Advanced Machine Learning Project 2

Tymoteusz Kwieciński
01171244@pw.edu.pl
320637

Mateusz Nizwantowski
01161932@pw.edu.pl
313839

Summer Semester 2024

1 Introduction

As a second project in the Advanced Machine Learning course, our goal was to earn as much money as possible (typical). Our task was to identify customers who will use company marketing offers. However, there is a catch: each feature that describes the customer costs 200$, and for each predicted correctly customer, we get 10$. So, we had to balance out a number of features we were using with the accuracy of our predictor.

2 Evaluation methodology

2.1 Dataset split

We divided the given set of observations into two subsets: the training set and the validation set.

Dataset	Number of observations	Factor of positive labels
Original dataset (training+validation)	5000	0.4992
training	4000	0.49925
validation	1000	0.499

Table 1: Train-validation datasets sizes

We evaluated all of our experiments on the training set using 5-fold cross-validation. There was no need for stratification since the dataset is well balanced. The validation dataset was used only to asses models and methods' performance in the report. We did not use it to choose the optimal strategy to avoid data leakage.

2.2 Metric

We defined our metric to evaluate the performance of classifiers and feature selection. Let us denote $n \times k$ matrix features as X, true values as y, and the model's predicted probabilities as y_{prob}. Let us assume that observations are sorted with decreasing order of y_{prob} for easier notation. Then:

$$\hat{y}_i = \begin{cases} 1 & \text{if } i \le \frac{1}{5}n \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Then our score is defined as:

$$\text{score} = \left[\sum_{i=1}^{n} \mathbb{1}_{\hat{y}_i = y_i} \cdot 10\hat{y}_i\right] \frac{5000}{n} - 200 \cdot k \tag{2}$$

The metric chooses only $\frac{1}{5}$ observations with the highest probabilities because our objective is to select only 1000 observations from 5000 ones. Moreover, we scale our score using the factor $\frac{5000}{n}$ to make the results comparable to those we are expecting on the final test set (this is the metric we are trying to maximize).

3 Methods

Since our classifier models need to operate on a strictly defined set of features, we decided to divide our solutions into two separate parts: selecting columns and training models. We believe that doing it this way, instead of everything at once, makes managing all the variables more straightforward - we found out that combining our custom metric with various models is quite a complex task in some cases. We tested a substantial amount of various methods. Below, we list the ones that gave the best results and were most carefully evaluated:

1. Feature selection methods:
    - Boruta
    - mRMR - Minimum Redundancy Maximum Relevance
    - Nearest Shrunken Centroids
2. Classifiers:
    - XGBoost
    - Random Forest
    - SVM

Additionally, using all of the pretrained classifiers, we tried to create ensembles of the three models.

3.1 Feature selection methods

3.1.1 Boruta

We used an implementation from package `boruta_py` [1], which offers quite a simple method that is very friendly for a user. This method selected from 8 to 18 features depending on the hyperparameters of the Random Forest passed for the Boruta method. Similar ones were chosen using the random forest mean decrease in impurity and perturbation importance, which is a sign that our method worked adequately. The selected features were: `[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 100, 101, 102, 103, 104, 105]`.

3.1.2 mRMR

We used a python package called `mRMR` [3], which allowed us to calculate feature importance for different relevance methods and different numbers of chosen features. Using three different measures - *random forest*, *F-statistic*, and *Kologomorov-Smirnov*, we calculated which features were most commonly occurring in the selected subsets. Features selected by mRMR method: `[100, 102, 105, 403, 466]`.

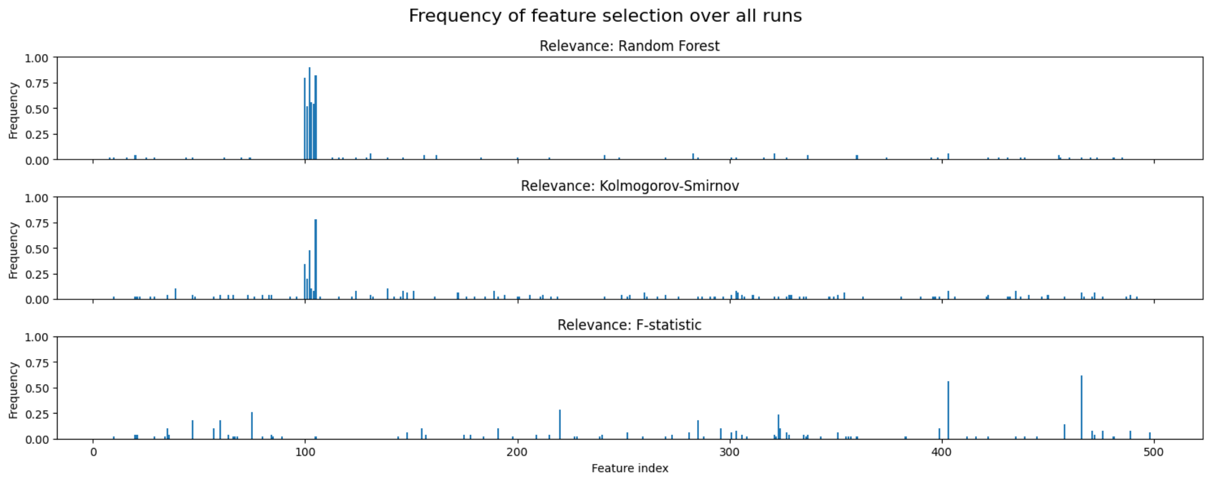

Figure 1: Frequency of different features in the subset selected by mRMR method

3.1.3 Nearest Shrunken Centroids

We used the implementation from [2]. Unfortunately, the model achieved very poor performance (very close to scores achievable by the dummy classifier). We decided to include its results since the features selected by this method were quite different from the previous ones and could potentially hold some new information. What's interesting 2 selected features were the same which were selected by the mRMR method. Features selected by NSC method: `[403, 458, 466, 489, 498]`.

3.2 Classifiers

Based on the set of features selected using previous methods, we used three different classifiers from the `sklearn` and `xgboost` libraries. We performed a grid search over the hyperparameters of the model and over all the feature subsets. The results are in the table 2. After that, using hand-picked, smaller subset of all possible collumns combinations, which we found out may give the best results, we run grid search again, but now with only fewer combination of columns.

3.2.1 Random Forest

It was the initial model we worked with. Its low sensitivity to non-optimal hyperparameters compared to XGBoost makes it an ideal model for prototyping and testing various theories. Random forest gave reasonable results, performing better than SVM but worse than XGBoost, at first, but at the end it was our worst classifier.

3.2.2 SVM

SVM is the model that at the end performed the best on Cross-validation as well as on validation. Fine tuning gave it a huge performance boost and it managed to beat both XGBoost and random forest.

3.2.3 XGBoost

We have chosen XGBoost as one of our models in the first place to leverage the possibility of customizing the loss of the XGBoost model to fit into our objective described above. Unfortunately, this attempt failed. Since we cared the most about the TP classes and wanted to eliminate FP, we used the *scale_pos_weight* parameter, which puts more or less weight on the positive class, but this hasn't changed the model's score by a lot.

3.2.4 First grid search results

Results of the grid search over hyperparameters and columns are in the 2. What is interesting is that the columns chosen by the grid search were quite consistent across the models. All of the classifiers trained on the five features selected by the *mRMR* selected used in the end columns `[100, 102, 105]`. Methods run on the *Boruta* subset always selected feature `100`, columns `1, 8` or `5` and `101, 105` or `103`. Details of our results, chosen hyperparameters, and implementation of our methods may be found on our GitHub repository.

Classifier	Feature selection method		
	Boruta	mRMR	NSC
Random Forest	6400	6650	4500
XGBoost	**6900**	**6900**	5400
SVM	6300	6650	4650

Table 2: Scores of different methods on validation set

3.2.5 Second grid search results

The second grid search resulted in the results, close to our predictions. It turned out that the models trained on columns [100, 102, 103, 105] or [100, 102, 105] were the best ones. The best models are in the 3.

Testing method	Method		
	SVM	XGBoost	Random Forest
Cross-validation	6863 ± 346	6750 ± 218	6725 ± 280
Validation set	6750	6450	6750
Used features	[100,102,103,105]	[100,102,103,105]	[100,102,105]

Table 3: Results of grid search over smaller subset of features

3.3 MDR - Multifactor Dimensionality Reduction

While searching for possible methods that could improve our solution, we stumbled upon an MDR. While other dimension reduction methods failed, MDR excelled. This method separates observations surprisingly well in contrast to PCA or t-SNE. We built a simple classifier based on the rule, which achieved suprisingly good performance. To leverage the information extracted using the MDR, we added the extracted features as on of the columns passed alongside the original features, which slightly improved performance of models trained with this column.

3.4 Ensemble

For the final submission, we decided to combine the results of our three best classifiers into weighted voting, but instead of using predictions as a vote, we used probabilities outputted by the models. Later on, we tried to create different types of ensembles and evaluated blending one as well. Unfortunately, all of this procedures turned out to simply average the results of the remaining classifiers and was not as groundbreaking as we expected.

4 Results

In order to achieve reproducibility, we created a notebook that builds all necessary prediction models and generates final outputs. We paid special attention to transparency in this file and averaged the results using multiple seeds to reduce the bias of the CV score estimator. The final scores we got from this file can be found in 4.

Testing method	Method				
	SVM	XGBoost	Random Forest	Averaging Ensemble	Blending Ensemble
Cross-validation	7136 ± 30	6975 ± 78	6658 ± 61	7026 ± 56	7062 ± 41
Validation set	7000	6850	6750	6850	6650

Table 4: Scores of different methods trained on dataset with features *[100, 102, 105]* with MDR

Our final submission contains predictions from the SVM model using *rbf* kernel trained on the whole dataset containing 5000 observations and features `100, 102, 105` along with artificial one coming from MDS.

References

[1] erikvdp. Implementation of *Boruta* algorithm in python. https://github.com/scikit-learn-contrib/boruta_py, 2023.

[2] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[3] Samuele Mazzanti. implementation of minimum redundancy - maximum relevance algorithm in python. https://github.com/smazzanti/mrmr, 2023.