Capstone Project
Machine Learning Engineer Nanodegree

Zhihua Zhang

August 23, 2020

Contents

1 Definition

1.1 Project Overview

Understanding existing customers and recognizing new customers are crucial for the development of any company. Bertelsmann Arvato Analytics, a Fortune 500 financial services company, is trying to identify its potential core customer base from the entire German population, which brings out this project that we worked on.

Our dataset is provided by Arvato and a basic summary is as follows:

1. Demographics data for the entire population in Germany: 891,211 people;
2. Demographics data and categories for Arvato's customers: 191,652 people;
3. Demographics data and responses for targets in marketing campaign: 42,982 people;
4. Demographics data for targets in marketing campaign: 42,833 people(Test data);

More specifically, the dataset has 366 features including personal information such as age and information outside individuals like household. They are mostly categorical data, but continuous data also exists such as personal income. Besides, unknown data is marked as -1 and missing values are just left there.

1.2 Problem Statement

Given data described above, we need to achieve two main tasks:

1. Identify potential customer base from the whole German population.
2. Apply what we learned and handy data to learn a model and predict how likely each target in Arvato's marketing campaign will convert to become a new customer.

The ultimate goal is identify potential customers as more and accurate as we can. To solve the problem, we prepare four different methods to handle class imbalance as well as four powerful models to make predictions.

1.3 Metric

One notable challenge of our task is class imbalance, we show that in figure 1

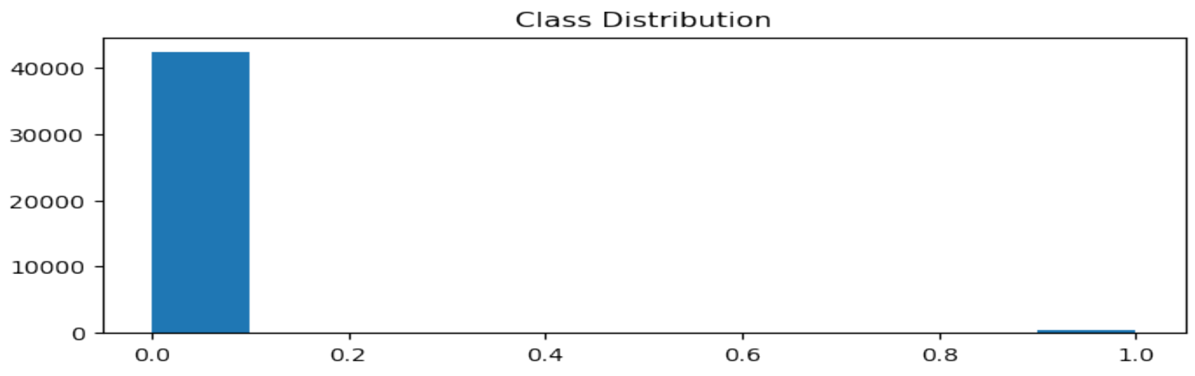


Figure 1: Class Distribution

We can see that there are only about 1% positive labels, which makes accuracy no longer impartial and alternative evaluation metrics need to be set.

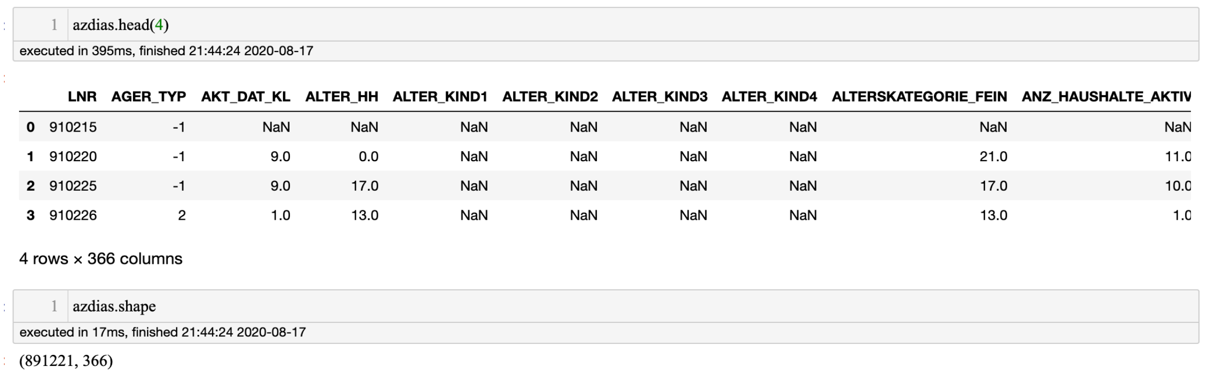
Recall and precision are good candidates as they measure model's ability to recognize positive instances. Combining them together gives us the F1-score that is the harmonic average of recall and precision and thus more all-around.

Besides, capability of distinguishing different classes or way say making ordered prediction is also valuable. This is because by adjusting threshold, we can still generate reasonable predictions. This leads us to take AUC as another proper metric.

2 Analysis

2.1 Data Exploration and Visualization

We first take a glance at our dataset.



```
azdias.head(4)
executed in 395ms, finished 21:44:24 2020-08-17
```

	LNR	AGER_TYP	AKT_DAT_KL	ALTER_HH	ALTER_KIND1	ALTER_KIND2	ALTER_KIND3	ALTER_KIND4	ALTERSKATEGORIE_FEIN	ANZ_HAUSHALTE_AKTIV
0	910215	-1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	910220	-1	9.0	0.0	NaN	NaN	NaN	NaN	21.0	11.0
2	910225	-1	9.0	17.0	NaN	NaN	NaN	NaN	17.0	10.0
3	910226	2	1.0	13.0	NaN	NaN	NaN	NaN	13.0	1.0

4 rows × 366 columns

```
azdias.shape
executed in 17ms, finished 21:44:24 2020-08-17
(891221, 366)
```

Figure 2: Data Exploration

From figure 2 we know that our demographic dataset includes lots of missing values and features seem to be mostly categorical. Thus, we visualize their distributions to have a better understanding.

```
_ = azdias.hist(column=azdias.columns[1::int(azdias.shape[1]//8)], figsize=(16, 7), bins=50)
executed in 2.50s, finished 22:20:49 2020-08-17
```

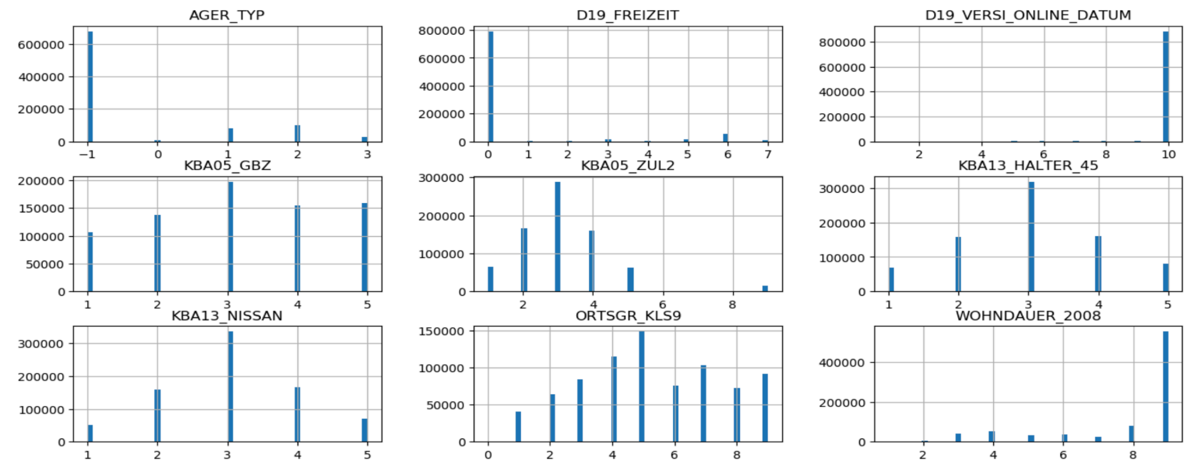
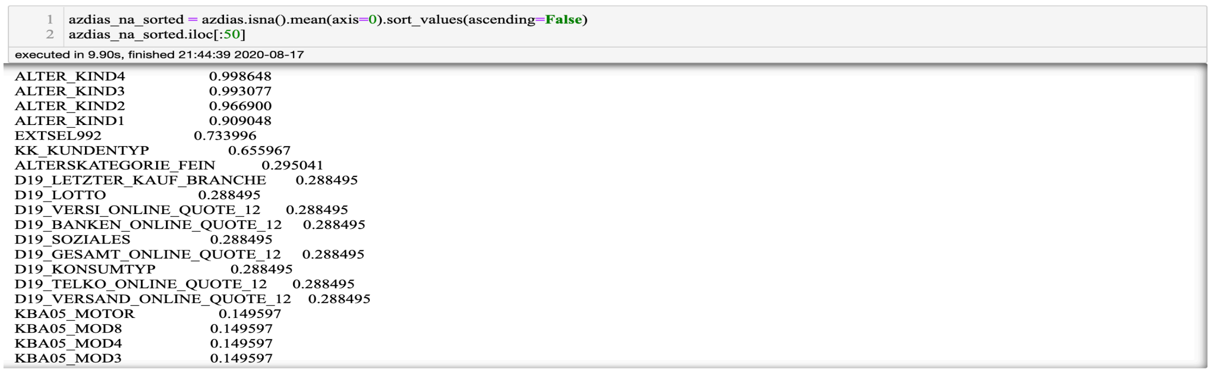


Figure 3: Feature Distribution

Figure 3 shows distribution of eight sequentially selected features from the first to the last. We can observe that they are indeed categorical features. This also indicates a possible direction of data cleaning, namely whether some categorical features are stored in their original formats such as string that should be converted into numerical data.

Now that we have a basic understanding of handy dataset, we check abnormalities that need to be addressed with.

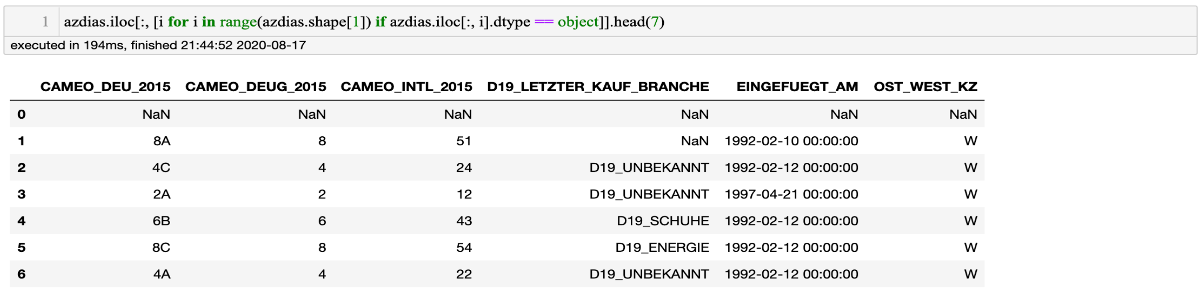


```
azdias_na_sorted = azdias.isna().mean(axis=0).sort_values(ascending=False)
azdias_na_sorted.iloc[:50]
```

executed in 9.90s, finished 21:44:39 2020-08-17

```
ALTER_KIND4                  0.998648
ALTER_KIND3                  0.993077
ALTER_KIND2                  0.966900
ALTER_KIND1                  0.909048
EXTSEL992                    0.733996
KK_KUNDENTYP                 0.655967
ALTERSKATEGORIE_FEIN         0.295041
D19_LETZTER_KAUF_BRANCHE     0.288495
D19_LOTTO                    0.288495
D19_VERSI_ONLINE_QUOTE_12    0.288495
D19_BANKEN_ONLINE_QUOTE_12   0.288495
D19_SOZIALES                 0.288495
D19_GESAMT_ONLINE_QUOTE_12   0.288495
D19_KONSUMTYP                0.288495
D19_TELKO_ONLINE_QUOTE_12    0.288495
D19_VERSAND_ONLINE_QUOTE_12  0.288495
KBA05_MOTOR                  0.149597
KBA05_MOD8                   0.149597
KBA05_MOD4                   0.149597
KBA05_MOD3                   0.149597
```

Figure 4: Missing Values



```
azdias.iloc[:, [i for i in range(azdias.shape[1]) if azdias.iloc[:, i].dtype == object]].head(7)
```

executed in 194ms, finished 21:44:52 2020-08-17

	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
5	8C	8	54	D19_ENERGIE	1992-02-12 00:00:00	W
6	4A	4	22	D19_UNBEKANNT	1992-02-12 00:00:00	W

Figure 5: Wrong Data Format

Figure 4 and 5 show that missing values and improper data formats need to be handled. Finally, we also examine the differences in the range of features, for example, offspring and birth-year start differently in nature. Result is shown in figure 6.

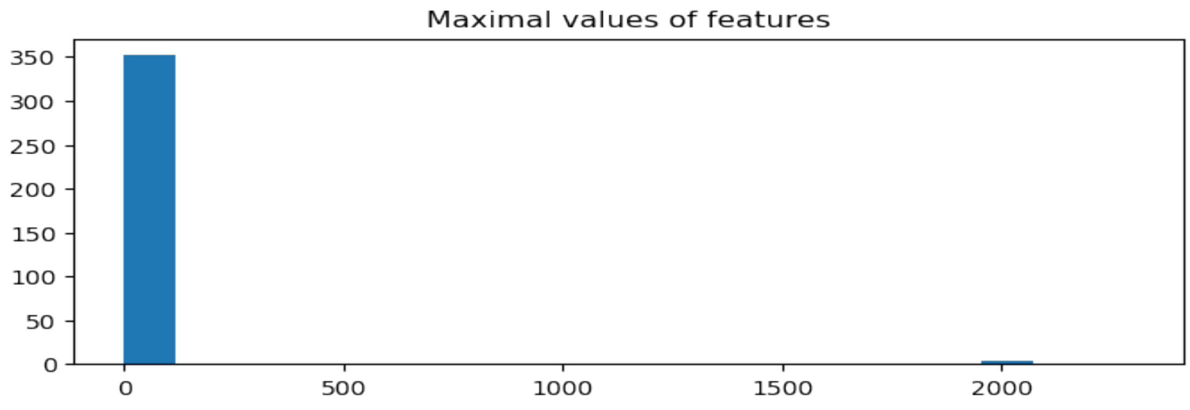


Figure 6: Maximal values of features

Above exploration informs us that we need to carefully deal with data before starting to train models. Details will be covered later in the Preprocessing section.

2.2 Algorithms and Techniques

2.2.1 Unsupervised Learning

For the first task, since we do not know people's responses, it is impossible to identify potential core customer base using supervised learning algorithms that require labels.

A good solution is to transform problem into anomaly detection that we treat prospective customers as unusual in the entire population. To do this, we apply K-means clustering technique on current customers and calculate the distance between each German individual and his or her closest customer cluster. Based on these distance estimations, we can tell how likely each individual is "anomaly", namely potential customer.

However, we have more than 360 features available, so it is very time-consuming to directly do clustering. To mitigate this pain, we do PCA first and select top PCs with high explanation capacity to be clustered.

2.2.2 Supervised Learning

Assuming we already have preprocessed data in hand, then to tackle this classification problem, we still need to complete three tasks:

1. Feature selection: With more that 350 features, it is wise to choose features that are useful over irrelevant ones for efficiency consideration. We do this via two steps.
   (1) Pearson correlation test: Features with high correlation would be redundant to keep. To avoid losing much useful information, we treat features with absolute correlation higher than 0.9 as duplicated and keep the one with largest variance.
   (2) Random Forest selection: Tree-based model is great to report features importance. We take advantage of this by training a random forest that select only one feature each time and keep ones with top 90% cumulative information gains.
2. Handling class imbalance: To deal with imbalanced dataset, typical methods include unbalanced assignment, over-sampling and under-sampling. Based on that, we made some improvements:
   (1) Under-sampling + Ensemble: We under-sample many different mini datasets to achieve class balance, train separate models on them, and average their predictions together to produce the final result.
   (2) Under-sampling + Over-sampling + Ensemble: We under-sample fewer mini datasets that are not balanced yet, then over-sample from minority to achieve balance, and finally average their outputs.
3. Model Selection and Development: The key step is to choose a powerful model to fully utilize our well-prepared datasets. Candidate models are Logistic Regression(benchmark), SVM, Random Forest and XGBoost. Final winner is the one that performs best with the simplest unbalanced weight assignment.

2.3 Benchmark

Although it is hard to find a publicly recognized benchmark because this project is a Kaggle competition in nature, we can simply regard Logistic Regression, a very naive classifier, as our benchmark.

In case our chosen model underperforms the benchmark, we then should thoroughly inspect the whole modeling process to figure bugs.

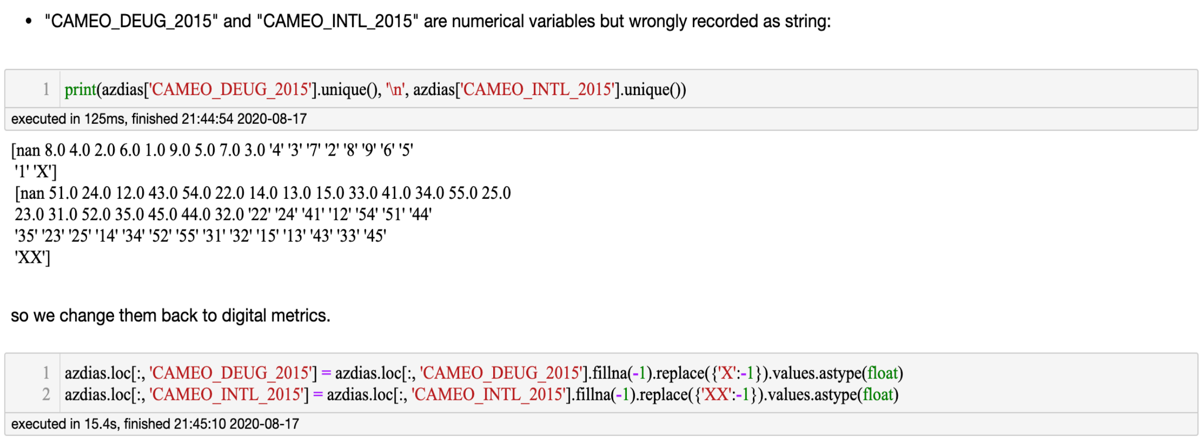
3 Methodology

3.1 Data Preprocessing

In Data Exploration section, we realize that abnormalities such as missing values and improper data formats need to be addressed with. To do this, we

1. Drop features with frequency of missing values higher than 0.6;

2. Correct features stored in wrong formats as shown in 7;



- "CAMEO_DEUG_2015" and "CAMEO_INTL_2015" are numerical variables but wrongly recorded as string:

```
print(azdias['CAMEO_DEUG_2015'].unique(), '\n', azdias['CAMEO_INTL_2015'].unique())
```

executed in 125ms, finished 21:44:54 2020-08-17

```
[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']
 [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']
```

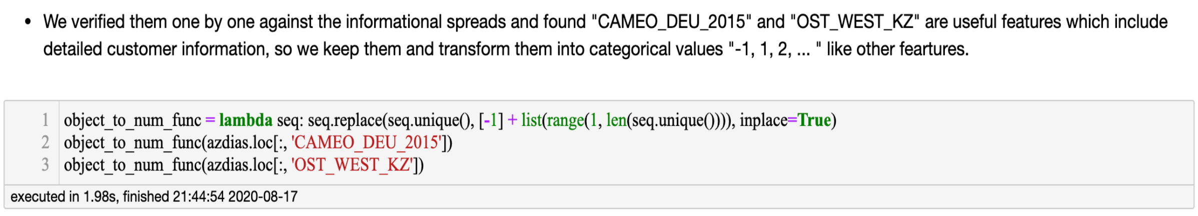
so we change them back to digital metrics.

```
azdias.loc[:, 'CAMEO_DEUG_2015'] = azdias.loc[:, 'CAMEO_DEUG_2015'].fillna(-1).replace({'X':-1}).values.astype(float)
azdias.loc[:, 'CAMEO_INTL_2015'] = azdias.loc[:, 'CAMEO_INTL_2015'].fillna(-1).replace({'XX':-1}).values.astype(float)
```

executed in 15.4s, finished 21:45:10 2020-08-17

Figure 7: Preprocessing 1

3. Transform text data into numerical format such as how 8 does.



- We verified them one by one against the informational spreads and found "CAMEO_DEU_2015" and "OST_WEST_KZ" are useful features which include detailed customer information, so we keep them and transform them into categorical values "-1, 1, 2, ... " like other feartures.

```
object_to_num_func = lambda seq: seq.replace(seq.unique(), [-1] + list(range(1, len(seq.unique()))), inplace=True)
object_to_num_func(azdias.loc[:, 'CAMEO_DEU_2015'])
object_to_num_func(azdias.loc[:, 'OST_WEST_KZ'])
```

executed in 1.98s, finished 21:44:54 2020-08-17

Figure 8: Preprocessing 2

4. Scale data to make features comparable: Figure 6 shows a variety of ranges of features, so we first eliminate this natural difference. Next, we scale them by their absolute maximum to keep variance information. This is done as figure 9 shows.

```
"""
    We first make different features have comparable ranges, so we subtract the non-negative
parts by its minimum. For example, [-1,2000,2002,2003] will become [-1,0,2,3] where "-1"
still represents the missing value but the variance range goes back to normal.

    Then, we scale features by their absolute maximal value to avoid variance being eliminated.
"""

customers_pca = customers[pca_features]
NonNegativeMin = np.where(customers_pca<0, np.inf, customers_pca).min(axis=0)
customers_pca = customers_pca - NonNegativeMin

from sklearn.preprocessing import MinMaxScaler
ss_customers = MinMaxScaler()
customer_scaled = ss_customers.fit_transform(customers_pca)
```

Figure 9: Data Scaling

3.2 Implementation

3.2.1 Unsupervised Learning

The implementation for PCA and K-means is simple, and we will just display results later for the sake of succinctness.

3.2.2 Supervised Learning

The final evaluation metric is AUC, so we aims at maximizing AUC along with watching F1-score to assess how well a candidate model works.

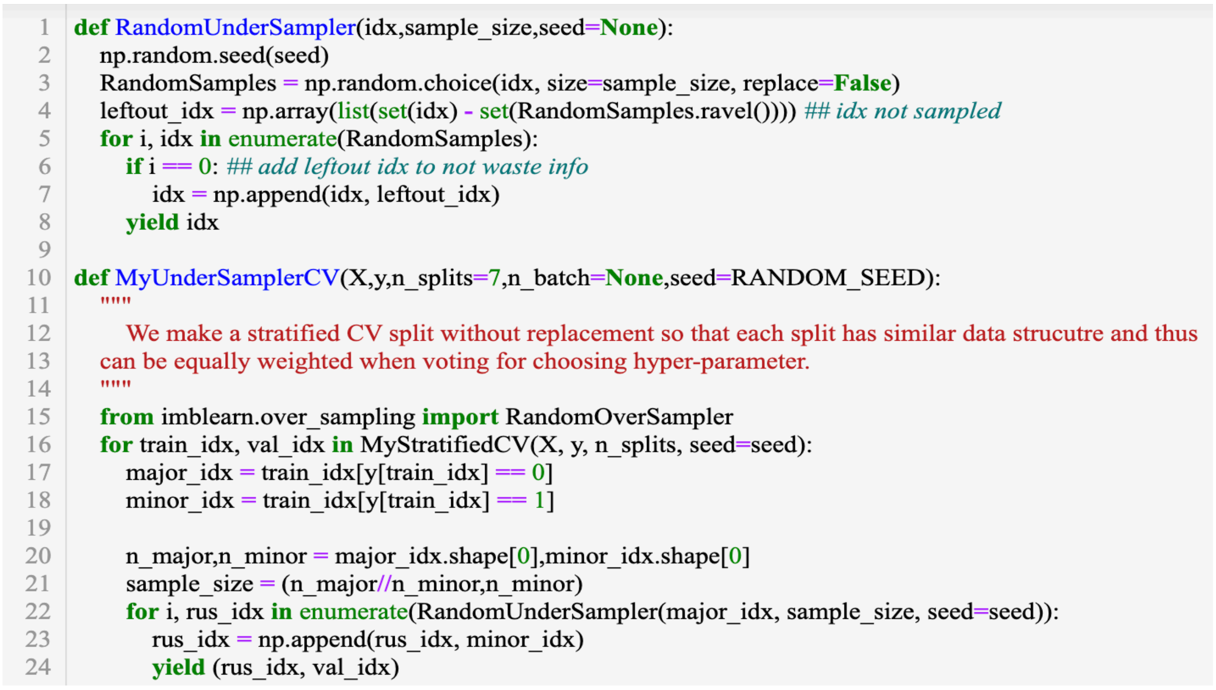
1. We start with simply assigning weight to minority to handle class imbalance. A simple snippet for developing XGBoost model is displayed in figure 10 for clarity:

```
import xgboost as xgb
def xgb_HyperParamTuning(CVsplits, dtrain, seed=RANDOM_SEED, max_bin=256, verbose=False):
    """
        A rought hyper-parameter tuning process for XGBoost using random girdsearch.
    The main purpose is to more efficiently determine the way to overcome class imbalance.
    """
    from sklearn.model_selection import ParameterSampler
    xgb_params= {'objective':'binary:logistic', 'eval_metric':'auc', 'tree_method':'hist', 'max_bin':max_bin, 'nthread':8}

    start = time.time()
    ##### ================== Hyper-Parameter Tuning Start
    ## We only tune the important ones, 'eta' and 'max_depth', using grid-search
    xgb_param_grid={'eta': np.logspace(-2, -1, 7)}
    xgb_param_dist = ParameterSampler(xgb_param_grid, 7, seed)
    xgb_rscv = MyRandomSearchCV(dtrain, xgb_params, xgb_param_dist, folds=CVsplits, maximize=True, verbose=verbose)
    xgb_params.update(xgb_rscv['best_params'])

    xgb_param_grid={'max_depth': np.arange(1,7)}
    xgb_param_dist = ParameterSampler(xgb_param_grid, 6, seed)
    xgb_rscv = MyRandomSearchCV(dtrain, xgb_params, xgb_param_dist, folds=CVsplits, maximize=True, verbose=verbose)
    xgb_params.update(xgb_rscv['best_params'])
    ###### ================== Hyper-Parameter Tuning End
    print("\t\t\t        Time:{cost}s".format(cost=round(time.time()-start,3)))
    return xgb_params, xgb_rscv

xgb_params_weighted, xgb_rscv_weighted = xgb_HyperParamTuning(my_cv, dtrain_weighted, verbose=True)
xgb_model_weighted = xgb.train(xgb_params_weighted, dtrain_weighted, xgb_rscv_weighted['cv_results'].shape[0], verbose_eval=False)
_ = print_report([xgb_model_weighted], X_train, X_eval, y_train, y_eval)
```

Figure 10: Implementation of XGBoost

The last line this code snippet will generate a report shown in Figure 17 to tell how well the model is doing on the evaluation set, which is randomly extracted before training models only for assessment purpose.

2. After determining the best model, we also need to choose an appropriate way to address with the class imbalance problem. Among four potential methods, we present how we implement under-sampling + ensemble in figure 11 for example. Others are implemented similarly.



```python
def RandomUnderSampler(idx,sample_size,seed=None):
    np.random.seed(seed)
    RandomSamples = np.random.choice(idx, size=sample_size, replace=False)
    leftout_idx = np.array(list(set(idx) - set(RandomSamples.ravel()))) ## idx not sampled
    for i, idx in enumerate(RandomSamples):
        if i == 0: ## add leftout idx to not waste info
            idx = np.append(idx, leftout_idx)
        yield idx

def MyUnderSamplerCV(X,y,n_splits=7,n_batch=None,seed=RANDOM_SEED):
    """
        We make a stratified CV split without replacement so that each split has similar data strucutre and thus
    can be equally weighted when voting for choosing hyper-parameter.
    """
    from imblearn.over_sampling import RandomOverSampler
    for train_idx, val_idx in MyStratifiedCV(X, y, n_splits, seed=seed):
        major_idx = train_idx[y[train_idx] == 0]
        minor_idx = train_idx[y[train_idx] == 1]

        n_major,n_minor = major_idx.shape[0],minor_idx.shape[0]
        sample_size = (n_major//n_minor,n_minor)
        for i, rus_idx in enumerate(RandomUnderSampler(major_idx, sample_size, seed=seed)):
            rus_idx = np.append(rus_idx, minor_idx)
            yield (rus_idx, val_idx)
```

Figure 11: Implementation of Under-Samplining

3.3 Refinement

3.3.1 Cross-Validation Splits

In the beginning, we simply called the Python "imblearn" package to do random over-sampling and sklearn's "StratifiedKFold" algorithm to split CV datasets. However, we then realize that the result was not actually stratified but just a "randomly selected" one. This is not ideal because we could not tell which data point is more important. Therefore, we self-split CV datasets for achieving stratification.

3.3.2 Sampling Technique

At first, we only planned to implement over-sampling and under-sampling techniques to overcome the class imbalance challenge. However, along the research process, we ponder that pure over-sampling attaches huge weight to positive sampling. This is dangerous because we pay too much attention on training set to clearly see the unknown future.

Moreover, pure under-sampling + ensemble is computationally expensive and may introduce much noise because each mini-batch is so small that model trained on it can easily become incapable. This makes us consider combining two sampling techniques together to avoid noise and overfitting and become more efficient.

3.3.3 Hyper-Parameter Tuning

When we decided on which model and class imbalance handling method to use, we only carried out a tough hyper-parameter tuning for efficiency. To fully exert algorithm's capacity, we need to not only make a finer search but also set a reasonable search range considering the specific "business scenario". Thus, we finally search parameters in:

1. eta: An unchanged range $[10^{-0.7}, 10^{-1}]$.
2. max_depth: An unchanged range [1,6].
3. min_child_weight: This one is different, we consider the size of our dataset and the objective function(specifically, its hessian) rather than searching blindly. The final range is $[1, ndata \times 10^{-2}]$ where $ndata$ is the size of training set.
4. gamma: Also, we take the objective function into account and set searching range as $[1, ndata \times 10^{-2}] \times ln(2)$. Here, $ln(2) = -ln(0.5)$ comes in as the expectation of binary logistic-loss function.

4 Results

4.1 Customer Segmentation

First apply scaling method described in previous sections, we then do PCA analysis for our customer dataset.

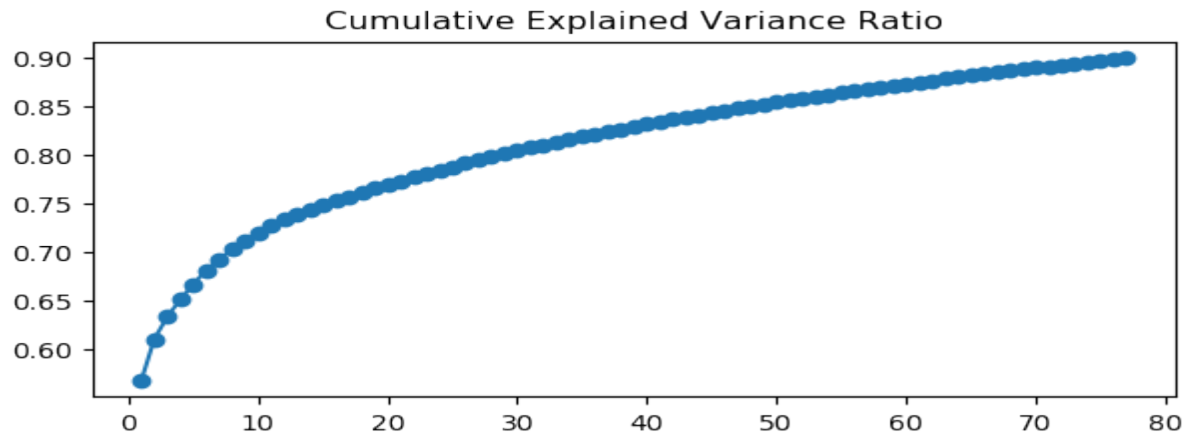


Figure 12: PCA Cumulative Variance

As shown by figure 12, we keep top 1/4 PCs that are good enough to explain 90% variance since adding more PCs will be inefficient. Next, we use K-means to separate our customers. The choice of number K is determined by drawing an elbow plot that is shown in figure 13.

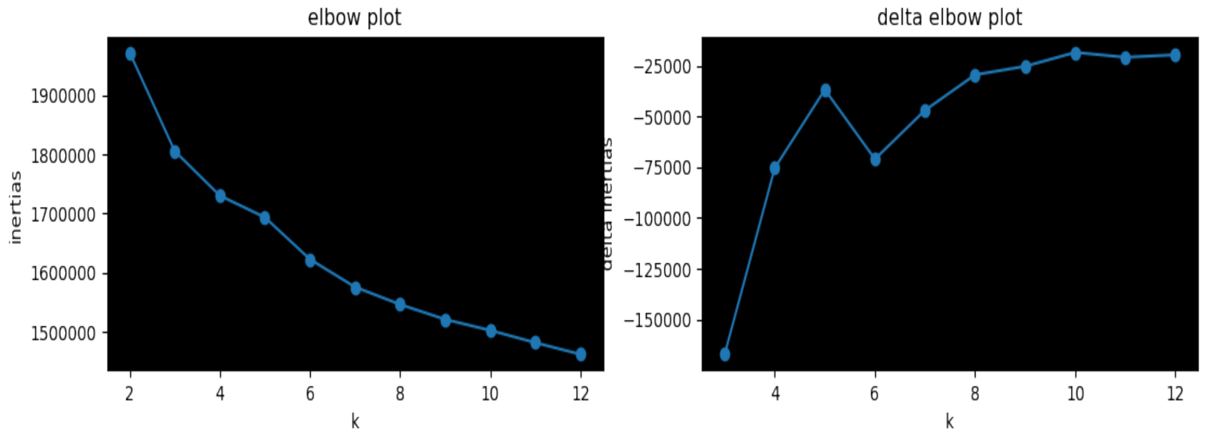


Figure 13: K-means Elbow Plot

Here, y-axis is the total distances between people and the cluster they belong to. We can observe a clear gap between K=4 and K=5, which leads us to choose K=4 at last.

Before making predictions, let us take a look at how well our segmentation is by visualizing the first PC against the second and the third PC. Note that this is only a 2-D representation that gives us a basic understanding instead of a perfect segmentation.

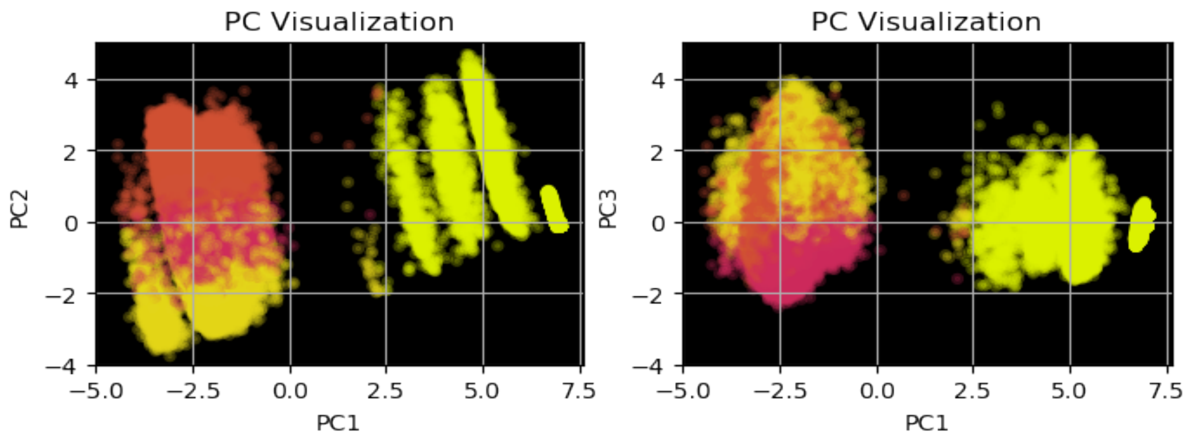


Figure 14: PCA 2-D Visualization

However, in figure 14 we can see a faint boundary in the left part that can potentially be well-separated in higher dimension.

Now, as we are confident about our clustering, we use what we learned to predict the entire Germany population. We do this also by calculating the distance between each individual and the cluster closest to him or her.

Figure 15 shows the result and two red lines are $0.5 \times AvgDist$ and $1.5 \times AvgDist$ where $AvgDist$ is the mean-distance of all current customers to their clusters. Based on this, we can predict people below the lower red line to be prospective customers and individuals above the higher one to be not.

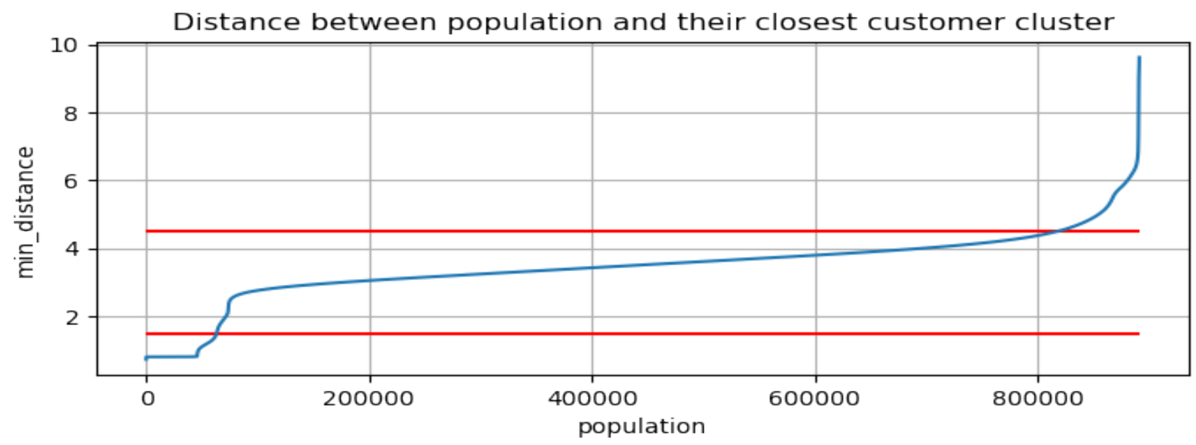


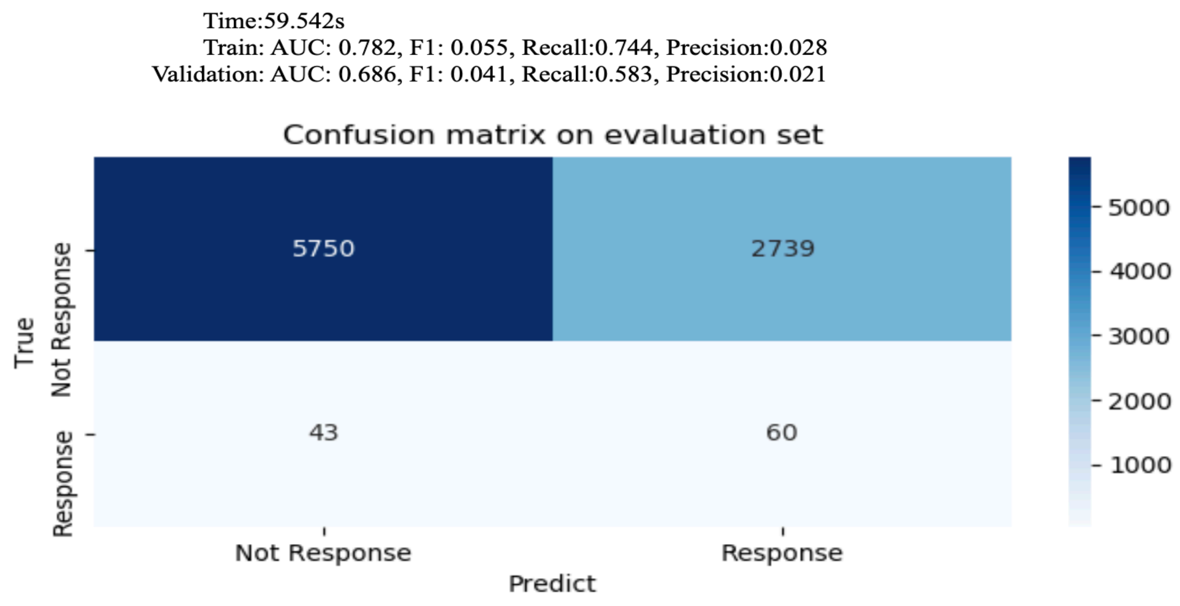
Figure 15: Population Prediction

4.2 Mail-Out Targets Prediction

4.2.1 Model Selection

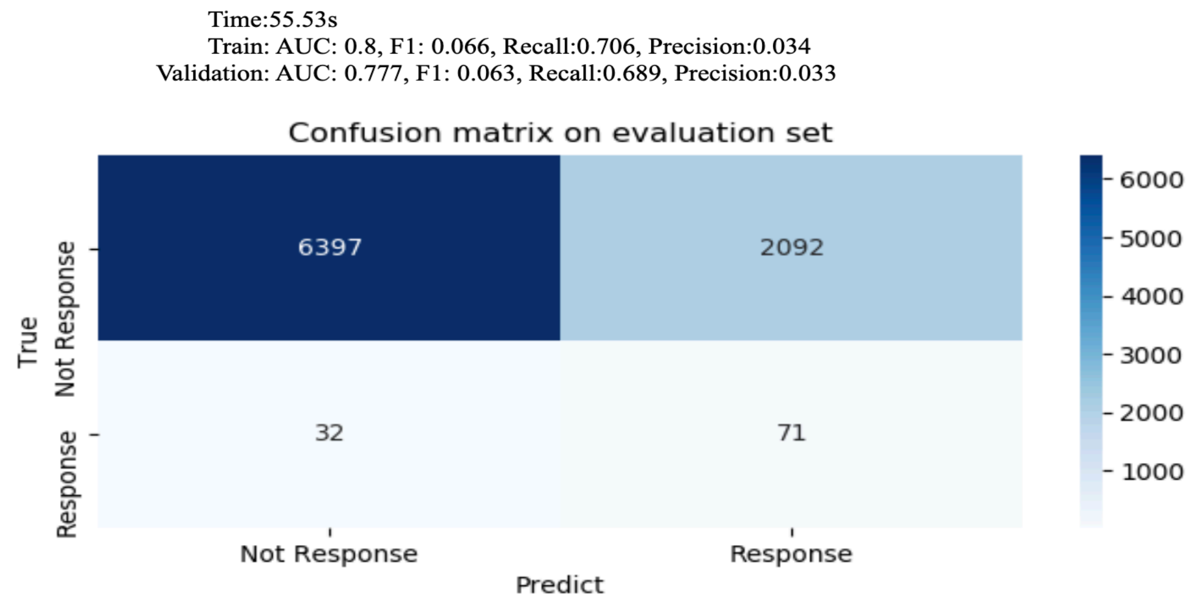
Following the implementation described above, we try benchmark Logistic Regression, SVM, Random Forest and XGBoost classifier in turn using the simplest unbalanced weight to first select appropriate model.

Finally we find XGBoost works best. Its result is reported in figure 17 against benchmark's in figure 16. The clear distinction in AUC indicates that XGBoost can easily outperform benchmark and has the capacity of making realiable predictions.



```
Time:59.542s
Train: AUC: 0.782, F1: 0.055, Recall:0.744, Precision:0.028
Validation: AUC: 0.686, F1: 0.041, Recall:0.583, Precision:0.021
```

Figure 16: Benchmark Logistic Regression



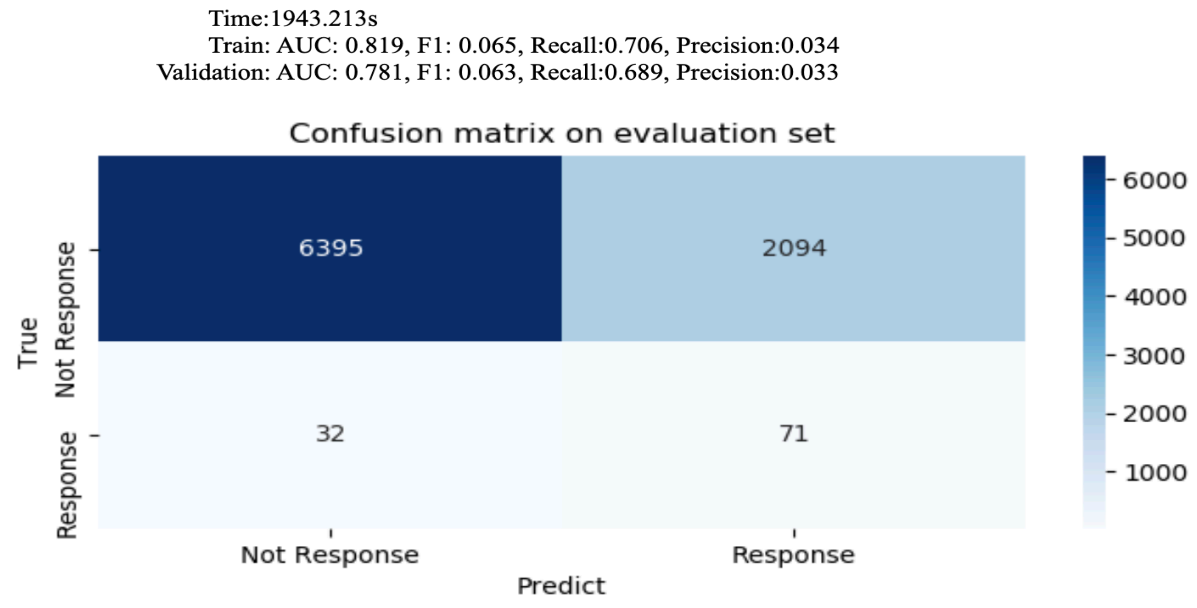
```
Time:55.53s
Train: AUC: 0.8, F1: 0.066, Recall:0.706, Precision:0.034
Validation: AUC: 0.777, F1: 0.063, Recall:0.689, Precision:0.033
```

Figure 17: XGBoost Weight Assignment

4.2.2 Class Imbalance Handling

Next, we need to determine which method to use for handling class imbalance. Four competitors are unbalanced weight assignment, over-sampling, under-sampling + ensemble, and over-sampling + under-sampling + ensemble.

We consider AUC on the evaluation set and computational cost to make decisions. Under-sampling + ensembles works best but runs slowly, and unbalanced weight assignment is almost the same as over-sampling. As over-sampling is theoretically better should we develop our models well, we finally choose to use it. Their performances are reported in figure 18 and 19.



```
Time:1943.213s
Train: AUC: 0.819, F1: 0.065, Recall:0.706, Precision:0.034
Validation: AUC: 0.781, F1: 0.063, Recall:0.689, Precision:0.033
```

Figure 18: XGB Under-Sampling + Ensemble

Time:89.942s
Train: AUC: 0.8, F1: 0.066, Recall:0.706, Precision:0.034
Validation: AUC: 0.777, F1: 0.063, Recall:0.689, Precision:0.033

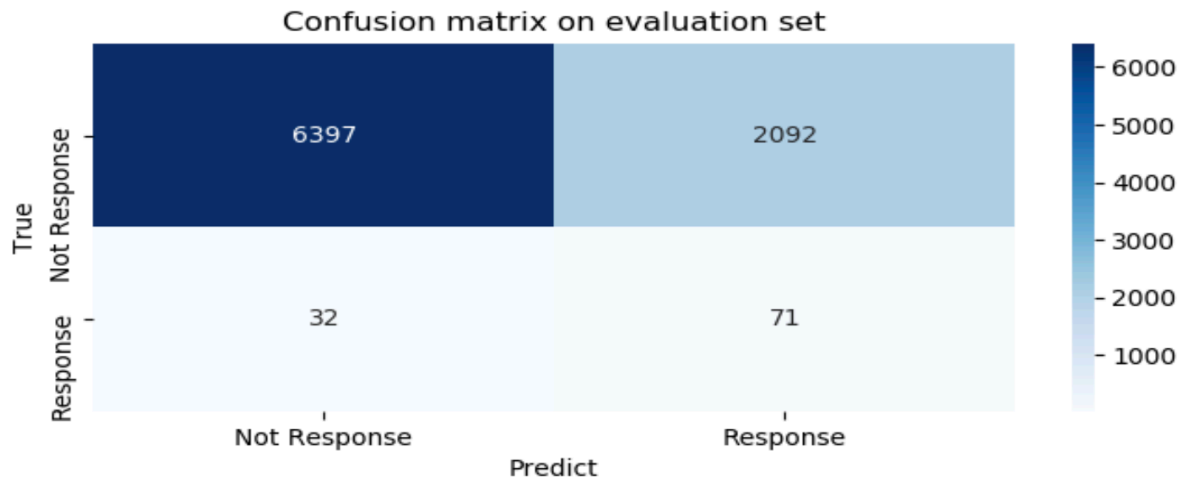


Figure 19: XGB Over-Sampling + Ensemble

Now as we are clear about the method to address with class imbalance and the model to develop, we carry out a finer hyper-parameter tuning.

4.2.3 Model Evaluation, Validation and Justification

Using a careful development described in the refinement section, our final model shows an increase of AUC on the cross-validation set from 0.771 to 0.7741. Evaluation set performance report is unavailable now because we do not know the labels of test set. Namely, we train our models only for prediction. All in all, it proves that our refinement process is necessary and the logic is well founded.

To validate the robustness of our model, we set different random seeds to check out its performance. AUC and F1-scores are almost the same, but we will not report them here for concision consideration.

Finally, as we have shown many times before, our well-developed model distinctly outperforms the benchmark in terms of AUC, F1-score, recall and precision. And because the difference is so large, we have enough reasons to believe that the final model is good enough to decently solve the problem.

5 Conclusion

5.1 Reflection

In this project, two challenges that are tricky, however, particularly interest me:

1. The lack of labels: To identify potential customer base from the entire Germany population, it is natural to think we are facing a prediction task. The problem is that we do not have access to labels. However, because in reality only a small fraction of people will end up converting to become new customers, we can treat this problem as anomaly detection and turn to embrace the unsupervised-learning

technique, such as K-means. This definitely opens my mind and provides me with ability to think about things from a different perspective.

2. The serious class imbalance: In our case, positive class is about 1%, which makes traditional model evaluation and training method lose efficacy. AUC and F1-score are widely known to do better than accuracy, however, a good way to develop model is worth pondering.

   Popular over-sampling method such as SMOTE is inappropriate because we have mostly categorical data. Pure sampling has its shortcoming, we make some improvements by combining under-sampling and ensemble or even over-sampling together. The result shows that under-sampling + ensemble is most powerful but with cost of efficiency, so we decided to use the second-best over-sampling at last.

5.2 Improvement

Although we have achieved a decent result at last, there are ways to make further improvements. In consideration of time and efforts required, we will not discuss them at this moment but leave them for future research or interested readers.

1. Feature Engineering:
   (a) In this project, we did not deeply dive into domain knowledge and ponder the business meaning of each feature to make best use of them.
   (b) For efficiency consideration, we set a correlation and feature importance threshold to filter out seemingly irrelevant features. However, this must cost final performance because we have lost some information.
2. Model Building:
   (a) We only used a single model, XGBoost, to complete this project. We can imagine that by stacking several models such as logistic regression, SVM, neural network, etc together to form a more powerful combination.
   (b) We did not choose the most powerful method, under-sampling + ensemble, at last out of efficiency consideration, which may not achieve the best result.
   (c) Further, we did not tune all hyper-parameters and only conducted a random grid-search for less important ones also for better efficiency. One with expertise and enough time can better carry out tuning process to grow the model.