Machine Learning Engineer Nanodegree Program

Customer Segmentation for Arvato financial solutions

Capstone project report

Vajo Lukic, October 2021

Project Definition

Project Overview

Problem Statement

The problem statement we are going to solve is to find out: “How can a mail-order company which is selling organic food - acquire new clients more efficiently?” When creating a marketing campaign, we have to decide who will be recipients of marketing material. Every time when a recipient does not respond to an offer, that increases the cost of the campaign and reduces the expected profit. For that reason, we want to narrow down the list of choices and to send offers only to those recipients who are the most likely to respond positively. But how can we know which contacts are potentially the best prospects? Well, we can use our past experiences and learn from them, or our gut feeling. Or we can also try to use the data if we have it, which will be more explained in the following sections of this proposal.

Since our goal is to try to predict whether a person is a potential customer or not, this is a pure classification problem. Inputs to our classification model will be relevant features of existing customers, and output will be a prediction - a clear “potential customer” yes/no answer, for a given list of individuals.

Metrics

By looking at the data in “azdias” (population) dataset and a customer dataset, we can see that we will be dealing with imbalanced data, since a population data set is roughly 4 times larger than the customer data set.

Possible choices for the evaluation metrics in that case are: F1 score, precision, recall and area under the receiver operating curve.

We will be using the area under the receiver operating curve to evaluate performance of different models because it is one of the best options for the imbalanced data.

Analysis

Data Exploration

All the data files for this project were provided by Udacity and Arvato. Main data files are:

- 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).

Two more files providing the metadata and description of attributes in those files are:

- DIAS Information Levels - Attributes 2017.xlsx - a top level list of attributes and descriptions, organized by informational category
- DIAS Attributes - Values 2017.xlsx - a detailed mapping of data values for each feature in alphabetical order

Addressing warning

We have received some warnings already at the beginning, while loading datasets.

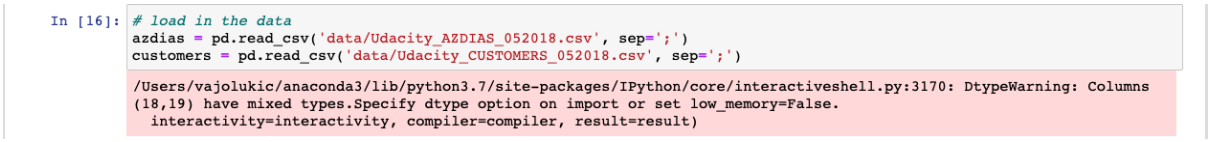


Fig.1 Warnings while loading datasets

This has created a need to analyze those columns and see which inconsistencies are there. We have discovered that some columns had mixed decimal values and values like ‘X’ and ‘XX’.

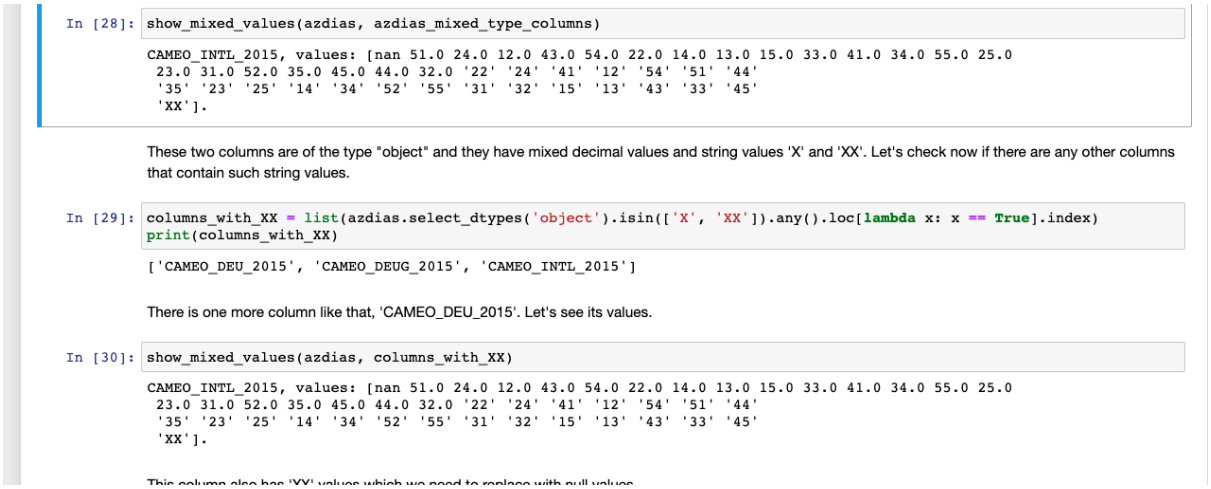


Fig. 2 Columns with mixed values containing ‘X’ and ‘XX

Later we had to replace those values with ‘null’ values.

Object type columns

The situation with mixed values has prompted us to look into object type columns more, and see if there are any other inconsistencies.

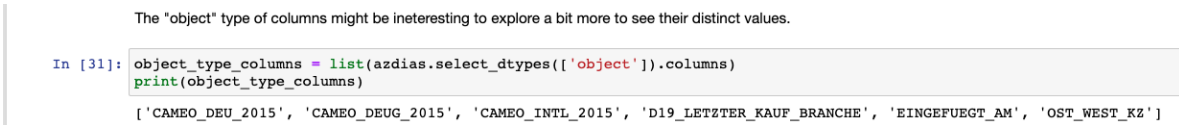


Fig. 3 List of object type columns

We have discovered that we will have to repair few more irregularities inside object type columns:

- CAMEO_DEU_2015: has an undefined value 'XX'. This will have to be replaced with null value.
- CAMEO_DEUG_2015: has an undefined value 'X'. This will have to be replaced with null value.
- CAMEO_INTL_2015: has an undefined value 'XX'. This will have to be replaced with null value.
- EINGEFUEGT_AM: this field has not been described in the additional document. If we translate it from German, it says it means "Inserting time". This information is not relevant so this field will be dropped later.
- OST_WEST_KZ: has two distinct string values which we have to convert to numerical values.
- CAMEO_DEUG_2015 and CAMEO_INTL_2015 will have to be converted to float type.

Integer type columns

The fact that there were such errors in the object type data proved to be useful also in case of other data types. The analysis of integer type of columns showed that we will also have to repair the data in following integer columns:

- LNR: this field seems to be an id of the record. It is not relevant for analysis so it will be dropped later.
- GEBURTSJAHR: has value "0" which has to be replaced with null value. This is also a real integer column.
- Most other integer columns have a very narrow range of distinct values. This indicates that they actually might represent categorical values although they have numerical values
- VERS_TYP and ANREDE_KZ are columns with binary values (1 and 2) which have to be converted to 0 and 1.

Decimal type columns

Analysis of decimal type columns also gave some insights and initiated some more data repair:

- ANZ_HAUSHALTE_AKTIV, ANZ_HH_TITEL, ANZ_PERSONEN, ANZ_TITEL: seem to be real numerical values according to the document "DIAS Attributes values 2017". These columns will be converted to integer type later.
- MIN_GEBAEUDEJAHR: has to be converted to integer type.
- EINGEZOGENAM_HH_JAHR: has to be converted to integer type.
- Decimal values actually look like integer values because they usually do not have a decimal part. This also indicates that there might be a lot of categorical values hidden inside decimal columns.

Discovering “unknown” values

So many inconsistencies in the data indicated that there are probably even more. We looked into the additional file “DIAS Attributes - Values 2017.xlsx” to study the data values and to understand better how they look. That is how we discovered a lot of attributes with meaning ‘unknown’. Clearly this indicates missing data for values of these attributes.

```
In [39]: unknown_meaning = ['unknown','unknown / no main age detectable', 'no transactions known']
         unknown_values = attribute_values[attribute_values['Meaning'].isin(unknown_meaning)]

In [40]: unknown_values.head()
```

Out[40]:

	Attribute	Description	Value	Meaning
0	AGER_TYP	best-ager typology	-1	unknown
5	ALTERSKATEGORIE_GROB	age classification through prename analysis	-1, 0	unknown
11	ALTER_HH	main age within the household	0	unknown / no main age detectable
33	ANREDE_KZ	gender	-1, 0	unknown
40	BALLRAUM	distance to next urban centre	-1	unknown

Fig. 4 Hidden missing values

Such attributes had a range of different values like "0", "-1" and "9", all of them with meaning “unknown”. Obviously we had to replace these values with real “null” values.

Cleaning the data

We have cleaned the data in both datasets by performing following transformations:

- Replaced 'X' and 'XX' values with real “null” values in both datasets
- Column named "OST_WEST_KZ" had two distinct string values which we converted to numerical values
- Column "GEBURTSJAHR" had value "0" which had to be replaced with a null value
- Columns VERS_TYP and ANREDE_KZ are columns with binary values (1 and 2) which had to be converted to 0 and 1
- Columns CAMEO_DEUG_2015 and CAMEO_INTL_2015 are actually ordinal and were converted to float type
- Column "EINGEFUEGT_AM" represents insert time, is not relevant and was added to the list of columns to be dropped
- Column 'LNR' is a row number, and was also added to the “to be dropped” list since it has no value for further analysis
- Different values which had “unknown” meaning were replaced with “null” values

Data Visualization

After replacing so many different values with “null” values, we had to analyse how many missing values there are, to decide what to do with them later.

Columns with missing data

We had to look into how much data each column (feature) is missing, to decide which ones to keep, and which ones to drop. If a column has too many missing values, it is not contributing anything to analysis and we have to drop it.

We tried to preserve the information within the data and minimize the number of columns that would be dropped and at the same time to remove those that have a lot of data missing. For that reason we’ve decided to drop only those columns that have 50% or more of missing data, or 29 columns.

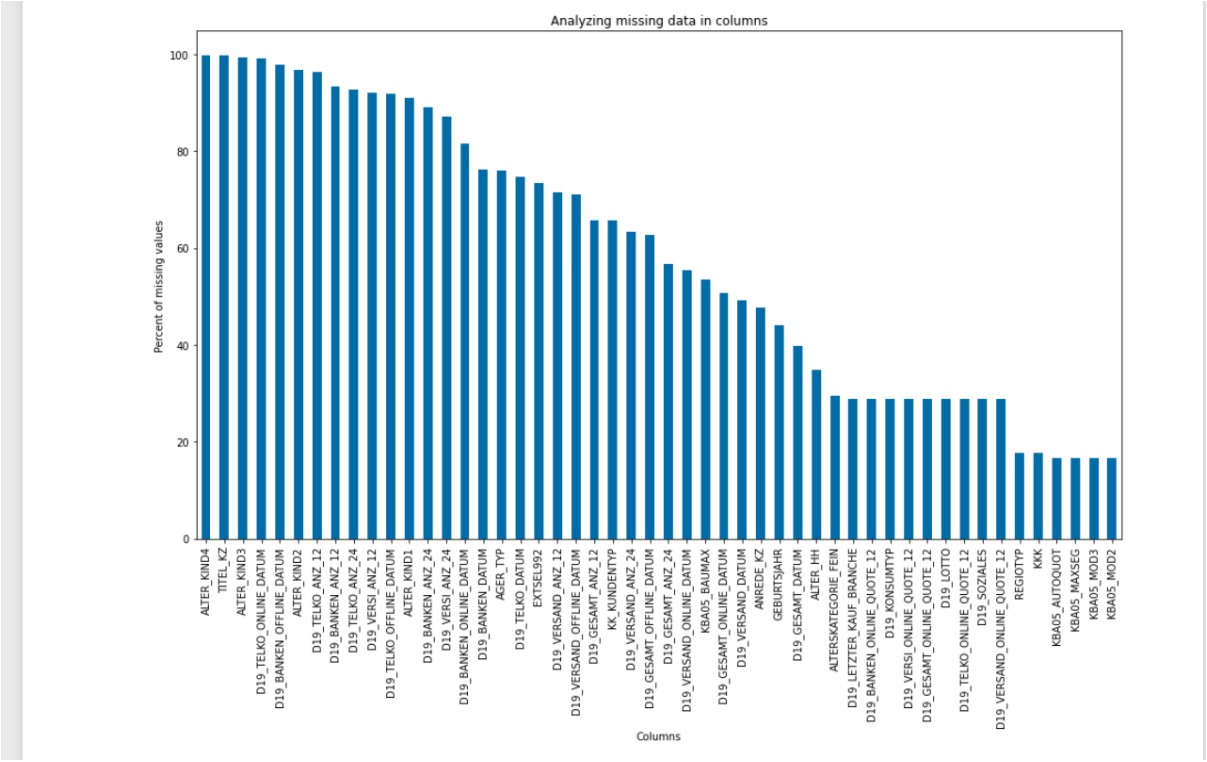


Fig. 5 Columns with missing data

Rows with missing data

We performed the same analysis also on the rows, trying to see which number or percentage of missing values there is in each row. (observation). If a row has too many values, then it might be really difficult to learn anything from it. Even if we impute missing values, due to the high number of missing values it might create biased data.

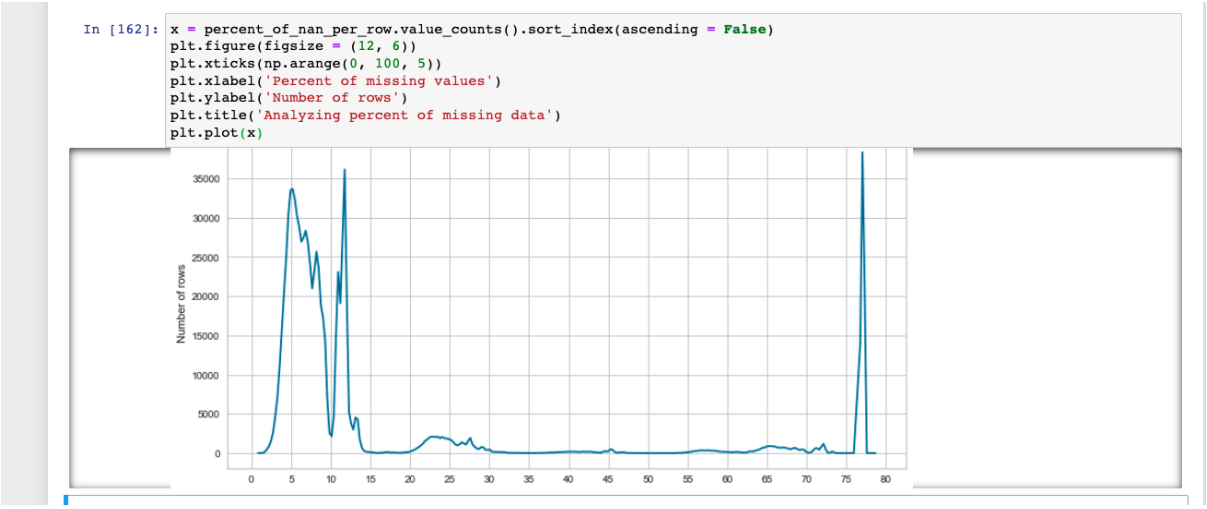


Fig. 6 rows with missing data

We can see that a high number of rows has a small percentage of missing values (< 30%). And a relatively small number of rows has a high percentage of missing values. We do not

want to drop too many rows and lose valuable insights we might get from them. To find a balance, we have decided to drop all rows with more than 30% of missing values.

Methodology

Data Preprocessing

Imputing missing values
It was necessary to replace missing values with synthetic values in order to provide the right data for later PCA analysis. Different replacement strategies were used for each type of data when imputing missing values:

- Missing values for nominal and binary features were replaced with the most frequent value of each feature.
- Missing values for numerical features were replaced with the mean value of each feature.

Feature encoding

We have performed so-called “one hot encoding” technique on all categorical features, to replace descriptive values with numerical values.

Encoding and scaling

```
In [90]: azdias_encoded = pd.get_dummies(azdias_reduced, drop_first = True, columns = azdias_categorical)
```

```
In [91]: azdias_encoded.head()
```

Out[91]:

	AKT_DAT_KL	ALTER_HH	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	ANZ_KINDER	ANZ_PERSONEN	ANZ_STATISTISCHE_HAUSHALTE	ARBEIT	BALL
1	9.0	3.0	21.0	11.0	0.0	2.0	12.0	3.0	
2	9.0	17.0	17.0	10.0	0.0	1.0	7.0	3.0	
3	1.0	13.0	13.0	1.0	0.0	0.0	2.0	2.0	
4	1.0	20.0	14.0	3.0	0.0	4.0	3.0	4.0	
5	1.0	10.0	10.0	5.0	0.0	1.0	2.0	2.0	

5 rows × 459 columns

Fig. 7 Encoding and scaling

Log transforming

For the purpose of having the right data for the K-means clustering step which is sensitive to skewed data, we’ve performed log transformation to remove any skewness in the data. This step had to be performed before standardization and scaling because they introduce negative values and log transformation does not work in that case.

Standardizing and scaling

These steps were needed to provide similar average values and variance for all features, as those were prerequisites for successful use of the K-means algorithm. This also removes dominance of features that have large scale values, when performing the dimensionality reduction.

Implementation

Customer segmentation

For this task we have used unsupervised learning techniques, to find out demographic common traits between existing company customers and the rest of the population in Germany.

Dimensionality reduction

After the last data preprocessing step we have ended with 458 features. Using so many features with any machine learning algorithm can result in overfitting. We want to avoid that, and for that reason we need to reduce the number of features. One good way to do that is to apply the technique called PCA or “principal component analysis”. This technique identifies those features which are highly correlated. By removing such features, we reduce the number of dimensions while at the same time we keep an information loss at the minimum.

Dimensionality reduction

```
In [96]: from sklearn.decomposition import PCA

In [97]: pca = PCA().fit(azdias_scaled_df)

In [98]: plt.figure(figsize = (16, 9))
         plt.xlabel('Number of components')
         plt.ylabel('Cumulative explained variance');
         plt.yticks(np.arange(0, 1.1, 0.05))
         plt.xticks(np.arange(0, 500, 50))
         plt.axhline(y = 0.8, color = 'r', linestyle = '-')
         plt.plot(np.cumsum(pca.explained_variance_ratio_))

Out[98]: [<matplotlib.lines.Line2D at 0x7fed81c6b750>]
```

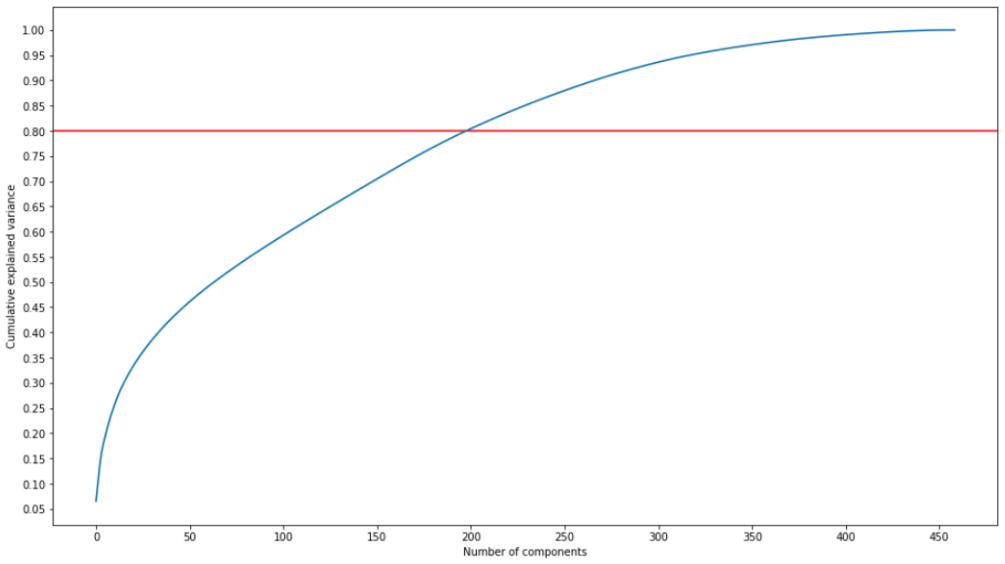


Fig. 8 Explained variance and number of components

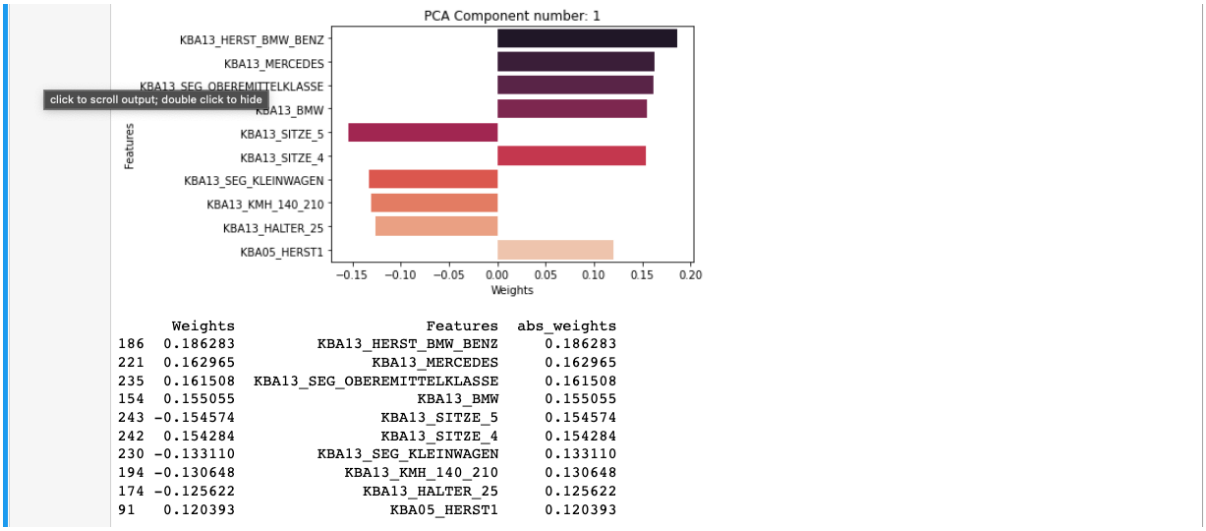
We have decided to keep only those features that explain up to 80% of variance. That resulted in keeping only 199 features out of the original 458.

Interpreting principal components

We can understand components that have been chosen by the PCA analysis if we look deeper into the weights of features that are included in those components. For example, if we analyze the component no.1 we can see that it is representing a certain group of people who:

- Mostly own german car brands such as BMW or Mercedes Benz (KBA13_HERST_BMW_BENZ)
- Who specifically own a car brand Mercedes (KBA13_MERCEDES)
- Who own a car belonging to upper-middle class (KBA13_SEG_OBEREMITTELKLASSE)

This component is represented in the picture below.



	Weights	Features	abs_weights
186	0.186283	KBA13_HERST_BMW_BENZ	0.186283
221	0.162965	KBA13_MERCEDES	0.162965
235	0.161508	KBA13_SEG_OBEREMITTELKLASSE	0.161508
154	0.155055	KBA13_BMW	0.155055
243	-0.154574	KBA13_SITZE_5	0.154574
242	0.154284	KBA13_SITZE_4	0.154284
230	-0.133110	KBA13_SEG_KLEINWAGEN	0.133110
194	-0.130648	KBA13_KMH_140_210	0.130648
174	-0.125622	KBA13_HALTER_25	0.125622
91	0.120393	KBA05_HERST1	0.120393

Fig. 9 Interpreting principal components

Finding the optimal number of clusters

Customer segmentation for population and existing customers was the main goal of this part of the project. We have chosen an K-Means algorithm to achieve this task. It is a simple and popular unsupervised machine learning algorithm.

K-Means algorithm tries to group similar objects together and to form clusters. Finding the optimal number of clusters is the hardest part of K-Means clustering. For that, we have used an “Elbow method” which takes into account the sum of square distances between each point and a center of the cluster. As the number of clusters increases, that sum will start to decrease. When we analyze the graph showing a number of clusters and a sum of square distances, we will see that at first, the graph curve falls rapidly until the point where it becomes almost parallel to the X-axis. The K value at an “elbow” of this curve represents the optimal number of clusters.

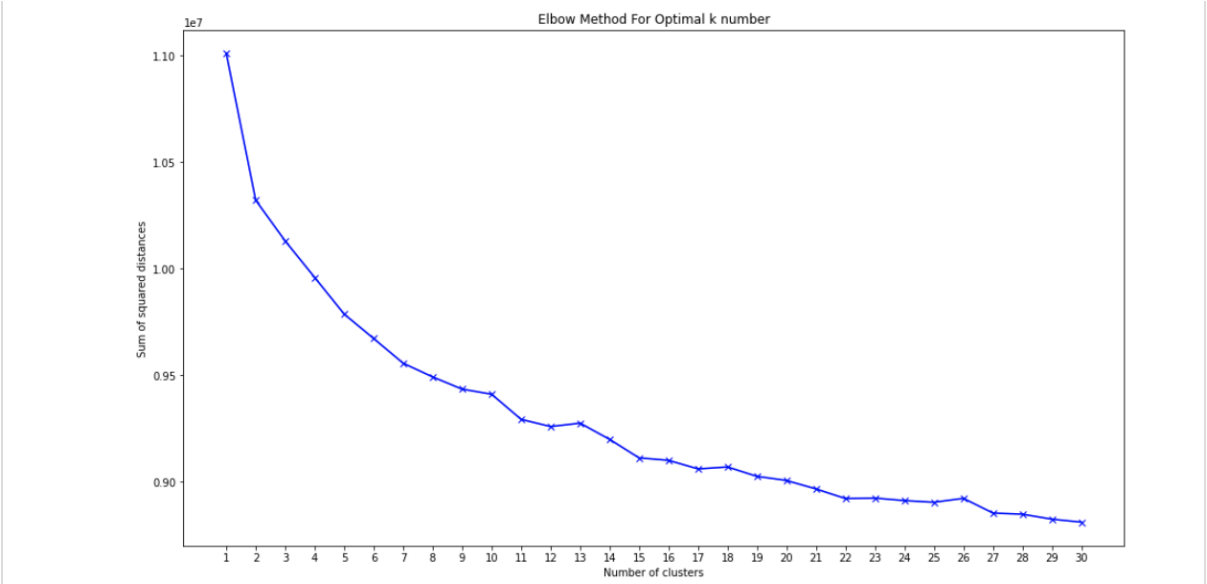


Fig. 10 Elbow method for finding optimal number of clusters

In the picture above we can see that after the 10 clusters, the change of sum of square distances is not decreasing so rapidly anymore. For that reason we have chosen 10 clusters as the optimal number of clusters to be analyzed further.

Analyzing population clusters

As expected by looking at the population clusters we see a much more uniform distribution than within customer clusters.

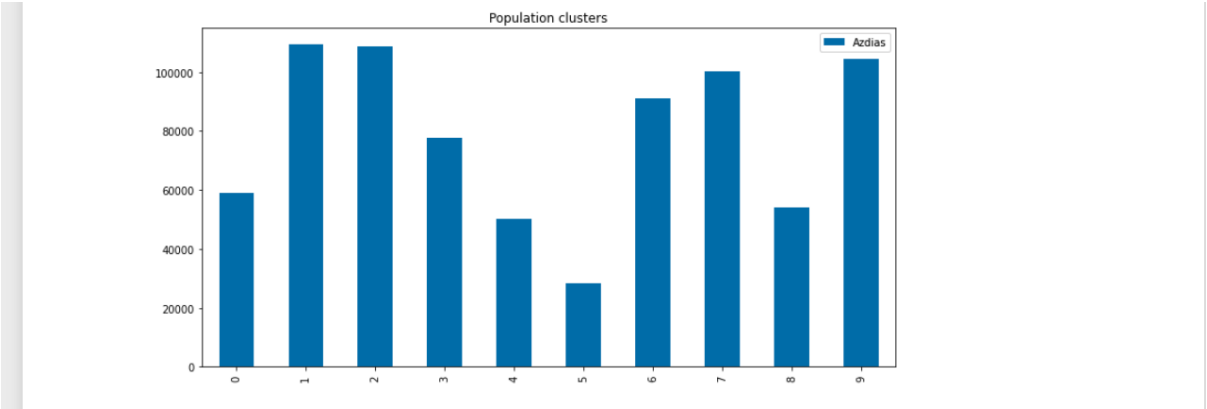


Fig. 11 Population clusters

Analyzing customer clusters

Customer clusters are quite different among themselves with clusters no. 0, 8, 4 and 2 especially standing out.

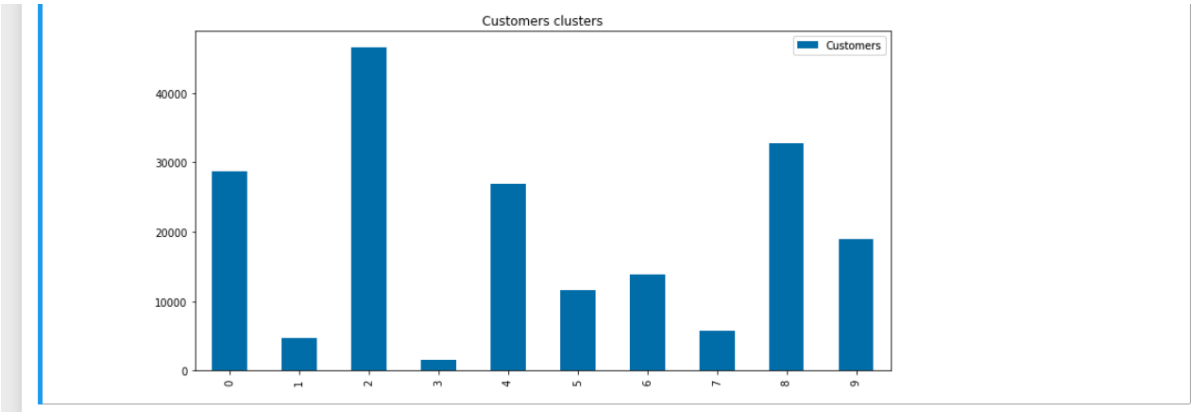


Fig. 12 Customers clusters

Comparing population and customer clusters

We also want to look at these two clusters together and to compare them. By looking at the picture below, we can identify those clusters that contain the biggest share of customers and at the same time have a high potential for new customers within the same clusters in the population.

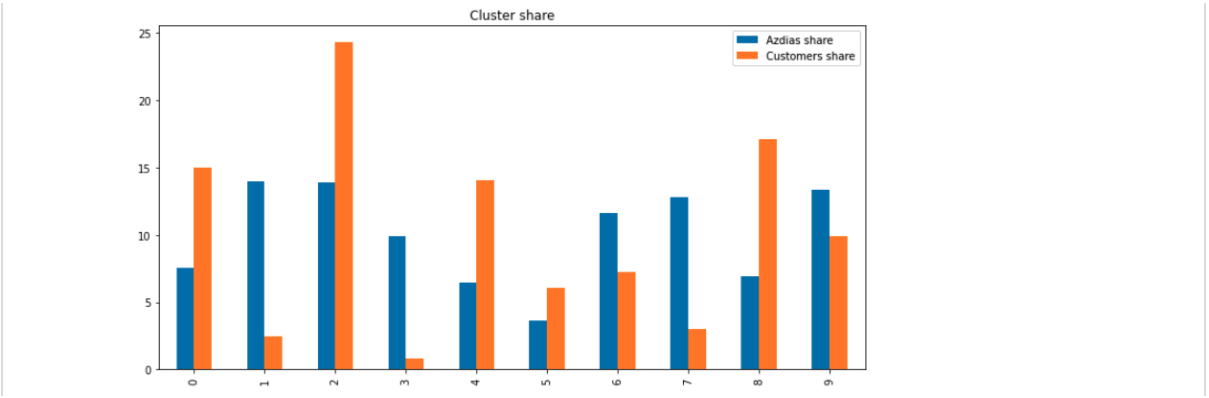


Fig. 13 Comparing population and customers clusters

We can see that the clusters 8, 4, 0 and 2 have the biggest potential.

Analyzing features of prominent clusters

We can further analyze each cluster to understand properties of the people belonging to each cluster. For example, if we look at the cluster no.0 we will see that the biggest weights have following features:

- GREEN_AVANTGARDE_1: belongs to the green avantgarde
- KBA05_HERST1: share of top German manufacturer (Mercedes, BMW)
- KBA05_KW3: share of cars with and engine power more than 119 KW
- FINANZ_ANLEGER: financial topology - investor

Out[229]:

	Features	Descriptions
423	GREEN_AVANTGARDE_1	Description missing
91	KBA05_HERST1	share of top German manufacturer (Mercedes, BMW)
107	KBA05_KW3	share of cars with an engine power of more tha...
118	KBA05_MOTOR	most common engine size in the microcell
186	KBA13_HERST_BMW_BENZ	share of BMW & Mercedes Benz within the PLZ8
82	KBA05_ANTG4	number of >10 family houses in the cell
65	FINANZ_SPARER	financial typology: money saver
12	CJT_TYP_1	Description missing
274	RT_KEIN_ANREIZ	Description missing
62	FINANZ_ANLEGER	financial typology: investor

Fig. 14 Features of important clusters

We can perform similar analysis for each cluster and understand which individuals are the best selling prospects and to target in our marketing campaign.

Customer campaign selection

For the second part of the project we have used supervised learning techniques for classification.

Creating train and validation datasets

As the first step we had to ensure that we will be training and validating our models on separate datasets. The main reason for this is to avoid overfitting. Having a separate dataset for validation is also valuable for later hyper parameter tuning. Here we have decided to keep 20% of data for validation, leaving the remaining 80% of data for model training.

Establishing a baseline

The next step was to establish a benchmark and to find a baseline model to which we can compare other models. That way we can choose the model that works the best with the data we have. Our choice for a baseline model was Logistic regression. Reason for that was because it is a simple and easy model to train. And it also had the lowest score of all the different models we have trained.

In [187]:
```
print('Accuracy on validation set: {:.2f}'.format(lr.score(X_validation, y_validation)))
print('Logistic regression ROC-AUC: {:.2f}'.format(roc_auc_score(y_validation, y_pred)))
```

```
Accuracy on validation set: 0.99
Logistic regression ROC-AUC: 0.61
```

Fig. 15 Defining a benchmark model

Testing different classification models

We have decided to try out multiple different classification models knowing that we do not have a lot of data to train the models on. Also, different models work in different ways and some might work better with our data than the others.

We have tried the following models: XGBoost, Gaussian Naive Bayes, K-nearest neighbor, AdaBoost, Random Forest and Gradient boosting. Eventually, it was the Gradient boosting model that had the best score. As evaluation metrics we have used “ROC - AUC” or area under the receiver operating curve.

```
In [190]: result_df
Out[190]:
```

	Model	Score
0	LR	0.611429
1	XGB	0.639996
2	GNB	0.509893
3	KNN	0.516694
4	AB	0.727124
5	RF	0.571227
6	GRB	0.753262

Fig. 16 Table with scores of different models

We’ve decided to go with that model and to try to find an even better set of parameters for that model than the default ones.

Refinement

Hyperparameter tuning

We wanted to find the best set of parameters for our Gradient boosting model. We have decided to use a “GridSearchCV " function for that, to be able to try different combinations of values for multiple parameters. This function goes through all different parameters we have specified and creates all possible combinations of their values, creating some sort of a grid. From that, it tries all these combinations on a given model, producing the best combination of parameters, based on the scoring metric of our choice.

The parameters we have used were: learning rate, max depth, number of estimators and max features. The reason for using only a limited number of parameters and their values was to try to get at least somewhat good results in a short amount of time.

Because we’ve had a limited amount of data to train on, and in order to improve the robustness of our model, we have used k-fold cross-validation with 5 folds.

The best set of parameter values was: learning_rate = 0.05, max_depth = 3, n_estimators = 100 and max_features = 100.

```
In [203]: best_estimator = clf.best_estimator_
          print("Best estimator on validation data: {}".format(best_estimator))

          Best estimator on validation data: GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                                     learning_rate=0.05, loss='deviance', max_depth=3,
                                     max_features=100, max_leaf_nodes=None,
                                     min_impurity_decrease=0.0, min_impurity_split=None,
                                     min_samples_leaf=1, min_samples_split=2,
                                     min_weight_fraction_leaf=0.0, n_estimators=100,
                                     n_iter_no_change=None, presort='deprecated',
                                     random_state=None, subsample=1.0, tol=0.0001,
                                     validation_fraction=0.1, verbose=0,
                                     warm_start=False)
```

Fig. 17 Set of parameters of the best estimator

With these values of parameters the model was slightly improved and had a ROC-AUC score of 0.76.

Further analysis of the best estimator is shown in this picture. It shows which features impact our model the most. Among them, the highest impact is from: “D19_SOZIALES”, “D19_KONSUMTYP_MAX”, “ANZ_KINDER” etc.

```
In [205]: feature_importances = pd.Series(best_estimator.feature_importances_, index = X_train.columns)
          feature_importances.nlargest(10).plot(kind = 'barh', figsize = (12, 6))

Out[205]: <AxesSubplot:>
```

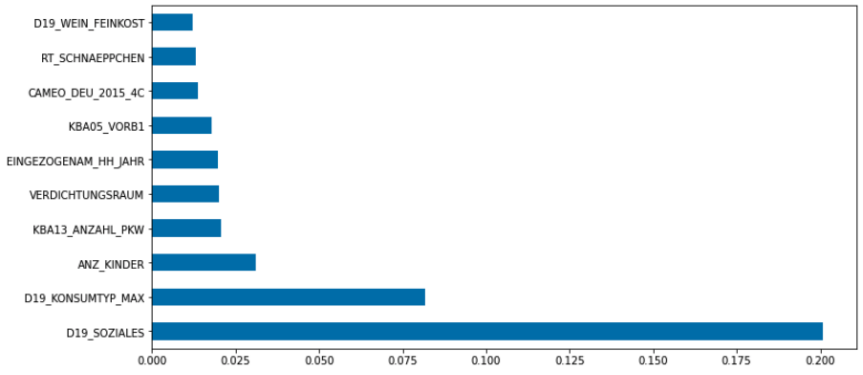


Fig. 18 Features with the biggest impact on the best estimator

Unfortunately we could not find a description for these features in the provided documents.

Results

The best estimator we have obtained in the previous step, was eventually used for a final test on a test dataset provided for Kaggle competition.

Preprocessing test data

Here we had to perform the same preprocessing steps on a provided test data set, before we could run our prediction model on it. We have performed data cleaning, imputing of missing values and scaling.

Making predictions on test data

After using the saved best estimator on a processed test data, we have got some results and saved them to a file.

Submitting results to Kaggle

After that, we used the saved file and made a submission to the Kaggle competition. The score of 0.78583 has been achieved with the 224th place on the scoreboard.

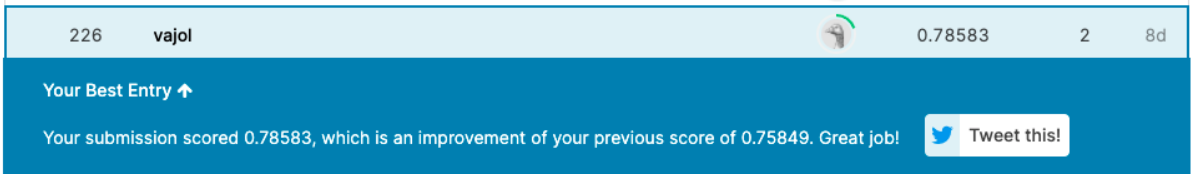


Fig. 19 Results of the Kaggle submission

Justification

The score of our final gradient boosting model was significantly better than the baseline Logistic regression model (0.79 vs. 0.61). Trying out different models paid off, because we discovered quite a big difference in their results. Optimizing hyper parameters of a chosen model has resulted in a slight improvement, and this can probably be improved even more by tuning other parameters.

This final result can be improved significantly if more effort and time are invested in it. In any case, the final solution is a decent result and a very good starting point for future work.

Conclusion

The project turned out to be a quite challenging one. The amount of work needed was much higher than what was anticipated. The reason was a very wide dataset with many features to explore and understand. There were a lot of preprocessed features that looked numerical, but were in fact categorical in nature. This resulted in a lot of analysis and data preprocessing.

Project structure, required tasks and deliveries has made it as close to real life tasks as possible. The amount of time spent on data preprocessing was also very close to what we can encounter in day-to-day work.

Improvement

Some possible improvements for this project could be to investigate any significant outliers and remove them, or to use a different imputation technique, where we'd use a smart algorithm to impute data based on distribution of each feature, or to try out an ensemble of different models instead of just one model.

References

Arvato Financial solutions:
https://finance.arvato.com/en/

Sklearn - PCA:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Sklearn - KMeans clustering:
https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Sklearn - model evaluation:
https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

Sklearn - model selection:
https://scikit-learn.org/stable/model_selection.html

Sklearn - GridSearchCV:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Gradient boosting hyperparameter optimization:
https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/

Kaggle submission:
https://www.kaggle.com/c/udacity-arvato-identify-customers/leaderboard#score