Machine Learning Engineer Nanodegree Project Report

Sergio Calderón Perez-Lozao - April 19th, 2020

Contents

Definition **2**
Project Overview 2
Problem Statement 2
Metrics 3
Analysis **3**
Data Exploration 3
Exploratory Visualization 4
Algorithms and Techniques 5
Benchmark 8
Methodology **8**
Data Preprocessing 8
Implementation 8
Constants 8
Experiment Tracking 9
Helpers 9
Models 9
Refinement 9
Customer and German population data 9
Training data provided 9
Mixing the two types of data 9
Results **10**
Model Evaluation and Validation 10
Justification 11

Definition

Project Overview

The German company Arvato Bertelsmann handles a lot of data and challenges us to use a small subset of it, which is public **demographic data to segment their customers**.

In addition, the company wants to know if, once this data has been analyzed and segmented, **it could be extrapolated to an application for the acquisition of new customers**.

Arvato also has the same information about these non-customers, since it is in the public domain. Therefore, it wants to check whether, with this data, it can be correctly classified as likely to become a new customer through different machine learning techniques, using supervised and unsupervised classification algorithms.

To carry out this task, the Udacity platform provides us with four data sets and two spreadsheets containing information on the attributes of the four other sets.

`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)

Problem Statement

To solve this problem we are going to divide it in two big blocks, **analysis** and **machine learning**.

In the analysis part, we will make an analysis of **which features are worthy of being kept and which are not**, due to their high cardinality, nullity or being identification columns, for example.

This analysis will be performed with the data sets `Udacity_AZDIAS_052018.csv` and `Udacity_CUSTOMERS_052018.csv`, which correspond to Arvato customers and a sample of the German population, as described above.

We will confront these two sets to a supervised machine learning model of binary classification in which the target is to which set it belongs, 1 if it is part of the clients and 0 if it is from the German population.

If this model has a poor AUC (0.5 would be the same as a random selection) it means that the problem is not separable and there is nothing that can be done with this data, but it results in a good AUC and even a good candidate for the Kaggle competition even if it is not trained with the same target.

If clients and non-clients can be well separated, it opens up three horizons for us to experiment:

- Using this data since the size is larger
- Use training data because they are the ones that provide us with the right target even if there is less sample

- Using both with the mixed target, we will have more sample and the effect of the training data provided will not be lost

These experiments will be carried out in the second part of the project, the one related to machine learning and Kaggle competition

Metrics

For **unbalanced problems like this it is more advisable to use the PR AUC**, but as the metric they evaluate in the Kaggle, and therefore our objective, is the AUC, we will use this metric to evaluate the models.

This metric allows us to know **how well we rank the data**, in this case leads, and therefore a high AUC is translated into the business problem to be solved, **knowing who to direct the mailout campaigns to**.

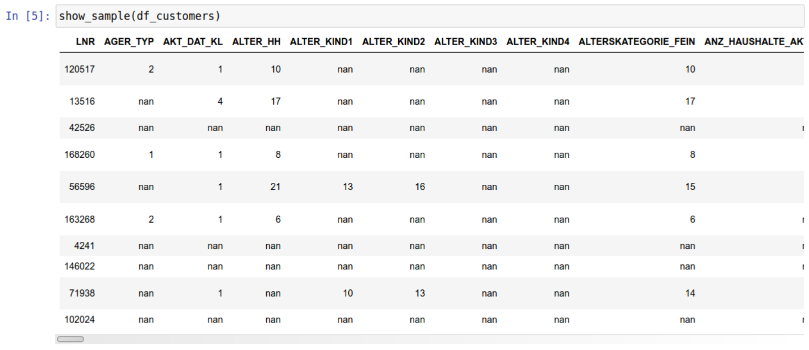
Analysis

Data Exploration

First of all, we read a representative sample of the data, otherwise it is a pretty big data set and it will make us iterate slower.

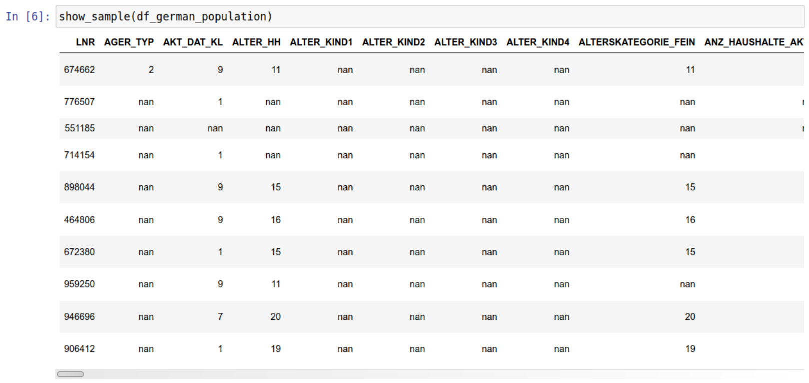
> If there is no signal in the representative sample of the data, there will be no signal in the total data set.

To see how the data is, we draw a sample of this sample and see the huge amount of nulls that both datasets have:



```
In [5]: show_sample(df_customers)
```

LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AK
120517	2	1	10	nan	nan	nan	nan	10	
13516	nan	4	17	nan	nan	nan	nan	17	
42526	nan	nan	nan	nan	nan	nan	nan	nan	
168260	1	1	8	nan	nan	nan	nan	8	
56596	nan	1	21	13	16	nan	nan	15	
163268	2	1	6	nan	nan	nan	nan	6	
4241	nan	nan	nan	nan	nan	nan	nan	nan	
146022	nan	nan	nan	nan	nan	nan	nan	nan	
71938	nan	1	nan	10	13	nan	nan	14	
102024	nan	nan	nan	nan	nan	nan	nan	nan	



```
In [6]: show_sample(df_german_population)
```

LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AK
674662	2	9	11	nan	nan	nan	nan	11	
776507	nan	1	nan	nan	nan	nan	nan	nan	
551185	nan	nan	nan	nan	nan	nan	nan	nan	
714154	nan	1	nan	nan	nan	nan	nan	nan	
898044	nan	9	15	nan	nan	nan	nan	15	
464806	nan	9	16	nan	nan	nan	nan	16	
672380	nan	1	15	nan	nan	nan	nan	15	
959250	nan	9	11	nan	nan	nan	nan	nan	
946696	nan	7	20	nan	nan	nan	nan	20	
906412	nan	1	19	nan	nan	nan	nan	19	

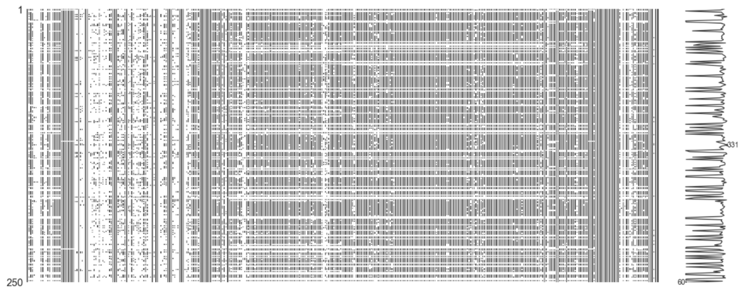
As we know from the spreadsheets provided, most of the columns are categorical although most have a numerical code.

The different statistics do not make much sense because of the huge number of columns. Comparing each level of every categorical feature and checking that they have similar volume distributions is a tedious job and difficult to simplify. When doing this type of analysis on a dataset like this one it is more useful to implement a model and draw conclusions from its important variables and biases.

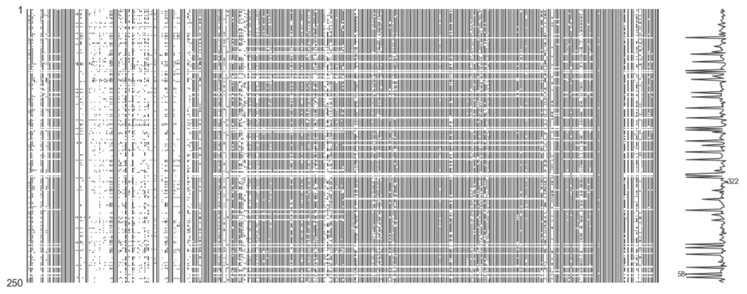
Exploratory Visualization

As we can see there are a lot of nulls at first sight, let's use the missingno library, very useful for this kind of tasks.

```
In [16]: msno.matrix(df_customers.sample(250))
Out[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2e35ae3128>
```

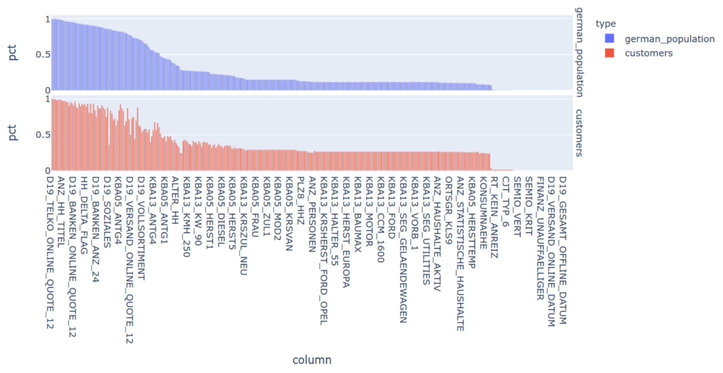


```
In [17]: msno.matrix(df_german_population.sample(250))
Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f2e34e17908>
```



Both follow similar patterns and we can also confirm that the datasets have many lost values. In the graphs above, black means that the value exists and white is a missing value. It is calculated on a sample of 250 rows of the total (already sampled, as we said before). To consider null values, some columns with values -1 and 0 are also included, as suggested in the spreadsheet explaining the variables (some of them).

For a better understanding, it is more useful to have a graph that simplifies the matrices above with the percentage of nulls that each column has, for each of the data sets. This is what is shown in the following graph, built with the plotly_express library.

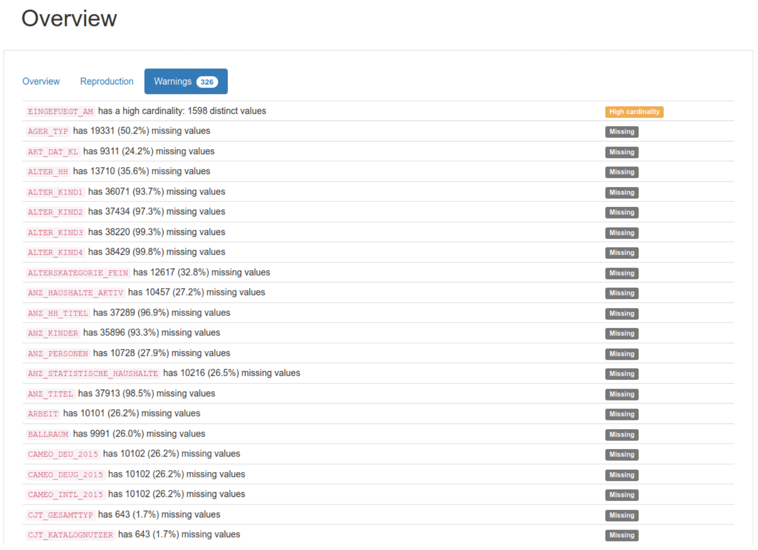


It's ordered from the column with the most nulls in the customers to the one with the least. We can see that they are distributed practically the same, but that there are columns that have a different percentage of nulls, perhaps this can be a determining factor when segmenting.

As we already know from the spreadsheets provided, most of the columns are categorical although most have a numerical code.

The different statistics do not make much sense because of the huge number of columns. Comparing each level of every categorical feature and checking that they have similar volume distributions is a tedious job and difficult to simplify. When doing this type of analysis on a dataset like this one it is more useful to implement a model and draw conclusions from its important variables and biases.

Even so we have the profiling part where we have all these insights for each of the datasets and we can see which variables are more similar and which less.


Overview

Overview | Reproduction | Warnings 326

`EINGEFUEGT_AM` has a high cardinality: 1598 distinct values	High cardinality
`AGER_TYP` has 19331 (50.2%) missing values	Missing
`AKT_DAT_KL` has 9311 (24.2%) missing values	Missing
`ALTER_HH` has 13710 (35.6%) missing values	Missing
`ALTER_KIND1` has 36071 (93.7%) missing values	Missing
`ALTER_KIND2` has 37434 (97.3%) missing values	Missing
`ALTER_KIND3` has 38220 (99.3%) missing values	Missing
`ALTER_KIND4` has 38429 (99.8%) missing values	Missing
`ALTERSKATEGORIE_FEIN` has 12617 (32.8%) missing values	Missing
`ANZ_HAUSHALTE_AKTIV` has 10457 (27.2%) missing values	Missing
`ANZ_HH_TITEL` has 37289 (96.9%) missing values	Missing
`ANZ_KINDER` has 35896 (93.3%) missing values	Missing
`ANZ_PERSONEN` has 10728 (27.9%) missing values	Missing
`ANZ_STATISTISCHE_HAUSHALTE` has 10216 (26.5%) missing values	Missing
`ANZ_TITEL` has 37913 (98.5%) missing values	Missing
`ARBEIT` has 10101 (26.2%) missing values	Missing
`BALLRAUM` has 9991 (26.0%) missing values	Missing
`CAMEO_DEU_2015` has 10102 (26.2%) missing values	Missing
`CAMEO_DEUG_2015` has 10102 (26.2%) missing values	Missing
`CAMEO_INTL_2015` has 10102 (26.2%) missing values	Missing
`CJT_GESAMTTYP` has 643 (1.7%) missing values	Missing
`CJT_KATALOGNUTZER` has 643 (1.7%) missing values	Missing

Algorithms and Techniques

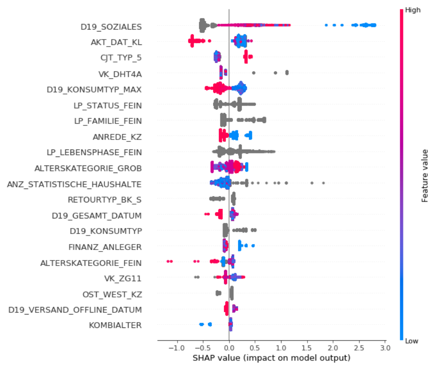
Regarding the algorithms, we will use two of the state-of-the-art algorithms for tabular data, XGBoost and CatBoost, combined with the sklearn pipeline that helps to embed the modeling and pre-processing of the data.

We could also try GBM of H2O and LightGBM, two very powerful gradient boosting too, plus the one of **H2O has the MOJO to make the model easily deployable**, but they have not been tested.

In addition to these algorithms, which will do the final task of assigning a probability of being prone to convert a client to each row, we will use unsupervised learning algorithms like KMeans and different tools and techniques like shap or eli5 to be able to interpret these black box models, very accurate but little explainable by themselves.

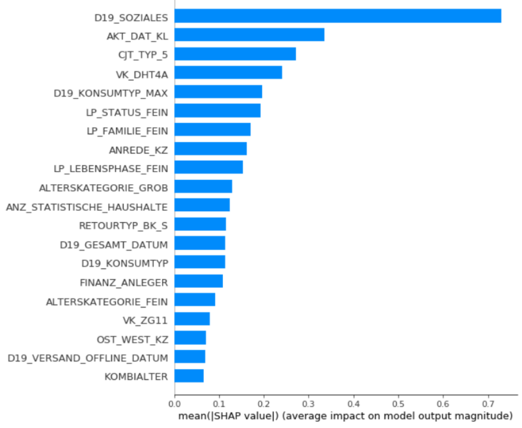
Here we can see the variables on which the first model we trained to make a prediction is most focused (trained with customer and German population data). The more red is higher the value of the variable and the more blue the opposite, the grey on the other hand are missing values or it is always present when dealing with categorical features since in this type of variables there is no order. The horizontal axis is the contribution of that variable to the model.

```
In [23]: explainer = shap.TreeExplainer(catboost)
         shap_values = explainer.shap_values(X_test)

         shap.summary_plot(shap_values, X_test)

         Setting feature_perturbation = "tree_path_dependent" because no background data was given.
```



More simplified but without taking into account the sign of the contribution, we can see this other graphic, also from the brilliant shap library.

```
In [24]: shap.summary_plot(shap_values, X_test, plot_type='bar')
```



The downside is that we don't have the meaning of none of the top three variables (`D19_SOZIALES`, `AKT_DAT_KL` and `CJT_TYP_5`) available. This would be unacceptable in a company, since data scientist must understand the business and its predictive models, but for this exercise we can turn a blind eye.

Now let's see how unsupervised learning techniques such as PCA + KMeans might help to better segment the data set.

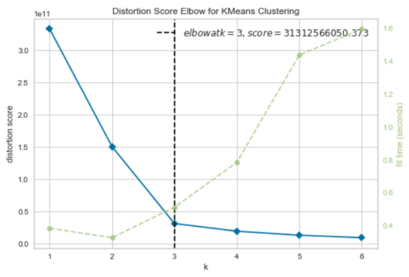
```
In [37]: pipeline_preprocess_cluster = make_pipeline(PandasTypeSelector(include='number'),
                                                     SimpleImputer(strategy='constant', fill_value=-99),
                                                     PCA(n_components=4, random_state=RANDOM_STATE))

         pipeline_preprocess_cluster.fit(X_train, y_train)

         X_train_transformed = pipeline_preprocess_cluster.transform(X_train)

         print(f'Explained variance ratio {pipeline_preprocess_cluster.named_steps["pca"].explained_variance_ratio_}')
         print(f'Singular values {pipeline_preprocess_cluster.named_steps["pca"].singular_values_}')

         Explained variance ratio [0.62628903 0.27869381 0.04550947 0.01937783]
         Singular values [463465.42704142 309167.33415982 124934.05735218  81523.44496766]
```

With only **four main components** we explain **95% of the variance** and on this **reduction of dimensionality** we will be able to apply a KMeans. Using the elbow method we choose an optimal 'k'.

KMeans Optimum `k`

We will choose `k` value with famous elbow method

```
In [38]: model = KMeans()
         visualizer = KElbowVisualizer(model, k=(1,7))

         visualizer.fit(X_train_transformed)
         visualizer.show()
```



```
Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3edc4d70f0>
```

As we can see, the groups are well differentiated and there is no overlap, although the size of the groups is not very similar. As we can see below, the groups do not discriminate differently between the test and train data, since the LIFT are the same in practically all the groups

```
In [39]: pipeline_kmeans = make_pipeline(pipeline_preprocess_cluster,
                                         KMeans(n_clusters=3, random_state=RANDOM_STATE))

         pipeline_kmeans.fit(X_train)

         # dtypes needed for applying `pipeline_kmeans`
         serialize_object_dump(X_train.dtypes, 'X_train_dtypes.pkl')

         df_cluster_train = pd.DataFrame(dict(dataset='train',
                                              cluster=pipeline_kmeans.predict(X_train),
                                              target=y_train,
                                              target_mean=y_train.mean()))

         df_cluster_test = pd.DataFrame(dict(dataset='test',
                                             cluster=pipeline_kmeans.predict(X_test),
                                             target=y_test,
                                             target_mean=y_test.mean()))

         df_cluster = pd.concat([df_cluster_train, df_cluster_test])

         df_cluster_grouped = (df_cluster
                               .groupby(['dataset', 'cluster'])
                               .agg(dict(target=['mean', 'count'], target_mean=['first']))
                               .reset_index())

         df_cluster_grouped.columns = ['dataset', 'cluster', 'target_mean_group', 'volume', 'target_mean']

         df_cluster_grouped['lift'] = df_cluster_grouped['target_mean_group'] / df_cluster_grouped['target_mean']

         df_cluster_grouped
```

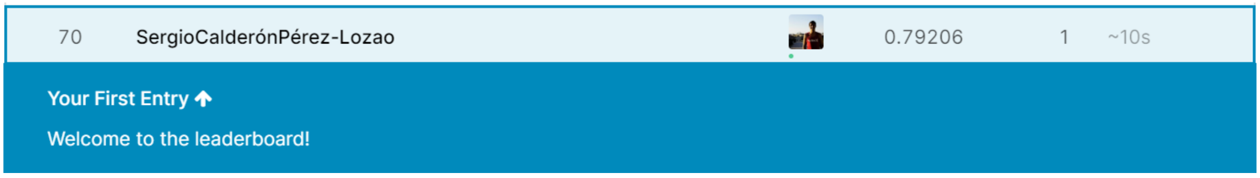
Out[39]:

	dataset	cluster	target_mean_group	volume	target_mean	lift
0	test	0	0.165744	18058	0.178298	0.929590
1	test	1	0.130333	10665	0.178298	0.730985
2	test	2	0.374535	3762	0.178298	2.100615
3	train	0	0.166311	83777	0.178229	0.933128
4	train	1	0.128717	49698	0.178229	0.722202
5	train	2	0.369109	18122	0.178229	2.070982

Finally , the clustering did not help and the first model that differentiates between customers and the German population with all available variables works relatively well. CatBoost does an auto-imatic treatment of categorical variables and is very efficient.

These were the parameters of the first algorithm used and the score of its submission to Kaggle.

```python
from catboost import CatBoostClassifier


# Some preprocessing code
params_catboost = dict(num_trees=30,
                       max_depth=3,
                       min_child_samples=25,
                       one_hot_max_size=4,
                       class_weights=class_weights,
                       cat_features=cat_features,
                       random_state=RANDOM_STATE,
                       eval_metric='AUC')

CatBoostClassifier(**params_catboost)
```



70 SergioCalderónPérez-Lozao 0.79206 1 ~10s

Your First Entry

Welcome to the leaderboard!

Benchmark

The reference point to beat is in the kaggle leaderboard and we can place it around the 0.78 that we have already achieved with the first submission.

The problem here is that the **leaderboard only corresponds to 30% of the test data**, so we may not really do as well (or as badly) as we thought from this ranking.

We have to work on implementing an algorithm that scores well on Kaggle but doesn't overfit in cross validation. In short, a more robust algorithm.

Methodology

Data Preprocessing

For the preprocessing of the data, the information provided in the spreadsheets has been taken into account, so that all these values are considered as null values: '0, -1, 'X', 'XX']`.

In addition, it has been implemented in such a way that pandas reads the data with its `category` type that makes the use of categorical variables much more efficient in memory than the typical `object` type used by default.

The nulls have been filled in for the steps that required it, adding the category `UNKNOWN` to the columns with nulls, in case they were categorical variables. For the numeric columns, no null treatment has been done since both XGBoost and CatBoost work well with nulls and are able to see patterns in them (if any).

In some of the pipelines that have been implemented, a step has been used to make an oversampling of the data to the positive class, since in the train daots only 1% of the data has the target with this value.

Implementation

For the implementation of the different pipelines and their comparison the python module `customer_segmentation` has been developed (link to documentation).

In this module we have four sub-modules:

- `constants.py`
- `experiment_tracking.py`
- `helpers.py`
- `models.py`

Constants

In this sub module we have all the project constants centralized, like for example `RANDOM_STATE` for the code reproducibility or also different `PATH_*` for the route management

Experiment Tracking

In this other sub-module we have the management of the experiments, performances and models orchestrated MLFlow.

Helpers

Here we have all the helpers of the project, such as `read_demographic_data` or `kaggle_submission` that facilitate tasks that are repeated over and over again and are general

Models

Finally, in this sub-module there are all the functions referred to pre-processing of data for the models, calculation of metrics and persistence of models

When applying this module, the two notebooks `Exploratory_Data_Analysis.ipynb` and `Machine_Learning_Algorithms_For_Kaggle.ipynb` are used, clearly separated.

Refinement

For the refinement and the different iterations between the models used, a tracking of the different experiments has been implemented with MLFlow:

- Using data from customers and the German population
- Using training data provided
- Using a mixture of the two approaches above

Customer and German population data

For this first section, the models trained in the first part of the project, corresponding to the first notebook, are used.

This experiment is made up of three models, one with all the variables, another with some less and finally, the one that uses the clustering explained above.

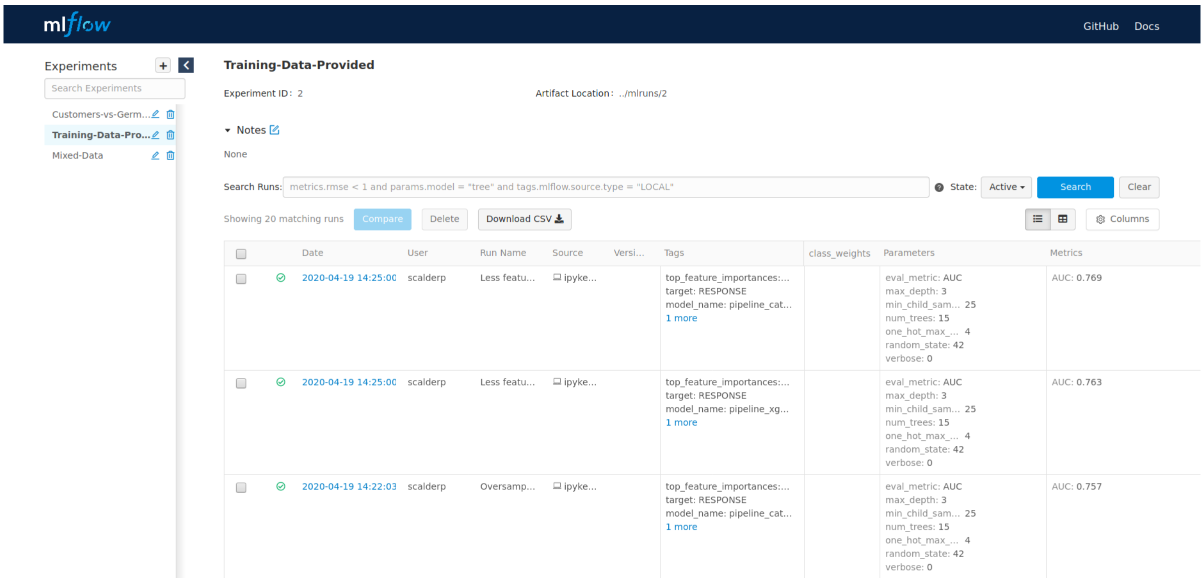
Training data provided

Here the sample is smaller and we will iterate with different pipelines ending in XGBoost and CatBoost with oversampling and testing less variables, taking into account the importance of the variables of the models trained before.

Mixing the two types of data

Finally, different modes will be trained, with approaches similar to the previous point but without oversampling, with all data together, to see if this can help improve the robustness and accuracy of the model.

Using MLFlow we can persist all these models and their metrics, parameters, etc and save the best one for direct reproducibility in a MLmodel file.



Experiments

Search Experiments

Customers-vs-Germ...

Training-Data-Pro...

Mixed-Data

Training-Data-Provided

Experiment ID: 2 Artifact Location: ../mlruns/2

Notes

None

Search Runs: metrics.rmse < 1 and params.model = "tree" and tags.mlflow.source.type = "LOCAL"

State: Active

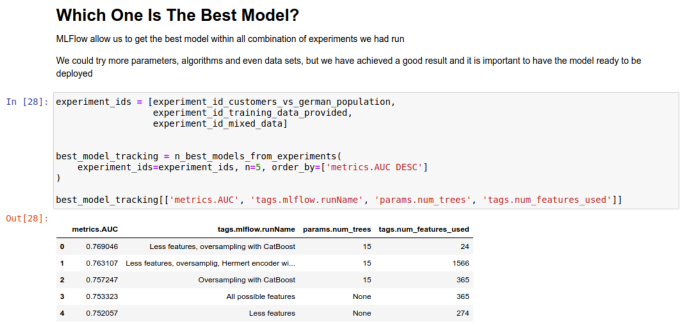
Showing 20 matching runs

Date	User	Run Name	Source	Versi...	Tags	class_weights	Parameters	Metrics
2020-04-19 14:25:00	scalderp	Less featu...	ipyke...		top_feature_importances:... target: RESPONSE model_name: pipeline_cat... 1 more		eval_metric: AUC max_depth: 3 min_child_sam... 25 num_trees: 15 one_hot_max_... 4 random_state: 42 verbose: 0	AUC: 0.769
2020-04-19 14:25:00	scalderp	Less featu...	ipyke...		top_feature_importances:... target: RESPONSE model_name: pipeline_xg... 1 more		eval_metric: AUC max_depth: 3 min_child_sam... 25 num_trees: 15 one_hot_max_... 4 random_state: 42 verbose: 0	AUC: 0.763
2020-04-19 14:22:03	scalderp	Oversamp...	ipyke...		top_feature_importances:... target: RESPONSE model_name: pipeline_cat... 1 more		eval_metric: AUC max_depth: 3 min_child_sam... 25 num_trees: 15 one_hot_max_... 4 random_state: 42 verbose: 0	AUC: 0.757

Results

Model Evaluation and Validation

The model that turns out to be the best according to the auc in test is an oversampling pipeline at twice the positive class, since if we balanced more it would perform much worse, and a CatBoostClassifier:



Which One Is The Best Model?

MLFlow allow us to get the best model within all combination of experiments we had run

We could try more parameters, algorithms and even data sets, but we have achieved a good result and it is important to have the model ready to be deployed

In [28]:
```
experiment_ids = [experiment_id_customers_vs_german_population,
                  experiment_id_training_data_provided,
                  experiment_id_mixed_data]


best_model_tracking = n_best_models_from_experiments(
    experiment_ids=experiment_ids, n=5, order_by=['metrics.AUC DESC']
)

best_model_tracking[['metrics.AUC', 'tags.mlflow.runName', 'params.num_trees', 'tags.num_features_used']]
```

Out[28]:

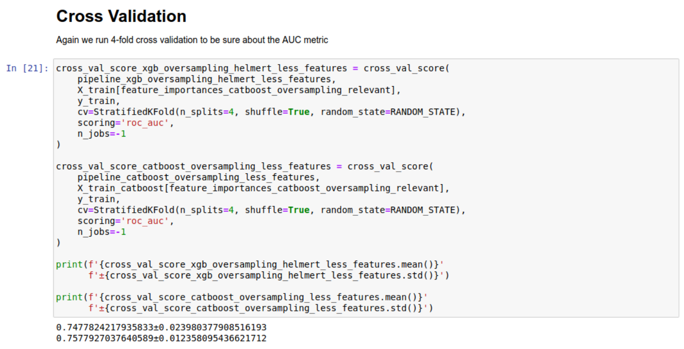
	metrics.AUC	tags.mlflow.runName	params.num_trees	tags.num_features_used
0	0.769046	Less features, oversampling with CatBoost	15	24
1	0.763107	Less features, oversamplig, Hermert encoder wi...	15	1566
2	0.757247	Oversampling with CatBoost	15	365
3	0.753323	All possible features	None	365
4	0.752057	Less features	None	274

Best model parameters:

```python
from imblearn.pipeline import make_pipeline
from catboost import CatBoostClassifier
from imblearn.over_sampling import RandomOverSampler


# Some preprocessing code
params_catboost = dict(
    num_trees=30,
    max_depth=3,
    min_child_samples=25,
    one_hot_max_size=4,
    cat_features=(
        set(cat_features)
        .intersection(feature_importances_catboost_oversampling_relevant)
    ),
    random_state=RANDOM_STATE,
    eval_metric='AUC',
    verbose=0
)

pipeline_catboost_oversampling_less_features = make_pipeline(
    RandomOverSampler(sampling_strategy=.02, random_state=RANDOM_STATE),
    CatBoostClassifier(**params_catboost)
)
```

In order to validate it, a cross validation is made and the average of the preditions in the folds and their standard deviation is calculated. It seems to be a very robust model, so it ends up being the one chosen for the last submission and we reult in a good classification within Kaggle, only one point away from the best student.

```
# Dependencies code and preprocessing
cross_val_score_catboost_oversampling_less_features = cross_val_score(
    pipeline_catboost_oversampling_less_features,
    X_train_catboost[feature_importances_catboost_oversampling_relevant],
    y_train,
    cv=StratifiedKFold(n_splits=4, shuffle=True, random_state=RANDOM_STATE),
    scoring='roc_auc',
    n_jobs=-1
)

print(f'{cross_val_score_catboost_oversampling_less_features.mean()}'
      f'±{cross_val_score_catboost_oversampling_less_features.std()}')
```

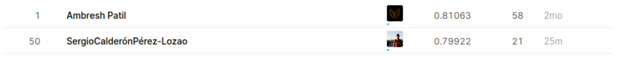

Cross Validation

Again we run 4-fold cross validation to be sure about the AUC metric

```
In [21]: cross_val_score_xgb_oversampling_helmert_less_features = cross_val_score(
             pipeline_xgb_oversampling_helmert_less_features,
             X_train[feature_importances_catboost_oversampling_relevant],
             y_train,
             cv=StratifiedKFold(n_splits=4, shuffle=True, random_state=RANDOM_STATE),
             scoring='roc_auc',
             n_jobs=-1
         )

         cross_val_score_catboost_oversampling_less_features = cross_val_score(
             pipeline_catboost_oversampling_less_features,
             X_train_catboost[feature_importances_catboost_oversampling_relevant],
             y_train,
             cv=StratifiedKFold(n_splits=4, shuffle=True, random_state=RANDOM_STATE),
             scoring='roc_auc',
             n_jobs=-1
         )

         print(f'{cross_val_score_xgb_oversampling_helmert_less_features.mean()}'
               f'±{cross_val_score_xgb_oversampling_helmert_less_features.std()}')

         print(f'{cross_val_score_catboost_oversampling_less_features.mean()}'
               f'±{cross_val_score_catboost_oversampling_less_features.std()}')

         0.7477824217935833±0.023980377908516193
         0.7577927037640589±0.012358095436621712
```

Justification

The obtained results are good, as we are only one point (and a couple of decimals) away from the best of the public leaderboard.



1	Ambresh Patil		0.81063	58	2mo
50	SergioCalderónPérez-Lozao		0.79922	21	25m

This seems to be a good enough result, besides that a very complete work has been done with different technologies and the objective of the project is fulfilled.

As future work there are several points that could be explored, like the algorithms mentioned above or for example a training with mixed data but weighing more strongly the records that belong to the training set provided.

Finally, a better understanding of the business could also be the key to improving the model obtained, but we have focused on being able to make all our work reproducible and easy to read for everyone else.

> Actionability is one of the most important parts of a model, as well as its interpretability