Capstone Project Report

UDACITY

Ricard Trinchet Arnejo

Machine Learning Engineer Nanodegree

Definition

Project Overview

This project corresponds to the Machine Learning Engineer Nanodegree program capstone.

The business problem relates to the german internationally-operating company Arvato Bertelsmann. The main target of the project is to extract useful information for the company, regarding two main areas.

First, detect which individuals from the german population are more likely to become customers of the company in the future. Second, detect those individuals who would be more likely to respond positively to a marketing email campaign.

This means that the project consist of two main areas:

- Use of an Unsupervised Learning method to detect groups of individuals in the german population. Then, detect which of those groups are more likely to become customers of the company, by comparing their characteristics with those of the already customers.

- Use of a Supervised Learning method to classify the target individuals of a marketing email campaign in two groups: those who will respond to the campaign and does who will not.

Using the model created in this last phase, we will predict the probability that each customer of a test dataset have to become customer. Those predictions will be submitted to the associated Kaggle competition. This will be a great way to check if the model is performing good enough.

Prior to the two described phases, the data must be in an adequate form. Thus, there is also a data cleaning/preprocessing phase that prepares the raw data to be accepted by the models.

The raw input data we have comes in 4 datasets:

- *Udacity_AZDIAS_052018.csv*: Contains demographics data for the general population of Germany. It has 891 211 rows and 366 features.
- *Udacity_CUSTOMERS_052018.csv:* Contains demographics data for customers of the company. It has 191 652 rows and 369 features.
- *Udacity_MAILOUT_052018_TRAIN.csv*: Contains data for individuals who were targets of an email marketing campaign, along with the response to that campaign. It has 42 982 individuals and 367 columns.
- *Udacity_MAILOUT_052018_TEST.csv*: Contains data for individuals who were targets of an email marketing campaign. It does not have a label with the response. It has 42 982 individuals and 367 columns.

Problem Statement

As stated previously, the problem has two sides:

- Detect potential new customers. For this, we will use the *k-means clustering* algorithm on the general german population. This will group the individuals based on their characteristics. Then, we can describe each group by looking at the centroid and get an idea of the main characteristics of the individuals belonging to that group.

For this first side, we are going to work with the azdias and customers datasets.

- Predict which of the targets of a marketing email campaign is going to respond positively. For this, we will use a classifier such as *Random Forest*, as it is easy to implement, generally has good results for most Machine Learning problems and prevents overfitting, as it considers different predictions of Decision Trees.

For this second problem, we will use the train and test datasets.

Before those two problems, we must *clean the data*. More precisely, we will take care of the following:

- Load the data to the working environment. Resample azdias and customers data.

As the azdias data has a high number of rows (around 900K) we will resample the data in order to speed up the computations. For the azdias, we take each 5th row and for the customers, we take each 3rd.

To be sure that this will not change the result, a Kolmogorov Smirnov test is performed on each variable of the full azdias data vs the resampled data. This test checks if two variables follow the same distribution.

- Clean the data

By this, we mean to remove variables with a large number of missings (the threshold was set to 50%, that is, drop the variables with more than 50% of missing rows), to drop highly correlated variables and constant variables. Also, the levels of the categorical variables are corrected.

We decided to delete the variables from the beginning, as there are 366 variables, which is a huge amount. That way, we could delete variables that will be unused from the very start and simplify posterior analyses.

We apply the cleaning steps to both azdias and customers. After that, we select the variables present in both datasets, as we must ensure that both have the same number of variables for successfully applying the algorithms. The train and test data are treated in a similar way.

- Dimensionality reduction

For this, we train a Random Forest on the join of azdias and customer data (with a label of 1 for the customers and 0 for the general). This allows us to select the most important

features. Further, we study which number of variables yields a better AUC score for the model and keep that variables list for later use.

To ensure those results were not just chance, we perform a permutation importance method, and obtain similar results.

Alternatively, we use the eli5 package to check if the variables obtained this way are much different. It turns out that they are similar.

Finally, we use the shap package to study the feature importance and obtain as well similar results.

- Preprocessing. That is, any step necessary to prepare the data to be used by a model.

The steps made were: StandardScaler to scale numerical features into the (0,1) interval, one hot encoding for the categorical variables, NA imputation with SimpleImputer, with the median strategy for the numerical variables and with a constant value for the categorical ones.

All of this steps were implemented using a scikit-learn pipeline and a column transformer, to specify the transformations for the numerical and categorical variables separately.

Metrics

For the first problem, we will use the elbow method to decide the number of clusters to choose. After, we will use the Silhouette score to check if the decided number gives good results in comparison to other numbers. The bigger the score is, the better. It has a maximum value of 1.

For the second problem, we will use the ROC-AUC score. This is a good metric for classification problems, specially for those with class imbalance, that is, where a label class occurs much more frequently than the other.

Analysis

Data Exploration and Preprocessing

The code for this section can be found in the notebook DataCleaningResample.ipynb

Check if the subsampled data and the full data follow the same distribution. For that, use the 2-sampled Kolmogorov Smirnov test.

We find that all sampled variables follow the same distribution as the original ones.

In [10]:
```
for variable in numerical_columns:

    n_vars = 0
    p_val = ks_2samp(azdias.loc[:, variable], azdias_full.loc[:,variable]).pvalue
    try:
        if (p_val.tolist()) < 0.05:
            print(f'Variable: {variable}; p-value: {p_val} ')
            n_vars += 1

    except:
        print(f'\tSth wrong for {variable}')

print(f'Number of variables with different distributions in original vs resampled: {n_vars}')
```

```
Number of variables with different distributions in original vs resampled: 0
```

Thus, we proceed to keep the sampled data.

Let's inspect the missing values from each dataframe.

```
missing_values_table(azdias).T
```

```
Your selected dataframe has 366 columns.
There are 273 columns that have missing values.
```

	ALTER_KIND4	ALTER_KIND3	ALTER_KIND2	ALTER_KIND1	EXTSEL992	KK_KUNDENTYP	ALTERSKATEGORIE_FEIN	D19_SOZIALES
Missing Values	178010.0	177008.0	172377.0	162073.0	130733.0	116923.0	52727.0	51398.0
% of Total Values	99.9	99.3	96.7	90.9	73.3	65.6	29.6	28.8

```
missing_values_table(customers).T
```

```
Your selected dataframe has 369 columns.
There are 273 columns that have missing values.
```

	ALTER_KIND4	ALTER_KIND3	ALTER_KIND2	ALTER_KIND1	KK_KUNDENTYP	EXTSEL992	KBA05_KRSZUL	KBA05_KRSKLEIN
Missing Values	63806.0	63471.0	62219.0	59989.0	37306.0	28351.0	18592.0	18592.0
% of Total Values	99.9	99.4	97.4	93.9	58.4	44.4	29.1	29.1

```
missing_values_table(train).T
```

```
Your selected dataframe has 367 columns.
There are 273 columns that have missing values.
```

	ALTER_KIND4	ALTER_KIND3	ALTER_KIND2	ALTER_KIND1	KK_KUNDENTYP	EXTSEL992	W_KEIT_KIND_HH	HH_DELTA_FLAG
Missing Values	42921.0	42788.0	42206.0	40974.0	25316.0	15948.0	9678.0	9678.0
% of Total Values	99.9	99.6	98.2	95.4	58.9	37.1	22.5	22.5

```
missing_values_table(test).T
```

```
Your selected dataframe has 366 columns.
There are 273 columns that have missing values.
```

	ALTER_KIND4	ALTER_KIND3	ALTER_KIND2	ALTER_KIND1	KK_KUNDENTYP	EXTSEL992	W_KEIT_KIND_HH	HH_DELTA_FLAG
Missing Values	42794.0	42632.0	42071.0	40820.0	25035.0	15809.0	9619.0	9619.0
% of Total Values	99.9	99.5	98.2	95.3	58.4	36.9	22.5	22.5

We see that some of the variables with the higher percentage of missings are the same for all datasets.

We proceed to eliminate those columns in the clean_data function. We show the results of this function applied to azdias.

```
%%time

azdias_clean = clean_data(azdias)
azdias_clean.shape
```

```
Initial df shape: (178244, 365)
Variables with missing values...
Your selected dataframe has 365 columns.
There are 273 columns that have missing values.
        Dropped 6 variables
Highly correlated variables...
        Dropped 63 variables
Constant variables
        Dropped 0 variables
Final df shape: (178244, 297)
CPU times: user 51.1 s, sys: 1.29 s, total: 52.4 s
Wall time: 52.9 s

(178244, 296)
```

Now, we do the same for customers and then we keep the variables that are in both datasets

```
final_vars = list(set(customers_clean.columns).intersection(set(azdias_clean.columns)))
len(final_vars)
```

```
285
```

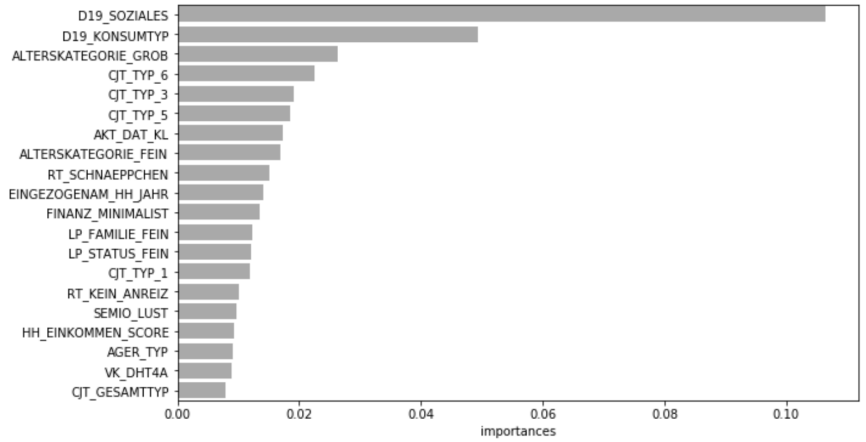
```
customers_clean = customers_clean[final_vars]
azdias_clean = azdias_clean[final_vars]

azdias_clean.shape, customers_clean.shape
```

```
((178244, 285), (63884, 285))
```

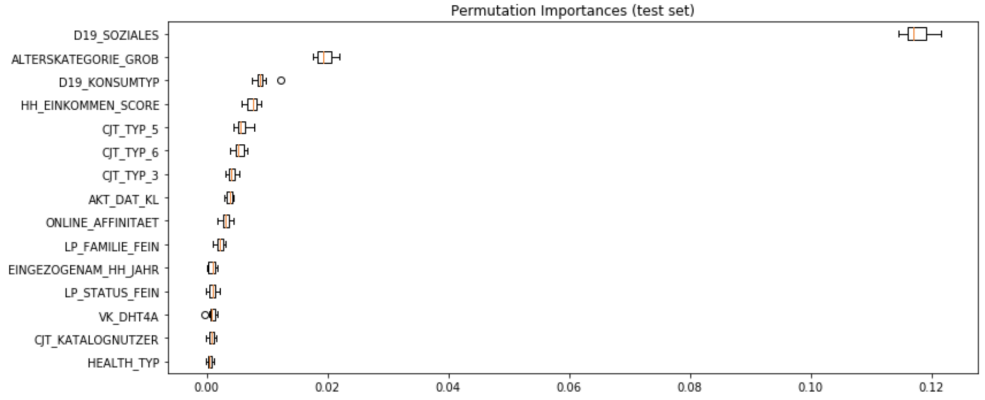
Now that the data is clean, we will define the Random Forest model to perform feature selection. We define the model through a pipeline that incorporates the previously described preprocessing steps.

```
def create_pipe(categorical_columns, numerical_columns):

    categorical_pipe = Pipeline([
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(drop = 'first'))
    ])
    numerical_pipe = Pipeline([
        ('imputer', SimpleImputer(strategy='median'))
    ])

    preprocessing = ColumnTransformer(
        [('cat', categorical_pipe, categorical_columns),
         ('num', numerical_pipe, numerical_columns)])

    rf = Pipeline([
        ('preprocess', preprocessing),
        ('classifier', RandomForestClassifier(random_state=SEED))
    ])

    return rf
```

The results are the following:



Now, we show the results of the *permutation importance* applied to the test set:



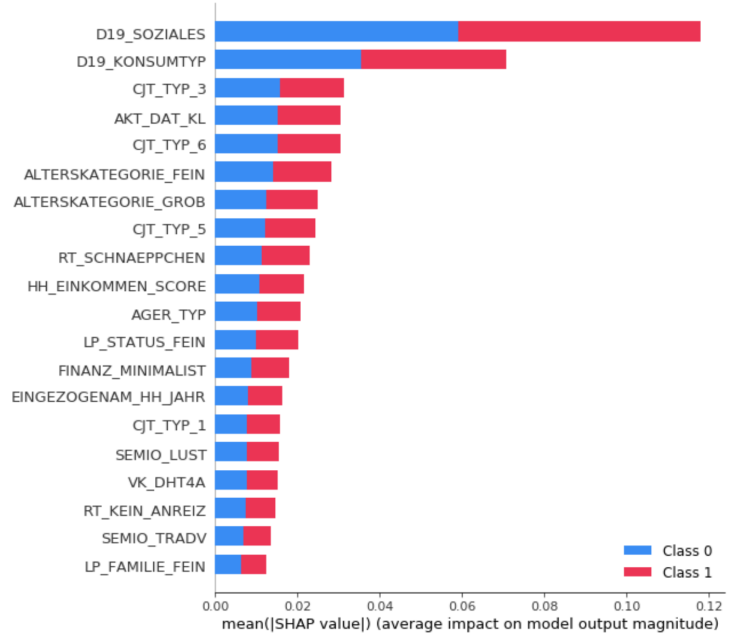
We observe that most of the most common variables are the same for both procedures.

As for the eli5 implementation, the results were the following:

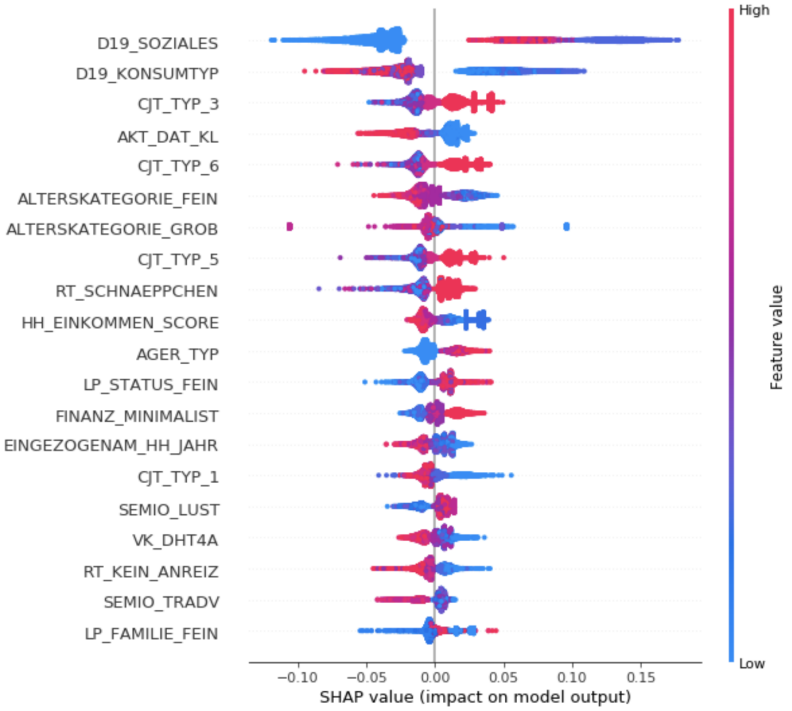
Weight	Feature
0.0594 ± 0.0018	D19_SOZIALES
0.0277 ± 0.0023	ALTERSKATEGORIE_GROB
0.0202 ± 0.0034	D19_KONSUMTYP
0.0068 ± 0.0024	CJT_TYP_3
0.0054 ± 0.0018	CJT_TYP_6
0.0037 ± 0.0026	RT_SCHNAEPPCHEN
0.0018 ± 0.0024	CJT_TYP_5
0.0007 ± 0.0012	FINANZ_MINIMALIST
0.0007 ± 0.0013	AKT_DAT_KL
0.0006 ± 0.0004	D19_RATGEBER
0.0006 ± 0.0008	SHOPPER_TYP
0.0006 ± 0.0006	SEMIO_KULT
0.0006 ± 0.0009	SEMIO_ERL
0.0006 ± 0.0005	SEMIO_REL
0.0006 ± 0.0007	KBA05_MOTOR
0.0005 ± 0.0012	D19_LETZTER_KAUF_BRANCHE_missing
0.0005 ± 0.0006	CAMEO_DEU_2015_missing
0.0004 ± 0.0003	D19_BANKEN_GROSS
0.0004 ± 0.0015	ALTERSKATEGORIE_FEIN
0.0004 ± 0.0007	KBA13_CCM_3000
	… 343 more …

We observe that just a few variables have weights that are not small (just six above 0.003). Nothing could be inferred for variables with weights smaller than 0.001.

Finally, the shap package method yields the following permutation importance.

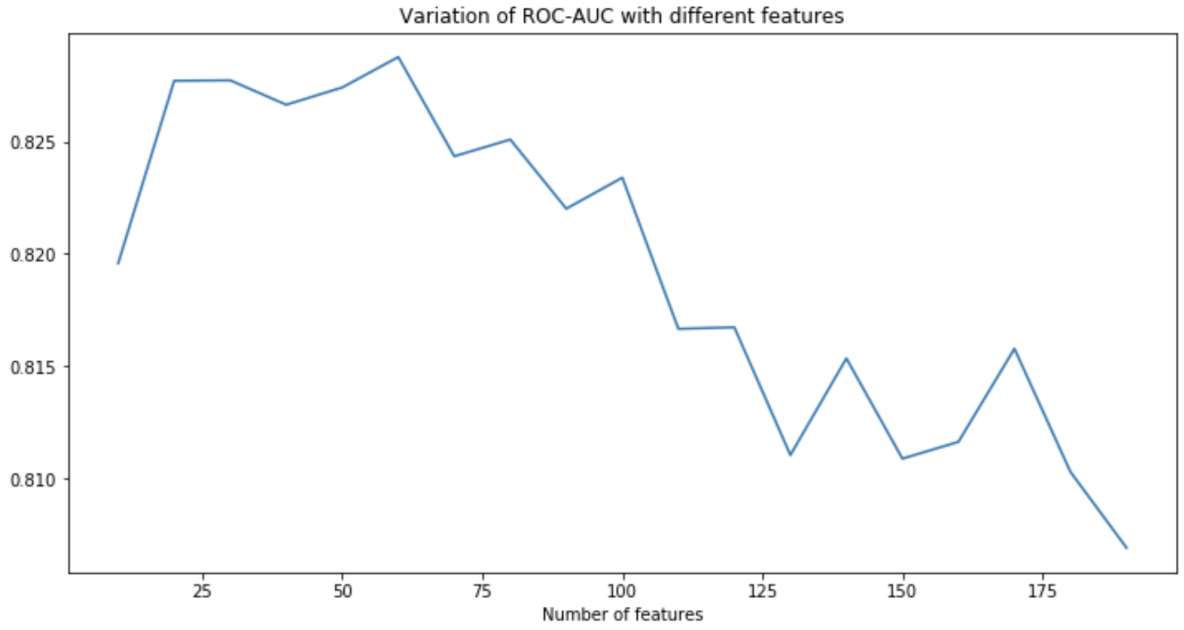


The selected variables are again very similar to the previous ones. With the SHAP package, we produced one more plot:



We observe that the high values of D19_SOZIALES are the ones that are affecting the predictions the most. But we also observe lower values that are affecting the predictions. Also, that the effects of this variable differ quite a lot.

Finally, we decide which number of the most important features to keep based on the ROC-AUC scores obtained:



We keep the 20 most important features, as it gives quite a good result and it's an easier to handle number compared to the number which yielded the best result (around 60).

Clustering

The code for this section can be found in the notebook CustomerSegmentationReport.ipynb

For the clustering problem, we use a dimensionality reduction technique, Principal Components Analysis (PCA), to reduce the number of dimensions used and help the k-means algorithm to better find the components.

We start with this pipeline, similar to the used in the previous section.

```
categorical_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(drop = 'first'))
])

numerical_pipe = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', MinMaxScaler())
])


numerical_columns, categorical_columns = var_lists(azdias_clean)

preprocessing = ColumnTransformer(
    [('cat', categorical_pipe, categorical_columns),
     ('num', numerical_pipe, numerical_columns)])

pca_pipeline = Pipeline([
    ('preprocess', preprocessing),
    ('pca', PCA()),

])
```

Then, we decide the number of components we want to use for PCA. We make this decision based on how much variance we would like to explain (more than 60%). We also want to choose a number which makes a difference with the initial number of variables (285).

Based on the following image, we decide to choose 60 components, as it explains more than 66% of the variance and still is a quite low number.

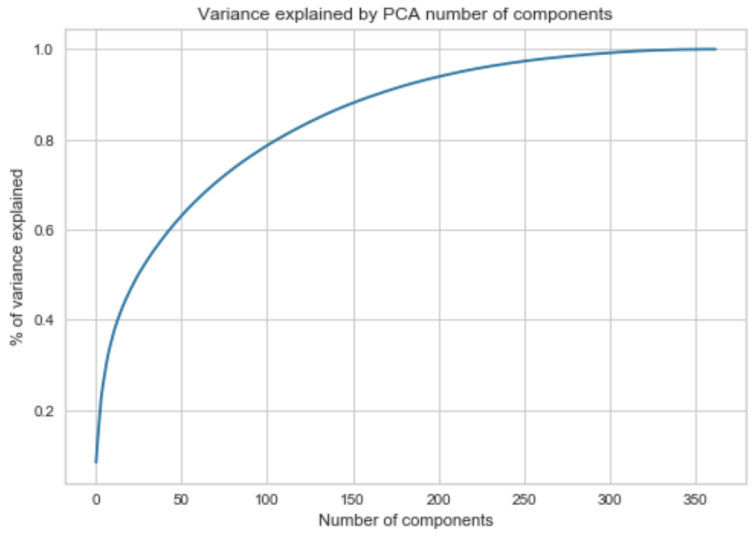
We update the previous pipeline with the number of components for PCA and fit it to the azdias_clean data.

Next, it is time to use the k-means algorithm. The first thing we want to do is to choose the appropriate number of clusters. For that, we will use the elbow method. That is, we will plot the distortion score and chose a value of k for which after it, the score starts to decrease slowly.

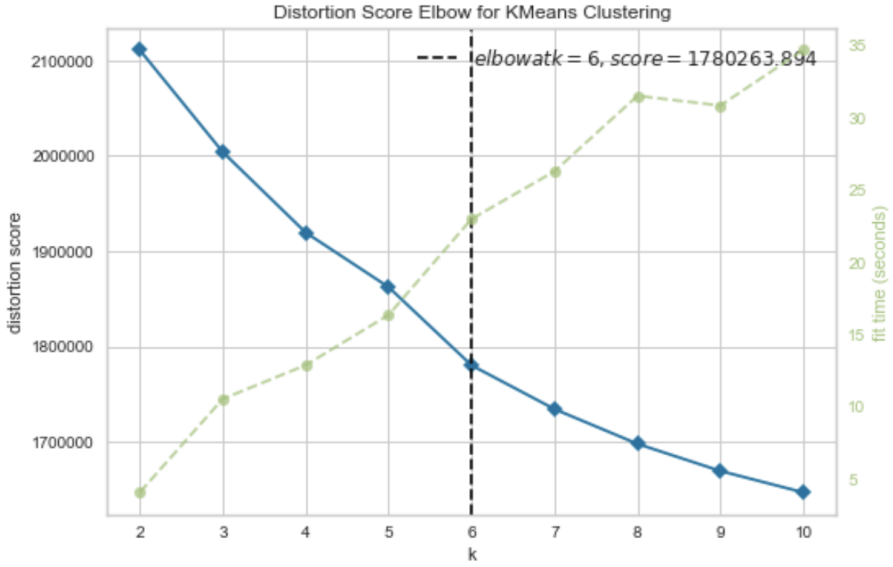
To make the plot, we use an implementation in the package yellowbrick.

```
%%time

decide_pca_components(azdias_clean, pca_pipeline)
```

```
Variance explained with 28 variables: 0.5148785143138102
Variance explained with 56 variables: 0.6514523205908614
Variance explained with 84 variables: 0.7433449373867859
Variance explained with 112 variables: 0.8110589090790504
Variance explained with 140 variables: 0.863697635158234

The 25.0% of explained variance is achieved with 4 components
The 50.0% of explained variance is achieved with 25 components
The 75.0% of explained variance is achieved with 86 components
The 90.0% of explained variance is achieved with 165 components
```




```
CPU times: user 206 ms, sys: 17.8 ms, total: 224 ms
Wall time: 219 ms
```



We can see that the suggested number of clusters to choose, based on that score and on the time to fit the algorithm, is k=6.

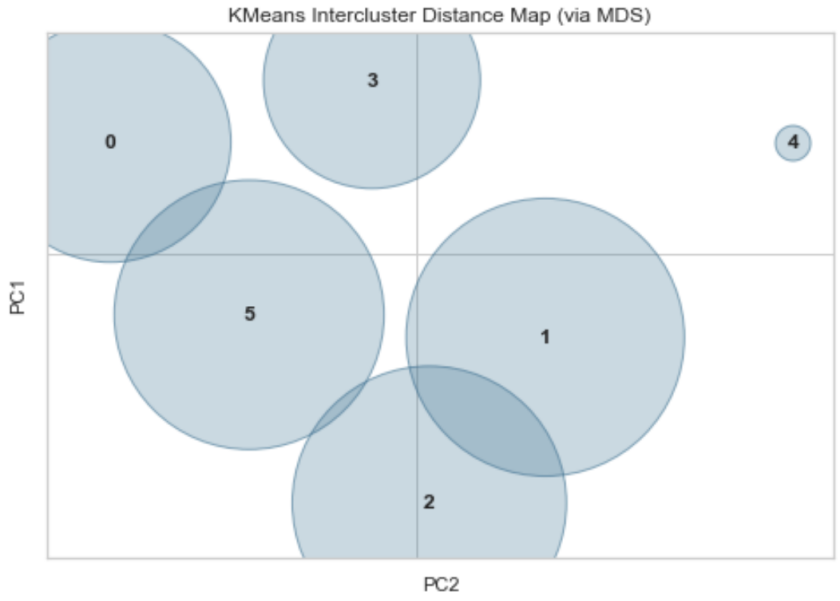
To check that this value is appropriate, we compute the Silhouette score for various values of k.

```
%%time

for i in range(2,12,2):
    kpipe = Pipeline([
    ('preprocess', preprocessing),
    ('pca', PCA(N_COMPONENTS)),
    ('kmeans', KMeans(i)),
    ])

    cluster_labels = kmeans_pipeline.fit_predict(sampled)
    print(f'Silhouette score for {i} clusters: {silhouette_score(sampled_pca, cluster_labels)} ')
```

```
Silhouette score for 2 clusters: 0.0835718566613641
Silhouette score for 4 clusters: 0.08346525796653692
Silhouette score for 6 clusters: 0.08330432911883902
Silhouette score for 8 clusters: 0.08523486695150384
Silhouette score for 10 clusters: 0.0835770049299996
CPU times: user 5min 26s, sys: 2min 32s, total: 7min 59s
Wall time: 2min 53s
```

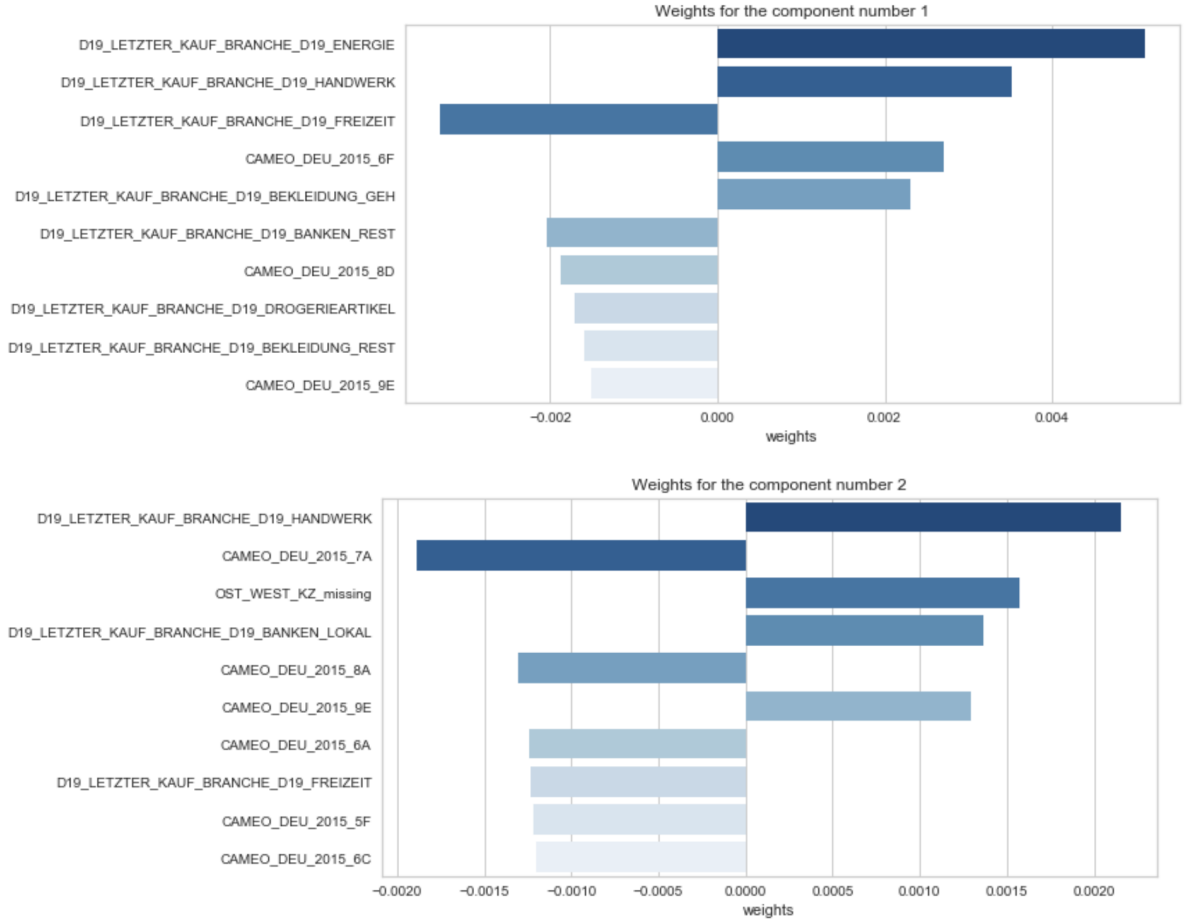
We can see that the scores are similar for all the values computed, and thus we decide to **keep the value of 6**.

Next, we visualize the clusters in the first to principal components:



We can observe a cluster (number 4) that is more separated than the others. In general, there are a few, clusters that overlap with each other. Apart from that, the clustering worked fine.

We could further interpret that plot by looking at the two first principal components weights, to see which variables are more represented in these components.

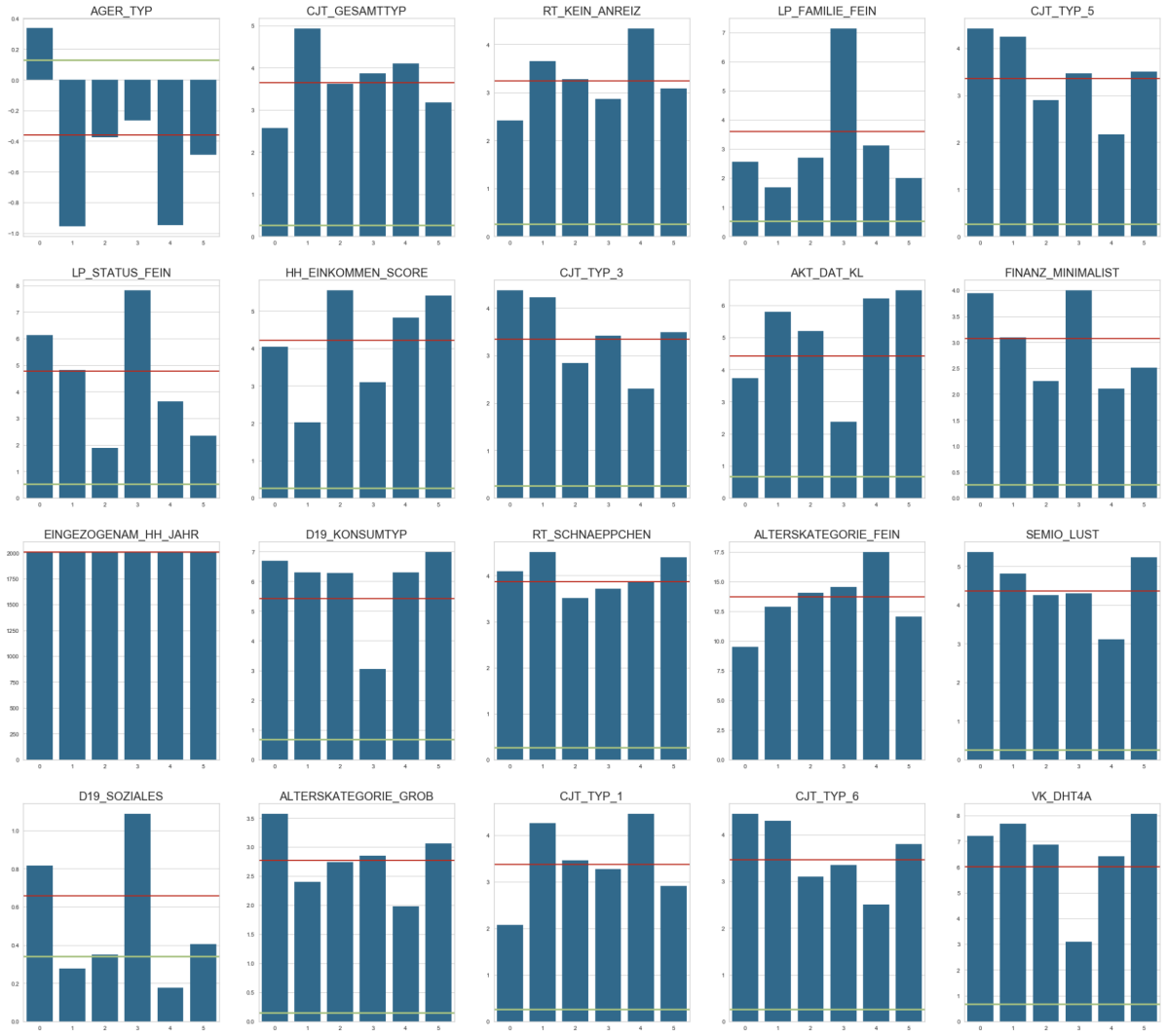


For example, the cluster 4 has high values of the first two components. That means that the elements of this cluster, have high influence of D19_LETZER_KAUF_BRANCHE_D19_ENERGIE and D19_LETZER_KAUF_BRANCHE_D19_HANDWERK.

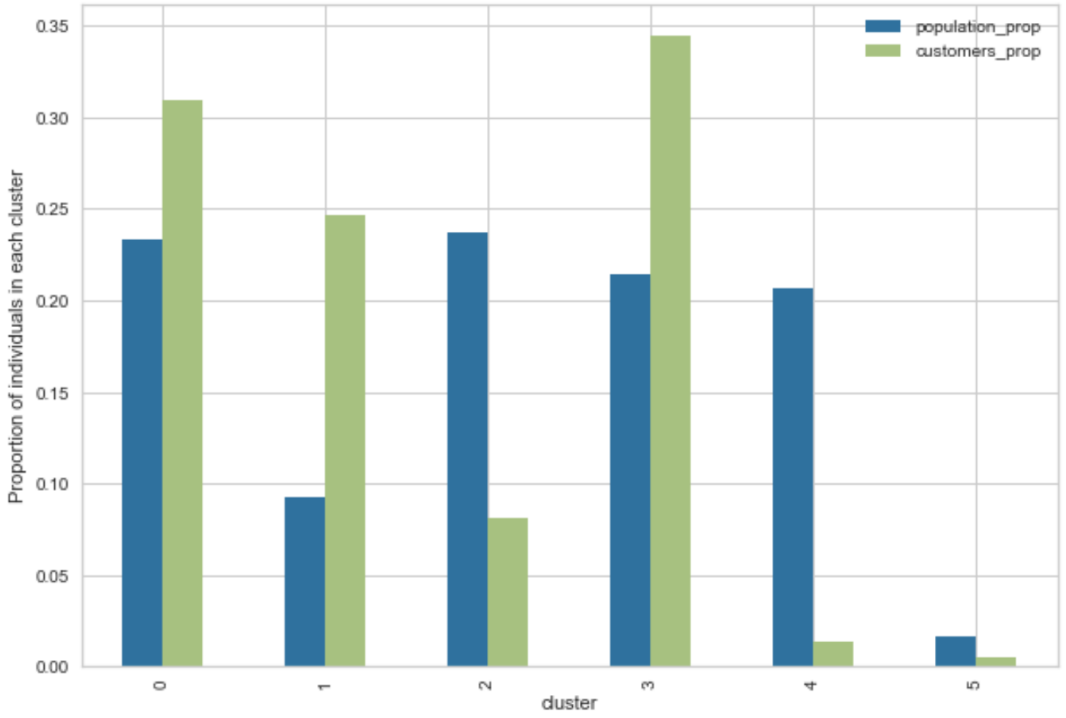
We can create a summary of each cluster by looking at the centroid for each cluster and for the most important features computed earlier. This could give us an idea of the characteristics of each group. To help with the visualization, we include a red line with the mean for all clusters and a green line with the mean for the customers population.

It is difficult to extract information for this plot, as we have different behaviour for different variables. For instance, it seems the cluster 0 has a behaviour close to customers for some variables (see AGER_TYP, LP_FAMILIE_FEIN), but not so close in other variables (see HH_EINKOMMEN_SCORE).

We probably need another way to detect potential customers.



With the algorithm already is defined and fitted to the data, let us use it on the customers data, to see on which cluster would each customer fall.



We see that there are a few clusters where the proportion of customers is much higher than the proportion of general population. Namely, the clusters 0, 1 and 3 are likely formed by potential customers for the company.

Thus, it would definitely payoff to further study those groups. The rise on the proportion is higher on clusters number 1 and 3. Those would be the number one candidates to study.

Supervised learning model

The code for this section can be found in the notebook Modeling.ipynb

We define the pipeline as follows:

```
def create_pipeline(df, model):

    categorical_pipe = Pipeline([
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('onehot', OneHotEncoder(drop = 'first'))
    ])

    numerical_pipe = Pipeline([
        ('imputer', SimpleImputer(strategy='median')),
        ('scaler', MinMaxScaler())
    ])


    numerical_columns, categorical_columns = var_lists(df)

    preprocessing = ColumnTransformer(
        [('cat', categorical_pipe, categorical_columns),
         ('num', numerical_pipe, numerical_columns)])

    classifier_pipe = Pipeline([
        ('preprocess', preprocessing),
        ('classifier', model)
    ])

    return classifier_pipe
```

This allows us for more flexibility defining the model.

To evaluate the results of the model, we will use the AUC score and use a Stratified 5-fold Cross Validation.

Baseline

As a baseline, we try with a Random Forest on the whole training set. This gives us a cross validation score of 0.61, which is far from perfect. We must try different approaches to improve the results.

First, we define the model using just the most important variables and get an improvement to 0.66, which is quite good, considering that it was a reduction from 285 to 20 variables.

We try to tune hyperparameters and get a best score of 0.71. The best parameters were:

n_estimators: 20, max_depth: 2, max_features: 2, min_samples_split: 2

We also tried with a PCA preprocessing step, but the base results were by far the worst, with around 0.56.

Lightgbm

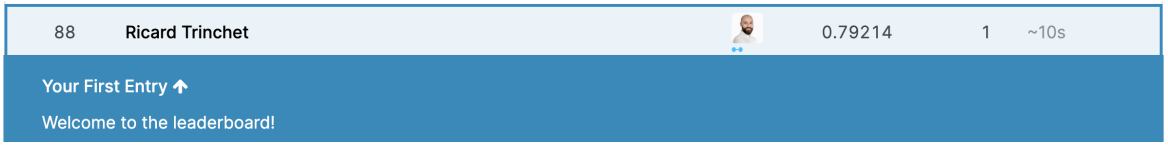
We decide to try with a different model, a Gradient Boosted tree. We choose the implementation of the lightgbm package.

This model performed better, with an out of the box performance of 0.745.

With hyperparameter tuning, the best result obtained was a 0.771 with the parameters:

n_estimators: 30, max_depth: 19, max_features: 1, num_leaves: 4

This was the first submission we make to Kaggle, obtaining the following result



Position 88 with 191 participants, the competition is fierce!

We try with different strategies to improve the results, but no one yielded much more improvement.

Those include:

- Oversampling and undersampling with SMOTE. Yielded worse results, around 0.73 on the leaderboard.
- Adding feature interactions for the most important variables. The features included square variables, sum, difference and multiplication. This approach gave the best leaderboard score (0.798)

Results

The final result for the Kaggle competition was an score of 0.798. This yielded the 63rd position at the time of writing this report.

This is not enough to land in the first positions of the competitions, but is good enough for the use case of the company.

To use in production, probably the best solution would be the model with the 20 most important variables. It had a quite good score on the test set (0.792), it was obtained with a quite short training time and just uses 20 variables. This would allow reduced inference times while in production. In general, using a simpler model is best for production environments.

Further improvement

It would be interesting to keep studying the posed problems on the following aspects:

- Generate an interactive report with the results of the customer segmentation. It would also be nice to be able to dynamically change the number of clusters and see the resulting plots.
- Deploy the supervised learning model (for example, using AWS) so that the marketing team in the company could use it to predict if a given individual is likely to respond to a campaign or not.

There is a lot of room for improvement in the understanding of the meaning of variables. The dictionaries provided are great, but for some of the most important variables, they don't give any information (for instance, the most important one according to the Random Forest, D19_SOZIALES).

If possible, it would be interesting to sit down with a person from the company, who would explain which are the most important business variables present and which demographic variables they expect to be the most relevant.