Capstone Project

Udacity - Machine Learning Engineer Nanodegree

Author: Saverio Pulizzi
Date of Report: August, 2021

Customer Segmentation Report for Arvato Financial Services

1. Definition

Arvato Financial Services is a company operating in the mail-order sales business in Germany and is a subsidiary of Bertelsmann.

The company wants to grow their customer base by better targeting clusters of the general population with their marketing campaigns.

The problem that will be worked on this project is:

"Can Arvato Financial Services predict individuals who are more likely to convert to become new customers?"

The problem that this project is trying to solve is whether machine learning and in particular unsupervised learning and supervised learning could be used to predict those individuals from a general population dataset that are more likely to convert and to become new customers.

In order to predict individuals who are more likely to convert to become new customers, the project will involve using the following techniques:

- Unsupervised learning to identify segments of the general population that are very similar to historical customers of the company. In particular, Principal Component Analysis will be used for dimensionality reduction, followed by K-means clustering to obtain the needed clusters.
- Supervised learning to build a model that will be able to predict the probability that a targeted individual will convert to become a new customer. This is a

classification task and different models will be tried from simple (e.g. Logistic Regression) to more sophisticated (e.g. Random Forest Classifier)

Once the model will be trained, it will be used to make predictions on the campaign data from the Kaggle Competition and the score on the leader-board will be used as the evaluation metric.

Since in this project we are trying to solve a binary classification problem with a highly unbalanced training dataset (the number of individuals who responded to the campaign is much lower than the rest) the evaluation metric used for is AUC for the ROC curve, relative to the detection of customers from the mail campaign.

A ROC, or receiver operating characteristic, is a graphic used to plot the true positive rate (TPR, proportion of actual customers that are labeled as so) against the false positive rate (FPR, proportion of non-customers labeled as customers).

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

The AUC, or area under the curve, summarizes the performance of the model. If a model does not discriminate between classes at all, its curve should be approximately a diagonal line from (0, 0) to (1, 1), earning a score of 0.5.

A model that identifies most of the customers first, before starting to make errors, will see its curve start with a steep upward slope towards the upper-left corner before making a shallow slope towards the upper-right.

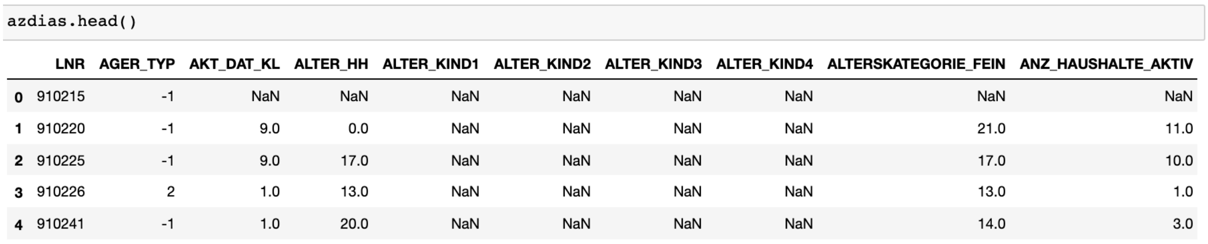
The maximum score possible is 1.0, if all customers are perfectly captured by the model first.

2. Analysis

Data Exploration

There are four data files associated with this project and 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:

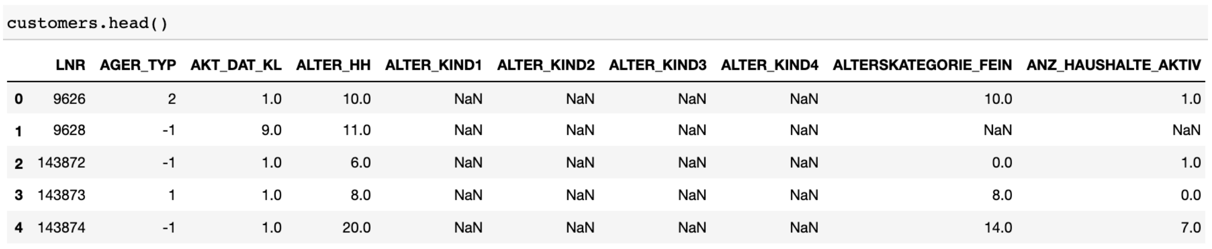
- Udacity_AZDIAS_052018.csv: Demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns).



```
azdias.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	11.0
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	10.0
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0	1.0
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	3.0

- Udacity_CUSTOMERS_052018.csv: Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns). This file contains three extra columns ('CUSTOMER_GROUP', 'ONLINE_PURCHASE', and 'PRODUCT_GROUP'), which provide broad information about the customers depicted in the file.



```
customers.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV
0	9626	2	1.0	10.0	NaN	NaN	NaN	NaN	10.0	1.0
1	9628	-1	9.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN
2	143872	-1	1.0	6.0	NaN	NaN	NaN	NaN	0.0	1.0
3	143873	1	1.0	8.0	NaN	NaN	NaN	NaN	8.0	0.0
4	143874	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	7.0

- Udacity_MAILOUT_052018_TRAIN.csv[1]: 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).

We use the information from the first two files to figure out how customers ("CUSTOMERS") are similar to or differ from the general population at large ("AZDIAS").

Then, we will use a combination of the customers, azdias and mailout_train datasets to make predictions on the test file ("MAILOUT_TEST"), predicting which recipients are most likely to become a customer for the mail-order company.

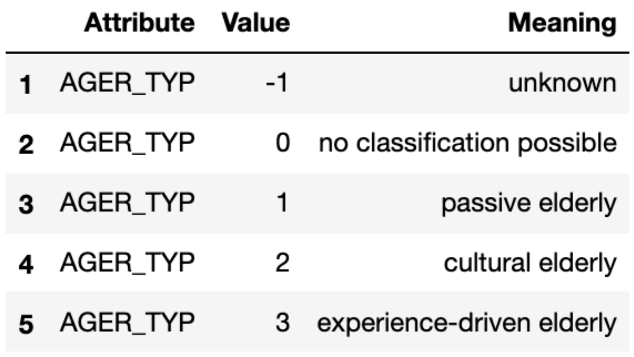
All of the remaining columns are the same between the three data files.
Metadata of the datasets can be found in the two Excel spreadsheets provided in the workspace:

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

[1] The original "MAILOUT" file included one additional column, "RESPONSE", which indicated whether or not each recipient became a customer of the company. For the "TRAIN" subset, this column has been retained, but in the "TEST" subset it has been removed; it is against that withheld column that your final predictions will be assessed in the Kaggle competition.

Dealing with “unknown” data

The excel spreadsheet (DIAS Attributes - Values 2017) including information about the data shows that the meaning of some of the collected data is “unknown”.



	Attribute	Value	Meaning
1	AGER_TYP	-1	unknown
2	AGER_TYP	0	no classification possible
3	AGER_TYP	1	passive elderly
4	AGER_TYP	2	cultural elderly
5	AGER_TYP	3	experience-driven elderly

The best strategy to deal with unknown data is to first convert them to null values and then to impute them with a measure of central tendency, one between median, mean and mode[2].

Dropping duplicates

After dealing with unknown values, we use a pandas general function (drop_duplicates) to clean the dataset of duplicate rows.
The function does not get rid of any rows for the general population dataset but when applied on the customers dataset it drops more than 20% of its rows and the total count changes from ~190k to ~150k.

Dealing with mixed data types

When downloading data, the below error about mixed data types pops up.

```
/opt/anaconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py:3146: DtypeWarning: Columns (19,20) have
mixed types.Specify dtype option on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

When exploring the information about the general population dataset, the first thing that can be noticed is that there are different data types.

[2] The imputing strategy depends on the type of data that needs to be imputed (categorical, numerical..)

```
azdias.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891221 entries, 0 to 891220
Columns: 367 entries, Unnamed: 0 to ALTERSKATEGORIE_GROB
dtypes: float64(267), int64(94), object(6)
memory usage: 2.4+ GB
```

Since it is not possible to train a model with object data types, the columns containing these values need to be explored further. The table below shows the columns including these values.

```
azdias.select_dtypes(include='object')
```

	CAMEO_DEU_2015	CAMEO_DEUG_2015	CAMEO_INTL_2015	D19_LETZTER_KAUF_BRANCHE	EINGEFUEGT_AM	OST_WEST_KZ
0	NaN	NaN	NaN	NaN	NaN	NaN
1	8A	8	51	NaN	1992-02-10 00:00:00	W
2	4C	4	24	D19_UNBEKANNT	1992-02-12 00:00:00	W
3	2A	2	12	D19_UNBEKANNT	1997-04-21 00:00:00	W
4	6B	6	43	D19_SCHUHE	1992-02-12 00:00:00	W

The following columns are explored to decide whether to keep them or to convert their categorical variables into dummy variables.

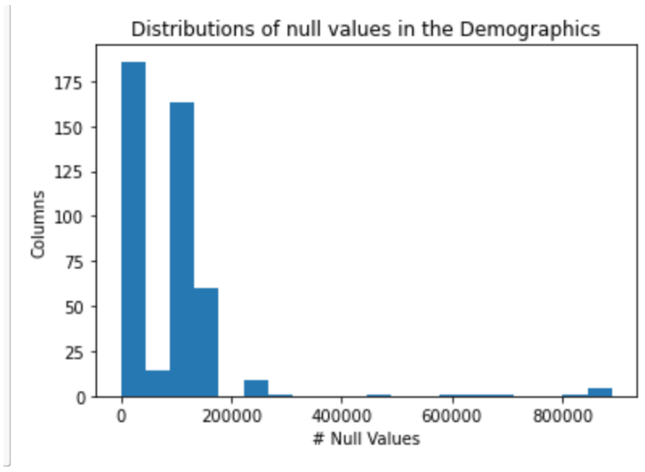
- CAMEO_DEU_2015: CAMEO classification 2015 - categorical variables we can split in dummy columns
- CAMEO_DEUG_2015: CAMEO classification 2015 - Uppergroup
- CAMEO_INTL_2015: CAMEO classification 2015 - international typology
- D19_LETZTER_KAUF_BRANCHE: categorical variables we can transform in dummy column
- EINGEFUEGT_AM: contains dates with timestamp (to drop_)
- OST_WEST_KZ: flag indicating the former GDR/FRG (-1, W, O) needs to be encoded with numbers

Moreover, the column LNR which contains unique values can be set as the dataframe index and the 3 extra columns in the customer dataset can be dropped.

Once the categorical variables are fixed, the data exploration part continues to identify the best strategy to clean null values.

Cleaning null values

A data driven approach is used to decide which columns need to be dropped.
In particular, the bar chart below is used to study the distributions of null values in the general population dataset.



Based on best practices, we pick a threshold of 30% and we use it to drop columns with more than 30% of missing values.

```
columns_to_drop = [s for s, v in (azdias.isnull().sum() > azdias.shape[0] * 0.30).items() if v]
print(columns_to_drop)
```

```
['AGER_TYP', 'ALTER_HH', 'ALTER_KIND1', 'ALTER_KIND2', 'ALTER_KIND3', 'ALTER_KIND4', 'EXTSEL992', 'KBA05_BAUMAX', 'KK
_KUNDENTYP', 'TITEL_KZ']
```

The result of this exploratory data analysis is a clean dataset, with a total of 432 features, 69 features more than the initial dataset (categorical columns converted into dummy variables) and ready to be preprocessed further imputing and scaling its values before applying the principal components analysis.

3. Implementation

In this section, we will provide details about the technical implementation.

Imputing remaining null

The remaining null values have been imputed using the median value of each column. The median is chosen over mean and mode since the biggest part of the dataset is made of ordinal data. However, a more complete impute strategy should have been designed depending on the type of data and applying mean to pure numerical data, mode on nominal data and median only to ordinal data.

Feature scaling

After imputing the remaining null values, the two datasets (customers and general population) contain only numerical data so we implemented a SimpleScaler to standardize the scale of the numerical columns in order to consistently compare the values of different features and correctly apply Principal component analysis. StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way.

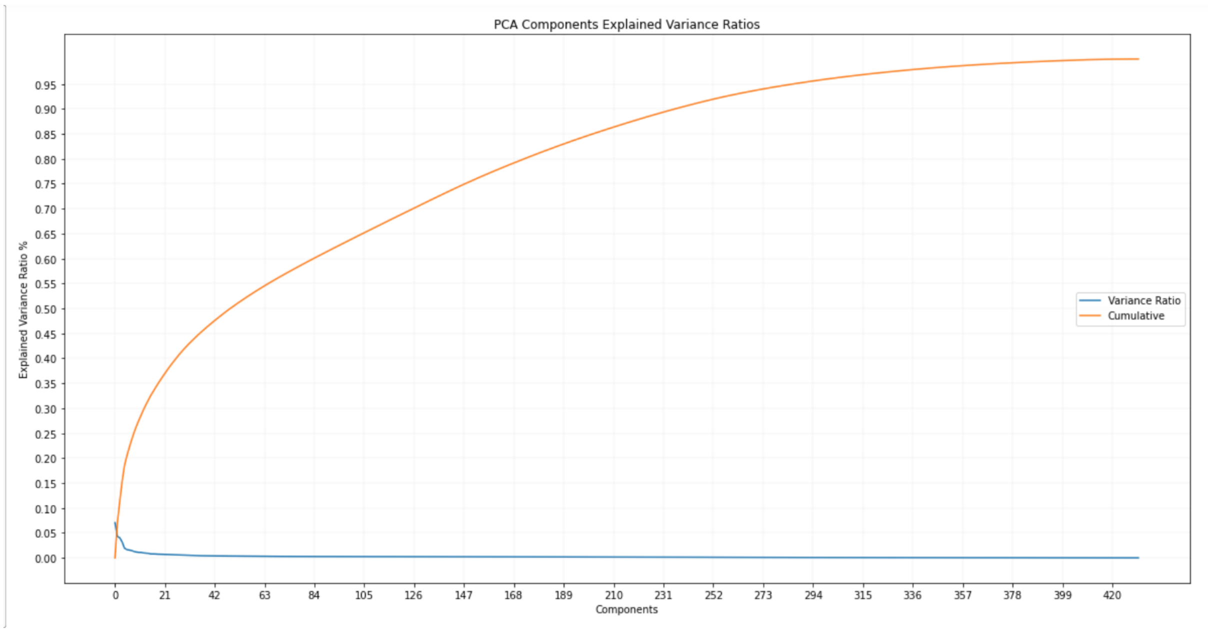
PCA

The principal component analysis has been implemented using sklearn's PCA class.

PCA allowed us to find the vectors of maximal variance in the data. We decided to start without setting any parameters so as to visualise on a line chart the explained variance by each principal component.

We plotted the cumulative explained variance function using matplotlib's plot() .

The pca without parameters returned all the 432 components (same as n. of features)
The plot showed that over 80% of the variance is explained by the first 170 components, 90% by the first 240 components.

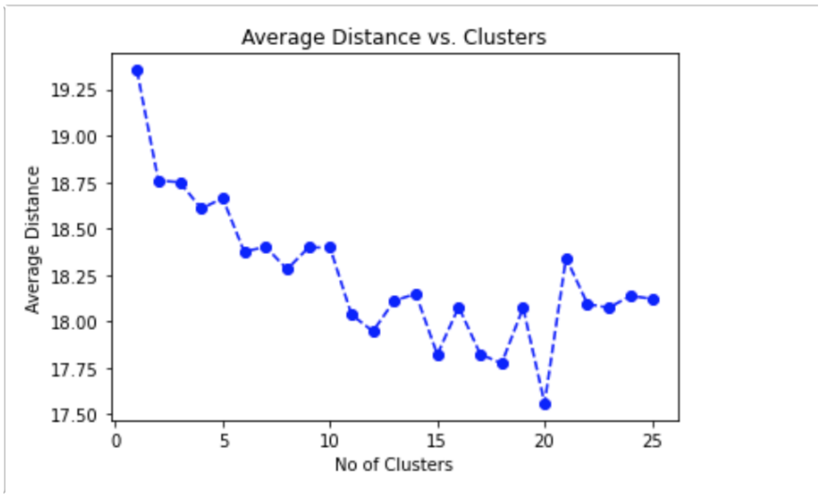


Clustering

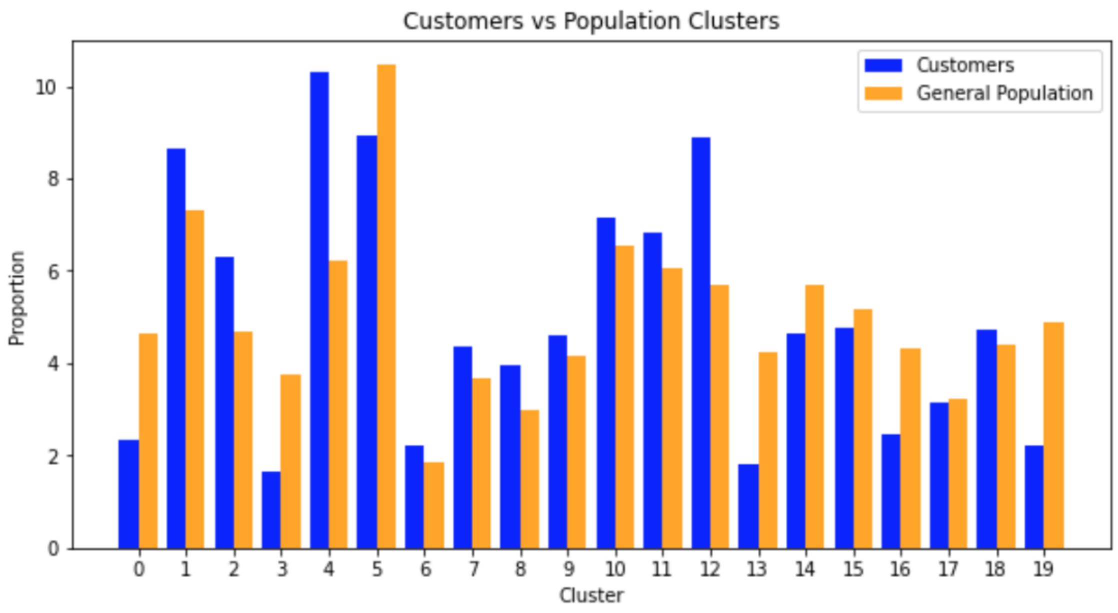
We applied k-means clustering to the dataset and used the average within-cluster distances from each point to their assigned cluster's centroid to decide on a number of clusters to keep.

- We used sklearn's KMeans class to perform k-means clustering on the PCA-transformed data.
- We computed the average difference from each point to its assigned cluster's center.

- We performed the above two steps for a number of different cluster counts.
- We used the elbow method to select a final number of clusters to use, re-fit a KMeans instance to perform the clustering operation. We obtain the cluster assignments for the general demographics data.

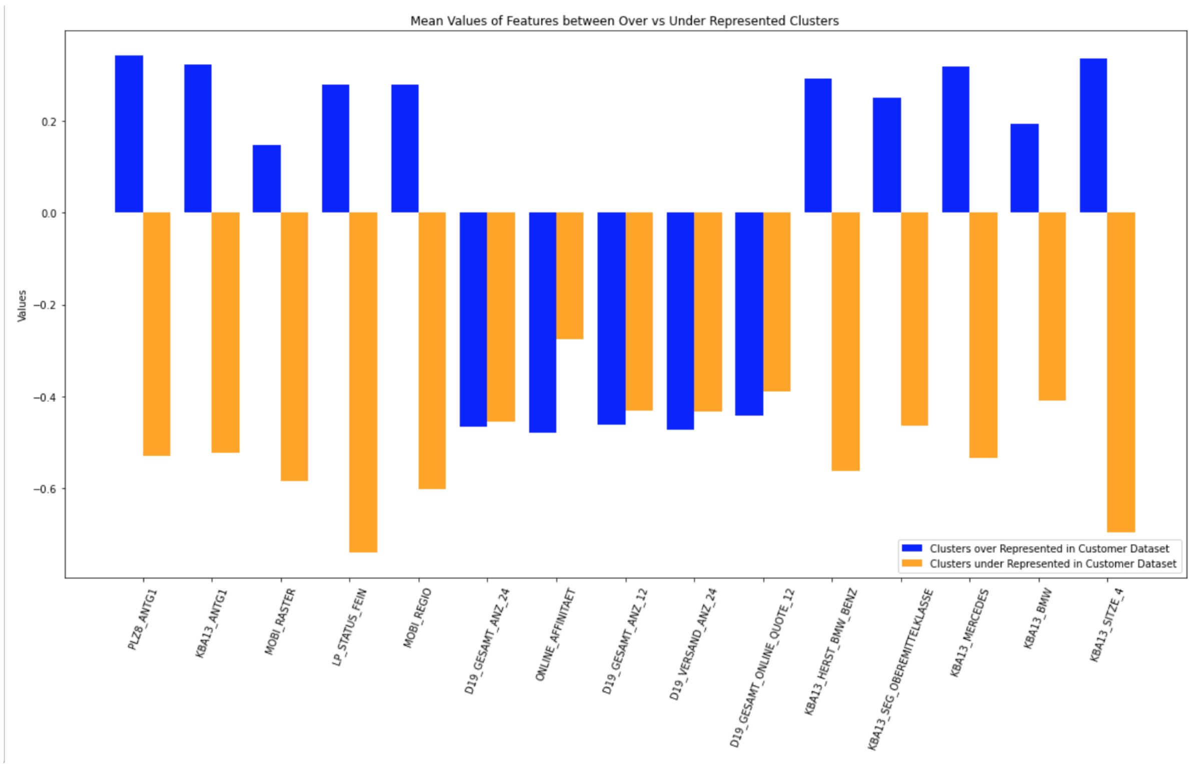


We selected 20 as the number of clusters that minimise the average distance, and we built a column chart to compare the percentage of individuals belonging to each cluster for both the customers and general population dataset.



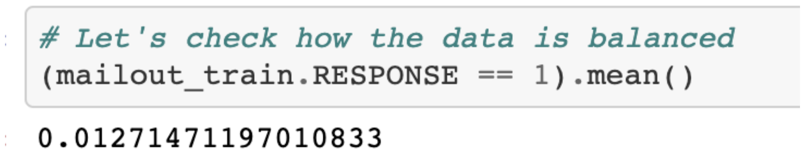
We could see that clusters 3 and 14 are significantly more represented in the customer dataset while the other clusters are more represented in the general population especially 2 and 7.

We used the .inverse_transform() method of the PCA and StandardScaler objects to transform centroids back to the original data space and interpret the retrieved values directly. In this way, we were able to analyse the principal components making up each cluster and defining the characteristics of the typical customers.



Algorithms and Techniques

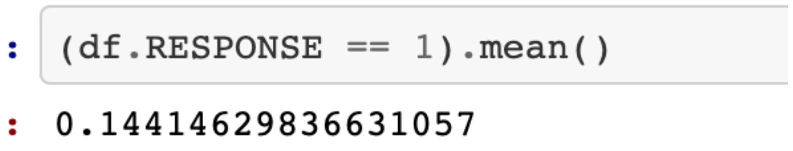
The mailout_train dataset provided is highly unbalanced indeed only 1.24% of the people responded to the mailout campaign.



```
# Let's check how the data is balanced
(mailout_train.RESPONSE == 1).mean()
```

```
0.01271471197010833
```

To make sure we have enough data to train our model, we build a training dataset made of the *customers*, *azdias* and *mailout_train* dataset combined. In this way, the underrepresented class increases by 13%.



```
(df.RESPONSE == 1).mean()
```

```
0.14414629836631057
```

We split the training set in *X* features set and *y* the target variable.

We train our model on 100% of the training data and we do not use any cross validation technique. We will test the model accuracy directly on the provided test set *mailout_test*.

Since our model training needs to be done on tabular data, we used Random Forest and XGBoost.

- **XGBoost**. Gradient boosting re-defines **boosting** as a numerical optimization problem where the objective is to minimize the loss function of the model by adding weak learners using gradient descent. Gradient descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. As gradient boosting is based on minimizing a loss function, different types of loss functions can be used. Gradient boosting does not modify the sample distribution as weak learners train on the remaining residual errors of a strong learner (i.e., pseudo-residuals). By training on the residuals of the model, it gives more importance to misclassified observations. Intuitively, new weak learners are added to concentrate on the areas where the existing learners are performing poorly. The contribution of each weak learner to the final prediction is based on a gradient optimization process to minimize the overall error of the strong learner.
- **Random Forest**. Random Forest is a **bagging** technique that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset. Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

Tree-based models do not require feature scaling so we did not scale our training data. We compared performances on the test set of XGBoost and Random Forest classifiers.

One of the major **challenges** of our solution is that given a limited computation capacity and given a training dataset containing more than 1M rows, we had to compare the two models **with their default parameters** and we did not use cross validation and model tuning (for example with Grid Search CV) for a more detailed analysis.

Random forest with default parameters scored above 0.83 while XG boost scored 0.76. Therefore, we selected a random forest model to test tuning some parameters.
We tried to increase the number of estimators and the min_samples_leaf parameters (more trees in the forest could increase the accuracy since the single predictions are averaged out of a higher number of votes) without success.

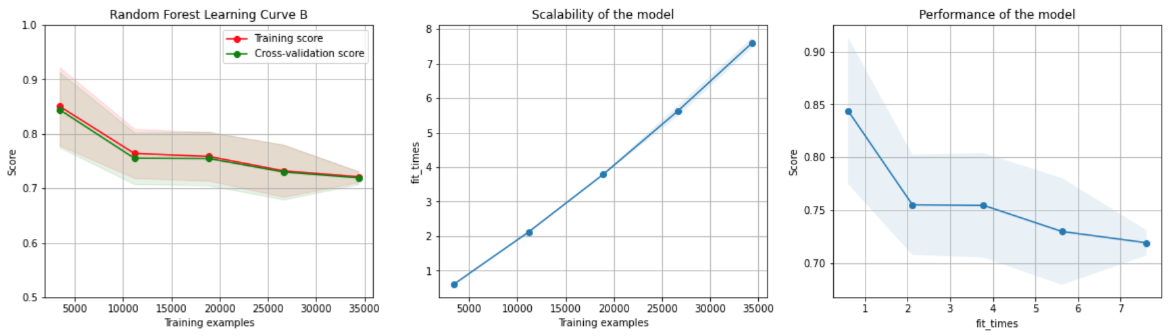
So, our final model selected is a random forest classifier with its default parameters.

Benchmark

In order to create a benchmark, we decided first to train a Random Forest model (with default parameters) on a sample of the training set using only the mailout_train dataset. The model scored on average 0.68 on cross validation.
The figure below shows the learning curve and the score of the trial.

```
plot_learning_curve(clf_B, 'Random Forest Learning Curve B', X_features.values, y_target.values, cv=cv, ylim=(0.5, 1),
```

```
<module 'matplotlib.pyplot' from '/opt/anaconda3/lib/python3.8/site-packages/matplotlib/pyplot.py'>
```



```
start = time()
res_rf = cross_val_score(clf_B, X_features.values, y_target.values, cv=cv, scoring='roc_auc', n_jobs=2)
end = time()
print("RF executed in {} seconds. CV average score: {}".format(end - start, res_rf.mean()))
```

```
RF executed in 23.61176609992981 seconds. CV average score: 0.6812054513802293
```

Justification

The Random Forest model on the final training dataset made of 1M rows performed 0.15 better than the benchmark of a random forest model training just on a sample of the dataset. Hence, we decide to pick random forest trained on a bigger dataset as our final model.

4. Results

The Customer Segmentation report shows that customers of the mail order company are positively correlated with indicators of wealth, they are top earners individuals who live in buildings/neighborhoods with a smaller number of households, they tend to be less mobile than the general population, drive luxury cars and shop mostly online.

Random Forest was the best supervised learning model to predict which individuals are most likely to convert into customers using a marketing campaign dataset with 432 estimators.

This model was used against a test dataset in the Kaggle competition, with a final score of 0.83386.

5. Conclusions

The results achieved are satisfactory and reflect a 7th position out of 500+ teams participating in the Kaggle competition.

The technical implementation adequately solves the problem of identifying a relevant customer segment and predicting whether or not an individual target of a marketing campaign would convert or not.

Different parts of the provided solution could be improved: imputing strategy, training dataset, model tuning, etc. For example when generating the training set, we could have assigned a 0 to the RESPONSE column only for those individuals clustered as not a likely customer. This could decrease possible noise created in the training data and improve the model performances.

Finally, a more precise model tuning using for example cross validation and Grid search for better hyperparameters could be introduced together with a more customised imputing strategy.

6. References

- [1] Udacity+Arvato: Identify Customer Segments. In Kaggle. Retrieved from: https://www.kaggle.com/c/udacity-arvato-identify-customers
- [2] Customer churn prediction in telecom using machine learning in big data platform Retrieved from: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0191-6
- [3] Receiver Operating Characteristic. In Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
- [4] Arvato. In Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/Arvato#cite_note-3
- [5] Customer Segmentation (Online Definition). In SearchCustomerExperience. Retrieved from: https://searchcustomerexperience.techtarget.com/definition/customer-segmentation
- [6] pandas.get_dummies. In Pandas Documentation. Retrieved from: https://pandas.pydata.org/docs/reference/api/pandas.get_dummies.html
- [7] Impute Missing Values with SciKit's Imputer – Python. In Medium. Retrieved from: https://medium.com/technofunnel/handling-missing-data-in-python-using-scikit-imputer7607c8957740
- [8] StandardScaler. In Scikit-Learn Documentation. Retrieved from:

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

- [9] Understanding Principal Component Analysis. In Medium. Retrieved from: https://medium.com/@aptrishu/understanding-principle-component-analysis-e32be0253ef0
- [10] K-Means Clustering in Python: A Practical Guide. In Real Python. Retrieved from: https://realpython.com/k-means-clustering-python/
- [11] Classification: ROC Curve and AUC. In Google Developers. Retrieved from: https://www.datanovia.com/en/wp-content/uploads/dn-tutorials/002-partitionalclustering/images/partitioning-clustering.png
- [12] RandomForestClassifier. In Scikit-Learn Documentation. Retrieved from: https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- [13] Understanding Random Forests Classifiers in Python. In Datacamp Community. Retrieved from: https://www.datacamp.com/community/tutorials/random-forests-classifier-python
- [14] GradientBoostingClassifier. In Scikit-Learn Documentation. Retrieved from: https://scikitlearn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html
- [15] Gradient Boosting with Scikit-Learn, XGBoost, LightGBM, and CatBoot. In Machine Learning Mastery. Retrieved from: https://machinelearningmastery.com/gradient-boostingwith-scikit-learn-xgboost-lightgbm-and-catboost/
- [16] Github repository: Machine-Learning-Engineer-Nanodegree-Program-Udacity. Retrived from: https://github.com/AilingLiu/Machine-Learning-Engineer-Nanodegree-Program-Udacity/blob/master/5.1%20Capstone%20Project%20-%20Arvato%20Finance/Customer_Segmentation_Report.ipynb
- [17] Github repository: Arvato-MLProject. Retrieved from: https://github.com/dilayercelik/Arvato-MLProject
- [18] SVC. In Scikit-Learn Documentation. Retrieved from: https://scikitlearn.org/stable/modules/generated/sklearn.svm.SVC.html
- [19] Elbow Method for Optimal Value k in K-Means. In GeeksforGeeks. Retrieved from: https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/
- [20] XGBoost vs Random Forest. Retrieved from: https://medium.com/geekculture/xgboost-versus-random-forest-898e42870f30