Proceedings of the of the 17th Conference on Computer Science and Intelligence Systems pp. 417–420
DOI: 10.15439/2022F294
ISSN 2300-5963 ACSIS, Vol. 30

XGBoost meets TabNet in Predicting the Costs of Forwarding Contracts

Aleksandra Lewandowska
Silesian University of Technology, Poland
aleklew480@student.polsl.pl

***Abstract*—XGBoost and other gradient boosting frameworks are usually the default choice for solving classification and regression problems for tabular data, especially in data science competitions, as they often, combined with proper data pre-processing and feature engineering, supply high accuracy of predictions. They are also fast to learn, easy to tune, and can supply a ranking of variables, making interpretation of learned models easier. On the other hand, deep networks are the top choice for complex data, such as text, audio, or images. However, despite the many successful applications of deep networks, they are not yet prevalent on tabular ones. It may be related to difficulties in the choice of the proper architecture and its parameters. A solution to this problem may be found in recent works on deep architectures dedicated to tabular data, such as TabNet, which has recently been reported to achieve comparable or even better accuracy than XGBoost on some tabular datasets. In this paper, we compare XGBoost with TabNet in the context of the FedCSIS 2022 challenge, aimed at predicting forwarding contracts based on contract data and planned routes. The data has a typical tabular form, described by a multidimensional vector of numeric and nominal features. Of particular interest is investigating whether aggregation of predictions derived from XGBoost and TabNet could produce better results than either algorithm alone. The paper discusses the competition solution and shows some added experiments comparing XGBoost with TabNet on competition data, including incremental model re-building and parameter tuning. The experiments showed that the XGBoost and TabNet ensemble is a promising solution for building predictive models for tabular data. In the tests conducted, such an ensemble achieved a lower prediction error than each of the algorithms individually.**

I. Introduction

ALMOST every company collects data on their products, services, or customers. Analyzing such data helps companies make informed business decisions to increase their profits. One example of such analysis is the task defined in the FedCSIS 2022 challenge [1], involving cost prediction of forwarding contracts based on contract details and planned routes. In that case, such cost estimation may support selecting the most profitable contracts.

Many examples of predictive analytics for supporting companies in increasing or maintaining their profits can be found [2]. Recommender systems [3], [4] may suggest to customers what products they may be interested in, thereby increasing the chances of buying a product. Predicting customer lifetime value [5] helps to select the top customers to whom more attention should be paid. Churn prediction [6], [7] can reduce customer losses.

Data collected by companies about customers, contracts, or products are often stored in database tables consisting of various attributes (nominal, numerical, date-time), thus their natural form for analytics is tabular, with mixed types of features. Over the past several years, gradient boosting frameworks have become particularly popular for analyzing such data, as combined with appropriate data pre-processing and feature engineering, they often achieve superior accuracy. In contrast, deep learning methods are the default choice for unstructured data, such as images, audio, or text. Deep networks are not often applied to tabular data, which may be due to the difficulty of selecting an appropriate architecture suitable for analyzing heterogeneous tabular formats. Therefore, several deep architectures dedicated to tabular data have been proposed over the past few years, such as TabNet [8], Net-DNF [9], Node [10], and TabTransofmer [11].

This paper attempts to apply one of those tabular-specific deep networks, namely TabNet, to forwarding contracts data in the context of the FedCSIS 2022 challenge, and compare it with XGBoost [12], one of the most popular gradient boosting frameworks. Of particular interest is investigating whether combining the two methods could produce better results than either algorithm alone. We also investigate the performance of the algorithms over time, training them in an incremental fashion. Section II describes the competition solution. In this section are described: data pre-processing steps and modelling, XGBoost hyper-parameters tunning, Tabnet hyper-parameters tunning and linear combination of both models as a result to the problem. Section III summarizes the work.

II. Competition solution

A. Competition data-sets

Competition consists of five data-sets: **test data set (main table)**, **test data set (routes table)**, **training data set (main table)**, **training data set (routes table)** and **fuel prices**. Test and training data sets differ in the fact that test data set does not include the estimated cost column, which is used for evaluation purposes. The presented solution's data pre-processing process is divided into two separate steps: **routing table** and **main table** pre-processing.

B. Routing table data pre-processing

In the first step, we selected and added new features, and aggregated the data by contract id. We identified the data that was missing in a significant manner and in the process

of simplification we removed those features from further processing.

The percentage of missing values in particular columns is respectively about 54.5% and 30.6%. Therefore, the number of columns in this step has significantly dropped.

The columns describing respectively longitude and latitude were dropped as extracted information can be found in other features, like the distance between starting and ending points.

The subset of features that have not been dropped can be associated with different transport methods. The present methods are ferry, train, truck (which can be considered as conventional transportation method).

To each method we have defined added features, which measure how much has been transported over how long time or distance. Created features:

- Train (weight-distance) feature = $train\_km{*}kg\_current$
- Ferry (weight-time) feature = $ferry\_duration * kg\_current$
- Truck (weight-distance) feature = $km * kg\_current$

As multiple records are used to define a particular transportation process, we decided to aggregate the features left. After the pre-processing process the routing table is summarized with one record per each contract.

C. Main table data pre-processing

Main table includes 36 columns, and only 8 of them has missing values. Only 3 columns include more than 50% of missing values. Therefore, those columns have been dropped.

In the first step we have extracted date and time information from **route_start_datetime** and **route_end_datetime** columns. From those columns we have extracted **year**, **month**, **day of the week**, **hour**, **the difference between start and the end** of the transportation process and converted those two columns into columns having **unix time** (*which defines the number of epochs since 01-01-1970*).

In the next step we put our attention towards the processing of categorical data, which can be shown on the feature **id_payer**. From the training set we selected the payers that accounted for over 1000 orders and then we applied one hot encoding technique. All not accounted payers are categorized as others. The same approach was used for: **currencies**, **first load country**, **last unload country**. Other categorical features were treated in a different manner. As the number of unique values was small enough, instead of using one hot encoding technique we have decided to assign a numerical value to each of values included within each category. To each value we assigned an integer, in our case we did not concern ourselves with the order of assigned numerical values. The numerical values used are integers from 0 to n where n depends on the number of unique values present in a particular categorical column.

Features for which assignment was applied are: **contract type**, **load size type** and **direction**. Then newly created features and other non-categorical features were joined with the data set created in the earlier step.

The resulting data set constitutes the input for our XGBoost and TabNet models. The same steps were applied towards test data sets in both routing table and main table data sets.

D. TabNet - training and evaluation

TabNet is an "interpretable canonical deep tabular data learning architecture."[2] The parameters that influence the training process of the model are: **optimizer** and **learning rate**, **hyper-parameter tuning**, **training process**. We used **Adam optimizer** and **exponentially decreasing learning rate of first value 2e-2**.

After choice of the optimizer and the learning rate we could turn to **hyper-parameters tuning**. For the training process we selected 4 parameters for testing [8]:

- **n_a** - "Width of the decision prediction layer. Bigger values give more ability to the model with the risk of over-fitting."
- **n_b** - "Width of the attention embedding for each mask. According to the paper n_d=n_a is usually a good choice."
- **batch size** - "Number of examples per batch. Large batch sizes are recommended."
- **n_steps** - "Number of steps in the architecture (usually between 3 and 10)"

To select the most suitable parameters for our problem, we have used the technique called **Grid Search**. To use this technique, firstly we had to define the range of the parameters that we wanted to check during testing. Next, we would try all possible combinations, train the model and evaluate the performance Selected parameter ranges are: **n_a** $=$ **n_b** $\in \{4, 8, 18\}$, **n_steps** $\in \{3, 5, 7\}$, **batch size** $\in \{1024, 2048\}$, **mask type** $\in \{"sparsemax", "entmax"\}$. The table below represents a subset of results of all performed experiments. Model consistently has performed better for **mask type** = "sparsemax" and only results with such value are presented.

Width	Steps	Batch	RMSE Val	RMSE Pre
8	5	2048	0.1634	0.1719
8	5	1024	0.1608	0.1720
8	3	1024	0.1587	0.1734
8	7	1024	0.1757	0.1751
4	7	2048	0.1646	0.1752
8	3	2048	0.1618	0.1772
4	5	1024	0.1661	0.1774
8	7	2048	0.1665	0.1782
4	3	1024	0.1687	0.1779
4	3	2048	0.1667	0.1875
4	5	2048	0.1670	0.2387
4	7	1024	0.1647	0.2704

TABLE I
RESULTS OF HYPER-PARAMETER TESTING, TABNET

Width - n_a, n_b
Steps - n_steps
Batch - batch_size
RMSE Val - RMSE value, calculated on the validation set
RMSE Pre - RMSE value, calculated on the preliminary testing set

The training data set has been sorted first based on the unix epoch value of route_start_datetime column. The validation set consists of the last 10% of the sorted rows of the data set.

The sorting took place because we want our model that has been trained on the archived records to perform well on the incoming data. The main goal is to predict future values.

This exactly describes the training process that took place. In the first iteration we trained our model only on the first 10% of the data set and evaluated it on the following 10%. Therefore, only 20% has been used at this step. In the next iteration we have moved the 10% used for evaluation into the training set, on which model has been additionally trained. Then the following 10% has been used for evaluation purposes. This process was taking place incrementally, till the moment in which the model was trained on 90% of the data and evaluated on the last 10%. The RMSE value evaluated at this step is denoted as RMSE Val. However, for our model to perform well on the test data set, the model should be trained on almost all available records. For that reason, we have moved the following 8.5% from the local validation set used for evaluation and additionally trained the model, which was then evaluated on the last 1.5% of the ordered training data set. Then we considered a TabNet model trained on 98.5% of the data set as a fully trained model, which then was evaluated on the preliminary testing set, supplied by the FedCSIS 2022 competition, which is the subset of the full testing data set. The results are available in I. Hyper-parameters of the model that **performed best** are:
n_a = n_b = **8**, n_steps = **5**, batch size = **2048**, mask type = **"sparsemax"**

The results of evaluation are: local validation = **0.1634**, preliminary validation = **0.1719**, final result = **0.1713**.

The results obtained for the model with **default hyper-parameters** are: local validation = **0.1587**, preliminary validation = **0.1733**, final result = **0.1735**

We can see that the model has improved, but not in a significant manner. In comparison to the model which obtained the best results on the preliminary test data set, both models differ by the **batch size**, and **n_steps** which are respectively **1024** and **3** for the model trained with default hyper-parameters.

E. XGBoost - training and evaluation

XGBoost "is an optimized distributed gradient boosting library"[3]. XGBoost model consists of two steps: **training** and **evaluation**. Those processes are quite similar to the processes that were described in the TabNet section of this paper. The training process is similar to the TabNet training process with a subtle differences. Similarly we use **Grid Search** technique for finding the optimal hyper-parameters for our model. The hyper-parameters that were tuned are:

- **learning rate** - "Step size shrinkage used in update to prevents over-fitting." [12]
- **min split loss** - "Minimum loss reduction required to make a further partition on a leaf node of the tree."[12]
- **max depth** - "Maximum depth of a tree. Increasing this value will make the model more complex and more likely to over-fit."[12]
- **colsample by tree** - "The subsample ratio of columns when constructing each tree"[12]

The defined ranges of interest are: **learning rate** $\in \{0.05, 0.85, 0.15\}$, **min split loss** $\in \{0, 0.1, 0.2\}$, **max depth** $\in \{3, 4, 5\}$, **colsample by tree** $\in \{0.55, 0.65, 0.75\}$.

Rate	Loss	Depth	Colsample	RMSE Val	RMSE Pre
0.085	0.2	5	0.65	0.1511	0.1617
0.150	0.2	4	0.65	0.1555	0.1623
0.150	0	4	0.75	0.1561	0.1623
0.085	0.2	4	0.75	0.1536	0.1626
0.085	0.1	5	0.65	0.1519	0.1628
0.085	0.2	5	0.75	0.1523	0.1629
0.150	0.2	4	0.75	0.1523	0.1632
0.085	0.1	4	0.75	0.1536	0.1633
0.085	0.1	4	0.65	0.1527	0.1636
0.05	0.2	5	0.75	0.1527	0.1636
0.05	0.2	5	0.65	0.1523	0.1653

TABLE II
RESULTS OF HYPER-PARAMETER TESTING, XGBOOST

*Results were sorted by RMSE Pre value
Rate - learning rate
Loss - min split loss
Depth - max depth
Colsample - colsample by tree
RMSE Val - RMSE value, calculated on the validation set
RMSE Pre - RMSE value, calculated on the preliminary testing set

The training process is quite like TabNet's training process described earlier. In the first iteration XGBoost model is trained only on 10% of the training data, and then evaluated on the following 10%. In the next iteration XGBoost model is trained on the 20% of the training data set and then evaluated on the following 10%. This process takes place till the model is trained on 90% of the data set. Then the model is trained enough that we can calculate RMSE Val value, evaluating the model on the last 10% of the training data set. Afterwards XGBoost model is trained on 100% of the training data set, and then model is evaluated on the preliminary testing data set and full testing data set. It is worth mentioning that in each iteration like in the last stage of training (training on the 100% of the data set) model is basically trained from zero - created is new instance of the model, and the model gets trained. In comparison to TabNet that makes a significant difference as TabNet model uses warm training (with use of already pretrained weights as a baseline for the model training). The XGBoost model that performed best:

Type	Learning Rate	Loss	Depth	Colsample
Best	0.085	0.2	5	0.65
Default	0.3	0	6	1

TABLE III
HYPER-PARAMETERS - BEST AND DEFAULT MODELS XGBOOST

We can notice that the default model has performed significantly worse than tuned model.

F. Combination of XGBoost and TabNet models

The final model is a combination of XGBoost and TabNet models both.

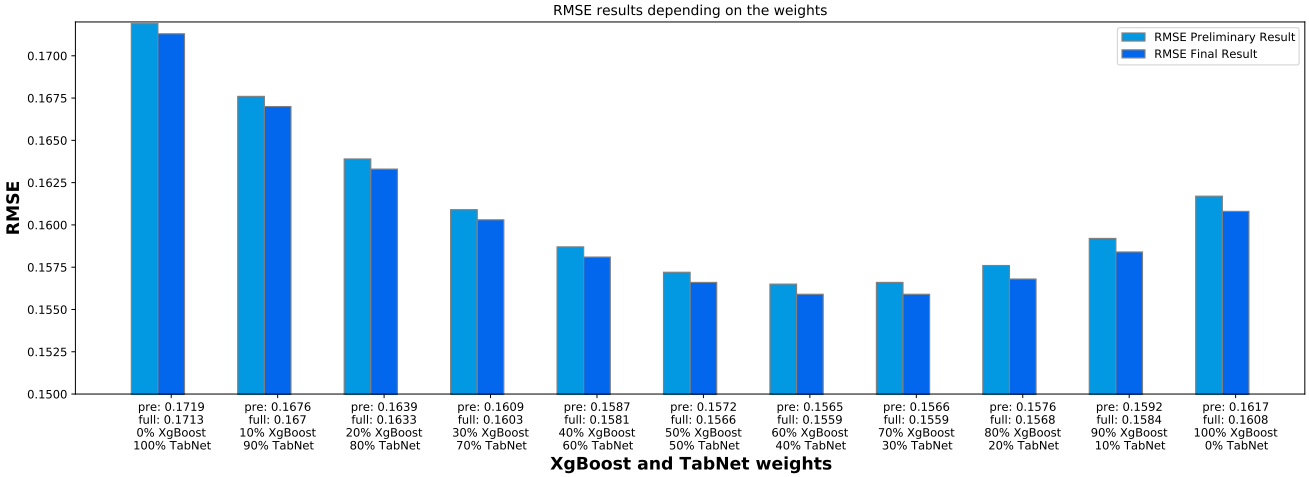

Fig. 1. Training results of XGBoost model depending on the used hyper-parameters

Type	Local Val	Preliminary Val	Final Result
Best	0.1511	0.1617	0.1608
Default	0.1739	0.2365	0.2362

TABLE IV
RESULTS - BEST AND DEFAULT MODELS XGBOOST

Results are generated independently for each one of the models. Then we use the assigned weights to generate the final results. The main problem at this stage was finding the right weights. For simplicity we have decided to check 10 different pairs. Starting with XGBoost being assigned 0 and TabNet 1, incrementing the weight assigned to the XGBoost by 0.1 and decreasing the weight assigned to TabNet by 0.1. Ultimately Fig. 1. depicts the preliminary and final evaluation of the generated results. We can notice that the best RMSE is generated for 0.6 assigned to XGBoost and 0.4 assigned to TabNet with final results of **0.1565** for preliminary validation and **0.1559** as a final result. It is worth noticing that XGBoost has significantly outperformed TabNet, with TabNet's results of **0.1719** and **0.1713** and XGBoost **0.1617** and **0.1608** as preliminary and final results respectively.

III. CONCLUSIONS

In this paper, we have introduced the approach that we have used for FedCSIS 2022 Challenge. We have described the data pre-processing, handling the missing data and two models that we have used for our solution: **XGBoost** and **TabNet**. It is also worth noticing that the process of training the XGBoost model takes much less time in comparison to the process of training the TabNet model. Although the result is worse than the baseline solution, we can notice how the combination of both models can outperform those two models working separately. Although XGBoost has significantly outperformed the TabNet model itself, the combination of both models has quite significantly outperformed both models working alone.

REFERENCES

[1] 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.

[2] E. W. Ngai, L. Xiu, and D. C. Chau, "Application of data mining techniques in customer relationship management: A literature review and classification," *Expert systems with applications*, vol. 36, no. 2, pp. 2592–2602, 2009. [Online]. Available: http://dx.doi.org/10.1016/j.eswa.2008.02.021

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2013.03.012

[4] I. Portugal, P. Alencar, and D. Cowan, "The use of machine learning algorithms in recommender systems: A systematic review," *Expert Systems with Applications*, vol. 97, pp. 205–227, 2018.

[5] S. Gupta, D. Hanssens, B. Hardie, W. Kahn, V. Kumar, N. Lin, N. Ravishanker, and S. Sriram, "Modeling customer lifetime value," *Journal of service research*, vol. 9, no. 2, pp. 139–155, 2006. [Online]. Available: http://dx.doi.org/10.1177/1094670506293810

[6] J. Ahn, J. Hwang, D. Kim, H. Choi, and S. Kang, "A survey on churn analysis in various business domains," *IEEE Access*, vol. 8, pp. 220 816–220 839, 2020. [Online]. Available: http://dx.doi.org/10.1109/ACCESS.2020.3042657

[7] D. L. García, À. Nebot, and A. Vellido, "Intelligent data analysis approaches to churn as a business problem: a survey," *Knowledge and Information Systems*, vol. 51, no. 3, pp. 719–774, 2017. [Online]. Available: http://dx.doi.org/10.1007/s10115-016-0995-z

[8] S. Ö. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 8, 2021, pp. 6679–6687.

[9] L. Katzir, G. Elidan, and R. El-Yaniv, "Net-dnf: Effective deep modeling of tabular data," in *International Conference on Learning Representations*, 2020.

[10] S. Popov, S. Morozov, and A. Babenko, "Neural oblivious decision ensembles for deep learning on tabular data," *arXiv preprint arXiv:1909.06312*, 2019.

[11] X. Huang, A. Khetan, M. Cvitkovic, and Z. Karnin, "Tabtransformer: Tabular data modeling using contextual embeddings," *arXiv preprint arXiv:2012.06678*, 2020.

[12] 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, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785