Capstone Project
Data Science Nanodegree

Customer Segmentation Report for Arvato Financial Services

Abhishek Fulzele

August 22, 2018

Project Overview

“You can have data without information, but you cannot have information without data.”
— Daniel Keys Moran

Machine learning has recently become more popular in the finance sector because of the availability of large quantities of data and more accessible computing resources. In banking, machine learning reshapes the financial services industry as never before. Leading banks and financial services companies are deploying AI technology, to optimize portfolios, decrease risk and underwrite loans; checking anomaly to prevent frauds and anti-money laundering amongst other things.

This project is one of the capstone projects provided by Udacity as a part of Data Science Nanodegree. The end goal of the project is to determine how the company would acquire new customers.

In this project, we are using supervised and unsupervised learning techniques to analyze customer demographics in Germany and predict potential customers which will increase the efficiency of acquiring the customers. Without a data-driven approach the company would waste a lot of time and this would be a cost ineffective process.

There are two sections in the project:

1. Unsupervised Learning which is used to identify the how the existing customers matched the segments of German population
2. Supervised learning to predict the likelihood of acquiring new customers.

All the supporting analysis and documentation is available at Github except for the datasets.

Problem Statement

Arvato is employing customer segmentation to work on the assumption that each customer is different and that their marketing campaigns will be best suited if they approach unique, smaller audiences with advertisements that are important to some customers and drive them to buy more.

One strategy would be to implement Dimensionality reduction techniques like PCA and then use K-Means clustering on the population and the customers dataset to find comparison between them. We can then use supervised learning on the clusters which are more similar to predict which customers are more likely to be converted.

Datasets

The datasets provided by the company includes the following:

a. *Udacity AZDIAS 052018.csv*: Demographics data for the general population of Germany; **891,211 persons (rows) x 366 features (columns)**. A sample of data is shown below in *figure 1.1*.

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTI
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0	
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	

Figure 1.1

b. *Udacity CUSTOMERS 052018.csv*: Demographics data for the general population of Germany; **191,652 persons (rows) x 369 features (columns)**. A sample of data is shown below in *figure 1.2*.

```
customers.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTI
0	9626	2	1.0	10.0	NaN	NaN	NaN	NaN	10.0	
1	9628	-1	9.0	11.0	NaN	NaN	NaN	NaN	NaN	
2	143872	-1	1.0	6.0	NaN	NaN	NaN	NaN	0.0	
3	143873	1	1.0	8.0	NaN	NaN	NaN	NaN	8.0	
4	143874	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	

Figure 1.2

c. *Udacity MAILOUT 052018 TRAIN.csv*: Demographics data for individuals who were targets of a marketing campaign; 42,982 persons (rows) x 367 (columns). The extra column in the dataset is the response of each customer. A sample of data is shown below in *figure 1.3*.

LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AK
1763	2	1.0	8.0	NaN	NaN	NaN	NaN	8.0	1
1771	1	4.0	13.0	NaN	NaN	NaN	NaN	13.0	
1776	1	1.0	9.0	NaN	NaN	NaN	NaN	7.0	
1460	2	1.0	6.0	NaN	NaN	NaN	NaN	6.0	
1783	2	1.0	9.0	NaN	NaN	NaN	NaN	9.0	5

Figure 1.3

d. *Udacity MAILOUT 052018 TEST.csv*: Demographics data for individuals who were targets of a marketing campaign; 42,833 persons (rows) x 366 (columns). A sample of data is shown below in *figure 1.4*.

LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AK
1754	2	1.0	7.0	NaN	NaN	NaN	NaN	6.0	
1770	-1	1.0	0.0	NaN	NaN	NaN	NaN	0.0	2
1465	2	9.0	16.0	NaN	NaN	NaN	NaN	11.0	
1470	-1	7.0	0.0	NaN	NaN	NaN	NaN	0.0	
1478	1	1.0	21.0	NaN	NaN	NaN	NaN	13.0	

Figure 1.4

e. *Diaz Attributes – Value 2017.xlsx:* Information of each features and its values. A sample of data is shown below in *figure 1.5*.

	Attribute	Description	Value	Meaning
0	AGER_TYP	best-ager typology	-1	unknown
1	AGER_TYP	best-ager typology	0	no classification possible
2	AGER_TYP	best-ager typology	1	passive elderly
3	AGER_TYP	best-ager typology	2	cultural elderly
4	AGER_TYP	best-ager typology	3	experience-driven elderly

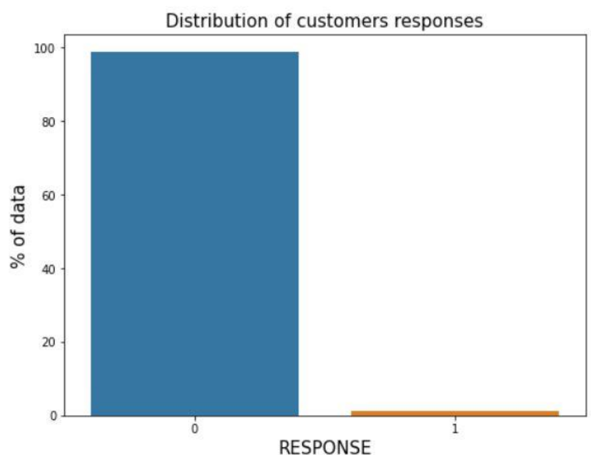
Figure 1.5

Metrics

Imbalanced classes are a fairly common problem in machine learning (specifically in classification), which arises in datasets with a large observer ratio in-class. Most machine learning algorithms work best when the number of samples in each class are about equal. This is because most algorithms are designed to maximize accuracy and reduce error.

In Machine Learning, performance measurement is an essential task. So, when it comes to a classification problem, we can count on an AUC - ROC Curve. When we need to check or visualize the performance of the multi - class classification problem, we use AUC (Area Under the Curve) ROC (Receiver Operating Characteristics) curve. It is one of the most important evaluation metrics for checking any classification model's performance. ROC is a probability curve and AUC represent degree or measure of separability. It tells how much model is capable of distinguishing between classes. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s

The response feature in the dataset is also highly imbalanced.



To solve the problem, we'll chose ROC-AUC metric to evaluate the performance of our models. This is also the required evaluation metric for the Kaggle submission

Data Preprocessing and Exploration

When loading the dataset, we get a panda warning of data conversion which looks like this.

```
C:\Users\bune1\anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3063: DtypeWarning: Columns (19,20) have mixed t
ypes.Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

To deal with this issue we explore the columns which are: ***'CAMEO_DEUG_2015', 'CAMEO_INTL_2015'.*** Further finding the unique values inside this column we get that the columns have Xs, XXs, string values and numeric values. I decided to convert the Xs and XXs into np.nan and then convert the column into numeric format.

After solving this issue, we now read the Attributes file and define a function to find the unknowns in the dataset. We'll insert this values in a dictionary so that we can later convert the unknown values into null values.

	Attribute	Description	Value	Meaning
0	AGER_TYP	best-ager typology	-1	unknown
1	AGER_TYP	best-ager typology	0	no classification possible
2	AGER_TYP	best-ager typology	1	passive elderly
3	AGER_TYP	best-ager typology	2	cultural elderly
4	AGER_TYP	best-ager typology	3	experience-driven elderly

```
description_unkwn
```

```
{'AGER_TYP': [-1],
 'ALTERSKATEGORIE_GROB': [9, -1, 0],
 'ALTER_HH': [0],
 'ANREDE_KZ': [-1, 0],
 'BALLRAUM': [-1],
 'BIP_FLAG': [-1],
 'CAMEO_DEUG_2015': [-1],
 'CAMEO_DEUINTL_2015': [-1],
 'CJT_GESAMTTYP': [0],
 'D19_KK_KUNDENTYP': [-1],
 'EWDICHTE': [-1],
```

Now, before converting the unknown values into Nulls, we need to correct the misspelled features. One such feature which is the datasets but not in the Attributes file is ***'CAMEO_INTL_2015'*** .This is spelled in the Attributes file as ***'CAMEO_DEUINTL_2015'.***

```
cameo_az = azdias.columns[azdias.columns.str.contains("CAMEO")].tolist()
cameo_az.sort()
cameo_az
```

```
['CAMEO_DEUG_2015', 'CAMEO_DEU_2015', 'CAMEO_INTL_2015']
```

```
cameo_desc = df_desc[df_desc['Attribute'].str.match('CAMEO')]['Attribute'].unique()
cameo_desc.sort()
cameo_desc
```

```
array(['CAMEO_DEUG_2015', 'CAMEO_DEUINTL_2015', 'CAMEO_DEU_2015'],
      dtype=object)
```

Missing Values

After converting the unknown values into Nulls, we'll investigate the number of null values in each column. I created a function which shows the percentage of null values ranging from 0 to 100 percent in each column as well as rows.

```
percentage_nulls(percentage_bins, 1, azdias, 'columns')
```

```
280 is the number of columns which has NaN entries less than or equal to 0% of total of 366 entries
245 is the number of columns which has NaN entries less than or equal to 10% of total of 366 entries
20 is the number of columns which has NaN entries less than or equal to 20% of total of 366 entries
10 is the number of columns which has NaN entries less than or equal to 30% of total of 366 entries
9 is the number of columns which has NaN entries less than or equal to 40% of total of 366 entries
9 is the number of columns which has NaN entries less than or equal to 50% of total of 366 entries
8 is the number of columns which has NaN entries less than or equal to 60% of total of 366 entries
7 is the number of columns which has NaN entries less than or equal to 70% of total of 366 entries
5 is the number of columns which has NaN entries less than or equal to 80% of total of 366 entries
5 is the number of columns which has NaN entries less than or equal to 90% of total of 366 entries
0 is the number of columns which has NaN entries less than or equal to 100% of total of 366 entries
```

```
percentage_nulls(percentage_bins, 0, azdias, 'rows')
```

```
891221 is the number of rows which has NaN entries less than or equal to 0% of total of 891221 entries
154646 is the number of rows which has NaN entries less than or equal to 10% of total of 891221 entries
132105 is the number of rows which has NaN entries less than or equal to 20% of total of 891221 entries
105811 is the number of rows which has NaN entries less than or equal to 30% of total of 891221 entries
100328 is the number of rows which has NaN entries less than or equal to 40% of total of 891221 entries
99968 is the number of rows which has NaN entries less than or equal to 50% of total of 891221 entries
93177 is the number of rows which has NaN entries less than or equal to 60% of total of 891221 entries
73517 is the number of rows which has NaN entries less than or equal to 70% of total of 891221 entries
0 is the number of rows which has NaN entries less than or equal to 80% of total of 891221 entries
0 is the number of rows which has NaN entries less than or equal to 90% of total of 891221 entries
0 is the number of rows which has NaN entries less than or equal to 100% of total of 891221 entries
```

The above image shows the percentage of null values in columns and rows. We can also represent the null values in a visualization to understand better.

```
plot_null_chart(azdias, 30)
```

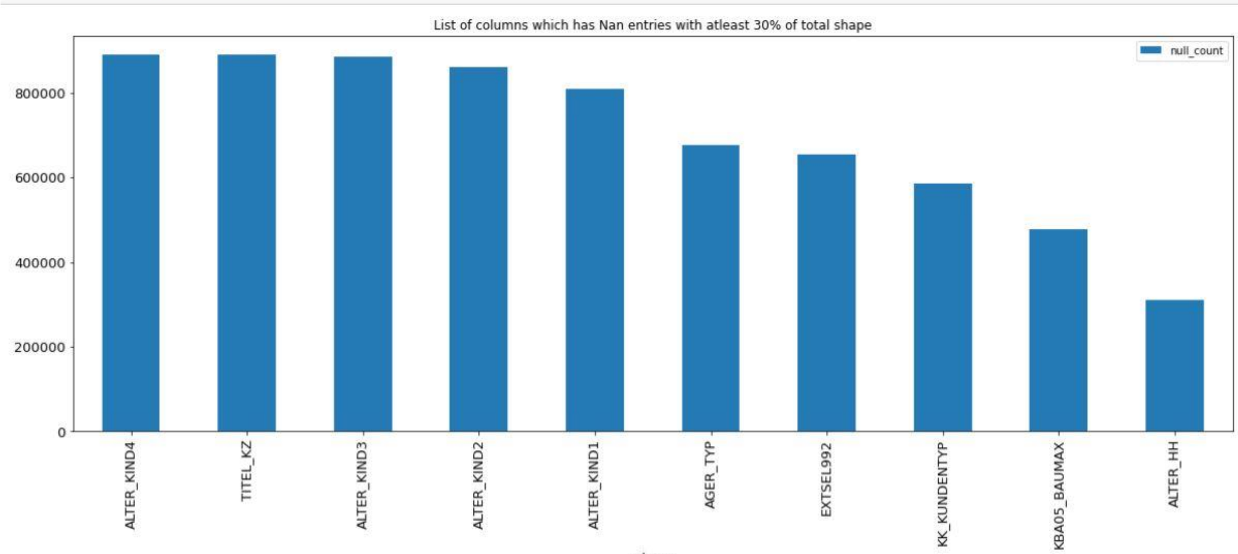


Image showing the list of columns which has null values more than or equal to 30%

```python
# second arguement is to get the number of elements we want in the x-axis
plot_complete_null_chart(azdias, az_null_col_df, 40)
```

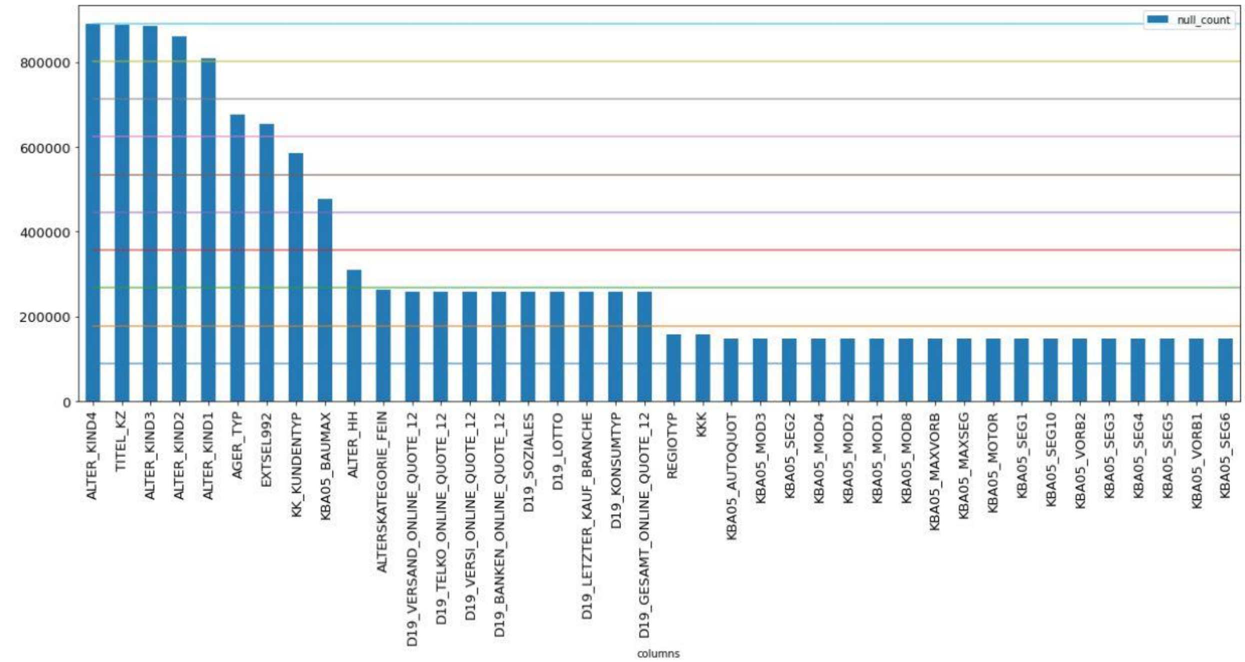


Image showing the null values sorted up to 40 features.

I decided to drop columns which has 30% or more null entries in it. This brings about inconsistency in the azdias and customers datasets. The azdias dataset remains with two more features than customers so I decided to drop them as well.

Feature Engineering

As the last step of preprocessing, I decided to do some feature engineering for the following columns:

- Converting 'CAMEO_DEU_2015' using label encoder from categorical into numerical.
- Removing 'D19_LETZTER_KAUF_BRANCHE' column which has values representing different columns.
- 'EINGEFUEGT_AM' is a column that has a lot of dates. So, I extracted years from the column, saved it and dropped the original column.
- Converting 'OST_WEST_KZ' which has binary values in the format 'W', 'O' into numeric.
- Binned and labelled features like 'ANZ_HAUSHALTE_AKTIV', 'ANZ_STATISTISCHE_HAUSHALTE', 'GEBURTSJAHR', 'KBA13_ANZAHL_PKW', 'MIN_GEBAEUDEJAHR', 'VERDICHTUNGSRAUM',
- Extracted information from PRAEGENDE_JUGENDJAHRE and created two more columns out of it.

After re-encoding the features, I used SimpleImputer() to impute the nans with the most-frequent values.

Dimensionality Reduction

Reducing the dimension of the feature space is called “dimensionality reduction.” There are many ways to achieve dimensionality reduction, but most of these techniques fall into one of two classes:

- Feature Elimination
- Feature Extraction

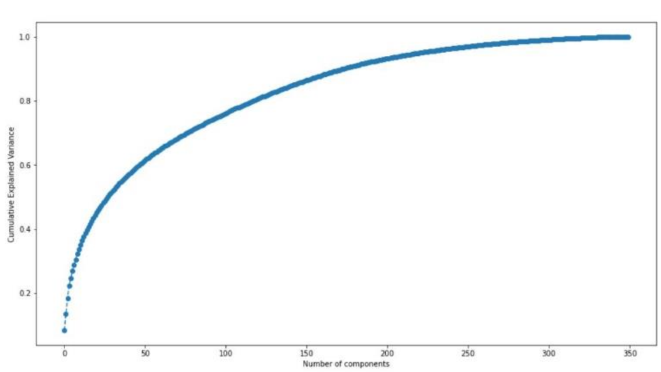
To perform Dimensionality Reduction, we will use PCA. *Principal component analysis* is a technique for feature extraction — so it combines our input variables in a specific way, then we can drop the “least important” variables while still retaining the most valuable parts of all the variables.

The goal of PCA is to represent your data X in an orthonormal basis W.

Before performing PCA, we need to scale the inputs. I used different scaling techniques like StandardScaler, RobustScaler and MinMaxScaler to check how the PCA differs. Here are the following results.

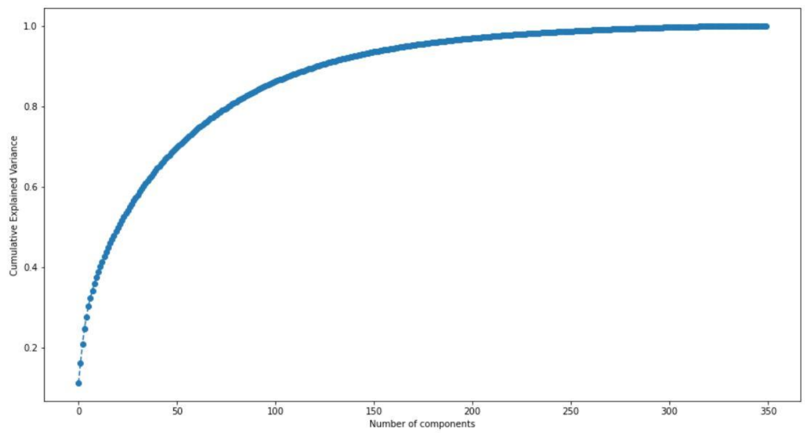
For StandardScaler:

```
10 components explain 0.3373941632212266 of variance.
20 components explain 0.4432029174344359 of variance.
30 components explain 0.513652932915443 of variance.
40 components explain 0.5666973793375343 of variance.
50 components explain 0.6103411603544119 of variance.
60 components explain 0.6474798937767071 of variance.
70 components explain 0.6793675755505096 of variance.
80 components explain 0.7082590959658471 of variance.
90 components explain 0.7345766503043226 of variance.
100 components explain 0.759271468569543 of variance.
110 components explain 0.7825292930413058 of variance.
120 components explain 0.8043990468289444 of variance.
130 components explain 0.8249445689601029 of variance.
140 components explain 0.844287995561l573 of variance.
150 components explain 0.8623961033464526 of variance.
160 components explain 0.8791557800755605 of variance.
170 components explain 0.8946647079748979 of variance.
180 components explain 0.9085715312736946 of variance.
190 components explain 0.9205757281851734 of variance.
200 components explain 0.9311499563708999 of variance.
210 components explain 0.9403853085015272 of variance.
220 components explain 0.9487711087254719 of variance.
230 components explain 0.9562858223771682 of variance.
240 components explain 0.9630462680692562 of variance.
250 components explain 0.9690719035532707 of variance.
260 components explain 0.9744919057464597 of variance.
270 components explain 0.9793061840008235 of variance.
280 components explain 0.9834911736227497 of variance.
290 components explain 0.9871552866536608 of variance.
300 components explain 0.9903969246502379 of variance.
310 components explain 0.9932749931917466 of variance.
320 components explain 0.9957767260964387 of variance.
330 components explain 0.9979253838031511 of variance.
340 components explain 0.9994342533101757 of variance.
```



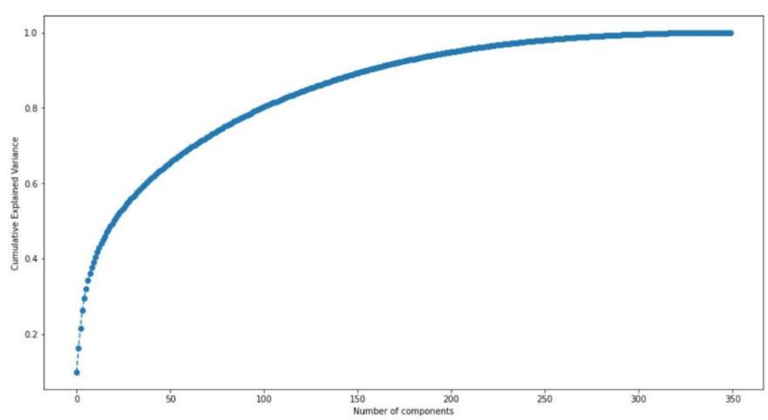
For RobustScaler:

```
10 components explain 0.3741395866218696 of variance.
20 components explain 0.4885672478286454 of variance.
30 components explain 0.5725343876192036 of variance.
40 components explain 0.639713522593659 of variance.
50 components explain 0.6937331308366043 of variance.
60 components explain 0.7386585039767132 of variance.
70 components explain 0.7760827513587526 of variance.
80 components explain 0.8082499961336063 of variance.
90 components explain 0.8359514101062232 of variance.
100 components explain 0.8594076213096937 of variance.
110 components explain 0.8792633078491379 of variance.
120 components explain 0.896226038615684 of variance.
130 components explain 0.9113663651782393 of variance.
140 components explain 0.9240067557639758 of variance.
150 components explain 0.9346047508833919 of variance.
160 components explain 0.9434211576717761 of variance.
170 components explain 0.9511457524204184 of variance.
180 components explain 0.9578325343493009 of variance.
190 components explain 0.9636980133155211 of variance.
200 components explain 0.968836447933545 of variance.
210 components explain 0.9733588856542664 of variance.
220 components explain 0.9772592124759564 of variance.
230 components explain 0.980683034660537 of variance.
240 components explain 0.9836808802825409 of variance.
250 components explain 0.9863842640700258 of variance.
260 components explain 0.9888420864189007 of variance.
270 components explain 0.9910523150917617 of variance.
280 components explain 0.9930399647488987 of variance.
290 components explain 0.9947942869809141 of variance.
300 components explain 0.9962891604089116 of variance.
310 components explain 0.9975626626244405 of variance.
320 components explain 0.9986048632428182 of variance.
330 components explain 0.9993460540520406 of variance.
340 components explain 0.9998417264309375 of variance.
```



For MinMaxScaler:

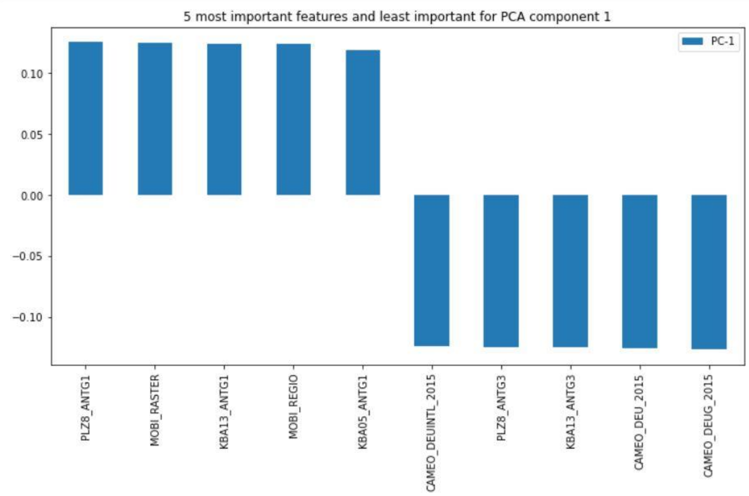
```
10 components explain 0.39210129439634683 of variance.
20 components explain 0.49432117926261815 of variance.
30 components explain 0.5587574335439899 of variance.
40 components explain 0.6095206550515282 of variance.
50 components explain 0.6518364822137939 of variance.
60 components explain 0.6886933054032457 of variance.
70 components explain 0.720934929904388 of variance.
80 components explain 0.7498930695329721 of variance.
90 components explain 0.776401630605671 of variance.
100 components explain 0.8004015775869696 of variance.
110 components explain 0.8221206344554854 of variance.
120 components explain 0.8417407245942324 of variance.
130 components explain 0.8595267011740569 of variance.
140 components explain 0.8760863876449425 of variance.
150 components explain 0.891368179263349 of variance.
160 components explain 0.9052371393159756 of variance.
170 components explain 0.9176853341211862 of variance.
180 components explain 0.9286577675585198 of variance.
190 components explain 0.938568839337587 of variance.
200 components explain 0.9475832528203623 of variance.
210 components explain 0.9556976209656863 of variance.
220 components explain 0.963006174900031 of variance.
230 components explain 0.9694604859264713 of variance.
240 components explain 0.9749548229884561 of variance.
250 components explain 0.9797998560749173 of variance.
260 components explain 0.9839642602302099 of variance.
270 components explain 0.9875710857641024 of variance.
280 components explain 0.9906447108659828 of variance.
290 components explain 0.9931876077538586 of variance.
300 components explain 0.9952918051765307 of variance.
310 components explain 0.9971075911485824 of variance.
320 components explain 0.9985340455942909 of variance.
330 components explain 0.9994480288916022 of variance.
340 components explain 0.9998969784989984 of variance.
```



After performing the PCA, I decided to chose StandardScaler to implement the PCA with 150 components on the cleaned dataset.

```
RobustScaler -> 85% with 100 components
StandardScaler -> 85% with 150 components
MinMaxScaler -> 85% with 130 components
```

Here is the 5 most positive and negative result of the information retained after performing PCA on StandardScaled dataset in the first principal component.



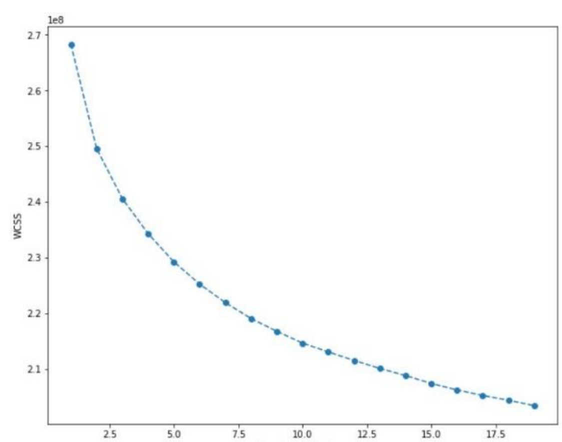
Clustering technique

A cluster refers to a collection of data points aggregated together because of certain similarities. We'll define a target number k, which refers to the number of centroids which we need in the dataset. A centroid is the imaginary or real location representing the center of the cluster.

K-means algorithm identifies k number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible. To get the number of optimum clusters we need to use the Elbow method. Elbow method gives us an idea on what a good k number of clusters would be based on the sum of squared distance (SSE) between data points and their assigned clusters' centroids.

In this project, we perform K-means clustering to find the relationship between the population and customers.

We'll perform K-Means with clusters ranging from 1 to 20 to determine the optimum clusters required using elbow method.



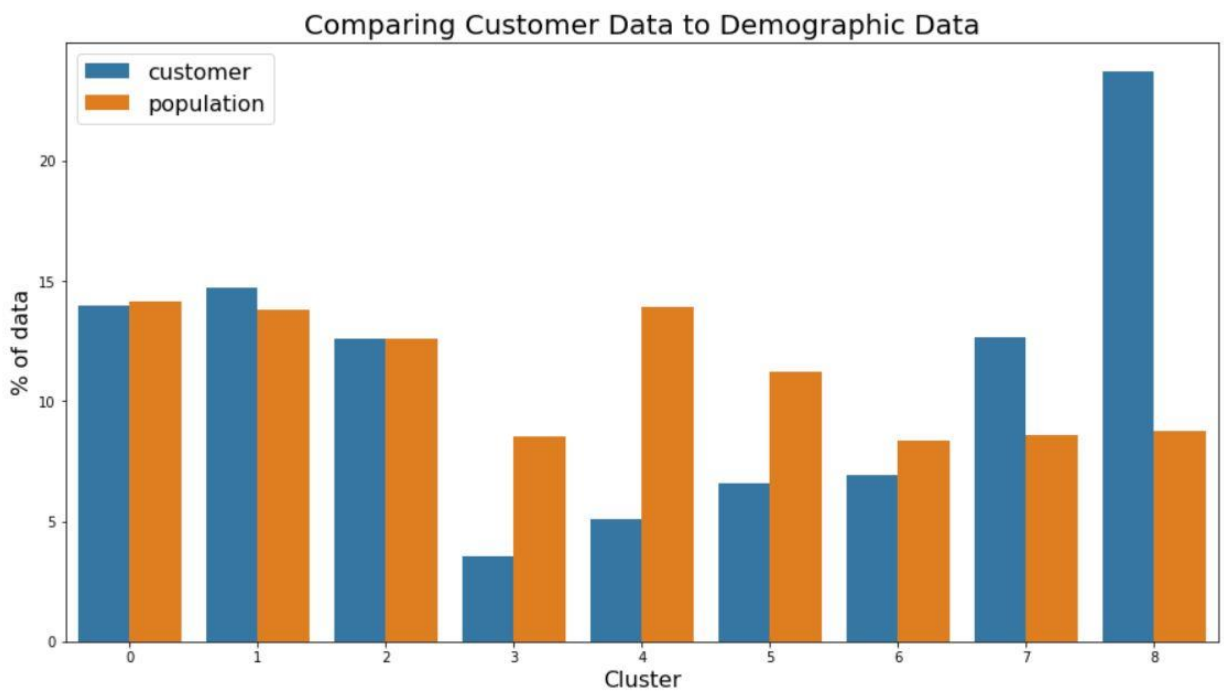
Based on the elbow method we can see that 9 is the optimum number for performing clustering.

After fitting 9-means clustering on the datasets, we get the following results.

	cluster	% of data	data
0	0	14.173813	population
1	4	13.927746	population
2	1	13.787265	population
3	2	12.622010	population
4	5	11.246705	population
5	8	8.763146	population
6	7	8.583954	population
7	3	8.556239	population
8	6	8.339121	population

	cluster	% of data	data
0	8	23.753470	customer
1	1	14.731910	customer
2	0	13.991505	customer
3	7	12.676100	customer
4	2	12.630706	customer
5	6	6.926617	customer
6	5	6.612506	customer
7	4	5.106130	customer
8	3	3.571056	customer

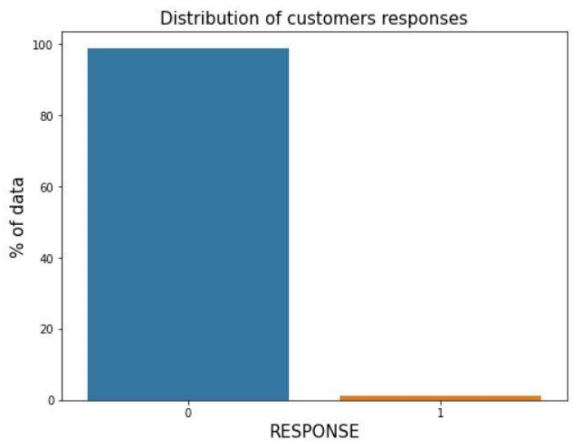
Comparing the clusters.



We can clearly see that clusters 0, 1, 3 are close to representing the approximate relationships. Clusters 7, 8 strongly represents the comparison.

Supervised Learning

We see that the high imbalance in the data is because of not proper proportion of customers responses.



A lot of customers responded negatively compared to the positive response. As you see, we have almost more than 95% of negative response and less than 5% of positive.

```
0    98.761696
1     1.238304
Name: RESPONSE, dtype: float64
```

The reason why we chose ROC-AUC for evaluating models is because the model would either give 98% accuracy whenever we perform supervised learning techniques. The ROC_AUC curve defines the model's capability to separate binary classes.

There are other techniques to deal with highly imbalance. I have noted down a few:

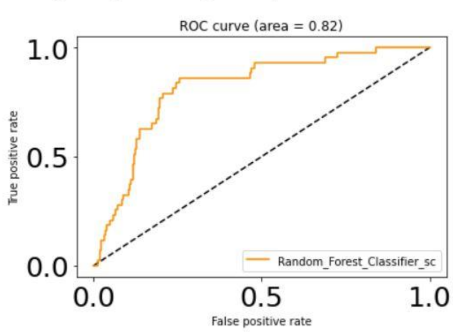
- Up sampling/Oversampling
- Under sampling/ Down Sampling
- Using SMOTE
- Using Ensemble techniques

As a baseline model, I have chosen Random Forest Classifier. After cleaning and scaling the MAILOUT_TRAIN and MAILOUT_TEST datasets, I performed Random Forest Classification.

Random_Forest_Classifier_sc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.815109

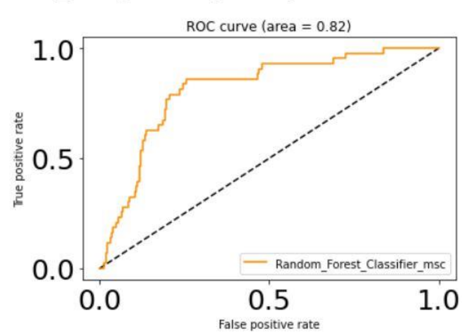
Random_Forest_Classifier_sc ROC plot



Random_Forest_Classifier_msc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.815415

Random_Forest_Classifier_msc ROC plot



Above is the ROC-AUC curve of the result where sc stands for StandardScaler and msc stands for MinMaxStandardScaler.

Our benchmark ROC_AUC value is now **0.815.**

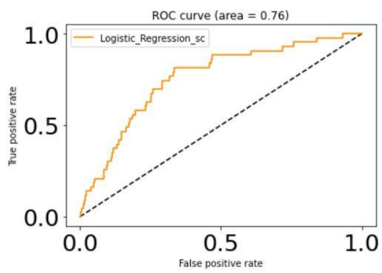
There were quite a few other classifiers which I wanted to try, with the ROC-AUC performance metrics:

- Logistic Regression

Logistic_Regression_sc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.757335

Logistic_Regression_sc ROC plot



Logistic_Regression_msc performance

Metric	Test
accuracy	0.987293
AUC	0.76298

- Gradient Boosting Classifier

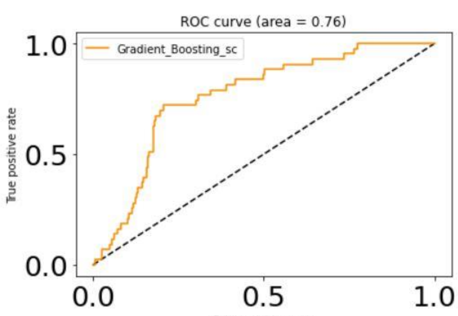
Gradient_Boosting_sc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.763342

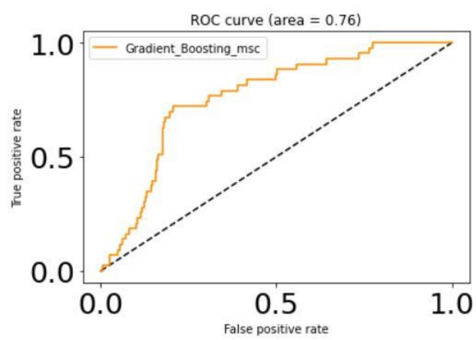
Gradient_Boosting_msc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.763342

Gradient_Boosting_sc ROC plot



Gradient_Boosting_msc ROC plot

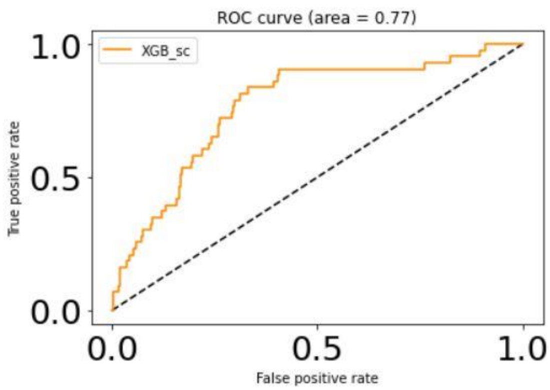


- XGBoosting Classifier

XGB_sc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.771082

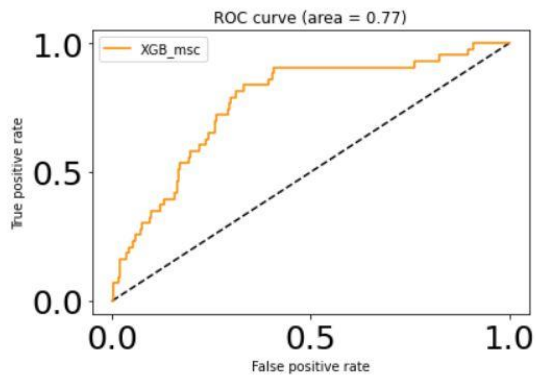
XGB_sc ROC plot



XGB_msc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.771082

XGB_msc ROC plot

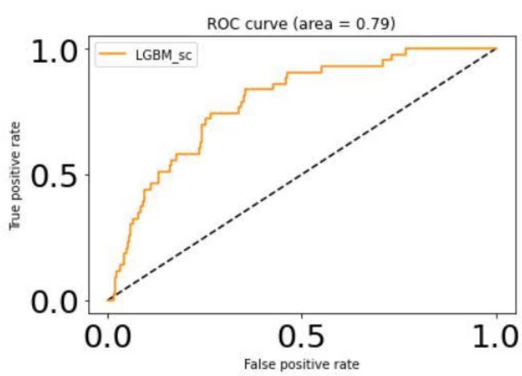


- LGBM Classifier

LGBM_sc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.788728

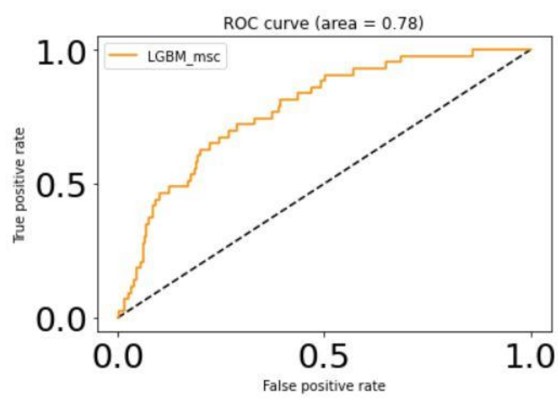
LGBM_sc ROC plot



LGBM_msc performance on the test data set:

Metric	Test
accuracy	0.987293
AUC	0.778315

LGBM_msc ROC plot



The Random Forest Classifier performs better than other models. I submitted results on the Kaggle and the results I got is 0.74.

164	Abhishek Fulzele	0.74151	16	15h

Improvement

There are several improvements that can be done to get better scores. Some are as follows:

- Use GridSearchCV
- Implement SMOTE with Cross Validation
- Random Sampling, Up Sampling and Down Sampling