Springleaf Customer Marketing Response

Robert Ciesielski

Executive Summary:
To do

Contents

1. Introduction

Springleaf Financial (currently OneMain Financial[1]) is a company that provides personal loans and optional insurance products to customers with limited access to traditional lenders, such as banks or credit card companies. The Springleaf's main marketing strategy is based on direct offers sent to customers by mail. The company is interested in improving its targeting efforts by efficiently focusing on the customers who are likely to respond to these offers. A the end of 2015, the company released an anonymized dataset with about 2,000 features to the Kaggle community[2], asking it to predict which customers will respond to a direct mail offer. The model evaluation metric suggested by Springleaf is the area under the receiver operator curve, roc_auc.

This report describes our effort in modeling customer adoption using the Springleaf dataset. We focus on finding an optimal method to reduce the dimensionality of this dauntingly wide dataset. We compare the performance of various ensemble-based models, such as Random Forest, two scikit-learn Gradient Boosting models,, XGBoost and LightGBM. The customer response is also modeled using a feed-forward neural network architecture with three and four hidden layers

2. Dataset, software and computing

The data is available from the Kaggle platform as two csv files, one for model training and the other for model prediction and evaluation. The training dataset contains more than 145,000 rows corresponding to customers and 1932 columns with features describing the customers. The features are anonymized and numbered as integers. An additional column describes the binary customer response for supervised learning. About 23 percent of the customers respond positively to the offer (Tab. 1).

The computation is done on a PC with 8 CPU units and 16 GB RAM, using latest versions of pandas, numpy, scipy, scikit-learn, xgboost, lighgbm, keras and tensorflow libraries, as implemented in the anaconda v1.7.2 package. The hyperparameter tuning is computed on the cloud using the SegMind platform[3]. An instance of 32 CPU units and 128 GB RAM allows us to increase performance speed by a factor of 5. When running the BayesSearchCV function from the latest scikit-optimize (0.8.1) package, the scikit-learn and scipy libraries are downgraded (from versions 0.24.1 and 1.5.3 to 0.23.2 and 1.4.1, respectively) to assure the compatibility with scikit-optimize.

[1] https://en.wikipedia.org/wiki/OneMain_Financial
[2] https://www.kaggle.com/c/springleaf-marketing-response/overview
[3] https://segmind.com/

Customer response	N. of rows	
No (0)	111,458	(76.7%)
Yes (1)	33,773	(23.3%)

Table 1. Binary customer response (target variable).

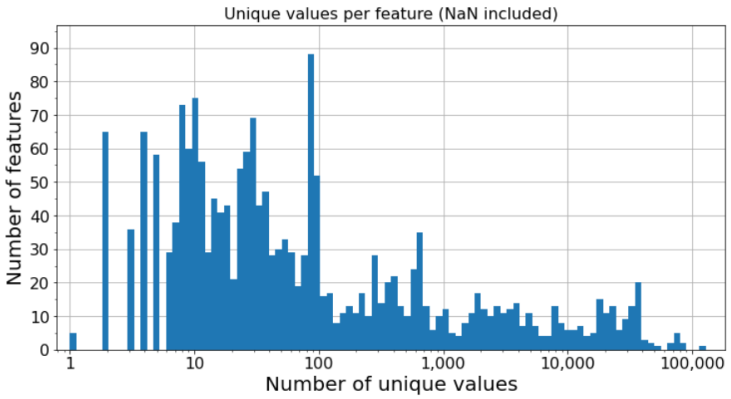

Figure 1. Number of feature unique values.

3. Data Wrangling

We begin with examining the dataset in terms of missing values, feature types, and unique values per feature. Fig.1 shows a histogram with the number of feature unique values. The spectrum is very wide and varies from 5 columns with just one unique value to a column with more than 100 thousand values. Features with only one unique value have no variance and are removed from the dataset. In this initial step, we inspect the content of the features that have at most five unique values. We find out that the majority of the two-value columns contain a NaN (not-a-number) and another value appearing as a string, a bool, or a number. For these columns, we preserve the NaNs but convert all the other values to 1. There are also four columns with the values of -99999 and 0. To unify the notation with the other columns, we replace -999999 by 1. A few columns contain logical variables denoted with boolean values False or True. We convert them to 0 and 1, respectively. Finally, we check for duplicates. After the replacement mentioned above, most of the columns with two unique values are duplicated, so we remove them from the dataset. We also find duplicate columns among those with 3 unique values. We remove them as well. None of the columns with 4 or 5 unique values are repeated. After this step, we are left with 1881 features (51 removed).

Tab. 2 shows the data type of the features in the dataset after the initial cleaning. Possible missing values are ignored when inferring the feature type. The dataset is dominated by integer columns (98%) with a small admixture of float and DateTime columns and categorical features represented by strings.

Feature type		N. of features
Numerical:	Integers	1,837
	Floats	10
	DateTimes	18
Categorical:	Strings	17

Table 2. Feature data types.

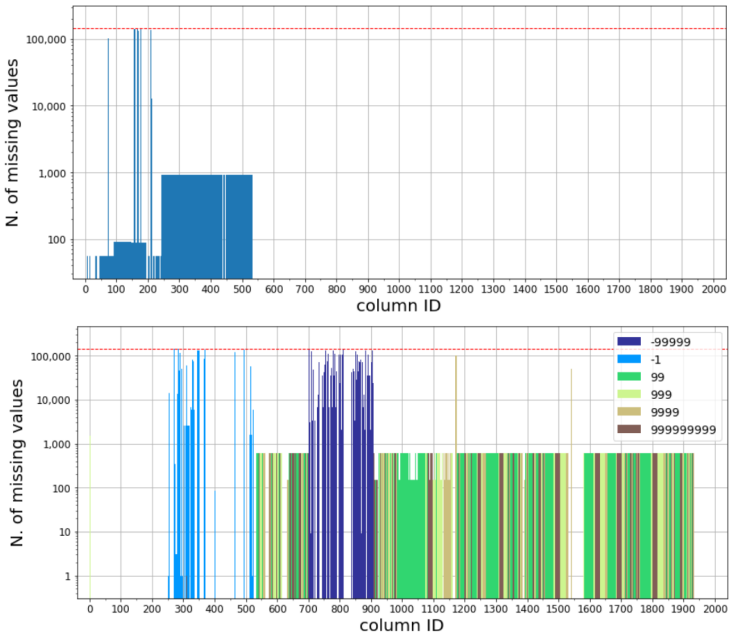

Figure 2. Number of rows with missing values per column, with the missing values explicitely denoted as NaNs (top) , or represented by frequently repeated negative or positive numbers given in the legend (bottom). The dotted red lines correspond to the total number of rows in the datasets.

The number of missing values per feature is shown in Fig. 2 (top). Only the first 530 columns in the dataset contain missing values explicitly denoted as NaNs. Their number is very high for a few columns and close to the total number of dataset rows (red line). Most of the columns have the missing value fraction below 1%. Missing values could also be encoded as integers, e.g. as the -1 digit or a very small or large numbers ($1-10^n$ or 10^n-1 for $n=2,3,\ldots$). By searching for the most frequently repeated column minimum and maximum values, we find those to be: -99999, -1 and 99, 999, 9999, 999999999. Fig. 2 (bottom) shows the frequency of their appearance. Overall, missing values are present in most of the columns in the dataset.

4. Exploratory Data Analysis and Feature Engineering

Given that the data is anonymized, we could blindly apply label- or one-hot encoding to convert categorical features into numerical ones and quickly proceed to the modeling stage. However, we will take an extra effort to explore and engineer the features, and hopefully get insight into the origin of the data. We inspect the columns by their types shown in Tab. 2 (in reversed order):

4.1 Categorical variables

Among 17 columns with string-encoded categorical features:

- One column contains more than 12,000 city names. Using an additonal dataset from the U.S. Geological Survey and U.S. Census Bureau, containing information on all Census-

recognized US cities and towns,[4] we convert these names to two new features with the city population's size and density. By explicitely matching the city and state names in both datasets, we are able to successfully convert 90% of the rows. The remaning rows are not matched because of some common differences in the notation (e.g. SANTA/SAN vs ST.) , abbreviations (e.g. SLC vs SALT LAKE CITY), or spelling mistakes (e.g. TYLER vs TAYLOR). Additionally matching by the ZIP code, which we find on one of numerical column in our data, increases the convertion rate to 96%. The remaining rows corresond to unincorporated locations (source: Wikipedia), with no information on the population size. For these, we set the population size and density to 0.

- Two columns are filled with about 1,200 and 600 unique names of professions. Both contain $\sim 90\%$ of empty rows. Fig.3 shows the most frequent words found in one of the columns, separately for the samples with the positive and negative customer response. Given that there is no obvious difference in customer response and that the columns are dominated by missing values, we remove them from the dataset.
- Two columns contain US state codes, 46 and 58 unique values each. The first one is filled with most of the US states, except for those in New England. The second one contains all the 51 states (DC included), plus additional codes such as PR, EE that could stand for Puerto Rico, Estonia, etc. We conclude that the first column corresponds to the states where the data were collected, while the second one may describe the customer's place of birth. We label encode the information contained in the first column, ordered by the increasing value of the average target response per state, see Fig. 4, and we replace the second column by a new column, filled with 1 (0) if the state code is the same (different) in both columns.
- One columns is filled with 50 unique values of two-letter codes XY with X, Y = A,B,C,D,E,F,U. We label-encode them in a similar manner as the US state information.
- Remaining 11 columns contain low-multipicity (3-10 values) letter-encoded features, as shown in Fig. 5. Two fo them shed light on the origin of the data: one contains abbreviations such as *IAPS*, *RCC*, and *CSC*, related to cancer treatements; another column, with words *Discharged*, *Dismissed* and *Discharge*

[4] https://simplemaps.com/data/us-cities

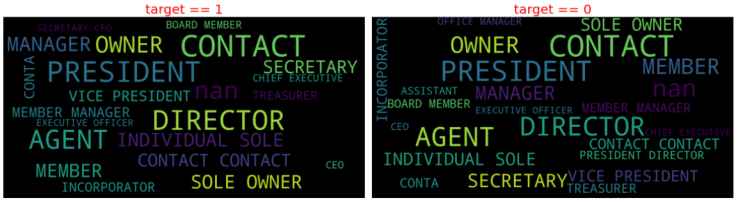

Figure 3. The bag of most frequent words in one of the two columns with profession names, shown separately for the subsamples with the target variable = 1 (left) and 0 (right).

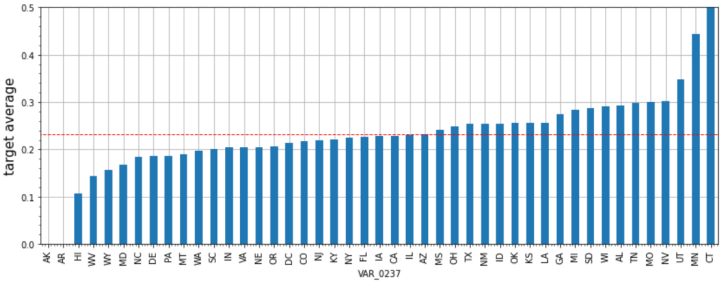

Figure 4. Average customer response (target variable) per US state. The average response for the entire dataset is shown as the red line. Deviations from this line indicate that the variable has potential to separate the 0 and 1 customer responses.

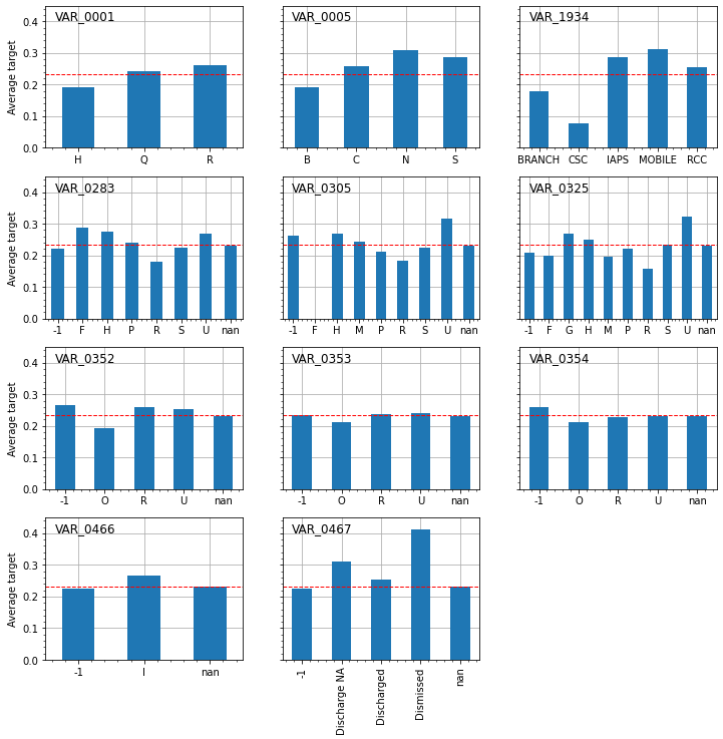

Figure 5. Average customer response for categorical features with low number of unique values (≤ 10).

NA, may describe a history of hospital admissions. So, we deal with a healthcare related marketing offer. We convert all the 11 features to the numberical ones using label encoding ordered by the average target response. This approach gives us slightly better performance than standard one-hot or label- encodings (tested with Decision Tree or Logistic Regression models). Also, we notice that the features in the second and third row of Fig. 5 are encoded using the same letters. We create additional features that count the number of appearances of each letter in each row of the dataset.

4.2 Numerical variables

4.2.1 DateTime features

Among 16 columns with the DateTime format, 3 columns are fully filled with data. The remaining 13 columns have a signif-

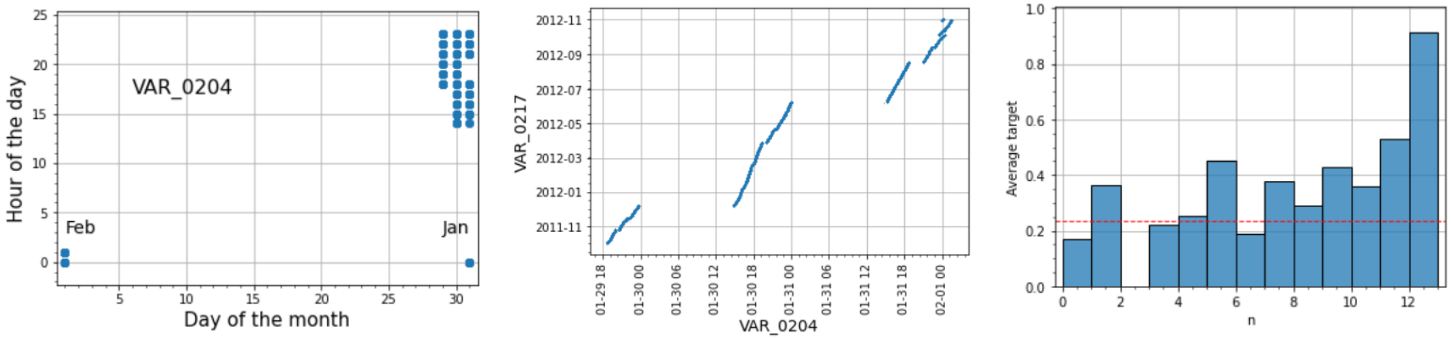

Figure 6. Left: the hour of the day versus the day of the month for entries in the column that covers 3 days (notice the jump from Jan to Feb on the horizontal axis). Middle: the time correlation between two columns with shortest time periods, explained in the text. Left: Average target response versus the number of non-zero entries in the DateTime columns summed over the rows.

icant number of empty rows (70-99%). One of the fully-filled columns spans the time range of mererly 3 days, another one of about a year, while all the others cover periods of 4 to 13 years. Fig. 6 (left) shows the hour of the day versus the day of the month for entries in the column that covers 3 days. There is a one-to-one correspondence between this column and the column that spans one year, as shown in the middle plot. No obvious correlations are found between any other columns. Under the assumption that during 3 afternoon shifts at the end of Jan 2014 somebody compiled information about other chronologically-ordered data, we remove the first column from the dataset. For other columns, we convert the DateTime entries to floats, by taking the difference w.r.t. to the smallest date within each column. Missing values are replaced by -1. Moreover, in each row we count non-zero DateTime entries and save the counts as an extra feature. Fig. 6 (right) shows a strong dependence of average target response on these counts. They may represent the number of treatments a customer underwent (the more treatments the higher the probability of the positive customer response).

4.2.2 Floats

There are 10 columns in this class and all exhibit steeply falling exponential distributions. One notable feature is that they contain two types of missing values: NaNs and -1 digits. The average target response for rows with these values differs from the overall average by 2 and 1%, respectively. For this reason, we replace NaNs by -2 to keep them separate from the -1 values.

4.2.3 Integers

This class is present in 98% of the columns. Since it would be impractical to inspect all of them in detail, we focus on some aggregated information. We group the columns by missing values that they contain, encoded as (see Sec. 3):

- negative numbers: -1 and -99999; Fig. 7 shows standard box plots (with the median, 1st and 3rd quantiles, Tukey's wiskers, etc) for 104 and 156 columns containing these numbers, after removing them from the spectra. The columns are ordered with increasing values of the median, showing a very large variation in the range of distributions, filled with positive numbers. A visual inspection of selected columns shows that they are dominated by steeply falling spectra, most likely representing medical measurements. There are also columns with numbers that look like e.g. a year or a ZIP code. Whle the -1 digit seems adjacent to the distributions, the -99999 number lies 5 orders of magnitute apart from the spectra, in most cases creating an enormous gap that might be problematic for modeling. To avoid possible issues, we replace it by -1.
- large positive numbers: 99, 999, 9999 and 999999999; present in 458, 238, 159 and 257 columns, respectively. Here, we discover much more complicated structure, as these numbers are accompanied by other adjacent large numbers. The columns consist of a set of numbers (994, 995, 996, 997, 998, 999) or (9994, 9995, 9996, 9997, 9998, 9999) or (999999994, 999999995, 999999996, 999999997, 999999998, 999999999), which are separated from the rest of the spectra by a gap not larger than two orders of magnitude (for most cases). Such a picture is difficult to understand without an additional domain knowledge, so we leave the data unchanged.

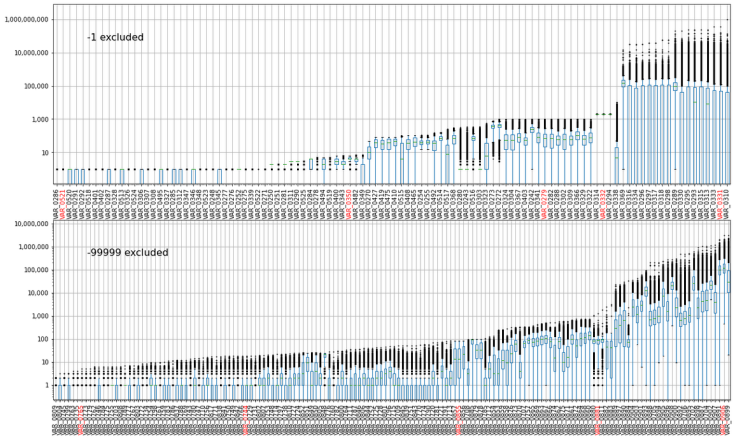

Figure 7. Box plots as a function of column ID for integer columns with missing values encoded as negative numbers: -1 (top) and -99999 (bottom), after excluding them from the data.

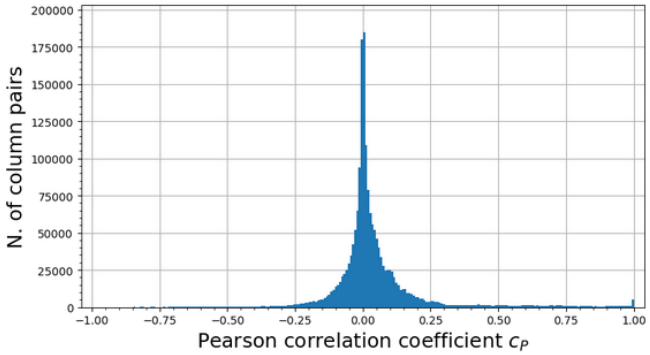

Figure 8. Pearson correlation coefficient for pairs of columns.

5. Dimensionality Reduction

The following four approaches are explored to reduce the dimensionality of the dataset, consisting of 1,892 columns.

5.1 Correlation Matrix Method

This method relies on a pair-wise comparison of columns in the dataset and removal of those that are highly similar. The level of similarity is measured in terms of the Pearson correlation coefficient c_P ($-1 < c_P < 1$), shown in Fig. 8. The distribution of c_P is centered at zero implying that the data consists of minimally correlated column pairs. A rather loose cut $|c_P| > 0.5$ would remove only 5% of columns, not sufficient for desired dimensionality reduction. In the plot, a small excess of columns at $c_P = 1$ corresponds to fully correlated pairs . We remove 12 duplicates from the dataset and are left with 1,880 columns.

5.2 Principal Component Analysis (PCA)

PCA allows us to create a new set of features (principal components, PCs) that are linear combinations of original features. Typically, less PCs than original features are needed to fully describe the variance of data. We exploit this property for dimensionality reduction. After scaling the data and applying PCA, we find that 500 and 700 PCs are needed to explain 97 and 99% of variance in the data, respectively. Unfortunately, we observe a significantly poorer modeling performance for the reduced datasets, or even for the full dataset after the transformation (all 1,880 PCs). For the latter, the roc_auc score of the

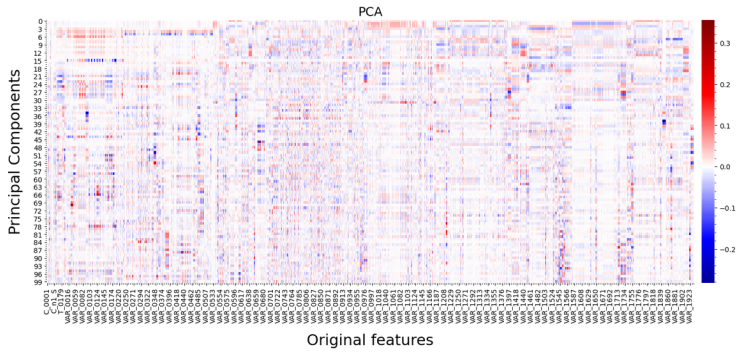

Figure 9. PC decomposition in the original feature basis (only the first 100 PCs are shown on the vertical axis).

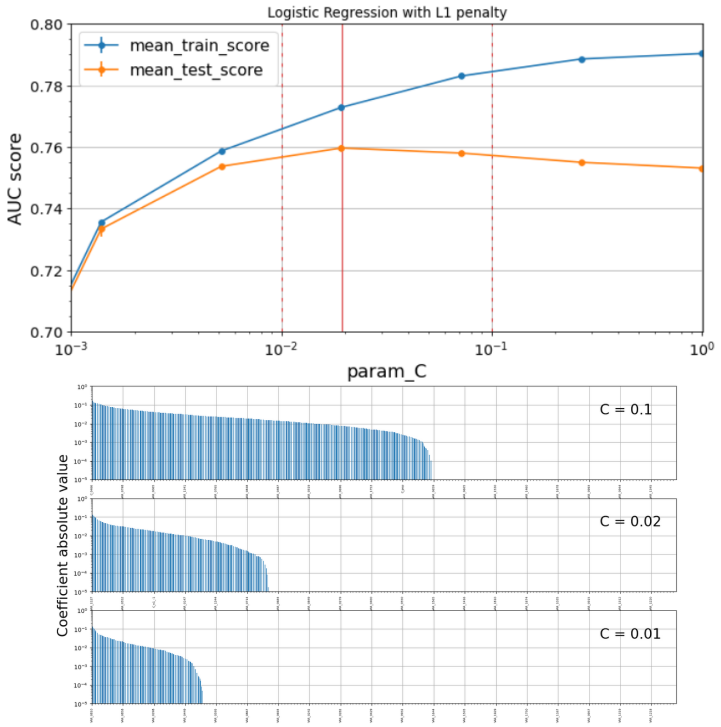

Figure 10. Score as a function of the L1 penalty parameter C (top) and feature coefficients (bottom) for the LR model.

Random Forest model drops from 0.76 to 0.67, while its run time increases by a factor of 5. We attribute this behavior to strong correlations between PCs, as show in Fig. 9. Each PC is a combination of almost all the original features (non-zero coeficients on the z axis), which affects discriminative power of models. This could be improved by using a Sparse PCA method, but we decide to drop this approach and try other solutions.

5.3 Logistic Regression with L1 Regularization

A regularization by adding the L1 penalty to the logistic regression (LR) model offers a way to prevent overfitting and remove redundant features. The magnitude of the penatly is controlled by the parameter C. Fig. 10 (top) shows the results of the fit to the data with the LR model, as a function of C. The blue (yellow) curve corresponds to the roc_auc score obtained for the train (test) set. In the test data, the maximum score of 0.767 is obtained for C=0.02 (the solid red line), The score varies very slowly around the maximum and drops to 0.765 for C=0.01 and 0.1 (dotted red lines). Fig. 10 (bottom) shows distributions of absoluate values of feature coefficients for fits with C=0.01, 0.02, and 0.1 (from top to bottom). Entries with 0 correspond to features removed by the model. The number of non-zero feaures amounts to 1096, 569, and 359, respectively. We select the latter two for the modeling in the next step.

5.4 Recursive Feature Elimination (RFE)

RFE is an interative procedure in which at each step a model is evaluated after removing n least important features, until

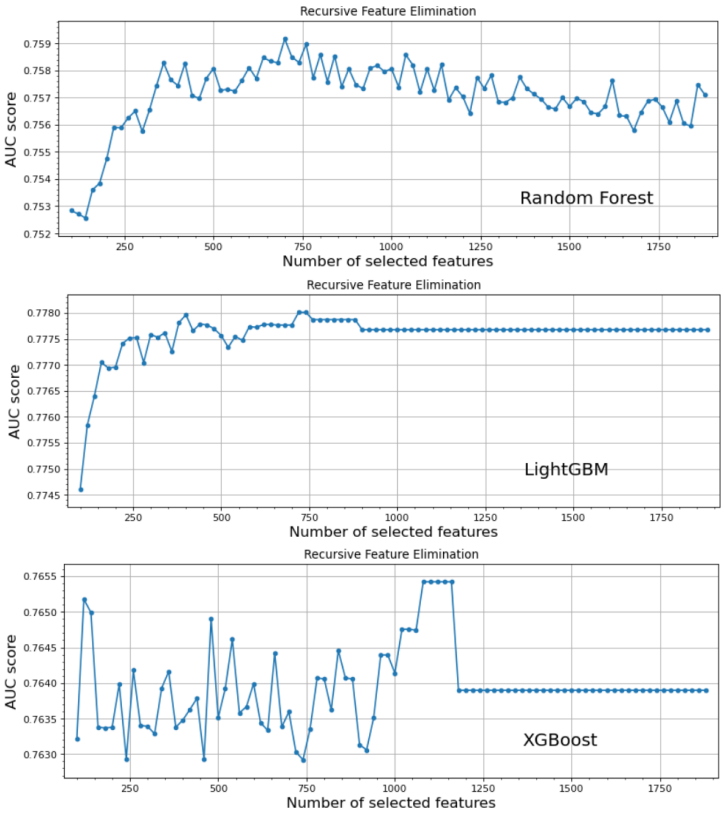

Figure 11. RFE with the Random Forest, LightGBM, and XGBoost models.

reaching minimal number of features, n_min. We apply RFE to three models: Random Forest, LighGBM and XGBoost, with default values of hyperparameters. The step parameter n is set to 20, and n_min to 100. Models are evaluated using the roc_auc metric and the two-fold cross-validation technique. The results are presented in Fig. 11, which shows that the maximal score is obtained when the number of selected features is 700, 360, and 1080 for Random Forest, LightGBM, and XGBoost models, respectively. For LighGBM, two regions obtain maximal score. To access the one with the smaller number of features, we repeat the RFE procedure 3 times.

Tab. 3 summarizes methods used to reduce the dimensionality of the dataset. Dots indicate those that will be used for modeling (less than 1,000 features). Notice that out of 360 features selected by both *LR, L-0.01* and *RFE, LGBM* models, only 150 (41%) are common.

	Method	N. of columns
•	(none)	1880
	LR, L1-0.1	1096
•	LR, L1-0.02	569
•	LR, L1-0.01	359
	RFE, XGB	1080
•	RFE, RF	700
•	RFE, LGBM	360

Table 3. Dimensionality reduction, summary.

6. Modeling

The customer response is modeled using Decision-Tree-based ensemble methods. We check the performance of the Random Forest classifier and two Gradient Boosting models from the scikit-learn library, as well as XGBoost and LightGBM. The baseline model of our choice is Logistic Regression. We also test a model based on the neural network. Model performances are evaluated using the area-under-the-ROC-curve (auc_roc) score.

The hyperparameter tuning for ensemble models is conducted in three steps. First, we test the performance of the models with their default settings. Then, we make one- and two-dimensional parameter scans using the scikit-learn GridSearchCV method, to identify those parameters and their ranges that maximize the score function. Finally, we perform a simultaneous multi-parameter Bayesian optimization using the scikit-optimize BayesSearchCV method.

6.1 Logistic Regression Classifier (Baseline)
6.2 Random Forest Classifier
6.3 Gradient Boosting Classifiers

Model	roc_auc	Acc.	Pre.	Rec.	time (min.)
GB	0.7777	0.794	0.621	0.293	22.40
HGB	0.7769	0.795	0.629	0.285	1.0
XGB	0.7785	0.794	0.627	0.284	2.2
XGBh	0.7791	0.795	0.628	0.286	1.1
LGBM	0.7783	0.795	0.629	0.287	0.5

Table 4. Gradient Boosting models, default parameters, full dataset.

6.3.1 XGBoost Model
6.3.2 LightGBM Model
6.4 Neural-Network Based Classifier
6.5 Results

The best result for LGBM on the RFE,LGBM sample (360 features) after BO. Mention that it gives 15th place on Kaggle's private leaderboard for this competition.

6.5.1 Confusion Matrix
6.5.2 Feature Importance

7. Summary

Hyperparameter set	RF score	RF ratio	XGB score	XGB ratio	LGBM score	LGBM ratio
Default (D)	0.753	1.00	0.7792	1.00	0.7783	1.00
Manual Tune (MT)	0.779	1.03	0.7868	1.01	0.7884	1.01
Bayesian Opt. (BO)			0.7967	1.02	0.7985	1.03
D, ratio to RF	1.000		1.035		1.034	
MT, ratio to RF	1.000		1.010		1.012	
BO, ratio to XGB			1.000		1.002	

Table 5. Parameter optimization for RF, XGB and LGBM models (full dataset).

Dataset	(n. cols)	auc_roc	ratio to Full
Full	(1880)	0.7985	1.000
LR, L1-0.02	(569)	0.7962	0.997
LR, L1-0.01	(359)	0.7959	0.997
RFE, LGBM	(360)	0.7998	1.002
RFE, RF	(700)	0.7971	0.998

Table 6. LightGBM with Bayesian optimization for reduced datasets defined in Tab. 3.

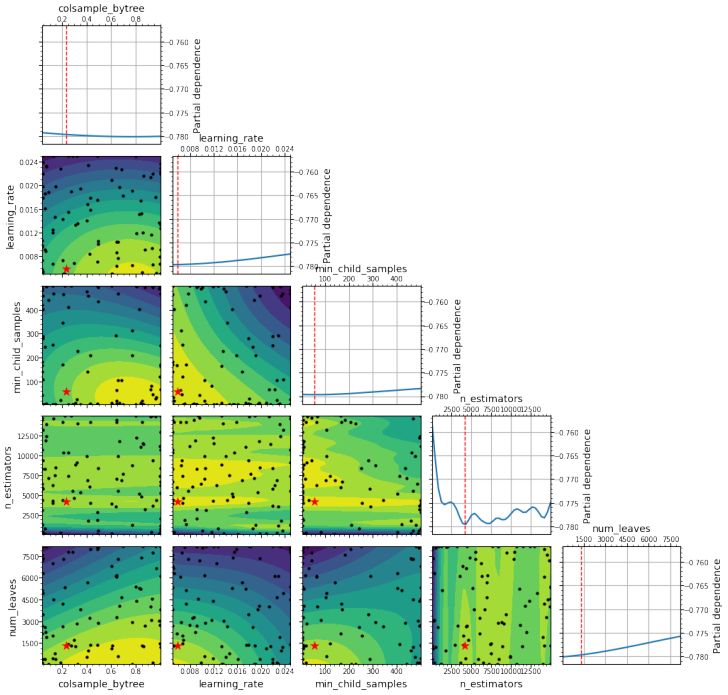

Figure 12. Decrease of ticket price (left) and revenue (right) for different number of closed runs.

	Model	auc_roc
0	NN3_360_360_360_1	0.7738
1	NN3_360_180_90_1	0.7742
2	NN3_360_720_180_1	0.7737
3	NN3_360_90_45_1	0.7728
4	NN4_360_180_90_45_1	0.7740
5	NN3_1800_940_470_1	0.7704

Table 7. NN models.

GradientBoosting (GB)	XGBoost (XGB)	LightGBM (LGB)	HistGradientBoosting (HGB)
learning_rate = 0.1	learning_rate = 0.3	learning_rate = 0.1	learning_rate = 0.1
n_estimators = 100	n_estimators = 100	n_estimators = 100 (num_iterations)	max_iter = 100
max_depth = 3	max_depth = 6	max_depth = None	max_depth = None
min_samples_split = 2	-	-	-
mn_samples_leaf = 1	min_child_weight = 1	min_child_samples = 20 (min_data_in_leaf)	min_samples_leaf = 20
max_leaf_nodes = None	-	num_leaves = 31	max_leaf_nodes = 31
-	colsample_bytree = 1.	colsample_bytree = 1. (feature_fraction)	-
max_features = 1.	colsample_bynode = 1.	feature_fraction_bynode = 1.	-
subsample = 1.	subsample = 1.	subsample = 1. (bagging_fraction)	-
-	gamma = 1.	-	-
-	reg_lambda = 0.	reg_lambda = 0. (lambda_l1)	-
-	reg_alpha = 1.	reg_alpha = 0. (lambda_l2)	l2_regularization = 0.
-	max_bin = 256 tree_method = hist	max_bin = 255	max_bins = 255 (<=255)
-	n_jobs = -1	n_jobs = -1	n_jobs = -1

Table 8. Default hyperparemeter values of gradient boosting models.