Comparing Transformer-based and gradient boosted decision tree (GBDT) Models on Tabular Data: A Rossmann Case Study

Coenraad Middel[1,2,3][0000−0002−4109−7457] and
Marelie H. Davel[1,2,3][0000−0003−3103−5858]

[1] Faculty of Engineering, North-West University, South Africa
[2] Centre for Artificial Intelligence Research, South Africa
[3] National Institute for Theoretical and Computational Sciences
cwmiddel@gmail.com

Abstract. Heterogeneous tabular data is a common and important data format. This empirical study investigates how the performance of deep transformer models compares against benchmark gradient boosting decision tree (GBDT) methods, the more typical modelling approach. All models are optimised using a Bayesian hyperparameter optimisation protocol, which provides a stronger comparison than the random grid search hyperparameter optimisation utilized in earlier work. Since feature skewness is typically handled differently for GBDT and transformer-based models, we investigate the effect of a pre-processing step that normalises feature distribution on the model comparison process. Our analysis is based on the Rossmann Store Sales dataset, a widely recognized benchmark for regression tasks.

Keywords: Tabular data · Transformer architectures · Gradient Boosting Decision Trees · Hyperparameter tuning · Rossmann Store Sales

1 Introduction

Almost all industries produce data that is stored in tabular format, with tabular data in general being the most common type of data structure [7]. Heterogeneous tabular data is traditionally modelled with GBDTs or random forests, with good results [19]. Some modern, novel deep learning models, employing the popular transformer architecture or attention mechanism, claim matching or superior performance [1,20,17] to the contemporary GBDT models. This remains a contentious issue with many factors influencing the performance and validity of comparisons between models.

In a large, recent study by McElfresh et al. [14], it was found that the performance improvements offered by such deep neural networks (DNNs) are negligible, especially when taking into account the considerable effort required when developing the DNN-based models. However, the hyperparameter (HP) tuning protocol used to develop models was restrictive (see Section 4.2) raising questions with respect to the method and results. In this study, we use a well-known

tabular benchmark and compare four algorithms using a more extensive HP tuning process. We compare two GBDT architectures and two transformer-based architectures on a regression task. Our goal is not to provide a definitive answer with regard to the comparative performance of these models, but rather to point out considerations to be taken into account when comparing such model architectures.

2 Background

There are several techniques for modelling tabular data. We focus on two families of methods: tree-ensemble methods, specifically GBDT models, because of their proven track record in modelling tabular data [15,8,6], and transformer-based models because of recent claims with regard to performance and feasibility [1,17,20].

The GBDT architecture is based on the idea of boosting residuals of weak learners, usually, decision trees, in an ensemble [10]. These models are popular solutions as their performance is considered the contemporary benchmark on heterogeneous tabular data [5,4,15]. The first model of interest, XGBoost is a highly scalable version of gradient boosting developed by Chen et al. [6], building on the work of Friedman et al. on gradient boosting [10] more than two decades ago. This technique is highly scalable as it can be parallelised and distributed over computers. It is used widely as one of the leading machine learning libraries [4] for regression, classification and other popular tasks. CatBoost is a more recent contribution to the GBDT model family and was developed by Yandex-based Dorogush et al. [18] A smart algorithm for built-in categorical feature processing is introduced, where XGBoost requires pre-processing, such as one-hot encoding, to deal with features of this nature. CatBoost ranks comparatively high, usually in the top three, in recent reviews comparing techniques on tabular data that we are aware of [14,12,5,11].

Transformers are popular for natural language tasks such as translation. The attention mechanism from Vaswani et al. [21] is versatile and can be used to augment DNN-based models to be used on tabular data. This is done by parsing the input variables to an embedding space that can be learned by a form of DNN. TabNet was developed by Arık et al. [1] and implements an attentive transformer mechanism in a way likened to that of GBDTs and the efficient way in which they calculate global feature importance but for instance-wise (or row-wise) feature selection. This sparse, instance-wise feature selection is learned from the data, to allow for the highlighting of significant features. It uses an uncommon sparsemax activation function to pass only certain features sequentially [13]. In contrast, the self-attention and intersample attention transformer (SAINT) model employs attention in two distinct ways [20]. The first is similar to that of TabNet, namely instance-wise attention, although it keeps the more commonly used softmax activation function for this step, not entirely blocking out features that are not

[4] https://xgboost.readthedocs.io/en/stable/index.html

deemed important for a given data sample, but highlighting salient features. The second form is called inter-sample attention and calculates the attention across a batch of samples after concatenating the embeddings. This enables column-wise attention. This is used where noisy features are present in a data sample and can be likened to a regularisation step.

Recent studies [14,12,19,5,11] concerned with deep neural networks and tabular data, compare them to modern GBDT with varying degrees of success. These studies agree that traditional GBDT architectures are still the more typically used method. However, papers proposing novel DNNs-based techniques for modelling tabular data claim architecture described therein is, at least on average, better than conventional machine learning techniques for tabular data [1,20,17]. On the other hand, McElfresh et al. [14] suggest that the comparison of model architecture performance is less important than the effort to optimise the HPs. We review the comparative studies and highlight the inconsistencies in their conclusions:

- McElfresh et al. is the most recent and largest comparative study, researching 176 datasets. Comparing specifically GBDT models and DNN model families, they conclude that GBDT is more performant, but by a small margin, usually not statistically significant [14]. They concluded that most of the algorithms have statistically similar performance (using $p<0.05$) with GBDT models most performant on datasets by less than 0.1%. Taking into consideration the difference in effort when training the models, the efficacy of DNNs comes into question.
- Grinsztajn et al. [12] use a repeatable benchmarking method to conclude that, beyond some specific cases, GBDT-based models tend to be more reliable and less computationally expensive. Included in this study are 45 tabular datasets from various domains with classification and regression tasks, three GBDT-based models and four DNN-based methods. The datasets are limited to, at most 50,000 samples with a strict filtering and selection regime. For HP optimisation, random grid search is preferred, although a section is dedicated to comparing it to the Bayesian optimisation implementation of Weights and Biases [3]. It is concluded that the optimisation does not seem to make a difference in the results, although the experiment is limited to only 200 random search iterations and reaches higher normalised test accuracy scores for all four models tested when using the Bayesian method.
- Borisov et al. [5] conclude that SAINT consistently outperforms classical GBDT-based models for very large datasets with more than approximately 1 million data points.
- While confirming that there is no obviously superior method even when compared to GBDT-based methods, Gorishniy et al. [11] identify a lack of effective baselines and research protocol as some of the main reasons why comparisons are inconclusive. The researchers compare an multilayer perceptron (MLP), ResNet-like model and a newly proposed transformer-based model with more classical approaches.

– Shwartz-Ziv et al. [19] conclude that XGBoost is superior to DNN-based architectures (neural oblivious decision ensemble (NODE) and TabNet, specifically) recreating comparisons from where these DNN-based models are first proposed. They also find that when the DNN-based models are used in conjunction with XGBoost as an ensemble the performance increases.

While the majority of research seems to indicate that GBDTs will outperform DNNs when considering the effort and computational expense for performance, McElfresh et al. [14] note that almost every algorithm they test is considered the best on some dataset while being the worst on another, as aligned with the 'no free lunch' theorem [22].

3 Experimental Setup

Comparing different architectures is challenging in that different HPs are applicable. In order to compare different architecture types fairly, the optimisation process should attempt to find the best combination of model, HPs, and dataset. This process can also be influenced by the stochasticity in the data splits during training and validation.

In our approach, we select a well-known benchmark dataset for regression, as described in Section 3.1. We split the dataset into a training set, a validation set that is used during the HP optimisation process and an unseen test set for measuring performance. While some surveys [19,18] only keep the training and test sets constant, we keep all three dataset splits constant throughout the model development process.

We define a protocol that aims to find a strong set of HPs for each model. The protocol itself is described in Section 3.2 and how it is applied for model development is described in Section 3.3. We compare the performance of the models using the root mean squared error (RMSE) evaluation metric.

3.1 Data

Rationale The Rossmann store sales dataset is hosted on Kaggle for a 2015 competition[5]. It is used regularly by the machine learning community to benchmark new architectures for regression modelling [9]. Our research is interested in answering questions regarding transformer-based models with a regression use case in mind. The original proposal paper for TabNet [1] and its results on this benchmark were considered, noting the comparative studies done to demonstrate its performance. RMSE is used as the main metric because it is directly comparable to that of the original TabNet proposal paper [1].

[5] `https://www.kaggle.com/competitions/rossmann-store-sales`

Description The dataset contains varying features, static categorical features mixed with time series sales data [1]. It consists of data for 1115 stores based in Germany across a six-week timeframe. The goal is to predict the sales of Rossmann stores from these variables. Three Rossmann benchmark datasets are obtained from Kaggle. These consist of a training dataset with time series data, a dataset with store information and a test dataset with no target column for submission to the Kaggle competition. The training dataset consists of 804,056 samples, each representing data at the granularity of an individual store on a specific day. The store data set includes metadata on the stores such as promo-

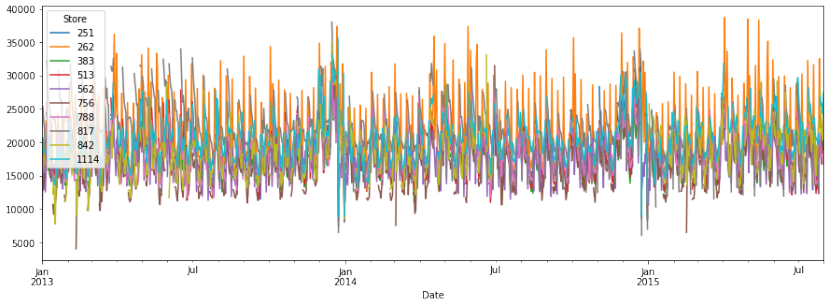

Fig. 1. Sales of the top ten stories across the six-week period provided, as they are found in the training dataset. This shows variations in sales because of seasonality and promotional intervals, among others: patterns that the algorithms can potentially learn.

tional intervals, distance from competition, the assortment of store content and store types. The features of the dataset is described in Table 1.

Other results There is little information available on the test set used in the TabNet paper. Because Kaggle does not share their test set results, we have to conclude that Arık et al. defined their own process and used it to define a test set. Their results are shown in Table 2 We used the values here as a reference and benchmark for our case study. It is assumed that the mean squared error (MSE) referred to here in the TabNet paper is the RMSE as with the code implementation of the Shwartz-Ziv et al. [19] study, because of the size ranges of the values. Also, if they are assumed to be MSE the results from the Shwartz-Ziv et al. [19] study would not corroborate the findings of Arik et al. [1].

No code or methodology is provided by Arık et al. [1] to replicate their results. This is also listed as an issue on the implementation's GitHub repository [6]. Because TabNet is such a landmark study, the first of the major tabular transformer models, we first recreate their results.

[6] https://github.com/dreamquark-ai/tabnet/issues/268

Table 1. Rossmann dataset features and data types showing both static and time-varying features.

Feature	**Data type**
Store	Integers
DayOfWeek	Integers
Date	Date
Sales	Integers
Customers	Integers
Open	Integers
Promo	Integers
StateHoliday	Categorical
SchoolHoliday	Integers
StoreType	Categorical
Assortment	Categorical
CompetitionDistance	Real Numbers
CompetitionOpenSinceMonth	Real Numbers
CompetitionOpenSinceYear	Real Numbers
Promo2	Integers
Promo2SinceWeek	Real Numbers
Promo2SinceYear	Real Numbers
PromoInterval	Categorical

Pre-processing The target variable, **Sales** (per day), is log-transformed in order to lower the skewness. This is done using the function.

$$f(x) = log(1 + x)$$

The effect is captured in Figure 2: The sales target variable has to be transformed back to the original domain for calculation of the RMSE. Other steps that are important include label encoding of categorical variables using skcikit-learn's [16] default process and building an 'IsPromoMonth' binary feature that indicates months that are in a promotional interval.

Table 2. Rossmann store sales results from the Arik et al. study in which TabNet was proposed [1].

Model	**Test MSE**
MLP	512.62
XGBoost	490.83
LightGBM	504.76
CatBoost	489.75
TabNet	**485.12**

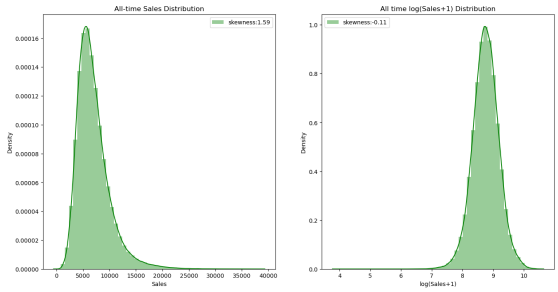

Fig. 2. Target variable distribution before log transformation (left) and after log transformation (right). Targets were log transformed to address target distribution skewness.

3.2 Protocol

HP optimisation is performed using Bayesian optimisation, specifically the algorithm as implemented by Weights and Biases [7]. For the initial run with each of the model architectures, we use the default HPs for that architecture. (These can be found in the appendix.) After the first run, Bayesian optimisation is used to suggest HP ranges for the next sweep. Where a model fails to find a minimum in the loss landscape during the default run, we adjust the HPs by increasing or decreasing the learning rate until a model is found that converges. The configuration from Weights and Biases for each of the models is then used to run a HP sweep for each model, updating the model HPs after each model convergence. This process is repeated until the validation score converges using different combinations of HPs. For GBDT architectures around 20 hours of compute time seem satisfactory. For transformer-based architectures, this minimum is significantly higher at around 100 to 120 hours.

Three separate sweeps are performed changing only the initialisation random seed between them, in order to test the architecture's sensitivity to random seeds [2]. For each sweep, the objective is set to minimize the validation score. The model with the best validation score is then found for each architecture type. The HP spaces suggested, and consequently investigated for the model are kept constant for three arbitrary random seeds, repeating the process described. Seeds are not changed during HP sweeps, in order to maintain the integrity of the Bayesian optimisation process.

For the different architectures, the code base associated with the paper in which the architecture was proposed, is used for the implementation. A layer of abstraction is added for use with the Weights and Biases application Programming Interface (API) to implement the HP sweeps. Logging and graphing of results is also done through this API.

[7] https://wandb.ai/site/articles/bayesian-hyperparameter-optimisation-a-primer

3.3 Model Development

In this section, we describe each model's default parameters, HP sweep ranges and parallel coordinate plots. The loss function is set to RMSE for all runs of the different models. We describe one optimisation (CatBoost) in detail. For the rest of the models, the HP defaults and ranges can be found in the appendix.

Table 3. CatBoost - Default HP values and HP search space during Bayesian optimisation.

Parameter Name	Default Values	Range	Distribution
Epochs	1,000	50 - 20,000	Uniform (integers)
Depth	6	3 - 12	Uniform (integers)
Learning Rate	0.01	0.005 - 0.02	Univorm (float)
L2 Leaf Reg	12	1 - 30	Uniform (integers)
Border Count	128	32 - 255	Uniform (integers)
Min Data in Leaf	3	1 - 20	Uniform (integers)
One Hot Max Size	2	2 - 255	Uniform (integers)
CTR Complexity	1	1 - 10	Uniform (integers)
Leaf Estimation Iterations	1	1 - 10	Uniform (integers)
Leaf Estimation Method	Newton	Newton, Gradient	Categorical

CatBoost The HP spaces investigated for CatBoost can be found in Table 3. Two graphs are relevant for each HP sweep. The first is a parallel coordinates plot, depicting the combination of HPs with its consequential scores after training. See Figure 3 for an example of a run for one seed.

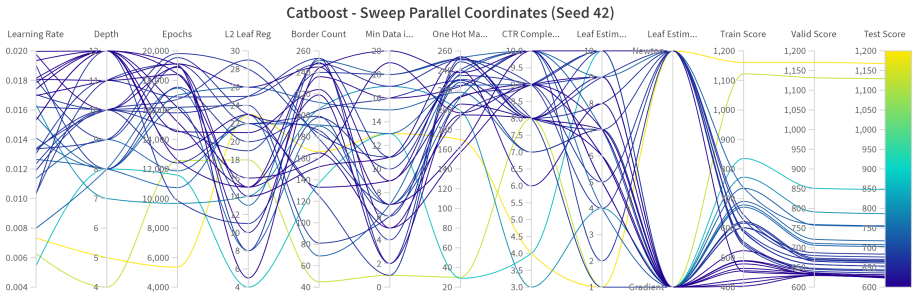

Fig. 3. A parallel coordinates plot showing the HPs for the CatBoost HP sweep, along with the RMSE scores achieved with each combination. When a good combination of HPs is found, the Bayesian algorithm iteratively optimises the search by updating its own model linking HPs to performance.

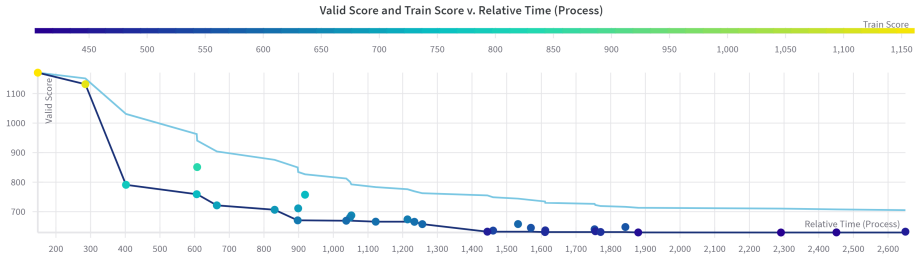

Fig. 4. CatBoost - Training and validation scores as process time increases for random seed 42. It can be seen that the validation scores reach an asymptote.

XGBoost The objective is set to regression with a squared error loss function (*reg:squarederror*). The default parameters used for the first run can be found in Table 7 in the appendix. Early stopping at 30 rounds was implemented for XGBoost to prevent overfitting. In the second and third random seed sweeps, the *dart* booster was removed as it caused overfitting.

TabNet As a sign of the complexity of DNN-based models, both TabNet and SAINT have more HPs available to tune. The default values for TabNet can be found in Table 8 in the appendix.

SAINT In our experiments with SAINT, we set the learning rate to 0.00001 as per the default code from the official SAINT repository for the first runs. However, we observed that the loss did not diminish, suggesting that the learning rate might not be optimal for the combination of the loss landscape and model architecture. We updated the learning rate and consequent ranges.

Table 4. Total runtime across all models developed per technique and seed, as well as average and maximum runtime of a single model training process.

	Seed	**CatBoost**	**XGBoost**	**TabNet**	**SAINT**
Runtime Total (Hours)	42	20.89	15.17	157.40	336.84
	7	21.00	19.23	40.12	44.98
	3	21.91	6.42	48.60	75.43
	Total	63.80	**40.82**	246.12	457.25
Runtime Avg (Hours)	42	0.67	0.12	1.17	7.66
	7	0.64	0.16	0.89	2.14
	3	0.59	0.06	0.97	3.14
	Avg	0.63	**0.12**	1.07	5.14
Runtime Max (Hours)	42	7.18	0.67	13.01	37.90
	7	3.10	1.13	3.83	15.42
	3	1.32	1.07	4.14	9.03
	Max	7.18	**1.13**	13.01	37.90

Table 5 shows the results for each seed, grouping them by initialisation seeds first, and then across all seeds.

Table 5. Performance across all models per technique and seed.

Metric	Seed	CatBoost	XGBoost	TabNet	SAINT
Min of Test Score	42	623.41	636.70	739.98	568.77
	7	623.64	657.84	847.52	604.43
	3	623.91	653.66	774.09	561.49
	Min	623.41	636.70	739.98	**561.49**
Average of Test Score	42	701.73	805.04	1,246.75	1,281.49
	7	745.54	922.44	1,529.19	1,863.28
	3	712.68	1,052.53	1,650.34	1,323.00
	Avg	**720.06**	919.88	1,513.19	1,409.57
StdDev of Test Score	42	130.91	455.10	465.46	1,057.73
	7	346.04	481.82	365.29	1,152.37
	3	320.01	609.34	957.82	1,195.88
	Avg	**284.01**	522.80	760.62	1,126.78
Run Count	42	31	105	135	44
	7	33	124	45	21
	3	37	122	50	24
	Total	101	351	230	**89**

4 Results

We show results comparing the architectures in terms of the test scores for the out-of-sample not used in the HP tuning process. We also discuss the implications of HP-tuning strategies, feature skewness and computational considerations in our research.

4.1 Performance Comparison

In Table 6 we provide RMSE results for the four models. In each case, we show the best model (minimum RMSE), average RMSE and the standard deviation across initialisation seeds. In this study, the best result is achieved by SAINT, but with a large variability across runs. On average, CatBoost has the lowest validation scores with the lowest standard deviation.

4.2 Implications of HP Tuning Strategies

In order to determine the importance of the HP tuning protocol used in this study, we implement the McElfresh et al. HP tuning method [14] for the Rossmann store sales. TabZilla [8] is implemented with tenfold cross-fold validation as

[8] https://github.com/naszilla/tabzilla

Table 6. The best model's performance (RMSE) on the Test set. The average and standard deviation of the RMSE is calculated for all runs across all random seeds.

Model	Minimum	Average	Std Dev
SAINT	**561.49**	1,409.57	1,126.78
TabNet	739.98	1,513.19	760.62
CatBoost	623.41	**720.06**	**284.01**
XGBoost	636.70	919.88	522.88

is standard practice with datasets obtained from OpenML [9]. However, we keep the train, validation and test sets the same between models but use the rest of their experimental protocol. In McElfresh et al. researchers keep the learning rate constant throughout the HP tuning process at 0.00003, at least for the SAINT model, where we vary this as part of our HP tuning process. It obtains relatively high importance in our sweeps, ranking fourth. Logging a Tabzilla run with the XGBoost model after importing the preprocessed Rossmann Store Sales data to OpenML we obtain the results in Figure 5. These runs were done following McElfresh et al. and their experimental methodology exacly [14]. In Figure 5, note that 'alpha' is the L1 regularization term, 'eta' refers to the learning rate and 'lambda' is the L2 regularisation term on the weights. In our method, more parameters are swept. See Figure 6 for detail of the HP sweep done using our method for the same model.

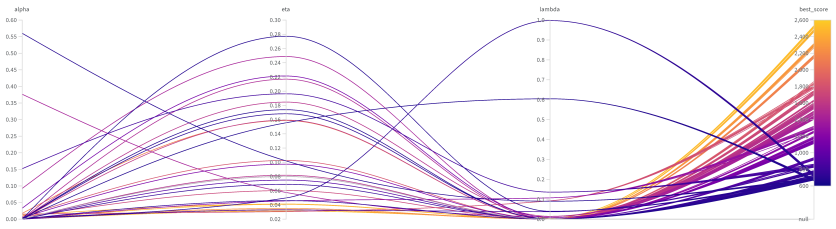

Fig. 5. HPs tested using the process from the proposers of Tabzilla [14]

When compared to the Bayesian optimisation protocol used for our work it can be seen that a lot less time is spent in HP spaces that are not viable. Comparing Figure 5 with Figure 6 it is evident that more time is spent searching for optimal HPs than eliminating bad combinations of HPs. This effect is aggravated by the fact that adding an additional parameter expands the HP space.

The best score obtained using the protocol as described in the original Tabzilla paper is 670,26. This score is on the same size validation sample dataset as our method, but the split is different because of cross-fold validation being applied. Our best result with the same model framework is 639,55.

[9] https://www.openml.org/

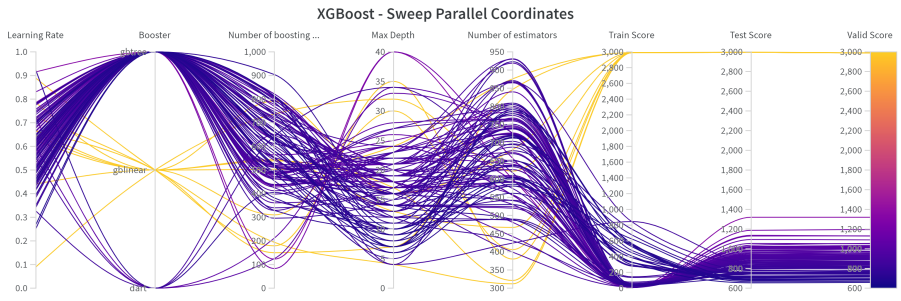

Fig. 6. The same task as Figure 5 but using the protocol defined here.

It is clear that in this setup a random grid search does not find as good a solution as Bayesian optimisation. A lot of precious computational time is spent on randomly generated HPs that yield poor results. Only 110 of the 329 individual runs, or close to 30% of runs are below 1,000 RMSE. This is in stark contrast with the Bayesian optimisation runs where 312 of the 351 runs, or 89% of the individual runs are within this range for the XGBoost architecture (across all random seeds). Even more notably, more than 50% of the runs are between 600 and 800 RMSE.

Using a Bayesian optimisation method has an advantage compared to the random grid search used in other comparative studies [14,12], as expected. What is interesting, is the large effect this could have when comparing models.

4.3 Addressing Skewness

It is observed by a recent study [14] that feature distribution and specifically the kurtosis negatively affects model performance. Using the same protocol as defined in Section 3.2, we initiate a HP sweep for two architectures, one from each model family. However, we do not perform the log transformation of the target variable, Sales. We compare these results to the log-transformed as obtained in 4. Our results corroborate the findings with both model types performing worse than when the target variable is transformed. This effect is more pronounced for the transformer-based model, again affecting the results of comparative studies.

For runs where the log-transform is not applied to the target variable our limited runs indicate that this affects the GBDT to a lesser degree than the DNN-based transformer model.

4.4 Computing Considerations

In total more than 1,000 hours worth of HP optimisation and training computational time, distributed between a Nvidia RTX3060 and RTX2080 Super graphics processing units (GPUs), was utilized for this study. Each of these is paired with a 12-core/24-thread AMD Ryzen 3900X CPU and 128GB or RAM.

5 Conclusion

In this study, we compared two GBDT-based and two transformer-based models on the Rossmann Stores Sales regression task. Our primary objective was to determine whether a more rigorous HP optimisation process would influence the comparative performance of architecturescompared to studies that used a less optimal methodology. In this analysis, the tuning process had a large effect, with SAINT resulting in the most performant model on the Rossmann Sales task of the four architectures tested. XGBoost remains the architecture with the least variance.

While not specifically investigated in this study, the above could explain why McElfresh et al. [14] found that the gap between the two families of methods became larger (transformers performed relatively worse) as dataset size increased. This is unexpected, as transformers are anticipated to be able to handle larger datasets with comparable ease. However, the effect of a suboptimal tuning protocol (and premature ending of training runs) is also expected to become more severe.

In addition, the effect of correcting the skewness of the target variable was investigated. This is typically done for DNN models but is not necessary for GBDT models and therefore usually omitted in comparative studies. (Note that this was only done for the target variable – none of the other features were transformed, which we leave as future work.)

To conclude: while more constrained evaluations (for example, best evaluation within a certain computational budget) are also valuable, we highlight the risks of attempting to compare the 'optimal' performance of different architectures without optimising according to the requirements of each architecture. Here, the HP tuning protocol and feature preprocessing have been shown to not only have a large effect on individual model performance but also to affect comparative performance.

References

1. Arik, S.Ö., Pfister, T.: Tabnet: Attentive interpretable tabular learning. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 6679–6687 (2021)
2. Bethard, S.: We need to talk about random seeds. arXiv preprint arXiv:2210.13393 (2022)
3. Biewald, L.: Machine learning experiment tracking. Weights & Biases https://www. wandb. com (2020)
4. Borisov, V., Broelemann, K., Kasneci, E., Kasneci, G.: Deeptlf: robust deep neural networks for heterogeneous tabular data. International Journal of Data Science and Analytics pp. 1–16 (2022)
5. Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., Kasneci, G.: Deep neural networks and tabular data: A survey. IEEE Transactions on Neural Networks and Learning Systems (2022)
6. Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., Chen, K., et al.: Xgboost: extreme gradient boosting. R package version 0.4-2 **1**(4), 1–4 (2015)

7. Chui, M., Manyika, J., Miremadi, M., Henke, N., Chung, R., Nel, P., Malhotra, S.: Notes from the ai frontier: Insights from hundreds of use cases. McKinsey Global Institute p. 28 (2018)
8. Dorogush, A.V., Ershov, V., Gulin, A.: Catboost: gradient boosting with categorical features support. arXiv preprint arXiv:1810.11363 (2018)
9. FlorianKnauer, W.C.: Rossmann store sales (2015), `https://kaggle.com/competitions/rossmann-store-sales`
10. Friedman, J.H.: Stochastic gradient boosting. Computational Statistics Data Analysis **38**, 367–378 (2002)
11. Gorishniy, Y., Rubachev, I., Khrulkov, V., Babenko, A.: Revisiting deep learning models for tabular data. Advances in Neural Information Processing Systems **34**, 18932–18943 (2021)
12. Grinsztajn, L., Oyallon, E., Varoquaux, G.: Why do tree-based models still outperform deep learning on typical tabular data? Advances in Neural Information Processing Systems **35**, 507–520 (2022)
13. Martins, A., Astudillo, R.: From softmax to sparsemax: A sparse model of attention and multi-label classification. In: International conference on machine learning. pp. 1614–1623. PMLR (2016)
14. McElfresh, D., Khandagale, S., Valverde, J., Ramakrishnan, G., Goldblum, M., White, C., et al.: When do neural nets outperform boosted trees on tabular data? arXiv preprint arXiv:2305.02997 (2023)
15. Nielsen, D.: Tree boosting with xgboost-why does xgboost win" every" machine learning competition? Master's thesis, NTNU (2016)
16. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
17. Popov, S., Morozov, S., Babenko, A.: Neural oblivious decision ensembles for deep learning on tabular data. arXiv preprint arXiv:1909.06312 (2019)
18. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: Catboost: unbiased boosting with categorical features. Advances in neural information processing systems **31** (2018)
19. Shwartz-Ziv, R., Armon, A.: Tabular data: Deep learning is not all you need. Information Fusion **81**, 84–90 (2022)
20. Somepalli, G., Goldblum, M., Schwarzschild, A., Bruss, C.B., Goldstein, T.: Saint: Improved neural networks for tabular data via row attention and contrastive pretraining. arXiv preprint arXiv:2106.01342 (2021)
21. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
22. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997). `https://doi.org/10.1109/4235.585893`

A HP Defaults and Ranges

A.1 XGBoost

Table 7. Default HP values and HP search space - XGBoost

Parameter Name	Default Value	Range	Distribution
Learning rate	0.3	0.07895 - 0.97295	uniform
Booster	gbtree	gbtree, gblinear	categorical
Number of estimators	200	25 - 976	Uniform (integers)
Max Depth	10	4 - 40	Uniform (integers)
Number of boosting rounds	200	64 - 952	Uniform (integers)

A.2 TabNet

Table 8. Default HP values and HP search space - TabNet

Parameter Name	Default Value	Range	Distribution
Width of decision prediction layer (n_d)	8	32 - 512	Uniform (integers)
Width of attention embedding for each mask (n_a)	8	32 - 512	Uniform (integers)
The number of steps (n_steps)	3	3 - 10	Uniform (integers)
Max epochs	200	1,000	Fixed Value
Learning rate initial	0.1	0.005 - 0.05	uniform
Learning rate scheduler	ReduceLROnPlateau	-	-
Batch size	1,024	128 - 5,048	Uniform (integers)
Virtual batch size	128	64 - 1,024	Uniform (integers)

A.3 SAINT

Table 9. Default HP values and HP search space - SAINT

Parameter Name	Default Value	Range	Distribution
Attention dropout	0.1	0.05 - 0.3	uniform
Attention heads	3	2 - 8	Uniform (integers)
Batch size	256	128 - 512	Uniform (integers)
Embedding size	32	16 - 64	Uniform (integers)
Epochs	100	50 - 200	Uniform (integers)
FF dropout	0.1	0.1 - 0.9	uniform
Learning rate	0.00001	5×10^{-5} - 0.001	uniform
Transformer depth	6	3 - 12	Uniform (integers)