

Machine Learning Engineer Nanodegree

Capstone Project

Rick Wuebker

October 15th, 2020

I. Definition

Project Overview

Arvato is an internationally active services company that develops and implements innovative solutions for business customers from around the world [1]. They have teamed up Udacity to offer a capstone machine learning project. Their client, a mail-order organic products company, would like to use machine learning to determine which members of the Germany population would be good candidates for their advertising campaign.

Datasets and Inputs

There are four datasets provided:

1. Udacity_AZDIAS_052018.csv; Rows: 891,221; Columns: 366. This dataset gives information on the German population.
2. Udacity_CUSTOMERS_052018.csv; Rows: 191,652; Columns: 369 This datasets gives information on the existing customers of the client.
3. Udacity_MAILOUT_052018_TRAIN.csv; Rows: 42,962; Columns: 367; A training dataset for the supervised prediction model which gives features of past marketing campaign attempts to acquire customers as well as the result.
4. Udacity_MAILOUT_052018_TEST.csv; Rows: 42,833; Columns: 366; A test dataset to accompany (3)

To go along with these datasets are two more files that discuss the features in the datasets, along with their information levels and value ranges:

1. DIAS Attributes - Values 2017.xlsx
2. DIAS Information Levels - Attributes 2017.xlsx

Problem Statement

Using data provided by Arvato that includes characteristics of current customers and for characteristics of the German population, create an unsupervised learning process to get more familiar with the German population and Arvato's current customers. And secondly, create a machine learning model to predict which people in Germany would be good candidates to target for their advertising campaign.

The strategy is to use unsupervised learning to determine features relevant from the current customers and population and also to get more familiar with the types of people that Arvato's products appeal to.

Secondly we will build a supervised learning model that predicts which members of the German population are good targets for a marketing campaign.

The tasks to be completed are as follows:

1. Data Processing: Explore the data and the attributes file to find any potential problems and to prepare the data for machine learning.
2. Customer Segmentation: Apply unsupervised learning algorithms to determine which of Germany's population aligns best with Arvato customers.
3. Customer Prediction: Develop a model that will predict whether a particular location is likely to become an Arvato customer.

In order to complete the tasks above we will apply the following specific techniques:

1. Go through the data and take notes of each feature and any transformations that might need to be made and place these items in a table called metadata for reference in cleaning scripts.
2. For Market Segmentation, we will use the following:
 - a. PCA (principal component analysis)
 - b. K-Means Clustering
3. Finally, for prediction we will test out the following classification algorithms:
 - a. Logistic Regression
 - b. XGBoost
 - c. BalancedRandomForestClassifier
 - d. BalancedBaggingClassifier
 - e. HistGradientBoostingClassifier

Metrics

Customer Segmentation: In this portion of the project we will make sure that our model to segment customers will capture at least 95% of the variance in the Germany data. This is capturing quite a bit of the variance in the dataset because we might want to use the cluster predictions as a feature in prediction.

Customer Prediction: Due to the highly imbalance nature of the data the main metric we will use to evaluate our model will be AUC-ROC [4]. But since we are also interested in obtaining as many customers from the German population as we can, we will also consider recall [5], which is to minimize false negatives.

II. Analysis

Data Exploration

Ultimately this data provided describes structures in Germany and the people that live in those structures and their spending habits and personality traits.

Data Prefixes

While getting to know the data, there are some large groups of similar features with various prefixes to differentiate them. Here are some of the largest groups:

- D19_ prefix. This is bank transaction data.
- FINANZ_ prefix: Data about the type of investor/saver
- KBA05_ prefix: Data about the car owners in the household and structure of building
- KBA13_ prefix: More data about structures in household and automobile info
- LP_ prefix: Data about where the people are in life and earnings
- PLZ8_ prefix: Data about the types of homes in the area
- SEMIO_ prefix: Data about personality

Unknown Features

There were 94 attributes in the data that weren't in the description files. These will be dropped as we don't know exactly what these were or how to recreate.

Ordinal Vs. Categorical

All of the data has been given as ordinal data, even ages are binned. Here is a snapshot of the data feature ALTER_HH which is described as "main age within the household":

0	unknown / no main age detectable
1	01.01.1895 bis 31.12.1899
2	01.01.1900 bis 31.12.1904
3	01.01.1905 bis 31.12.1909
4	01.01.1910 bis 31.12.1914
5	01.01.1915 bis 31.12.1919
6	01.01.1920 bis 31.12.1924
7	01.01.1925 bis 31.12.1929
8	01.01.1930 bis 31.12.1934
9	01.01.1935 bis 31.12.1939
10	01.01.1940 bis 31.12.1944
11	01.01.1945 bis 31.12.1949
12	01.01.1950 bis 31.12.1954
13	01.01.1955 bis 31.12.1959
14	01.01.1960 bis 31.12.1964
15	01.01.1965 bis 31.12.1969
16	01.01.1970 bis 31.12.1974
17	01.01.1975 bis 31.12.1979
18	01.01.1980 bis 31.12.1984
19	01.01.1985 bis 31.12.1989
20	01.01.1990 bis 31.12.1994
21	01.01.1995 bis 31.12.1999

The values are the integers between 0 and 21, which appear to go with the birth year in the column to the right. This implies that the lower the number, the older the main age is. This makes sense to be given as integers in this format. However we need to note the zero value means unknown.

Ordinal data works only if there is a logical order to the values, which is the case for ALTER_HH above. However, there are some features where the values do not seem to take on an order. For example, here is the feature that describes vacation habits (GFK_URLAUBERTYP):

1	Event travelers
2	Family-oriented vacationists
3	Winter sportspeople
4	Culture lovers
5	Nature fans
6	Hiker
7	Golden ager
8	Homeland-connected vacationists
9	Package tour travelers
10	Connoisseurs
11	Active families
12	without vacation

The values are the integers between 1 and 12. We can see that these don't necessarily take on any particular order. So an ordinal approach would not make much sense here. Instead, we should do some encoding to split these values up into each their own feature column of true/false represented as 1/0.

Unknown Values

Within the dataset, as we see with ALTER_HH above they have usually given the value 0 as unknown. However some features also use -1 or 9 as the value for unknown as in the feature for "share of Volkswagen cars in the area (KBA05_HERST2):

-1, 9	unknown
0	none
1	very low
2	low
3	average
4	high
5	very high

Also, in the case above (KBA05_HERST2) the "none" value fits as it comes after "very low", however sometimes the integer 1 gets assigned a "high" value as in

Some variables actually use zero as a valid value, as in the feature above (KBA05_HERST2) and for the amount of online orders for mail order items (D19_VERSAND_ONLINE_QUOTE_12):

0	no Online-transactions within the last 12 months
1	10% Online-transactions within the last 12 months
2	20% Online-transactions within the last 12 months
3	30% Online-transactions within the last 12 months
4	40% Online-transactions within the last 12 months
5	50% Online-transactions within the last 12 months
6	60% Online-transactions within the last 12 months
7	70% Online-transactions within the last 12 months
8	80% Online-transactions within the last 12 months
9	90% Online-transactions within the last 12 months
10	100% Online-transactions within the last 12 months

We see here that zero represents none, instead of unknown.

Each of these cases needs to be addressed in preprocessing the data.

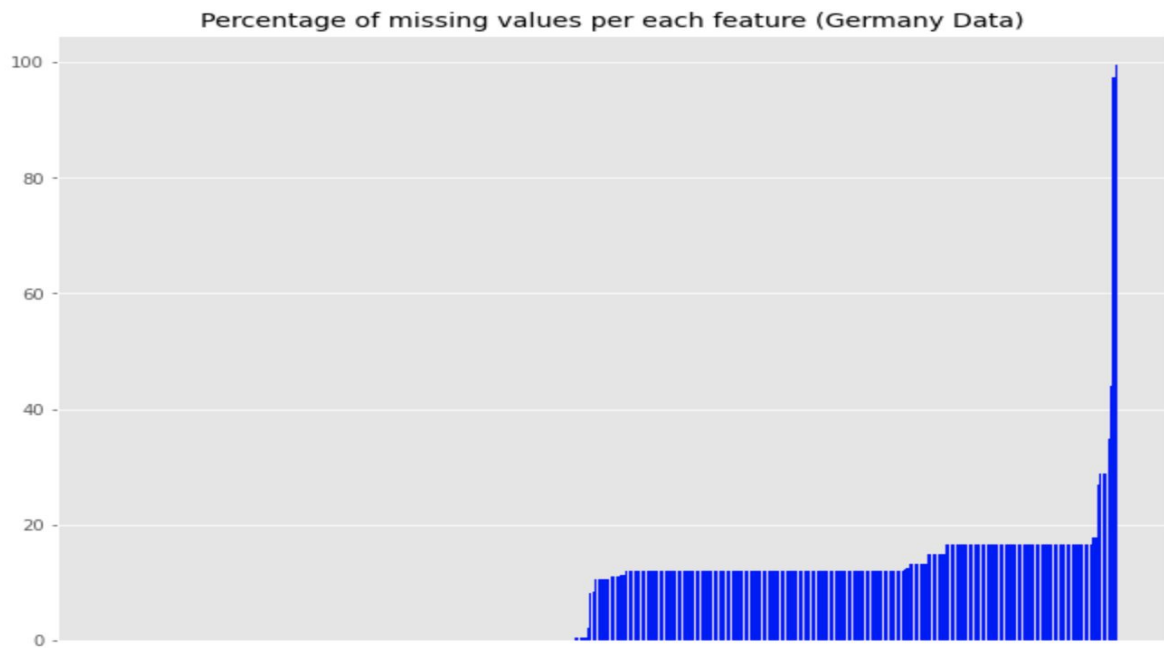
Exploratory Visualization

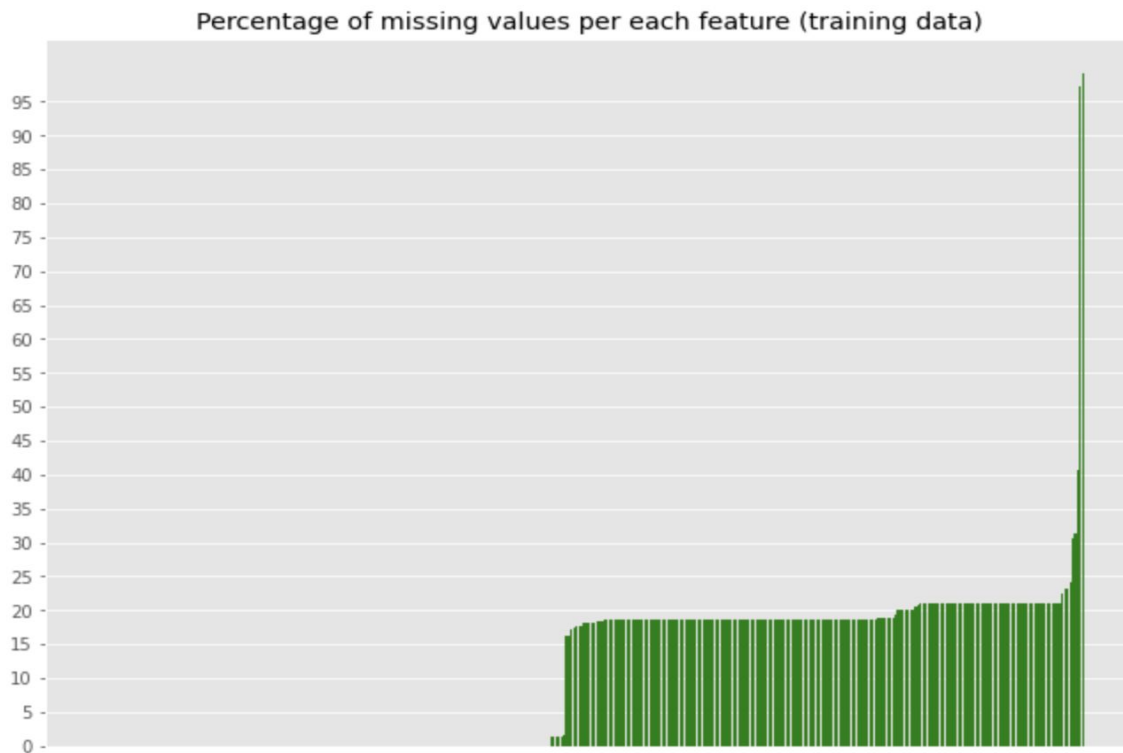
Here is a small snapshot of the data:

	ALTER_HH	ANZ_HAUSHALTE_AKTIV	ANZ_HH_TITEL	ANZ_PERSONEN	ANZ_TITEL	BALLRAUM
0	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	11.0	NaN	2.0	NaN	6.0
2	17.0	10.0	NaN	1.0	NaN	2.0
3	13.0	1.0	NaN	NaN	NaN	4.0
4	20.0	3.0	NaN	4.0	NaN	2.0

Missing Values

One of the first issues we need to address is the NaN values. It would be useful to see if there are any features that have an extraordinary amount of NaN values. Here are the percentage of Nan values for the Germany data for our customer segmentation:





For our training data for our prediction model it might be best to drop columns with greater than 30% missing data.

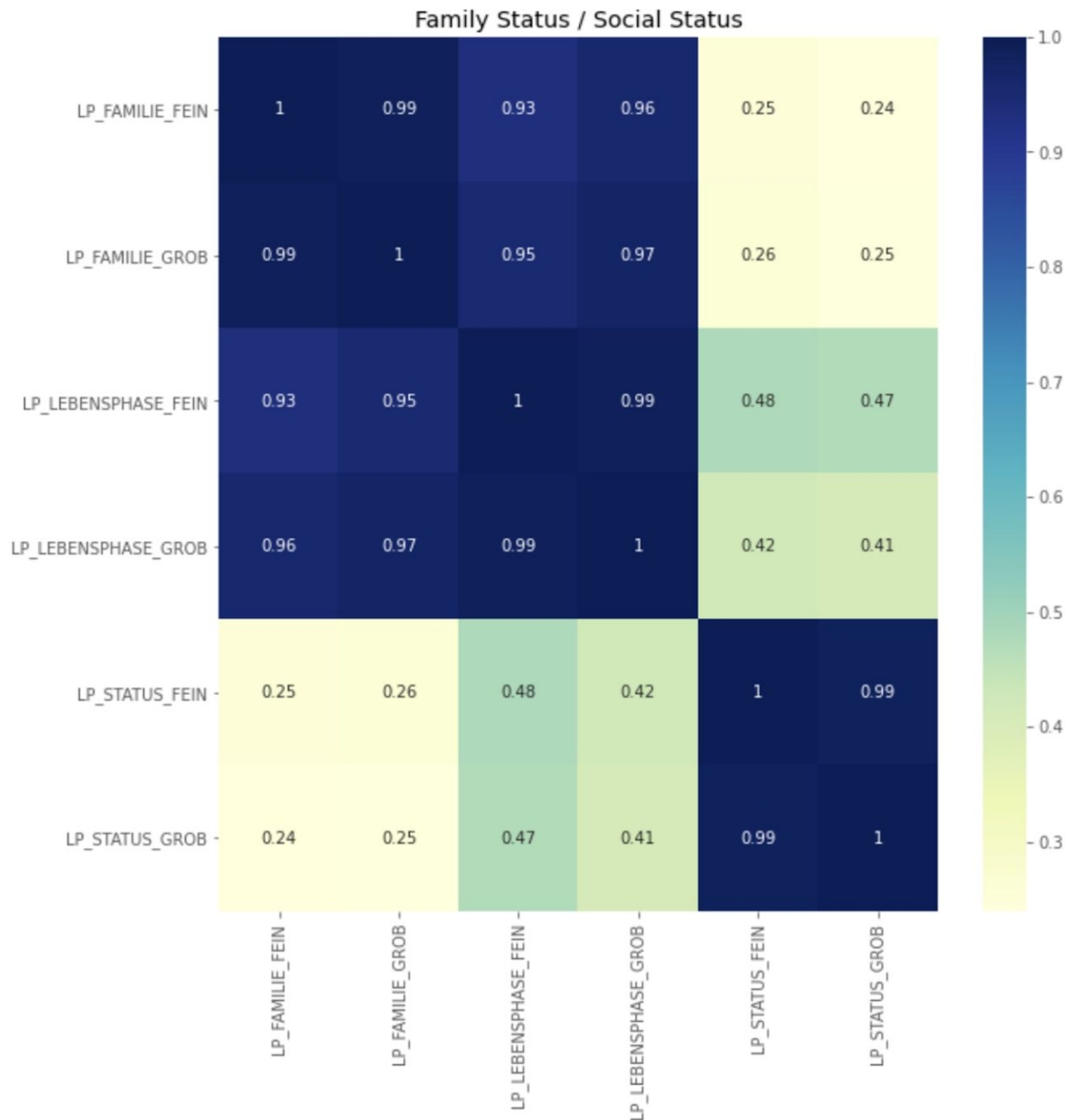
Correlations

Next we will take a look at some heat maps to see how much correlation exists between our features. The overall feature set is too large for one map, so I will mention some notable findings..

Family type / Social Status

```
lp_corr = cd_germany.data.filter(regex='^LP_', axis=1).corr()
```

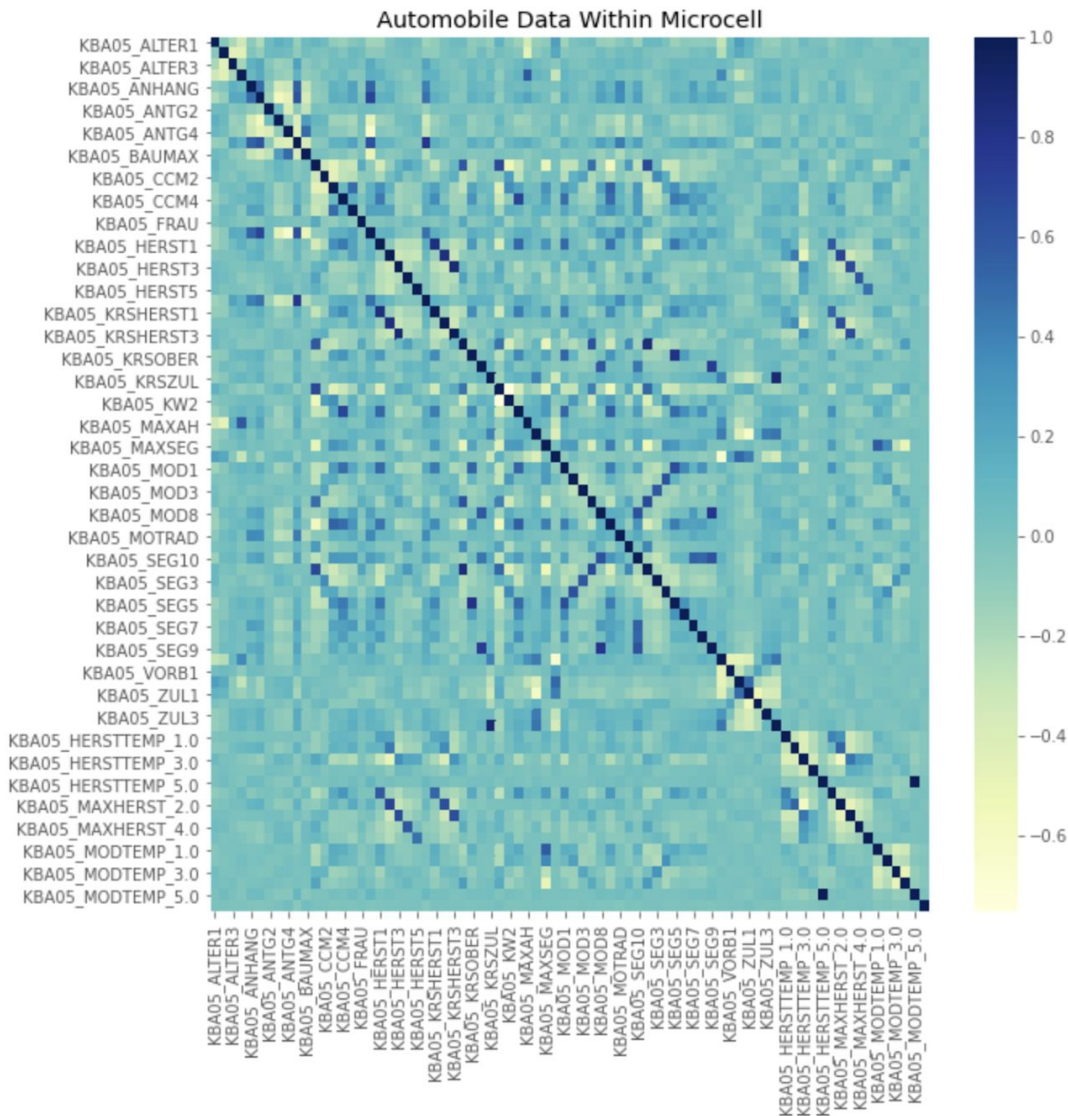
```
f, ax = plt.subplots(figsize=(10,10))
sns.heatmap(lp_corr, annot=True, cmap="YlGnBu")
plt.title('Family Status / Social Status')
plt.show()
```



We see from the above that LP_FAMILIE_FEIN (fine grained family group, e.g. “family with teenager”) and LP_FAMILIE_GROB (more general family grouping, e.g. “single”, “coupe”, “family”) are almost perfectly correlated, which makes sense as they are measuring effectively the same thing. Perhaps the more fine grained should be one-hot-encoded to see if one of those features has predictive power for Arvato customers.

```
kb5_corr = cd_germany.data.filter(regex='^KBA05_', axis=1).corr()
```

```
f, ax = plt.subplots(figsize=(10,10))
sns.heatmap(kb5_corr, annot=False, cmap="YlGnBu")
plt.title('Automobile Data Within Microcell')
plt.show()
```



We see from the above that there are lots of strong correlations scattered among the KBA05 data just showing that we do have some redundancy of information here.

Algorithms and Techniques

There are effectively two objectives that we need to choose algorithms for:

1. Market Segmentation
2. Customer Prediction

Market Segmentation

For determining characteristics of Arvato customers compared to characteristics of the German population, we need to group the population into clusters, and see which clusters are most likely to become Arvato customers. In order to do this, we need to use our data for the German population, and Arvato customers and find out which features are the strongest factors that split up these clusters into the most efficient groupings.

One of the challenges here, as we have seen from the correlations, is that there are redundant features within our datasets. We need to reduce our feature set into the most independent features. One of the best ways to do this is with PCA (principal component analysis) [7]. This technique will be used to make sure we meet our objective of capturing at least 90% of the variance in the dataset and reduce our feature set to linear combinations of the existing features that capture this variance.

After we have our principal components, we will use K-Means clustering [8] to determine how to efficiently group our observations into clusters. K-Means is an efficient clustering algorithm that mathematically determines the squared euclidean distances between each of the observations and finds cluster centers that minimize the difference between each point and the nearest cluster, given an amount of clusters, “k”. We will use the “elbow method” [9] to determine the correct value of “k”.

To recap, for market segmentation we will use the following:

1. PCA to reduce the dimensionality into the most independent linear combinations of features
2. K-Means to cluster the German and Arvato customer data

PCA, Principal Components Analysis

PCA is a technique that is used to reduce the dimensionality of data into linear combinations of the features. This technique distills the feature set into what is called “principal components”. It is a transformation of the data into uncorrelated components. There is a bit of information loss, but PCA tries to keep it at a minimum. The result is a smaller set of features that are uncorrelated which means they are suitable for linear algorithms like K-Means or Logistic Regression.

K-Means

K-means is a clustering algorithm. This means that it will put your data into “k” groups. So if you run the algorithm with a k parameter of 5, the algorithm will put your data into 5 groups. The way

it does this is by computing distances between the observations, usually the euclidean distance, and then it groups observations with the smallest distances together, then solves for where the center of each of the “k” clusters should be so that the centers of the clusters are as far apart as the algorithm can make them. This is an excellent algorithm to group quantitative or encoded categorical data, to see which observations cluster together.

Customer Prediction

For prediction of which customers from the training dataset would most likely become new Arvato customers we will use a logistic regression model as well as an assortment of ensemble methods, in particular, gradient boosting algorithms that perform particularly well in kaggle competitions. These algorithms are an extremely popular algorithm for large feature sets as it works well with data that might be correlated. It also works extremely well with data that isn't linearly related to the response variable while not overfitting.

Gradient Boosting algorithms are ensemble techniques that use smaller decision stumps to avoid overfitting, and puts more emphasis on incorrect predictions during training. Some can be extremely fast like XGBoost and HistGradientBoostingClassifier.

Another issue with the prediction training set is the sparse amount of positive cases compared to the negative cases. To handle this, we will try with and without SMOTE (synthetic minority over-sampling technique) [11] in order to even out the dataset. Essentially, SMOTE is a technique that we can apply to the positive response observation to synthetically create new observations with combinations of those observations. This could help with better prediction results since we have so few positive observations.

To recap, for customer prediction we will use four methods and determine which is most accurate:

1. XGBoost
2. HistGradientBoostingClassifier - new sklearn implementation inspired by LightGBM
3. BalancedBaggingClassifier - comes with imblearn package for imbalanced datasets
4. BalancedRandomForestClassifier - comes with imblearn package
5. LogisticRegression

The models above, with the exception of Logistic Regression, are what is known as ensemble methods. This means they combine more simple, “weak-learners”, with different weights to compute their prediction. I chose to work mainly with ensemble methods as they have better performance than many algorithms without overfitting. I will discuss the differences below

XGBoost [12]

XGBoost is a very popular gradient boosting algorithm. “Boosting” implies that the algorithm iterates as it adds new models to its ensemble, by focusing on the errors made by the previous model. XGBoost is an implementation of boosting that is known for its speed and accuracy with complex data. The main reason XGBoost is faster than a traditional Gradient Boosting Machine is its use of histogram based sampling. At each iteration, the boosting algorithm needs to determine the most optimal split of the data to build the next tree. In order to do this, it would have to calculate all the splits possible across features. Instead of doing this, XGBoost and other histogram-based methods, bin the features into groups. This makes them much faster, however there is a chance of losing a bit of accuracy.

HistGradientBoostingClassifier

This is another algorithm that is histogram-based. Again, known for its speed with complex datasets. This is a relatively new implementation within sklearn and I’m curious how it performs compared to XGBoost.

Balanced Bagging Classifier

This classifier is found in the imblearn python package [13]. Bagging classifiers operate differently than boosting classifiers. Bagging algorithms randomly sample the data into “bags” before training for each of the weak learners. This helps the model avoid overfitting to the entire dataset while training. However a problem with traditional bagging classifiers is that when used with a highly imbalanced dataset, the subsampling will favor the majority class. This implementation adds an additional step before taking its subsamples to make sure it undersamples the majority class, which balances out each of the subsets. Since our dataset is highly imbalanced, it will be interesting to see how it performs.

Balanced Random Forest Classifier

This algorithm is also from the imblearn package. Random forest classifiers also use bagging to subsample the data before building each tree. This package adds a step to undersample the majority class while choosing its subsamples. Again, it will be interesting to see if there are any significant improvements against the boosting classifiers.

Logistic Regression

This is the only algorithm of the prediction group that isn’t an ensemble method. However it is a tried and true method. Logistic Regression is a linear model that uses cross entropy loss as its loss function [14]. Our data is very complex but, if we first decompose the dataset into its principal components before sending them into the regressor we will have our best shot at good results. Let’s see how it stacks up.

Benchmark

This is an extremely imbalanced dataset, roughly 80 negative cases per each positive case. This makes accuracy not a very good metric, as one could predict all negative and get a great accuracy score. This is why we will be using AUC under the ROC curve for our metric. **[4]**

We want to find a model that beats pure guessing, which would have an $AUC = 0.50$. So our benchmark model would be to guess all are negative. This would produce an accuracy score of 0.988 and an AUC of 0.50 and recall of 0.0. We want to see if one of our models above can achieve a (hopefully significantly) higher AUC than 0.50.

III. Methodology

Data Preprocessing

Metadata DataFrame

In order to better keep track of the necessary transformations of the data a new dataframe was created and stored in metadata.csv. This contains the necessary transformations for each feature.

Unknown Features

There were 94 attributes in the data that weren't in the description files. These will be dropped as we don't know exactly what these were or how to recreate.

Encodings

These are some of the data that have been given as ordinal but the ordering doesn't seem to make sense so these will be one-hot-encoded [6]. Examples are given for each.

- GFK_URLAUBERTYP: Ex: "nature lover", "winter sports people"
- HEALTH_TYP: Ex: "unknown/not possible", "critical reserved", "sanitary affine"
- FINANZTYP: Ex: "low financial interest", "unremarkable", "saver", "main focus is house"
- NATIONALITAET_KZ: Ex: "Sounding german", "Sounding Foreign", "Assimilated names"
- ZABEOTYP: "green", "smart", "fair_supplied", "seeking orientation"
- ANREDE_KZ: "male", "female"
- CAMEO_DEU_2015: "Work-Life-Balance", "Wealthy Best Ager", "Empty Next"
- D19_KONSUMTYP: "Universal", "Modern", "Versatile"
- D19_KK_KUNDENTYP: "regular customer", "active customer", "new customer"
- GEBAEUDETYP: "residential building", "company building", "mixed building"
- GREEN_AVANTGARDE: "belongs", "does not belong"
- KBA05_HERSTTEMP: "promoted", "stayed upper level", "demoted"
- KBA05_MAXHERST: "Top-German", "VW-Audi", "Ford/Opel"
- KBA05_MODTEMP: "promoted", "new building", "demoted"
- OST_WEST_KZ: "East", "West"
- TITEL_KZ: "Dr.", "Prof."
- PRAEGENDE_JUGENDJAHRE: This will be split based on "avant garde" = 1 and "mainstream" = 0

Miscellaneous Issues

- KBA05_ prefix data is given as ordinal, however the unknown value is either -1 or 9, which would make it either the smallest or largest in the group. These need changed to all -1 or 0 for unknown.
- LNR: This appears to be an index for each observation. This will be dropped.
- RELAT_AB: This needs unknown values changed from [-1,9] to just [-1]
- SEMIO_ prefix data also needs unknowns to only be [-1] instead of [-1,9].
- MIN_GEBAEUDEJAHR: This is the date the building was first recorded in the data. This will be dropped.
- CAMEO_DEUG_2015 has a value of 'XX' which is not explained in the attributes file, CAMEO_DEU_2015 has a value of 'X' which is not explained in the attributes file. These will be dropped.

Unknown Values

For many of the features the “unknown” were labeled as -1 or 0. But sometimes 0 meant none, not unknown. The following features have a 0 that means none:

- AGER_TYP
- BIP_FLAG
- All D19_ prefix data
- All KBA05_ prefix data
- All KBA13_ prefix data
- ONLINE_AFFINITAET
- All PLZ8_ prefix data

Finally, after we have converted all unknown values to -1, we can set them to NaN values and calculate what percentage of each feature is NaN to determine if we need to drop that feature.

Preprocessing Steps

The preprocessing pipeline will work as follows within a script following the notes above:

1. Remove unknown features
2. Set “0” values to “-1” where appropriate
3. Set “9” values to “-1” where appropriate
4. Set all “-1” values to NaN
5. One-Hot-Encode the identified features above
6. Drop features according to the percentage of NaN threshold
7. Drop rows with NaN values
8. For Segmentation we will normalize all values between 0 and 1

Implementation

For implementing the data processing, I have created a metadata.csv file that lists features of each feature, for example, uses “9” for “unknown” so we can switch these to NaN values later. The metadata file is used heavily for data processing for both Market Segmentation and Prediction.

Market Segmentation

For market segmentation, the training will be done on the German Population dataset. The following describes the steps:

1. Remove unknown features of German and Customer datasets
2. Drop special cases from both datasets
3. Set “0” values to “-1” where appropriate on both datasets
4. Set “9” values to “-1” where appropriate on both datasets
5. Set all “-1” values to NaN on both datasets
6. One-Hot-Encode the identified features above on both datasets
7. Drop features according to the percentage of NaN threshold on both datasets
8. Drop rows with NaN values on both datasets
9. Normalize all values between 0 and 1 on both datasets
10. Fit a PCA with n=total features to determine how many will capture 90% of the variance on the german dataset only
11. Fit a PCA with n = the amount of features necessary from (10) on German dataset
12. Look at the top 5 components and see what they are composed of
13. Transform both Germany and Customer data with the fit pca model
14. Using the transformed data from (13) we will run k-means on the German data.
15. We will predict the transformed customer data from (13) and k-means in (14)
16. We will analyze the results of the clusters

Customer Prediction

For customer prediction we will perform the following steps:

1. Remove unknown features of Train and Test datasets
2. Drop special cases from both datasets
3. Set “0” values to “-1” where appropriate on both datasets
4. Set “9” values to “-1” where appropriate on both datasets
5. Set all “-1” values to NaN on both datasets
6. One-Hot-Encode the identified features above on both datasets
7. Gather features according to the percentage of NaN threshold on both datasets
8. Drop the same features, the union of (7) from both datasets

9. Drop any additional features due to one-hot-encoding that might be in one and not the other to keep the feature set of Germany data and Arvato customer data the same
10. Drop rows with NaN values on both datasets
11. Normalize all values between 0 and 1 on both datasets
12. Fit a PCA with the training dataset
13. Fit a logistic regression model on the training dataset with PCA inputs
14. Fit the gradient boosting algorithms with the data before Normalizing and PCA

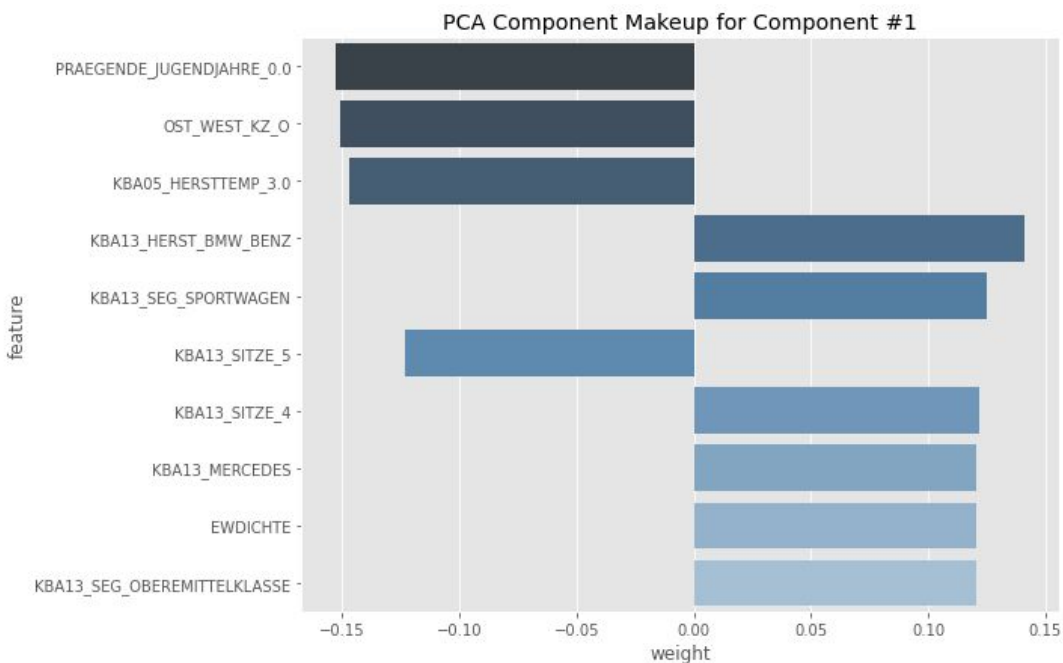
IV. Results

Model Evaluation and Validation

Market Segmentation

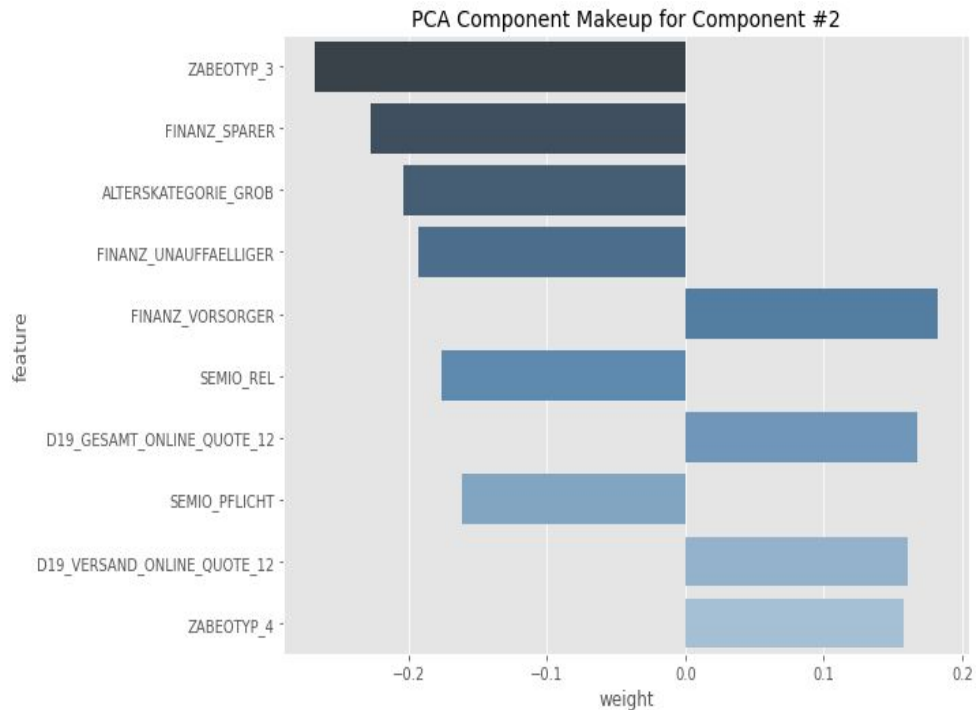
It was found that 200 components would be needed for the PCA model to retain 95% of the variance in the dataset. This reduces the features set from 370 to 200. Examining the first five features shows how it found linearly independent features in the data.

Component 1 main composition area:



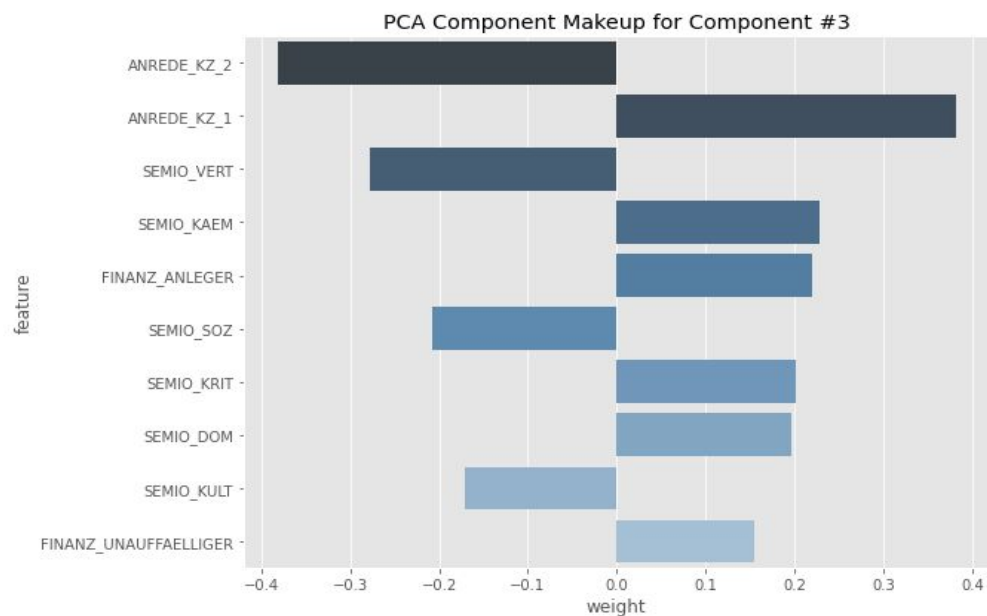
This component has a negative coefficient on PRAEGENDE_JUGENDJAHRE_0.0, which is the group that grew up with a more mainstream culture versus avant garde. So the fact that it is negative implies that it is more avant garde types. It also has a negative weight on OST_WEST_KZ_O which is the East flag. This implies that it is more avant garde types in West Germany.

Component 2 main composition area: Financial Affinities



This component seems to be collecting those that are not savers as it is putting a negative coefficient on FINANZ_SPARER which is the affinity to save. ZABEOTYP_3 is the “fair supplied” energy users. There is also a negative weight on ALTERSKATEGORIE_GROB, which is an age signal. The negative coefficient implies it is collecting the younger profiles.

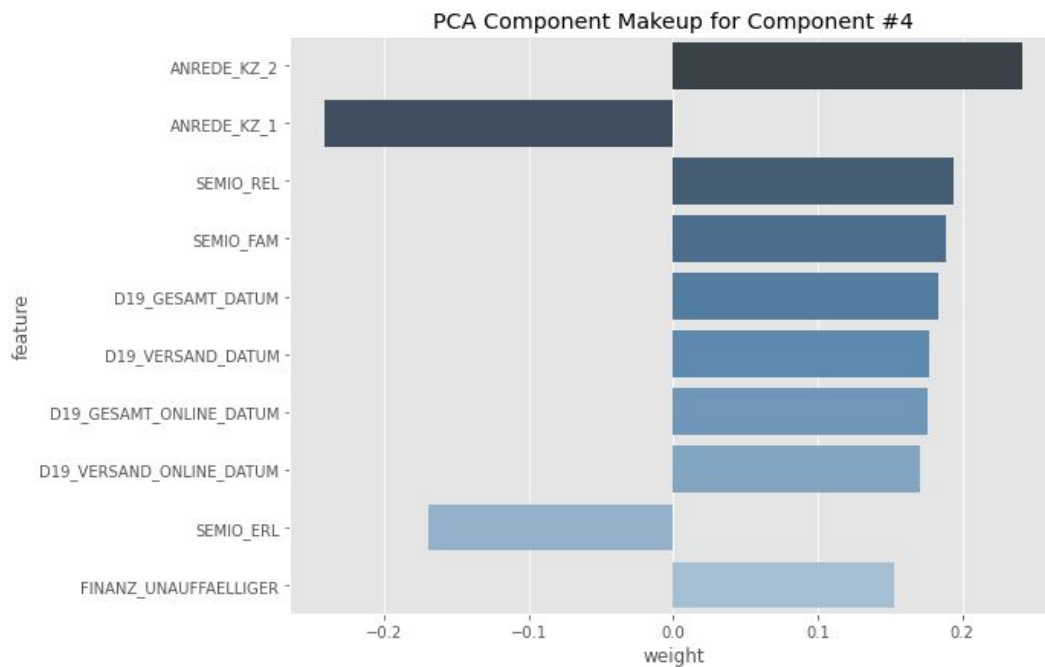
Component 3 main composition area: Gender



This component is putting a negative coefficient on women (ANREDE_KZ_2) and high coefficient on men (ANREDE_KZ_1). It also includes some personality traits like SEMIO_VERT

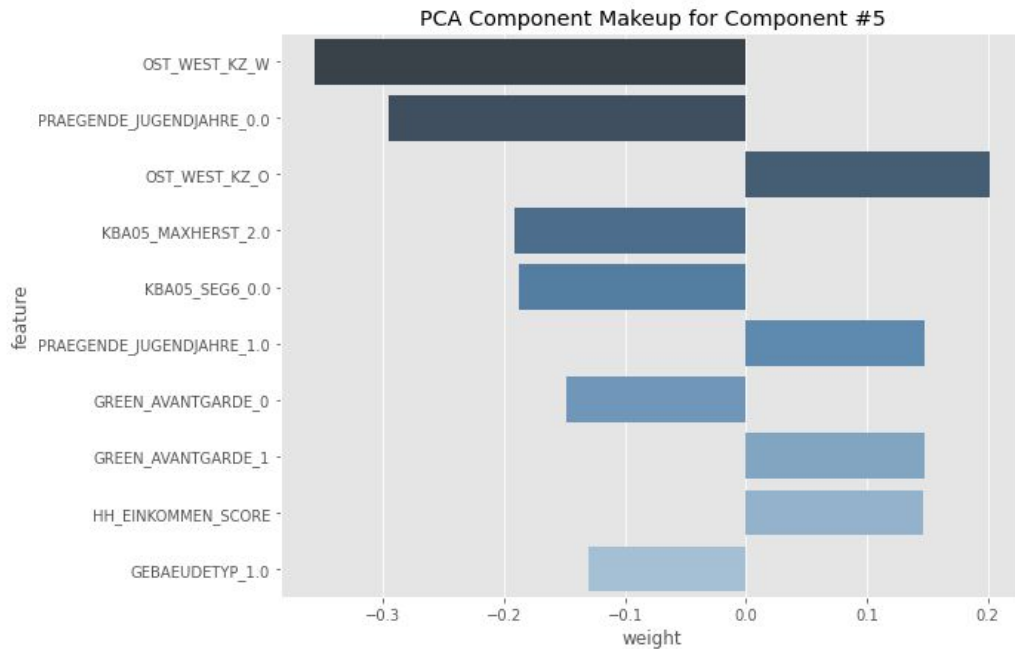
which is the propensity for the person to be “dreamily”. Since the coefficient is negative these could be more practical types. The positive coefficient on SEMIO_KAEM implies these people have a more “fightfull” attitude.

Component 4: Gender, in particular women:



This component puts a positive weight on women and a negative weight on men and also includes more personality types. There are positive weights on SEMIO_REL (affinity towards religion) and SEMIO_FAM (affinity towards family).

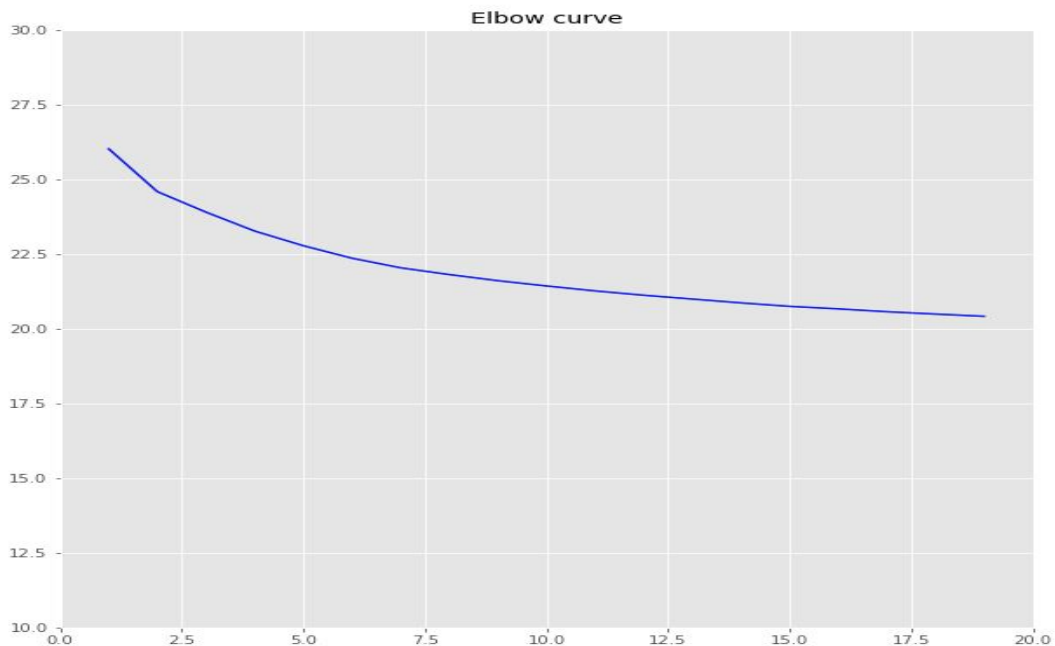
And the last one we will look at in detail is component 5:



This component puts a negative weight on Western Germany indicating it is individuals from the East. Also there is a negative weight on PRAEGENDE_JUGENDJAHRE which implies these individuals were raised in a more avant garde culture.

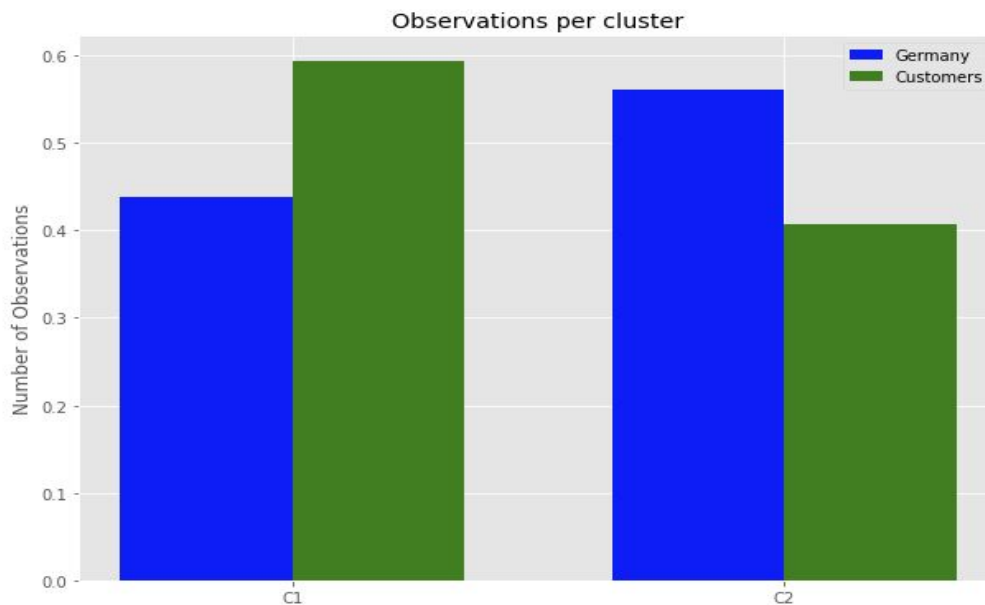
ELBOW CURVE

In order to determine the amount of clusters we will take a look at an elbow chart:

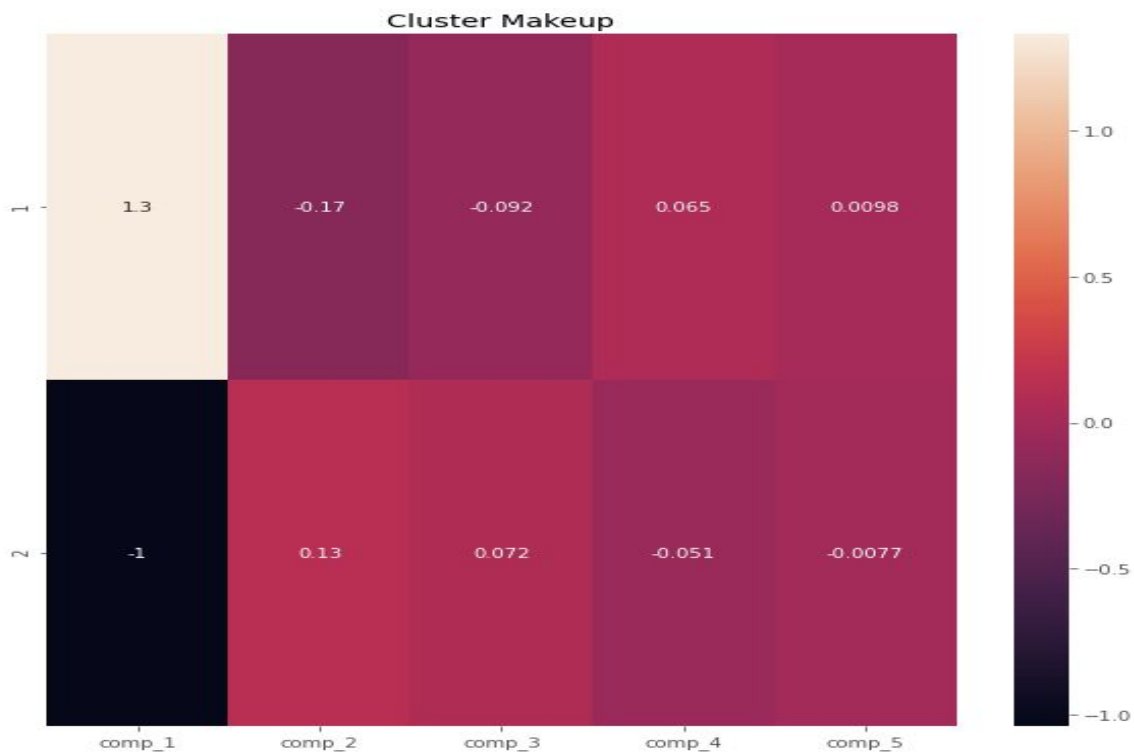


From this curve, it appears that 7 clusters is the sweet spot where returns start to diminish afterwards slowly. Also, there is a huge gain from 1 to 2 clusters. That might be interesting to take a look at to see what those clusters are composed of.

First looking at $K = 2$, here are the distributions of percentages per cluster:

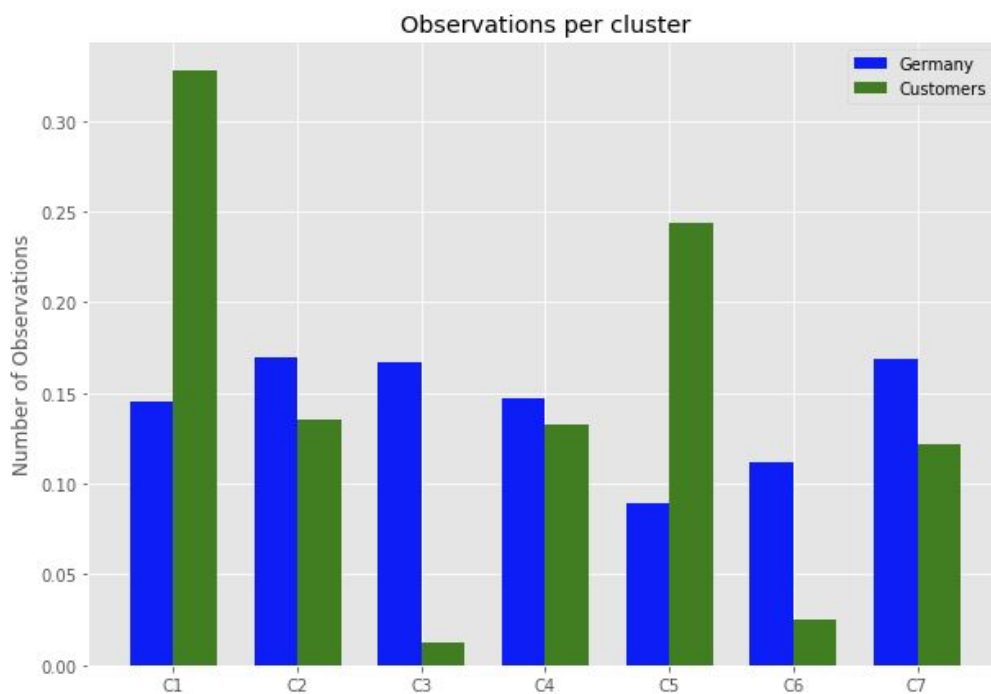


It appears that almost 60% of Arvato's customer's are in cluster one, and a little over 40% are in cluster two. This isn't a terribly large separation. Let's take a look at the composition of the clusters:

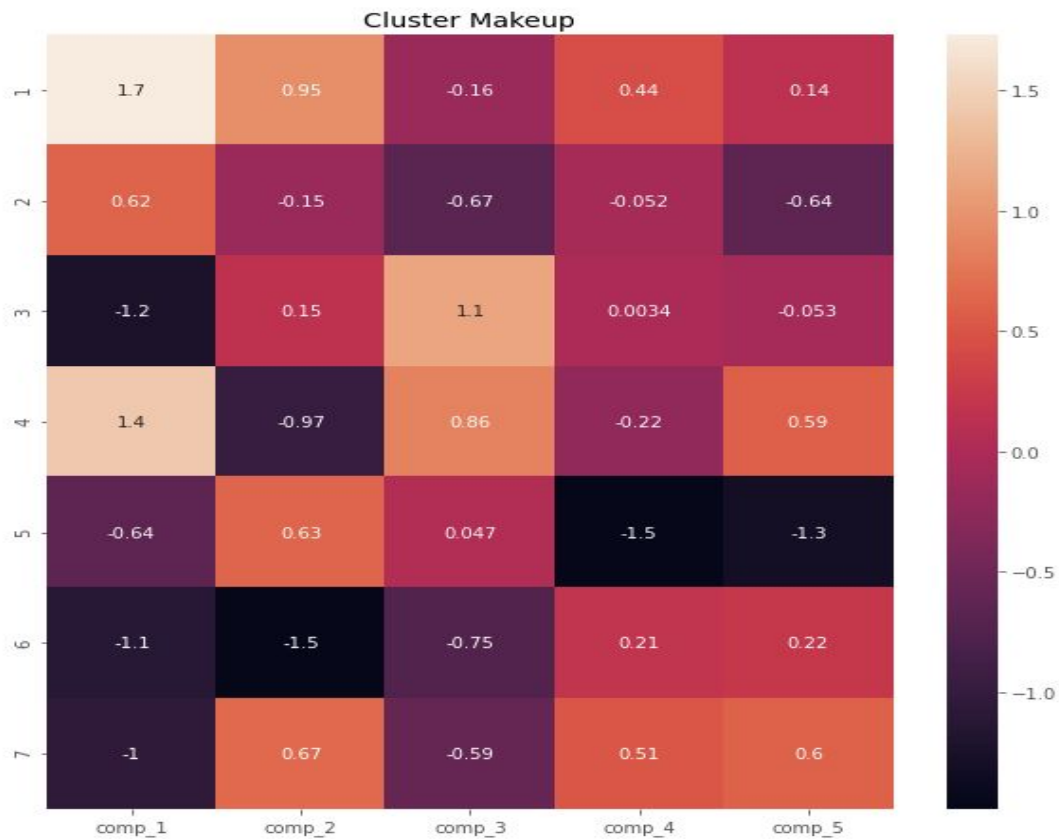


There is a heavy positive coefficient for cluster one, on component one, which was Western Germans that grew up with a more avant garde culture.

Moving on to take a look at a K-Means with 7 clusters:



Here we see that clusters one and five make up the largest percentages of Arvato's customers. Let's take a look at the make up of clusters one and five.



Looking at cluster one, it has large coefficients on components one and two. These are West Germans who grew up with avant garde culture and component two is sorting out younger individuals who are non-savers.

Looking at cluster five, the largest of the top five components is component four, which splits on Gender, mainly positive coefficient for women, but since this is a negative coefficient it implies men. And component five which again splits on East/West Germany and cultural upbringing. Which makes this cluster very similar to cluster one.

Based on these results, I'm going to transform the train and test prediction data to create a binary feature that would be if they are part of cluster one or cluster five.

Prediction

Each of the following sections has their own jupyter notebook of similar name on github.

Initial Training

To begin, I used the same pipeline for market segmentation. I added scaling and pca to the pipeline for Logistic Regression as that model assumes the inputs are linearly independent.

For the random forest classifier and boosting algorithms I found scaling had no effect so I removed the scaling and PCA from those pipelines.

The only hyperparameter I used was one that comes with XGBoost for unbalanced data called "scale_pos_weight" which puts more weight on the positive targets in unbalanced datasets. I found a value of 180 seemed to work the best. The other algorithms didn't provide this hyperparameter and nothing similar.

Here are the initial results:

Algorithm	AUC w/ NA	AUC w/o NA
HistGradientBoostingClassifier	0.565	0.69
BalancedRandomForestClassifier	0.592	0.689
Logisticregression	0.624	0.686
BalancedBaggingClassifier	0.5816	0.667
XGBoost	0.558	0.662

I report the AUC with two different training sets. The first one, is keeping all rows, and imputing the values of all those NaN values. This performs worse on cross validation.

The second training set is with dropping rows that don't have at least 85% of non-nan values. This performs better on cross-validation.

The problem with the second group, is that in order to submit on Kaggle, we need to submit a prediction probability for all rows, meaning, we can't drop any. So although the training on a dataset where we drop rows with large amounts of NA values will not generalize.

I tested this by submitting the algorithms trained with dropping NA rows for Logistic Regression and HistGradientBoostingClassifier to kaggle. The results were not good. The Logistic Regression AUC on Kaggle dropped to 0.538 and for HistGradientBoostingClassifier dropped to 0.5808. So the LightGBM inspired algorithm generalized better than the linear model.

predictions.csv a minute ago by rwueb add submission details	0.58083
predictions.csv 6 minutes ago by rwueb add submission details	0.53803

So moving forward, I will use a training set that does not drop rows, and instead imputes on median numerical values and most frequent categorical values.

SMOTE

A technique used to help train imbalance datasets is SMOTE **[10]**, Synthetic Minority Oversampling Technique. This can be implemented in python in the imblearn package. I'm already using two classifiers from this package (BalancedRandomForestClassifier and BalancedBaggingClassifier). Essentially SMOTE takes your positive observations and synthetically creates new observations with them.

In order to test with SMOTE I wanted to first split my data, to have an "out of sample" set to test the results on. Basically, this is a set of data that retains the natural imbalance.

Then with SMOTE I will create more observations and do a cross validation to see what the results are on a more "balanced" data set. After I have that score, I will test my model on my out of sample scores to see how well my SMOTE model works on out of sample data.

Here are the results:

Algorithm - SMOTE 1:1 Balance	In Sample AUC	Out of Sample AUC
BalancedRandomForestClassifier	0.993	0.516
HistGradientBoostingClassifier	0.991	0.518
BalancedBaggingClassifier	0.991	0.512
XGBoost	0.991	0.517
Logisticregression	0.794	0.563

From the table above, we can see that the classifiers did outstanding on the smote balanced datasets, however when I tested a SMOTE trained model against an imbalance dataset the results were not much better than guessing.

Instead of using SMOTE to make 100% balanced datasets, I will try with a 1:3 ratio of positive to negative samples. We can do this with the SMOTE class from imblearn. Here are the results:

Algorithm - SMOTE 1:3 Balance	In Sample AUC	Out of Sample AUC
BalancedRandomForestClassifier	0.99	0.522
HistGradientBoostingClassifier	0.986	0.5
XGBoost	0.986	0.527
BalancedBaggingClassifier	0.985	0.521
Logisticregression	0.791	0.527

Here the results are not much better.

Feature Selection

Next we will try to pick some features intuitively based on what we have learned in the Market Segmentation section and see if we can do better.

Here are some of the fields chosen based on them showing up in Market Segmentation:

WOHNLAG	GEBURTSJAHR	LP_STATUS_GROB
HH_EINKOMMEN_SCORE	CAMEO_DEUG_2015	D19_KONSUMTYP
FINANZTYP	VERS_TYP	ALTER_HH
ZABEOTYP	SEMIO_SOZ	GREEN_AVANTGARDE

We will also add in the feature we discussed in Market Segmentation which is a binary feature that is 1 if that observation was predicted to be in cluster 1 or cluster 5 and 0 otherwise.

Here are the results:

Algorithm - Feature Selection	In Sample AUC
BalancedRandomForestClassifier	0.57
HistGradientBoostingClassifier	0.545
XGBoost	0.524
BalancedBaggingClassifier	0.539
Logisticregression	0.566

These results are similar to our initial predictions with way less features.

Feature Engineering

In this final section I will try to quantitatively figure out which features and first level interactions are the most important. To do this I will split the training dataset into the observations where the RESPONSE = 0 and the other where the RESPONSE = 1. Then I will calculate the mean of each feature and compare the differences of means using the standard errors of the two mean calculations [11]. I also computed the differences in the distributions between first order interactions. These were calculated in the notebook "Create_Interactions.ipynb".

I sorted them by how many standard deviations are between the means of the two distributions. The idea is that the larger the difference between the two distributions (response=0 and response=1) could imply better predictive power.

I selected the top 500 features and here are the results:

Algorithm - Feature Engineering / Interactions	AUC
Logisticregression	0.732
XGBoost	0.681
BalancedRandomForestClassifier	0.646
HistGradientBoostingClassifier	0.622
BalancedBaggingClassifier	0.582

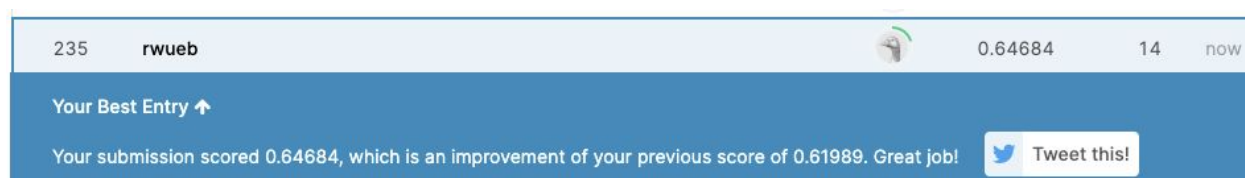
These are the best results yet. I don't trust the Logistic Regression model because it failed to generalize when I submitted the first results to Kaggle. So after that, it looks like XGBoost is the winner.

I used AWS Sagemaker hyperparameter tuning to get the best params for an XGBoost model with these features and these are the outputs:

```
{'_tuning_objective_metric': 'validation:auc',  
 'alpha': '439.5109181794774',  
 'colsample_bylevel': '0.35621991418149157',  
 'colsample_bynode': '0.12930564658433663',  
 'colsample_bytree': '0.5604870818559543',  
 'early_stopping_rounds': '10',  
 'eta': '0.3237766035412486',  
 'eval_metric': 'auc',  
 'gamma': '6.835606394556246',
```

```
'lambda': '313.30483435447155',  
'max_delta_step': '7',  
'max_depth': '15',  
'min_child_weight': '3',  
'num_round': '108',  
'objective': 'binary:logistic',  
'scale_pos_weight': '180',  
'subsample': '0.8654307820386923'}
```

After applying these to my XGBoost model I retrained again locally and these are the results: 0.646



A screenshot of a Kaggle competition leaderboard. The top bar shows a user with ID 235 and username 'rwueb' has a score of 0.64684, which is an improvement from their previous score of 0.61989. The bar also shows 14 other submissions and the time 'now'. Below the bar, a blue banner reads 'Your Best Entry ↑' and 'Your submission scored 0.64684, which is an improvement of your previous score of 0.61989. Great job!'. A 'Tweet this!' button is visible on the right.

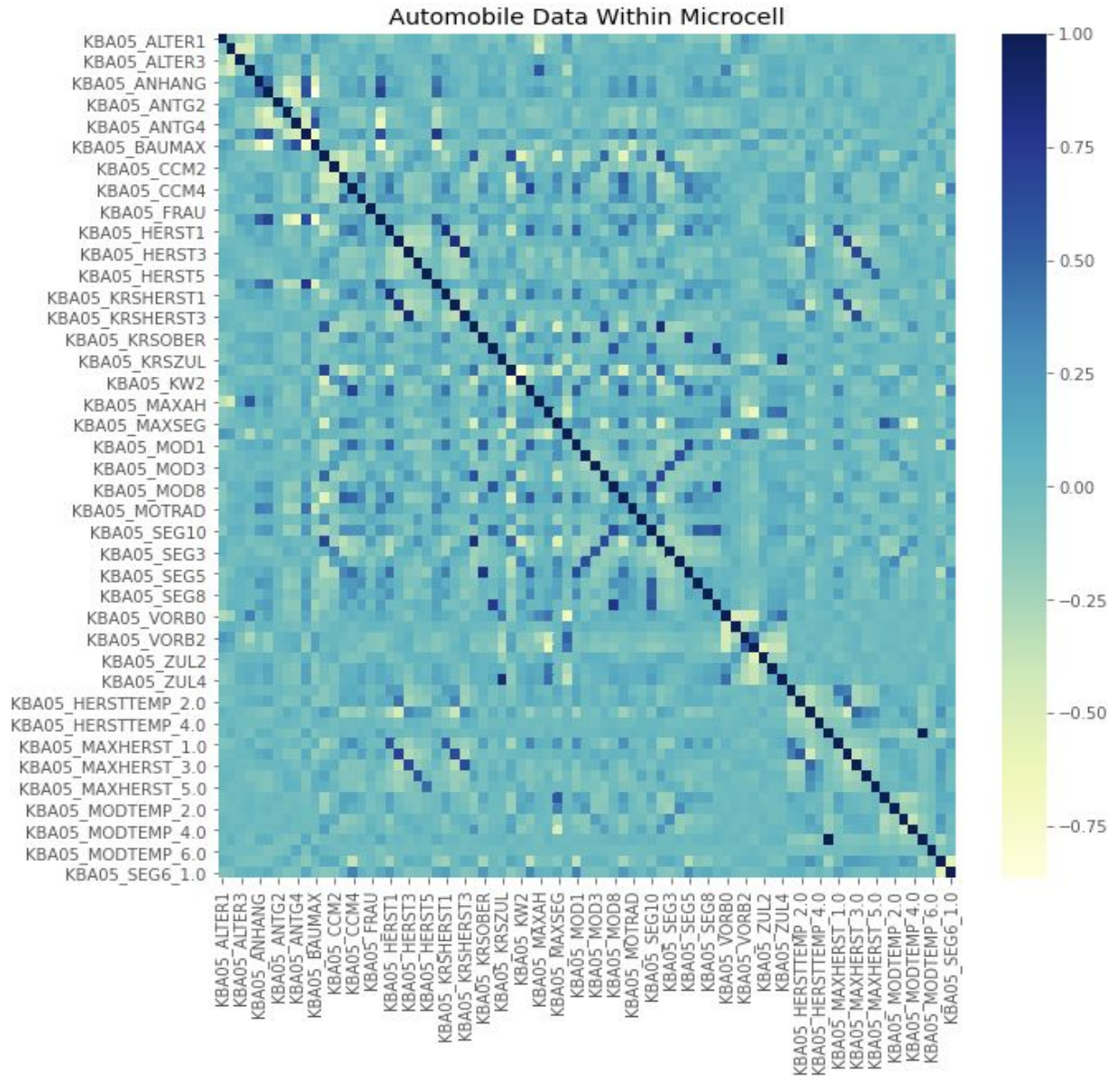
Justification

I believe I have done a great job at inspecting the Market Segmentation of Arvato's customer's versus the rest of the German population. There were many interesting insights and findings there including how different cultures affect their propensity to purchase organic mail-order products.

The prediction portion was very challenging. Given the highly imbalanced dataset we are dealing with and so many missing values we have found a model that significantly beats an AUC of 0.50 and is definitely better than blind guessing.

V. Conclusion

Free-Form Visualization



The visualization above is a correlation matrix of the KBA05_ prefix data. This is information on automobiles in the micro cell. It is interesting to see the different patches of heavy correlation and represents how complex and correlated all the features are in this dataset. I tried dropping these features and the resulting model showed a slight improvement, however in my interactions data these features showed back up again. There must be some kind of secondary effects going on here.

Reflection

I haven't had much experience dealing with a very large, completely dirty dataset. This was definitely a challenging experience and I have learned quite a bit.

The data preparation was definitely the most difficult part. I believe proper data cleaning is what is necessary to find good features and getting a reliable model.

The most interesting aspect of this project was trying to come up with new features and I really enjoyed trying to quantify which features and interactions would be best predictors.

I believe my findings are certainly useful for the problem but there is much room for improvement.

Improvement

Further data preparation would be a definite improvement and more visualizations of the features' histograms would be useful. There are so many features in this set one could analyze them for months I feel.

I believe that if I used my final solution as a benchmark there is definitely a better solution. It would just require more time and data mining through the dataset.

References

- [1] Arvato-Bertelsmann [Online]
Available: <https://www.bertelsmann.com/divisions/arvato/#st-1>
[Assessed October 30th, 2020]
- [2] Kaggle [Online]
Available: <https://www.kaggle.com/c/udacity-arvato-identify-customers>
[Assessed October 30th, 2020]
- [3] Accuracy [Online]
Available: https://en.wikipedia.org/wiki/Accuracy_and_precision
[Assessed October 30th, 2020]
- [4] AUC-ROC [Online]
Available: https://en.wikipedia.org/wiki/Receiver_operating_characteristic
Assessed October 30th, 2020
- [5] Recall [Online]
Available: https://en.wikipedia.org/wiki/Precision_and_recall
[Assessed October 30th, 2020]
- [6] One-hot-encoding [Online]
Available: <https://en.wikipedia.org/wiki/One-hot>
[Assessed October 30th, 3030]
- [7] Principal Component Analysis [Online]
Available: https://en.wikipedia.org/wiki/Principal_component_analysis
[Assessed November 5th, 2020]
- [8] K-Means Clustering [Online]
Available: https://en.wikipedia.org/wiki/K-means_clustering
[Assessed November 5th, 2020]
- [9] Elbow-method Clustering [Online]
Available: [https://en.wikipedia.org/wiki/Elbow_method_\(clustering\)](https://en.wikipedia.org/wiki/Elbow_method_(clustering))
[Assessed November 5th, 2020]
- [10] SMOTE [Online}
Available: https://rikunert.com/SMOTE_explained

[Assessed November 5th, 2020]

- [11] Standard Deviation between two means [Online]
Available: <https://stattrek.com/sampling/difference-in-means.aspx>
[Assessed November 27th, 2020]

- [12] XGBoost Original Paper [Online]
Available: <https://arxiv.org/pdf/1603.02754.pdf>
[Assessed November 30th, 2020]

- [13] What makes LightGBM so fast [Online]
Available:
<https://towardsdatascience.com/what-makes-lightgbm-lightning-fast-a27cf0d9785e>
[Assessed November 30th, 2020]

- [13] imblearn python package [Online]
Available: <https://imbalanced-learn.readthedocs.io/en/stable/index.html>
[Assessed November 30th, 2020]

- [14] Logistic Regression [Online]
Available: https://en.wikipedia.org/wiki/Logistic_regression
[Assessed November 30th, 2020]