Capstone – Report

Customer Segmentation – Arvato

Clement Palfroy

DEFINITION

Introduction

Arvato is an internationally active services company that develops and implements innovative solutions for business customers from around the world. These include SCM solutions, financial services and IT services, which are continuously developed with a focus on innovations in automation as well as data and analytics.

One of Arvato missions is to help customers extract valuable insights from massive datasets. Those insights are then used to make informed and mathematically based decisions. For many years now, marketing managers searched for ways to optimize their ads targeting. When large datasets are available, Machine Learning represents an ideal tool to identify hidden patterns in customer behaviour (Badea, 2014; Kim & Ahn, 2009).

In this project, Arvato is helping a Mail-order company to understand its customers segments in order to identify with high accuracy, their next customers. General demographic data will be analysed to better understand customer characteristics. From there, a ML model will be built and trained to predict the likelihood of an individual to become a customer.

Resources

Four datasets will be used in this project:
Udacity_AZDIAS_052018: The demographic data for the general population of Germany. 891211 persons (rows) X 366 features (columns).

Udacity_CUSTOMERS_052018: The demographic data for the company customers. 191652 persons (rows) X 369 features (columns).

Udacity_MAILOUT_052018_TRAIN: The demographic data for individuals who were targeted by the marketing campaign, and if they became client or not (1/0). 42982 persons (rows) X 367 features + label (columns).

Udacity_MAILOUT_052018_TEST: The demographic data for other individuals who were also targeted by the marketing campaign, but without the information if they became client or not. 42833 persons (rows) X 366 features (columns).

Each row of the demographics files represents a single person, and includes 5 levels of information: Personal (eg gender, financial/heath/social status, mind affinity …) , Household (family size, income…), Microcell (CAMEO classifications, most common car manufacturer…), Macrocell (population density, distance from city center…) and finally Community (inhabitants, unemployment rate…).

Those datasets were provided by Arvato in the context of the Machine Learning ND from Udacity.

Problem Statement

The goal of this project is to determine the potential of an individual (given its demographic characteristics) to become client of a given company.

How are the company clients different than the rest of the population?
The demographic data of the general population was studied with the help of unsupervised learning algorithms. Segments were defined. Then, it was applied to the customer data to discover what features are specific to the company customer.

How can we use the previous findings to better predict the likelihood of a person to become a customer?
Two different learning algorithms were built and trained using marketing results from previous campaign.

Metrics

Before creating the models, the training data was analysed. The class label distribution was found to be highly imbalanced, more precisely 42430 examples were labelled “0” (or not responsive) against only 532 examples labelled “1” (or responsive).

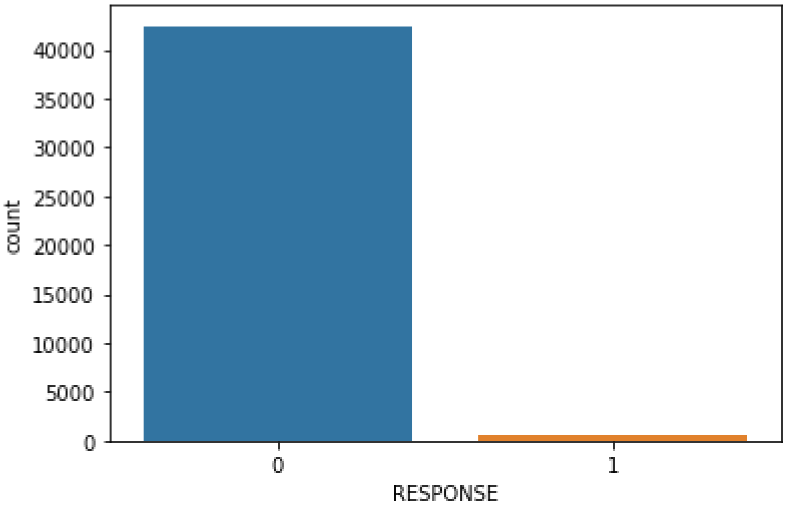


Figure 1: Count plot of the training label dataset.

In other words, 98% of the targeted individuals did not became customers. Accuracy would have been a bad metric to choose for this problem since the accuracy might always be more than 98% even if the model predicts all zeros. To address this imbalance while evaluating the model we need to chose a metric which will take this class imbalance into accounts. The usual metrics used for imbalanced binary classification are *Precision and Recall* or *Area under Receiver Operating Curve (AUROC)*.

Specifically, a ROC curve is a way to quantify the trade-off between true and false positive rate (T/F PR). As ultimately, the Kaggle competition associated to this project uses AUROC, we used it to train and evaluate our models.

ANALYSIS

In order to feed the datasets to PCA transform and supervised models, much data pre-processing had to be performed. This data cleaning was performed step by step with the help of (relatively) modular functions. This way, it became easy to join all these functions into a single one to efficiently clean similar datasets. More specifically, it includes:

1) Addressing specific unknown representations

A lot of missing values have been encoded in a specific way depending on the features. Some columns labelled their unknown values “-1” while other labelled them “0” or “9”. The attribute-values excel file were used to automate the process of identifying specific representations of unknown values and replace them with np.NaN. No less than 232 columns were concerned.

2) Deleting data scarce columns

Before dropping or imputing rows with unknown values, the percentage of missing values in columns was investigated.

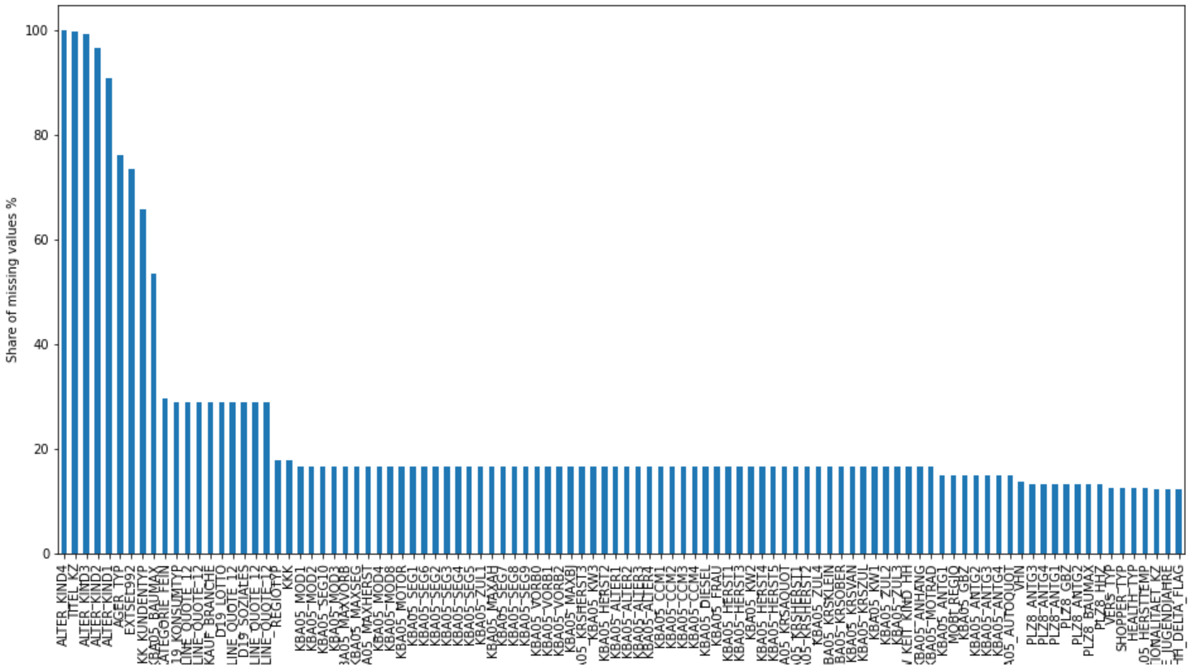


Figure 2: Top 100 missing columns for the general population dataset

The results were very similar for both the general and customer dataset. It was decided to drop all columns with more than 32% missing data. The same 9 columns were dropped in each dataset.

3) Deleting data scarce rows

In a similar way, the number of missing values per row was investigated.

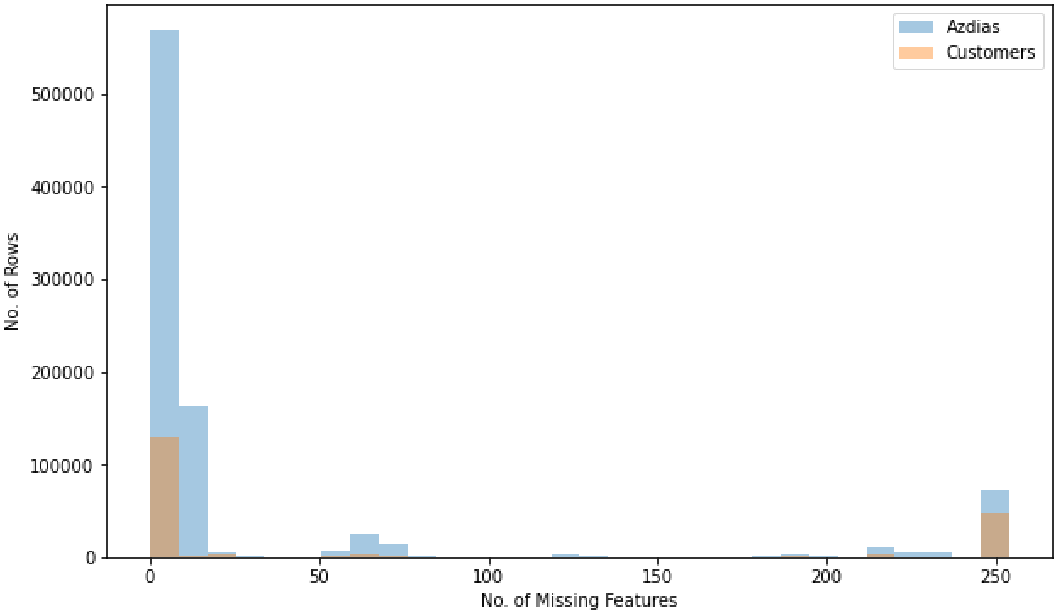


Figure 3: Number of rows VS Number of missing features

As we can see, most rows (and again, for both datasets) have less than 50 missing features. Also, a non-negligible part misses a big quantity of values (~250 : 70% of features).It was decided to drop all rows with more than 50 features missing. For the general population dataset, 153933 rows were dropped (~ 17%).

4) Re-encoding

Additionally, specific features had to be re-encoded, for example:

- “EINGEFUEGT_AM” string value which contains the data of an event. This column was reduced to the year of each date.
- “OST_WEST_KZ” string value which tells us in which part of Germany the person was before the unification (West or East) .This column was simply binary encoded.
- The columns “CAMEO_DEU_2105” and “D19_LETZTER_KAUF_BRANCHE” are filled with many different categorical values. To one-encode them all would have increase way too much the number of parameters/features. For simplicity’s sake, those columns were dropped.

5) Imputation

We previously dropped columns and rows with missing values based on defined threshold. Yet, we still have missing values. Common approaches to solve this problem include filling them with the mean of the values or the most common values in their associated columns. For this project, the later was chosen.

6) Standardization

We can now centre and scale each feature independently (if a feature has a variance that is orders of magnitude larger than others, it might dominate the objective function and make the estimator unable to learn from other features correctly).To do so, we used the StandardScaler from sklearn. With it, the standard score of a sample x is calculated as:

$z = (x - u) / s$

Where u is the mean of the training samples and s is the standard deviation.

Feel free to explore the notebook “1) Data exploration and pre-processing” to see how each data transformation have been implemented in code.

METHODOLOGY

Customer segmentation

Now that our data is clean and ready, our goal is to divide the population into segments. This way, we will be able to highlight features which might characterize the customers.

As the number of features in the data is relatively high, we can verify the variance explained by each feature in the dataset. By using a dimensionality reduction technique, we can effectively reduce the number of features which do not vary much in the data. Since we cannot go through each and every feature ourselves to decide if the feature is varying, we can use a statistical approach to find how much variance is explained by each feature. One such algorithm is Principal Component Analysis (PCA).

After data cleaning, we are now left with 354 features, which is still a lot! The first step was to fit a PCA instance (using sklearn) on the general population dataset and explore the variance of the created components.

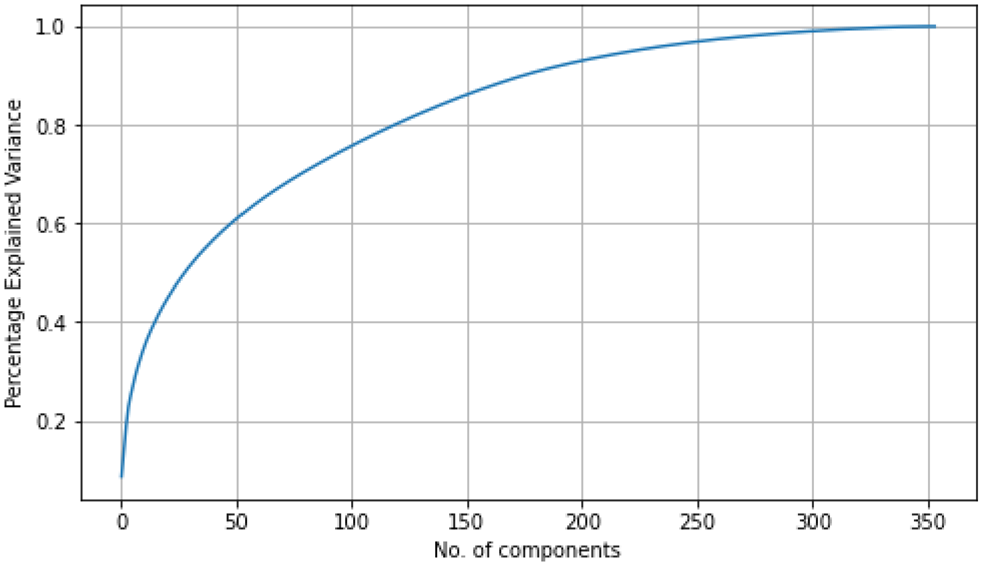


Figure 4: PCA explained variance ratio VS Number of components

It turns out that only half of the features (PCA components) can explain 90% of the variance. We therefore decided to transform our datasets into the 175 top components of our PCA.

Now that the total quantity of data has been drastically reduced, it is time to divide our datasets into clusters. With the help of the reduced features, the K-means algorithm will separate the general population into a specified number of clusters. Those clusters will contain information to understand the similarities in the general population and customer data.

The number of cluster (k) is the most important hyperparameter of this unsupervised learning algorithm. One method for choosing a "good" k, is to choose based on empirical data. A bad k would be one so high that only one or two very close data points are near it, and another bad k would be one so low that data points are really far away from the centres.
We want to select a k such that data points in a single cluster are close together but that there are enough clusters to effectively separate the data. We can approximate this separation by measuring how close your data points are to each cluster centre, the average centroid distance between

cluster points and a centroid. After trying several values for k, the centroid distance typically reaches some "elbow"; it stops decreasing at a sharp rate and this indicates a good value of k. In order to find it, several k values were tested, and the centroid distance calculated.

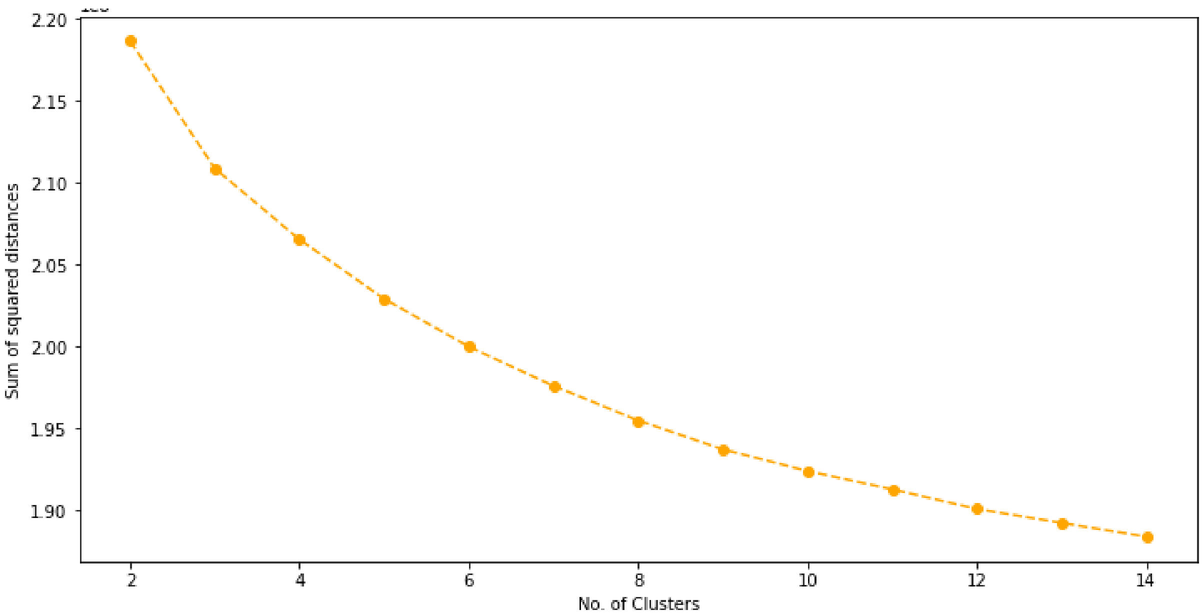


Figure 4: Cluster distance VS Number of clusters

Unfortunately, no obvious "elbow" was found. However, we could say that after 8, the sum of squared distances decreases very slowly. This indicates that there is enough separation to distinguish the data points in each cluster, but also that we included enough clusters so that the data points aren’t extremely far away from each cluster.

An “8-means” algorithm was therefore fitted on the general population dataset and apply to both datasets. This way, each individual was assigned to one of the 8 clusters. Here is the distribution of the general population and customers within those clusters.

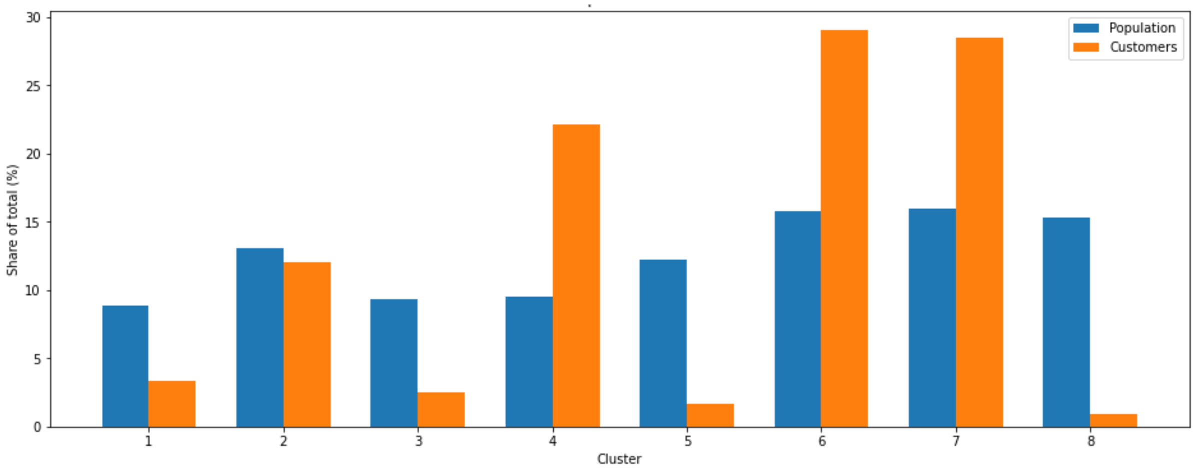


Figure 5: Share of clusters in general population VS customers

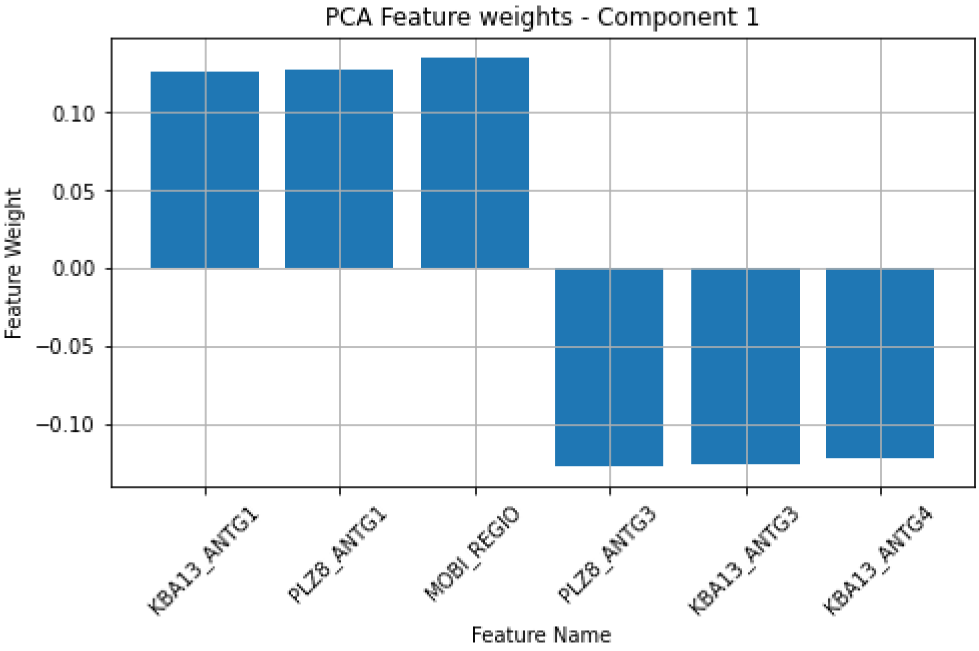
The plot shows how clusters are distributed across both datasets. Clusters are equally spread across the general population and show only small differences in size. This can be expected from dataset that represents the general population very well. In contrast, the customers dataset has a large imbalance across clusters. Cluster 4, 6 and 7 stand out specifically: ~75% of customers are mapped within those 3 clusters.

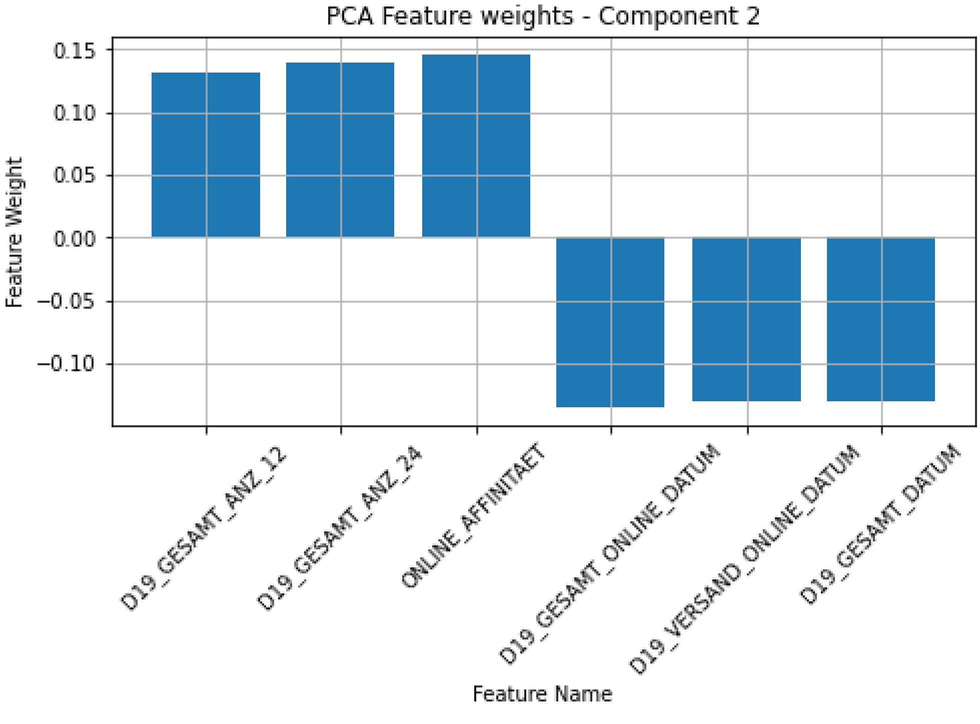
Now that we know customers tend to fall into 3 specific clusters, let's try to understand what they represent. More specifically, we need to look at which PCA components make up those clusters.

It was found that:

- Cluster 4 is highly positively correlated to the 3rd PCA component and negatively correlated to the 2nd component.
- Cluster 6 is highly positively correlated to the 1st PCA component and negatively correlated to the 2nd component.
- Cluster 7 is highly positively correlated to the 1st PCA component.

In other words, customer seems positively related to the 1st and 3rd PCA components and negatively correlated to the 2nd one. However, the results are still not interpretable (PCA components doesn't have real-life meaning all by themself). Instead, we will have to decompose and analyze those component make-ups just like we did with clusters. The following figures show the composition of the aforementioned PCA components.


PCA Feature weights - Component 1
Feature Weight
0.10
0.05
0.00
−0.05
−0.10
KBA13_ANTG1
PLZ8_ANTG1
MOBI_REGIO
PLZ8_ANTG3
KBA13_ANTG3
KBA13_ANTG4
Feature Name


PCA Feature weights - Component 2
Feature Weight
0.15
0.10
0.05
0.00
−0.05
−0.10
D19_GESAMT_ANZ_12
D19_GESAMT_ANZ_24
ONLINE_AFFINITAET
D19_GESAMT_ONLINE_DATUM
D19_VERSAND_ONLINE_DATUM
D19_GESAMT_DATUM
Feature Name

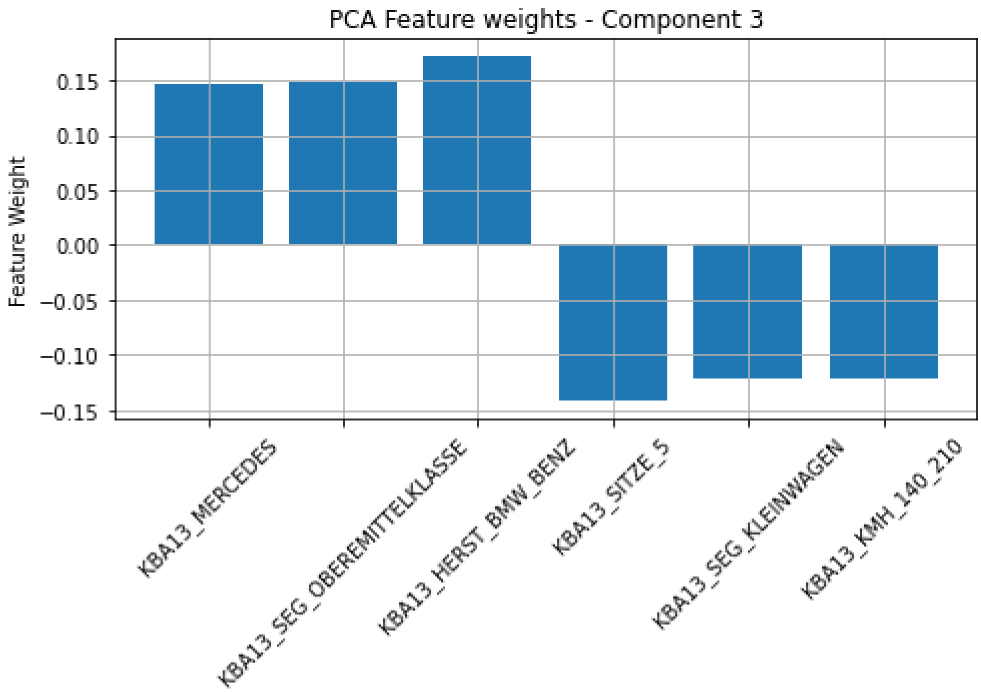


Figure 6: Top 3 PCA components makeup.

Unfortunalty, not all primary components had a clear documentation which would have help the interpretation.

Ultimately, by focusing on the interpretable features, how they relate to the PCA components, and how those PCA components relate to the clusters, the following conclusions were drawn:

- Customers tend live in neighborhoods with a small number of houses.
- They tend to live in small families.
- They tend to travel a lot.
- There neighborhoods count a big share of BMW & MERCEDES.

I would tend to think that much of those characteristics are more of less direct consequences of a financially secure situation. A deeper analyse could help understand in detail what kind of people constitute each cluster. This would help understand the existing customers and their behaviour according to demographics data.

Feel free to explore the notebook “2) Dimensionality reduction and clustering” to see how the PCA transform and the K-means algorithm have been implemented.

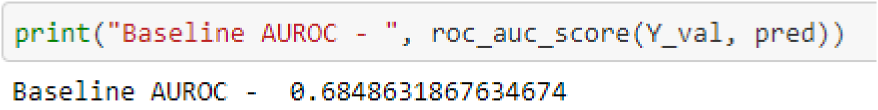
Customer acquisition

Now, our goal is to use supervised machine learning to predict the likelihood of an individual to become a customer. The file “Udacity_MAILOUT_052018_TRAIN.csv” contains the same features as the general population and customers demographic data plus an extra column “RESPONSE” that states whether a person became a customer following an ad campaign. The data was cleaned using the “all-in-one” function built while investigating the datasets.

20% of the dataset was reserved for validation. Each model was assessed based on its AUROC score on the validation set.

Benchmark

First, we created a simple logistic regression model to set a benchmark performance that we will aim to improve. Here is the result:



```
print("Baseline AUROC - ", roc_auc_score(Y_val, pred))
```

```
Baseline AUROC -  0.6848631867634674
```

Our baseline is therefore an AUROC of 68%.

The rest of the models investigation was performed exclusively on SageMaker instances to reduce training time.

XGBoost model

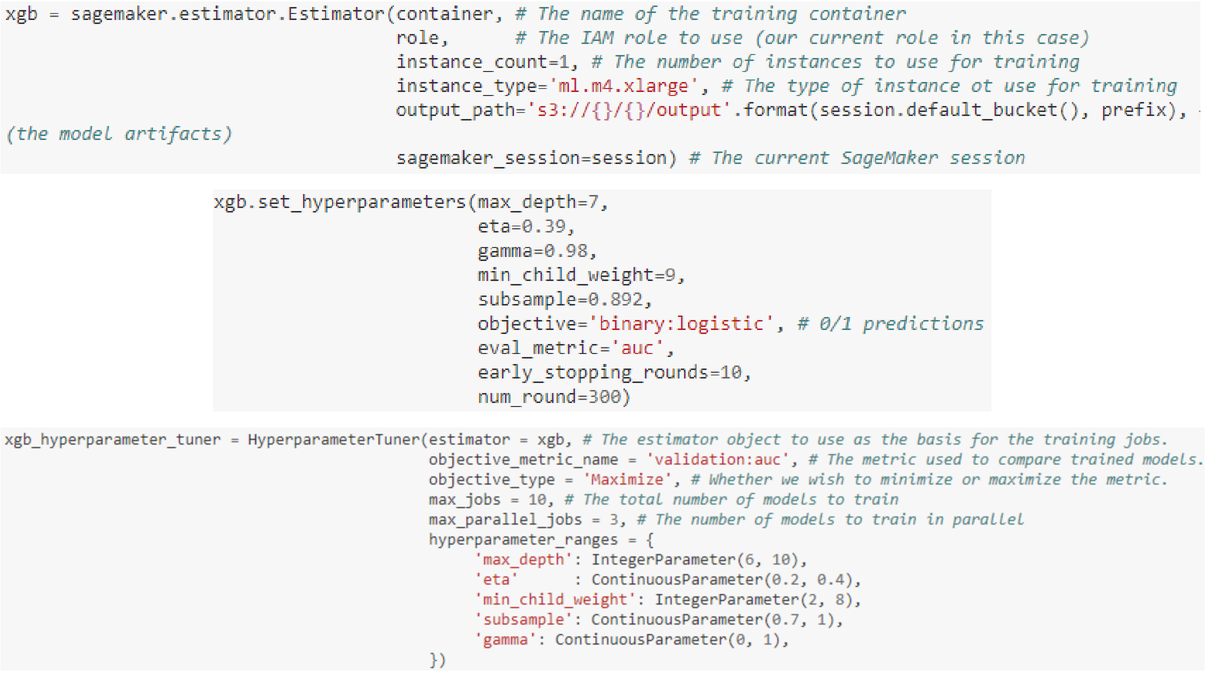
XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. When it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.

The SageMaker hyperparameter tuning function was used to investigate different XGBoost architectures and retain the one that perform the best. More precisely, the function uses a Bayesian search method which treat hyperparameter tuning like a regression problem. First, the function makes guesses about which hyperparameter combinations are likely to get the best results and runs training jobs to test these values. After testing the first set of hyperparameter values, the function uses regression to choose the next set of hyperparameter values to test.

In our case, the following hyperparameters were investigated:

- The "max_depth" of the tree.
- The "eta" (learning rate equivalent).
- The "min_child_weight".
- The "subsample" ratio of training instances (to prevent overfitting).
- "gamma", the minimum loss reduction required to make a further partition on a leaf node of the tree.

Here are the additional parameters used during this task:



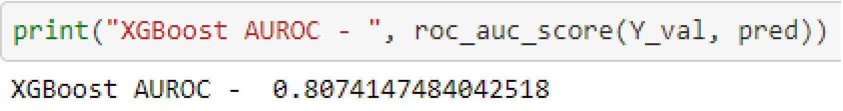
```
xgb = sagemaker.estimator.Estimator(container, # The name of the training container
                                    role,      # The IAM role to use (our current role in this case)
                                    instance_count=1, # The number of instances to use for training
                                    instance_type='ml.m4.xlarge', # The type of instance ot use for training
                                    output_path='s3://{}/{}/output'.format(session.default_bucket(), prefix), 
(the model artifacts)
                                    sagemaker_session=session) # The current SageMaker session
```

```
xgb.set_hyperparameters(max_depth=7,
                        eta=0.39,
                        gamma=0.98,
                        min_child_weight=9,
                        subsample=0.892,
                        objective='binary:logistic', # 0/1 predictions
                        eval_metric='auc',
                        early_stopping_rounds=10,
                        num_round=300)
```

```
xgb_hyperparameter_tuner = HyperparameterTuner(estimator = xgb, # The estimator object to use as the basis for the training jobs.
                                               objective_metric_name = 'validation:auc', # The metric used to compare trained models.
                                               objective_type = 'Maximize', # Whether we wish to minimize or maximize the metric.
                                               max_jobs = 10, # The total number of models to train
                                               max_parallel_jobs = 3, # The number of models to train in parallel
                                               hyperparameter_ranges = {
                                                    'max_depth': IntegerParameter(6, 10),
                                                    'eta'      : ContinuousParameter(0.2, 0.4),
                                                    'min_child_weight': IntegerParameter(2, 8),
                                                    'subsample': ContinuousParameter(0.7, 1),
                                                    'gamma': ContinuousParameter(0, 1),
                                               })
```

Figure 7: From top to bottom: the estimator object (SageMaker instance with a built-in XGBoost framework), the initial hyperparameters (including our chosen metric(s)) and the tuner parameters (mainly the ranges of values within which the hyperparameters will be allowed to variate).

Via a batch transform job, the best performing model made predictions from the validation set. Here is the result:



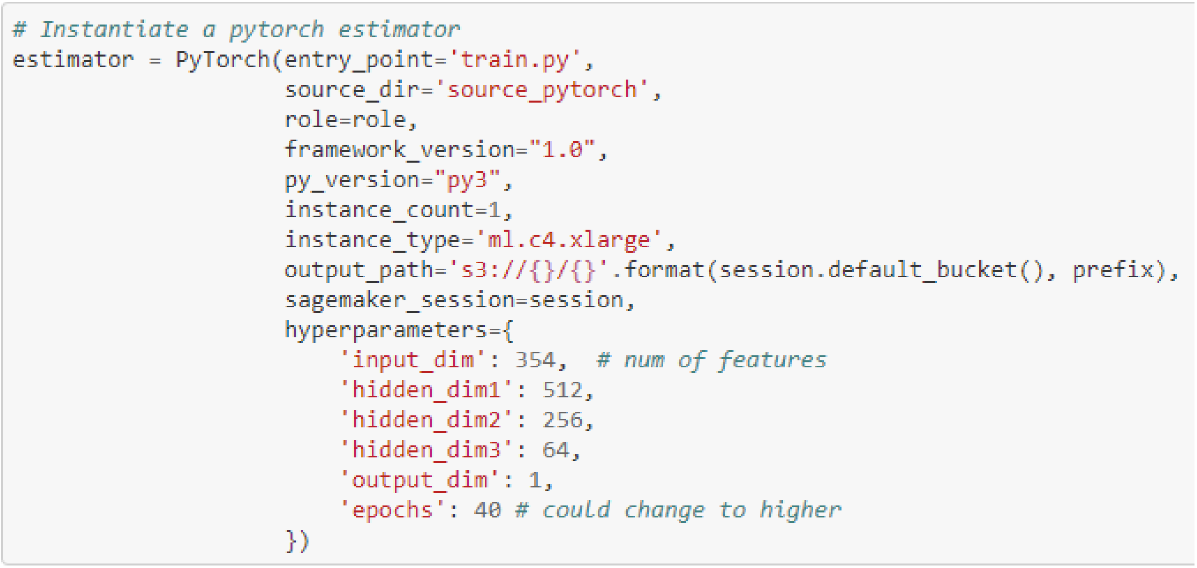
```
print("XGBoost AUROC - ", roc_auc_score(Y_val, pred))

XGBoost AUROC -  0.8074147484042518
```

Without much surprise, our XGBoost model performed better than the benchmark model: 68% < 80%.

Custom Pytorch ANN

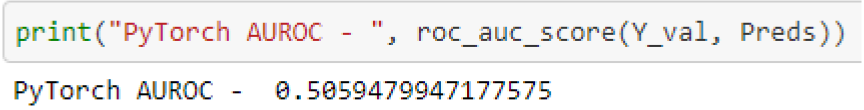
Finally, more out of curiosity than everything else, a classic neural net composed of 4 fully connected layers (and two 30% dropout layers) was implemented. To create and deploy such model, SageMaker requires a model, a train and a predict script to automatically build the model within a chosen instance.



```
# Instantiate a pytorch estimator
estimator = PyTorch(entry_point='train.py',
                    source_dir='source_pytorch',
                    role=role,
                    framework_version="1.0",
                    py_version="py3",
                    instance_count=1,
                    instance_type='ml.c4.xlarge',
                    output_path='s3://{}/{}'.format(session.default_bucket(), prefix),
                    sagemaker_session=session,
                    hyperparameters={
                        'input_dim': 354,  # num of features
                        'hidden_dim1': 512,
                        'hidden_dim2': 256,
                        'hidden_dim3': 64,
                        'output_dim': 1,
                        'epochs': 40 # could change to higher
                    })
```

Figure 7: Pytorch estimator parameters (additional hyperparameters like the batch size and the learning rate are implemented within the “train.py” script.)

Here is the result of the model:



```
print("PyTorch AUROC - ", roc_auc_score(Y_val, Preds))
```

```
PyTorch AUROC -  0.5059479947177575
```

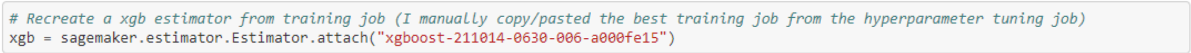
As we can see the result is extremely poor (reminder: the benchmark model scored 0.68). It is a very good example of the limitation of an algorithm which doesn’t account for class imbalance nor the metric of interest.

Feel free to explore the notebook “3) Model creation and training” to see how each model has been implemented.

RESULTS

The final predictions were made on the test data which was provided in the file "Udacity_MAILOUT_052018_TEST.csv" using our trained XGBoost model.

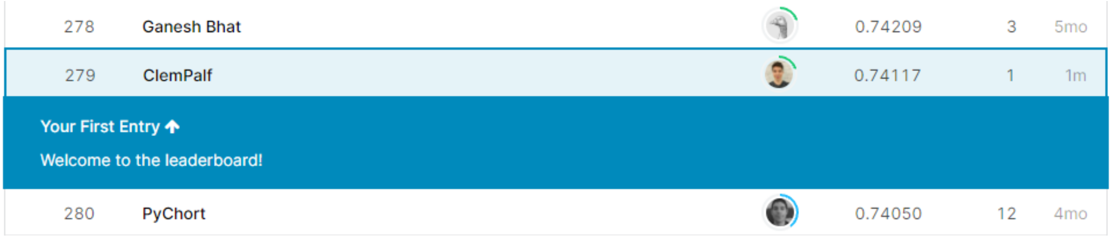
Quick tip: within SageMaker, to create an estimator from a previous training job, you can use the estimator functionality "attach":



```
# Recreate a xgb estimator from training job (I manually copy/pasted the best training job from the hyperparameter tuning job)
xgb = sagemaker.estimator.Estimator.attach("xgboost-211014-0630-006-a000fe15")
```

It can be very useful if you trained your model days ago and don't want to go through the process again. All you need is the name of the training job, which will point to the S3 address where the model artifacts and scripts are stored.

Finally, the model predictions were submitted to the Kaggle website. Here is the result:



278	Ganesh Bhat	0.74209	3	5mo
279	ClemPalf	0.74117	1	1m
Your First Entry ↑ Welcome to the leaderboard!				
280	PyChort	0.74050	12	4mo

Surprisingly, the model performed way poorer than on the validation set (80% against 74%). The difference could be explained that in the competition, we weren't allowed to get rid of rows with too much missing values. I do think it highlights the limitation of simple imputer hypothesis like the one we took (fill NaN value with the most common one in the column).

Many Kaggle competitions were won with model similar to the XGBoost framework, and it only take a few line of codes for anyone to create and tune such model. While everyone can get their hands on powerful frameworks and licensed APIs, data cleaning and optimisation is another story. A data-centric approach generally requires a thorough analyses of the different features and even sometimes, specific business knowledge. In our case it could have mean:

- Exploring and understanding each feature individually. For example, I suspect that some columns had numerical categorical data (like 1 = mean something, 2 = mean something else) which should have been one-encoded.
- There was a lot of missing values within rows. More advanced imputation techniques (again, specific to each feature) could have been investigated.

- We could have added (crafted) other features to the dataset. For example, for each individual, its **cluster number** (from our PCA/K-mean processing). Ofc this feature should also be one-encoded.

In order to finish this project in a timely manner I did not perform such advanced analysis. Nonetheless, this project represented a great challenge and an opportunity to become more efficient with the powerful tool that is SageMaker.

Badea, L. M. (2014). Predicting Consumer Behavior with Artificial Neural Networks. *Procedia Economics and Finance, 15*, 238-246. https://doi.org/https://doi.org/10.1016/S2212-5671(14)00492-4

Kim, J., & Ahn, H. (2009). A New Perspective for Neural Networks: Application to a Marketing Management Problem. *J. Inf. Sci. Eng., 25*, 1605-1616.

-