Pázmány Péter Catholic University

Faculty of Information Technology and Bionics

Data Mining

REPORT

ITK DM&ML 2019

APS Failure at Scania Trucks

(KAGGLE COMPETITION)

STUDENT:

IULIIA ALEKSEENKO

ajulyav@gmail.com

Kaggle nickname:

Julia Ako

Introduction

This report presents the analysis and overview of the competition dataset, its main problems and features, the description of pre-processing steps, the ideas behind chosen algorithms used for classification, and the evaluation methods of performance.

Also, the report contains the main most accurate models (with look at the public and private results of the Kaggle competition). Those methods which were not covered in the classes are given the explanation.

Data overview

There are some problems with the given data which influence the result of classification.

Imbalanced classes – in the train data the number of false class samples are significantly higher than the true class samples (False – 55968, True – 1032). The following Figure represents this situation.

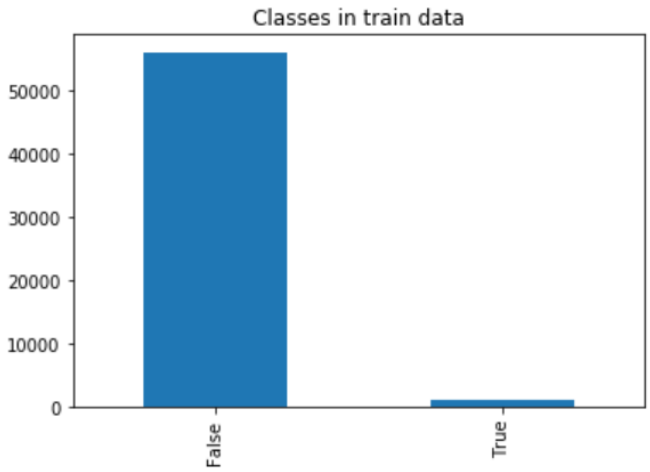

Figure 1 – Class distribution in the train data

Solution

Since this may lead to the problem when the model will be trained better to predict the False class much better than the True class, the two main ideas were tested (undersampling and oversampling). The undersampling was not chosen because we already have too little True class data, thus, if in total we have only 2064 instances to train on (1032 – true and 1032 – false after undersampling), then the model is underfitted.

Oversampling – generating new samples in the classes which are under-represented[1].

The following oversampling techniques were tested:

- Random – the most naive strategy is to generate new samples by randomly sampling with replacement the current available samples.
- SMOTE (Synthetic Minority Oversampling Technique) generates synthetic data based on the feature space similarities between existing minority instances (for this K-nearest is used) [2].

- ADASYN (Adaptive Synthetic Sampling Method) - ADASYN generates samples of the minority class according to their density distributions (also, K-nearest neighbors used) [3].

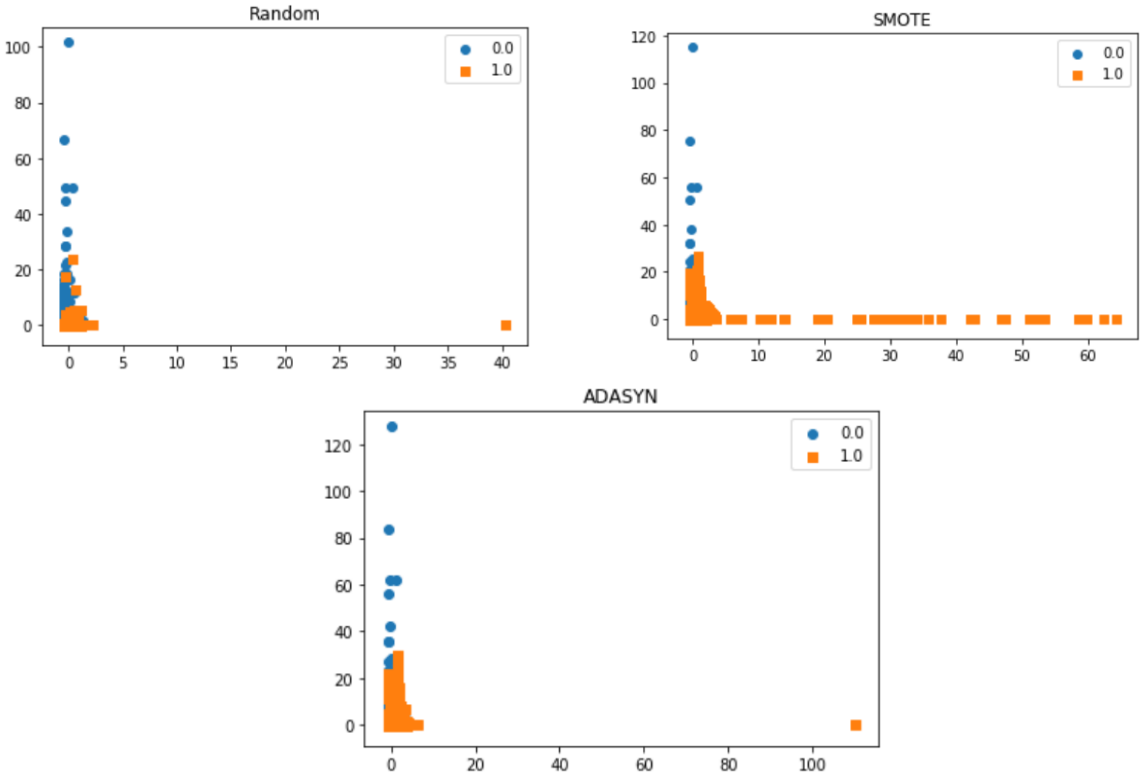

Figure 2 – Oversampling methods (after StandardScaler which will be described later)

Results:

XGboost classifier (will be described later) showed better performance with the SMOTE algorithm. ADASYN performed a little bit worse in this case, but Random oversampling lead to overfitting of the model in some cases (XGboost).

Missing values is the second biggest problem with this dataset. Train a model on the dataset with missing values can affect the model's quality and accuracy. Also, many algorithms cannot handle dataset if there are missing values (even oversampling step can be affected by this problem).

For example, in 'ab_000' (`data_train['ab_000'].isnull().sum()`) the total number of missing values is 43954 of 57000, 'ad_000' – 14275, 'af_000' – 2391.

Solution

- remove all columns where the percentage of missing values is greater than a threshold. This idea was refused later, because if in the train dataset this attribute can be deleted since does not give useful information for classification, it can turn out that in the test dataset this attribute has less missing values and can be really important factor to take into account;
- replace every missing value in both train and test datasets.

The following methods were tested:

- encode a missing value as a great negative number (for example, -9999). It will work for this data, because in all the data there are no negative numbers;
- impute the missing values with median values;
- impute the missing values with mean values.

	class	aa_000	ab_000
0	False	3256	-9999.0
1	False	3474	-9999.0
2	False	1332	-9999.0
3	False	519842	-9999.0
4	False	712	8.0
...	...	...	...
56995	False	8	0.0
56996	False	64084	-9999.0
56997	False	2670	-9999.0
56998	False	1406	-9999.0
56999	False	41464	-9999.0

-9999 encoding

	class	aa_000	ab_000
0	0.0	3256.0	0.0
1	0.0	3474.0	0.0
2	0.0	1332.0	0.0
3	0.0	519842.0	0.0
4	0.0	712.0	8.0
...	...	...	...
56995	0.0	8.0	0.0
56996	0.0	64084.0	0.0
56997	0.0	2670.0	0.0
56998	0.0	1406.0	0.0
56999	0.0	41464.0	0.0

Median imputing

	class	aa_000	ab_000
0	0.0	3256.0	0.734325
1	0.0	3474.0	0.734325
2	0.0	1332.0	0.734325
3	0.0	519842.0	0.734325
4	0.0	712.0	8.000000
...	...	...	...
56995	0.0	8.0	0.000000
56996	0.0	64084.0	0.734325
56997	0.0	2670.0	0.734325
56998	0.0	1406.0	0.734325
56999	0.0	41464.0	0.734325

Mean imputing

Figure 3 – Missing values replacement

Results:

- Encoding with a great negative value showed even better results than median imputing in some cases (Random Forest).

- Median impute was more powerful with XGboost, and the mean one lead to less classification performance.

Normalization is the next important step for this task. Since the data contains variables of different scales (for example, aa_000 varies much more than ab_000). Because the two columns differ in scale, they must be standardized to have a common scale when building a machine learning model.

StandardScaler from scikit-learn was used for this purpose (standardize features by removing the mean and scaling to unit variance) [4].

Performance evaluation

To evaluate the models, a confusion matrix was used in the series of experiments. The Figure below shows this matrix of one built model.

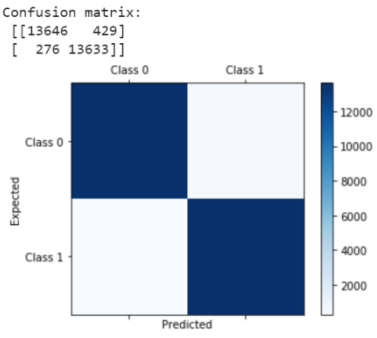

Figure 4 – Confusion matrix

It is easy to rely on this matrix when evaluating the model, because it shows not only the diagonal values indicating many correct predictions, but also the number of wrong predictions.

Also, it is important because we have the challenge metric (cost of false positive – 100, and cost of false negative – 3500). Thus, false negative misclassification will cost much more than false positive prediction.

Additionally, it was built a cost matrix for the most accurate models on confusion matrix (`cost_matrix = conf_mat[0,1]*100+conf_mat[1,0]*3500`).

Warning: In my first attempts I tried to rely on accuracy score as it is the most general way of evaluating a model, but it was a wrong solution which I realize later. In the situation with the imbalanced data, this metric will show a high accuracy even for the worst classifier because it always classifies the greatest class without considering and analyzing the attributes.

Models:

The main two models were tested during this competition:

- Random Forest Regressor with a threshold after regression;
- XGBoost model for classification.

The first method is not a trivial way of solving a classification problem with a regressor. Since it shows really promising results on the first submission, I worked on this idea and tried to find optimal threshold which gives the best result.

The second method is based gradient boosted decision trees with better results on speed and performance.

The next Table represents the best submissions of the Random Forest Regressor, pre-processing line and results on private and public data, cost matrix.

Preprocessing: **- 9999 encoding** **- normalization** Random Forest Regressor **(n_estimators=20, random_state=0)** **Threshold = 0** **Public result = -7.02105** **Private result = -6.01052** **cost_matrix = 85100**	**Preprocessing:** **- median imputing** **- normalization** Random Forest Regressor **(n_estimators=20, random_state=0)** **Threshold = 0** **Public result = -7.05263** **Private result = -6.91578** **cost matrix = 81300**	**Preprocessing:** **- 9999 encoding** **- normalization** Random Forest Regressor **(n_estimators=20, random_state=0)** **Threshold <=0.1** **Public result = -6.70526** **Private result = -7.68421** **cost matrix = 92100** **Threshold <0.1** **Public result = -6.73684** **Private result = -5.70526** **cost matrix = 88200**	**Preprocessing:** **- 9999 encoding** **- normalization** Random Forest Regressor **(n_estimators=20, random_state=0)** **Threshold < 0.2** **Public result = -8.49473** **Private result = -9.55789**

Thus, the best performance was achieved by the model with threshold<0.1 (9999 encoding, normalization, Random Forest Regressor (n_estimators=20, random_state=0) although the cost matrix is not the minimum in this case.

The mean imputing did not work for this algorithm and showed lower results with different thresholds.

The next Table shows the results on XGBoost best submissions.

Preprocessing: **- median imputing** **- SMOTE** **- normalization** **XGBClassifier (random_state=40)** **Public result = -6.21052** **Private result = -7.93684** **cost_matrix = 487300**	**Preprocessing:** **- median imputing** **- SMOTE (ratio =1)** **- normalization** **XGBClassifier (random_state=40)** **Public result = -7.00000** **Private result = -7.57894** **cost_matrix = 1079000**

ADASYN and random oversampling both performed worse for XGBoost algorithm and showed higher cost matrixes. Also, the different imputing methods were tested, and the best one – median was chosen.

Results:

In summary, in my case the best model was Random Forest Regressor with a threshold <0.1 in the end (- 9999 encoding and normalization). It shows both good results on Kaggle results and one of the best cost matrixes.

Resources:

1 https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html

2 (1, 2) N. V. Chawla, K. W. Bowyer, L. O.Hall, W. P. Kegelmeyer, “SMOTE: synthetic minority over-sampling technique,” Journal of artificial intelligence research, 16, 321-357, 2002.

3 He, Haibo, Yang Bai, Edwardo A. Garcia, and Shutao Li. “ADASYN: Adaptive synthetic sampling approach for imbalanced learning,” In IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence), pp. 1322-1328, 2008.

4 https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html