Bertelsmann / Arvato Project

Osarugue Laura Egharevba

1 Project Overview

Bertelsmann Arvato is a global services company headquartered in Gütersloh, Germany. The dataset for this project was provided by Arvato and this project is made up of two major parts:

1.1 Customer Segmentation:

The first of which is a Customer Segmentation problem where the end point is focused on how to acquire new Arvato clients more efficiently. To do this, we were provided with two datasets:

- 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).

1.2 Supervised Learning:

This part of the project focuses on predicting individuals most likely to become future customers of Arvato. The datasets for this project are:

- 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).

2 Project Statement

2.1 Customer Segmentation:

Using unsupervised learning techniques, perform customer segmentation and identify parts of the general population that best describes the core customer base for Arvato . This is achieved by creating clusters of Arvato’s existing customers super-imposed against clusters of the general population to identify both underrepresented and overrepresented clusters

2.2 Supervised Learning:

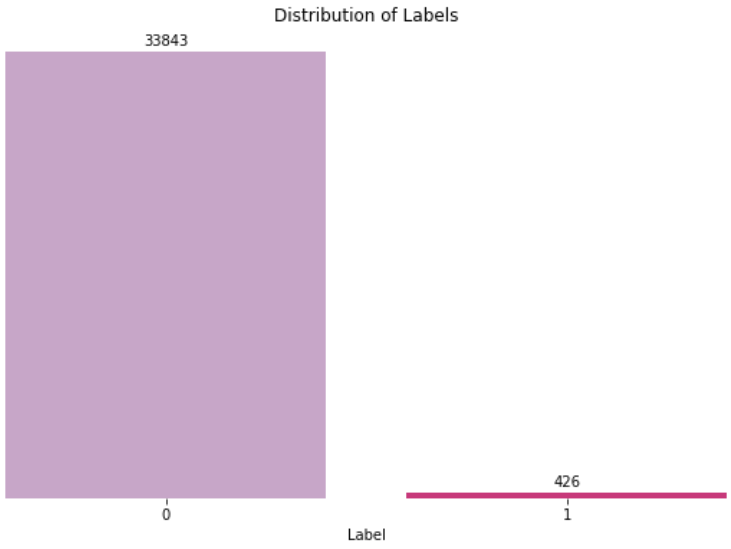
Using supervised learning techniques, predict future customers of Arvato. These are individuals that would respond positively (i.e., a “1” in the “RESPONSE” column) to the marketing campaign

3 Metrics

For my unsupervised learning part of this project, I used the elbow method to determine my optimal number of “k” to use for my clusters.

For the supervised learning part of this project, I decided to use “ROC_AUC” and this was because of the highly unbalanced nature of my dataset. I cared about predicting accurately for both the majority and minority class and “ROC_AUC” fulfilled that bit.

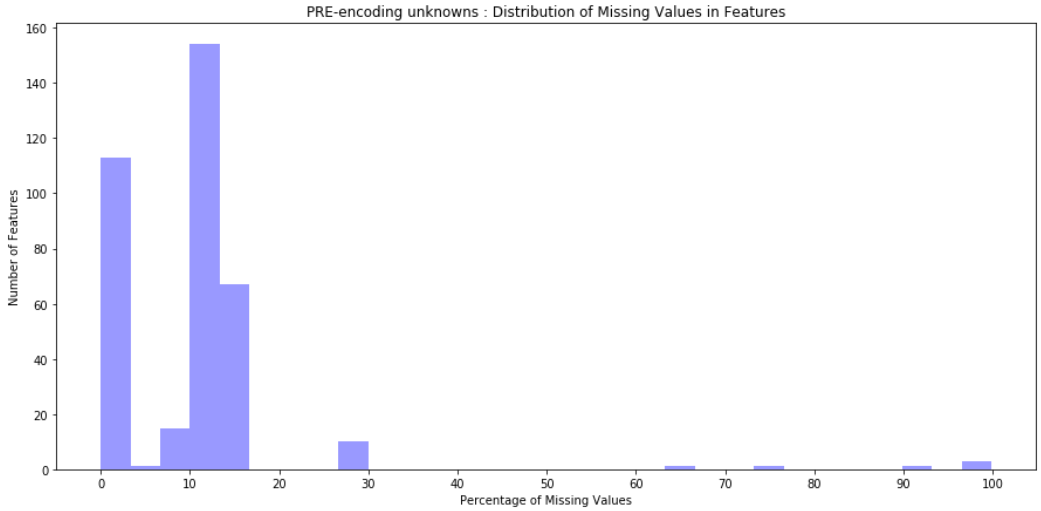
The highly unbalanced nature of my labels is represented below by the plot:



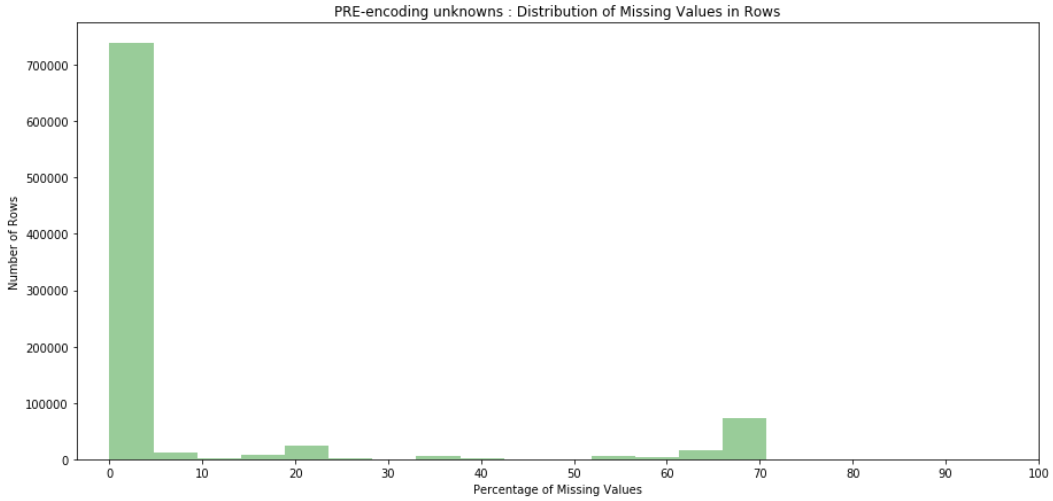
4 Data Exploration, Visualization and Preprocessing

My Udacity_AZDIAS_052018.csv dataset was made up of 891221 rows and 366 columns, 267 were floats, 93 were integers and 6 were objects. However, by going through the attached documentations, I noticed that the bulk of the unknowns were incorrectly encoded. While some were encoded as “-1”, others were encoded as “X”, “XX”, “0”, “9”, “10”.

Pre-encoding my unknowns properly, I decided to see the number of NaNs I already had. Of interest to me were those rows and columns that had NaNs exceeding 30% of its size.

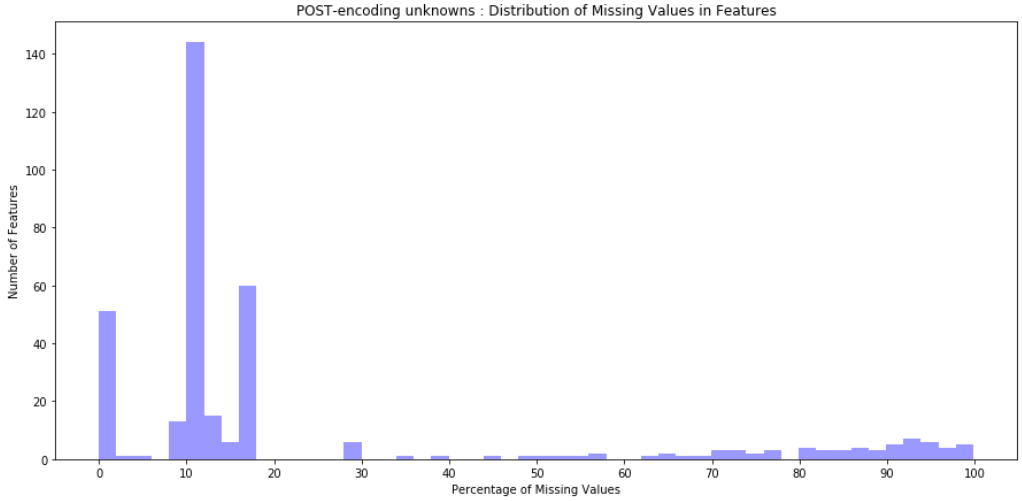


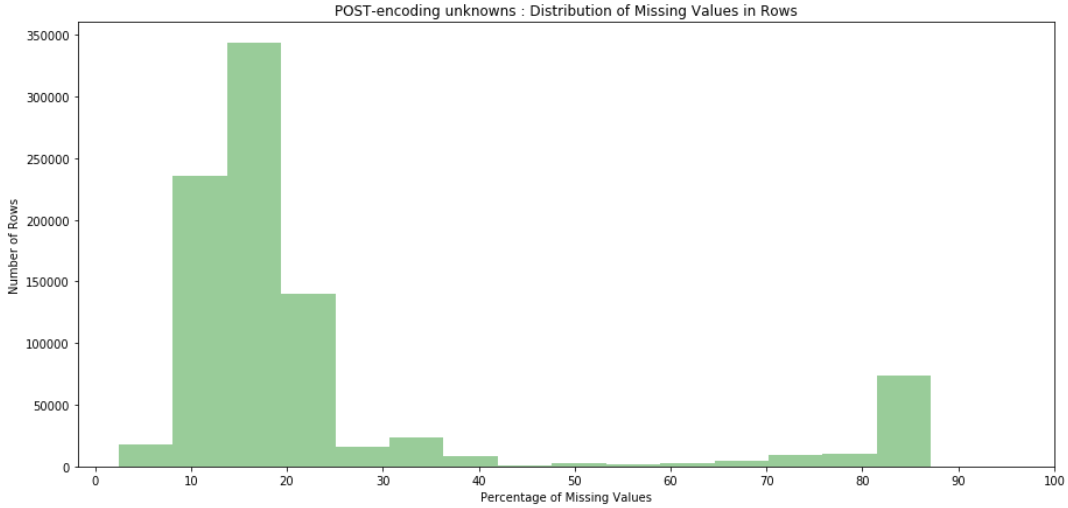
Above is the plot showing distribution of missing values in the columns (features), and just for reference purposes, these are the columns with NaNs exceeding 30% - “alter_kind1”, “alter_kind2”, “alter_kind3”, “alter_kind4”, “extsel992” and “kk_kundentyp”



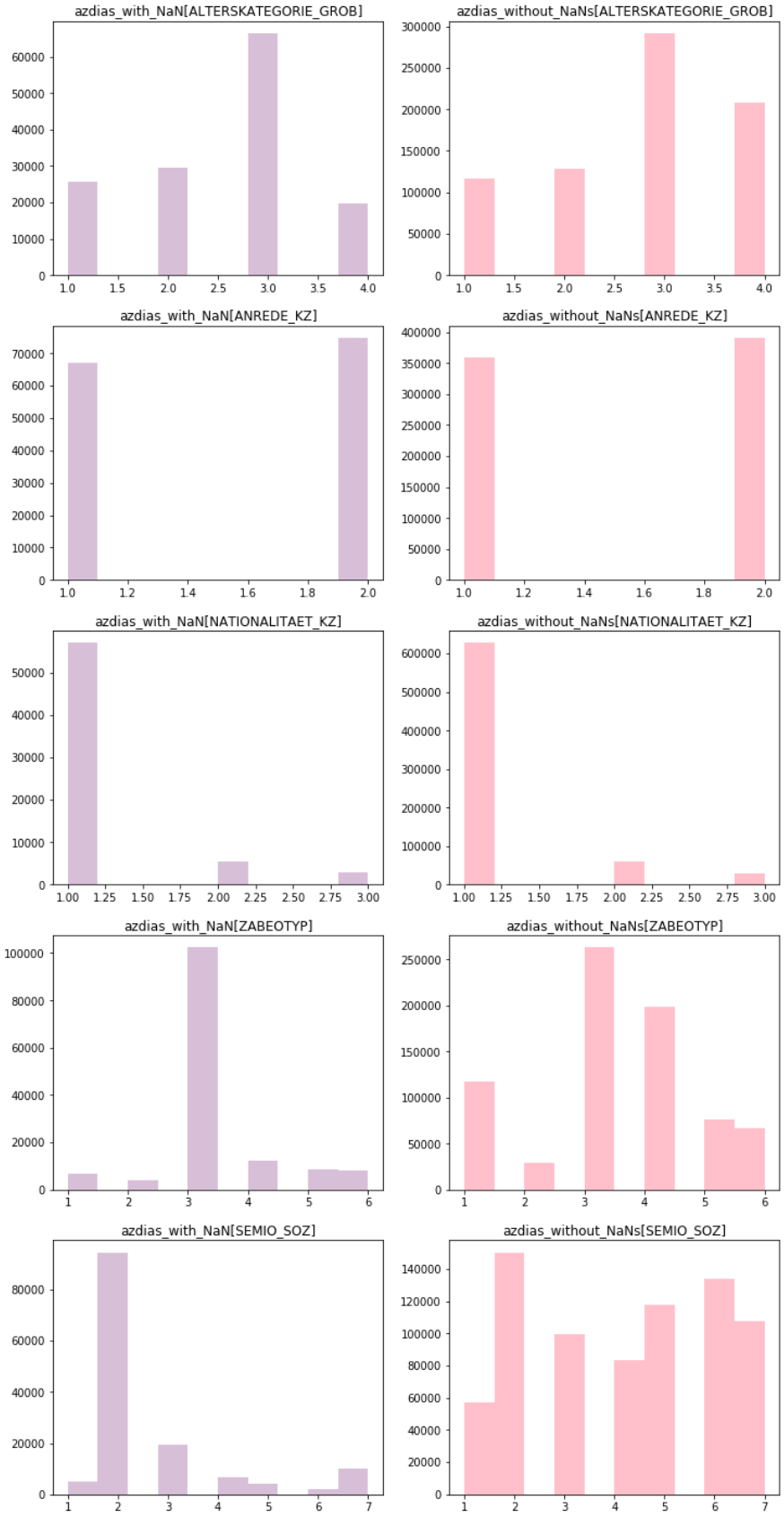
Above is the plot showing distribution of missing values in the rows, and the number of rows with NaNs exceeding 30% were 105805

My next step was to properly encode the unknowns and plot my distribution plot anew to compare the changes, and as can be seen below, the numbers of both columns and rows with more that 30% NaNs escalated quickly. The number of rows became 141931 and columns became about 69:





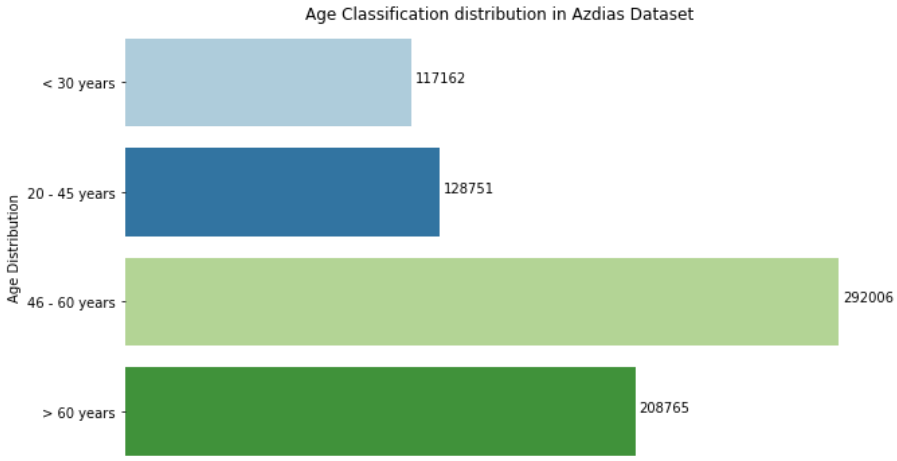
To determine if I could drop rows with missing values > 30%, I decided to separate my dataset into 2 datasets, based on “rows with NaNs exceeding 30%”, and plotted any 5 columns from each dataset to compare trend / pattern and see if it was aptly captured in the dataset without NaNs. As shown in the figure below, I found that all patterns were appropriately captured in the dataset without NaNs. With this justification, I dropped all rows with NaNs > 30%

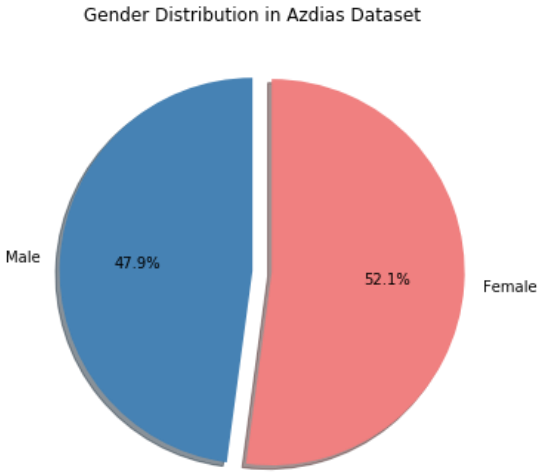


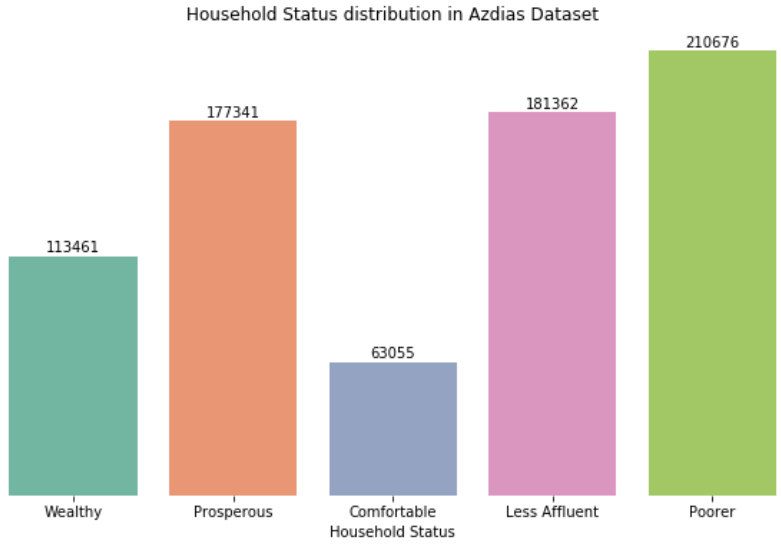
My next step was to investigate the 6 object columns I had because I was certain they had been improperly encoded.

- For “EINGEFUEGT_AM” which was a datetime encoded incorrectly as object, I split into three columns: Year, Month and Day and dropped this initial column
- For "CAMEO_INTL_2015", “CAMEO_DEU_2015” and “CAMEO_DEU_2015”, I kept only "CAMEO_INTL_2015" as that was an internationally accepted standard and dropped the others. I went ahead to encode "CAMEO_INTL_2015" into 5 values – Wealthy, Prosperous, Comfortable, Less Affluent and Poorer
- For “OST_WEST_KZ”, I encoded “W” as 0 and “O” as 1
- Lastly, I dropped some similar columns from my dataset

Before encoding my categorical variables as dummy variables, I decided to get some general stats from my dataset like gender distribution, age classification distribution, et al as shown below:







Lastly, to conclude my preprocessing stage, I converted my categorical variables to dummy variables.

My Udacity_AZDIAS_052018.csv dataset became 749290 rows and 389 columns after preprocessing

I also created all these in a function for ease of application to my other datasets

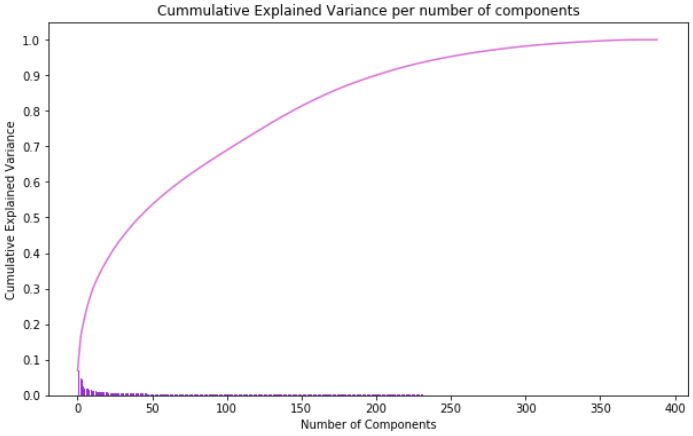
Asides scaling my dataset using Sklearn's StandardScaler function , I decided against using the simple imputer to fill in my remaining NaNs as I felt that introduced some bias, I instead used SKlearn's Iterative Imputer which is a strategy for imputing missing values by modeling each feature with missing values as a function of other features in a round-robin fashion. This added some extra computation time to my notebook, but I think it was well worth it in terms of quality of my models

5 Implementation

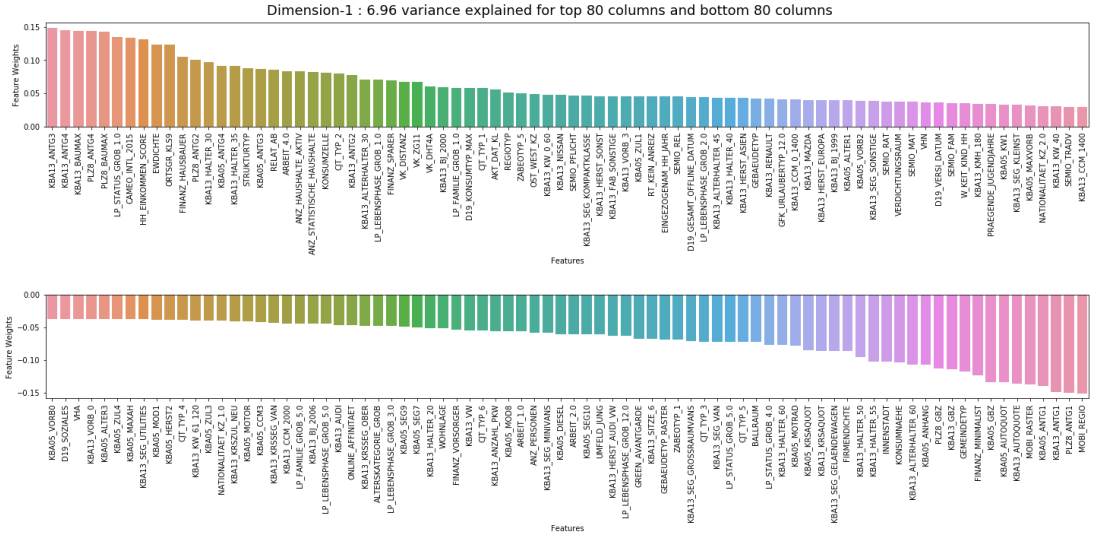
5.1 Customer Segmentation

Before applying my clustering technique (Kmeans), I decided to reduce the dimensions of my dataset to avoid the curse of dimensionality. I used PCA to achieve this.

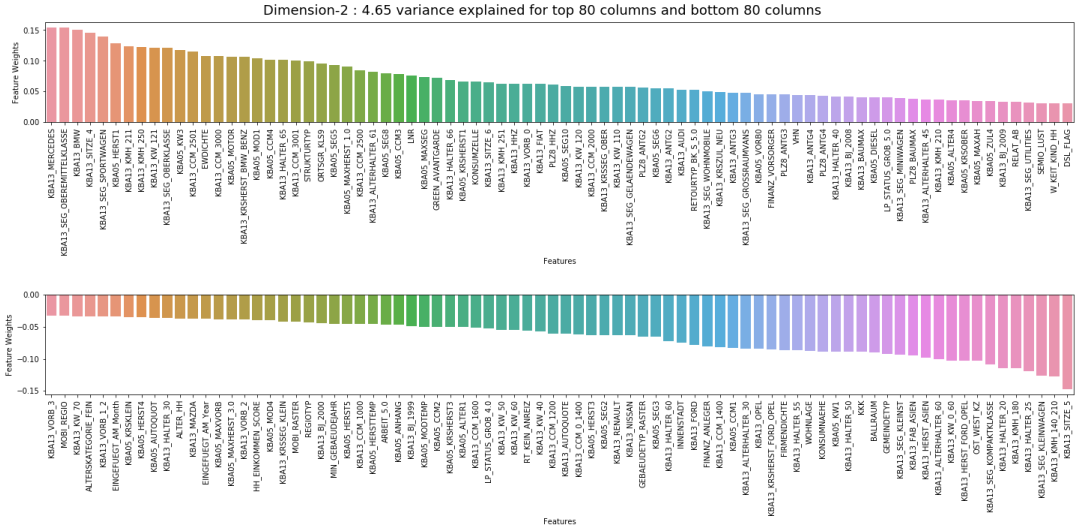
I first ran PCA on all the features I had (389), and then made a plot of cumulative explained variance against number of components. The aim was to achieve at least 90% variance, and this was achieved with 200 components as opposed to the initial 389 components I started with



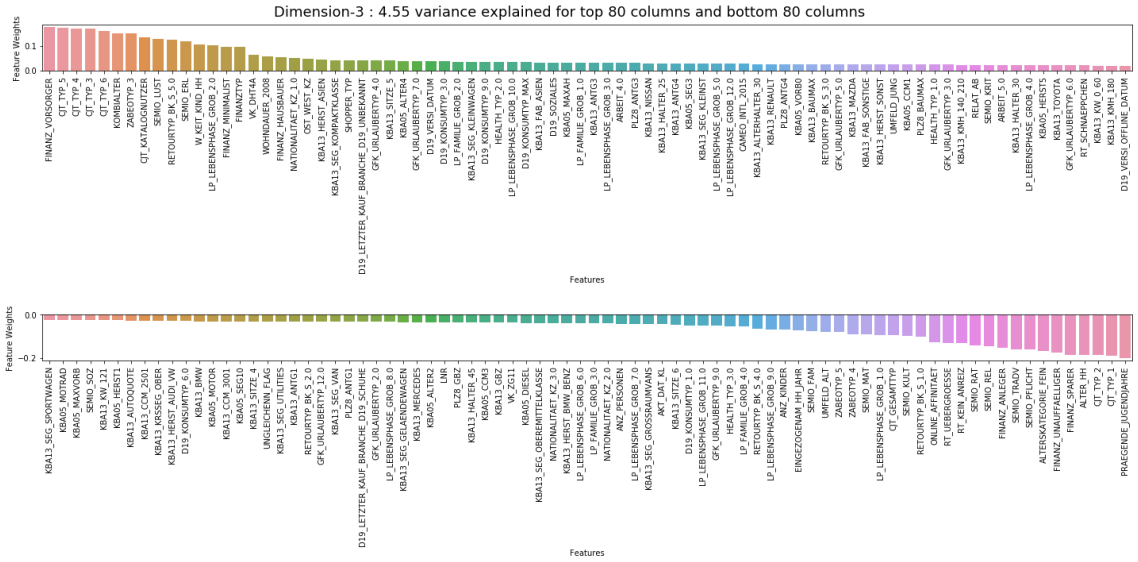
And below is the variance explained for top 80 and bottom 80 columns for Dimension 1



Below is the variance explained for top 80 and bottom 80 columns for Dimension 2

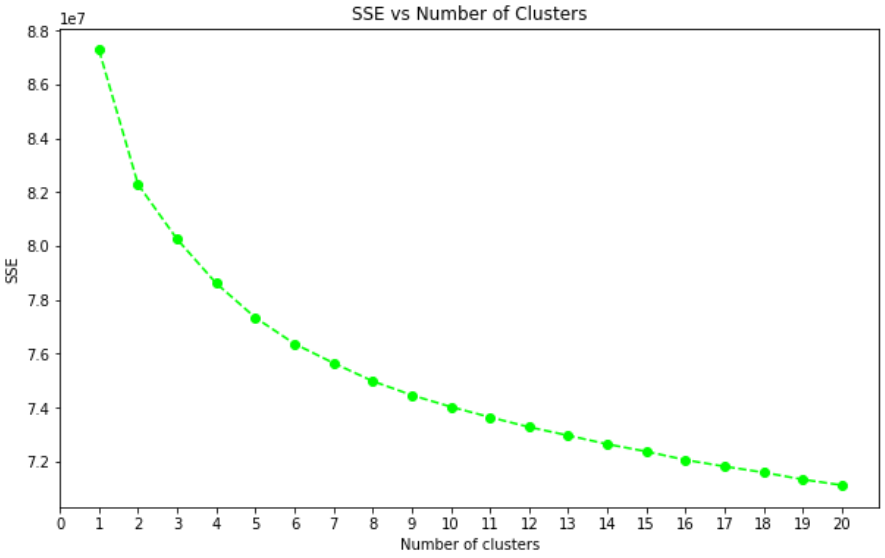


Below is the variance explained for top 80 and bottom 80 columns for Dimension 3



I re-ran my PCA with just 200 columns as determined above and KMeans was applied to the resulting dataset.

To determine my optimal “k”, I used the elbow method. I ran KMeans for number of clusters between 1 and 20 inclusive and plotted as shown below:



I determined my optimal “k” to be 12 and re-ran my KMeans algorithm to output 12 clusters.

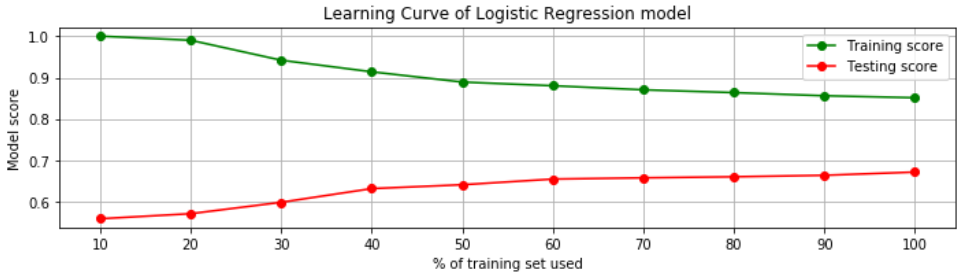
I did this for both the Udacity_AZDIAS_052018 and Udacity_CUSTOMERS_052018 dataset.

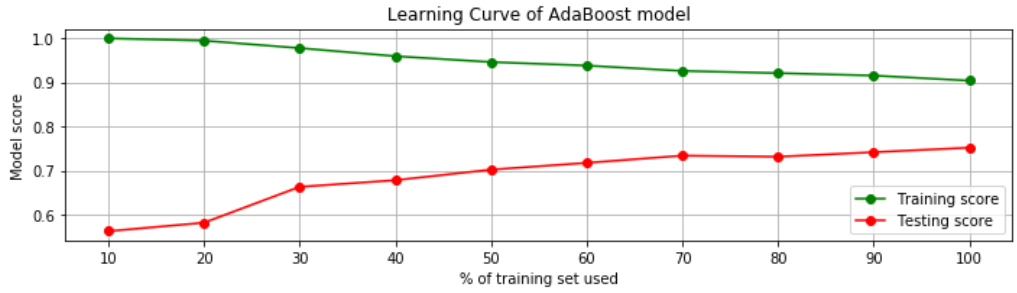
5.2 Supervised Learning

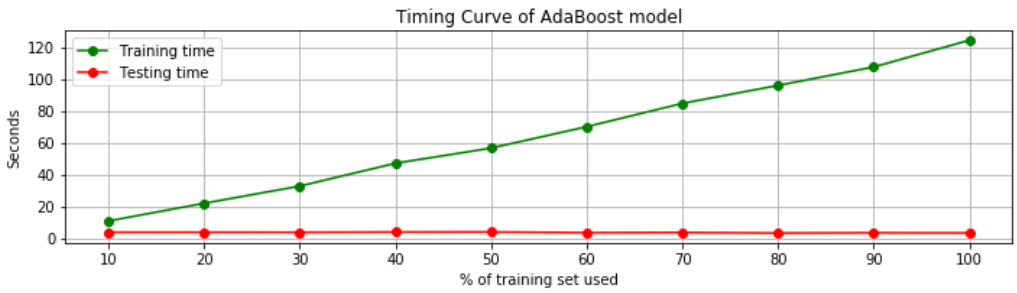
Here, I decided to experiment with 4 learners: Linear Regression, Adaboost, Random Forest and XGBoost. I used base settings for all classifiers and “ROC_AUC” was my preferred metric.

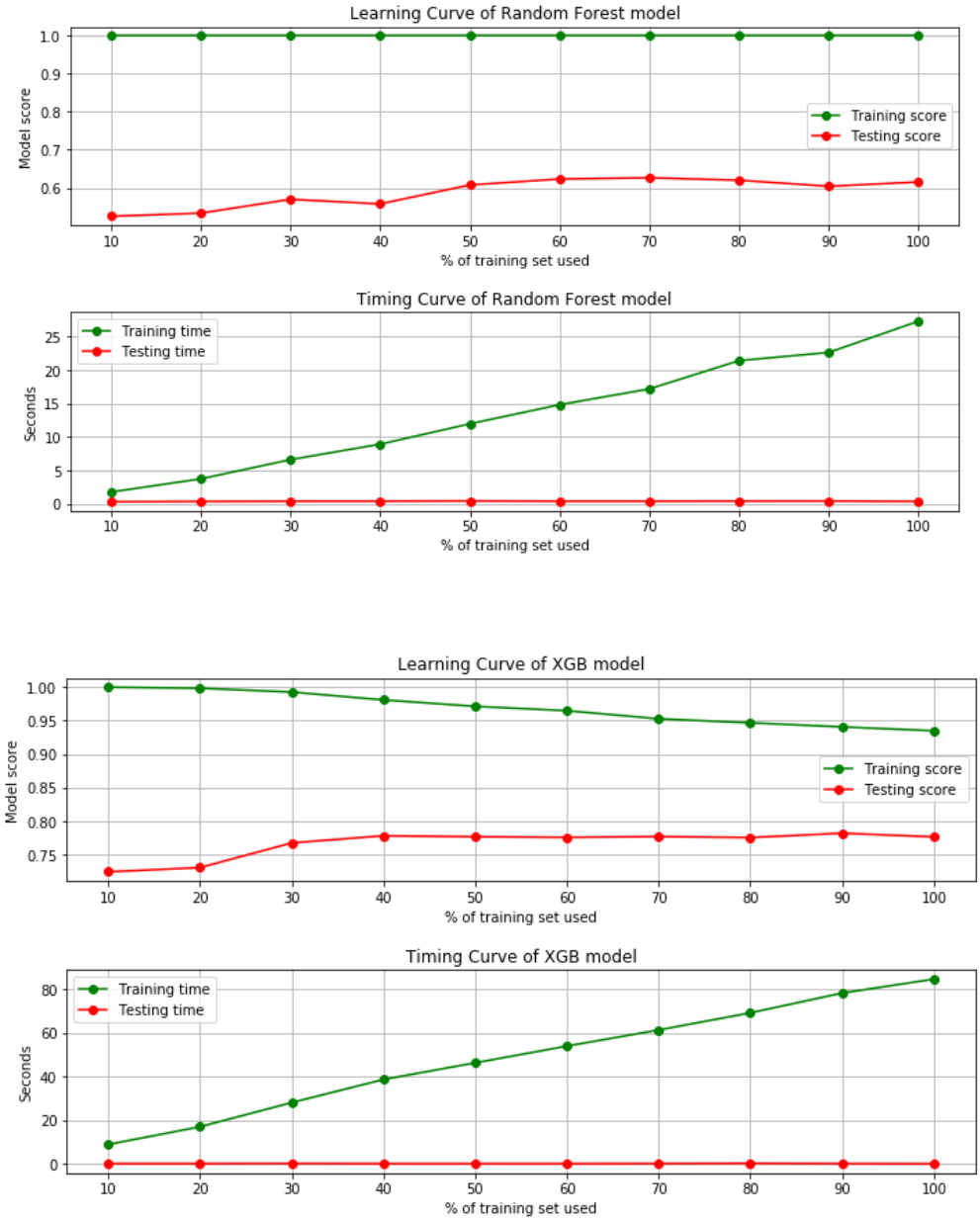
As a result of the unbalanced set of my dataset, I decided to use StratifiedKFold to split my data, and I used 5 folds

I did a plot of both the learning curve and timing curve for each learner and below was the plot I received:









My best performer was XGBoost with base settings and my score was 0.78. My result is captured in more details in the “Result” session of this document. I decided to refine my XGBoost learner some more to get better performances

6 Refinement

Here, I decided to use Sklearn’s GridSearchCV to find the best parameters that would improve my model. I was constrained by computation time, and so I ran GridSearchCV on limited parameters (this can be expanded on to get a better performing model). I was able to get some improvement in my score which became 0.79 and my best parameters were at :

{“colsample_bytree”: 0.5, “gamma”: 1.0, “learning_rate”: 0.1, “max_depth”: 2, “min_child_weight”: 8, “n_estimators”: 200, “scale_pos_weight”: 50, “subsample”: 0.5}

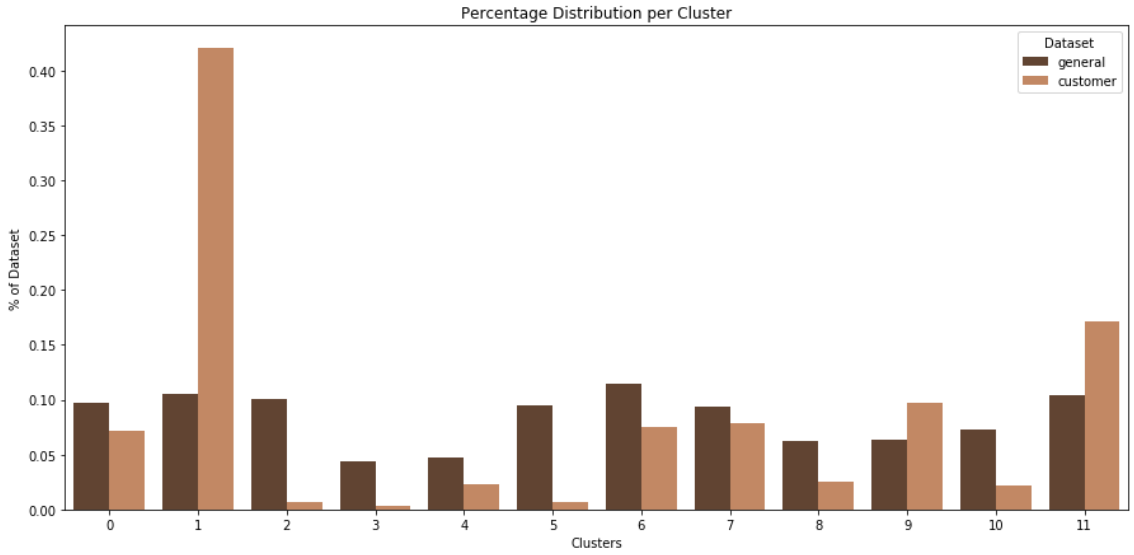
I did apply this to my kaggle submission and my score was 0.67592 at position 138.

I decided to further tweak my learner and use an XGBRegressor instead of XGBClassifier with my best parameters from above and that improved my kaggle submission score to 0.77110 at position 106

7 Model Evaluation, Validation and Justification

7.1 Unsupervised Learning

Below is my result from clustering both my General population and Customer dataset:



Here, I can see that cluster 1 is over-represented in my Customer dataset, while cluster 2 is under-represented in my Customer dataset.

Looking at some of the attributes of cluster 1 as compared to cluster 2, I can hazard a guess that cluster 1 are wealthy urban dwellers with families, well ingrained into the society and have great purchasing power, while cluster 2 are younger, openminded households that are less calculated in their spending, thus have higher financial interest and typically live in poor neighborhoods

Here is a table that compares some characteristics of Cluster 1 against Cluster 2:

Underrepresented cluster – Cluster 2	Overrepresented cluster – Cluster 1
Born in the 80s	Born in the 50s
Comfortable households	Prosperous households
Low investors	Very high investors
Lower income	High income
Higher financial interest	Lower financial interest
Average purchasing power	High purchasing power
Poor neighborhood	Average neighborhood
7 years of residence	Length of residence more than 10 years

7.2 Supervised Learning

For my supervised learning, I did make use of 4 learners with base settings. Below are the roc_auc scores and time to train and test:

Model	ROC_AUC_ Train_score	ROC_AUC_ Test_score	Train_time	Test_time
Logistic Regression	0.85	0.67	38.08	0.02
Random Forest	1	0.62	27.29	0.38
AdaBoost	0.9	0.75	124.54	3.35
XGBoost	0.93	0.78	84.49	0.17

XGBoost performed the best with unseen (test) data, and Random forest had the fastest training time. I proceeded with XGBoost, and after finetuning my results using GridSearchCV, my final result on test data was 0.79 with parameters:

{“colsample_bytree”: 0.5, “gamma”: 1.0, “learning_rate”: 0.1, “max_depth”: 2, “min_child_weight”: 8, “n_estimators”: 200, “scale_pos_weight”: 50, “subsample”: 0.5}

StratifiedKFold with 5 folds split my data into train and crossvalidation (cv) set. It trained my model on the train set and validated on the cv (test) set using roc_auc. My final score outputted is the average score for all 5 folds

On the Kaggle competition, my XGBoost model tweaked with the parameters listed above had a score of 0.67592 at position 138.

I further decided to use an XGBRegressor with objective as “binary:logistic” and that gave me better scores on my Kaggle competition with 0.77110 at position 106

8 Reflection and Improvement

This project focused on clustering both the general population and the customer dataset as well as predicting future customers. One part I found really interesting and challenging was how to handle missing data without introducing bias or losing too much relevant information from the dataset. This was a challenging bit for me that had to be handled in stages. Also, the superior method (Iterrative Imputer) I chose took a long time to complete on my laptop which was a major challenge for me as well.

The performance metrics achieved in my supervised learning model, as well as in the Kaggle competition are fairly okay, crossing the 0.75 mark, I am confident that this can be improved upon.

I believe further tweaking of the parameters in the XGBoostRegressor model would provide better results

As part of future developments, I believe more work can be done to tweak the parameters to improve performance as well as add an interactive visualization element say in an app for users

9 References

Tianqi Chen and Carlos Guestrin (2016): XGBoost: A Scalable Boosting Tree System Available online at http://dmlc.cs.washington.edu/data/pdf/XGBoostArxiv.pdf

Xgboost documentation (2016) Available on https://xgboost.readthedocs.io/en/latest/#

Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011