CAPSTONE PROJECT ADESHOLA AFOLABI

MACHINE LEARNING NANODEGREE 23/10/2020

Arvato Customer Segmentation Report and Click Prediction

DEFINITION

PROJECT OVERVIEW

The essence of this project is to help a mail-order company identify several ways in which digital campaigns can be optimized and personalized for its end users. The personalization referred to here is majorly about identifying segments within already existing customers and extending that to the general population (referred to as Azdias) in the dataset. The new segments identified can now help with the digital campaigns the company will execute and the type of product offerings the "customers to-be" will be interested in. In addition, the outcome of this segmentation will be fed into a model (a click prediction model) that will help when optimizing marketing campaigns (via emails) the type of users that will engage (respond) to the campaign or not.

This project employs several Data Science and Machine Learning algorithms to achieve the final goal. To simplify further, the project was divided into two:

- Unsupervised Learning Approach
- Supervised Learning Approach

PROBLEM STATEMENT

The ultimate problem to be solved in this project is:

- Identify how the mail-order company can acquire new users/clients efficiently. Upon identifying these users, a marketing campaign is subsequently sent to them.

After identifying the right users, the following problems will be solved:

- Customer segments will be created from the general populace based on the analysis of attributes of already established customers
- The output of the customer segmentation will be fed into a machine learning algorithm (supervised) which will predict whether or not an individual will respond to a marketing campaign.

METRICS

The evaluation metric for this project is AUC for the ROC curve, relative to the detection of customers from the mail campaign. A ROC, or receiver operating characteristic, is a graphic used to plot the true positive rate (TPR, the proportion of actual customers that are labeled as so) against the false positive rate (FPR, the proportion of non-customers labeled as customers).

True Positive Rate (TPR) = (True Positive) / (True Positive + False Negative)

This is also called sensitivity or recall.

False Positive Rate (FPR) = (Flase Positive) / (False Positive + True Negative)

The line plotted on these axes (below) depicts the performance of the algorithm as we sweep across the entire output value range. We start by accepting no individuals as customers (thus giving a 0.0 TPR and FPR) then gradually increase the threshold for accepting customers until all individuals are accepted (thus giving a 1.0 TPR and FPR)

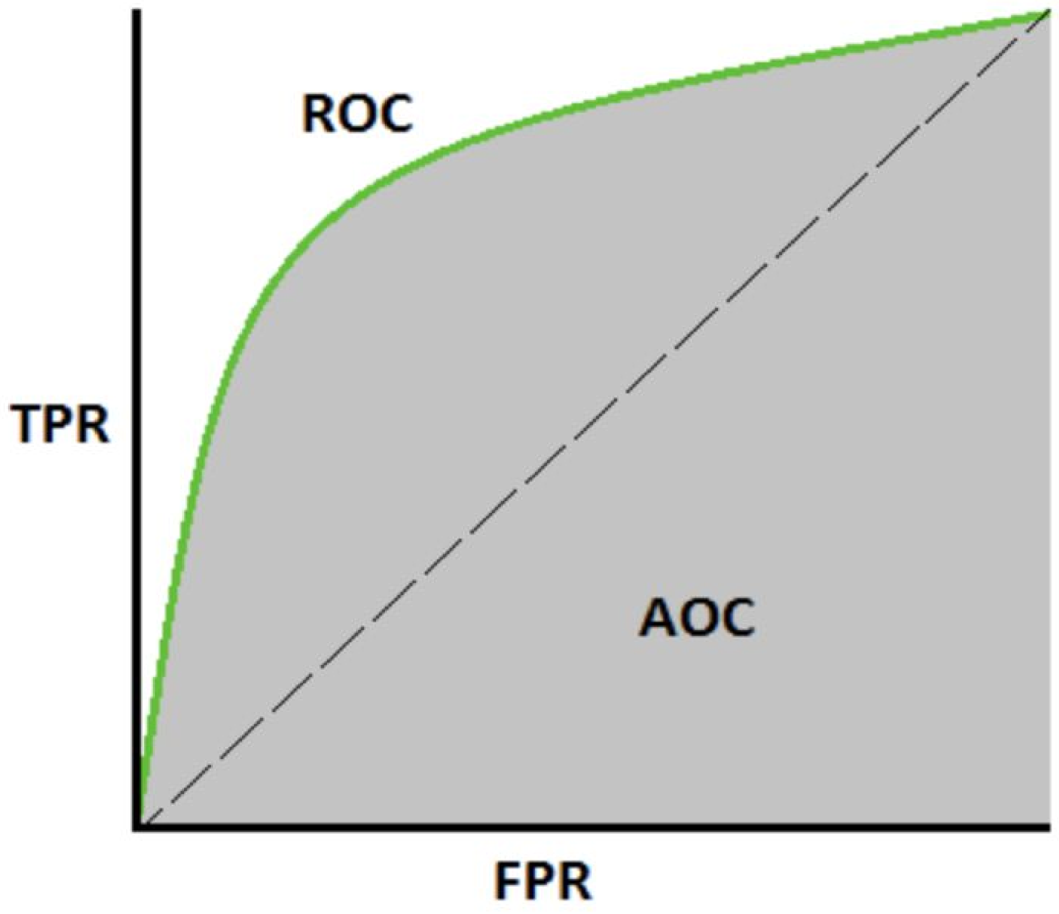


Fig 1. AUC ROC Curve

ANALYSIS

DATA EXPLORATION

The datasets used for this project is provided on the Udacity workspace and it is divided into four namely:

- 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).

Each data point represents a user and the corresponding attributes such as age, gender, location, employment classification, social status, range of transactions across different channels, etc. The already established customers(Udacity_CUSTOMERS_052018.csv) of the mail-order company will be used to identify lookalikes from the general population(Udacity_AZDIAS_052018.csv) of people in Germany and this analysis will be used to make predictions on which user ranks best for a marketing campaign. This will be done with the train and test CSV files provided.

In addition, a data dictionary was provided explaining each of the attributes in the dataset.

- DIAS Attributes - Values 2017: this contains 4 columns namely: Attribute, Description, Value, and Meaning. This helps in understanding each of the attributes and also, their significance based on the rankings
- DIAS Information Levels - Attributes 2017: this is a top-level list of the attributes and descriptions. They are organized by informational category

EXPLORATORY DATA ANALYSIS AND VISUALIZATIONS

This is the very first step of any Data Science task. A range of questions come to mind, and as a result, answering these questions will help in solving the bigger problem. Some of the questions I asked myself were:

1. Is the data clean? Are there missing values or mixed data types?
2. What level of granularity does each attribute have in terms of values?
3. What informs my decision to either drop a row or column?
4. Are the attributes consistent across all datasets?
5. Is the distribution even across numerical attributes?

In approaching each of these questions, I knew it was important to have a reusable code in order to avoid repetition since I would be dealing with two datasets.

Is the data clean?

Two key observations were noted upon loading the dataset into a Pandas dataframe.

- Two attributes have mixed data types (containing numbers and a string)

- There were a lot of missing values for both Azdias (general population) and the customer datasets.

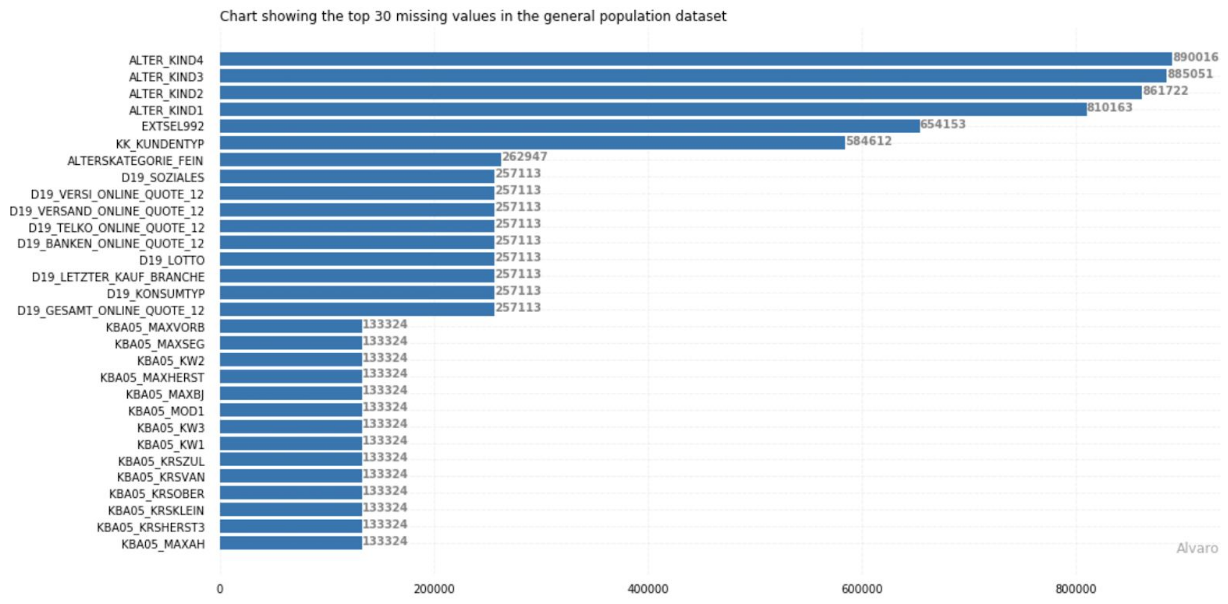


Fig 2. Top 30 missing values for Azdias Dataset

From the chart above, there were a lot of attributes that had missing values. An example is the ALTER_KIND4 attribute which had 890 016 missing values out of 891 211 (99.86% of the entire dataset). This will definitely not add any useful information to our data and subsequently, the model.

A similar trend can be seen in Fig 3. below for the customers' dataset. The only difference here is that the customers' dataset has 191 652 rows and missing values constitutes for over 90% in some of the attributes.

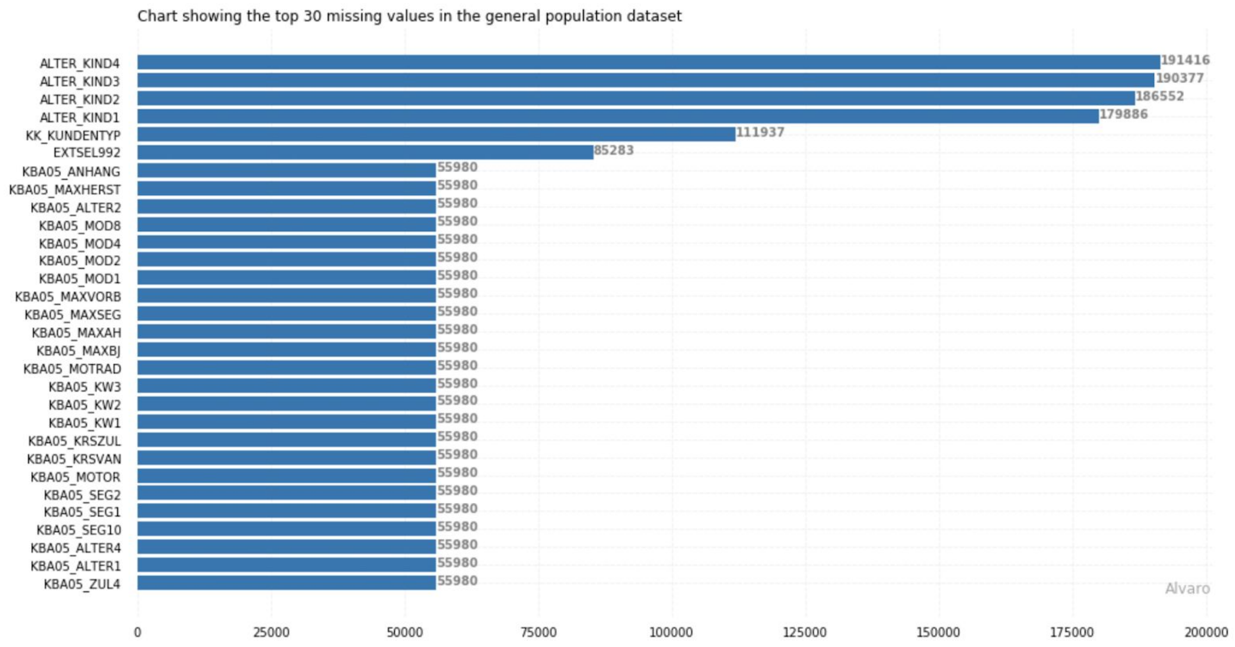


Fig 3. Top 30 missing values for the Customers' Dataset

Upon answering this question, it was necessary to come up with a reusable data cleaning script that will handle these scenarios. Here are some of the methods I came up with:

- A method to handle mixed datatypes (change the less dominant representation to NaN)
- A method to handle all NaN values using the mean of the data

What is the level of granularity of non-numerical attributes?

Cardinality is a problem when building any machine learning algorithm. When the cardinality of datasets becomes too high, compute might be a problem and also, some key information can be lost in that cardinal space. The peculiarity of this dataset is such that there are already a lot of attributes to deal with and creating more attributes when handling categorical columns (such as using one-hot encoding) will further compound the dataset. In order to have a sense of what was happening, the level of granularity of the categorical attributes had to be observed. Here are my findings:

- CAMEO_DEU_2015 has a relatively low cardinality. It has 45 unique attributes
- CAMEO_DEUG_2015 has a relatively low cardinality. It has 19 unique attributes
- CAMEO_INTL_2015 has a relatively low cardinality. It has 43 unique attributes
- D19_LETZTER_KAUF_BRANCHE has a relatively low cardinality. It has 35 unique attributes

- EINGEFUEGT_AM has a high cardinality. It has 5162 unique attributes
- OST_WEST_KZ has a binary classification with 2 unique attributes

It is worth noting here that the threshold for cardinality was set at 100. Any attribute with more than that was classified as high cardinality.

What informs my decision to either drop a row or a column?

To answer this question, I had to dive deeper into the data structure and the dictionary provided. Apart from the missing data (NaN), it was also evident that some attributes represented as 0, 9, -1 were either unknown or missing. In order to deal with that, I had to isolate the features that had these representations and think of ways to add them with the NaN rows. Upon doing this, the NaN values went up significantly and I realized many attributes with greater than or equal to 30% missing values had to be dropped across columns. Digging further, it was also necessary to apply a similar condition for 14% missing values across the rows. Fig 4 below shows the new chart after including new NaN rows from representations of 0, 9, and -1 that were either unknown or missing in the Azdias dataset.

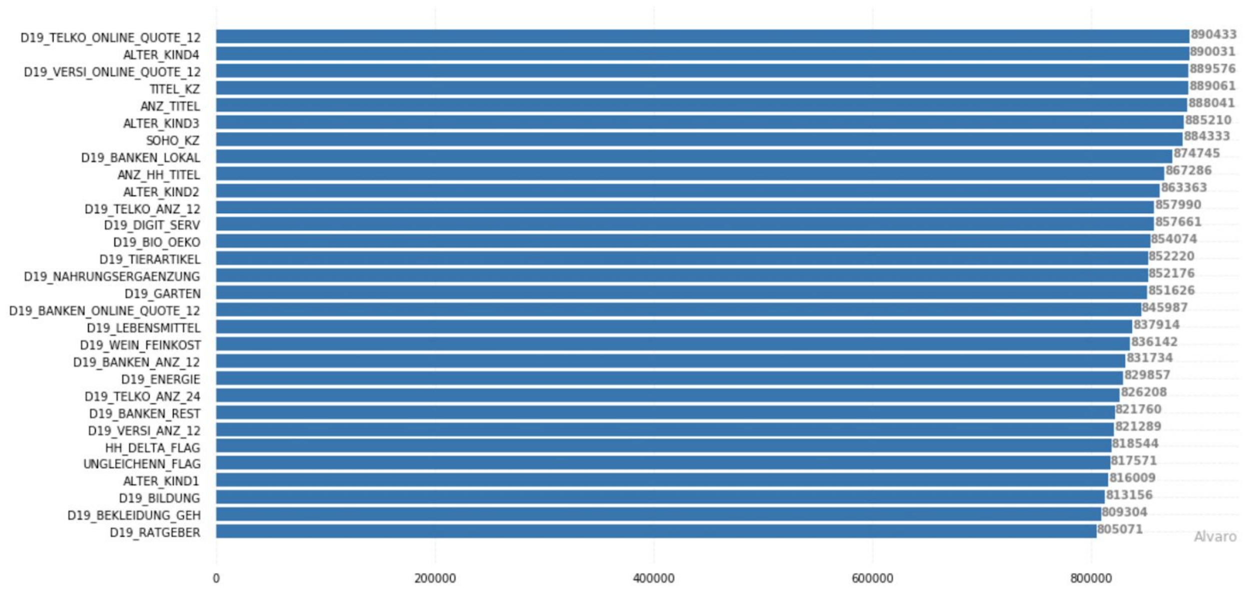


Fig 4 New NaN values after correcting the representations of 0, -1 and 9

Are the attributes consistent across all datasets?

The Azdias and Customers' dataset are very similar and comprise of 99.2% of the same attributes with the exception of the following:

- Product Group (which has cosmetic and food, cosmetic, and lastly food)
- Customer Group (either a single buyer or multi-buyer)
- Online Purchase (0 or 1 which indicates whether you buy online or not)

For consistency's sake, and to be able to identify clusters across the same attributes, these had to be dropped. However, these features came in handy after the clustering algorithm was completed. I was able to identify each of the dominant groups in these clusters.

Is the distribution even across numerical attributes?

Traces of outliers were detected in the numerical attributes. The distribution of the data was not normal as well, which could potentially skew the outcomes. To effectively handle this, I scaled the data before fitting it into a model. I also ensured I avoided Data Leakage by scaling appropriately.

DATA ENGINEERING AND CLEANING

This step of the project was an interesting one. Using some of the best practices in the software engineering class, a clean_data was was written. It has some of the following methods:

- identify_columns (to automatically detect numerical and categorical columns)
- Handle_mixed_dtypes (to take care of columns with mixed datatypes such as CAMEO_DEUG_2015 and CAMEO_INTL_2015
- handle_unknown (to handle valid representations of -1, 0, 9 as NaN values)
- check_nan_before (to check NaN values before handling unknown)
- check_nan-after (to check NaN values after handling unknown representations)
- plot_nan (a plotter for NaN values)
- drop_na (a method to drop NaN values based on certain thresholds)
- engineering (creating new features and dropping old ones)
- handle_nan (use of mean and mode to fill NaN values)

To elaborate further on one of the most important methods, I would be diving deeper into the engineering bit.

- The PRAEGENDE_JUGENDJAHRE was broken down into two; Demography and REUNIFICATION by mapping the values appropriately. The attributes explains dominating movement in Germany from the 50s to the 90s
- The WOHNLAG attribute which indicate the kind of neighborhood a user resides in. It was engineered into two (rural and non-rural neighborhood)

- LP_LEBENSPHASE_FEIN was broken down further based on the type of income and the demography of users based on the income level. They were termed "DEMOGRAPHY" and "AFFLUENCE"
- OST_WEST_KZ refers to flag indicating german reunification; in which the German Democratic Republic (GDR) became part of Federal Republic of Germany (FRG) to form the reunited state of Germany. The Western side was represented as 0 and the eastern side was represented as 1

Upon performing data engineering on this four attributes, the next step was to apply the handle NaN method. After all was done and completed, the new data had a shape of ((891221, 303), (191652, 303)) for the Azdias and Customers' dataset respectively. The total column dropped as a result of either missing values or data engineering was 66.

ALGORITHMS AND TECHNIQUES

The approach to solving this problem is divided into two:

- The unsupervised learning approach (the use of PCA and KMeans)
- The supervised learning approach (the use of any supervised learning algorithm)

UNSUPERVISED LEARNING

This approach will be used to identify segments within the dataset. To begin with, a principal component analysis (PCA) will be done; this helps with dimensionality reduction and also, identifying attributes that are most important in identifying the segments. The KMeans algorithm will use the output of the PCA to identify the segments within the dataset by clustering similar attributes together. The number of clusters (k) will be determined using the elbow method and each of the cluster centroid determines how far a datapoint(user) is to the cluster.

SUPERVISED LEARNING

Upon clustering the dataset into different segments, and identifying the most important attributes using PCA, a supervised algorithm can be trained. Using the train dataset provided which has the extra field 'RESPONSE', we try a host of supervised machine learning algorithms or deep learning algorithms and determine which will perform best on the test dataset. A validation dataset will also be created from the train dataset to give an idea of the model performance. The ROC-AUC, accuracy and f1 score will be the evaluation metric to look out for.

BENCHMARK

For the supervised learning approach, a way to benchmark the resulting model will be to use the top ranked models on Kaggle to compare the performance of my model. Right now, the team/person on the leaderboard has a score of 84.7% (using the AUC-ROC final score). Any value in the range of + or - 5% of the leaderboard will be acceptable for me. I will be looking to optimize for AUC-ROC score of between 79 - 89%.

There are no clear benchmarks to use for the unsupervised learning approach, however, from the additional fields provided in the Customers' dataset (PRODUCT_GROUP, CUSTOMER_GROUP and ONLINE_PURCHASE), we can map the resulting clusters back to these attributes and identify the most important clusters from the Azdias dataset which can now be used for more informed and personalized digital campaigns.

METHODOLOGY

PRINCIPAL COMPONENT ANALYSIS (PCA)

PCA attempts to reduce the number of features within a dataset while retaining the "principal components", which are defined as weighted, linear combinations of existing features that are designed to be linearly independent and account for the largest possible variability in the data. You can think of this method as taking many features and combining similar or redundant features together to form a new, smaller feature set. In order to perform PCA on the dataset using Sagemaker, I had to do two things:

1. Perform One Hot Encoding (to convert all categorical data to numerical)
2. Scaling of the entire dataset (using MinMaxScaler)

Upon completing these steps, it was necessary to grant necessary permissions and specify some parameters in order to use Sagemaker to train the PCA model. Some of the steps are:

1. Grant Sagemaker full IAM access (this would allow Sagemaker communicate with other services without any restrictions)
2. Store the current Sagemaker session
3. Create an s3 default bucket using the same session (this enables that the bucket is created in the same region)
4. Provide the s3 output path for each bucket (the Azdias and Customers' data)
5. Specify PCA Sagemaker parameters such as role, the instance count and size, number of components and the session)

6. Convert the scaled data to a numpy array and subsequently store it as a record set (this is the requirement for all Sagemaker algorithms in order to train optimally)
7. Train the PCA model

Once the training is complete, the model artifacts are stored in the specified s3 output bucket where they can be retrieved for further analysis.

PCA MODEL ATTRIBUTES

The PCA model attributes are stored as MXNET arrays (for computational speed) and two major attributes can be retrieved here; the V and S components makeup. These two components will eventually come in handy when there is a need to explain the variance of the initial data attributes and the PCA transformed attributes. Here are some findings:

- Picking top 50 attributes gave a variance of around 66% for both Azdias and Customers
- Top 100 attributes gave a variance of >80% for both Azdias and Customers data
- The top 150 attributes gave a variance of 90% for both Azdias and Customers data

While selecting the top n components to use in the PCA transformed data model, I typically want to include enough components to capture about 80-90% of the original data variance. In this project, I am looking at generalizing over a lot of data and I will aim for about 80% coverage; which leaves me with the top 100 attributes.

To display some of the components and their corresponding weights, the method display_method in cell 33 was used. The method took in some input parameters and joined the PCA transformed data with the original data to get a sense of the top attributes and their corresponding weights for both Azdias and Customers' dataset.

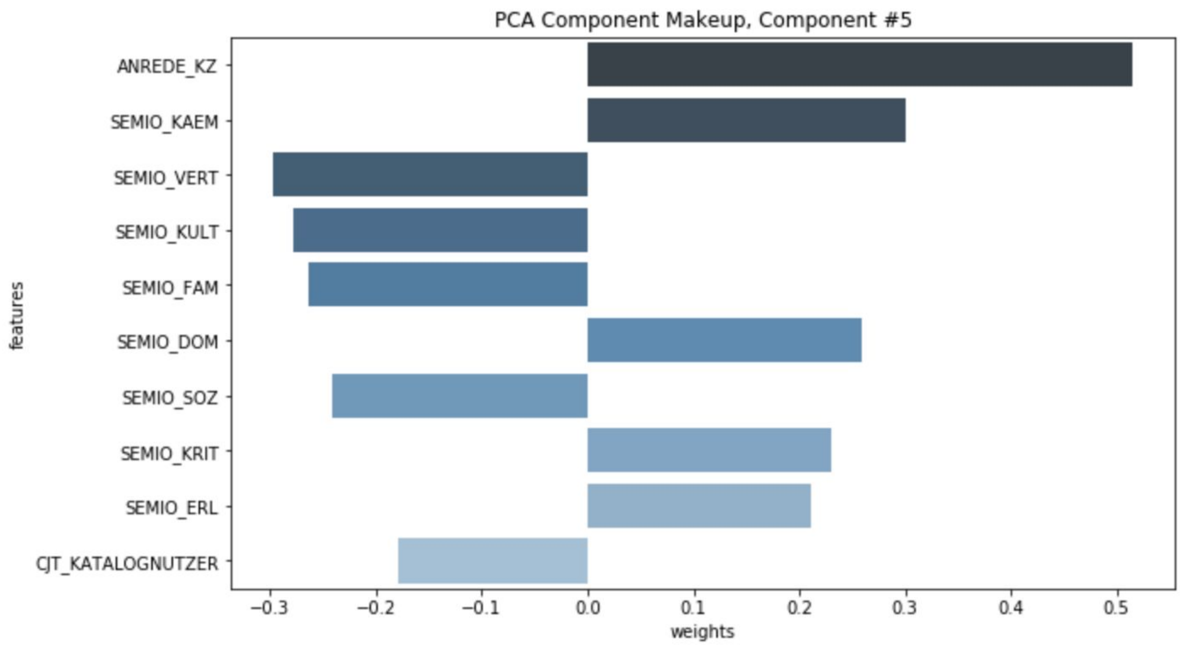


Fig. 5 PCA Component Makeup for the Azdias Dataset

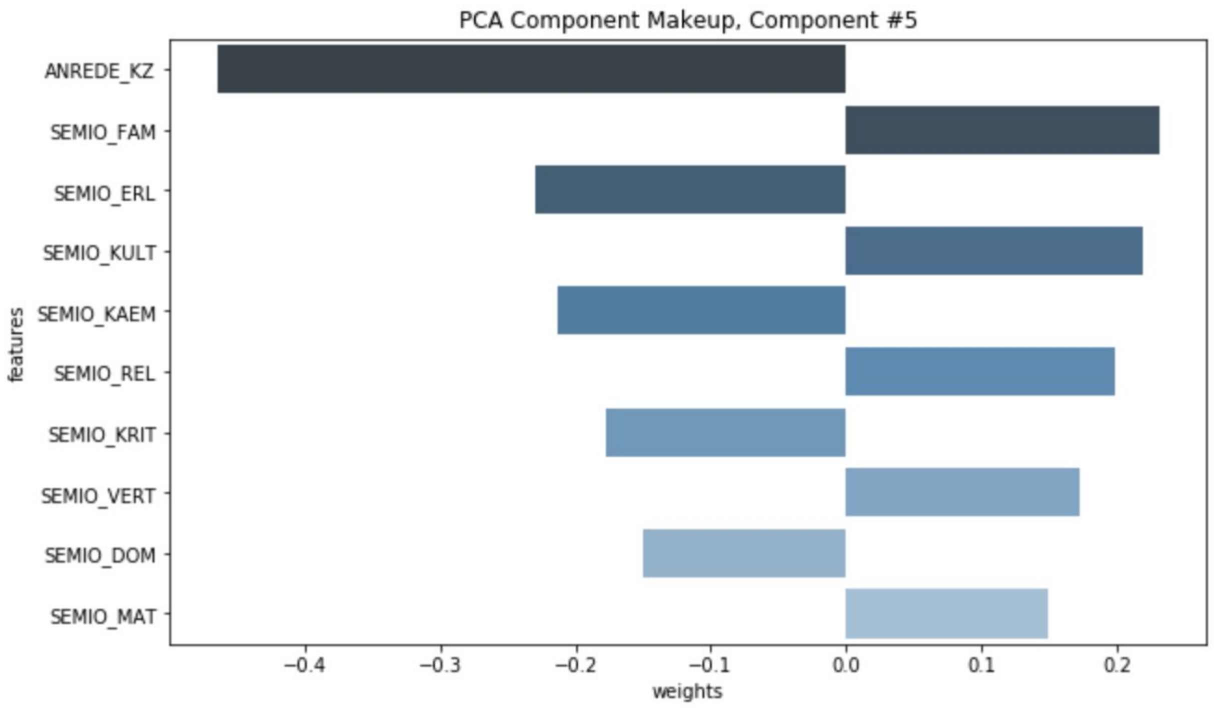


Fig. 6 PCA Component Makeup for the Customers' Dataset

KMEANS IMPLEMENTATION

The feedback from the PCA attributes is used to build a k-means model. K-means is a clustering algorithm that identifies clusters of similar data points based on their component makeup. Since we would have reduced the feature space to 'n' PCA components, we can then cluster on the PCA transformed dataset.

One method for choosing a "good" k is to choose based on empirical data. A bad k would be one so high that only one or two very close data points are near it, and another bad k would be one so low that data points are really far away from the centers.

We want to select a k such that data points in a single cluster are close together but that there are enough clusters to effectively separate the data. You can approximate this separation by measuring how close your data points are to each cluster center; the average centroid distance between cluster points and a centroid. After trying several values for k, the centroid distance typically reaches some "elbow"; it stops decreasing at a sharp rate and this indicates a good value of k.

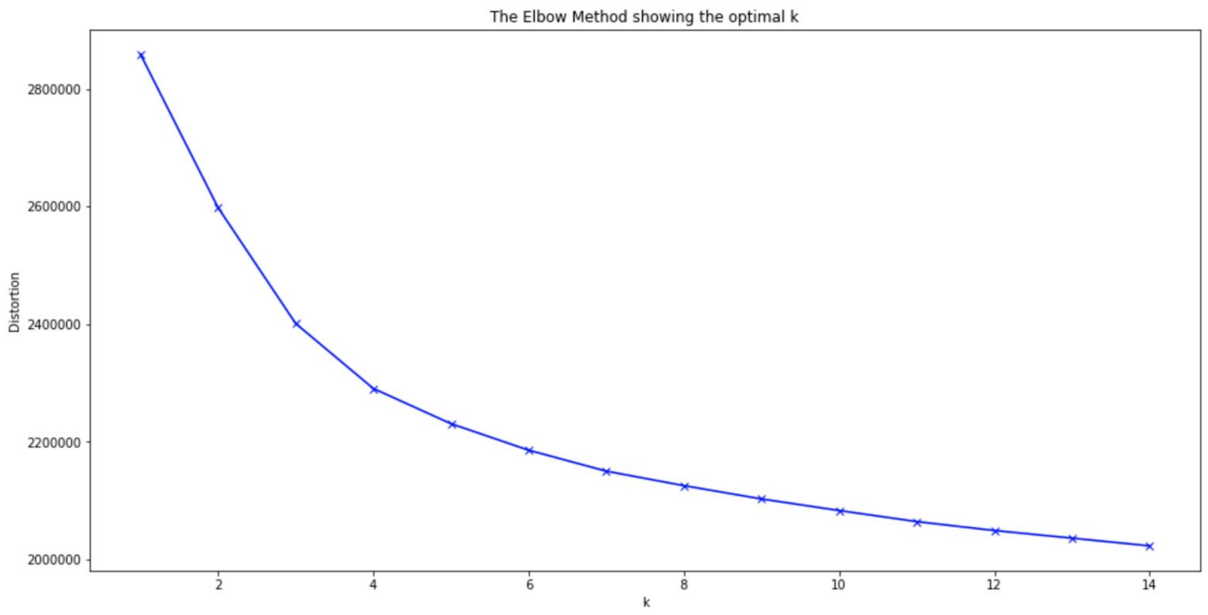


Fig. 7 The Elbow Method showing the optimal K for the Customers' Dataset (PCA)

A similar chart was generated for the Azdias PCA transformed dataset. From the image above, the curve takes a steep turn between k = 4 and k=6. From this, I will be leaning more towards picking a value of k=5.

USE OF SAGEMAKER KMEANS ESTIMATOR

A KMeans estimator requires a number of parameters to be instantiated such as the role, instance type and instance count, which allows us to specify the type of training instance to use, and the model hyperparameters(e.g the optimal k to use).

Before fitting the Sagemaker KMeans model, some steps are necessary. These are:

1. Convert the PCA transformed data to a numpy array
2. Convert the numpy array from 1 above to a record set (recommended data type for all Sagemaker models for faster training)
3. Train the KMeans model

Upon training the model, the next step was to deploy and use the deployed model endpoint to predict the clusters that the data points belong to. Here are the deployment steps:

1. Deploy the trained KMeans model
2. Use the endpoint predict method to predict the clusters
3. Store the results in a list
4. Obtain the cluster information from the list and convert back to a dataframe

CLUSTER ANALYSIS AND RECOMMENDATIONS

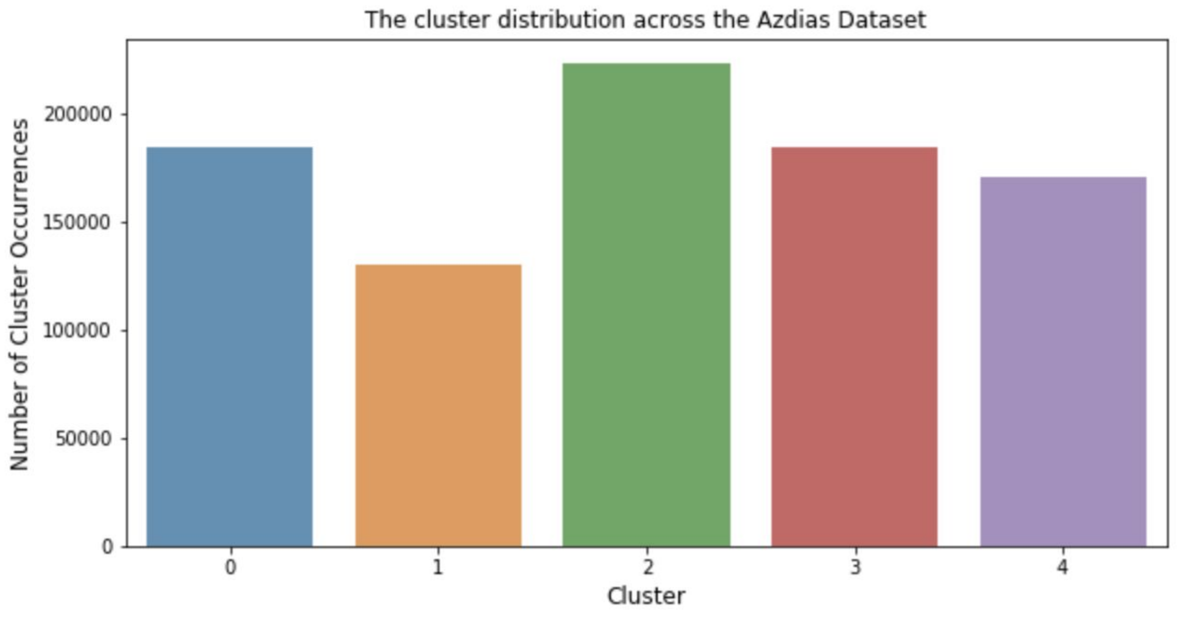


Fig. 8 The Cluster Distribution across the Azdias Dataset

From Fig. 8 above, it is evident that the second cluster is the most dominant cluster out of the five clusters created. In order to get a sense of how to apply the distribution across the clusters, it is necessary to dive deeper into what the already existing customers cluster look like. Before doing this, let’s see the distribution of clusters across the customer dataset and also what the graph looks like placing them side by side.

Fig. 9 below shows a slightly different trait with the cluster distribution. The third cluster here is the dominant cluster, followed by the second cluster which was dominant in the Azdias dataset. However, for the combined chart in Fig. 10 below, we can see a lot of dominance in cluster 2 and 3. The next step here will be to interpret these clusters into how we can use them to solve business problems.

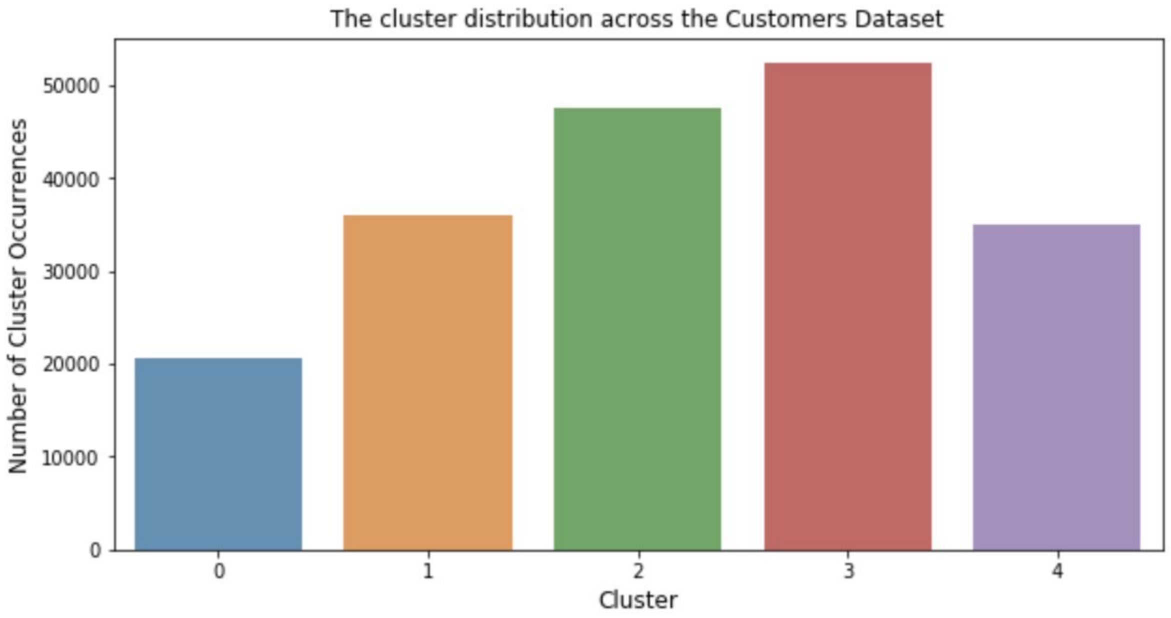


Fig. 9 The Cluster Distribution across the Customers’ Dataset

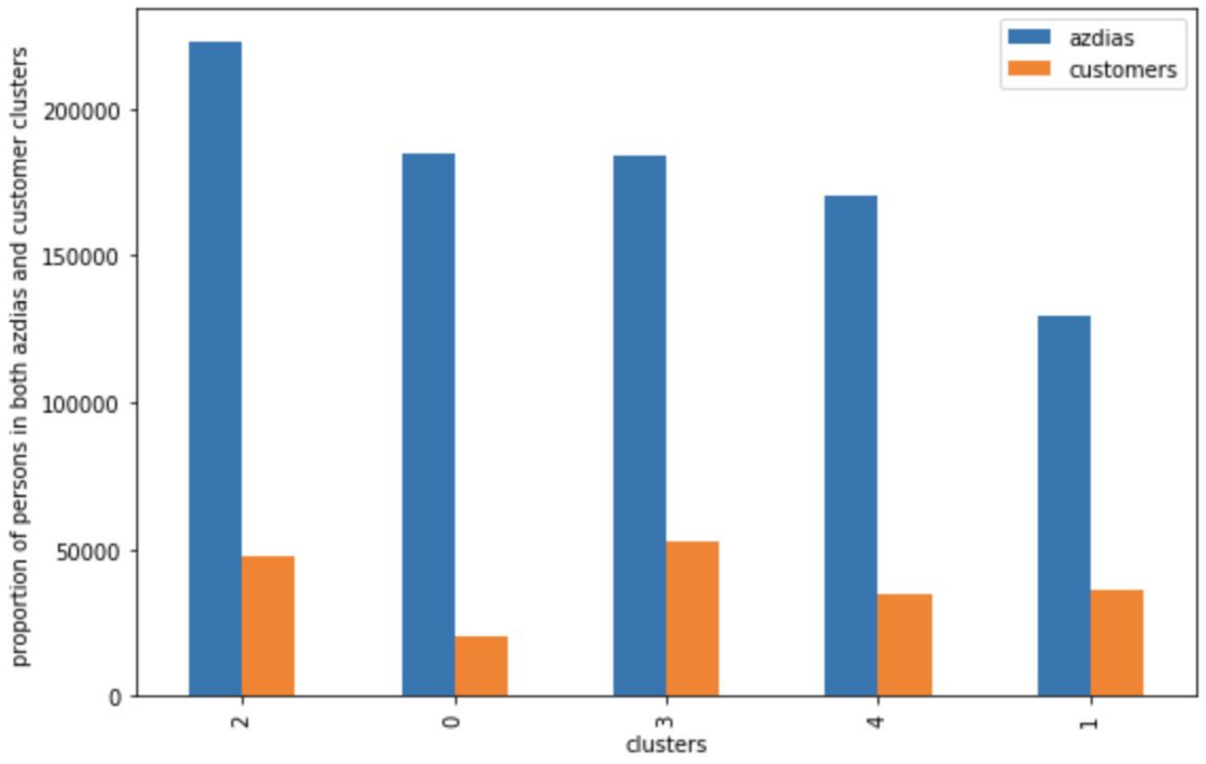
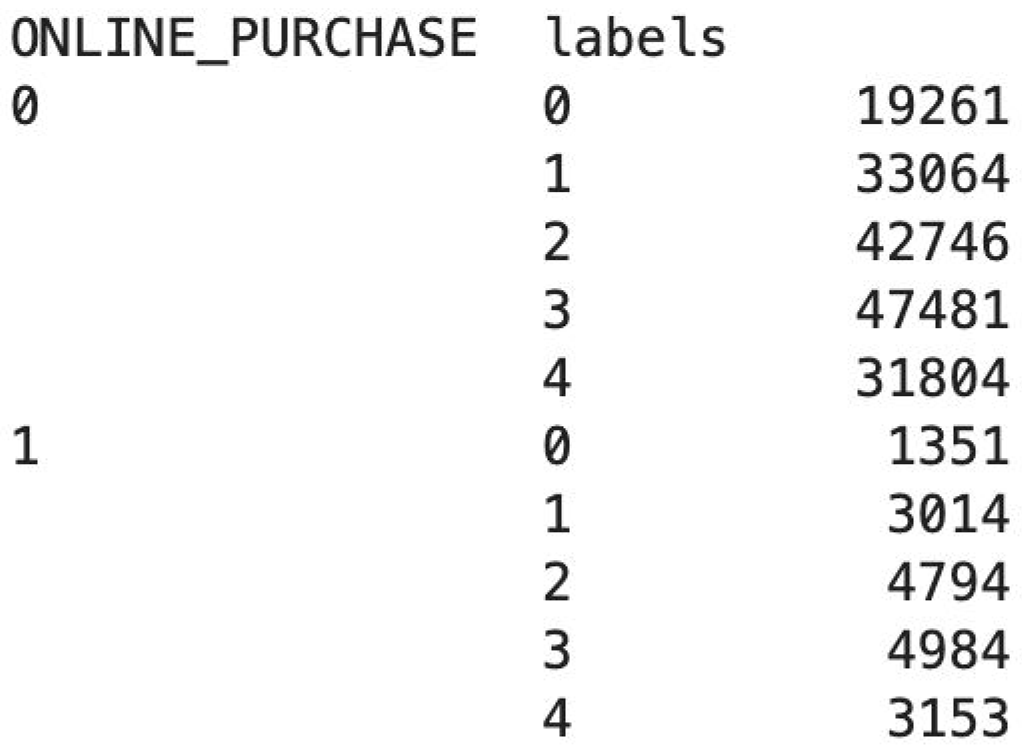


Fig. 10 Combined chart showing the cluster distribution for both Azdias and Customers Dataset

CLUSTER INFERENCE AND RECOMMENDATIONS

The initial customer data which has online purchase, customer group and product group was joined with the clusters; what this means is that we can do further analysis to know the most important clusters on the customers dataset which we can now be extended to Azdias in the event that there are marketing campaigns. The ultimate aim here is to be able to reach the right audience with the right product offering through the right channel.

ONLINE PURCHASE

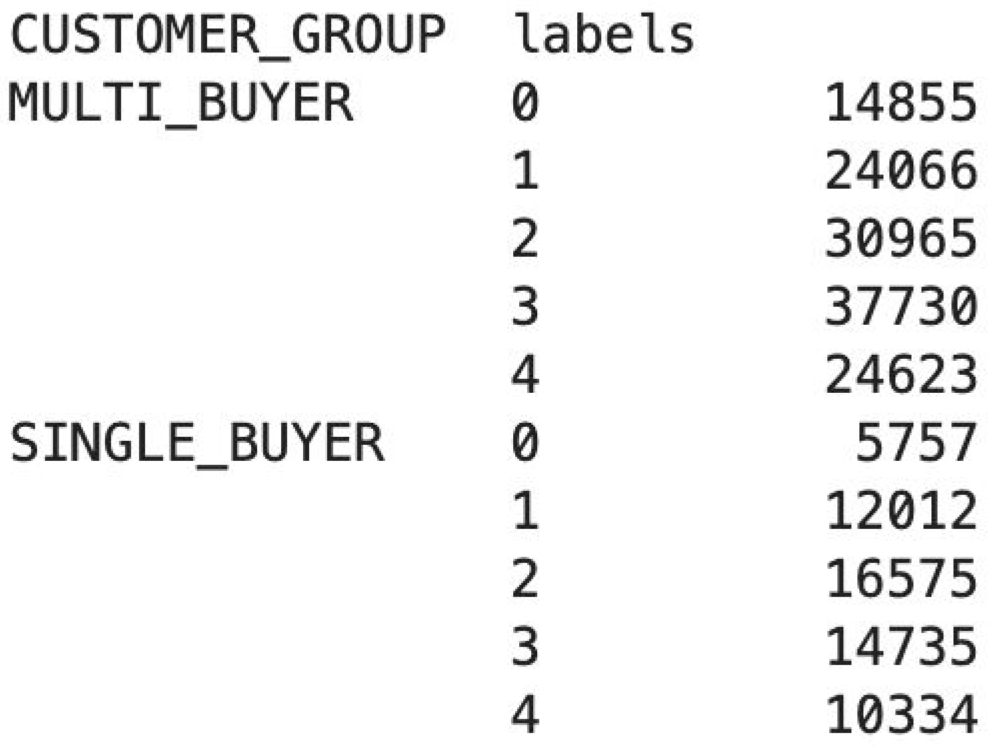


```
ONLINE_PURCHASE  labels
0                0         19261
                 1         33064
                 2         42746
                 3         47481
                 4         31804
1                0          1351
                 1          3014
                 2          4794
                 3          4984
                 4          3153
```

Fig. 11 Distribution of online/offline purchasers across the Customers’ clusters

From Fig. 11 above, it appears more clusters are in 0(offline purchase). If the target is to acquire new users from the Azdias dataset that are offline, it will be logical to go with clusters 2 and 3 (depending on the bucket size). However, if the focus is on online purchase, it is advisable to focus more on clusters 2 and 3.

CUSTOMER GROUP



```
CUSTOMER_GROUP  labels
MULTI_BUYER     0          14855
                1          24066
                2          30965
                3          37730
                4          24623
SINGLE_BUYER    0           5757
                1          12012
                2          16575
                3          14735
                4          10334
```

Fig. 12 Distribution of Customer Groups across the clusters

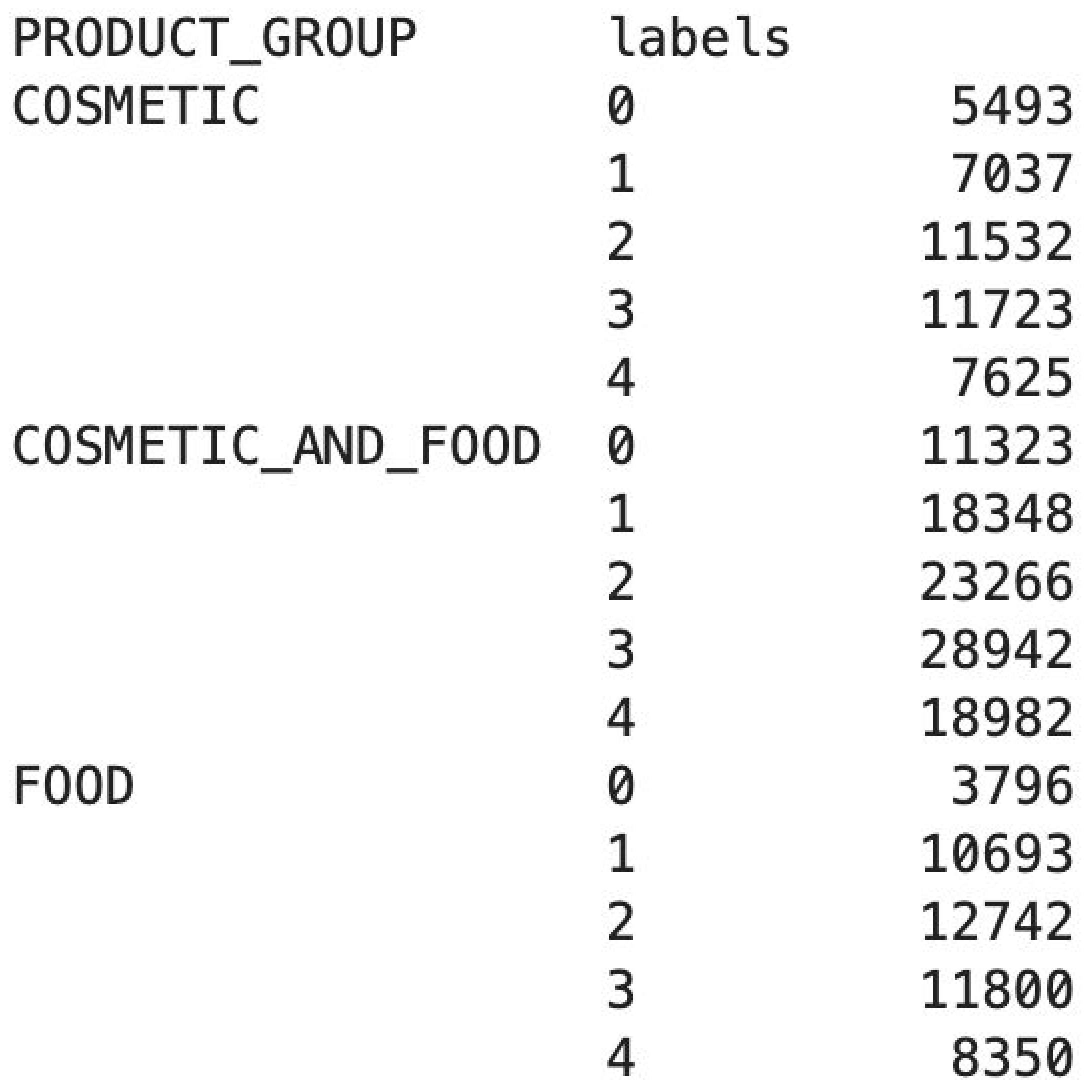
From the above, to place emphasis on either a single buyer or a multiple buyer, the cluster with the most frequent occurences will take precedence.

- Single buyer : consider clusters 2, 3,1 in this order
- Multi buyer : consider clusters 3, 2, 4 in this order

PRODUCT GROUP

From Fig. 13 below, the product group has three major categories: cosmetic, cosmetic and food, and food. In order to focus marketing campaigns on the type of product, it will be advisable to consider the following clusters per product offering.

- Cosmetic: consider clusters 3 and 2 in this order
- Cosmetic and food: consider clusters 3 and 2 in this order as well
- Food: consider cluster 2, 3 and 1 in this order



```
PRODUCT_GROUP      labels
COSMETIC           0          5493
                   1          7037
                   2         11532
                   3         11723
                   4          7625
COSMETIC_AND_FOOD  0         11323
                   1         18348
                   2         23266
                   3         28942
                   4         18982
FOOD               0          3796
                   1         10693
                   2         12742
                   3         11800
                   4          8350
```

Fig. 13 Distribution of the Product Group across the five clusters

SUPERVISED LEARNING APPROACH

To train a supervised learning algorithm, two things are paramount;

1. The target variable ('RESPONSE' in this case)
2. The features to be used for training

Using the train dataset provided which has the extra field 'RESPONSE', we try a host of supervised machine learning algorithms or deep learning algorithms to determine which will perform best on the test dataset. A train/test split is done on the "Udacity_MAILOUT_052018_TRAIN.csv" dataset to have a validation dataset. The performance of these models are monitored and the best model is deployed. Here are the steps I took in this stage:

1. Determined the ratio of response to non response which was about 1.2% to 98.8% respectively
2. Split the data immediately to avoid data leakage by stratifying the data using the response column. The new train data was 80% and the validation data was 20%
3. Applied the clean data class developed during the unsupervised learning approach to do the following : handle mixed data type, handle unknown, drop NaN values based on a certain threshold, data engineering (creating new attributes out of thee existing ones) and finally, handling NaN values
4. Converted the remaining categorical column to numerical columns (using pd.get_dummies)
5. Scaled the entire dataset using MinMaxScaler()
6. Oversampling of the minority class using SMOTE algorithm
7. Undersampling of the majority class using RandomUnderSampler
8. Applied the same steps from 3 to 5 above for the validation and test dataset
9. Built the model trying a range of algorithms but eventually settled for CatBoost and Extreme Gradient Boosting algorithms. The ROC_AUC metric was used

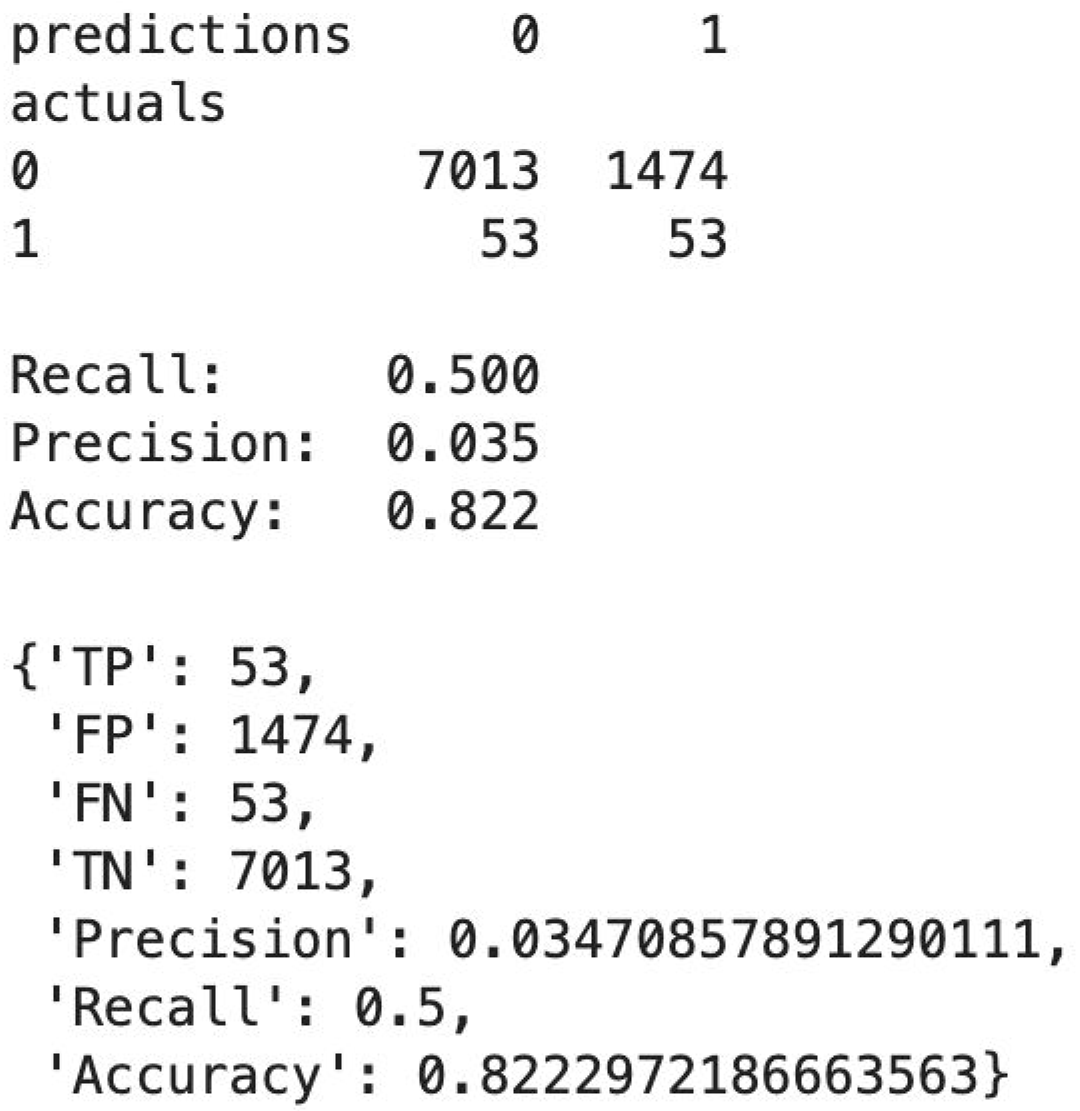
MODELING, EVALUATION, FEATURE IMPORTANCE

The approach here is to try a couple of machine learning algorithms with different data candidates. The data candidates are:

1. Scaled data without any form of sampling
2. Scaled data with oversampling of the minority class (where responses = 1 are fewer)
3. Scaled data with undersampling of the majority class (where responses = 0 are dominant)

Combining these candidates with the Catboost algorithm, here are the results:

- Candidate 1 (Catboost + scaled data + hyperparameter tuning): this gave a pretty decent performance with the classifier general performing well for the majority class (response = 0) and averagely well for the minority class (response = 1)



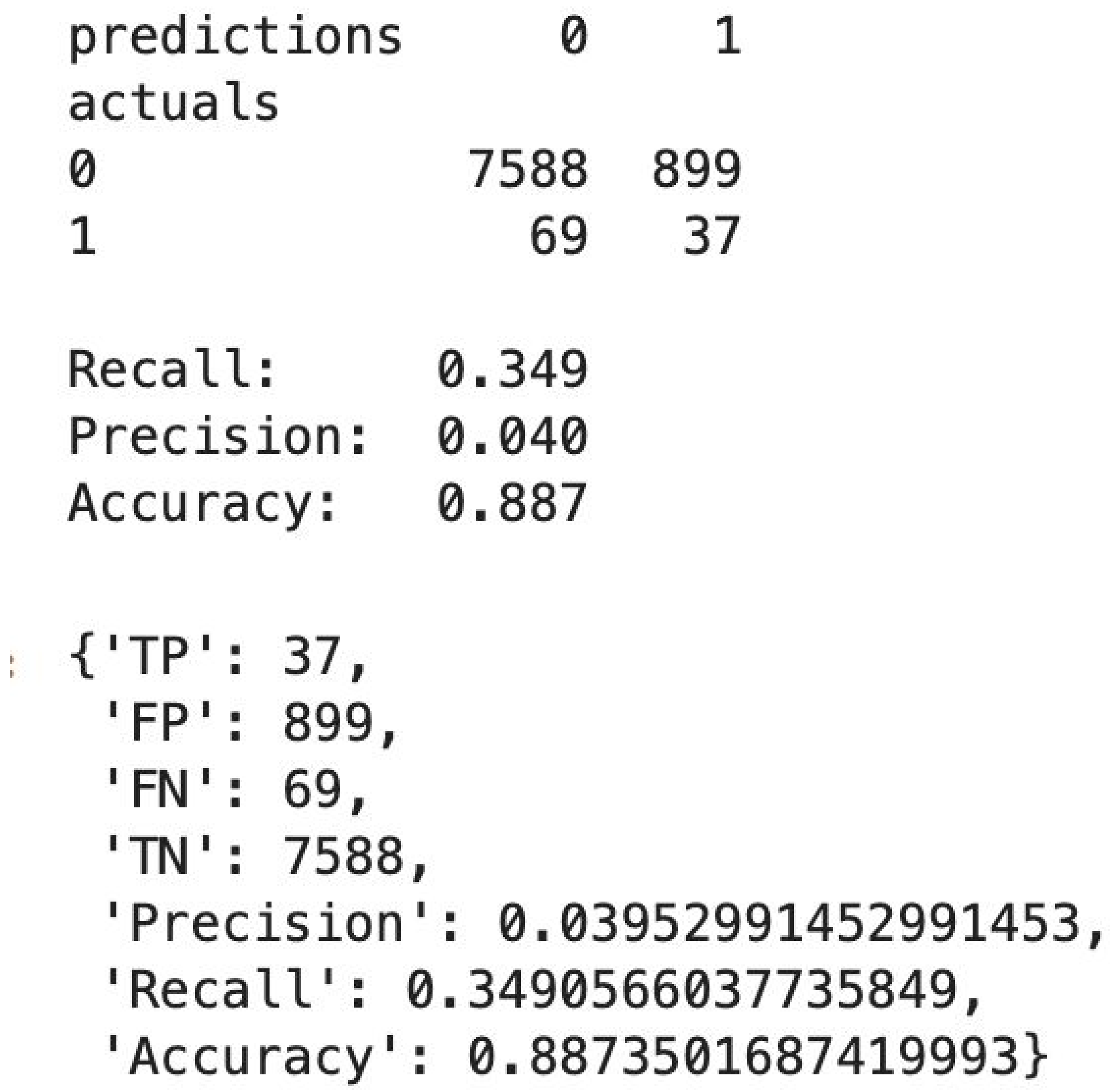
```
predictions     0     1
actuals
0            7013  1474
1              53    53

Recall:     0.500
Precision:  0.035
Accuracy:   0.822


{'TP': 53,
 'FP': 1474,
 'FN': 53,
 'TN': 7013,
 'Precision': 0.03470857891290111,
 'Recall': 0.5,
 'Accuracy': 0.8222972186663563}
```

Fig. 14 Performance of the Catboost algorithm on the validation dataset

- Candidate 2(CatBoost + hyperparameter tuning + Undersampling of majority class): from Fig. 15 below, the classifier performed better predicting the majority class (response = 0) and performed worse predicting the minority class (response = 1)



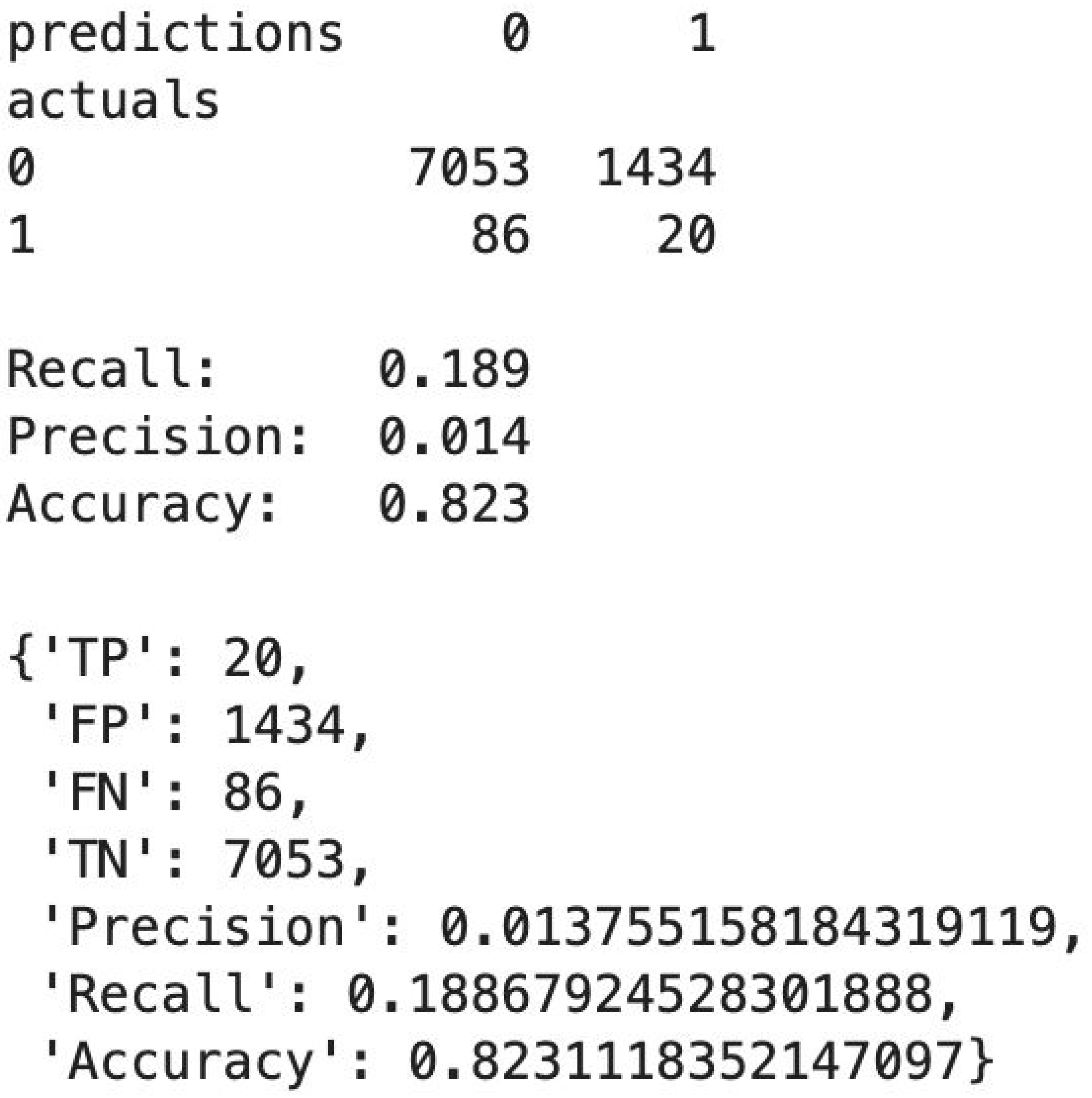
```
predictions     0    1
actuals
0            7588  899
1              69   37

Recall:     0.349
Precision:  0.040
Accuracy:   0.887


{'TP': 37,
 'FP': 899,
 'FN': 69,
 'TN': 7588,
 'Precision': 0.03952991452991453,
 'Recall': 0.3490566037735849,
 'Accuracy': 0.8873501687419993}
```

Fig. 15 Performance of the classifier on the undersampled data

- Candidate 3 (Catboost + oversampling + hyperparameter tuning): Fig. 16 below shows that the classifier performed decently well for the majority class (response = 0) but really worse for the minority class (response = 1)



```
predictions     0     1
actuals
0            7053  1434
1              86    20

Recall:     0.189
Precision:  0.014
Accuracy:   0.823

{'TP': 20,
 'FP': 1434,
 'FN': 86,
 'TN': 7053,
 'Precision': 0.013755158184319119,
 'Recall': 0.18867924528301888,
 'Accuracy': 0.8231118352147097}
```

Fig. 16 Performance of the Classifier on the oversampled data

From the 3 candidates above, it is obvious that candidate 1 performed best for the CatBoost algorithm and we would be exploring the feature importance using SHAP values.

FEATURE IMPORTANCE

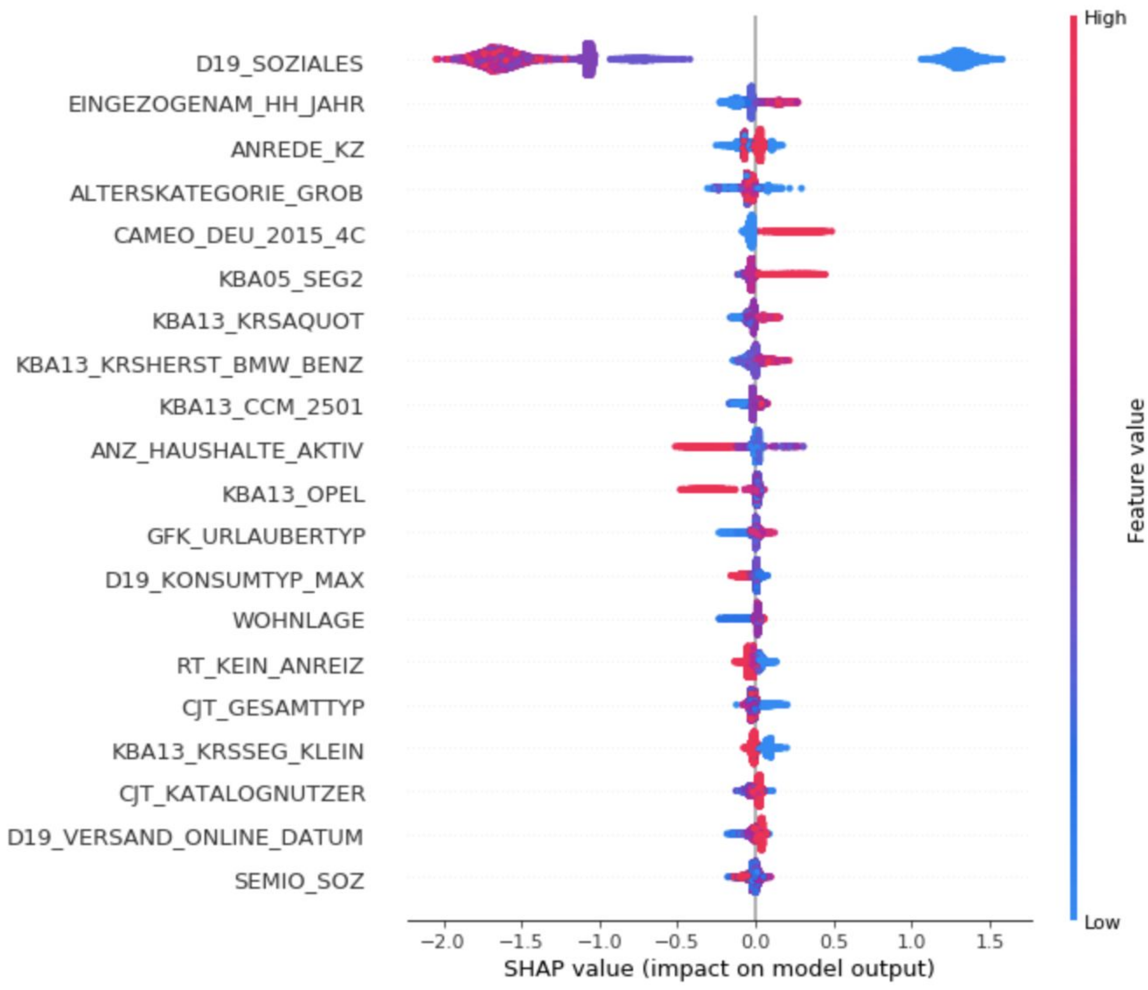
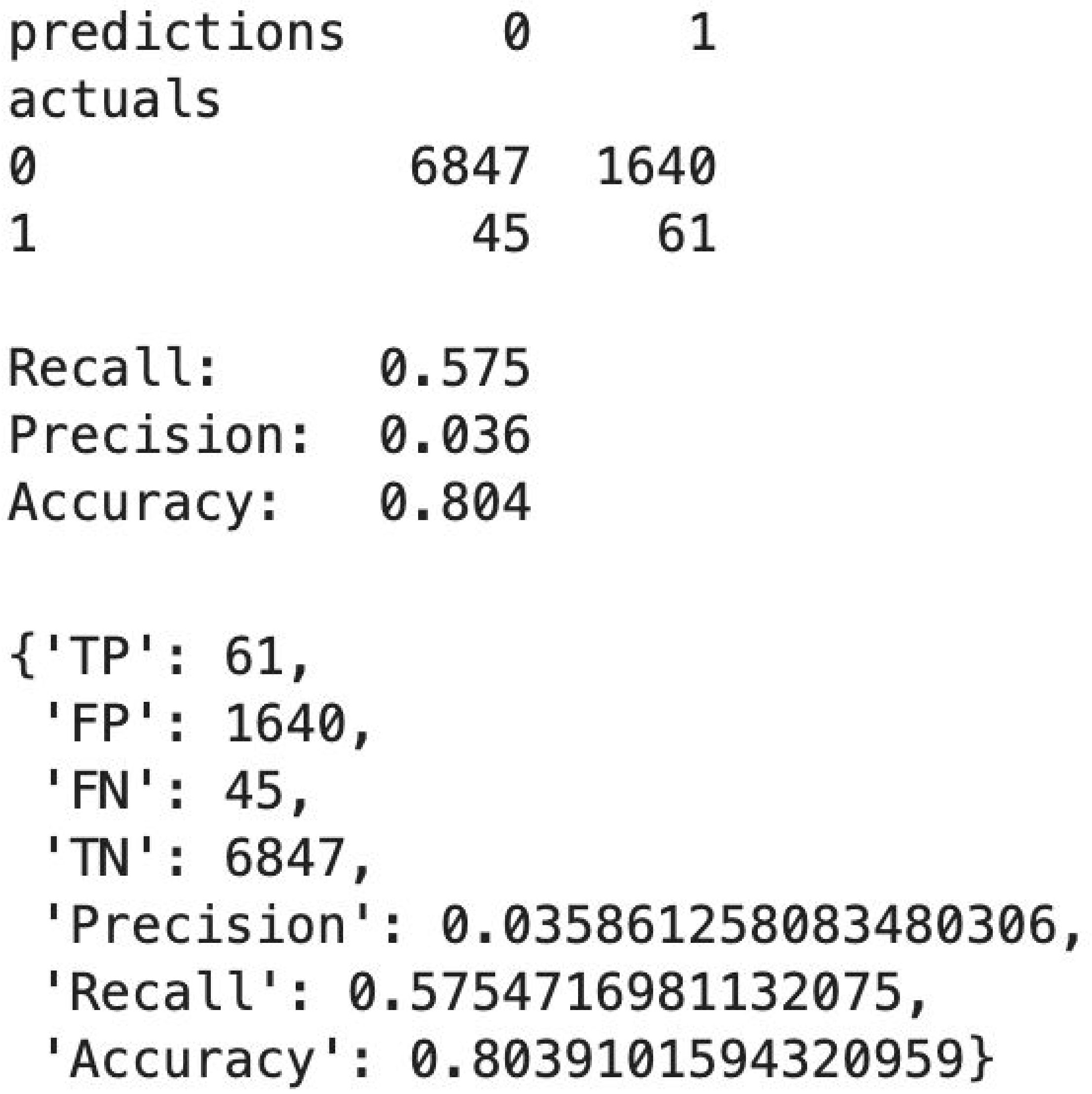


Fig. 17 The Feature importance of the best performing CatBoost Algorithm (Candidate 1)

EXTREME GRADIENT BOOSTING ALGORITHM

The steps here will basically be a repetition of the candidate trials for the XGBoost algorithm. The best candidate + algorithm will be selected.

- Candidate 1 (XGBoost + scaled data + hyperparameter tuning): Fig.18 below indicates that classifier performed okay with the majority class (response =0) and performed better predicting the minority class (response = 1)



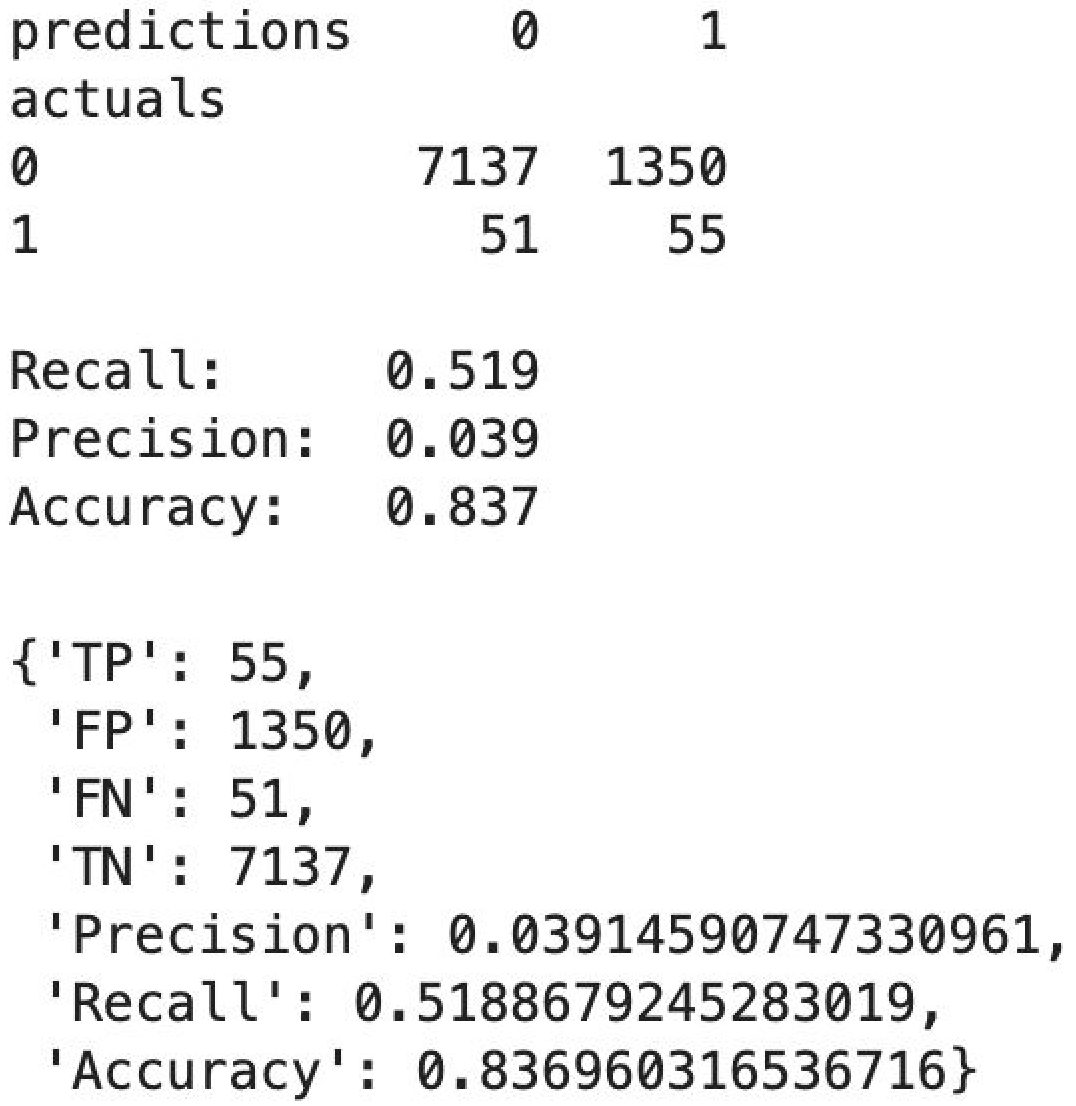
```
predictions     0     1
actuals
0            6847  1640
1              45    61

Recall:     0.575
Precision:  0.036
Accuracy:   0.804


{'TP': 61,
 'FP': 1640,
 'FN': 45,
 'TN': 6847,
 'Precision': 0.035861258083480306,
 'Recall': 0.5754716981132075,
 'Accuracy': 0.8039101594320959}
```

Fig. 18 XGBoost Classifier with scaled data

- Candidate 2 (XGBoost + undersampled data + hyperparameter tuning): from Fig.19 below, this appears to be the best performing model so far. Performs very well on classifying the majority class and also performs decently well classifying the minority class as well (response = 1)



```
predictions     0     1
actuals
0            7137  1350
1              51    55

Recall:     0.519
Precision:  0.039
Accuracy:   0.837


{'TP': 55,
 'FP': 1350,
 'FN': 51,
 'TN': 7137,
 'Precision': 0.03914590747330961,
 'Recall': 0.5188679245283019,
 'Accuracy': 0.836960316536716}
```

Fig. 19 XGBoost Classifer with undersampled majority class

GRID SEARCH

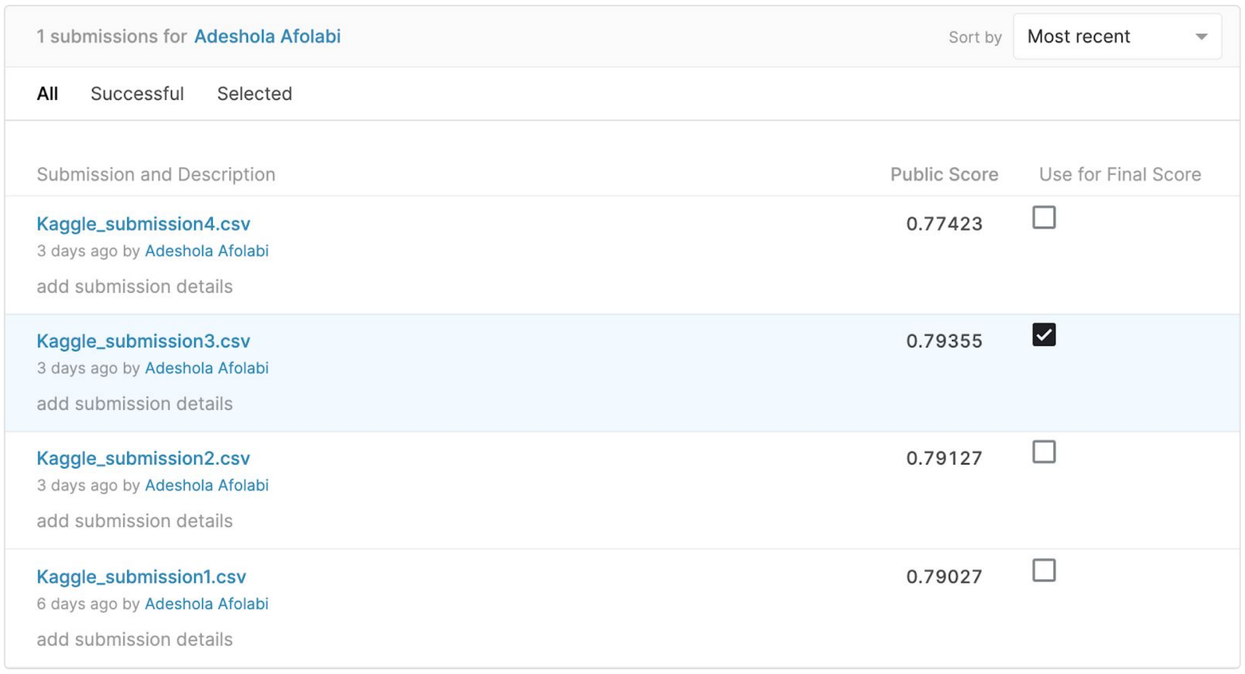
In order to identify the best hyperparameters to use, I tried a grid search algorithm which took so long. The essence of this was to get the best algorithm and use the hyperparameters from the model to train or further improve the classifier.

USE OF THE ENTIRE DATASET

Upon identifying the best model to use, I decided to not split my data into train and validation set anymore. I fit the entire dataset on the classifier in order to have an improved performance since the classifier will be exposed to more dataset to train with. This led to a slight improvement of about 0.23%.

KAGGLE COMPETITION AND RESULT

After finalizing on the best model to use, I was able to predict the probability values using the test dataset provided. The result was saved as a DataFrame with two comuns (LNR and RESPONSE). This was uploaded on Kaggle and I achieved a score of 79.355% after four submissions. This is within range of my acceptance threshold since my aim was to achieve + or - 5% of the leaderboard. The winner on the leaderboard has a score of 84.739% and the next score to the leader was 81.063%.



1 submissions for Adeshola Afolabi

Sort by: Most recent

All | Successful | Selected

Submission and Description	Public Score	Use for Final Score
Kaggle_submission4.csv 3 days ago by Adeshola Afolabi add submission details	0.77423	☐
Kaggle_submission3.csv 3 days ago by Adeshola Afolabi add submission details	0.79355	☑
Kaggle_submission2.csv 3 days ago by Adeshola Afolabi add submission details	0.79127	☐
Kaggle_submission1.csv 6 days ago by Adeshola Afolabi add submission details	0.79027	☐

Fig. 20 Final Kaggles Submission

CONCLUSION

The mail order company can rely on the output of the analysis and predictions to do two major things;

1. Identify segments within the customer dataset that can be used for the Azdias dataset which will help with marketing strategies. For example, if I want to acquire new users

that will shop online based on the present COVID situation, I should focus more on clusters 2 and 3 of the Azdias dataset

2. In order to send mails of new product offerings, a ranked probabilities of users in descending order can be used to target people with the highest propensity to respond. E.g if the sample size of mails to be sent out is 10 000 and the top users that fall within this bucket have probability of 80% and above, they should be targeted.

REFLECTION

The entire project can be summarized as follows:

- Understanding of the attributes provided in the dataset
- Writing a generic script that will be reusable for cleaning, exploration and engineering
- Dimensionality reduction using principal component analysis
- Use of Sagemaker KMeans as the unsupervised learning approach to identify clusters within the Azdias and Customers' dataset
- Building a supervised learning algorithm to determine users that will respond to campaigns from the mail order company

CHALLENGES

Some of my challenges were:

- Understanding the attributes of the dataset took a very long time
- Writing the generic script that will be reusable was quite hectic
- Working with Sagemaker endpoints with requests greater than 4000 made me do a lot of research. I could not find a solution and I had to chunk the requests
- Stoping and starting training and deployment jobs for any modification

IMPROVEMENT

For the supervised learning approach, some major improvements that can be made are:

- Better data engineering: I noticed most numerical columns were not well ordered. An example would be poor households having a value of 7 and rich households having a value of 1. Since models rely on weights, it would be interpreted that 1 > 7 and the model might find it difficult to train. Due to the inconsistency and the number of attributes, I could not write a script to fix this
- Data Point Dictionary Provided: some features that were present in the data did not have any explanation in the excel files provided. It was difficult to interpret what these fields meant.