AWS Machine Learning Engineer Nanodegree

Capstone Report

Customer Segmentation for Arvato Financial Solutions

Domain Background

Arvato is a German services company that provides consulting services in the domain of financial services, information technology (IT), big data analytics, etc. It leverages predictive analytics, leading-edge platforms, and big data to deliver value to clients and help them see the bigger picture using data. It has a team of around 7000 IT, analytics, and legal experts in 15 countries with a focus on Europe[1].

In this project, Arvato aims to utilize demographic data to identify the attributes of customers who convert and use this information to better target marketing outreach campaigns for a client (a mail-order company selling organic products) on people who have a higher propensity to convert and become a customer.

Problem Statement

The problem statement for this project can be summarized as given the data for customers and the general population, identifying characteristics that differentiate customers. Also, use the customer information (characteristics like demographics, etc.) to predict whether the customer is likely to convert to a marketing campaign or not.

Datasets and Inputs

There are four data files associated with this project:

- `Udacity_AZDIAS_052018.csv`: Demographics data for the general population of Germany; 891 211 persons (rows) x 366 features (columns).
- `Udacity_CUSTOMERS_052018.csv`: Demographics data for customers of a mail-order company; 191 652 persons (rows) x 369 features (columns).
- `Udacity_MAILOUT_052018_TRAIN.csv`: Demographics data for individuals who were targets of a marketing campaign; 42 982 persons (rows) x 367 (columns).
- `Udacity_MAILOUT_052018_TEST.csv`: Demographics data for individuals who were targets of a marketing campaign; 42 833 persons (rows) x 366 (columns).

Additional metadata is also provided to better understand the columns and what they represent.

Solution Statement

The solution to this project can be broken down into two major sub-parts -

1. Customer Segmentation - Using unsupervised techniques such as clustering segment the customer and the general population data to identify common characteristics of people who become customers, that is, respond positively to our campaigns.

2. Supervised Learning Model - Build a machine learning model to predict whether an individual (given his/her characteristics) is likely to convert to a customer using historical mail-order campaigns data.

Benchmark Model

We can use a logistic regression model with default model parameters as a benchmark model for our classification task (1 - customer conversion, 0 - non-conversion). Using a simple logistic regression model is a common benchmark for classification tasks in machine learning.

Evaluation Metrics

We can use F1-Score[2] as an evaluation metric. This is because the dataset is highly imbalanced and using a standard metric like accuracy can give misleading results.

F1-Score takes into account both precision and recall and is defined as ->

$$F1\ Score\ =\ 2 * \frac{Precision * Recall}{Precision\ +\ Recall}$$

$$Precision\ =\ \frac{TP}{TP + FP},\ Recall\ =\ \frac{TP}{TP + FN}$$

$$Where,$$
$$TP\ =\ True\ Positive$$
$$FP\ =\ False\ Positive$$
$$FN\ =\ False\ Negative$$

Project Analysis

The following is a step-by-step analysis of the methods used to complete this project -

Data Cleaning and EDA

Columns with a large proportion of missing values
A number of columns had missing values with some columns having a high proportion of missing values. For example - ALTER_KIND1, ALTER_KIND2, ALTER_KIND3, ALTER_KIND4 have more than 90% of their values as missing.

On further examination, found that the columns EXTSEL992 and KK_KUNDENTYP also have a large proportion of missing values.

These columns were dropped during the data-preparation steps.

Examining Categorical Fields and their values
The following is a snapshot of the categorical fields in the dataset

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

Observations from the above categorical columns in the population dataset.

- The CAMEO_DEUG_2015, CAMEO_INTL_2015 columns are categorical fields storing categories as numeric values.
- The CAMEO_DEU_2015 and CAMEO_INTL_2015 columns have an additional value XX while the column CAMEO_DEUG_2015 has an additional value X, I'll assume it as missing or undefined.
- The EINGEFUEGT_AM columns store the date information and would need to be converted to datetime.
- The OST_WEST_KZ column has only two unique values representing the two regions - W for west and O for east.

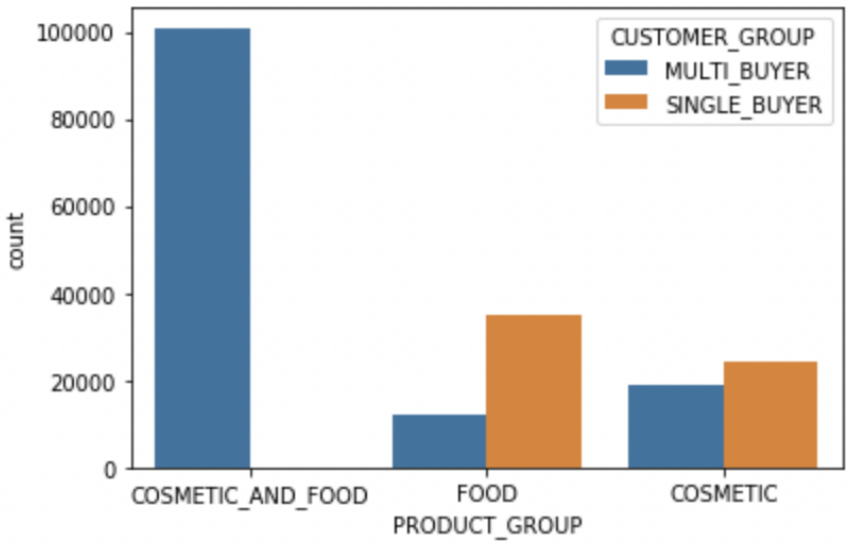
Customers dataset

The customers dataset has all the fields from the population dataset. In addition to those fields, it also has three customer-specific fields - `CUSTOMER_GROUP`, `ONLINE_PURCHASE`, `PRODUCT_GROUP`

The following is a snapshot of these fields -

	CUSTOMER_GROUP	ONLINE_PURCHASE	PRODUCT_GROUP
0	MULTI_BUYER	0	COSMETIC_AND_FOOD
1	SINGLE_BUYER	0	FOOD
2	MULTI_BUYER	0	COSMETIC_AND_FOOD
3	MULTI_BUYER	0	COSMETIC
4	MULTI_BUYER	0	FOOD

An interesting observation here is that customers in the "COSMETIC_AND_FOOD" product group almost exclusively are multi-buyers.



Date Cleaning Steps

After exploring both the population and the customers datasets, wrote a function to clean the data implementing the following steps -

1. Remove the columns that are not present in both the datasets - CUSTOMER_GROUP, PRODUCT_GROUP, and, ONLINE_PURCHASE.
2. Remove columns with large proportion of missing values - ALTER_KIND1, ALTER_KIND2, ALTER_KIND3, ALTER_KIND4, KK_KUNDENTYP, EXTSEL992.
3. In the categorical column, CAMEO_DEUG_2015 replace X with null and in the columns CAMEO_DEU_2015 and CAMEO_INTL_2015 replace XX with nulls.
4. Impute missing values. For categorical columns, replace missing values with the mode and for the numerical columns replace missing values with the median.
5. Convert the EINGEFUEGT_AM to pandas datetime type.

The following is a snapshot of the data cleaning function -

```
def clean_data(df, cols_to_drop):
    # drop columns not required
    df = df.drop(cols_to_drop, axis=1)

    # replace 'X' in CAMEO_DEUG_2015 with NaN
    df['CAMEO_DEUG_2015'] = df['CAMEO_DEUG_2015'].replace('X', np.nan)

    # replace 'XX' in CAMEO_DEU_2015 and CAMEO_INTL_2015 with NaN
    df['CAMEO_DEU_2015'] = df['CAMEO_DEU_2015'].replace('XX', np.nan)
    df['CAMEO_INTL_2015'] = df['CAMEO_INTL_2015'].replace('XX', np.nan)

    # impute missing values in categorical columns with mode
    df.fillna(df.select_dtypes(include='object').mode().iloc[0], inplace=True)

    # impute missing values in numerical columns with median
    df.fillna(df.select_dtypes(include='number').median().iloc[0], inplace=True)

    # convert EINGEFUEGT_AM to datetime
    df['EINGEFUEGT_AM'] = pd.to_datetime(df['EINGEFUEGT_AM'])

    return df
```

Feature Engineering

In this step, we add some additional features, encode categorical data, scale the features, perform dimensionality reduction with PCA.

Additional Temporal Features

Extracted time-based features such as the year, month, and day information from the datetime column EINGEFUEGT_AM

```
def add_temporal_features(df, date_col):
    # convert to datetime
    df[date_col] = pd.to_datetime(df[date_col])

    # extract temporal features
    df[date_col+"_Year"] = df[date_col].dt.year
    df[date_col+"_Month"] = df[date_col].dt.month
    df[date_col+"_Day"] = df[date_col].dt.day

    # drop the datetime column
    df = df.drop(date_col, axis=1)

    return df
```

Encoding Categorical Features

Categorical fields need to be converted to numerical form before they can be fed into algorithms such as clustering or classification. Used one-hot encoding to encode the categorical columns present the data.

```
def encode_cat_features(df, cat_cols):
    # get one-hot encoded features
    dummies = pd.get_dummies(df[cat_cols])
    # concat the features
    df = pd.concat([df, dummies], axis=1)
    # drop the categorical features
    df = df.drop(cat_cols, axis=1)

    return df
```

Perform Feature Scaling

Feature scaling is an important step, particularly when working with linear or distance-based models. It brings the features to a comparable scale. Since we plan to pass this data for PCA (Principal Component Analysis), it becomes an important step to accomplish. For this, StandardScaler from the sklearn module is used.

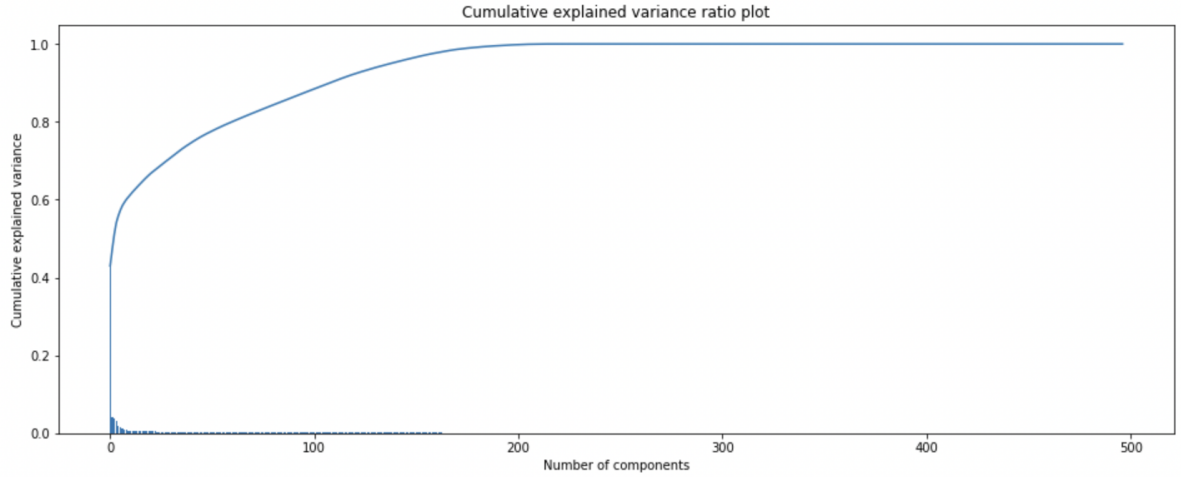
```
from sklearn.preprocessing import StandardScaler

# initialize the standard scaler
sc = StandardScaler()
# transform the population dataset
azdias_scaled_df = sc.fit_transform(azdias_df)
```

Dimensionality Reduction with PCA

PCA(Principal Component Analysis) is used here to reduce the high number of features present in the dataset.

We plot a PCA scree plot to determine the number of components to use.

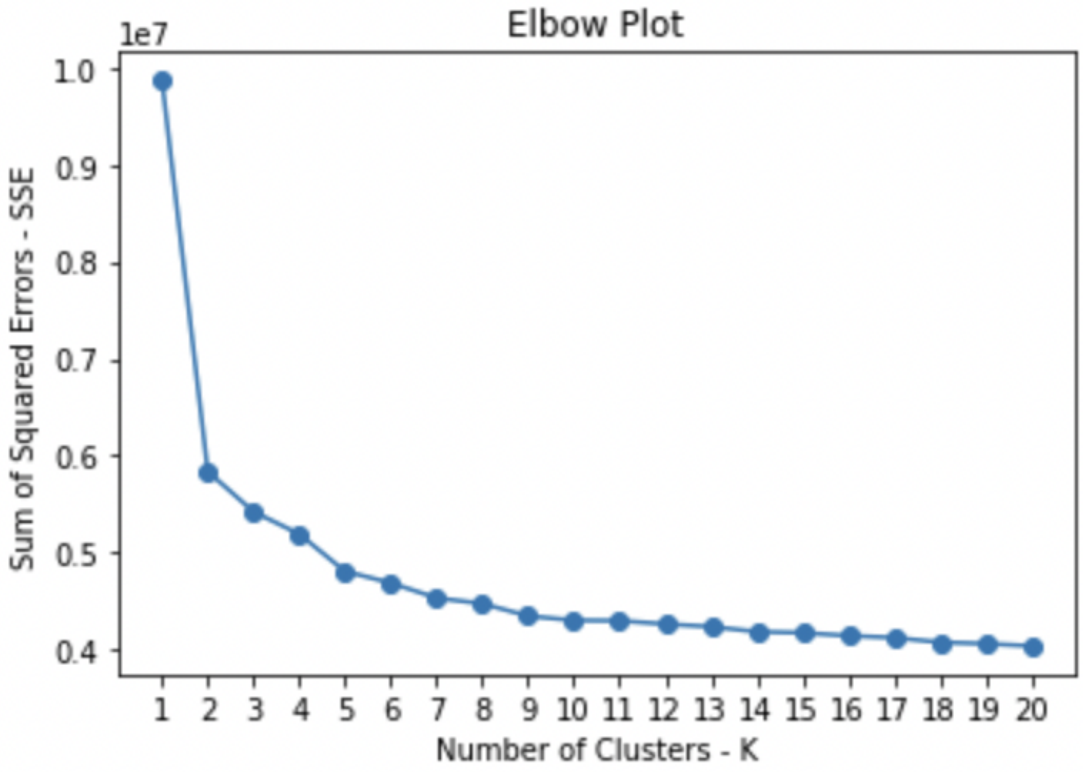


From the above figure, we can see that majority of the variance is explained by ~150 features (as against the ~500 total features present in the data).

Customer Segmentation

The data from the above step is used to perform customer segmentation using K-Means as the clustering algorithm.

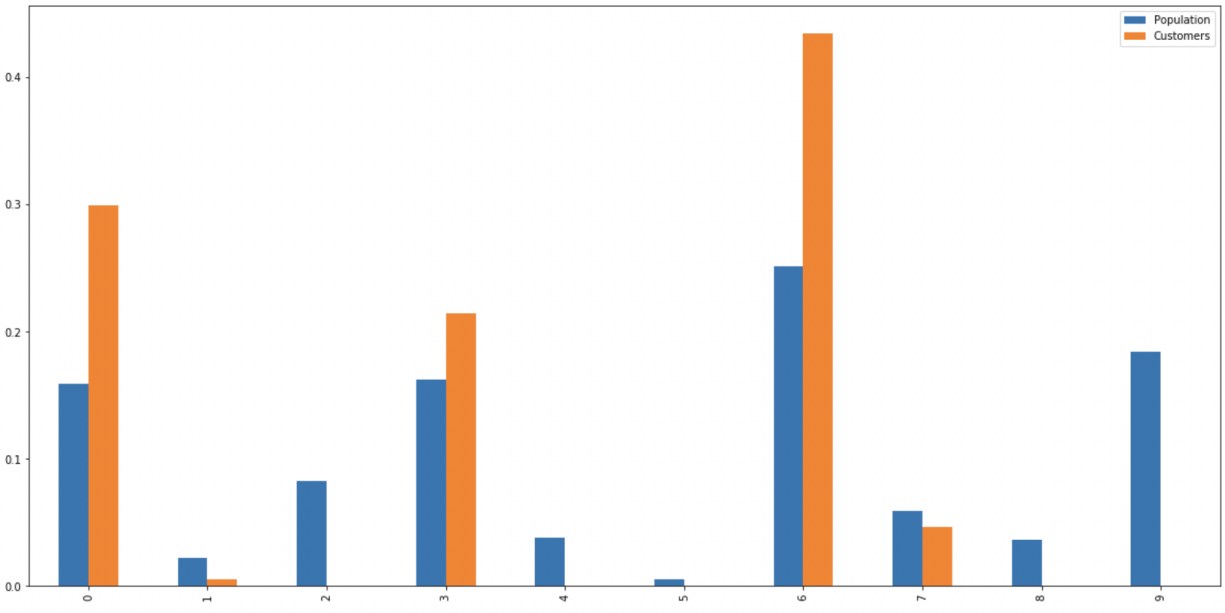
Since K is a hyperparameter, we used the elbow method to find the optimal number of clusters to use. In this method, we plot the sum of squared distances of each data point to its nearest cluster against K ranging from 1 to 20.



We find an elbow at k=2. Although we found an elbow in the plot but using k=2 will only result in two clusters, to get a more granular segmentation used k=10.

The clustering algorithm is trained on the populations dataset with K=10, we then use this trained model to predict the cluster labels for the population and the customers datasets.

The following chart compares the cluster distribution of both the population and the customers dataset.

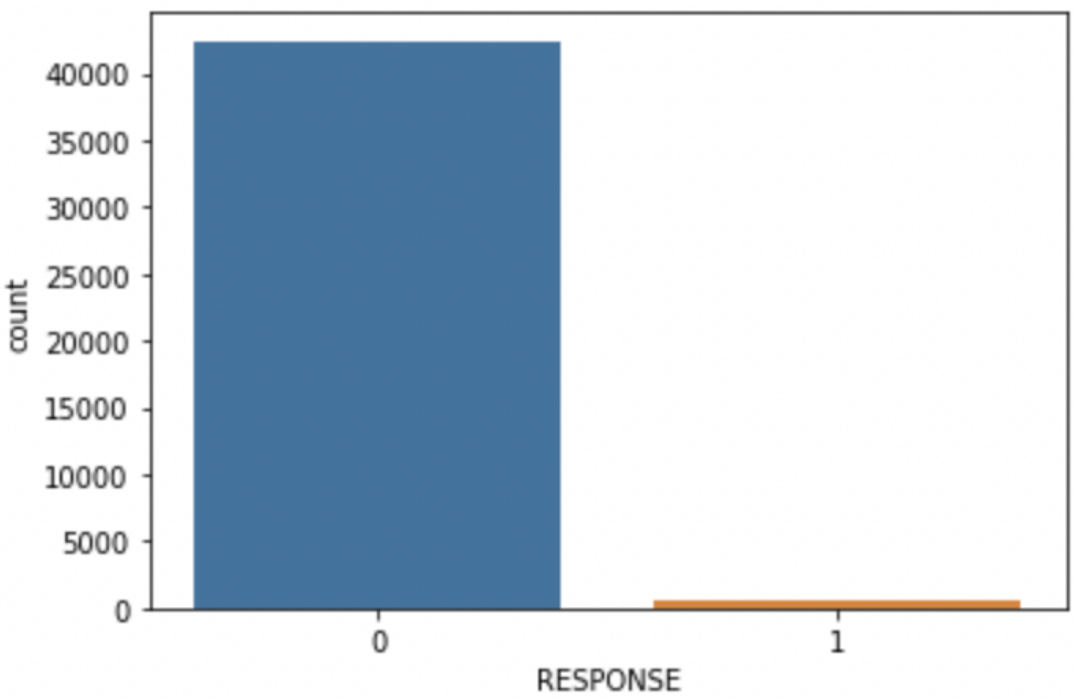


We can see that the datapoints in clusters 0, 1, 3, 6, and 7 share common characteristics between the population and the customers dataset. This can imply that these clusters carry characteristics that can help us better identify customers from the general population.

Model to predict customer conversion

The dataset has all the columns from the general population dataset (azdias) along with one additional column RESPONSE which is the target value to predict in the classification task.

The target value distribution looks as follows -



The dataset is imbalanced with a large number of datapoints having the label 0.

The same data cleaning, preprocessing, featurization, and scaling steps are applied to this data that we did to the population data in the above steps.

The data is then split into training and validation datasets. We use the training set to train the model and the validation set to test its performance.

The following models are trained on the training set -

- Logistic Regression (baseline)
- Decision Tree
- Random Forest

Let's look at the results of all the models -

Logistic Regression Model

```
# initialize the model
lr = LogisticRegression(max_iter=5000)
# fit the model on the train set
lr.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=5000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
# predict on the validation set
y_pred = lr.predict(X_val)
# f1-score on the validation set
print(f"F1 Score: {f1_score(y_val, y_pred)}")
print(f"ROC AUC Score: {roc_auc_score(y_val, y_pred)}")
```

```
F1 Score: 0.0
ROC AUC Score: 0.5
```

Decision Tree Model

```
# initialize the model
dtree = DecisionTreeClassifier()
# fit the model on the train set
dtree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# predict on the validation set
y_pred = dtree.predict(X_val)
# f1-score on the validation set
print(f"F1 Score: {f1_score(y_val, y_pred)}")
print(f"ROC AUC Score: {roc_auc_score(y_val, y_pred)}")
```

```
F1 Score: 0.043689320388349516
ROC AUC Score: 0.5184437109361275
```

Random Forest Model

```
# initialize the model
rf = RandomForestClassifier()
# fit the model on the train set
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

```
# predict on the validation set
y_pred = rf.predict(X_val)
# f1-score on the validation set
print(f"F1 Score: {f1_score(y_val, y_pred)}")
print(f"ROC AUC Score: {roc_auc_score(y_val, y_pred)}")
```

```
F1 Score: 0.0
ROC AUC Score: 0.49921433060967946
```

We get a very low (almost zero) F1-Score for all these models because the recall of the model is very bad due to the data being very heavily imbalanced.
Also, the model performance for models such as Decision Tree and Random Forest doesn't beat the baseline performance of the logistic regression model.

Improvement

As a potential area of improvement, we can use methods like Upsampling, downsampling, SMOTE, etc. to address the imbalance in the data.

Upsampling of the minority class is used to address this problem. After using upsampling we found a considerable improvement in model performance.

The following is the comparison with the baseline before upsampling.

Model	F1-Score	ROC AUC
Baseline (Logistic Regression)	0.0	0.5
Gradient Boosting	0.0	0.499

The following is the comparison with the baseline after using upsampling in the training dataset.

Model	F1-Score	ROC AUC
Baseline (Logistic Regression)	0.037	0.602
Gradient Boosting	0.067	0.689

References

1. Arvato Financial Services Website - https://finance.arvato.com/en/about-us/
2. F1 Score - Wikipedia - https://en.wikipedia.org/wiki/F-score