Arvato Financial Solutions Case Study

Augustine Chang

Company Overview

Arvato Financial Solutions, a Bertelsmann subsidiary, provides debt collection services by implementing strategies in effective user identification and advanced fraud detection for an overall better decision making. The company effectively helps secure financial transactions between end-users, and offers various products and services, including but not limited to:

- Product and customer service
- Activation and management of financial agreements
- Administration of payment methods and credit reference
- System and data management

Project Domain

Arvato, the services company, is looking to help its client, a mail company based in Germany to better understand its customer segments and identify the most probable customers. Dataset is provided by Arvato.

Company's marketing goal is to target an audience that are most likely to respond to a mail and convert to a customer, by applying customer segmentation techniques such as k-means cluster to identify people that are similar into k number of groups, based on distance between the similarity of the features.

About data

For this particular project, Arvato is looking to analyze demographics data for customers of mail-order sales company in Germany. Provided are following datasets:

- **Customer demographic information** (191,652 customers, 369 features), which include categories of: person, household, building, Microcell, Grid, Postcode, RR1_ID, PLZ8, Community. Additional 3 columns indicate the customer group, product group, and purchase group to which the person belongs
- **Population demographic information** (891,221 people, 366 features)
- Data describing the demographic information and range of existing values
- **Training and testing data**, including labels for whether a customer converted or not given in training data (training set 42,962 people, 366 feature columns & testing set 42,833 people, 366 feature columns)

- Binary labels (0 meaning yes, the person converted to a customer or 1 meaning no, the person did not convert)

Problem Statement

Targeting consumers has mainly been driven by business experience and intuition, making it hard to predict or track the success of campaigns. With data, we can make more accurate predictions that lead to a greater outcome (i.e., increase in profitability).

There are two problems that need to be addressed. First, the client company is looking to identify people in Germany who are most likely to become a customer. The proposed solution is to identify and segment customers based on their attributes using pca analysis and unsupervised machine learning algorithms, such as k means cluster.

Second problem is to classify given demographic information of some people whether they will become a customer or not. This is a binary classification problem. I will be predicting the probability of customer making a purchase given a mail ad- binary classification problem using models such as logistic regression, which takes in demographic features as inputs and outputs the likelihood that a given person will convert to a purchase (notified as 1) or not convert (notified as 0). I will then test the model on unseen data points (other customers)- predict on fit model. Considering boosting methods to learn from iterations of weaker models.

A consistent approach to preprocessing data will be taken for all 3 steps and provided as a function. Details to data wrangling is outlined in the Data section.

Metrics

I used the roc auc score to evaluate performance of models.

To fully understand the importance of ROC AUC score, consider the case of a random classifier, which assigns the score/ probability between zero and one for each instance. If we choose a threshold of 0.5, the score above the threshold will be positive, and everything under the threshold will be predicted as negative. Now as this threshold varies, the True Positive Rate (proportion of positive cases identified correctly) and False positive Rate (proportion of negative cases identified correctly) will vary. This probabilistic relationship is captured by the ROC, and the area under the ROC is the score that we will measure.

Data Exploration and Preprocessing

Following problems exists within the dataset, which requires additional cleaning:

Inconsistency in data types, on columns CAMEO_DEUG_2015 and CEMO_INTL_2015. Unknown variables have a string type format denoted ‘X’ or ‘XX’, while the rest are float types. I applied a lambda function to quickly replace string values with np.nan.

Check columns 18, 19

```
azdias.iloc[:, 18].unique()
```

```
array([nan, 8.0, 4.0, 2.0, 6.0, 1.0, 9.0, 5.0, 7.0, 3.0, '4', '3', '7',
       '2', '8', '9', '6', '5', '1', 'X'], dtype=object)
```

```
azdias.iloc[:, 19].unique()
```

```
array([nan, 51.0, 24.0, 12.0, 43.0, 54.0, 22.0, 14.0, 13.0, 15.0, 33.0,
       41.0, 34.0, 55.0, 25.0, 23.0, 31.0, 52.0, 35.0, 45.0, 44.0, 32.0,
       '22', '24', '41', '12', '54', '51', '44', '35', '23', '25', '14',
       '34', '52', '55', '31', '32', '15', '13', '43', '33', '45', 'XX'],
      dtype=object)
```

```
customers['CAMEO_DEUG_2015'].unique()
```

```
array([1.0, nan, 5.0, 4.0, 7.0, 3.0, 9.0, 2.0, 6.0, 8.0, '6', '3', '8',
       '9', '2', '4', '1', '7', '5', 'X'], dtype=object)
```

```
# change 'X' values to np.nan
customers['CAMEO_DEUG_2015'] = customers['CAMEO_DEUG_2015'].apply(lambda x: np.nan if x == 'X' else x)
customers['CAMEO_INTL_2015'] = customers['CAMEO_INTL_2015'].apply(lambda x: np.nan if x == 'XX' else x)
```

I then converted the unknown values that were indicated as -1’s and 0’s (sometimes 10’s) by importing the excel sheet that explains the range of data for customers and demographics data. After cleaning up this file, I iterated through each of the columns in customers and demographics dataframe to replace unknown values with np.nan.

Convert unknown values to nan's

```
In [13]: df = pd.read_excel('DIAS Attributes - Values 2017 (1).xlsx', index_col=None, header=1)

In [14]: df = df.drop('Unnamed: 0', axis = 1)

In [15]: df = df[df.Meaning.str.contains('known') == True].dropna().reset_index(drop = True)

In [16]: df['Value'] = df['Value'].apply(lambda x: list(map(int, x.split(','))) if ',' in str(x) else [x])

In [17]: df1 = pd.DataFrame(df.Value.tolist(), columns = ['value1', 'value2'])

In [18]: df1['Attribute'] = df['Attribute']

In [19]: df1.head()
```

Out[19]:

	value1	value2	Attribute
0	-1	NaN	AGER_TYP
1	-1	0.0	ALTERSKATEGORIE_GROB
2	0	NaN	ALTER_HH
3	-1	0.0	ANREDE_KZ
4	-1	NaN	BALLRAUM

```
In [20]: common = list(set(df.Attribute).intersection(list(customers.columns)))

In [21]: eplace unknown values with NAN
         column in tqdm(common):
         customers[column] = customers[column].replace({np.int64(df1[df1['Attribute'] == column]['value1'].values[0]): np.nan})
         customers[column] = customers[column].replace({np.int64(df1[df1['Attribute'] == column]['value2'].values[0]): np.nan})

         100%|██████████| 233/233 [00:01<00:00, 143.00it/s]
```

Meaningless rows that lack substantial data need to be filled in and / or dropped. I first iterated through each column to drop those that have more than 50% null values, which turned out to be 16 columns: ['ALTER_KIND1', 'ALTER_KIND2', 'ALTER_KIND3', 'ALTER_KIND4', 'D19_BANKEN_ANZ_12', 'D19_BANKEN_ANZ_24', 'D19_GESAMT_ANZ_12', 'D19_TELKO_ANZ_12', 'D19_TELKO_ANZ_24', 'D19_VERSAND_ANZ_12', 'D19_VERSAND_ANZ_24', 'D19_VERSI_ANZ_12', 'D19_VERSI_ANZ_24', 'KBA05_BAUMAX', 'KK_KUNDENTYP', 'TITEL_KZ']

Next, I dropped the rows with more than 50% nan values, which turns out to be 50786 rows. I replace the rest of nan values with the most frequent value within each column.

I label encoded 8 categorical variables as final step in data preprocessing: ['D19_LETZTER_KAUF_BRANCHE', 'CAMEO_DEUG_2015', 'PRODUCT_GROUP', 'CUSTOMER_GROUP', 'EINGEFUEGT_AM', 'CAMEO_INTL_2015', 'OST_WEST_KZ', 'CAMEO_DEU_2015']

```
# Drop columns that have more than 50% nan values
nan_pct = pd.DataFrame(data.isnull().sum() * 100 / len(data), columns = ['percentage'])
data = data.drop(list(nan_pct[nan_pct['percentage'] > 50].index), axis=1)

# drop rows that have more than 50% nan values
nan_pct_row = pd.DataFrame(data.isnull().sum(axis=1) * 100 / len(list(data.columns)), columns=['percentage'])
data = data.drop(list(nan_pct_row[nan_pct_row['percentage'] > 50].index))

nan_column = data.columns[data.isna().any()].tolist()
# fill nan with most common value
for column in nan_column:
    data[column] = data[column].fillna(data[column].value_counts().index[0])

cat_var = list(set(data.columns) - set(data._get_numeric_data().columns))
data[cat_var] = data[cat_var].astype(str)
labelencoder = LabelEncoder()
for col in cat_var:
    data[col] = labelencoder.fit_transform(data[col])
```

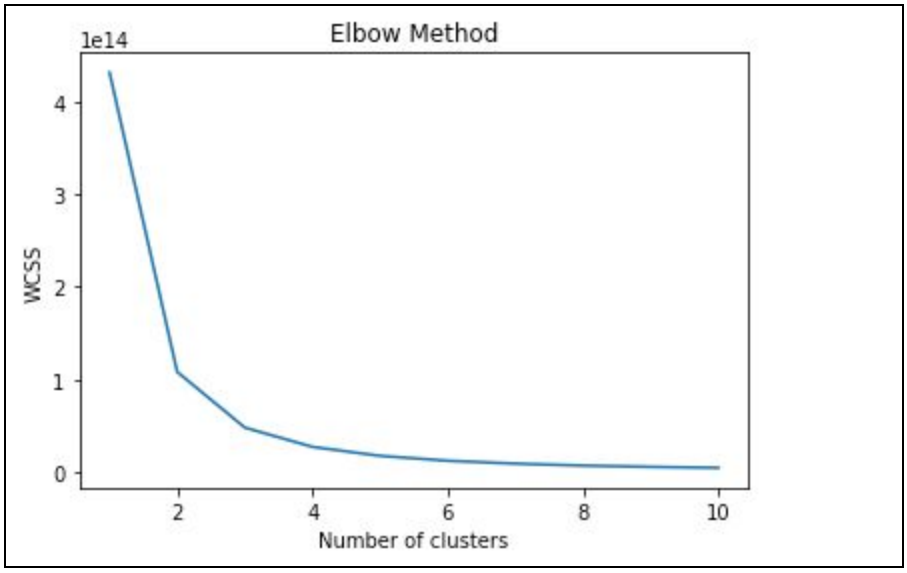
Implementations

Implementation of algorithms is divided into 2 parts - the customer segmentation model using k means cluster, and application of various binary classifiers to evaluate which person will most likely convert to a customer.

Modeling K-means cluster

After data preprocessing, I applied feature selection by using pca on the clean data, reducing variables for both customer and population database down to 150.

I used the elbow method to identify the ideal number of clusters to use as input for the k means cluster model. I decided to go with 4 clusters.



In implementing the k-means cluster for both customers and population groups, I find that there are clusters 0 and 1 in the customer group that represent the majority of the population group, based on the number of people that belong to the group. Therefore, the company can target this population as the future customers.

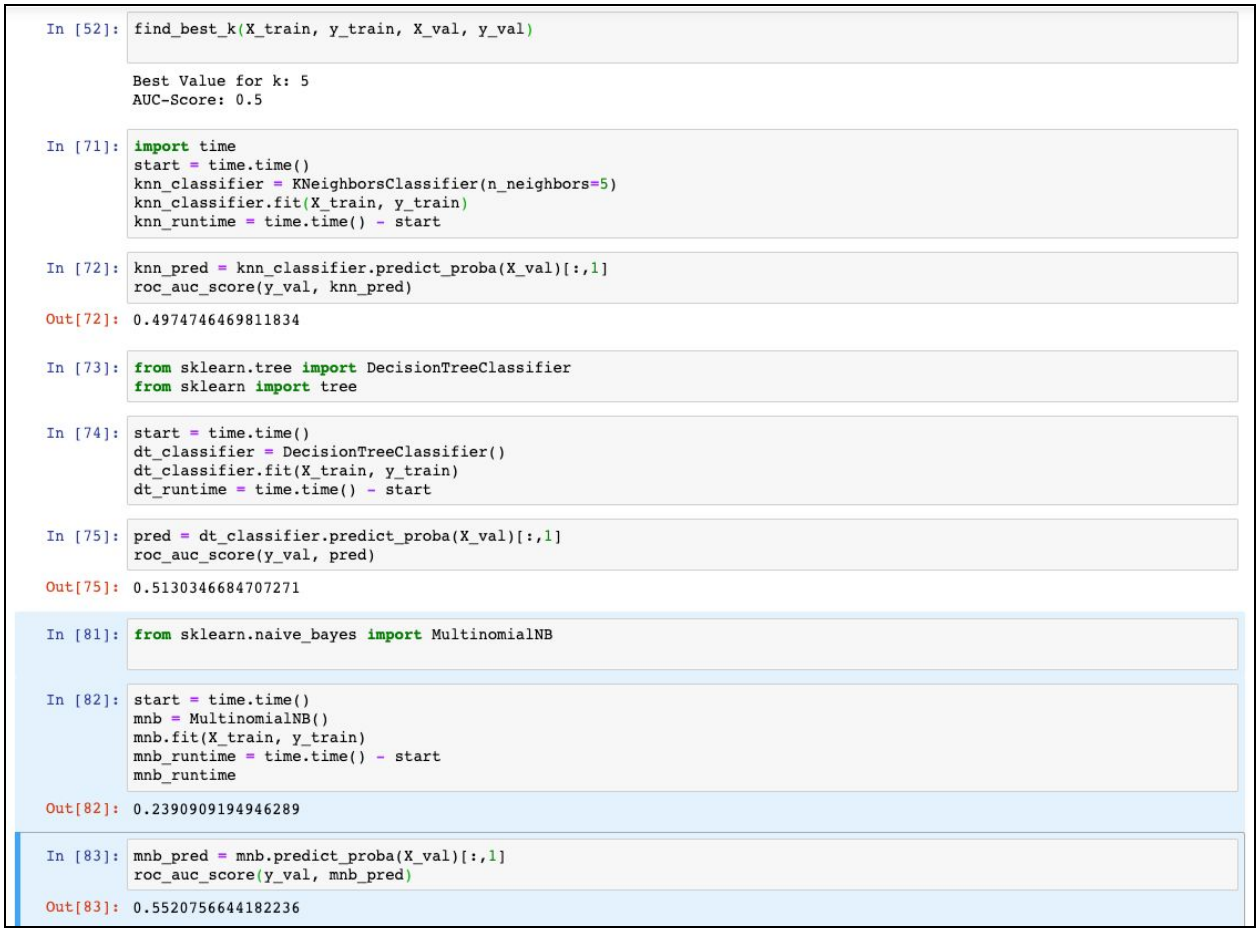
```
In [41]: # sample k means
         kmeans = KMeans(n_clusters=4, init='k-means++', max_iter=300, n_init=10, r
         kmeans.fit(customers1)

Out[41]: KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
                n_clusters=4, n_init=10, n_jobs=None, precompute_distances='auto',
                random_state=0, tol=0.0001, verbose=0)

In [42]: customers_clusters = kmeans.predict(customers1)
         customers_clusters = pd.Series(customers_clusters)
         customers_clusters.value_counts().sort_index()

Out[42]: 0    35225
         1    35374
         2    35077
         3    35190
         dtype: int64

In [43]: azdias_clusters = kmeans.predict(azdias1)
         azdias_clusters = pd.Series(azdias_clusters)
         azdias_clusters.value_counts().sort_index()

Out[43]: 0    353193
         1    352558
         2     42599
         3     42899
         dtype: int64
```

Modeling the binary classifier

In this binary classification problem, the plan is to predict which part of the population are more likely to become customers of the mail order company. Given the train dataset of 42,962 rows and 366 columns, I applied previous preprocessing techniques to clean data for downstream process of modeling. I held out the RESPONSE column as the target variable, which is a binary that indicates whether the person has converted to a customer (1) or not (0). I chose logistic regression as the benchmark model to predict for this, which performed at 74% ROC AUC score.

I split the train and test data at 8 to 2 ratio, and stratified the dataset to ensure class balance during training. Then I tested a few models to see what works best. It is interesting to see how models have performed around similar roc auc scores at two levels.

The K Nearest Neighbors with best k at 5 performed at around 50% roc auc score, as well as the decision tree and multinomial naive bayes classifier, which on average total performed around 52% and unfortunately did worse than the benchmark model.



```
In [52]: find_best_k(X_train, y_train, X_val, y_val)

         Best Value for k: 5
         AUC-Score: 0.5

In [71]: import time
         start = time.time()
         knn_classifier = KNeighborsClassifier(n_neighbors=5)
         knn_classifier.fit(X_train, y_train)
         knn_runtime = time.time() - start

In [72]: knn_pred = knn_classifier.predict_proba(X_val)[:,1]
         roc_auc_score(y_val, knn_pred)

Out[72]: 0.4974746469811834

In [73]: from sklearn.tree import DecisionTreeClassifier
         from sklearn import tree

In [74]: start = time.time()
         dt_classifier = DecisionTreeClassifier()
         dt_classifier.fit(X_train, y_train)
         dt_runtime = time.time() - start

In [75]: pred = dt_classifier.predict_proba(X_val)[:,1]
         roc_auc_score(y_val, pred)

Out[75]: 0.5130346684707271

In [81]: from sklearn.naive_bayes import MultinomialNB

In [82]: start = time.time()
         mnb = MultinomialNB()
         mnb.fit(X_train, y_train)
         mnb_runtime = time.time() - start
         mnb_runtime

Out[82]: 0.2390909194946289

In [83]: mnb_pred = mnb.predict_proba(X_val)[:,1]
         roc_auc_score(y_val, mnb_pred)

Out[83]: 0.5520756644182236
```

The Boosting models (adaboost and gradient boosting models) have quickly learned from its weaker models to perform at an average of 79% roc auc score. The final winner was the gradient boosting classifier, which performed on the test set at 80% roc auc score. These models have in comparison to the benchmark logistic regression model performed 6% better.

```
In [76]: from sklearn.ensemble import AdaBoostClassifier

In [70]: start = time.time()
         adaboost_clf = AdaBoostClassifier()
         adaboost_clf.fit(X_train, y_train)
         ab_runtime = time.time() - start
         ab_pred = adaboost_clf.predict_proba(X_val)[:,1]
         roc_auc_score(y_val, ab_pred)

Out[70]: 0.7840052132041307

In [77]: from sklearn.ensemble import GradientBoostingClassifier

In [80]: start = time.time()
         gbt_clf = GradientBoostingClassifier()
         gbt_clf.fit(X_train, y_train)
         gb_runtime = time.time() - start
         gb_pred = gbt_clf.predict_proba(X_val)[:,1]
         roc_auc_score(y_val, gb_pred)

Out[80]: 0.8032977164043482
```