Capstone Project

Yau Chi Pui, Chris

Machine Learning Engineer Nanodegree

9th February, 2020

Customer Segmentation Report for Arvato Financial Services

Definition

Project Overview

Nowadays, almost every company wants to find out their potential customers in order to convert them to the company's customers efficiently. Especially for some mail-order sales companies. The reason is that most people do not respond to the mail, identifying which groups of population have the greatest potential to become company's customers is essential. It can help those companies to allocate resources and boost the sales significantly.

As there are some data provided by Bertelsmann Arvato Analytics, about the demographics data for customers of a mail-order sales company in Germany, demographics information for the general population, and demographics data for individuals who were targets of a marketing campaign. We can use these data to build the models and make predictions which are also applicable to other companies.

Problem Statement

As the general population is tremendous, and most of the individuals do not respond to the mail-sales, it is a challenge for those mail-order sales companies to target potential customers effectively. For instance, only 532 individuals responded to the mail over 42,962 individuals according to the data set provided, which is about only 1.24%. Almost all mail-order sales companies are facing the same problems.

We will first extract useful features from the first two datasets by using Principal Component Analysis (PCA), which can retain the 'principal components' of the features. Next, we will use these features to train a k-means model that segment the general population to different groups and find out which parts of the population are more likely to be part of the mail-order company's main customer base. Last, we will use the 'TRAIN' subset to train a supervised model. We will use both LinearLearner and XGBoost model this time, to predict the 'TEST' subset. This solution is applicable to other companies in this domain if they have enough data.

Metrics

We will first use F1 score for Linear Learner and AUC score for XGBoost as there is a large output class imbalance. We cannot simply use accuracy to evaluate the model. In order to maxmize recall, precision and accuracy score at the same time, F1 and AUC score are two of the most appropriate scores to evaluate the model. Then, we will upload the result to the Kaggle competition. It will use accuracy to evaluate the model while improving accuracy is our final goal.

Analysis

Data Exploration

The datasets used in this project are all provided by Bertelsmann Arvato Analytics, including the demographics data for customers of a mail-order sales company in Germany, demographics information for the general population, and demographics data for individuals who were targets of a marketing campaign.

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

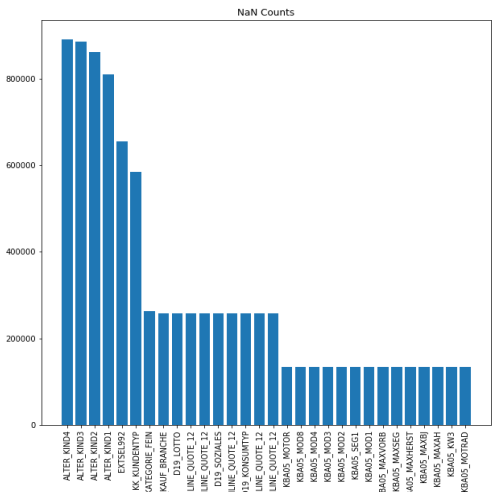
Each row of the demographics files represents a single person, but also includes information outside of individuals, including information about their household, building, and neighborhood. The "CUSTOMERS" file contains three extra columns ('CUSTOMER_GROUP', 'ONLINE_PURCHASE', and 'PRODUCT_GROUP'), which provide broad information about the customers depicted in the file. The original "MAILOUT" file included one additional column, "RESPONSE", which indicated whether or not each recipient became a customer of the company. For the "TRAIN" subset, this column has been retained, but in the "TEST" subset it has been removed.

Data Visualizations

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Columns: 366 entries, LNR to ALTERSKATEGORIE_GROB
dtypes: float64(267), int64(93), object(6)
memory usage: 2.4+ GB
```

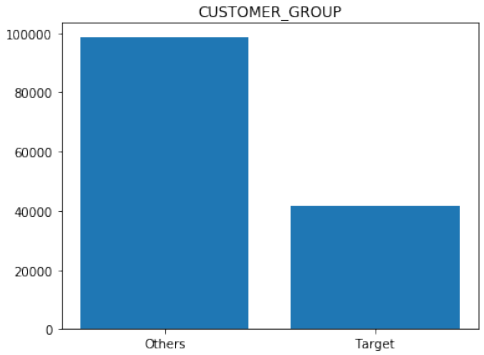
	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALT
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	
3	910226	2	1.0	13.0	NaN	NaN	NaN			
4	910241	-1	1.0	20.0	NaN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

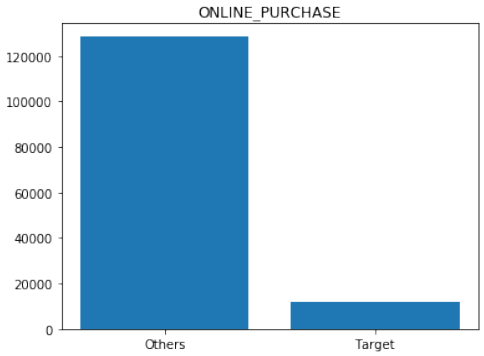
5 rows × 366 columns



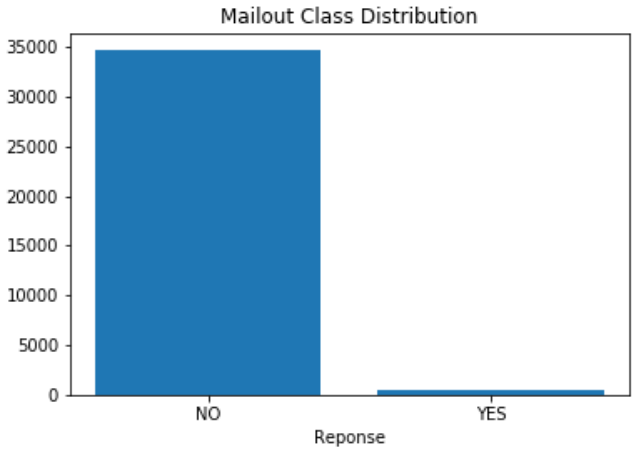
The above figure is a brief dataframe showing some sample rows and information of the datasets, which can give us some understanding on characteristic of the datasets. And the right hand side figure tells us the features which have most NaN values. As we have up to 366 features while some of the features have very high NaN quantities, it is preferred to drop some of the features and then use PCA to reduce the dimensions. It is very helpful when using k-means to cluster the population.

These two figures shows the ‘CUSTOMER_GROUP’ and ‘ONLINE_PURCHASE’ in customers.csv. These two columns are some of the important information that we are going to use to analyze the core group of the company. The reason is that in the ‘CUSTOMER_GROUP’, the targeted customers are all multi buyers, while the ‘ONLINE_PURCHASE’ means they have online purchased before.





From the below figure, we can clearly see that the mailout_train.csv has imbalance class distribution. Model performance would be significantly affected if we use accuracy to evaluate as usual. Therefore, we are going to use F1 score and AUC score to evaluate the models. These scores perform well in this situation.



Implement

Algorithms and Metrics

The algorithms we will use are Principal Component Analysis (PCA), K-Means, Linear Learner and XGBoost. After preprocessing the data, we will use PCA to reduce the dimensions and extract useful features as it can make K-Means to cluster the population much more easier. We can tune the explained variance to extract different number of components. Then, we will use K-Means to

cluster the general population. We need to first determine the k, which is the number of clusters we are going to use. After clustering the general population, we can fit the customers.csv to the trained K-Means model. So that we can divide the customers into different groups. Last, we will use Linear Learner and XGBoost on mailout dataset separately. We choose to use binary classifier for the Linear Learner and binary:logistic for the XGBoost. We use two different algorithms because we want to find out which algorithms are better in this case.

Metrics
As we said before, we will use F1 score for Linear Learner and AUC score for XGBoost as there is a large output class imbalance. We cannot simply use accuracy to evaluate the model. In order to maxmize recall, precision and accuracy score at the same time, F1 and AUC score are two of the most appropriate scores to evaluate the model. Then, we will upload the result to the Kaggle competition. It will use accuracy to evaluate the model while improving accuracy is our final goal.

Benchmark
There is a Kaggle competition that is exactly doing the same thing. So, the domain, problem statement, and intended solution are all related to our project. We will use the Kaggle competition to test our model's accuracy first. Then, we will use the scoreboard of the Kaggle top performers as a benchmark. The top Kaggle performers have approximately 0.8 scores. Therefore, our model should at least have similar score.

Methodology

Preprocessing
Most of the preprocessing steps are done in 1. Data Exploration an Cleaning.ipynb and 3.1 Predictor Data Peperation.ipynb. The process are almost the same, they are just handling different datasets. The former is handling azdias.csv and customers.csv while the later one is handling mailout_train.csv. The steps are as below:

1. Convert all the category columns to numeric columns
2. Drop useless columns
3. Drop columns that have high NaN proportions
4. Drop rows that have high NaN proportions
5. Fill the rest NaN value with 0
6. Use MinMaxScaler to normalize all the data between 0 and 1

	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	ANZ_HH_TITEL	ANZ_KINDER	ANZ_PERSONEN	ANZ_STATISTISCHE_HAUSHALTE	ANZ_TITEL
count	35100.000000	35100.000000	35100.000000	35100.000000	35100.000000	35100.000000	35100.000000	35100.000000	35100.000000	35100.000000
mean	0.458697	0.065050	0.490532	0.381556	0.015311	0.002444	0.014872	0.084171	0.017013	0.004772
std	0.337360	0.216828	0.289558	0.185667	0.034623	0.018902	0.066051	0.056698	0.038857	0.052012
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.380952	0.320000	0.002283	0.000000	0.000000	0.041667	0.002710	0.000000
50%	0.500000	0.000000	0.476190	0.400000	0.004566	0.000000	0.000000	0.083333	0.005420	0.000000
75%	0.750000	0.000000	0.714286	0.480000	0.015982	0.000000	0.000000	0.125000	0.018970	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Example DataFrame after 6 steps of preprocessing

We also have some preprocessing steps in 3.2 Building Predictor.ipynb right before building the models. The steps are as follow:

1. Separate the training data based on the labels
2. Separate the label column from the features
3. Split 30% of the data to validation set
4. Concatenate the label back to the features in first column as required by AWS
5. Shuffle the data

Refinement

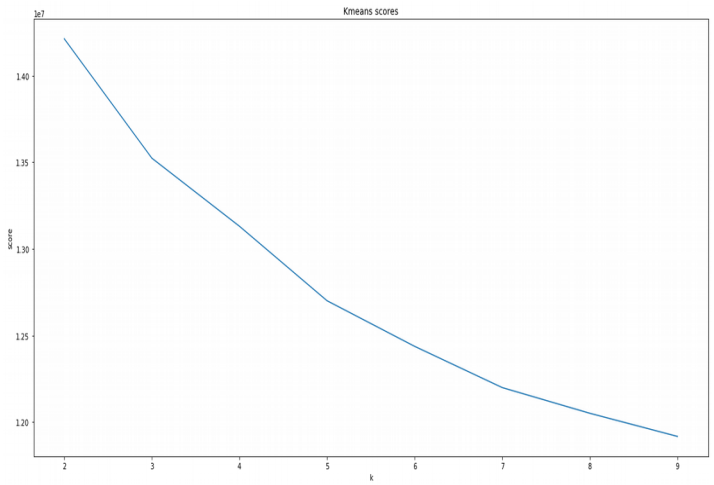
For the PCA, we define a function to calculate the number of components to be used based on the explained variance we pick. A higher explained variance means more features are going to be kept. So that the dataset can keep more information. The downside is more features means more difficult for K-Means to cluster the population. In this case, we will pick 142 components as we want to reach 90% of explained variance.

```
#define a function to calculate the number of components
def calculate_components(comp_variance, variance=0.9):
    '''
    This function is used to calculate the number of components,
    giving a desired total explained variance
    input: data explained variance ratio, expected variance
    output: number of components
    '''

    for i in range(comp_variance.shape[0]):

        explained_variance = comp_variance[:i+1].sum()

        if explained_variance > variance:
            break

    print('num_of_components: {} explained_variance: {}'.format(i+1, explained_variance))
    return i+1
```

```
#calculate the number of components
num_of_components = calculate_components(azdias_fit_pca.explained_variance_ratio_)
```

```
num_of_components: 142 explained_variance: 0.900449999183804
```

After deciding how many features we are going to use, we need to use K-Means to cluster the general population. For K-Means, we can adjust the number of k based on different needs. In general, we can calculate the score and then use elbow method to decide k. However, in this case, elbow method seems useless as there are no obvious ‘elbow’. To decide the k, we will choose 5. As too few groups is hard to target while too many groups is difficult to find the characteristics.



For the last part, which is using Linear Learner and XGBoost, as we are going to use AWS for cloud computing, we will use hypyerparameter tuner to decide most of the parameters automatically. For the Linear Learner, we will let the tuner to decide learning rate and l1 regularization value. For the XGBoost, it will automatically choose alpha, maximum depth, eta, gamma, minimum child weight, subsample and lambda.

```
Linear.set_hyperparameters(feature_dim=358,
                           predictor_type='binary_classifier',
                           binary_classifier_model_selection_criteria='f_beta',
                           learning_rate=0.0001,
                           use_bias=True,
                           l1=0,
                           positive_example_weight_mult='balanced',
                           epochs=100)
```

```
#build a hyperparameter tuner
Linear_hyperparameter_tuner = HyperparameterTuner(estimator=Linear,
                                                  objective_metric_name='validation:binary_f_beta',
                                                  objective_type='Maximize',
                                                  max_jobs=30,
                                                  max_parallel_jobs=3,
                                                  hyperparameter_ranges={
                                                    'learning_rate': ContinuousParameter(0.0001, 0.01),
                                                    'l1': ContinuousParameter(0.0001, 0.01),
                                                   })
```

For the Linear Learner, we need to define bias, predictor type, positive example weight and epochs by hand. As these parameters need not to be changed or optimized, we do not need any automation. XGBoost is similar, we will decide positive weight, number of rounds, objective, etc. by ourselves.

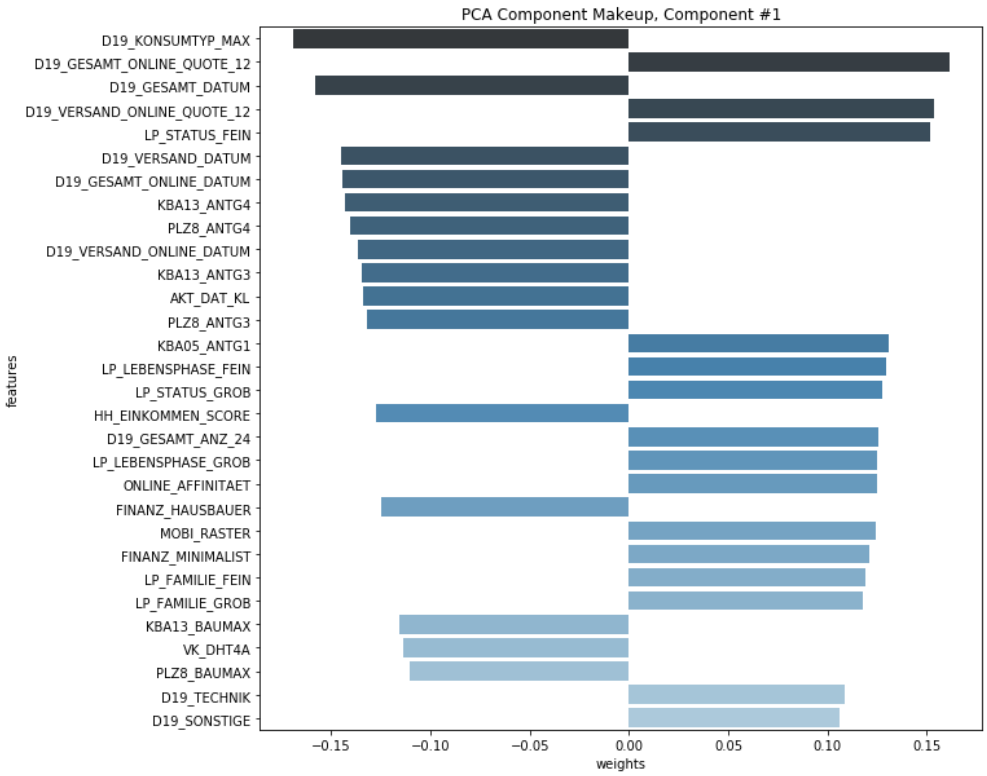
```
xgb.set_hyperparameters(alpha=1,
                        max_depth=5,
                        eta=0.2,
                        gamma=3,
                        min_child_weight=5,
                        subsample=0.7,
                        objective='binary:logistic',
                        early_stopping_rounds=100,
                        num_round=4000,
                        eval_metric='auc',
                        scale_pos_weight=80)
```

```
WARNING:root:There is a more up to date SageMaker XGBoost image. To use the newer image, please s
       get_image_uri(region, 'xgboost', '0.90-1').
```

```
#build a hyperparameter tuner
xgb_hyperparameter_tuner = HyperparameterTuner(estimator=xgb,
                                               objective_metric_name='validation:auc',
                                               objective_type='Maximize',
                                               max_jobs=20,
                                               max_parallel_jobs=3,
                                               hyperparameter_ranges={
                                                'alpha': ContinuousParameter(1, 100),
                                                'max_depth': IntegerParameter(1, 9),
                                                'eta': ContinuousParameter(0.1, 0.5),
                                                'gamma': ContinuousParameter(1, 4),
                                                'min_child_weight': ContinuousParameter(2, 100),
                                                'subsample': ContinuousParameter(0.6, 0.9),
                                                'lambda': ContinuousParameter(2, 500)
                                                })
```

Postprocessing

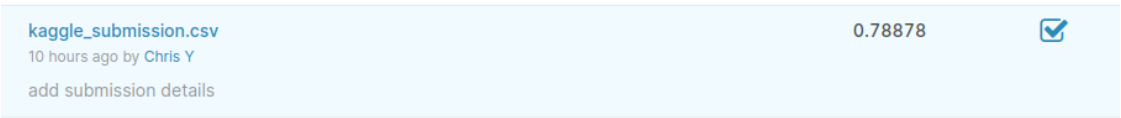
We need to postprocessing the result of the K-Means clustering as the result is very complex and difficult for human beings to interpret. After we fit the customers.csv to the trained K-Means model, we can find out which cluster is the core and which is our potential customers in general population. To interpret the cluster, we need to use a heatmap to visualize the attribute values of each cluster. And then use the trained PCA and a function to visualize the main features of that cluster. Here is an example:



Results

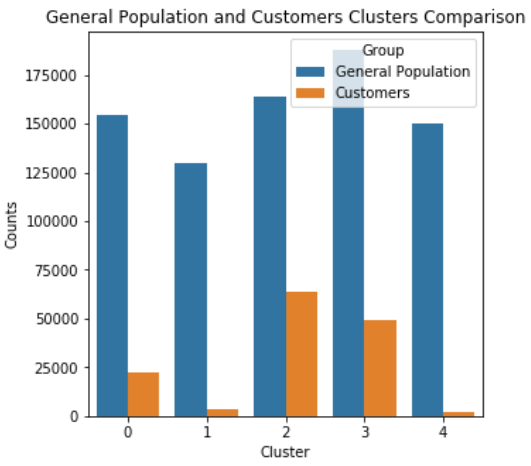
Models Performance

For the PCA and K-Means, the results are quite good. We can extract useful features, reduce dimensions and cluster population as expected. The results will be shown in next part. For the Linear Learner, as our benchmark is Kaggle's scoreboard, the score is not satisfactory, which is only 0.60273, compared to top performers, about 0.8. For the XGBoost, the score on Kaggle is 0.78878, which is very closed to the top performers.



Significant Quantities

After we fit the customers.csv to the trained K-Means model, we can plot a graph like this. It is obvious that Cluster 0 and Cluster 1 contain most of the customers. To decide which Cluster is the core, which is our target, we need to use 'ONLINE_PURCHASE', 'CUSTOMER_GROUP', and 'PRODUCT_GROUP' columns.



In the ‘CUSTOMER_GROUP’, 1.0 means multi buyer while 0.0 means single buyer. As a result, we can obviously see that Label 2, which is Cluster 2 contains most of the multi buyers. Therefore, Cluster 2 is the company’s core group and the general population who are labeled as Cluster 2 are target customers.

Label	CUSTOMER_GROUP	
0	1.0	16472
	0.0	6121
1	1.0	2013
	0.0	1238
2	1.0	47068
	0.0	16513
3	1.0	31839
	0.0	17281
4	1.0	1237
	0.0	632

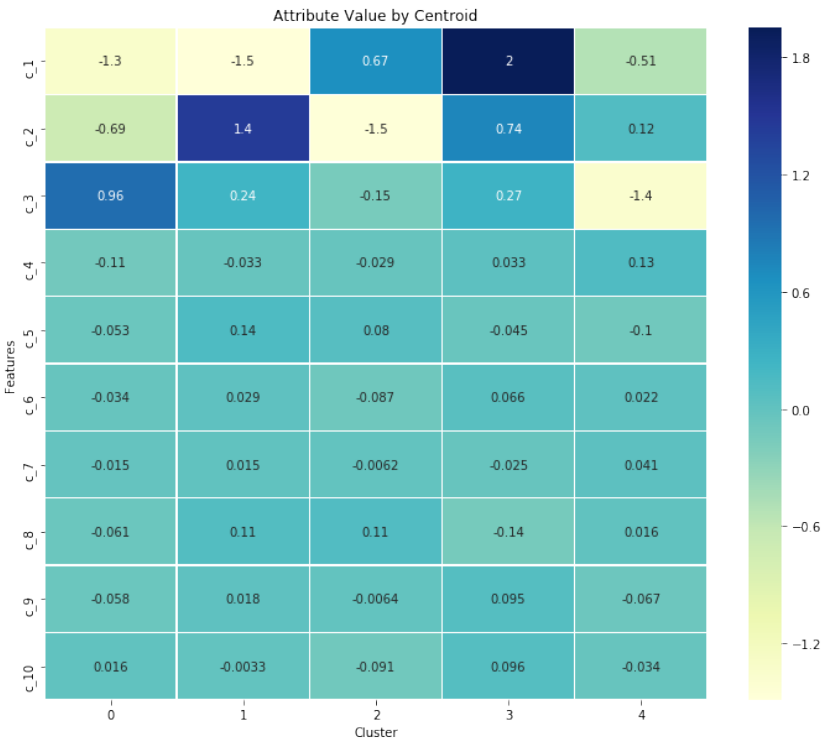
In the ‘ONLINE_PURCHASE’, if the company wants to target online market, they can target Cluster 3 as this group has a higher chance to purchase online. ‘PRODUCT_GROUP’ is used to decide the target customers based on the product types.

Label	ONLINE_PURCHASE	
0	0.0	21634
	1.0	959
1	0.0	2056
	1.0	1195
2	0.0	61076
	1.0	2505
3	0.0	41991
	1.0	7129
4	0.0	1529
	1.0	340

Name: ONLINE_PURCHASE, dtype: int64

Justification

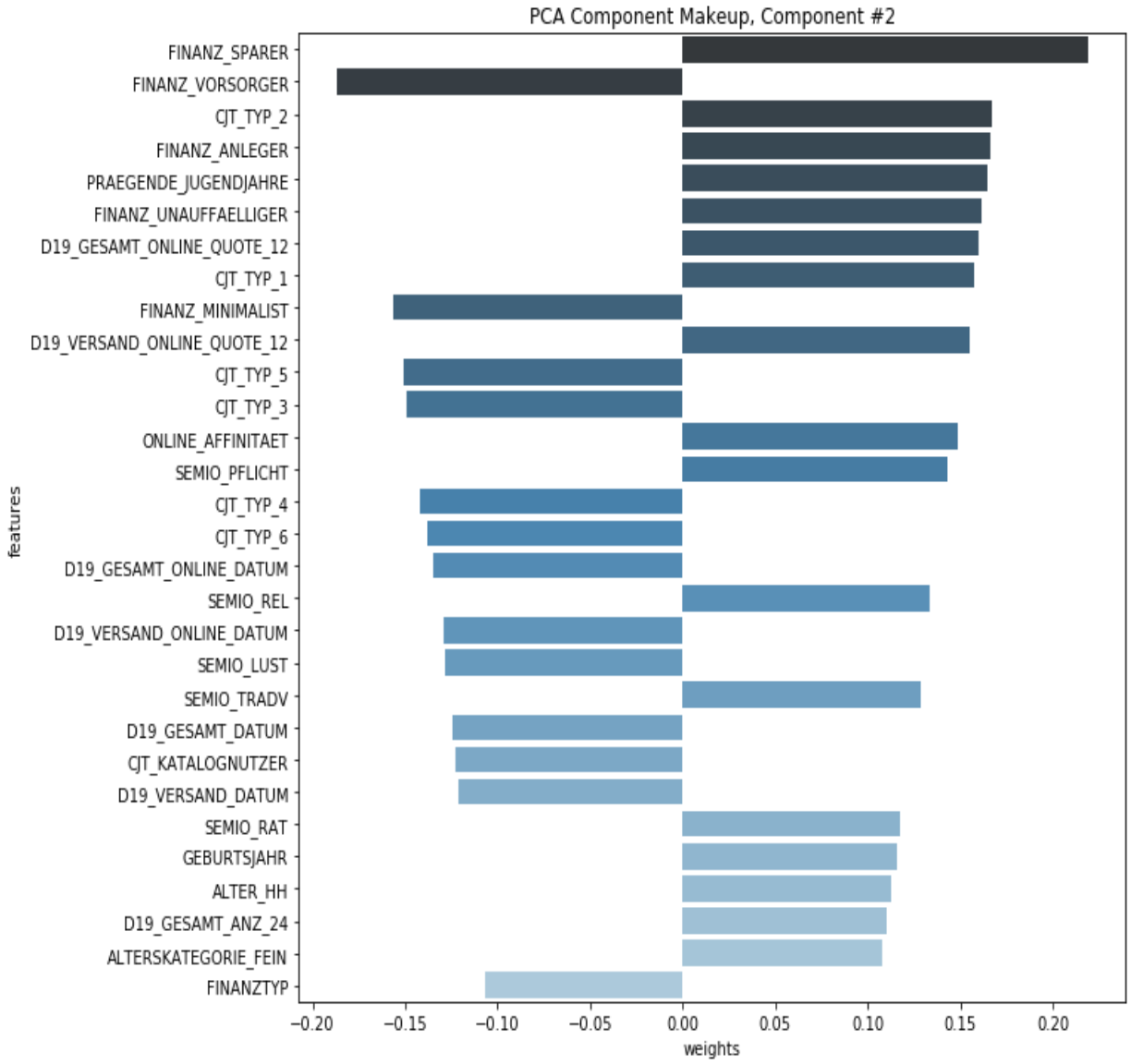
If we know the target cluster, we would also want to know the features of that group. We can do it buy plotting a heatmap and then interpret by a function. For instance, the target potential customers is in Cluster 2. From the heatmap we can see that c_2 has the largest weight in Cluster 2. Then we can use a custom function and PCA to display the features in c_2.



Here's how:

```
#display the weight of the components
def display_component(pca_components, features_list, component_num, n_weights=30):
    '''
    This function is used to display the weight of each features
    in a single particular component.
    input: data pca components, features list, # of particular component
    output: display a bar chart showing the weight distribution
    '''

    comp_row = pca_components[component_num-1]

    components = pd.DataFrame(list(zip(comp_row, features_list)),
                          columns=['weights', 'features'])

    components['abs_weights'] = components['weights'].apply(lambda x: np.abs(x))
    sorted_weight_data = components.sort_values('abs_weights', ascending=False).head(n_weights)

    ax=plt.subplots(figsize=(10,10))
    ax=sns.barplot(data=sorted_weight_data,
                   x="weights",
                   y="features",
                   palette="Blues_d")
    ax.set_title("PCA Component Makeup, Component #" + str(component_num))
    plt.show()
```

We can use the above function to display what are inside the c_2, component #2. From the figure below, it shows that 'FINANZ_SPARER', 'FINANZ_VORSORGER', 'CJT_TYP_2' are some of the most important features in component #2.



Conclusions

To conclude, we have used PCA and K-Means to cluster the general population in order to find the target customers based on different needs, such as online purchase, product type, etc. It is shown that these two models successfully do what are expected. We clustered the population into five groups, Cluster 2 has the highest probability to become multi buyer. And the main features of these customers are ‘FINANZ_SPARER’, ‘FINANZ_VORSORGER’, ‘CJT_TYP_2’. We also successfully to build a XGBoost to predict whether a customer would response to a mail sent by the company. We have a score of 0.788, which is very closed to the top performers in Kaggle scoreboard.