PREDICT CUSTOMER RESPONSE FOR MARKETING CAMPAIGN

By Ngoc “Naomi" Nguyen

1 Project Overview

Predicting campaign response accurately will help companies target the most potential customers while saving marketing expense. To put it in numerical terms, if your overall response rate is 5% but you were able to predict the 10% most potential customers with a response rate of 80%, your return on investment would increase by 8 times compared to targeting everyone. Furthermore, targeting the wrong customer might backfire, making the product less appealing[1].

This can be done via a supervised learning model, learning from the past responses/ non-responses. This project aims to do that for a marketing campaign from a client of Arvato Financial Services. This type of modeling often face a big challenge of imbalanced class problem[2] as the response rate is very low. Tackling imbalanced class problem could be used using a combination of sampling methods and boosting models[3].

1.1 Problem Statement

We will predict who will respond to a mailout campaign. This is a binary classification problem with highly imbalanced class, with only about 1.23% of our customers in the data responds positively to the campaign. Our model will be evaluated using area under the receiver operating characteristics curve (AUC) as this is the metrics for the Kaggle competition. We will also discuss the model’s performance with respect to other metrics such as sensitivity, recall, and area under precision-recall curve so that one can evaluate the model with different objectives.

Firstly, I will choose Gradient boosting machine to be a benchmark model. After that, I will examine the data in terms of its completeness and inter-correlations between features for proper data preprocessing. I will employ a number of techniques. Firstly, I will also use random forest (XGBoost library[4]) because of its ability to reduce overfitting, which could be an issue for this small dataset, and simple, fast-to-run, small-data-tolerant models: logistic regression and naive bayes. The tree-based models will also be used for feature selection and tree-embedding features to be fed for logistic regression and naive bayes.

Beside trying out different algorithms, I also experiment with different data for training: In one experiment, I attempt to match each positive response to most similar negative responses to reduce the unbalanced prevalence between the two classes and signify their differences at the same time. In another experiment, I use the data on general population and customers instead of the training data that they provided and label the population's instances as negative and customers' as positive. This is to make use of much bigger dataset (over 1 million instances as compared to 43000 labelled instances in the training data)

1.2 Metrics

Since the data is imbalanced, we should not focus on accuracy because of the model's inclination towards the majority class. Instead, we should focus on the model's ability to rank a random true positive as more probable than a random negative, which is what AUC represents. Therefore, the evaluation metric is AUC but other metrics will be reported.

2 Analysis

2.1 Data Origin

The main dataset (train) we will work with is the demographics data for targeted individuals of the marketing campaign provided by Bertelsmann Arvato Analytics. It will include the individual's response to the campaign, which we aim to predict, as well as each's information on various levels. The dataset has info about nearly 43000 individuals with 365 independent variables and a response variable. Below are some examples of the provided information:

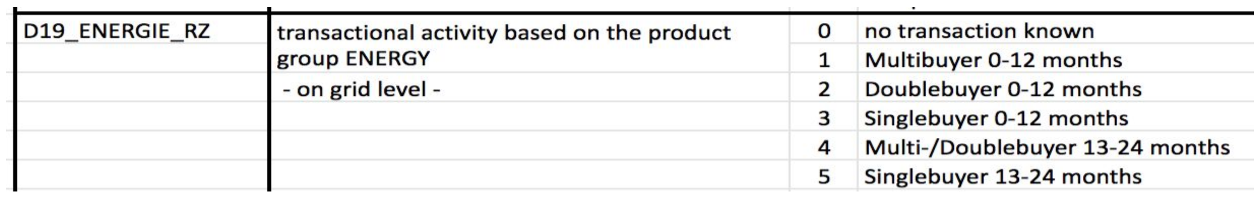
- Person: financial capability, consumption pattern, insurance, etc.
- Household: transaction activity, household structure, etc.
- Community: unemployment rate, population, etc.

There are two other datasets that will be used as a point of reference for these projects. They include the same kind of information as the main dataset but without the response variable: One for general population in Germany (azdias) and the other for customers of the mail-order company (customers).

Data description can be found in info_level and attributes explaining the type of info and value meaning for each variable. Unfortunately, not all variables are explained in these files. In addition, there are many variables and variable groups that I don't understand such as PLZ8, Microcell, 125mx125m Grid so this project will focus more on quantitative methods rather than domain knowledge.

2.2 Data Exploration

At a glance, this dataset has mostly ordinal variables, e.g. different levels of affinity, thus can be treated as numerical. There are only a few categorical variables and even fewer numerical variables. These ordinal variables could be treated either as categorical or numerical in tree-based models since tree-based models are invariant to scale. Here is one example of ordinal variables in the dataset:

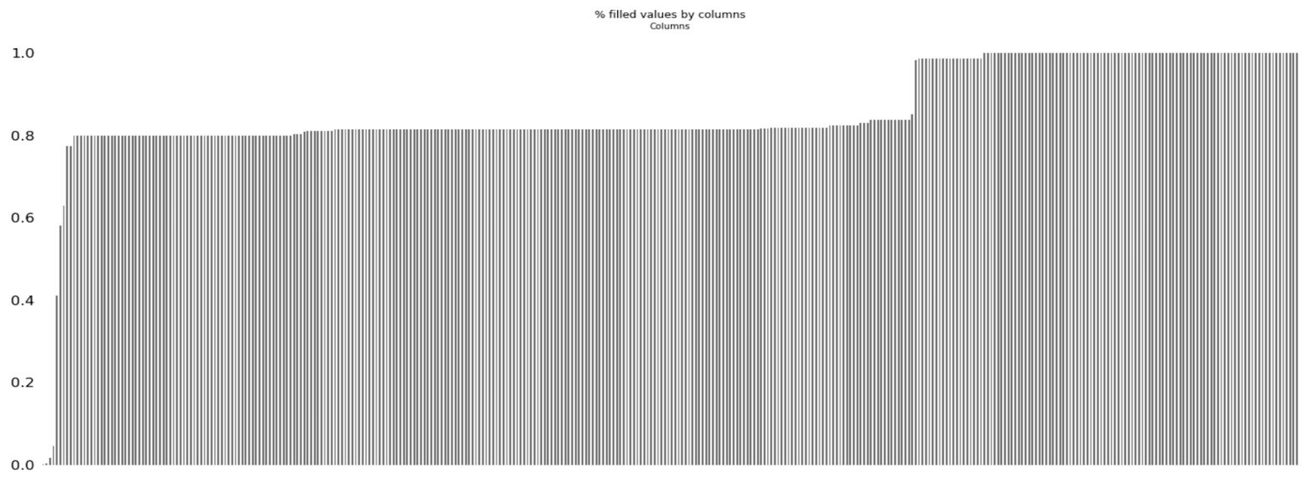


D19_ENERGIE_RZ	transactional activity based on the product group ENERGY	0	no transaction known
		1	Multibuyer 0-12 months
	- on grid level -	2	Doublebuyer 0-12 months
		3	Singlebuyer 0-12 months
		4	Multi-/Doublebuyer 13-24 months
		5	Singlebuyer 13-24 months

2.2.1 Missing Values

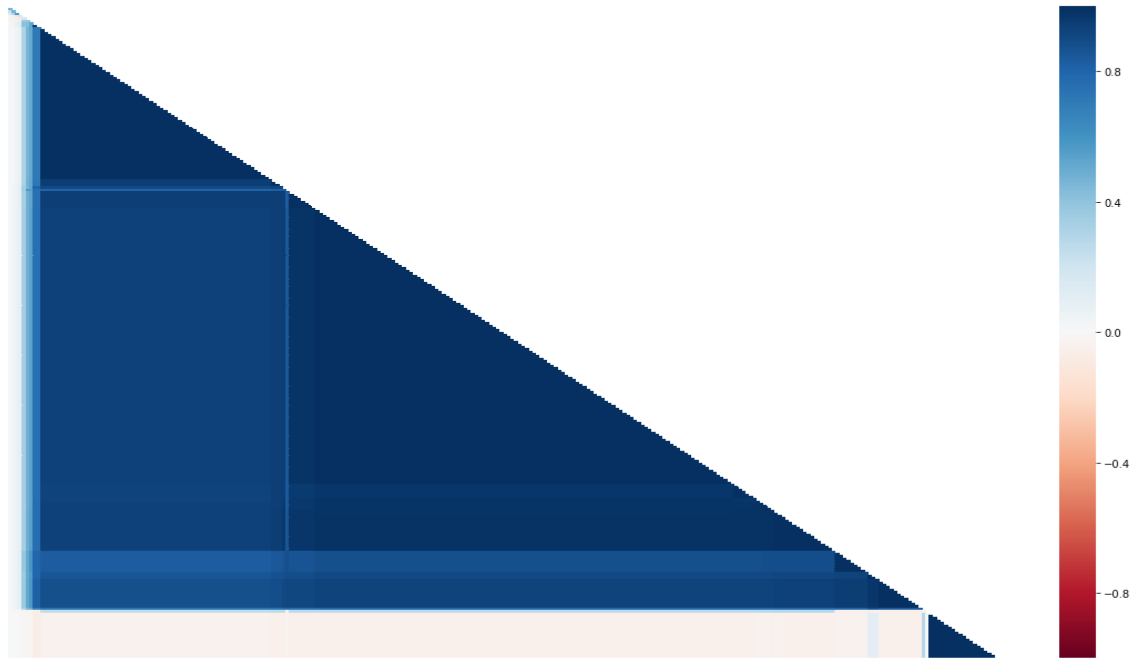
In addition, there are a lot of missing values. Below is a bar plot of density of columns in the dataset, sorted in descending order of missing values. There are a few columns with mostly missing data. More interestingly, there are groups of variables that have the same number of missing observations. This might mean that the missingness is actually informative and not random.

Fig 1. Column completeness



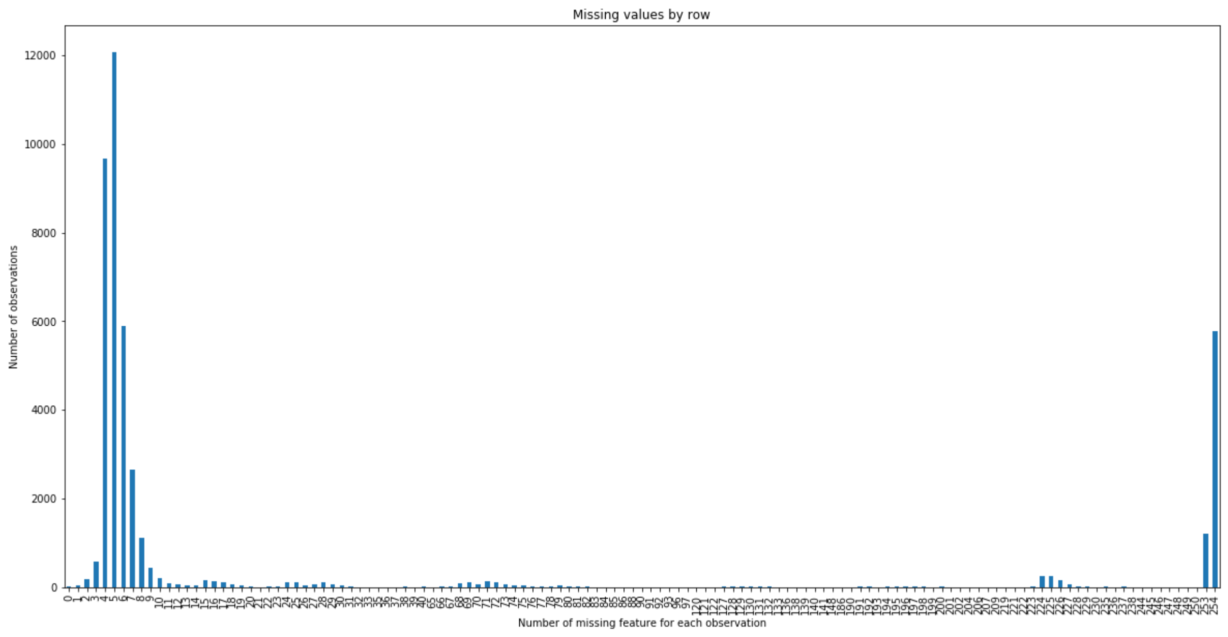
Let's look at the nullity correlation matrix below. This matrix describes how the missing values occurrences are correlated among the columns. We can see clearly that there are two clusters of correlation very close to or equal to 1 yet they are uncorrelated to each other.

Fig 2. Nullity correlation matrix



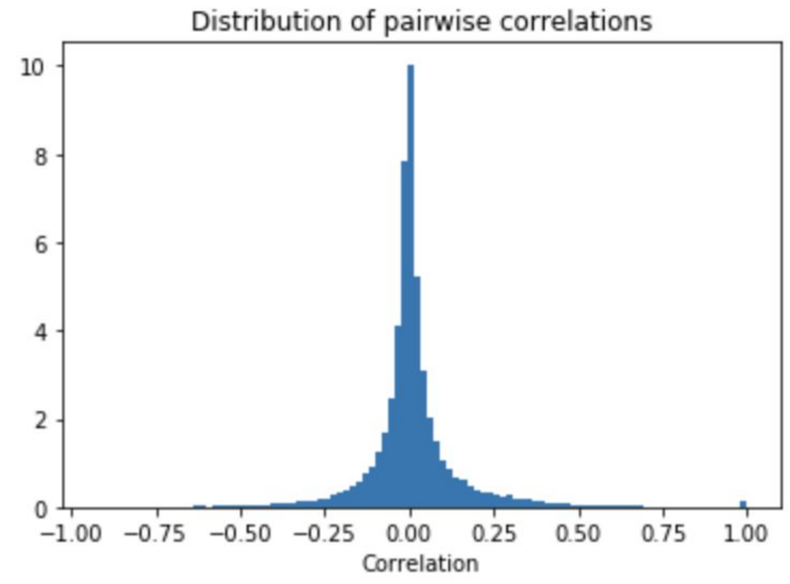
The distribution of number of missing values by observation tells us the same story: there are 2 peaks: one represents the individuals we have a lot of information and one represents those we don't. All the visualizations helped us to see that the information we don't have on those who don't are from the same variables. With this kind of informative missingness, we might not want to impute or delete the missing values but actually use it. Tree-based algorithms such as Gradient Boosting or Random Forest would be good at this.

Fig 3. Missing values by observation



2.2.2 Correlations

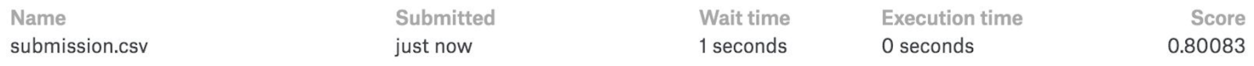
Fig 4. Histogram of pairwise correlations



There are not a lot of high correlations: under 4/10000 of the pairs are greater than 90% correlated. The only duplicate pair is `KBA13_HERST_SONST` and `KBA13_FAB_SONSTIGE`, which means we can just remove one of them. This is a good sign for logistic regression and parametric models in general. The pairs with highest correlations are:
ANZ_STATISTISCHE_HAUSHALTE ANZ_HAUSHALTE_AKTIV 0.9900891752635137
KBA13_HERST_SONST KBA13_FAB_SONSTIGE 1.0
LP_FAMILIE_GROB LP_FAMILIE_FEIN 0.9824581109517294
LP_LEBENSPHASE_GROB LP_FAMILIE_GROB 0.9850158139734342
LP_LEBENSPHASE_GROB LP_LEBENSPHASE_FEIN 0.9932450226523616

2.3 Benchmark

The benchmark model is a Gradient Boosting machine. I chose this algorithm because it has shown effectiveness in many problems and it doesn't require missing value handling. Since missing values in the problem is partly random and partly systematic as shown above, I want to see how later I could beat the benchmark with no missing value handling. The 5-fold validation AUC is 73.9% (± 1.3%). However, the performance on Kaggle's public leaderboard is 80.1% AUC, deviates greatly from the validation score.



Name	Submitted	Wait time	Execution time	Score
submission.csv	just now	1 seconds	0 seconds	0.80083

2.4 Algorithms and Techniques

a. Gradient Boosted Trees (GB)

Gradient boosting trees is a classification tree-based algorithm, which means it uses a collections of decision trees to make prediction. A decision tree will split the data into chunks, called leaves, based on a set of conditions iteratively. The choice of split is to maximize the class homogenity within each leaf and decision tree does so in greedy manner. Decision trees are better immune to outliers and multicolinearity than logistic regression or neural networks. They are also invariant with feature scale. They can learn interactions among features and can discover non-linear relationships. However, it is easy to overfit with decision trees. Decision trees are also not great at predicting values outside of seen range in training, which should not an issue to a classification problem.

Gradient boosted trees works by assembling multiple trees iteratively. However, instead of treating every instance the same, it assumes instances with wrong predictions previously are harder cases and therefore assigns higher weight in the next iteration. Details on the algorithm can be found here.

Beside the drawbacks and benefits of decision trees, gradient boosted trees is a helpful tool to tackle imbalanced class problem thanks to its boosting mechanism described above: if the model initially gravitates towards majority class, instances of minority class will have higher weights in the next iterations. Furthermore, the gradient boosted trees model implemented is from XGBoost library, which also handles missing values effectively. Altogether, we do not need much feature preprocessing with this model.

b. Random Forest (RF)

Random forest is also an ensemble of trees but assembling the trees independently. Each tree is run using a bootstrapped set of training instances and a subset of features (gradient boosted trees in XGBoost library can also be configured to do this). These trees will vote for the final predictions. By using an ensemble of independent trees using different datasets, random forests alleviates the overfitting problem of decision trees, which is very helpful for our small dataset. The random forest implemented is also from XGBoost library, which means preprocessing missing values are not required

c. Logistic Regression (LR)

Logistic Regression is a classification models which depicts the relationship between the explanatory features and the log probability of the positive class. This statistical model allows us to measure the statistical significance of each feature. It is interpretable, more immune to overfitting, and very fast to run. On the other hand, it is not flexible and is susceptible to multicolinearity and outliers.

The logistic regression model is implemented with scikit-learn library[5].

d. Naive Bayes (NB)

Naive Bayes is a classification model using Bayes' Theorem but with a strong assumption of independence among the explanatory features. However, in practie Naive Bayes has worked quite well even when the data is sparse and correlated.

The Naive Bayes model is implemented with scikit-learn library.

e. Negative downsampling

When there is a high class imbalance, the models will tend to predict the majority class because a naive guess of the majority class is mostly right. By reducing the number of the majority class, i.e., negative downsampling, we balance the prevalence between the majority and minority class thus reducing the model's tendency to predict the majority class.

Since the dataset is small, I implement negative downsampling with one little twist: I match 5 most similar non-responses for each instance of positive response. That way, I can reduce the feature dimension for the columns that are different.

f. Transfer learning

Since the main dataset is quite small. I will also try to solve a different problem: differentiating the customers and the general population. I will then use the trained models to apply to our main dataset and evaluate its performance.

g. Tree-based embedding

To incorporate the stability of logistic regression or naive bayes and the flexibility of tree-based ensemble, I transformed the result of tree-based ensemble into an embedding such that the entries are leaf the instances fell into and the entries are then one-hot encoded. This is inspired by Xinran He and coauthors in Practical Lessons from Predicting Clicks on Ads at Facebook[6].

h. Principal components (PCA)

Parametric models such as logistic regression often have a strong assumption of interdependence among the features, which is not often the case. To address this problems, one could transform the feature space into independent (orthogonal) features, namedly principle components. Each principal component will be constructed such that the first component explain the most variance in the data and each component after will contain the most remaining variance while being orthogonal to the previously constructed component. This is also a well-known technique for dimensionality reduction when one wants to retain the most variance in the data while having the fewest features.

3 Methodology

3.1 Data Preprocessing

3.1.1 Basic Data Cleaning

Firstly, I:

- Remap values representing missing values to missing values
- Remap different values yet have the same meaning in `LP_FAMILIE_GROB` and `LP_STATUS_GROB` to have the same values
- Seperate GEBAEUDETYP into 2 variables: building type and indicator for any known household
- Get year of EINGEFUEGT_AM

I delete columns with high correlations with at least one another columns and also delete columns with too many missing data:

Deleted Columns	Reason
KBA13_HERST_SONST	Duplicate completely with KBA13_FAB_SONSTIGE
LP_FAMILIE_FEIN, LP_LEBENSPHASE_GROB	Highly duplicate with LP_FAMILIE_GROB
`ALTER_KIND`s	Over 90% missing
'KK_KUNDENTYP', 'EXTSEL992'	Over 90% missing

Later on, I will try both leaving missing values as is and imputing missing values. I impute the missing values with the following procedure:

- Fill missing values in columns starts with 'KBA' as -1.
- For the other columns, I trained an iterative imputer using the demographics data for general population and then apply the imputer to the `train` dataset. The imputer is a Baysian Ridge regressor. All the imputed variables are numerical/ ordinal variables. After that I clipped the imputed values to be within the range of data before imputation. The imputed data is used to fed models implemented by scikit-learn
- I also tried scaling and PCA on the variables for Logistic Regression and Naive Bayes but it decrease performance significantly

3.1.2 Feature Selection and Creation

Firstly, I create time difference variables among the time variables. The time difference between year of birth and `EINGEFUEGT_AM`, a variable that I don't have information on appears in the top 20 features 11 times out of 50 run of random forests while the individual features have much lesser feature importance.

My first experiment with feature selection is running 10-time 5-fold cross-validation using random forest with default parameters in XGBoost library. Then, I will track top 20 features in each of the 50 runs and select features that appear in the top for at least 5 times. There are 56 such features. This feature selection helps improve the performance for logistic regression and naive bayes but doesn't help tree-based algorithms much. However, it helps reduce variance (overfitting) in both cases.

My second experiment is try to map each example with positive response with most similar example with negative response (negative sampling). Then for each of such pairs, I will extract different columns then retain the most frequent different columns. This leads to a similar effect with the first experiment but with lesser extent.

3.3 Results

3.3.1 Methodology

I tried a combination for each of the options:

- Model: Logistic Regression, Naive Bayes, Random Forest, Gradient Boost
- Training data: Train directly on train dataset or create a different dataset consists of the general population of Germany `azdias` and of the company's customers `customers`
- Features used: Used the original features plus newly created features and/or create tree embedding using Gradient Boost/ Random Forest then use the embedding as features for naive bayes and logistic regression. PCA features are also tried but it reduced performance significantly.

Each experiment is then run with stratified 5-fold cross validation with different number of repeats depending on runtime of an experiment. For models with description of transfer learning, the reported 'Mean CV AUC' is actually the AUC on the `train` dataset, which does not participate in the training or validation phase.

Below is a summary of the model performance are included. Experiments with PCA features have low performance so I did not track it.

Model	Mean CV AUC	Std CV AUC	Kaggle's Public Leaderboard
LR*	68.8	2.0	
NB*	63.3	2.2	
RF*	76.5	2.1	
GB*	75.8	1.8	

RF with negative sampling			77.253
NB with RF embedding*	76.3	1.6	
LR with RF embedding*	75.3	2.	
NB with GB embedding*	76.7	1.7	
LR with GB embedding*	69.6	1.9	
NB with GB embedding - transfer learning*	76.8	2.1	79.529
LR with GB embedding - transfer learning*	75,7	2.2	
RF - transfer learning	77.1		
RF - hyper-parameter tuned	77.4	2.5	80.444
*Models use only top-performing features, without * use the full set of features NB/LR with GB embedding* behaves similarly with NB/LR with RF embedding* respectively			

3.3.2 Hyper-parameter tuning
I applied random search (implemented with scikit-learn) for tuning hyper-parameters for random forest. Random search has been shown to be more effective than grid search[7]. My strategy is to search the hyper-parameters iteratively. If any of the best hyper-parameters are at the edges of the range, I will extend the range in that direction. I also eyeball check the mean performance corresponding to each hyper-parameters to make sure it the best hyper-parameters are “best” because of randomness.

Hyper-param	Search space	Hyper-param of best model
Maximum depth	3, 4, 5, 6,	5
Minimum weight in each leaf	10, 20, …, 90	100
Number of trees	10, 20, …, 90	20
Weight ratio of positive instances to negative instances	1, 5, 10, 20, 30, 40, 60,80, 100	10
Reg_alpha	0.001, 0.01, 0.1, 1, 1.3, 1.5	1.3

Learning rate	1, 0.3, 0.1, 0.01, .005, 0.001	0.001
% of features used in each tree	0.2, 0.5, 0.8, 0.9, 1	0.7
% of data used in each tree	0.7, 0.8, 0.9, 1	0.8
Gamma	0.01, 0.1, 0.3, 0.5, 1, 1.5, 2	0.3

The best performing models has mean cross-validated AUC of 77.4% and achieved 80.4% on Kaggle's public leaderboard. The best performing model outperforms the benchmark by 0.3 percentage AUC points. However, the score on the public leaderboard only accounts for 30% of test data so the cross-validated score would be more unbiased. In this metrics, the tuned models outperforms by 3.5 percentage AUC points (77.4% compared with 73.9%) even though with a higher standard deviation.

14 NaomiNguyen 0.80444 11 22m

4 Discussion

My two best submissions are a naive bayes with tree-embedding features from random forest model learned from data from both `azdias` and `customers` and a hyper-parameter-tuned random forest trained directly on the `train` data. However, the first one is much more stable across validation folds, even though the latter produce better performance based on Kaggle's public leaderboard.

It is also very interesting to see that using `azdias` and `customers` data could be so informative in predicting responsive for the marketing campaign. This probably means that the campaign acts more of a tool to reach out customers who already have interest in the service but haven't heard about the company rather than to convince/ attract customers who do not have interest at first. I would wish to look at which customers do these two models predict different and which the same. I think the customers where transfer learning predicts positively are the customers who have interest, so the company just need to reach out to them. The customers where the direct training predicts positively yet the transfer learning predicts negatively are harder to work with but it is possible to help them realize the benefit of the service. These customers may need offers of free trial/ discounts.

Another interesting point is that since the dataset is quite small, it's very important to have the right training data. The standard deviations among the folds are about 2%, which is quite significant. For example, in the model with Naive Bayes and tree-embedding features from

Gradient Boost transfer learning, there is a fold where train and test performance is around 76.5% where as there is another fold that has above 79.5 performance, which also have the similar performance on Kaggle's public leaderboard. This is different when I train using the population and customers data; models very stably across folds and remains about the same between training and validation set.

In all models, the hero features that differentiate between the responses and non-responses are `D19_KONSUMTYP`(consumption type), `ANZ_PERSONEN`(number of people known in the household), `D19_SOZIALES`, `RT_SCHNAEPPCHEN`, `KBA05_SEG2`(share of small and very small cars in the microcell), and `D19_KONSUMTYP_MAX` (which I don't have information on).

Error analysis should be conducted for further research to better understand which customers are likely to be missed by the model. For model performance improvement, deep learning could be used to learn complex interactions among the features. Model stacking should also be used to get the best of multiple models. Automated hyper-parameter tuning using Bayesian Optimization[8] would help to explore the hyper-parameter space better. I also could have extract the neighborhood's social-economic data from the postcodes to determine the relative affinity of each individual as well.

References

1. *Response Modeling*, Big Data Analytics http://www.bigdatanalysis.com/response-modeling/
2. C.X. Ling, C. Li, "Data Mining for Direct Marketing—Specific Problems and Solutions", Proc. Fourth Int'l Conf. Knowledge Discovery and Data Mining (KDD '98), pp. 73-79, 1999
3. Sun, Yanmin, Andrew KC Wong, and Mohamed S. Kamel. "Classification of imbalanced data: A review." International Journal of Pattern Recognition and Artificial Intelligence 23.04 (2009): 687-719.
4. XGBoost library https://xgboost.readthedocs.io/en/latest/index.html
5. Scikit-learn library https://scikit-learn.org/stable/index.html
6. Xinran He et. al., Practical Lessons from Predicting Clicks on Ads at Facebook, Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, p.1-9, August 24-27, 2014, New York, NY, USA [doi>10.1145/2648584.2648589]
7. Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." Journal of Machine Learning Research 13.Feb (2012): 281-305.
8. Snoek, Jasper, Hugo Larochelle, and Ryan P. Adams. "Practical bayesian optimization of machine learning algorithms." Advances in neural information processing systems. 2012.