Machine Learning Engineer Nano degree
Capstone Proposal

Nyaribo Maseru
March 10th, 2020

Customer Segmentation Using Data and Machine Learning Algorithms For Arvato Financial Solutions

I. **Definition**

Project Overview

This project is a partial fulfilment for Udacity's Machine Learning NanoDegree program. The project is based on real-life data science problem provided by Bertelsmann Arvato Analytics.In this project we are provided with demographic datasets of customers of a mail-order company in Germany, and demographic data of the general population of Germany.

The mission of the project is to make predictions based on data and machine learning algorithms on individuals who are mostly likely to become customers for a mail order sales company in Germany, instead of solely relying on gut feeling and intuition from senior experienced managers.

Problem Statement

The problem is how the mail-order company is to increase efficiency in customer acquisition process. Thus, instead of the company reaching out to all people in Germany, and then targeting them with marketing campaigns, it can just use the trained model to reach out to the people identified as becoming most likely new customers, and then to do targeted advertising.

The problem can be broken into two parts. The following are the approaches i that i shall use to solve the problem.

Part 1: Use unsupervised algorithms to identify customers to target for the mail campaign. This shall involve the following steps;

1. Cleaning and preprocessing the dataset
2. Dimensionality Reduction using PCA model.
3. Clustering using KMeans model.
4. Create a new datasets for target and no target features.

The steps shall involve analysis of demographics data for customers of a mail-order sales company in Germany, comparing it against demographics information for the general population. Then using PCA and KMean, apply the findings to the dataset with demographics information for targets of a marketing campaign for the company.

Part 2: Use a supervised model to predict which individuals are most likely to convert into becoming customers for a marketing campaign for the company. This shall involve the following steps;

1. Cleaning and preprocessing the dataset
2. Dimensionality Reduction using PCA model.
3. Clustering using KMeans model.
4. Split the Mailout_Train dataset into a training and test dataset

5. Use Amazon LinearLearner to Train and tune the model.
6. Test the model with the Mailout_Test dataset

We know the goal of the mail-order company is to increase efficiency towards customer acquisition for targeted advertising. The model design is optimised to know users who should NOT be targeted. That is, we want to have as few false positives (0s classified as 1s) as possible.

Metrics

For Part 1 of the solution problem is to reduce the features, with an explained variance of 0.85 of the preprocessed dataset. For clustering using KMeans i shall use the elbow method to cluster the data, while limiting the training of only 15 cluster. This is due to time it takes the algorithms to resolve the large data.

While for Part 2 of the solution, we do not want to optimise for accuracy only. Instead, we want to optimise for a metric that can help us decrease the number of false positives or negatives. In light of this, we want to build a model that has as many true positives and as few false negatives, as possible.

This corresponds to a model with a high recall: true positives / (true positives + false negatives). The matrix to use for the model shall be Recall.
I will assume that performance on a training set will be within about 5% of the performance on a test set. So, for a recall of about 0.85, I'll aim for a bit higher, 0.90%.

II. Analysis

Data Exploration

Part 1: The data files include the following;

1. Udacity_AZDIAS_052018.csv. The file has demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns). See below image.

```
Udacity_AZDIAS_052018.describe()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_
count	8.912210e+05	891221.000000	817722.000000	817722.000000	81058.000000	29499.000000	6170.000000	1205.000000	628274.000000	798073.0
mean	6.372630e+05	-0.358435	4.421928	10.864126	11.745392	13.402658	14.476013	15.089627	13.700717	8.2
std	2.572735e+05	1.198724	3.638805	7.639683	4.097660	3.243300	2.712427	2.452932	5.079849	15.6
min	1.916530e+05	-1.000000	1.000000	0.000000	2.000000	2.000000	4.000000	7.000000	0.000000	0.0
25%	4.144580e+05	-1.000000	1.000000	0.000000	8.000000	11.000000	13.000000	14.000000	11.000000	1.0
50%	6.372630e+05	-1.000000	3.000000	13.000000	12.000000	14.000000	15.000000	15.000000	14.000000	4.0
75%	8.600680e+05	-1.000000	9.000000	17.000000	15.000000	16.000000	17.000000	17.000000	17.000000	9.0
max	1.082873e+06	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000	25.000000	595.0

8 rows × 360 columns

```
Udacity_AZDIAS_052018.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	...	VHN	VK_D
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	11.0	...	4.0	
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	10.0	...	2.0	
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0	1.0	...	0.0	
4	910241	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	3.0	...	2.0	

5 rows × 366 columns

2. Udacity_CUSTOMERS_052018.csv. Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns). See below image.

```
Udacity_CUSTOMERS_052018.describe()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE
count	191652.000000	191652.000000	145056.000000	145056.000000	11766.000000	5100.000000	1275.000000	236.000000	139810.000000	141725.
mean	95826.500000	0.344359	1.747525	11.352009	12.337243	13.672353	14.647059	15.377119	10.331579	4.
std	55325.311233	1.391672	1.966334	6.275026	4.006050	3.243335	2.753787	2.307653	4.134828	14.
min	1.000000	-1.000000	1.000000	0.000000	2.000000	2.000000	5.000000	8.000000	0.000000	0.
25%	47913.750000	-1.000000	1.000000	8.000000	9.000000	11.000000	13.000000	14.000000	9.000000	1.
50%	95826.500000	0.000000	1.000000	11.000000	13.000000	14.000000	15.000000	16.000000	10.000000	1.
75%	143739.250000	2.000000	1.000000	16.000000	16.000000	16.000000	17.000000	17.000000	13.000000	4.
max	191652.000000	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000	25.000000	523.

8 rows × 361 columns

```
Udacity_CUSTOMERS_052018.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	...	VK_ZG11
0	9626	2	1.0	10.0	NaN	NaN	NaN	NaN	10.0	1.0	...	2.0
1	9628	-1	9.0	11.0	NaN	NaN	NaN	NaN	NaN	NaN	...	3.0
2	143872	-1	1.0	6.0	NaN	NaN	NaN	NaN	0.0	1.0	...	11.0
3	143873	1	1.0	8.0	NaN	NaN	NaN	NaN	8.0	0.0	...	2.0
4	143874	-1	1.0	20.0	NaN	NaN	NaN	NaN	14.0	7.0	...	4.0

5 rows × 369 columns

Part 2: The data files include the following;

1. Udacity_MAILOUT_052018_TRAIN.csv: Demographics data for individuals who were targets of a marketing campaign; 42 982 persons (rows) x 367 (columns). See below image.

```
mailout_train.describe()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AK
count	42962.000000	42962.000000	35993.000000	35993.000000	1988.000000	756.000000	174.000000	41.000000	34807.000000	35185.000
mean	42803.120129	0.542922	1.525241	10.285556	12.606137	13.783069	14.655172	14.195122	9.855058	6.706
std	24778.339984	1.412924	1.741500	6.082610	3.924976	3.065817	2.615329	3.034959	4.373539	15.151
min	1.000000	-1.000000	1.000000	0.000000	2.000000	5.000000	6.000000	6.000000	0.000000	0.000
25%	21284.250000	-1.000000	1.000000	8.000000	9.000000	12.000000	13.000000	13.000000	8.000000	1.000
50%	42710.000000	1.000000	1.000000	10.000000	13.000000	14.000000	15.000000	15.000000	10.000000	2.000
75%	64340.500000	2.000000	1.000000	15.000000	16.000000	16.000000	17.000000	17.000000	13.000000	7.000
max	85795.000000	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000	25.000000	438.000

8 rows × 361 columns

```
mailout_train.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	...	VK_DHT4A
0	1763	2	1.0	8.0	NaN	NaN	NaN	NaN	8.0	15.0	...	5.0
1	1771	1	4.0	13.0	NaN	NaN	NaN	NaN	13.0	1.0	...	1.0
2	1776	1	1.0	9.0	NaN	NaN	NaN	NaN	7.0	0.0	...	6.0
3	1460	2	1.0	6.0	NaN	NaN	NaN	NaN	6.0	4.0	...	8.0
4	1783	2	1.0	9.0	NaN	NaN	NaN	NaN	9.0	53.0	...	2.0

5 rows × 367 columns

2. Udacity_MAILOUT_052018_TEST.csv: Demographics data for individuals who were targets of a marketing campaign; 42 833 persons (rows) x 366 (columns). See below image.

```
mailout_test.describe()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AK
count	42833.000000	42833.000000	35944.000000	35944.000000	2013.000000	762.000000	201.000000	39.000000	34715.000000	35206.000
mean	42993.165620	0.537436	1.518890	10.239511	12.534029	13.942257	14.442786	14.410256	9.822584	6.749
std	24755.599728	1.414777	1.737441	6.109680	3.996079	3.142155	2.787106	2.279404	4.410937	14.839
min	2.000000	-1.000000	1.000000	0.000000	2.000000	4.000000	6.000000	9.000000	0.000000	0.000
25%	21650.000000	-1.000000	1.000000	8.000000	9.000000	12.000000	13.000000	13.000000	8.000000	1.000
50%	43054.000000	1.000000	1.000000	10.000000	13.000000	14.000000	15.000000	14.000000	10.000000	2.000
75%	64352.000000	2.000000	1.000000	15.000000	16.000000	17.000000	17.000000	16.000000	13.000000	7.000
max	85794.000000	3.000000	9.000000	21.000000	18.000000	18.000000	18.000000	18.000000	25.000000	379.000

8 rows × 360 columns

```
mailout_test.head()
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV	...	VHN	VK_DH
0	1754	2	1.0	7.0	NaN	NaN	NaN	NaN	6.0	2.0	...	4.0	
1	1770	-1	1.0	0.0	NaN	NaN	NaN	NaN	0.0	20.0	...	1.0	
2	1465	2	9.0	16.0	NaN	NaN	NaN	NaN	11.0	2.0	...	3.0	
3	1470	-1	7.0	0.0	NaN	NaN	NaN	NaN	0.0	1.0	...	2.0	
4	1478	1	1.0	21.0	NaN	NaN	NaN	NaN	13.0	1.0	...	1.0	

5 rows × 366 columns

Initial observations on the datasets.

The problem with the dataset for both parts of the project is that the data is heavily unbalanced. You have values that have different value type and missing NaN values. It is difficult to categorise the data to meaningful features without having a summary features from the Dias_Attributes and Dias_information.

Exploratory Visualisation

Below is an image of a summary file I created, that shows characteristic of the feature about the data, that shall aid during preprocessing of the data. The features are 276 (rows) x 3 (columns).

The features summary has a column **type** and **missing or unknown** column that has information about the value of the data (ordinal, numeric, mixed, interval and categorical), and that of values [-1, 0] that does not have any information.

```
feat_Info = mm.create_new_summary_features(df=Udacity_AZDIAS_052018, df_Att=Dias_Attributes, df_feat_summary=Feature_Summary)
```

```
# Features Summary
feat_Info.head()
```

	attribute	type	missing_or_unknown
0	AGER_TYP	categorical	[-1, 0]
1	ALTERSKATEGORIE_GROB	ordinal	[-1, 0, 9]
2	ANREDE_KZ	categorical	[-1, 0]
3	CJT_GESAMTTYP	categorical	[0]
4	FINANZ_MINIMALIST	ordinal	[-1]

```
# Feature Summary for type of data
feat_Info.groupby(['type']).size().reset_index(name='Count')
```

	type	Count
0	categorical	22
1	interval	1
2	mixed	7
3	numeric	7
4	ordinal	239

```
# Check shape of new feature summary
feat_Info.shape
```

```
(276, 3)
```

Below are the features to drop from the dataset that are not found in the summary file.

```
cols_to_drop = list(check_cols_dtypes)
cols_to_drop
```

```
['D19_LETZTER_KAUF_BRANCHE', 'EINGEFUEGT_AM']
```

Next is an image of the features that shall need to be re encoded that are found in the datasets.

```
# Binary_String columns are:
binary_str_attribute
```

```
['OST_WEST_KZ']
```

```
# Binary_Numeric columns are:
binary_num_attribute
```

```
['ANREDE_KZ', 'GREEN_AVANTGARDE', 'SOHO_KZ', 'VERS_TYP']
```

```
# Multi_level_attribute columns are:
multi_level_attribute
```

```
['AGER_TYP',
 'CJT_GESAMTTYP',
 'FINANZTYP',
 'GFK_URLAUBERTYP',
 'LP_FAMILIE_FEIN',
 'LP_FAMILIE_GROB',
 'LP_STATUS_FEIN',
 'LP_STATUS_GROB',
 'NATIONALITAET_KZ',
 'SHOPPER_TYP',
 'TITEL_KZ',
 'ZABEOTYP',
 'GEBAEUDETYP',
 'CAMEO_DEUG_2015',
 'CAMEO_DEU_2015',
 'D19_KONSUMTYP']
```

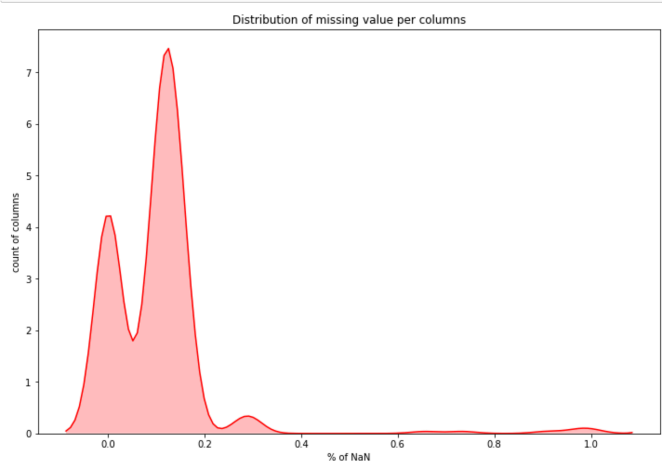
The below image shows features that shall require feature engineering.

```
# Get Mixedl Features

mixed_feat = list(feat_df['attribute'][feat_df['type'] == 'mixed'])
mixed_feat
```

```
['LP_LEBENSPHASE_FEIN',
 'LP_LEBENSPHASE_GROB',
 'PRAEGENDE_JUGENDJAHRE',
 'WOHNLAGE',
 'CAMEO_INTL_2015',
 'KBA05_BAUMAX',
 'PLZ8_BAUMAX']
```

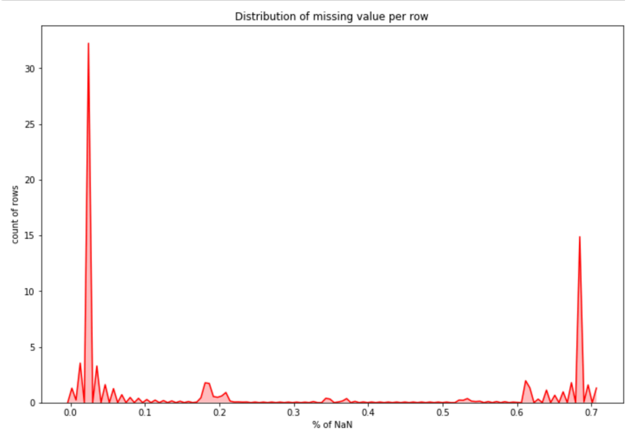
The below image shows the distribution of missing values on the columns.



Note on Missing values on columns:

- Most NaN Values fall above > 30% of the features columns. Thus ill shall drop columns above 30%.

The below image shows the distribution of missing values on the rows.



Note on Missing values on Rows:

- Most NaN Values fall below <= 50% of the rows. Thus ill shall drop rows <= 50% from the dataframe

Algorithms and Techniques

The objective of part 1 of the problem is to provide the mail company with a group customer to target. Once the datasets have been pre processed and cleaned, the next step is to do feature selection. To do this we shall need apply dimensionality reduction and clustering on both part 1 and part 2 of the project. This shall involve the use of PCA and K-Means.

While part 2 of the problem is to use a supervised technique to predict the response (0s or 1s) of the customer that were targeted to be positive or negative, I shall still use PCA and K-Means to cluster the data, but the model that I shall use for training and testing the data shall be Amazon`s LinearLearner model.

PCA Algorithm.
PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called *components*, which are composites of the original features that are uncorrelated with one another. They are also constrained so that the first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

K-Means Algorithm.
K-means is an unsupervised learning algorithm. It attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

Linear Learner Algorithm

Linear models are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples (*x*, *y*). *x* is a highdimensional vector and *y* is a numeric label. For binary classification problems, the label must be either 0 or 1. The best model optimises either of the following:

- Continuous objectives, such as mean square error, cross entropy loss, absolute error.
- Discrete objectives suited for classification, such as F1 measure, precision, recall, or accuracy.

Benchmark

In this section I shall use K-Means on the full datasets without PCA and use silhouette score as initial benchmark. Below is an image of the results. It is clear the best score is 0.048 n_clusters = 2.

Benchmark Using silhouette_score For Feature scaing

```
# %%time
# range_n_clusters = [2, 3, 4, 5, 6]

# for n_clusters in range_n_clusters:

#     clusterer = KMeans(n_clusters=n_clusters)
#     preds = clusterer.fit_predict(Azdias_scaled)
#     centers = clusterer.cluster_centers_

#     score = silhouette_score(Azdias_scaled, preds)
#     print("For n_clusters = {}, silhouette score is {})".format(n_clusters, score))
```

```
For n_clusters = 2, silhouette score is 0.04813678860782285)
For n_clusters = 3, silhouette score is 0.03484548825454606 6)
For n_clusters = 4, silhouette score is 0.0381720744295519)
For n_clusters = 5, silhouette score is 0.03146166789974724)
For n_clusters = 6, silhouette score is 0.02774136624361057)
CPU times: user 17h 21min 53s, sys: 1h 24min 5s, total: 18h 45min 59s
Wall time: 13h 2min 31s
```

Below is an image of the silhouette score on the reduced Azdias features of 214 with an explained variance of 0.85. It is clear the best score is 0.053 n_clusters = 2.

Using silhouette_score for clustering

```
# %%time
# range_n_clusters = [2, 3, 4, 5, 6]

# for n_clusters in range_n_clusters:

#     clusterer = KMeans(n_clusters=n_clusters)
#     preds = clusterer.fit_predict(pca_azdias)
#     centers = clusterer.cluster_centers_

#     score = silhouette_score(pca_azdias, preds)
#     print("For n_clusters = {}, silhouette score is {})".format(n_clusters, score))
```

```
For n_clusters = 2, silhouette score is 0.053798456867878625)
For n_clusters = 3, silhouette score is 0.039349699790037274)
For n_clusters = 4, silhouette score is 0.043010961524264646)
For n_clusters = 5, silhouette score is 0.03588469074109654)
For n_clusters = 6, silhouette score is 0.03192460736412464)
CPU times: user 7h 36min 9s, sys: 58min 5s, total: 8h 34min 14s
Wall time: 6h 4min 24s
```

Once we get the number of features to use, we use silhouette score to apply the n_components in this case 2 on the 214 reduced PCA features. The results shows that 0.58 are captured on cluster 1 on the general population. Below is an image of the results on Azdias dataset.

```
km = KMeans(n_clusters=2, random_state=42)
```

```
km_azdias = km.fit(pca_azdias)
```

```
labels_azdias = km_azdias.predict(pca_azdias)
```

```
class_count = pd.Series(labels_azdias).value_counts()

cls_azdias = pd.DataFrame({'cluster': class_count.index,
                           '% of data': 100*class_count.values/len(labels_azdias)})
cls_azdias['data'] = 'general population'

# print cls_azdias
cls_azdias
```

	cluster	% of data	data
0	1	58.224341	general population
1	0	41.775659	general population

For the customer segment we shall apply the same score silhouette score and reduced features as Azdias dataset. The results shows that 0.98 of customers are captured on cluster 1 of the data . The image below shows the customer and general population data.

```
# cluster predictions for the  customer demographics data.
customers_pca = pca.transform(Customer_scaled)
labels_customers = km.predict(customers_pca)
```

```
cus_class = pd.Series(labels_customers).value_counts()

cls_customers = pd.DataFrame({'cluster': cus_class.index,
                              '% of data': 100*cus_class.values/len(labels_customers)})
cls_customers['data'] = 'customers data'

# print cls_azdias
cls_customers
```

	cluster	% of data	data
0	1	94.91849	customers data
1	0	5.08151	customers data

Supervised Model Benchmark

For part 2 of the project, I chose two use several models to compare the results on unbalanced data. I then checked on the recall score. The models I used were Adaboost classifier and Logistic regression classifier. Below image is for Adaboost classifier. The recall score is 38.24 % with accuracy of 50.48%

```
# predict the target on the test dataset
abc_y_pred = abc_model_unsampled.predict(test_x)
print('\nTarget on test data',abc_y_pred)
```

```
Target on test data [0 0 0 ... 0 0 1]
```

```
# Calculation of accuracy
print ('accuracy_score: {:.2%}'.format(accuracy_score(y_feat,abc_y_pred )))

# Calculation of f1 score
print ('f1_score: {:.2%}'.format(f1_score(y_feat, abc_y_pred)))

# Calculation of recall_score
print ('recall_score: {:.2%}'.format(recall_score(y_feat, abc_y_pred)))

# Calculation of precision_score
print ('precision_score: {:.2%}'.format(precision_score(y_feat, abc_y_pred)))
```

```
accuracy_score: 50.48%
f1_score: 43.57%
recall_score: 38.24%
precision_score: 50.63%
```

Below image is for Logistic regression classifier. The recall score is 24.14% with accuracy of 49.55%

III. Methodology

Data Preprocessing
The following are the steps taken to handle the abnormal values and characteristics of the data.

- Step 1: Create a features summary that shall provide a benchmark on the features and values that need to be cleaned before data preprocessing. I shall then use this summary file and below on the other datasets. I shall use the Azdias dataset first, then later define a cleaning function and My_Module.py file to be used on part 2 since I shall use AWS SageMaker .
- Step 2: Convert the main dataset to have a column type, and column of missing or unknown values from the feature summary.
- Step 3: Remove columns that are not found in feature summary.
- Step 4: Features engineering - Use the summary feature to map the attribute values for values in Azdias.

```
# predict the target on the test dataset
lr_y_pred = lor_unsampled.predict(test_x)
print('\nTarget on test data',lr_y_pred)
```

```
Target on test data [1 1 0 ... 1 0 0]
```

```
# Calculation of accuracy
print ('accuracy_score: {:.2%}'.format(accuracy_score(y_feat,lr_y_pred )))

# Calculation of f1 score
print ('f1_score: {:.2%}'.format(f1_score(y_feat, lr_y_pred)))

# Calculation of recall_score
print ('recall_score: {:.2%}'.format(recall_score(y_feat, lr_y_pred)))

# Calculation of precision_score
print ('precision_score: {:.2%}'.format(precision_score(y_feat, lr_y_pred)))
```

```
accuracy_score: 49.55%
f1_score: 32.36%
recall_score: 24.14%
precision_score: 49.09%
```

- Step 4: Convert the missing or unknown values to numpy NaN values.
- Step 5: Remove outlier row columns that have 0.3 of NaN values of the dataset.
- Step 6: Remove outlier columns that have => 0.50 of NaN values of the dataset.
- Step 7: Data Normalization using sklearn Imputer and Scaler algorithms .

Implementation

The implementation for the project is broken down in three parts.

1. Dimensionality and reduction using PCA
2. K-Means clustering
3. LinearLearner with SageMaker

1. Dimensionality and reduction using PCA.

Implementation of PCA was done on both part 1 and part 2 of the project . The PCA model was trained on the preprocessed data, on two Jupyter Notebooks.

a. Arvato_Project_Workbook_Features_Final.ipynb
b. Arvato_Supervised_Learning_Model.ipynb on SageMaker .

Below are the steps taken to execute this section;
Step 1. Explore the preprocessed data
Step 2. Creating and Instantiating the PCA model
Step 3. Data variance
Step 4. Data variance vs dimensionality
Step 5. Component Makeup

2. K-Means clustering.

The next step in implementation is K-Means. Before training we have to determine the K value, then we use the reduced dimension data to train our k_Means model.

Below are the steps taken to execute this section;
Step 1. Determining the optimal number of clusters for K-Means clustering
Step 2. Creating and Instantiating the K-Means model
Step 3. Predicting customers labels
Step 4. Visualisation
Step 5. Natural groupings

3. LinearLearner with SageMaker

In this section we use the My_Module.py that contains functions I created from part 1 of the project to use on the Mailout_Test and Mailout_Train. To build our supervised model using Amazon SageMaker we will divide the section to the following steps;

Step 1. Load preprocessed data from S3
Step 2. Splitting the data
Step 3. Training Imbalanced data
Step 4. Create a LinearLearner Estimator
Step 5. Convert data into a RecordSet format
Step 6. Evaluating model.

Refinement

The data used on part 2 of the project was highly imbalanced. Below are steps taken to arrive at a good model.
Step 1. Train and tune for Recall
Step 2. Train and tune for precision and use hyper parameter -(positive_example_weight_mult)
Step 3. Train and tune for Recall and use hyper parameter -(positive_example_weight_mult)

1. Model tuned to Recall only result :

```
print('Metrics for simple, LinearLearner.\n')

# get metrics for linear predictor
metrics = evaluate(linear_predictor,
                   test_x_np,
                   test_y_np,
                   True) # verbose means we'll print out the metrics
```

```
Metrics for simple, LinearLearner.

predictions (cols)  0.0    1.0
actuals (rows)
0.0                 138  12590
1.0                   3    158

Recall:     0.981
Precision:  0.012
Accuracy:   0.023
```

```
# Deletes a precictor.endpoint
def delete_endpoint(predictor):
        try:
```

From the image, you can see that the model got a high Recall score of 0.981, but misclassified 12590 responses as False Positives.

2. Model tuned to precision and positive_example_weight_mult hyper parameter to sort out the imbalanced data.

```
[86]: # instantiate a LinearLearner to target_precision

# include params for tuning for higher recall
# *and* account for class imbalance in training data
linear_precision = LinearLearner(role=role,
                                 train_instance_count=1,
                                 train_instance_type='ml.c4.xlarge',
                                 predictor_type='binary_classifier',
                                 output_path=output_path,
                                 sagemaker_session=sagemaker_session,
                                 epochs=20,
                                 binary_classifier_model_selection_criteria='precision_at_target_recall',
                                 target_recall = 0.9,
                                 positive_example_weight_mult='balanced')

# create RecordSet
formatted_train_data = linear_precision.record_set(train_x_np, labels=train_y_np)
```

The image below shows the result of the above tuning set to precision as a target .

```
Metrics for balanced(Precision ), LinearLearner.

predictions (cols)   0.0   1.0
actuals (rows)
0.0                 5673  7049
1.0                   36   131

Recall:     0.784
Precision:  0.018
Accuracy:   0.450
```

```
test_np = reduced_test_feat.values.astype('float32')
```

```
print('Predict Mail_test_out with tuned balanced(Precision )')
prediction_batches = [precision_predictor.predict(batch) for batch in test_np]
test_preds = np.concatenate([np.array([x.label['predicted_label'].float32_tensor.values[0] for x in batch])
                             for batch in prediction_batches])
```

```
Predict Mail_test_out with tuned balanced(Precision )
```

```
# Resize the reduced_test_feat dataset
test_preds.resize((12889,),refcheck=False)
```

```
print('Accuracy {:.3%}'.format(accuracy_score(test_y_np, test_preds)))
print('f1_score {:.3%}'.format(f1_score(test_y_np, test_preds)))
print('recall {:.3%}'.format(recall_score(test_y_np, test_preds)))
print('precision {:.3%}'.format(precision_score(test_y_np, test_preds)))
```

```
Accuracy 44.68%
f1_score 2.25%
recall 49.10%
precision 1.15%
```

The model performed slightly better during the training , but who applied to the test dat, the recall score is 49.10% while accuracy is at 50%.

3. Model tuned to Recall and positive_example_weight_mult hyper parameter to sort out the imbalanced data.

The image below shows the result of the above tuning set for Recall as a target

```
[91]: %%time
# instantiate and train a LinearLearner to target_precision

# include params for tuning for higher precision
# *and* account for class imbalance in training data
linear_recall = LinearLearner(role=role,
                              train_instance_count=1,
                              train_instance_type='ml.c4.xlarge',
                              predictor_type='binary_classifier',
                              output_path=output_path,
                              sagemaker_session=sagemaker_session,
                              epochs=20,
                              binary_classifier_model_selection_criteria='recall_at_target_precision', # target recall
                              target_precision = 0.9,
                              positive_example_weight_mult='balanced')

# train the estimator on formatted training data
linear_recall.fit(formatted_train_data)
...
```

The image below shows the result .

```
Metrics for tuned balanced (recall), LinearLearner.

predictions (cols)   0.0   1.0
actuals (rows)
0.0                 8068  4654
1.0                   62   105

Recall:     0.629
Precision:  0.022
Accuracy:   0.634
```

```
prediction = [recall_predictor.predict(batch) for batch in test_np]
test_preds_recall = np.concatenate([np.array([x.label['predicted_label'].float32_tensor.values[0] for x in batch])
                            for batch in prediction])
```

```
test_preds_recall.resize((12889,),refcheck=False)
```

```
print('Predict Mail_test_out with tuned balanced (recall)')
print('Accuracy {:.2%}'.format(accuracy_score(test_y_np, test_preds_recall)))
print('f1_score {:.2%}'.format(f1_score(test_y_np, test_preds_recall)))
print('recall {:.2%}'.format(recall_score(test_y_np, test_preds_recall)))
print('precision {:.2%}'.format(precision_score(test_y_np, test_preds_recall)))
```

```
Predict Mail_test_out with tuned balanced (recall)
Accuracy 67.79%
f1_score 2.67%
recall 34.13%
precision 1.39%
```

```
cnf_matrix = confusion_matrix(test_y_np, test_preds_recall)
cnf_matrix
```

```
array([[8680, 4042],
       [ 110,   57]])
```

```
print ('True Positive: {}'.format(cnf_matrix[1][1]))
print ('True Negative: {}'.format(cnf_matrix[0][0]))
print ('False Positive: {}'.format(cnf_matrix[0][1]))
print ('False Negative: {}'.format(cnf_matrix[1][0]))
```

```
True Positive: 57
True Negative: 8680
False Positive: 4042
False Negative: 110
```

It is clear this is the best tuned model. The False positive are reduced while the accuracy and precision increased. Though the recall score reduced to 34%.

IV. Results

Model Evaluation and Validation

The LinearLearner is used as binary classifier. The algorithm allows us to focus on the minority class accuracy trying to maximise True Positives and minimises False Negative (Recall). The model as feature that takes care on highly unbalanced data ('positive_example_weight_mult = balanced') addition, SageMaker allows us to deploy the model and create an API in order to put the model in production. After training the model on the training data from mail_out.

For the Benchmark models that I used, the following was done to ensure robustness of the model.

Step 1. Oversample minority class. To compensate imbalance in data we need to resample our data. For this data we use oversampling which can be defined as adding more copies of the minority class. Oversampling can be a good choice when we don't have enough data to work with. In this project we will use the resampling module from Scikit-Learn to randomly replicate samples from the minority class. Below is an image of unstapled data;

```python
# Balance the data

from sklearn.utils import resample

# Separate input features and target
# y = label
# X = features
y = LABEL
X = FEATURES

# setting up testing and training sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# concatenate our training data back together
X = pd.concat([X_train, y_train], axis=1)
# X = pd.concat([pd.DataFrame(X_train), pd.DataFrame(y_train)], axis=1)
# separate minority and majority classes
postive_response = X[X.RESPONSE==0]
negative_response = X[X.RESPONSE==1]

# upsample minority
negative_response_upsampled = resample(negative_response,
                          replace=True, # sample with replacement
                          n_samples=len(postive_response), # match number in majority class
                          random_state=42) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([postive_response, negative_response_upsampled])

# check new class counts
upsampled.RESPONSE.value_counts()
```

```
1    31829
0    31829
Name: RESPONSE, dtype: int64
```

Step 2. Choose the model to use.

I used Sklearn`s GridSearchCross validation that uses stratifiedKfold on the model to get the same results on the model.

Below is an image of the function I used to get the model.

Find a benchmark model to use

```
import time
# Define get_classifier function to fit different classifiers on balanced data
# to find the best performing classifier algorithm
def get_classifier(clf, param_grid, X=X_feat, y=y_feat):

    # cross validation uses StratifiedKFold
    # scoring roc_auc available as parameter
    start = time.time()
    grid = GridSearchCV(estimator=clf, param_grid=param_grid, scoring='roc_auc', cv=5, verbose
=0)
    print("Training {} :".format(clf.__class__.__name__))
    grid.fit(X, y)
    end = time.time()
    time_taken = round(end-start,2)

    print(clf.__class__.__name__)
    print("Time taken : {} secs".format(time_taken))
    print("Best score : {}".format(round(grid.best_score_,4)))
    print("*"*40)

    return grid.best_score_, grid.best_estimator_, time_taken
```

Below is the result on the test classier to use.

	best_score	time_taken	best_est
XGBClassifier	0.633906	315.62	XGBClassifier(base_score=0.5, booster='gbtree'...
LogisticRegression	0.656174	12.50	LogisticRegression(C=1.0, class_weight=None, d...
RandomForestClassifier	0.506946	54.93	(DecisionTreeClassifier(class_weight=None, cri...
AdaBoostClassifier	0.588307	238.92	(DecisionTreeClassifier(class_weight=None, cri...
XGBClassifier	0.960828	396.09	XGBClassifier(base_score=0.5, booster='gbtree'...
LogisticRegression	0.762734	16.84	LogisticRegression(C=1.0, class_weight=None, d...
RandomForestClassifier	0.992677	29.22	(DecisionTreeClassifier(class_weight=None, cri...
AdaBoostClassifier	0.820870	244.34	(DecisionTreeClassifier(class_weight=None, cri...

Justification

Both models have not show a satisfying results, in these we have tried to compensate the imbalance of positive label and focus on recall to get the best predictive result for our minority class. Here below we are comparing results between three models:

	Recall	**Precision**	**Accuracy**	**F1**
`Linear Learner(tuned Precision)`	**49.10%**	**1.15%**	**44.68%**	**2.25%**
`Linear Learner(tuned recall)`	**34.13%**	**1.39%**	**67.79%**	**2.67%**
`Logisitic Regression`	**24.14%**	**49.09%**	**49.55%**	**32.36%**
`AdaBoost Classifier`	**38.24%**	**50.63%**	**50.48%**	**43.57%**

V. Conclusion

Reflection
In this project we tried different approaches and techniques on a model capable of predicting if a customer has the potential to respond positively to mail-order marketing campaign or not. However, after evaluation of the model, we can observe that our models are not performing well and some improvements has to be made. Additionally, we have not been able to establish a relation between our clustering model and the supervised data.

Improvement
I might consider using a custom Pytorch with Convolution Neural Network and try tune the model .

References

In this project i leveraged most of the materials from the Machine Learning Engineer Nanodegree classroom and student community at Udacity to complete the project. I am grateful to Udacity and Bertelsmann Arvato Analytics, for the datasets training for this project.

1. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
2. https://sagemaker.readthedocs.io/en/stable/linear_learner.html#sagemaker.LinearLearner
3. https://medium.com/@jaouad.eddadsi/finding-new-customers-using-data-and-machine-learning-algorithms-5da8bbeae798
4. https://medium.com/@venkateshrajagopalan86/customer-segmentation-for-arvato-financial-services-42ac87870b3c
5. https://docs.aws.amazon.com/sagemaker/latest/dg/k-means.html
6. https://docs.aws.amazon.com/sagemaker/latest/dg/pca.html
7. https://github.com/keyvantaj/Capstone_Project/blob/master/Project_Report.pdf
8.https://github.com/Jaouadeddadsi/DSND-Bertelsmann-Arvato-Project/blob/master/README.md

----