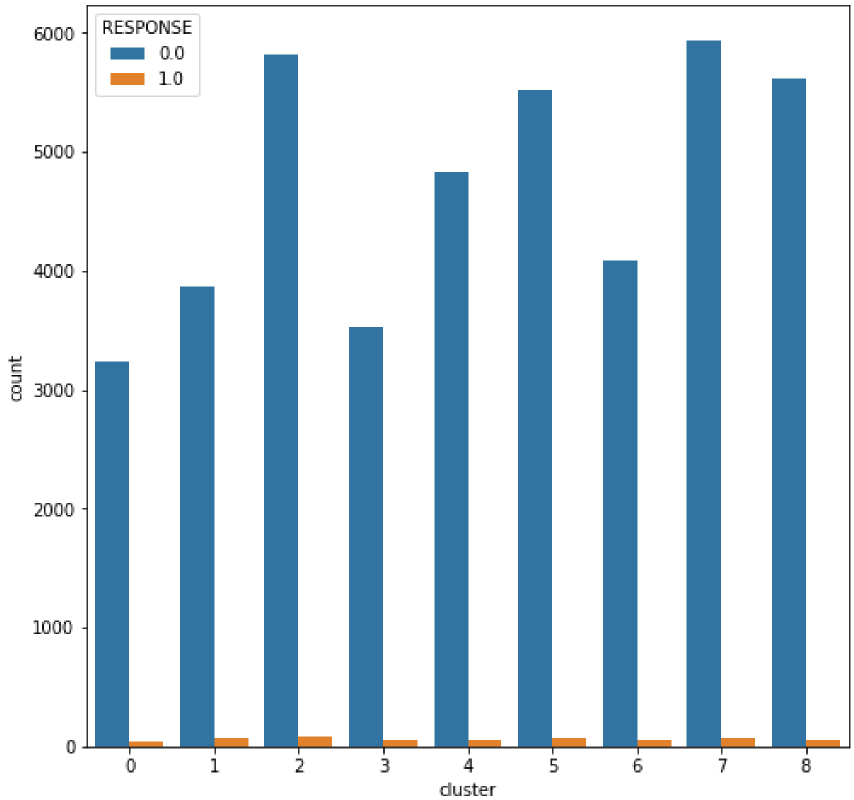

Figure 13-Bar-plot positive responses vs negative responses across clusters identified in the Unsupervised version of the project

I chose not to create a master function to do all the cleanup because I suspected that there were cleaning steps that could be optimized and changed for better results (which turned out to be true, at some point I mentioned that I had cleaning step that removed rows missing more than a certain threshold of data which ended up being pretty bad for the model), so maybe a few more lines of code but still very modular and very clear.

There were quite a few classifiers I wanted to try for this portion of the project:

- LogisticRegression
- RandomForestClassifier
- XGBClassifier
- LGBMClassifier
- GradientBoostingClassifier
- MLPClassifier

In principle there was a possibility of any of these to work for this problem, but I decided to let the data speak its truth and pit them against each other.

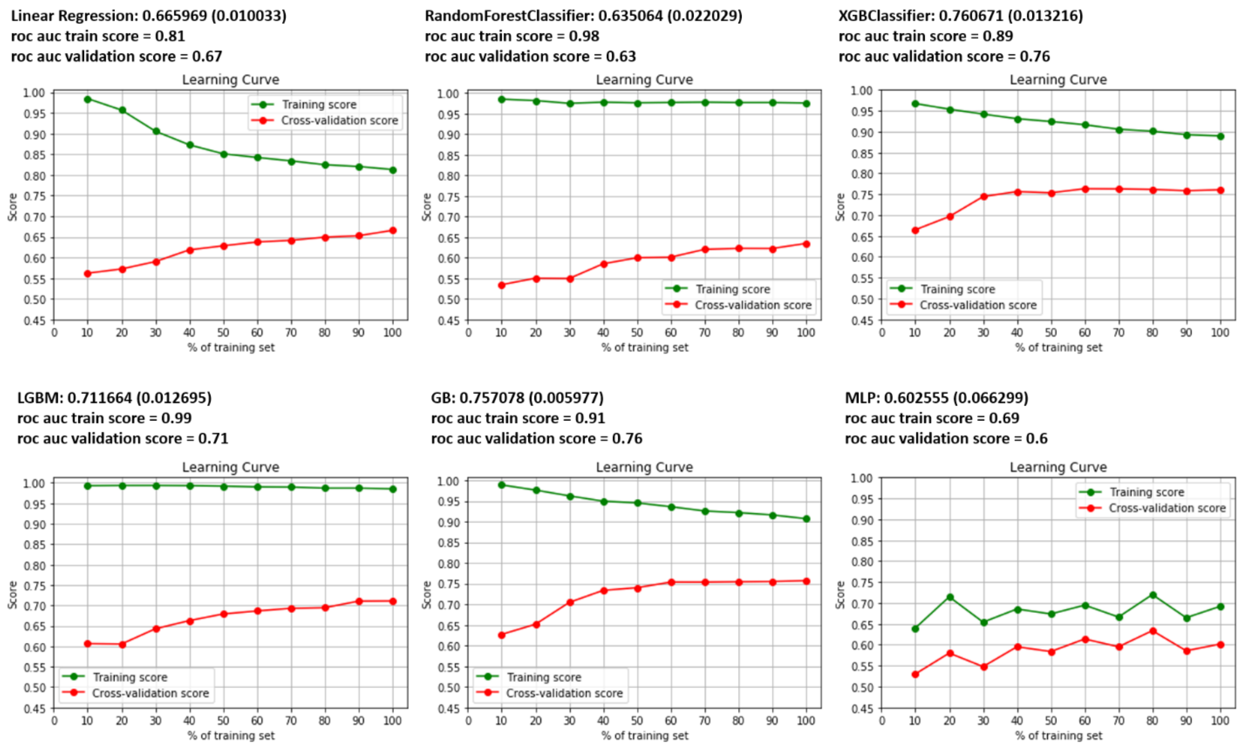

Figure 14-Base model performance comparison

Based on the model comparisons it became clear that there were 3 winners. Gradient Boost, XGB and LGBM, so out of these 3 I selected LGBM and XGB for further optimization and testing.

Once I had decided on the models I would be using I tested which data scaling method would be a good candidate to apply to my data.

Out of all the available automated tuning approaches I chose a Bayesian optimization approach since despite being one of the most time consuming, it is reported to be the best performing one (Koehrsen, 2018).

The process of hyperparameter tuning is entirely available on my MODEL TESTING file [linked here].

Results

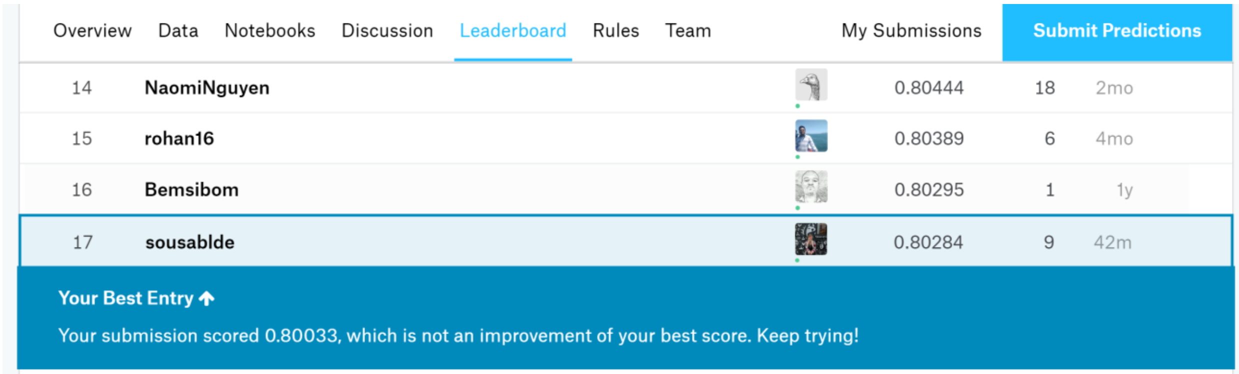

Figure 15-Kaggle score

My model landed me on the top third of the Kaggle leaderboard, which is really exciting, it means that there is room for improvement (patience with running xbgoost for more iterations, using techniques to deal with the class imbalance like SMOTe, play a little bit more with my data cleaning techniques).

I also identified D19_SOZIALES as the feature with the greatest impact in the model. Since this is one of the features with no descriptors in the metadata I will assume that it is related to social classifiers and descriptors of the individuals in the dataset.

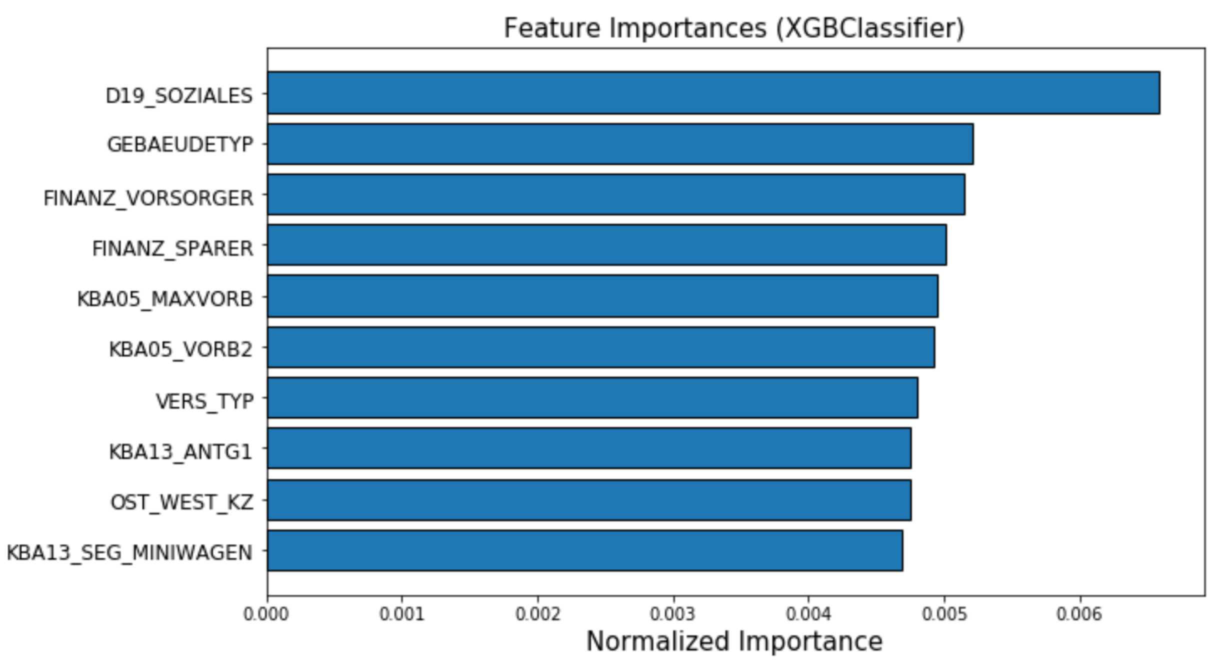

Figure 16-Features with greater impact on model performance

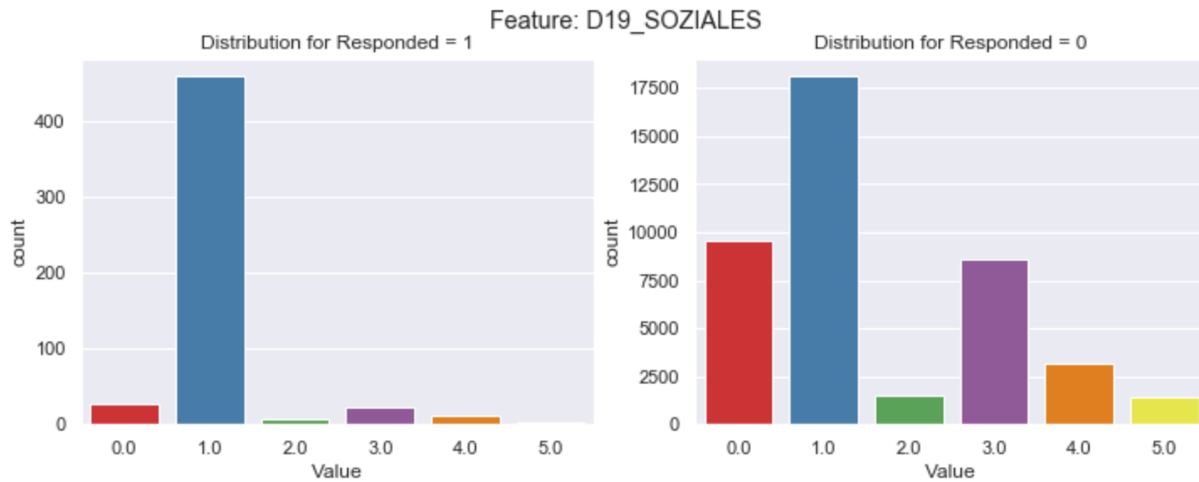

Figure 17-Response distribution on the feature identified as the most impactful in the model

But above all if you refer to my benchmark on figure 13, the ROC-AUC score originally obtained with LR was around 0.66 but with the optimized xgboost it went up to over 0.8.

At the moment the greatest impediment to further testing and optimization is the shear time it takes to do the optimization (a minimum of 36 hours at a time) which is in itself a testament of how valuable cloud based training of machine learning models is (if I still had amazon credits I would definitely attempt to run these models on sagemaker).

Improvements and continued work

This project was incredible challenging and gratifying, I am excited to see if I have room to improvement with my approach. In the future I plan to manipulate the following segments of my approach to this problem:

- Changing the missing value threshold of columns to drop
- Revisit the row_dropper and test if changing the parameters can improve outcomes
- Remove redundant features based on grob and fein descriptors
- Apply sampling techniques to balance classes
- Run BayesSearchCV with different scaling models

Thank You

Works Cited

Brems, M. (2017, April 17). A One-Stop Shop for Principal Component Analysis. *Towards Data Science*.

Computerwoche. (1983, October 21). Bertelsmann vertreibt Rechner von TI. *Computerwoche*.

Garbade, M. J. (2018, September 12). Understanding K-means Clustering in Machine Learning. *Towards Data Science*.

Koehrsen, W. (2018, July 3). Automated Machine Learning Hyperparameter Tuning in Python. *Towards Data Science*.

Name, N. (1999, 9 June). Darmstädter Echo. *neue Ziele*.

Paperlein, J. (2012, April 5). E-Commerce ist weltweit ein Thema. *Horizont*, p. 14.

Raj, J. T. (2019, March 11). A beginner's guide to dimensionality reduction in Machine Learning. *Towards Data Science*.

Schuler, T. (2010, June 18). Erst Drucker, dann Verleger. *Berliner Zeitung*, p. 30.