Machine Learning Engineer Nanondegree

Capstone Project

Customer Segmentation Report for Arvato Financial Services

Thorsten László Schmid

June 2021

Definition

Project Overview

The ‘Customer Segmentation Report for Arvato Financial Services’ was my project of choice for the Udacity Machine Learning Nanodegree program.

Many companies in the B2C sector are utilizing Targeted Advertising to boost their business. 'Targeted advertising is a form of advertising, including online advertising, that is directed towards an audience with certain traits, based on the product or person the advertiser is promoting’ (Wikipedia, Targeted advertising, 2020). In other words, the objective is to bring together the product with the people likely to buy it.

Arvato, a German services company (more details see Wikipedia, Arvato, 2020) is dealing with this issue in one of its projects as well. The client is a mail-order company selling organic products. The objective is to acquire new clients more efficiently by reaching out to the people becoming most likely new customers.

This project answers two essential questions to make targeted advertising feasible to Arvato. First, we want to acquire some insights about characteristic properties of the customers. The second issue gets us right into the context of Targeted Advertising. Here the question ‘Which individuals are most likely to become new customers when targeted by an email campaign’ (Udacity, 2020) will be answered in order to find appropriate candidates for email campaigns. Therefore, our project consists of two subprojects.

- Create a Customer Segmentation Report to identify the attributes of the population that best describe the core customer.
- The second task is to build a model that is able to predict if an arbitrary person is likely to become a customer or not.

Furthermore, there is a Kaggle competition. We can upload the prediction results of our favorite model and see how it performs compared to the models of other students.

Problem Statement

According to Timo Reis, Arvatos’ senior key account manager, the problem statement is ‘How can our client acquire new clients more efficiently.’ (Udacity, 2020)

The task is to figure out which people are most likely to become new customers. As mentioned in the project overview the solution consists of two parts.

We will match the dataset of the company's clients against a bigger set of Germany's inhabitants in the Customer Segmentation part. A typical solution to problems of that kind is to apply unsupervised learning techniques. We will use a combination of Dimensionality reduction via Principal Component Analysis (PCA). Afterwards we will apply a K-Means Clustering and finally extract the relevant attributes of the customer.

In the second task, we build a model that can predict whether a person is likely to become a customer or not. That means we have a binary classification problem. The Data science toolbox contains many items to solve supervised learning tasks of this kind. Our contenders are a Logistic Regression, an MLP Classifier and an XGBoost algorithm representing three important groups of machine learning models, namely linear models, neural networks and tree-based approaches. During the project, I became a fan of Scikit-Learn’s ExtraTreesClassifier because it is much faster than a

Random Forest Classifier at similar quality (Aznar, 2020). Therefore, we have added an Extra Trees Classifier to our list of contenders. A classical Random Forest Classifier will serve as benchmark.

Datasets and Inputs

We are dealing with demographic data having many features. 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. As the term 'demographic' might not suggest the data also provides information about things like spending behavior and habits in consumption etc.

The primary Data consists of four files:

- **Udacity_AZDIAS_052018.csv**: (891221 rows, 366 features) Demographics data for the general population of Germany
- **Udacity_CUSTOMERS_052018.csv**: (191652 rows, 369 features) Demographics data for customers of a mailorder company
- **Udacity_MAILOUT_052018_TRAIN.csv**: (42962 rows, 367 features) Demographics data for individuals who were targets of a marketing campaign
- **Udacity_MAILOUT_052018_TEST.csv**: (42833 rows, 366 features) Demographics data for individuals who were targets of a marketing campaign

In addition two EXCEL-files **'DIAS Information Levels - Attributes 2017.xlsx'** and **'DIAS Attributes - Values 2017.xlsx'** are included that can be used as legend for the columns, their meaning and the categorical values associated to them. We will refer to the **'DIAS Attributes - Values 2017.xlsx'** EXCEL-file as **'Documentation'** for the rest of this report.

Metrics

The segmentation report makes use of metrics to perform the PCA and the K-Means algorithm. The silhouette score helps us to find the appropriate k for the K-Means algorithm. We will see that the elbow method favored in the capstone proposal does not give a clear result.

'The silhouette value is a measure of how similar an object is to its own cluster (cohesion) compared to other clusters (separation)' (Wikipedia, Silhouette_(clustering), 2021). It is calculated using the mean intra cluster distance *(a)* and the mean nearest-cluster distance *(b)* for each sample (scikit-learn, 2021).

$$silhouette = (b - a)/\max(a, b)$$

The values are in the interval *[-1, 1]*. 'The silhouette score of 1 means that the clusters are very dense and nicely separated. The score of 0 means that clusters are overlapping. The score of less than 0 means that data belonging to clusters may be wrong/incorrect' (Kumar, 2021) or (Géron, 2016, p. 246).

Regarding the classification task, we also need a criterion to measure the quality of the predictions of our models. Let us look at the statistics of our target column 'RESPONSE'.

RESPONSE	COUNT	PERCENTAGE
0	42430	98.762 %
1	532	1.238 %

Statistics of the target variable 'RESPONSE' in the MAILOUT_TRAIN dataset

The ratio between class 0 and class 1 is roughly 99:1. That means we have to deal with a severe imbalance (Brownlee, 2021). An adequate measure for imbalanced problems is the *receiver operator characteristic* (ROC). The ROC plots the *recall* against the *false positive rate* or equivalently the *sensitivity* against *1-specifity* (Géron, 2016, p. 97). One big advantage of using ROC is that the curve is insensitive to disparities in the class proportions (Kuhn & Johnson, 2013, p. 264). One way to compare classifiers is to measure the *area under curve* (AUC). The better the classifier the nearer the area to 1.0. The more random a classifier the nearer is the AUC to 0.5, meaning that the classifier is purely random (Géron, 2016, p. 97).

Analysis

Data Exploration

First Look

The very first thing I did is create HTML-profile-reports of the data using the 'pandas_profiling' – python-library. This gives o quick overview of the features to get a feeling for the data.
The feature ***'LNR'*** is unique and has no missing values so it can be used as index in all pandas DataFrames.

	AZDIAS	**CUSTOMERS**	**TRAIN**	**TEST**
#features	366	369	367	366
#samples	891221	191652	42962	42833
#missing cells	33492923 (10.3%)	13864522 (19.6%)	2217179 (14.1%)	2186757 (13.9%)

Some statistical information about the datasets

The **CUSTOMERS** dataset has **369** features - three more than **AZDIAS**. These features are ***CUSTOMER_GROUP, ONLINE_PURCHASE, PRODUCT_GROUP*** and they were removed. The **TRAIN** test set has the additional column ***RESPONSE*** that is the target column of the supervised learning task. Apart from the extra columns mentioned above **all column names are the same**. Therefore, after defining ***'LNR'*** as the index we have **365** common columns left in the datasets.

The Documentation

As mentioned above the features are documented in the **'DIAS Attributes - Values 2017.xlsx'**-file. To make the analysis of the documentation easier, the **Documentation** class was implemented, where the EXCEL file is mapped to a dictionary. Here you can see the representation of one key as an example.

EXCEL:

Attribute	**Description**	**Value**	**Meaning**
RELAT_AB	share of unemployed in relation to the county the community belongs to	1	very low
		2	low
		3	average
		4	high
		5	very high
		-1,9	unknown

Python Dictionary:

```
'RELAT_AB':{
  'desc':'share of unemployed in relation to the county the community b
elongs to',
  'items':{
    -1:'unknown',
    1:'very low',
    2:'low',
    3:'average',
    4:'high',
    5:'very high',
    9:'unknown'
  },
  'item_keytype':'int'
}
```

Types of Features

The following Numpy-data types were recognized in the datasets:

- int (int64)
- float (float64)
- object

The features **CAMEO_DEU_2015, CAMEO_DEUG_2015, CAMEO_INTL_2015, D19_LETZTER_KAUF_BRANCHE, EINGEFUEGT_AM, OST_WEST_KZ** had the data type **'object'**. **D19_LETZTER_KAUF_BRANCHE** and **EINGEFUEGT_AM** contained date time values whereas **CAMEO_DEU_2015** and **OST_WEST_KZ** were nominal – they had string values. **CAMEO_DEUG_2015** and **CAMEO_INTL_2015** were encoded incorrectly. They contained the values **'X'** and **'XX'**, which were not part of the Documentation whereas all other values were numerical. All other features have numerical values (integer or float).
Therefore, from a feature engineers' point of view we have categorical and pure numerical features. Apart from the features with the object data type, the categorical features had numerical values. With a few exceptions (handled later) a sensible ordering was recognizable, but no equal spacing.

Overall, we have pure numerical features and categorical features where the vast majority is ordinal and a few features are nominal.

Feature Consistency - Datasets vs. Documentation

The next step was to check if the number of documented features matches the number of features in the datasets. The number of documented features was **314**, which is way below **365**.
The next question that came up was, if at least all of the **314** features occur in the datasets. An analysis showed that **42** attributes of the documentation were not included in the datasets meaning that there were **272** common attributes left. So **93** features are undocumented. It came out that many names of those unmapped attributes were similar to attribute names in the datasets. For example, the documentation contains the feature ***D19_BANKEN_DIREKT_RZ*** whereas we can find ***D19_BANKEN_DIREKT*** in the dataset, which is the same name without the suffix **'_RZ'**. After removing the suffix **31** more features could be mapped. Four more features were reconstructed by renaming:

- ***'CAMEO_DEUINTL_2015'*** to ***'CAMEO_INTL_2015'***
- ***'D19_BUCH'*** to ***'D19_BUCH_CD'***
- ***'KBA13_CCM_1400_2500'*** to ***'BA13_CCM_1401_2500'***
- ***'SOHO_FLAG'*** to ***'SOHO_KZ'***

At the end, I checked if the renamed attributes' value-ranges match with the documentation in order to ensure consistency. Five features of the documentation could not be mapped. Finally, **56** out of **93** features remain undocumented.

My decision was to keep the undocumented features, as they can be informative and important.

Treating the 'unknown' category

If we look at the documentation, we will discover that the category ***'unknown'*** is used to address unknown attribute values. The codes used for this category are **{-1, 0, 9}**. We can also observe that many features use two codes for the unknown category. For example ***ALTERSKATEGORIE_GROB*** uses **{-1**, **0}** and ***KBA05_ALTER*** uses **{-1, 9}** to identify the unknown category.
Another issue is that the unknown category biases the spacing. The attribute ***RELAT_AB*** from above has the code **-1** for ***'unknown'*** the other codes are **{1, 2, 3, 4, 5, 6, 7}** – there is no code **0** assigned. Admittedly, most features are ordinal so there is just ordering and no scale, but the algorithms applied on our data like PCA or scaling algorithms or any other algorithms using metrics are not aware of that.
Furthermore, an important question is which code to assign to the unknown category in terms of sensible ordering to maintain the monotony. For example, should we use the value -1 or 9? Taking everything in account, I decided to replace the unknown category by `np.nan`.

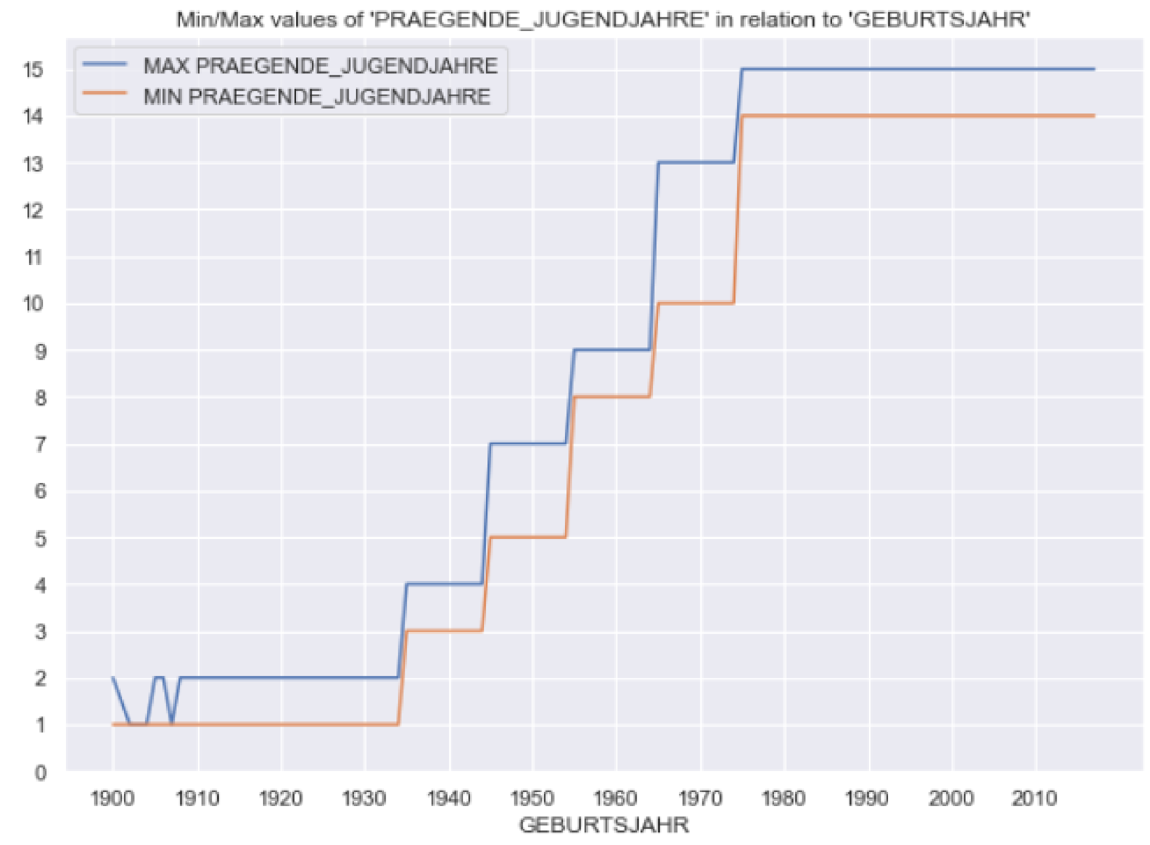
The consequence of that decision is that we need to identify unknown values within the undocumented attributes for the sake of consistency, too. Therefore, the undocumented attributes were carefully analyzed in order to find those values.

For example the features **ALTERSKATEGORIE_FEIN** , **VERDICHTUNGSRAUM** have the unknown category-code **'0'**. The feature **KOMBIALTER** seems to have the unknown category **'9'**.
The nominal feature **D19_LETZTER_KAUF_BRANCHE** had the value **'D19_UNBEKANNT'** ('unbekannt' meaning 'unknown') that was replaced by `np.nan`.
For feature **ALTER_HH** the term **'unknown / no main age detectable'** instead of **'unknown'** is used.

Mentionable features

The feature **GEBURTSJAHR** (year of birth) had the value **0** occurring 392318 times which corresponds to a proportion of 44%. As mentioned above **0** was interpreted as a missing value. Consequently as we will see below, the feature would have been a candidate for removal, but I wanted to keep it, as the age of the target group is important in the marketing context. Therefore, the decision was to reconstruct the feature by imputation. This was done by choosing **GEBURTSJAHR** together with the most correlated and anti-correlated features and applying a multivariate imputation approach using scikit-learns IterativeImputer on the data (scikit-learn, sklearn.impute.IterativeImputer, 2021).

It turned out that the feature **PRAEGENDE_JUGENDJAHRE** had a range of values that was too small. The documentation describes this feature as 'dominating movement in the person's youth (avantgarde or mainstream)' or in own words the 'formative years of a young person'. If we look at this feature in relation with **GEBURTSJAHR** we can observe that all people born in 1975 or later (up to 2017) had their formative years in the 1990s. That of course does not make any sense for all persons born in 1990s or later. An analogous conclusion can be drawn when looking at the people born 1935 or earlier. Therefore, the feature biases the data. The decision was to drop **PRAEGENDE_JUGENDJAHRE**.



The feature PRAEGENDE_JUGENJAHRE in relation to GEBURTSJAHR. For x-values >= 1975 the y-values are constantly either 14 or 15 both mean that the formative years are in the 1990ies. Analogously we observe y-values being either 1 or 2 for x-values<=1935. The y-values 1 or 2 mean that the formative years took place in the 1940ies.

The feature **CAMEO_INTL_2015** turned out to be a compound feature. It contained information about the wealth of a person and the temporal family status. It was split into two sub-features called **CAMEO_INTL_2015_WEALTH** and **CAMEO_INTL_2015_FAMILY_AGE**.

The features **LP_FAMILIE_GROB** and **LP_STATUS_GROB** proved to have a wrong Documentation. For example, let us look at **LP_STATUS_GROB**.

The EXCEL Documentation:

Attribute	**Description**	**Value**	**Meaning**
LP_STATUS_GROB	social status rough	1	low-income earners
		2	
		3	average earners
		4	
		5	
		6	independants
		7	
		8	houseowners
		9	
		10	top earners

When analyzing the distinct **LP_STATUS_GROB - LP_STATUS_FEIN** combinations in AZDIAS it turns out that it rather should be:

Attribute	Description	Value	Meaning
LP_STATUS_GROB	social status rough	1	low-income earners
		2	average earners
		3	independants
		4	houseowners
		5	top earners

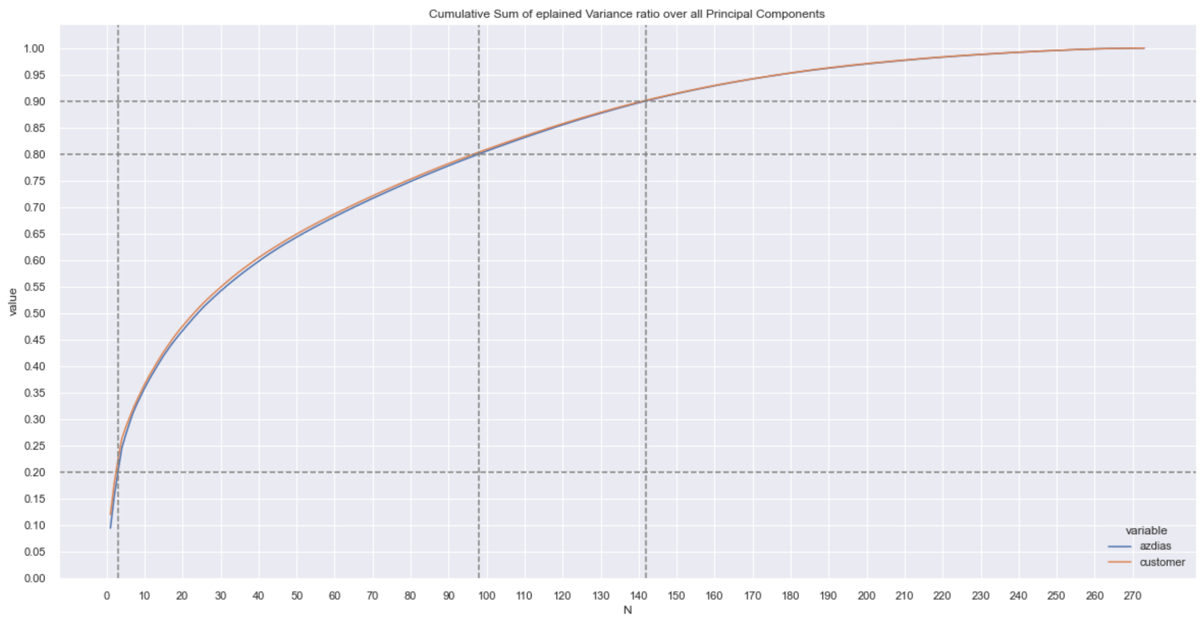
Algorithms and Techniques

PCA

In order to reduce the negative effects of high dimensionality (a.k.a. curse of dimensionality see. Hastie, Tibshirani, & Friedman, 2008 ch. 2) on K-Means, we apply the PCA Algorithm on our Data before clustering. As we know from our course, PCA helps us to reduce the dimensionality by focusing on the features with high variance.

Explained Variance

To get a clue of which principal components capture most of the data's variance, we take look at the cumulated sum of the explained variance ratio.



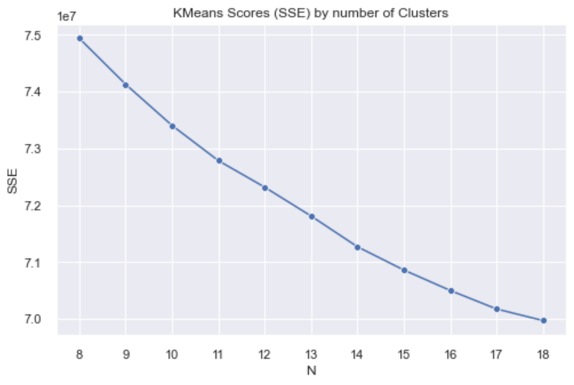
The cumulated sum of explained variance for the AZDIAS and the CUSTOMERS dataset. The first 3 components capture about 20% of the variance, the first 98 components capture 80% and the first 142 components capture 90% of the data's variance.

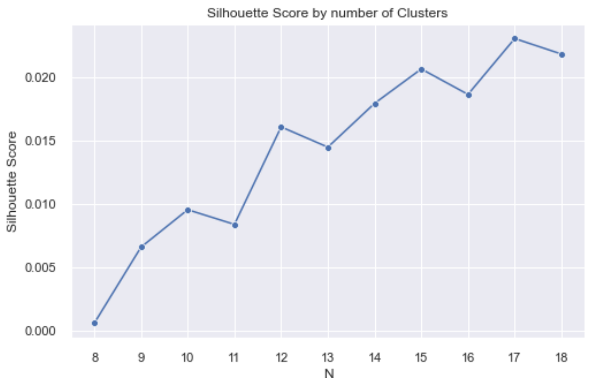
My objective was to keep 90% of the data's variance. Therefore, I chose the first **N=142** principal components.

K-Means-Clustering

Choosing the right K

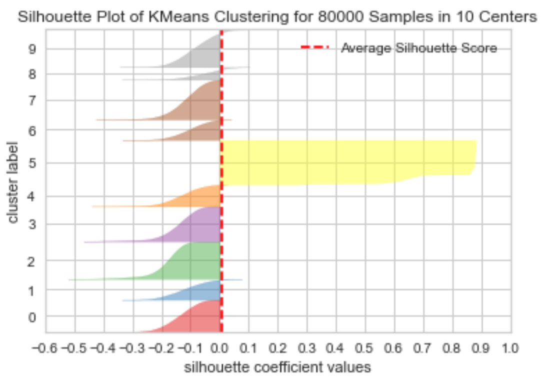
My basic Idea was to use the elbow method. This approach is based on the graphical analysis of the (negative) inertia, which is equal to the SSE (see also Géron, 2016, p. 244). The problem was that the elbow was not obvious, that is why we used the silhouette score additionally.

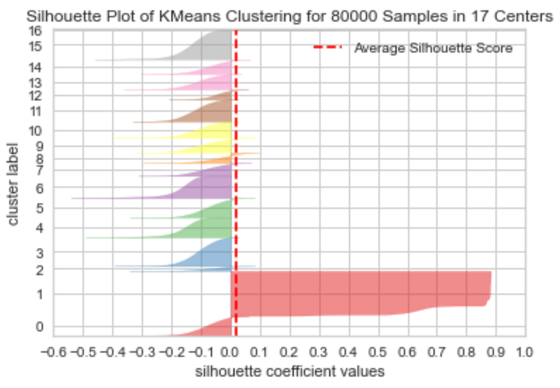




These plots show the Elbow Curve on the left and the corresponding silhouette scores on the right hand side. Further examinations were made at N=10, 12, 15, 17.

As mentioned in the 'Metrics' section above the silhouette score is a measure for the cluster cohesion, and therefore an indicator for the quality of K-Means' separation. Now according to Géron, 2016, p. 246, 247 we can get more valuable information when using silhouette diagrams.





The silhouette diagrams for N=10 and N=17 clusters. The dotted red line represents the average silhouette score. The hight of a knife shape indicates the number of members in a cluster. The width represents the sorted coefficients in the cluster.

Sihouette diagrams were only generated for the peaks at N=10,12,15,17, as the generation was time consuming. The average silhouette score is close to 0, that means that the clusters could be overlapping. But we have one wide dominating cluster with max coefficients close to 0.9 and all other clusters even have negative coefficients. Negative coefficients mean that some samples could even be assigned to the wrong cluster. My decision was to choose **N=10** as all clusters were close to the average silhouette coefficient and the scaling of the y-axis showed that they were similar in size.

Classification models

The basic idea was to use at least one model of the following categories:

- Linear Models
- Neural Networks
- Tree Ensemble based approaches

Because most of the features are converted to ordinal features, tree-based approaches are expected to perform better, as they usually do not depend on scaling.

Logistic regression

Logistic regression is our candidate representing linear models. The algorithm is designed for binary classification. Just like a linear regression, it calculates a weighted sum of its input features, but instead of returning this sum directly, it determines the probability of the input belonging to the positive class (label '1') and basing on that it returns the estimated class label (see also Géron, 2016, p. 142, 143).

MLP Classifier
Designing a well performing neural network can consume a lot of time and effort and many hardware resources when being properly trained. So I chose the easy way and decided to use Scikit-Learns' MLPClassifier. We can build deep feedforward networks with this classifier very easily (for more information see: scikit-learn, 1.17. Neural network models (supervised), 2021).

XGBoost
The first time I got in touch with this nowadays very popular and successful algorithm was within the sentiment analysis project of this course – and I found it cool. 'XGBoost' stands for e**X**treme **G**radient **B**oosting and is an implementation of gradient boosted decision trees with focus on performance (Brownlee, A Gentle Introduction to XGBoost for Applied Machine Learning, 2021).

Extra Tress Classifier
Like already mentioned in the Problem Statement, we added the Extra Trees Classifier to the list of the competitors. Extra Trees stands for **Ext**remely **Ra**ndomized **Trees**. The Extra Trees Classifier is similar to the RandomForestClassifier. It turned out to be very handy as it was much faster and the obtained results were almost the same. The two algorithms have a lot in common (see also Aznar, 2020):

- They are tree ensembles composed of a large number of decision trees
- Almost the same growing tree procedure

Differences:

- Random Forest operates on subsamples and Extra Trees takes the whole sample
- Cut point selection:
  - Random Forest calculates optimum split point
  - Extra Trees relies on randomization.

Benchmark
As we have to deal with many categorical values, I decided to use a Tree Ensemble-type of algorithm as benchmark so we do not have to care about scaling. Therefore, it should be not surprising that the choice fell on a classical **Random Forest Classifier.**

Methodology
Data Preprocessing
Removed features
Overall, **74** features were explicitly removed after a sorrow revision. A complete list can be found in Appendix A.
The general strategy was to keep the rough version if a rough and a fine version of the same attribute were available. For example, **LP_LEBENSPHASE_FEIN** was removed and the rough version **LP_LEBENSPHASE_GROB** was kept. Furthermore, two or more fine-grained features could be replaced by one summarizing feature. E.g., the features **KBA13_AUDI** and **KBA13_VW** were removed and **KBA13_AUDI_VW** was kept instead.
The formal justification was the correlation between the fine and the rough features respectively the

correlation between the sum of the fine features and the rough feature in case of multiple fine features vs. one rough feature.

Outliers

Outliers can have a massive effect when algorithms are using MSE or related metrics relying on squares and their sums. K-Means uses the *l2*-Norm as default metric. In the supervised learning part, we use Logistic Regression and a Deep Neural Net. Those algorithms also make use of *l2-penalization.* Outliers were checked among the numerical attributes. To detect and fix them I was applying Tukeys fence method based on the inter quartile range (readthedocs, 2021). The used multiplier was 2.2. Any detected outlier was capped. Outlier handling was applied when 3% or more of the data were identified as outliers for a feature. The following columns were identified to contain outliers for the AZDIAS dataset:

```
['ANZ_HAUSHALTE_AKTIV', 'ANZ_HH_TITEL', 'ANZ_KINDER',
'ANZ_STATISTISCHE_HAUSHALTE'].
```

Columns with low Standard Deviation

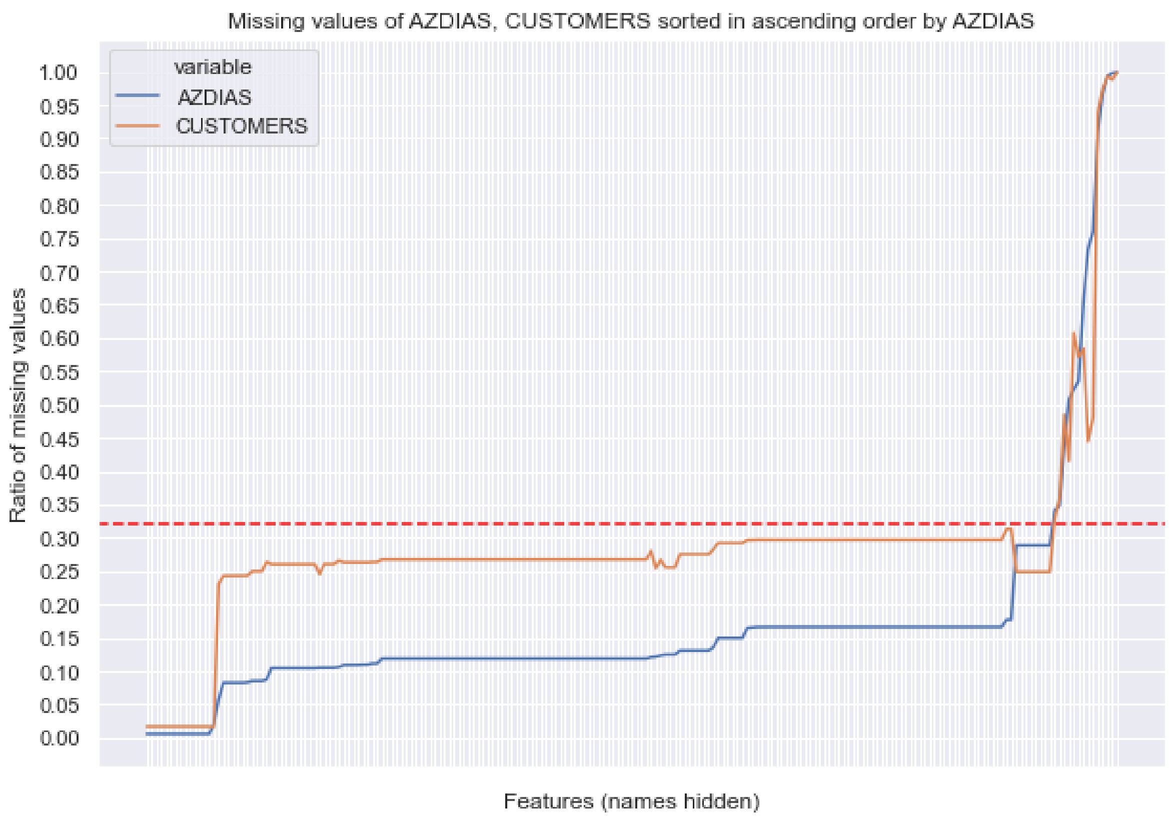
Columns with a Standard deviation below 0.25 were removed. Normally this step is not necessary, because PCA gives those features a very low rank. It is necessary for technical reasons. The removal of outliers can make a feature have (almost) constant values and this causes exceptions when using the *IterativeImputer* with the parameters *min_value* and *max_value,* if *min_value* equals *max_value.* This step takes place after the Outlier handling. The columns identified for the AZDIAS dataset were:

```
['ANZ_HH_TITEL', 'ANZ_KINDER', 'ANZ_TITEL', 'D19_TELKO_ONLINE_DATUM', '
DSL_FLAG', 'SOHO_KZ']
```

Missing Values

Removal

After cleaning the data and handling outliers it is time to care about missing values. In the first step, we want to know which features contain missing data and its amount. Therefore, the percentage of missing values was claculated for each feature of CUSTOMERS and AZDIAS datasets. Due to the high correlation between the missing values of CUSTOMERS and AZDIAS (approx. 0.9), it makes sense to sort the data by the values of AZDIAS in order to find a threshold for removal. Now when generating a line plot we can see that most of the data has missing values below 32% (red dashed line), which is our threshold.



The features of CUSTOMERS and AZDIAS with missing values. Due to high correlation (0.9) between the missing values of CUSTOMERS and AZDIAS, the dataset was sorted ascending by AZDIAS. The red line is at y=0.32 and it shows the threshold for removal.

Overall, we have **203** features with missing values and 14 of them are candidates for removal. That corresponds to a percentage of about 7%. The candidates for removal were:
`['AGER_TYP', 'ALTER_HH', 'ALTER_KIND1', 'ALTER_KIND2', 'ALTER_KIND3', 'ALTER_KIND4', 'ALTERSKATEGORIE_FEIN', 'D19_LETZTER_KAUF_BRANCHE', 'EXTSEL992', 'GEBURTSJAHR', 'KBA05_BAUMAX', 'KK_KUNDENTYP', 'TITEL_KZ', 'VERDICHTUNGSRAUM']`

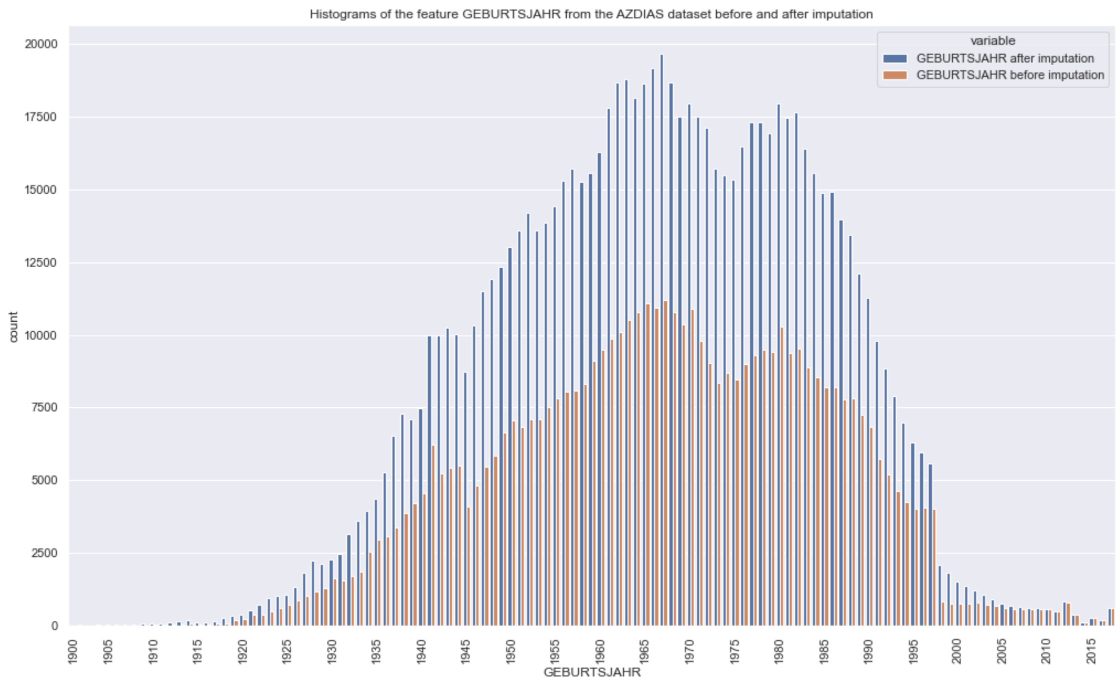
Taking into account that we want to keep **GEBURTSJAHR** and **ALTERSKATEGORIE_FEIN** for the reconstruction of **GEBURTSJAHR,** we have 12 features left to delete.

Imputation

After removing **12** features, we still had **191** features left with missing values. The missing values of those features were reconstructed by Imputation. As there are dependencies among the features, a multivariate imputation approach was most suitable. Therefore, I used Scikit-Learn's IterativeImputer (scikit-learn, sklearn.impute.IterativeImputer, 2021).

Imputation was done in two stages. The first objective was the reconstruction of the feature **GEBURTSJAHR**. The other features were reconstructed in a second step. The reason for this strategy was that different parameters of the imputer had to be used due to achieve acceptable computation time. The imputation of **GEBURTSJAHR** was done in a more costly manner than the imputation of the other features. I evaluated two candidates for the underlying estimator – an *ExtraTreesRegressor* and the default *BayesianRidgeRegressor*. Furthermore, I played around with different values for the parameters *'imputation_order'* ('ascending' vs. 'descending') and *'initial_strategy'* ('median' vs. 'mean'). The quality of the **'GEBURTSJAHR'**-imputation was much better when setting the *sample_posterior*-parameter of the Imputer to *True* (in case of an underlying

BayesianRidgeRegressor), meaning that the estimator produced an output distribution from which samples are drawn instead of single prediction values. The neighbors of **‘GEBURTSJAHR**’ were selected explicitly from the most correlated and anti-correlated features.



The feature GEBURTSJAHR before (with the value ‘0’ replaced by *np.nan*) and after imputation.

In the next step the other features were imputed with the parameters ‘*n_nearest_features’=4* and *sample_posterior=False*.

Scaling

PCA is an algorithm for which normalization is Important. Therefore, before applying PCA we let a *StandardScaler* pre-transform the data in the customer segmentation task (Raschka, 2014).

In the prediction task, we used a MinMaxScaler for the MLPClassifier and the Logistic Regression. We also could have thought about using a StandardScaler, but as we might have been little too careful handling outliers, I chose the MinMaxScaler and as Raschka pointed out it depends very much on the application (Raschka, 2014). Therefore, there is no right or wrong. Our tree based models; the ExtraTreesClassifier and XGBoost do not need scaling at all (stackexchange, 2021).

Implementation

Segmentation

Pure python code is provided in the files `common_functions.py` and `featureengineering.py`. The file `common_functions.py` contains commonly used functions. The file `featureengineering.py` provides functionality to make data preprocessing easier. Therefore, we have three important classes:

Documentation is a wrapper-class around a dictionary storing the data from the *'DIAS Attributes - Values 2017.xlsx'* **–** file.

FeatureEngineer is a class that provides atomic data preprocessing operations. It uses an instance of Documentation.

PreProcessor provides high-level methods for data preprocessing. To do that it uses an instance of the class *FeatureEngineer* under the hood. The core method '`process()`' does the preprocessing. The transformation takes place in five steps. At the end of each step, the processed data is stored on disk. Methods to load the resulting DataFrames of the processing steps also exist.

```python
class PreProcessor:
    # ...
    # definition of member variables (not shown)
    # ...

    def __init__(self, feature_engineer=None,
                 azdias=None, customers=None,
                 mailout_train=None, mailout_test=None,
                 root_path='.', out_dir='tmp_dat_prod',
                 missing_values_threshold=0.32):
        # ...
        # initialization of member variables (not shown)
        # ...


    def process(self):
        """
        Perform all preprocessing steps on the given datasets.
        These steps are:
        - 1 cleaning
        - 2 handle outliers and features with low std_dev
        - 3 handle missing values and recode some features
        - 4 impute
        - 5 scale
        :return: None
        """
        self.step1_clean()
        self.step2_handle_outliers_and_low_std_dev()
        self.step3_handle_missing_and_recode_composite_features()
        imputed_azdias, imputed_customers, imputed_mailout_test, \
        imputed_mailout_train = self.step4_impute()
        self.step5_scale(
            imputed_azdias, imputed_customers,
            imputed_mailout_test, imputed_mailout_train
        )

    def step1_clean(self):
        # implementation not shown; see code

    def step2_handle_outliers_and_low_std_dev(self):
        # implementation not shown; see code

    def step3_handle_missing_and_recode_composite_features(self):
        # implementation not shown; see code

    def step4_impute(self):
        # implementation not shown; see code

    def step5_scale(self, imputed_azdias, imputed_customers,
                    imputed_mailout_test, imputed_mailout_train):
        # implementation not shown; see code
    def load_step_1_cleaned_dfs(self):
        """
        Load all Processed dataframes in Phase 'clean' and return
        them as tuple
        :return: tuple of DataFrames. It has the form
        (azdias, customers, mailout_train, mailout_test)
        """
        # implementation not shown

    def load_step_2_outliers_handled_dfs(self):
        # should be clear implementation not shown

    def load_step_3_missing_handled_dfs(self):
        # should be clear implementation not shown

    def load_step_4_imputed_dfs(self):
        # should be clear implementation not shown

    def load_step_5_scaled_dfs(self):
        # should be clear implementation not shown

    # ...
    # here comes the other code
```

Classification

All models were available in the Scikit-Learn library, except for XGBoost which is available as python library via PyPi. As the XGBoost implementation followed the Sciki-Learn-API, it could be used the same way as the other models.

Refinement

Hyperparameter Tuning

Scikit-Learn's GridsearchCV was used to tune the hyperparameters of our models. GridsearchCV helps to find the best possible parameter-combinations for an estimator given the collections of parameter-candidates. Furthermore, the algorithm uses cross-validation to find the best values (Géron, 2016, p. 75). Additionally the training data set was split into a training and a validation set using a train-test-split with test-size 20%.
The hyperparameters of ExtraTreesClassifier and XGBoost were tuned in iterations in order to save computing time. When available, special parameters for imbalanced learning were used.
The parameters of Logistic Regression and the MLPClassifier were tuned in one step.

At the beginning, I used a RandomForestClassifier with the default parameters apart from the random seed. The score was in the region of 60% what was too bad in my opinion. I wanted to have a competitive benchmark. That' s why I decided to add some more parameters. First, we had to deal with the class-imbalance. Therefore the parameter *class_weight* was set to *'balanced'* (scikit-learn, sklearn.ensemble.RandomForestClassifier, 2021). To mitigate the risk of overfitting the parameters *n_estimators, max_features, max_depth* have been specified (see Stackoverflow, 2021).

Summary of the tuned hyperparameters

Extra Trees Classifier

Ideas taken from (Stackoverflow, 2021) and (Brownlee, How to Develop an Extra Trees Ensemble with Python, 2021)

- Imbalance parameter *class_weight='balanced'*
- Step 1: Tune *n_estimators, max_features*
- Step 2: Tune *max_depth, min_samples_split*

XGBoost

The idea was taken from (Jain, 2016)

- Imbalance parameter *scale_pos_weight=total_negative_examples/total_positive_examples*
- Step 1: Find initial version for *n_estimators*, *learning_rate*
- Step 2: Tune *max_depth, min_child_weight*
- Step 3: Tune *gamma*
- Step 4: Tune *subsample, colsample_bytree*
- Step 5: Tune *reg_alpha*
- Step 7: Finalize *n_estimators*, *learning_rate*

MLPCLassifier

Here we need additional scaling, so we used Scikit-Learn's Pipeline to orchestrate the model with a MinMaxScaler.

- Set *activation='logistic', solver='adam', learning_rate='adaptive'*
- Tune *hidden_layer_sizes*

Logistic regression

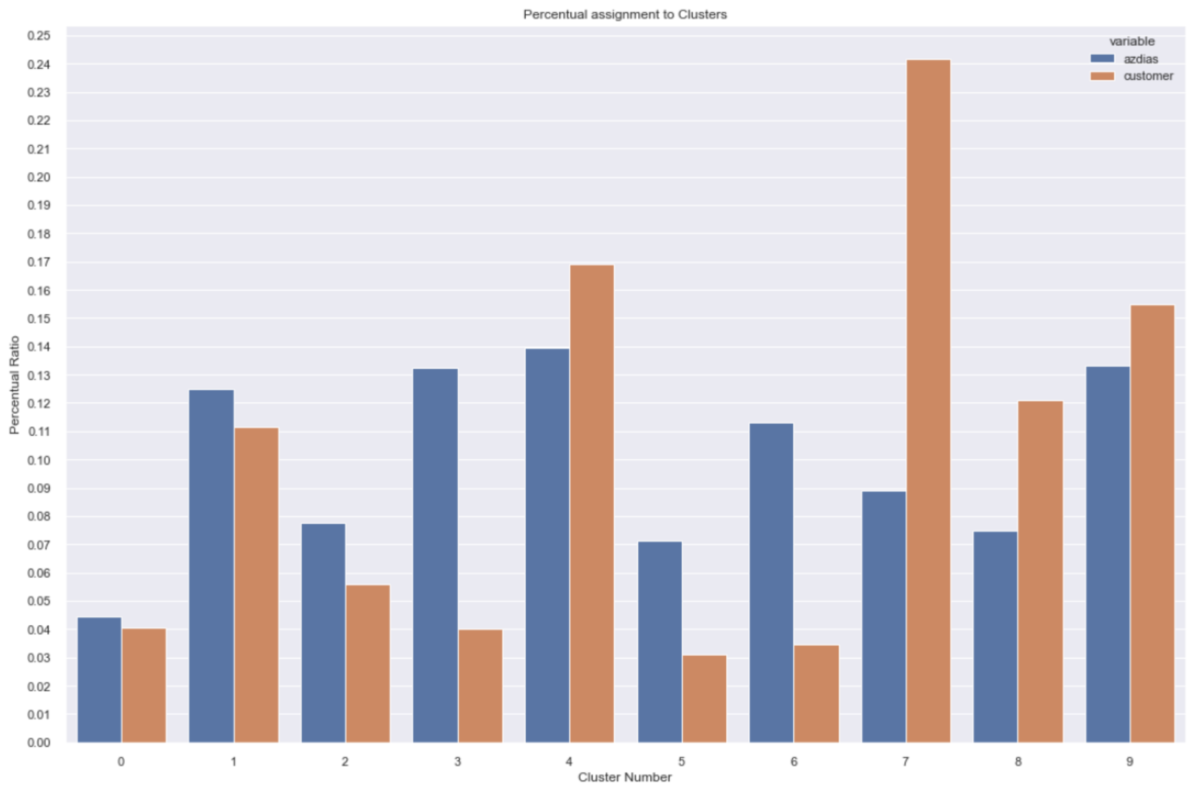
- Use Pipeline like with MLPClassifier
- Penalization parameter was set to 'l2'
- Tune *max_iter,C*

Results

Model Evaluation and Validation

Customer Segmentation

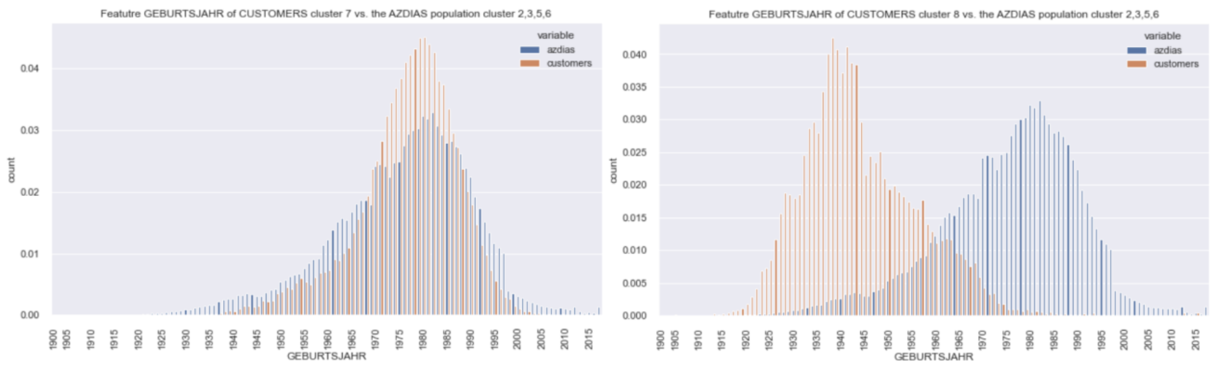
Training a K-Means with k=10 clusters on AZDIAS and applying on AZDIAS and CUSTOMERS gave the following plot:



The proportions of the clusters after applying our K-Means Model on AZDIAS and CUSTOMERS.

The clusters 0,1,2,3,5,6 were dominated by AZDIAS and the clusters 4,7,8,9 were dominated by the CUSTOMERS dataset. The analysis approach I took was to consider samples from dominated AZDIAS clusters as typical non-customers and samples from CUSTOMERS-dominated clusters as typical customers. In addition, we focus only on the clusters that show a clear domination. That means the clusters 2,3,5,6 make up our non-customer population and 7,8 represent the typical customers. So I compared customers from cluster 7 and 8 separately against the 'whole' non-customer population (2,3,5,6). The comparison was made visually by comparing proportional histograms of the features identified as important due to the PCA factor-loadings. The most important features of the first five clusters were used for this purpose.

Two quite distinct customer groups could be identified from clusters 7 and 8. The feature GEBURTSJAHR gives a first impression.



The feature GEBURTSJAHR of CUSTOMERS-cluster number 7 and 8 vs. AZDIAS clusters 2,3,5,6

The obtained findings
Cluster 7:

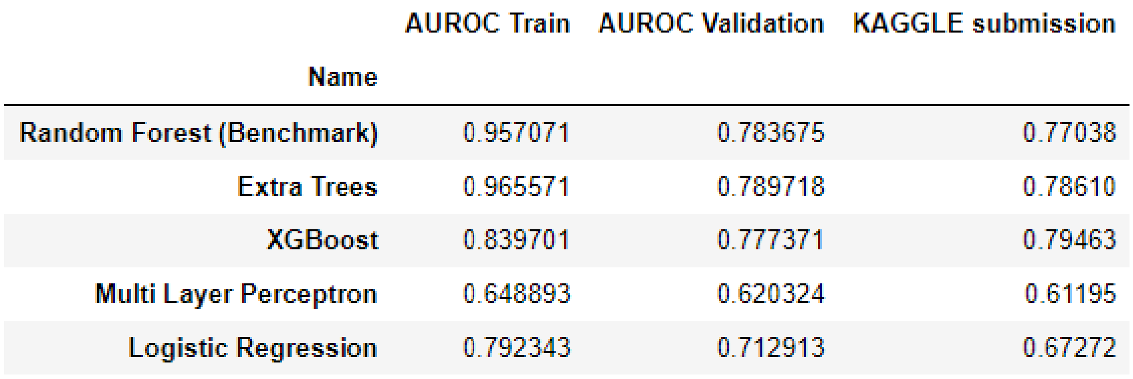
- Mid-aged people average year of birth 1976 , standard deviation approx. 11 years
- Mainly couples, a few multiperson households
- About 52% female
- Comfortable wealth
- High share of single high-income and earner couples and high earner
- All Western Germany
- Town size 20000 to 50000
- Common building type 3-5 family homes
- Low share of luxury cars in the neighborhood
- 10 to 20km to city center; 30 to 40km to next urban center
- High affinity towards events
- Not familiarly minded
- Very socially minded
- Very dreamily minded

Cluster 8:

- Middle to very low mobility with concentration on low
- Older Average Year of Birth 1944, standard deviation approx. 11 years
- Social status rather wealthy
- High income
- Approximately 71% male
- Slightly increased mail-ordering activity within the last 12 month (D19_GESAMT_DATUM)
- Consumption Type mainly gourmet
- Approx. 98% Western Germany
- Very high share of money savers
- Rather high online affinity
- Rather high share of 1-2 family homes
- High share of luxury cars
- Mainly 0 to 10km to city center; 0 to 20km to next urban center
- Rather religious
- Critically minded
- High affinity towards events
- Mainly not dreamily minded at all

Classification

Regarding the classification task, I have collected the AUROC scores for the training-set and the validation-set. Furthermore, I made a Kaggle-Submission for each model.



Name	AUROC Train	AUROC Validation	KAGGLE submission
Random Forest (Benchmark)	0.957071	0.783675	0.77038
Extra Trees	0.965571	0.789718	0.78610
XGBoost	0.839701	0.777371	0.79463
Multi Layer Perceptron	0.648893	0.620324	0.61195
Logistic Regression	0.792343	0.712913	0.67272

Surprisingly the Random Forest algorithm performed quite well even though the parameters were not systematically tuned, but they were chosen basing on the expierience made with the ExtraTreesClassifier and the two classifiers are pretty similar.

The ExtratreesClassifier performed slightly better than the benchmark model over all three scores.

Noticeably the RandomForestClassifier and the ExtraTreesClassifier showed the highest training score (95.7% and 96.6%), but also the highest gap between training and validation score. This might indicate that the model tends to over fit.

The scores of XGBoost look most consistent. Its training score (83.4%) is lower than the score of the Extra Trees Classifier, but the validation score (77.7%) is at the same level with Extra Trees (79.0%). Therefore, we can assume that XGBoost generalizes best among the contenders. Moreover, XGBoost had the best Kaggle-Score (79.5%) among all models.

The Logistic Regression showed a training score (79.2%) that was in XGBoosts range, but the validation score declined relatively strong (71.3%), what might indicate overfitting. Finally, the lowest value was reached with the Kaggle submission (67.3%).

The scores of the MLPClassifier (64.9%, 62.0%, 61.2%) also look very consistent, but are not competitive.

Taking into account the information from above, we can conclude that XGBoost is the winner:

- It generalizes best
  - The score on the validation set is similar to ExtraTrees
  - Highest Kaggle-score
- Was easy and efficient to train (see. *03_Arvato Project Workbook.ipynb*)

Conclusion

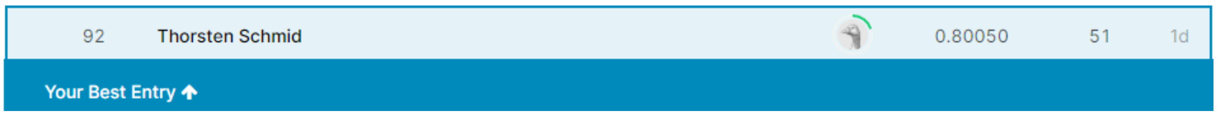
Reflection

The project was really challenging and it made a lot of fun. It was cool to get in touch with a real life problem. The main challenges were dealing with high dimensional categorical data and imbalanced learning. The key in this project is data exploration and feature engineering in order to improve data-quality. The more I knew about the data, the more I saw that there were some features with a bad data-quality biasing the data (See 'Mentionable Features' above). I did not know why but assumedly either it was a problem with the software used in the data-acquisition phase or the data was simply pre-processed before being given to us.

The undocumented features proved to be important, so in the real project we would need to ask the data-provider for a complete documentation.

Personally, I learned a lot regarding data science. Most times, I loved the project, but sometimes I hated it particularly when things did not work out like I expected. I experimented a lot, especially with multivariate imputation and resampling techniques.
Scikit-Learn's IterativeImputer is a great way to realize multivariate imputation. It helped me to reconstruct the GEBURTSJAHR-feature and it seemed to be a good choice for the imputation of the whole dataset. In addition, the possibility to exchange the underlying regression model is very helpful. IterativeImputer is still marked experimental, but if I did this on my own it would have been even more experimental and I hope it will be released as stable soon.

Regarding resampling I have played around with the *imblearn*-library (https://imbalanced-learn.org/), but this experiment was not successful.
Alternatively, I tried to up sample the minority class in a creative way by using a combination of clustering and a KNN search to add some CUSTOMERS data to the training set. I really got an improvement by applying XGBoost on the new data (Kaggle-score 0.80050). Unfortunately, I realized that I accidentially operated on scaled data, what is a typical beginner's mistake. So all the splits (train_test_split, cross-validations) may have led to biased input data for the algorithm. This fact made a promising approach a lucky strike.


92 Thorsten Schmid 0.80050 51 1d

Your Best Entry

I tried to do it the right way, but then I had no success, the results were even worse compared to perform the classification based on the 'original' training data. So finally after a lot of unsuccessful tries I gave up resampling and decided not to include this in my solution.

Improvement

Customer Segmentation

The silhouette analysis showed that K-Means was close to its limits or already beyond. I have also thought about checking out and trying other clustering algorithms like K-Modes and a Hierarchical DBSCAN (HDBSCAN). However, due to my lack of experience with those algorithms and the fact that I had already spent plenty of time with the segmentation task I decided to stop examining those approaches.

If other techniques than unsupervised learning would have been allowed in the segmentation task we could have tried several other interesting alternatives to PCA for feature reduction (or selection) like:

- Convert to Supervised learning task
    - Label CUSTOMERS with 1
    - Label AZDIAS with 0
    - Train a suitable model like ExtraTreesClassifier and use its feature importance for feature reduction
- Use an Autoencoder
- Do feature reduction using plain old statistics. Just do a chi-squared test between each feature in CUSTOMERS and AZDIAS to see which histograms (or more exactly distributions) differ most and keep those ones.

Classification

When talking about classification, we have some space left for improvements:

- Tested more parameter-combinations with GridSearchCV – enlarge the grid.
- Try randomized search on the parameters.
- Try BayesSearchCV. It uses Bayesian optimization to find parameter-values, meaning it is more efficient as it does not need the full search space like GridSearchCV (scikit-optimize, 2021).
- Try resampling techniques

Appendix A: List of removed features

The following features were removed after a sorrow revision:

Attribute	Meaning	Removal Reason
PRAEGENDE_JUGENDJAHRE	Formative youth-years	Values not plausible/does not cover full range. See Section 'Mentionable Features'
EINGEZOGENAM_HH_JAHR	Year when person moved in house or building	Does not cover expected range. Hardly any values below 1997.
EINGEFUEGT_AM	Date of insertion	Is meta information
MIN_GEBAEUDEJAHR	Year the building was first mentioned in our Database	Not concise, kind of meta information
ALTERSKATEGORIE_GROB	Age through prename Analysis	- Strange spacing (-1,0,1,2,3,4,9) - Meaning of value 9 'uniformliy distibuted' does not fit to the meanings of the other values
ALTERSKATEGORIE_FEIN	Undocumented	- Highly correlated to GEBURTSJAHR, - Use to reconstruct GEBURTSJAHR and delete the feature afterwards
LP_STATUS_FEIN	social status fine	- No sensible ordering detectable (would be a candidate for one hot encoding) - We will keep the corresponding rough categorization LP_STATUS_GROB instead - Reconstructable from LP_STATUS_GROB
LP_LEBENSPHASE_FEIN	lifestage fine	- High cardinality (40 categories). - We will keep the corresponding rough categorization LP_LEBENSPHASE_GROB instead
LP_FAMILIE_FEIN	family type fine	We will keep the corresponding rough categorization LP_FAMILIE_GROB instead
CAMEO_DEU_2015	CAMEO classification 2015 - detailled classification	- High cardinality (44 Categories) - Highly correlated to 'CAMEO_DEUG_2015 ' and 'CAMEO_INTL_2015'

		- Can be reconstructed afterwards - No sensible ordering detectable (would be a candidate for one hot encoding)
GEMEINDETYP	Undocumented	- Has strange values/spacing. - Values: {11,12,21,22,30,40,50}
KBA13_BJ_1999	share of cars built between 1995 and 1999 within the PLZ8	Regarding the construction year of the car the KBA_BJ_... features cover a range between 1995 and 2009. This range is too small
KBA13_BJ_2000	share of cars built between 2000 and 2003 within the PLZ8	See KBA13_BJ_1999
KBA13_BJ_2004	share of cars built in 2004 within the PLZ8	See KBA13_BJ_1999
KBA13_BJ_2006	share of cars built between 2005 and 2006 within the PLZ8	See KBA13_BJ_1999
KBA13_BJ_2008	share of cars built in 2008 within the PLZ8	See KBA13_BJ_1999
KBA13_BJ_2009	share of cars built in 2009 within the PLZ8	See KBA13_BJ_1999
KBA13_HALTER_20	share of car owners below 21 within the PLZ8	We will use the rougher KBA13_ALTERHALTER_... features instead
KBA13_HALTER_25	share of car owners between 21 and 25 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_30	share of car owners between 26 and 30 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_35	share of car owners between 31 and 35 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_40	share of car owners between 36 and 40 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_45	share of car owners between 41 and 45 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_50	share of car owners between 46 and 50 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_55	share of car owners between 51 and 55 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_60	share of car owners between 56 and 60 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_65	share of car owners between 61 and 65 within the PLZ8	See KBA13_HALTER_20
KBA13_HALTER_66	share of car owners above 65 within the PLZ8	See KBA13_HALTER_20
KBA13_CCM_1000	share of cars with less than 1000ccm within the PLZ8	We will use the rougher **KBA13_CCM_0_1400** instead

KBA13_CCM_1200	share of cars with 1000ccm to 1199ccm within the PLZ8	We will use the rougher **KBA13_CCM_0_1400** instead
KBA13_CCM_1400	share of cars with 1200ccm to 1399ccm within the PLZ8	We will use the rougher **KBA13_CCM_0_1400** instead
KBA13_CCM_1500	share of cars with 1400ccm to 1499ccm within the PLZ8	We will use the rougher **KBA13_CCM_1401_2500** instead
KBA13_CCM_1600	share of cars with 1500ccm to 1599ccm within the PLZ8	We will use the rougher **KBA13_CCM_1401_2500** instead
KBA13_CCM_1800	share of cars with 1600ccm to 1799ccm within the PLZ8	We will use the rougher **KBA13_CCM_1401_2500** instead
KBA13_CCM_2000	share of cars with 1800ccm to 1999ccm within the PLZ8	We will use the rougher **KBA13_CCM_1401_2500** instead
KBA13_CCM_2500	share of cars with 2000ccm to 2499ccm within the PLZ8	We will use the rougher **KBA13_CCM_1401_2500** instead
KBA13_CCM_3000	share of cars with 2500ccm to 2999ccm within the PLZ8	Clashes in range with feature **KBA13_CCM_2501**
KBA13_CCM_3001	share of cars with more than 3000ccm within the PLZ8	Clashes in range with feature **KBA13_CCM_2501**
KBA13_KMH_110	share of cars with max speed 110 km/h within the PLZ8	We will use the rougher **KBA13_KMH_0_140** instead
KBA13_KMH_140	share of cars with max speed between 110 km/h and 140km/h within the PLZ8	We will use the rougher **KBA13_KMH_0_140** instead
KBA13_KMH_180	share of cars with max speed between 110 km/h and 180km/h within the PLZ8	We will use the rougher **KBA13_KMH_140_210** instead
KBA13_KMH_210	Undocumented	We will use the rougher **KBA13_KMH_140_210** instead
KBA13_KMH_250	See KBA13_KMH_0_140	Clashes with feature **KBA13_KMH_211**
KBA13_KMH_251	See KBA13_KMH_0_140	Clashes with feature **KBA13_KMH_211**
KBA13_KW_30	share of cars up to 30 KW engine power - PLZ8	We will use the rougher **KBA13_KW_0_60** instead
KBA13_KW_40	share of cars with an engine power between 31 and 40 KW - PLZ8	We will use the rougher **KBA13_KW_0_60** instead
KBA13_KW_50	share of cars with an engine power between 41 and 50 KW - PLZ8	We will use the rougher **KBA13_KW_0_60** instead
KBA13_KW_60	share of cars with an engine power between 51 and 60 KW - PLZ8	We will use the rougher **KBA13_KW_0_60** instead

KBA13_KW_70	share of cars with an engine power between 61 and 70 KW - PLZ8	We will use the rougher **KBA13_KW_61_120** instead
KBA13_KW_80	share of cars with an engine power between 71 and 80 KW - PLZ8	We will use the rougher **KBA13_KW_61_120** instead
KBA13_KW_90	share of cars with an engine power between 81 and 90 KW - PLZ8	We will use the rougher **KBA13_KW_61_120** instead
KBA13_KW_110	share of cars with an engine power between 91 and 110 KW - PLZ8	We will use the rougher **KBA13_KW_61_120** instead
KBA13_KW_120	share of cars with an engine power between 111 and 120 KW - PLZ8	We will use the rougher **KBA13_KW_61_120** instead
KBA13_BMW	share of BMW within the PLZ8	The rougher KBA13_HERST_BMW_BENZ was kept
KBA13_MERCEDES	share of MERCEDES within the PLZ8	The rougher KBA13_HERST_BMW_BENZ was kept
KBA13_AUDI	share of AUDI within the PLZ8	The rougher KBA13_HERST_AUDI_VW was kept
KBA13_VW	share of VOLKSWAGEN within the PLZ8	The rougher KBA13_HERST_AUDI_VW was kept
KBA13_FORD	share of FORD within the PLZ8	The rougher KBA13_KRSHERST_FORD_OPEL was kept
KBA13_OPEL	share of OPEL within the PLZ8	The rougher KBA13_KRSHERST_FORD_OPEL was kept
KBA13_FIAT	share of FIAT within the PLZ8	The rougher KBA13_KRSHERST_EUROPA was kept
KBA13_PEUGEOT	share of PEUGEOT within the PLZ8	The rougher KBA13_KRSHERST_EUROPA was kept
KBA13_RENAULT	share of RENAULT within the PLZ8	The rougher KBA13_KRSHERST_EUROPA was kept
KBA13_MAZDA	share of MAZDA within the PLZ8	The rougher KBA13_KRSHERST_ASIEN was kept
KBA13_NISSAN	share of NISSAN within the PLZ8	The rougher KBA13_KRSHERST_ASIEN was kept
KBA13_TOYOTA	share of TOYOTA within the PLZ8	The rougher KBA13_KRSHERST_ASIEN was kept

KBA13_ANTG1	Undocumented	- We still have feature 'KBA05_ANTG1' (number of 1-2 family houses in the cell) which very probably could mean the same…
KBA13_ANTG2	Undocumented	Like KBA13_ANTG1
KBA13_ANTG3	Undocumented	Like KBA13_ANTG1
KBA13_ANTG4	Undocumented	Like KBA13_ANTG1
KBA13_BAUMAX	Undocumented	Like KBA13_ANTG1
KBA13_GBZ	Undocumented	Like KBA13_ANTG1
KBA05_ZUL1	share of cars built before 1994	Regarding the construction year of the car we only cover a range between 1995 and 2009. This range is too small
KBA05_ZUL2	share of cars built between 1994 and 2000	Regarding the construction year of the car we only cover a range between 1995 and 2009. This range is too small
KBA05_ZUL3	share of cars built between 2001 and 2002	Regarding the construction year of the car we only cover a range between 1995 and 2009. This range is too small
KBA05_ZUL4	share of cars built from 2003 on	Regarding the construction year of the car we only cover a range between 1995 and 2009. This range is too small

References

Aznar, P. (2020, June 17). *What is the difference between Extra Trees and Random Forest? | Quantdare*. Retrieved from quantdare.com: https://quantdare.com/what-is-the-difference-between-extra-trees-and-random-forest/

Brownlee, J. (2021, January 23). *A Gentle Introduction to Imbalanced Classification*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/what-is-imbalanced-classification/

Brownlee, J. (2021, May 24). *A Gentle Introduction to XGBoost for Applied Machine Learning*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/

Brownlee, J. (2021, June 05). *How to Develop an Extra Trees Ensemble with Python*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/extra-trees-ensemble-with-python/

Dabbura, I. (2021, January 24). *K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks*. Retrieved from towardsdatascience.com/: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

Géron, A. (2016). *"Hands-on Machine Learning with Scikit-Learn, Keras and Tensorflow", 2nd Edition.* O'Reilly.

Goel, A. (2021, January 24). *Are you solving ML Clustering problems using K-Means?* Retrieved from towardsdatascience.com: https://towardsdatascience.com/are-you-solving-ml-clustering-problems-using-k-means-68fb4efa5469

Hastie, T., Tibshirani, R., & Friedman, J. (2008). *The Elements of Statistical Learning.* Stanford CA: Springer.

Jain, A. (2016, March 1). *XGBoost Parameters | XGBoost Parameter Tuning*. Retrieved from www.analyticsvidhya.com: https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

Kuhn, M., & Johnson, K. (2013). *Applied Predictive Modeling.* New York: Springer.

Kumar, A. (2021, January 24). *KMeans Silhouette Score Explained With Python Example*. Retrieved from dzone.com: https://dzone.com/articles/kmeans-silhouette-score-explained-with-python-exam

Müller, A., & Guido, S. (2016). *"Introduction to Machine Learning with Python". 1st Edition.* O'Reilly.

Raschka, S. (2014, July 11). *About Feature Scaling and Normalization*. Retrieved from sebastianraschka.com: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

readthedocs. (07. February 2021). *How to Check for Outliers - datatest 0.11.1 documentation*. Von datatest.readthedocs.io: https://datatest.readthedocs.io/en/stable/how-to/outliers.html abgerufen scikit-learn. (2021, May 24). *1.17. Neural network models (supervised)*. Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

scikit-learn. (2021, April 17). *Importance of Feature Scaling*. Retrieved from scikit-learn.org: https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

scikit-learn. (30. January 2021). *sklearn.ensemble.ExtraTreesClassifier*. Von scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html abgerufen

scikit-learn. (2021, June 05). *sklearn.ensemble.RandomForestClassifier*. Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=randomforest#sklearn.ensemble.RandomForestClassifier

scikit-learn. (04. February 2021). *sklearn.impute.IterativeImputer*. Von scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.impute.IterativeImputer.html abgerufen

scikit-learn. (2021, January 24). *sklearn.metrics.silhouette_score*. Retrieved from scikit-learn.org: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html#sklearn.metrics.silhouette_score

scikit-optimize. (2021, June 05). *skopt.BayesSearchCV*. Retrieved from github.io: https://scikit-optimize.github.io/stable/modules/generated/skopt.BayesSearchCV.html

stackexchange. (2021, April 19). *machine learning - What are the implications of scaling the features to xgboost? - Cross Validated*. Retrieved from stackexchange.com: https://stats.stackexchange.com/questions/353462/what-are-the-implications-of-scaling-the-features-to-xgboost

Stackoverflow. (2021, June 05). *How do I solve overfitting in random forest of Python sklearn?* Retrieved from stackoverflow.com: https://stackoverflow.com/questions/20463281/how-do-i-solve-overfitting-in-random-forest-of-python-sklearn

Udacity. (2020, October 14). *Arvato Final Project*. Retrieved from Youtube: https://youtu.be/qBR6A0IQXEE

Wikipedia. (2020, October 24). *Arvato*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Arvato

Wikipedia. (2020, October 24). *Targeted advertising*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Targeted_advertising

Wikipedia. (2021, January 24). *Silhouette_(clustering)*. Retrieved from wikipedia.org: https://en.wikipedia.org/wiki/Silhouette_(clustering)

Yadav, J. (24. January 2021). *Selecting optimal number of clusters in KMeans Algorithm(Silhouette Score)*. Von medium.com: https://medium.com/@jyotiyadav99111/selecting-optimal-number-of-clusters-in-kmeans-algorithm-silhouette-score-c0d9ebb11308 abgerufen