Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 421–424
DOI: 10.15439/2022F299
ISSN 2300-5963 ACSIS, Vol. 30

Using gradient boosting trees to predict the costs of forwarding contracts

Sławomir Pioroński
Faculty of Mathematics and Computer Science
Adam Mickiewicz University
Uniwersytetu Poznańskiego 4 Street
61-614 Poznań, Poland
Email: slawomir.pioronski@amu.edu.pl

Tomasz Górecki
Faculty of Mathematics and Computer Science
Adam Mickiewicz University
Uniwersytetu Poznańskiego 4 Street
61-614 Poznań, Poland
Email: tomasz.gorecki@amu.edu.pl

***Abstract*—When selling goods abroad or bringing them into the country from foreign partners, we face the problem of delivery. The division of responsibilities related to this between the manufacturer and the recipient sometimes varies. In such a situation, it is reasonable to use the services of a forwarding company. Then a forwarding contract is concluded, which specifies the details of the service, but the most important issue remains the selection of its price. In this paper, we present results obtained using the LightGBM method on the forwarding contracts pricing challenge held as part of the FedCSIS 2022 conference.**

***Index Terms*—data mining competition, LightGBM, forwarding contracts**

I. Introduction

A FORWARDING contract is a contract in which one of its parties – the forwarder undertakes to perform various services related to transportation in the course of his own business, such as sending or receiving a shipment, and the other party – to pay remuneration in return. A good forwarder will not only efficiently organize transportation but will also help reduce the cost of the transaction. However, to remain price competitive and still make a profit, he must have an good tool for predicting the cost of executing such a contract.

To predict forwarding contract costs based on tabular data we can use various machine learning methods [1]. We decided to use a gradient boosting algorithm, especially LightGBM, because of its speed and accuracy.

The rest of this paper is structured as follows. We first present the related work and then we give a short description of FedCSIS 2022 challenge. In Section IV we describe the data processing steps. Section V contains the description of the model used in our experiment. Next section presents results. Finally, in Section VII we summarize the findings and discuss possible future work.

II. Related work

Gradient boosting is a machine learning technique, which produces a prediction model in the form of an ensemble of weak prediction models, mainly decision trees. It creates the model like other boosting methods do, but it generalizes them by allowing optimization of an arbitrary differentiable loss function. Gradient boosting was first presented in 1997 [2], and has been refined over the last decade. There are many different implementations:

- XGBoost – an algorithm written by Tianqi Chen [3]. Probably the best known and most used implementation.
- LightGBM – Microsoft's algorithm [4].
- Catboost – an algorithm by the Russian company Yandex, designed to deal with categorical data [5].

LightGBM has many of XGBoost's advantages, including sparse optimization, parallel training, multiple loss functions, regularization, bagging, and early stopping. A main difference between the two lies in the construction of trees. LightGBM does not grow a tree level-wise – row by row. Instead it grows trees leaf-wise. Besides, LightGBM does not use the sorted-based decision tree learning algorithm as XGBoost. Instead, it implements a highly optimized histogram-based decision tree learning algorithm, which yields great advantages on both efficiency and memory consumption. The LightGBM algorithm utilizes two novel techniques called Gradient-Based One-Side Sampling and Exclusive Feature Bundling which allow the algorithm to run significantly faster while maintaining a high level of accuracy.

We can find many examples of LightGBM being used in machine learning competitions [6].

III. FedCSIS 2022 challenge

A. Data

The available training data set [7] contains a five-year history of contracts accepted by a Polish company. It consists of two main tables. The first one contains basic information about the contracts (36 features), and the second one describes the main sections of the planned routes associated with each contract (60 features). In the train set, we have 330 055 contracts and in the test set, we have 72 452 contracts. In addition, participants had an additional table (4 features) containing historical wholesale fuel prices.

B. Task

The theme of the competition was to forecast the costs associated with the execution of forwarding contracts, based on data from contracts and planned routes. The goal of the competition was to prepare and develop a model that forecasts the costs of individual orders as accurately as possible.

C. Evaluation

The solutions were assessed by the root mean square error:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2},$$

where y_i is an observed value, $\hat{y}_i$ is a predicted value and n is the number of data records in the data set.

Initial scores were evaluated via the KnowledgePit online platform [8] and published on a challenge leaderboard calculated on a small subset of the test set fixed for all participants. The final score was published after the challenge using the remainder of the test data set.

IV. Data processing

Data from both main tables were used for training. Since the data in the second table contains at least 2 rows for each contract (one row for each route step), it needs to be processed accordingly. In this case, the choice was made to include information for step two of the planned route with each contract. In addition, average monthly fuel prices are added for each contract, calculated from an additional third table, based on the month the route began. This resulted in a training data size of $330\ 055 \times 98$. Descriptions of each column can be found on the competition page [8].

A. Adding new features

Intuitively, an important factor affecting the price of the contract is the route duration, so a new column was created: hours_diff – the difference between route_end_datetime and route_start_datetime expressed in hours. The natural logarithms (with a lower bound equal to -1) of the following features were then created: hours_diff, km_total, km_nonempty, km_empty, train_km. Pearson correlation coefficients increased significantly for the first three characteristics listed. For example, for hours_diff it is 0.63, but for its logarithmic version, we get 0.89.

The numerical columns train_intervals and ferry_intervals were transformed into categorical variables with values of "0", "1", and "2+", which correspond to their numerical values.

At this point, it is worth mentioning that we used the Microsoft's FLAML library [9] (version 1.0.1). It contains many facilities, one of which is the automatic generation of the following numeric features for each datetime feature: year, month, day, minute, second, day of the week, day of the year, and quarter.

B. Repairing route datetime data

Real-world data from companies typically contains human errors. This case is no different. By checking in how many cases route_start_datetime is later than route_end_datetime we get 47 (0.01%) samples in the training set and as many as 405 (0.56%) in the test set. We see that for the training set, this is a very small number of examples that could easily be removed. However, in the case of the test set, this could determine the outcome of the competition.

Unfortunately, we don't know what this data should look like. Nevertheless, in reviewing this data, there are two types of errors:

1) Dates are in the wrong order.
2) The month or day was entered incorrectly.

The route_start_datetime and route_end_datetime are based on the estimated_time column from the second table, which contains information about the planned time of arrival at a given route step point, so it was the values from this column that were corrected. This was done with a simple script and then verified manually. Incorrect datetimes typically occurred for the first 2 or 3 steps within a given contract and were the same with date accuracy (no time information). For each contract, initial datetimes equal in date were selected. For simplicity, we will use a single date. In the first case the difference between the selected date with the last date was checked, if they were less than 14 days then selected datetimes were moved to the last places. If a type 1 error was not detected, the presence of a type 2 error was checked. Here we compared the number of days for the selected date and the next date after it (let's call it a comparison date). Assuming we have 1 after 31 and each month has 31 days, then if all we had to do was increase the number of days of the selected date by a maximum of 7 to get the number of days of the comparison date, we set the month and year of the selected datetimes to the largest possible so as not to exceed the comparison date. Any other instances, e.g., incorrectly entered days, were corrected manually through individual decisions.

C. Deleting some data

Often, some of the data we have is unnecessary and may even degrade the performance of the model. By comparing the values of the id_currency column, we can see that the training set has 6 unique additional values than the test set. Similarly, for the step_type feature from the second table, we obtain one unique redundant value in the training set. The temperature column, which reports the temperature level in a refrigerated trailer, was removed due to the high percentage of missing values (89% in both the training and test sets) and the way the values in it were recorded. This is string-type data with individual temperatures, temperature ranges, or additional information about the cooling mode. We have 760 unique values in the training set, 194 in the test set, and 99 values common to both sets.

After removing such contracts from the training set, and after removing the id_contract and temperature columns, we obtain a data set of size $329\ 349 \times 102$. These operations did not result in a large reduction of the data set, as the number of rows decreased by only 0.21%, but we still got rid of unnecessary data.

FLAML automatically discards features with constant values during training. In this case, we had 5 such features, all from the second table: step, train, train_km, train_line, tail_fin.

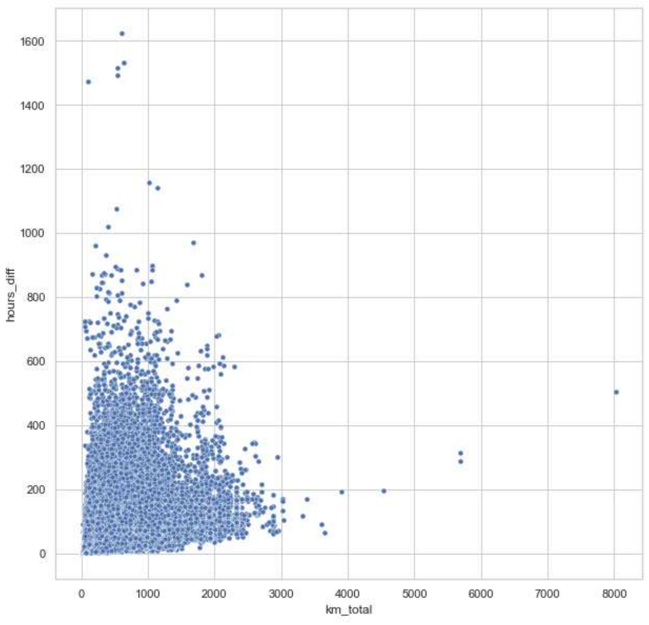

Fig. 1. Relationship between km_total and hours_diff (test set).

V. Model

A. Training strategy

LightGBM models, among other things, automatically perform feature selection and handle missing values. So what remains is the selection of appropriate hyperparameters. The FLAML library has solutions for this purpose. It offers two methods to optimize hyperparameters: CFO [10] and BlendSearch [11]. Both methods require a low-cost initial point from which the search begins if such a point exists. In the case of the former, the search gradually moves toward higher-cost regions if needed. It is a local search. The latter method combines the local with global search, i.e., it starts checking new starting points before the local search reaches full convergence. We used the "auto" mode to select the method, which should result in the selection of the CFO.

B. Output scaling

Since the training set contained data from 2016-01 to 2020-10 and the test set from 2020-09 to 2021-11, we decided to scale the model output. Outputs for the 2020 test data were multiplied by 0.998, while outputs for subsequent months of 2021 were multiplied by 1.001, 1.002, up to 1.011.

A measure like RMSE is sensitive to single, large errors. Thus, model errors for outliers can significantly degrade the final result, and the occurrence of outliers can be easily observed in Fig. 1. Hence, the idea was born to additionally scale values for samples that may be outliers. Thus, in the next step, for the selected points suspected of being outliers, the corresponding model results were multiplied by 1.01.

C. Outlier detection

Some of the most popular methods for finding outliers are isolation forest [12] and local outlier factor [13]. An isolation forest was chosen because of its shorter operating time.

In this case, it is necessary to pass only numeric type features to the model. The datatime type features have been removed and the corresponding year, quarter, month, and day of week features have been added in their place. In addition, the data set was expanded to include values for step one and last step from the second table (in some cases step two = last step). Next, columns with constant values and categorical features with more than 5% of missing values were removed. In other cases, they were imputed with the most frequent value. 2 of them were two-valued, so they were mapped to binary values. The Target Encoder was then fitted on the training set, i.e. the average target value for each class was taken.

The popular open-source machine learning library scikit-learn [14] (version 1.0.2) was used to train the model (with default hyperparameters). Mentioning this is important for the interpretation of the results since in this implementation the scores returned by the isolation forest can take negative values.

For points for which the isolation forest score was less than -0.05, were from 2021, and hours_diff was greater than 50 the final values were increased an additional 1%.

VI. Experimental results

A. Hyperparameter tuning

The final solution was created using an Intel Core i5-8300H processor and 16 GB of RAM (DDR4, 2400 MHz) on Microsoft Windows 10. It was found after 1617 seconds, with the search time set at 1800 seconds. During this time, the model scored 12 times better on the validation set (10% of the training set). To evaluate the quality of the model, we used the same measure as in the competition, the RMSE. After 33 seconds, we reached an RMSE of 0.1438, and the final model obtained an RMSE of 0.1298. Increasing the search time to 2700-3600 seconds can achieve an RMSE of 0.1265, but such models gave very poor results on the public competition test set, for instance: RMSE = 0.1530.

B. Obtained model

The selected model consists of 5243 trees, with a maximum number of 509 leaves in a single tree, the exact values of all tuned hyperparameters [9] can be found in Table I. Unfortunately, due to the size of the trees, we cannot visualize them in a meaningful way, but for practical reasons, the importance of individual features during prediction is useful. For this purpose, we will use total gain [3]. The features with the highest values are shown in Fig. 2. Unsurprisingly, we see that the distance traveled and the route time have a very large impact on the final prediction result.

On the public part of the test set, the model achieved a score of 0.1458. Here we can see that correcting the dates in Section IV-B was the right thing to do, because without it we achieved a score of 0.1469.

TABLE I
FINAL HYPERPARAMETERS (TO FIVE DECIMAL PLACES).

Hyperparameter	Value
n_estimators	5243
num_leaves	509
min_child_samples	8
learning_rate	0.01079
log_max_bin	6
colsample_bytree	0.34362
reg_alpha	0.00198
reg_lambda	0.01343

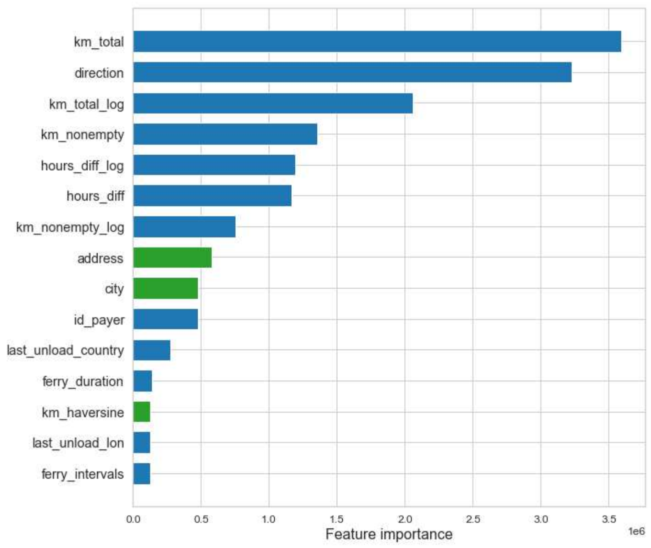

Fig. 2. The 15 features with the highest value of total gain. The features in the first table have been colored in blue, and those in the second in green.

C. Model output handling

The scaling idea was born out of the large discrepancy between the results on random subsets of the training data and the public portion of the test set. After using scaling of the model output described in Section V-B to the model from Section VI-B, our result improved. In this configuration, the RMSE on the public part of the test set is 0.1438. As can be seen, the intuition was correct, since linear dependencies with a precision of one month should be easily learned by the model, so they were not present in the training data. The reason may have been inflation was not openly recorded in the data.

Since such simple scaling does not exhaust the possibilities for improving the score, we used the isolation forest to look for values worth further improving. As a result, the RMSE changed from 0.1438 to 0.1434.

VII. CONCLUSIONS

This paper presents a powerful regression model that can deliver excellent predictions of the costs of forwarding contracts. We chose the LightGBM, because of its simplicity and speed. The model achieved the performance of the RMSE score of 0.1420 on a test set placing fourth (out of more than 50 teams that added a total of nearly 2 000 correctly formatted solutions) in the FedCSiS'2022 competition. Our model lost by only 2.606% to the best solution and lost only 0.9155% to third place. At the same time, it was better than the baseline model (baseline model placed fifth) by 3.873%. Worth adding, that the model got better final results than the results on the preliminary data set (0.1434). In addition, it is worth emphasizing that all calculations were performed in less than an hour on the average CPU.

The solution is pretty simple, so we have a lot of options here in terms of future work. For example, the topic of scaling model output is not exhausted. The idea of scaling outputs for points suspected of being outliers came up on the last day of the competition, so this was tested very briefly. So it is worth checking the outputs for other points suspected of being outliers and other multipliers. Taking it a step further, it is possible to see if the multiplier can be calculated for each point separately depending on some features.

REFERENCES

[1] T. Hastie, R. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*, ser. Springer Series in Statistics. Springer, 2009. ISBN 9780387848570
[2] L. Breiman, "Arcing the edge," 1997.
[3] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016. doi: 10.1145/2939672.2939785 pp. 785–794.
[4] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017.
[5] A. V. Dorogush, V. Ershov, and A. Gulin, "Catboost: gradient boosting with categorical features support," *ArXiv*, vol. abs/1810.11363, 2018. doi: 10.48550/ARXIV.1810.11363
[6] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "M5 accuracy competition: Results, findings, and conclusions," *International Journal of Forecasting*, 2022. doi: 10.1016/j.ijforecast.2021.11.013
[7] A. Janusz, A. Jamiołkowski, and M. Okulewicz, "Predicting the costs of forwarding contracts: Analysis of data mining competition results," in *Proceedings of the 17th Conference on Computer Science and Intelligence Systems, FedCSIS 2022, Sofia, Bulgaria, September 4-7, 2022*. IEEE, 2022.
[8] "Fedcsis 2022 challenge: Predicting the costs of forwarding contracts," https://knowledgepit.ml/fedcsis-2022-challenge/, accessed: 2022-06-20.
[9] C. Wang, Q. Wu, M. Weimer, and E. Zhu, "Flaml: A fast and lightweight automl library," in *MLSys*, 2021.
[10] Q. Wu, C. Wang, and S. Huang, "Frugal optimization for cost-related hyperparameters," in *AAAI'21*, 2021.
[11] C. Wang, Q. Wu, S. Huang, and A. Saied, "Economical hyperparameter optimization with blended search strategy," in *ICLR'21*, 2021.
[12] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*, 2008. doi: 10.1109/ICDM.2008.17 pp. 413–422.
[13] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '00. New York, NY, USA: Association for Computing Machinery, 2000. doi: 10.1145/342009.335388 pp. 93—104.
[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.