Machine Learning Engineer Nanodegree

Capstone Project

Jair Miranda
April 29st, 2021

Project Overview

Customer segmentation looking to general population data, and Arvato Company clients data, the idea identify with general population data those who have the greatest chance of becoming customers. In the project I used a lot of different algorithms and also created pipelines to clean up the data, and made some feature engineering.

Problem Statement

Customer segmentation is a common problem on machine learning projects, mainly if you're looking to improve the marketing campaign results. Arvato Financial Solucions has a lot of demographic and customer data, and we ll try to find patterns on this data, understanding the difference between customers and non-customers. We have train, test dataset too, but all 4 datasets have a serious problem with nullity data, and wrong types. The most interesting thing about this project is that it is a real problem, and we can violate it in any company, besides that having a good result can generate high profitability for the company

It was hard to deal with all data, heavy for the memory.

Data 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).

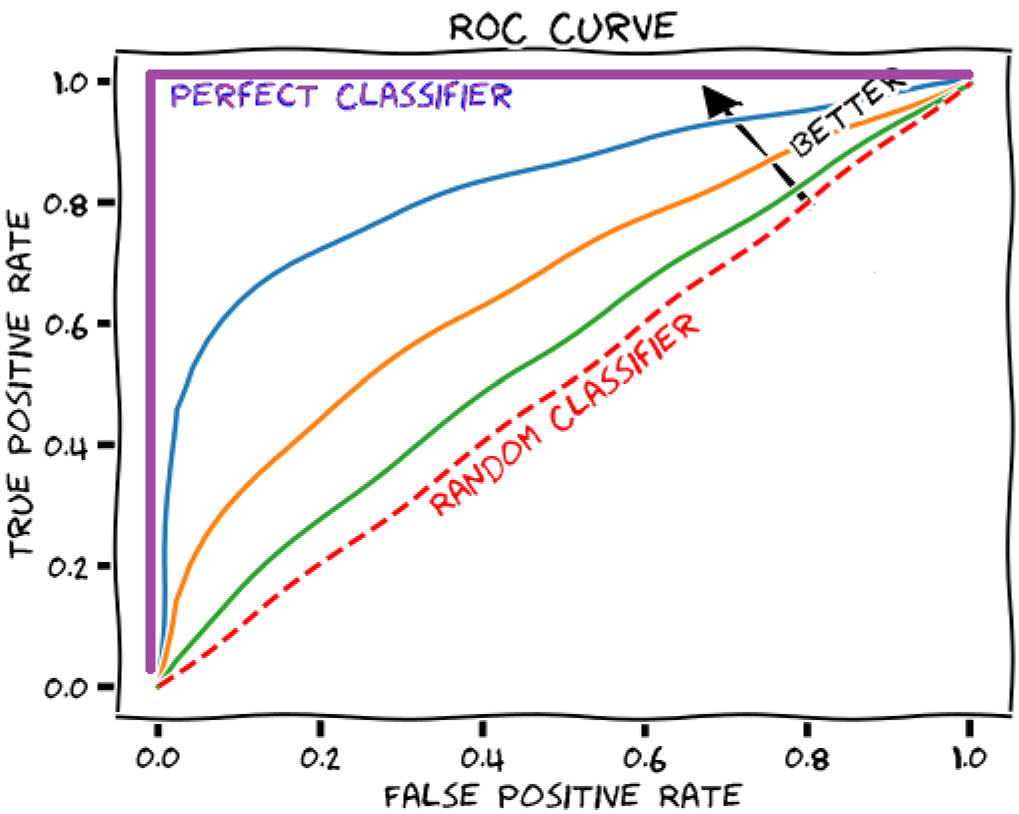
Solution Statement.

We ll start analyse the data and use catboost to check feature importance The goal is to create a model or model pipeline to understand what's the change of new lead to be a client for that we ll:

```
1- Analyse the Data.
2- Clean Up the Data.
3- Create Data Wrangler pipeline.
4- Make Unsupervided Analyses.
5- Create Unsupervided Pipeline.
6- Make Supervised Learning.
7- Create Supervised Pipeline.
8- Evaluate.
```

Metrics

Measuring Performance: AUC (AUROC)



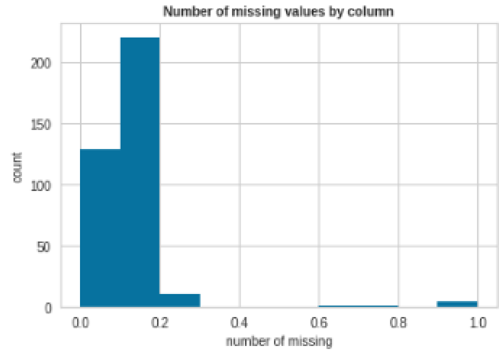
Analyses

I used Azdias dataset to make analyses

Most of the columns have less than 20% missing values. I drop those columns that have greater than 20% missing values And below graph shows the top 50 names of columns along with its missing percent.

In [14]:
```
plt.hist(missing_by_col)
plt.title("Number of missing values by column",fontsize=10,fontweight="bold")
plt.xlabel("number of missing",fontsize=10)
plt.ylabel("count",fontsize=10)
```

Out[14]: `Text(0, 0.5, 'count')`



alt text I also removed any columns that are not in attr or info dataframe.

After that we moved to 243 columns, better than 366.

After that I did a lot of transformation in different columns, and create a pipeline for feature selection, you can check here -> alt text

Algorithms and Techniques

First I apply catboost to discovery the feature importance and after that I saw 80 of values = 1 in D19_SOZIALES are customers. Unfortunately I don't have description for these columns. I can use later to reach better results.

After That I did some transforms using sklearn pipeline and Column Transform:

Time to create or pipeline of transformation

In [16]:
```
# Using Pd Get Dummies to transform category columns to Dummies

azdias_df = pd.get_dummies(azdias_df, columns=CATEGORICAL_COLUMNS)
```

In [21]:
```
# Saving Columns to compare with others dataframes
import json

with open('../data/cleaned_data/azdias_columns.json', 'w') as jsonfile:
    json.dump({'columns':azdias_df.columns.to_list()}, jsonfile)
```

In [17]:
```
#Numeric transformation
numeric_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# For Binary
binary_pipeline = SimpleImputer(missing_values=np.nan, strategy='most_frequent')

# Log Transform
transformer = FunctionTransformer(np.log1p)

log_pipeline = Pipeline(steps=[
    ('natural_log', transformer),
    ('imputer', SimpleImputer(missing_values=np.nan, strategy='median')),
    ('scaler', StandardScaler())
])
```

In [18]:
```
transformers = [('numeric', numeric_pipeline, list(numeric_columns_final)),
               ('binary', binary_pipeline, binary_columns),
               ('log', log_pipeline, skewed_columns)
               ]
```

In [19]:
```
column_transformer = ColumnTransformer(transformers=transformers, remainder='passthrough')
```

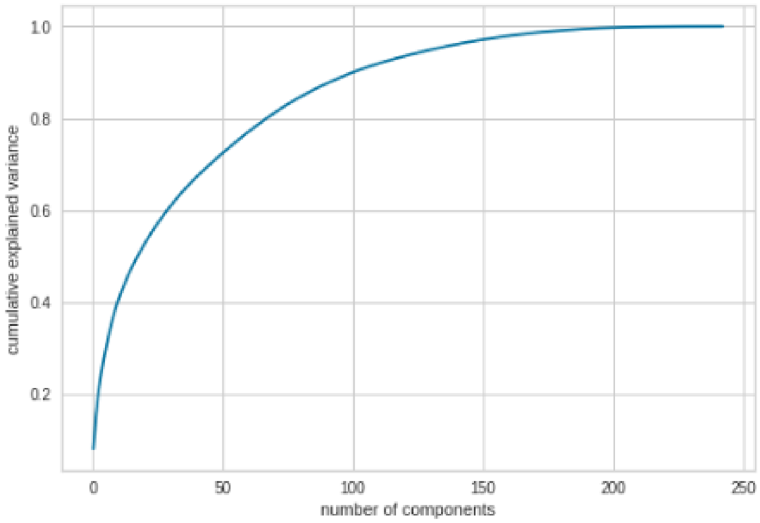
In [18]:
```

column_transformer.fit(azdias_df)
```

Out[18]:
```
ColumnTransformer(remainder='passthrough',
                  transformers=[('numeric',
                                 Pipeline(steps=[('imputer',
                                                  SimpleImputer(strategy='median')),
                                                 ('scaler', StandardScaler())]),
                                 ['KBA13_HERST_ASIEN', 'LP_STATUS_FEIN',
                                  'WOHNLAGE', 'PLZ8_ANTG1', 'KBA13_KW_110',
                                  'FINANZ_ANLEGER', 'ORTSGR_KLS9',
                                  'KBA13_FAB_SONSTIGE', 'KBA13_KW_60',
                                  'KBA13_BJ_2009', 'INNENSTADT', 'KBA13_KW_120',
```

I use also PCA; We have 97% of explicability with 160 features

```
Out[29]: <Figure size 720x1080 with 0 Axes>
```




```
<Figure size 720x1080 with 0 Axes>
```

```
In [30]: # The first 5 components explain 27.43% of variance
pca.explained_variance_ratio_[:5].sum()
```

```
Out[30]: 0.27431025412905397
```

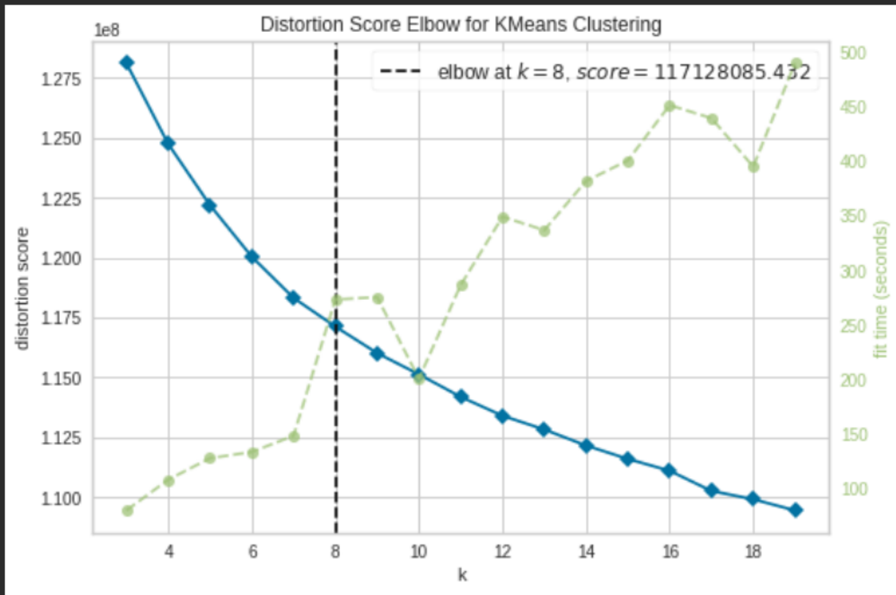
```
In [31]: for i in np.arange(10, 190, 10):
    print('{} components explain {}% of variance.'.format(i, pca.explained_variance_ratio_[:i].sum().
```

```
10 components explain 39.4% of variance.
20 components explain 52.0% of variance.
30 components explain 60.4% of variance.
40 components explain 66.8% of variance.
50 components explain 72.1% of variance.
60 components explain 76.7% of variance.
70 components explain 80.9% of variance.
80 components explain 84.39999999999999% of variance.
90 components explain 87.4% of variance.
100 components explain 89.8% of variance.
110 components explain 91.8% of variance.
120 components explain 93.4% of variance.
130 components explain 94.8% of variance.
140 components explain 96.0% of variance.
150 components explain 97.0% of variance.
160 components explain 97.89999999999999% of variance.
170 components explain 98.5% of variance.
180 components explain 99.0% of variance.
```

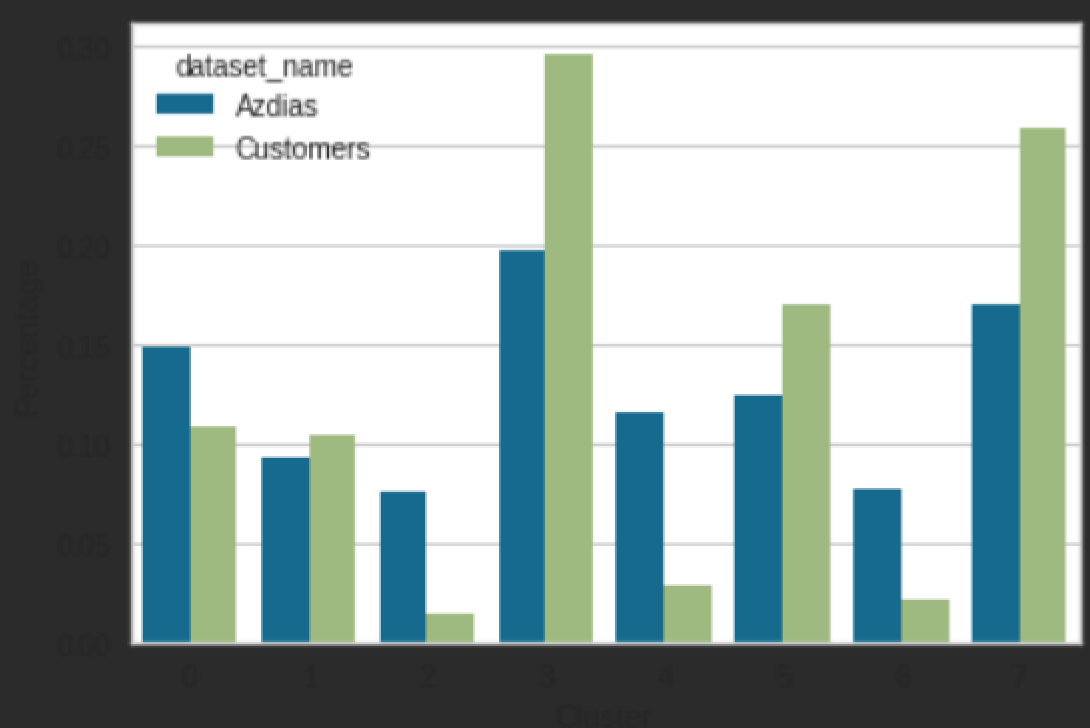
And we can understand better the data here Jupyter Notebook alt text

And Knn for feature selection:

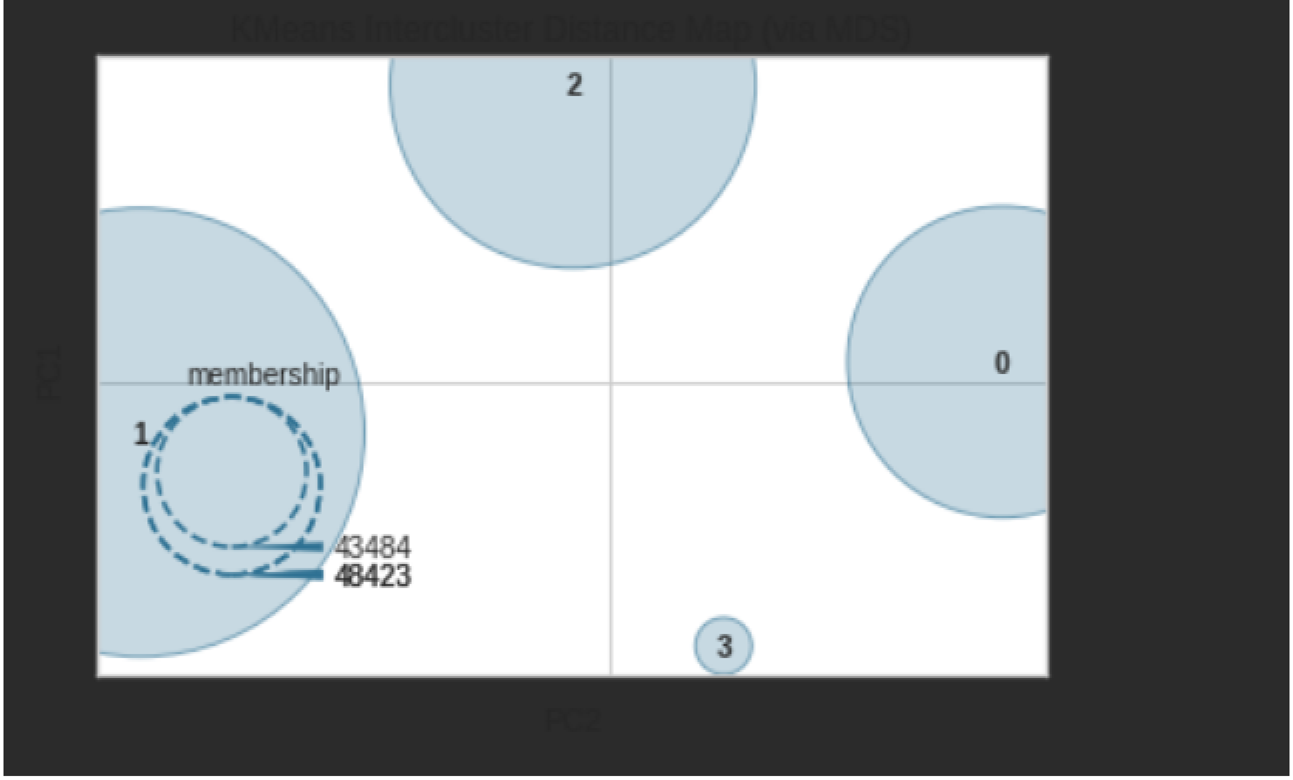
```
FutureWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25).
FutureWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25).
FutureWarning: 'n_jobs' was deprecated in version 0.23 and will be removed in 1.0 (renaming of 0.25).
```




```
8 <AxesSubplot:title={'center':'Distortion Score Elbow for KMeans Clustering'}, xlabel='k', ylabel='distortion score'>
```



But with 4 cluster we have a very good distance between the centroids.



For Supervised learning i used catboost and adaboost, and Logistic Regression with GridSearchCV.

BenchMark Model

Kaggle learboard was my benchmark, i dont have so many time to try, but i did 18 attempsts and the best result was using D19_SOZIALES on features

Methodology

DataPreprocessing

I created a dataframe with all null vales from info and attr dataframe, we 0,-1,X,XX and sometime 9 or 10 in some columns.

I created and applied 5 different pipelines to pre- processing data: 1- Data Wrangler Pipeline -> data_wrangler.py. 2- Feature Engineer Pipeline -> feature_engineer.py. 3- Preparing for unsupervised -> models/unsupervised_transform.joblib. 4- Unsupervised Learning pipe -> models/unsupervised_transform.joblib. 5- Supervised Learning -> train.py.

Implementation

After create pipelines i start to train using this Jupyter Notebook I think i documented everything very well

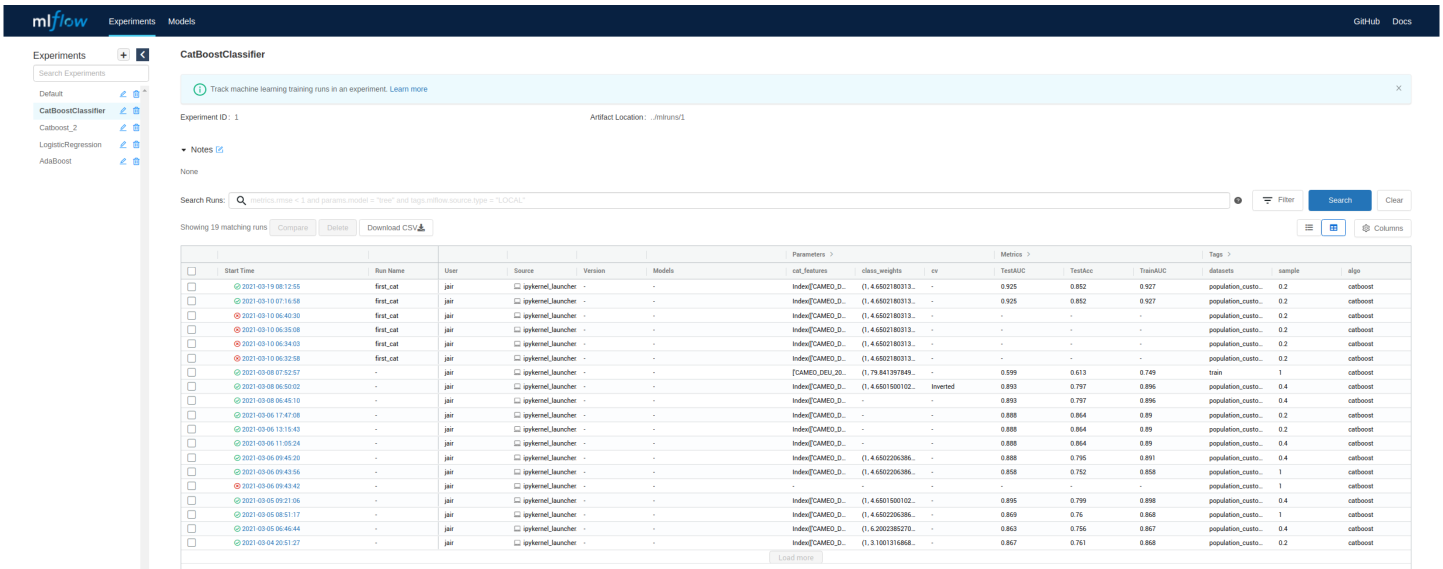
Refinement

I trid to use Hyperopt for bayesian hyperparameters tuning, but i did't had time to finish it with mlfow, so i used GridSearchCV with RepeatedStratifiedKFold, to find the best algo. Also i did some test with the most importante feature D19_SOZIALES, but i did't had time to make reports with that. Was hard to work with this amount of data, my computer crashed a lot of times.

Results

Model Evaluation and Validation

I used Mflow to track the improvements, it worked very well, and could document the features n of columns and dataset used.



Justification

My final solution is still catboost with all features. the pipeline transformation show me a lot of data interpretation, but I couldn't find anything to beate Catboost

Conclusion

It was a little disappointing to make so many transformations, understand the data and have such a bad result even with Gridserchcv, 50%, nothing better than random walk, while catboost proved to be very efficient, doing a wonderful job.

Refection

I think in the real life I can do a better work with the features if I have more time using cluster with feature importance, and using bayesian method to improve and find better hyper parameters. Was a great project, kind of hard because I had this dirty data, and need a good computer power to process everything well. But 79.9% is not so bad, isn't?