Information Fusion 81 (2022) 84–90

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

Tabular data: Deep learning is not all you need

Ravid Shwartz-Ziv [*], Amitai Armon

IT AI Group, Intel, Israel

ARTICLE INFO

Keywords:
Tabular data
Deep neural networks
Tree-based models
Hyperparameter optimization

ABSTRACT

A key element in solving real-life data science problems is selecting the types of models to use. Tree ensemble models (such as XGBoost) are usually recommended for classification and regression problems with tabular data. However, several deep learning models for tabular data have recently been proposed, claiming to outperform XGBoost for some use cases. This paper explores whether these deep models should be a recommended option for tabular data by rigorously comparing the new deep models to XGBoost on various datasets. In addition to systematically comparing their performance, we consider the tuning and computation they require. Our study shows that XGBoost outperforms these deep models across the datasets, including the datasets used in the papers that proposed the deep models. We also demonstrate that XGBoost requires much less tuning. On the positive side, we show that an ensemble of deep models and XGBoost performs better on these datasets than XGBoost alone.

1. Introduction

Deep neural networks have demonstrated great success across various domains, including images, audio, and text [1–3]. There are several canonical architectures for encoding raw data into meaningful representations in these domains. These canonical architectures usually perform well in real-world applications.

In real-world applications, the most common data type is tabular data, comprising samples (rows) with the same set of features (columns). Tabular data is used in practical applications in many fields, including medicine, finance, manufacturing, climate science, and many other applications that are based on relational databases. During the last decade, traditional machine learning methods, such as gradient-boosted decision trees (GBDT) [4], still dominated tabular data modeling and showed superior performance over deep learning. When deep neural networks are applied to tabular data, many challenges arise, such as lack of locality, data sparsity (missing values), mixed feature types (numerical, ordinal, and categorical), and lack of prior knowledge about the dataset structure (unlike with text or images). Although the "no free lunch" principle [5] always applies, tree-ensemble algorithms, such as XGBoost, are considered the recommended option for real-life tabular data problems [4,6,7].

Recently, there have been several attempts to develop deep networks for tabular data [8–10], some of which have been claimed to outperform GBDT. However, each work in this field used different datasets since there is no standard benchmark (such as ImageNet [11] or GLUE [12]). As a result, it is challenging to compare tabular data models accurately, noting that some lack open-source implementations. In addition, papers that attempted to compare these models often did not optimize them equally. Thus, it became unclear whether deep learning models for tabular data surpass GBDT and which deep learning models perform best. These obscurities impede the research and development process and make the conclusions from these papers unclear. As the number of papers on deep learning for tabular data is rising, we believe it is time to rigorously review recent developments in this area and provide more substantiated conclusions that will serve as a basis for future research. The main purpose of this study is to explore whether any of the recently proposed deep models should indeed be a recommended choice for tabular dataset problems. There are two parts to this question: (1) Are the models more accurate, especially for datasets that did not appear in the paper that proposed them? (2) How long do training and hyperparameter search take in comparison to other models?

To answer these questions, we evaluate these recently proposed deep learning models and XGBoost on diverse tabular datasets with the same tuning protocol. We analyze the deep models proposed in four recent papers using eleven datasets, nine of which were used in these papers. We show that in most cases, each deep model performs best on the datasets used in its respective paper but significantly worse on other datasets. Additionally, our study shows that XGBoost usually outperforms deep models on these datasets. Furthermore, we demonstrate that the hyperparameter search process was much shorter for XGBoost. On the positive side, we examine the performance of an

* Corresponding author.
E-mail addresses: ravid.ziv@intel.com (R. Shwartz-Ziv), amitai.armon@intel.com (A. Armon).

https://doi.org/10.1016/j.inffus.2021.11.011
Received 22 June 2021; Received in revised form 24 September 2021; Accepted 9 November 2021
Available online 27 November 2021

ensemble of deep models combined with XGBoost and show that this ensemble gives the best results. It also performs better than an ensemble of deep models without XGBoost, or an ensemble of classical models.

Of course, any selection of tabular datasets cannot represent the full diversity of this type of data, and the ”no free lunch“ principle means that no one model is always better or worse than any other model. Still, our systematic study demonstrates that deep learning is currently not all we need for tabular data, despite the recent significant progress.

The paper is organized as follows: Section 2 provides a background and overview of the models for tabular data. Next, Section 3 presents the experimental setup and results. Finally, the discussion, conclusions, and suggested future work are included in Section 4.

2. Background

Traditionally, classical machine learning methods, such as GBDT [4] dominate tabular data applications due to their superior performance. Researchers and practitioners use several GBDT algorithms, including XGBoost, LightGBM, and CatBoost [4,13,14]. GBDT learns a series of weak learners to predict the output. In GBDT, the weak learner is the standard decision tree, which lacks differentiability. Despite their differences, their performance on many tasks is similar [14].

XGBoost Model. The XGBoost algorithm [4] is an extendible gradient boosting tree algorithm, that achieves state-of-the-art results on many tabular datasets [15,16]. Gradient boosting is an algorithm in which new models are created from previous models' residuals and then combined to make the final prediction. When adding new models, it uses a gradient descent algorithm to minimize the loss. XGBoost is one of the most popular GBDT implementations.

2.1. Deep neural models for tabular data

As mentioned above, several recent studies have applied deep learning to the tabular data domain, introducing new neural architectures to achieve improved performance on tabular data [8,10,17–21]. There are two main categories of these models, which we describe briefly below.

Differentiable trees. As a result of the excellent performance of ensembles of decision trees on tabular data, this line of work looks for ways to make decision trees differentiable. Classical decision trees cannot be used as components of end-to-end training pipelines, as they are not differentiable and do not allow gradient optimization. Several works address this issue by smoothing the decision functions in the internal tree nodes to make the tree function and tree routing differentiable [10,18,22].

Attention-based models. Since attention-based models are widely used in different fields, several authors have also proposed using attention-like modules for tabular deep networks. Recent works have suggested both inter-sample attention, in which features of a given sample interact with each other, and intra-sample attention, in which data points interact with each other using entire rows/samples [8,19,23].

In addition, the literature offers other architecture designs that do not fit explicitly into these two categories: (1) Regularization methods that learn a “regularization strength" for every neural weight, through the use of large-scale hyperparameter tuning schemes [24,25]; (2) Explicit modeling of multiplicative interactions, which considered different ways to incorporate feature products into the MLP model [26]; and (3) 1D-CNN, which utilizes the advantages of convolutions in tabular data [21].

As noted above, the community developed various models that were evaluated using different benchmarks; these models have rarely been compared.

Among the recently proposed deep models for learning from tabular data, we examine four models that have been claimed to outperform tree ensembles and have attracted significant industry attention: TabNet [8], NODE [10], DNF-Net [9], and 1D-CNN [21]. Below, we briefly describe the key ideas of these models that are relevant to this research.

TabNet. TabNet is a deep learning end-to-end model that performed well across several datasets [8]. It includes an encoder, in which sequential decision steps encode features using sparse learned masks and select relevant features for each row using the mask (with attention). Using sparsemax layers, the encoder forces the selection of a small set of features. The advantage of learning masks is that feature selection need not be all-or-nothing. Rather than using a hard threshold on a feature, a learnable mask can make a soft decision, thus relaxing classical (non-differentiable) feature selection methods.

Neural Oblivious Decision Ensembles (NODE). The NODE network [10] contains equal-depth oblivious decision trees (ODTs), which are differentiable such that error gradients can backpropagate through them. Like classical decision trees, ODTs split data according to selected features and compare each with a learned threshold. However, only one feature is chosen at each level, resulting in a balanced ODT that can be differentiated. Thus, the complete model provides an ensemble of differentiable trees.

DNF-Net. The idea behind DNF-Net [9] is to simulate disjunctive normal formulas (DNF) in deep neural networks. The authors proposed replacing the hard Boolean formulas with soft, differentiable versions of them. A key feature of this model is the disjunctive normal neural form (DNNF) block, which contains (1) a fully connected layer; and (2) a DNNF layer formed by a soft version of binary conjunctions over literals. The complete model is an ensemble of these DNNFs.

1D-CNN. Recently, 1D-conventional neural network (CNN) achieved the best single model performance in a Kaggle competition with tabular data [21]. The model is based on the idea that the CNN structure performs well in feature extraction. Still, it is rarely used in tabular data because the feature ordering has no locality characteristics. In this model, an fully connected layer is used to create a larger set of features with locality characteristics, and it is followed by several 1D-Conv layers with shortcut-like connections.

2.2. Model ensemble

Ensemble learning is a well-known method for improving performance and reducing variance through training multiple models and combining their predictions [27]. It enhances classifier performance by combining the outputs from many submodels (base learners) trained to solve the same task. The final prediction is obtained by combining the predictions made by each base learner. Consequently, ensembles tend to improve the prediction performance and reduce variance, leading to more stable and accurate results. Ensemble learning assumes that different machine learning methods may perform better or have mistakes in different situations. Hence, we would expect that a method that uses multiple machine learning methods would produce superior results. In the literature on ensemble learning, a variety of methods have been explored. Intuitively, using more data for training the base learners helps reduce their bias, and ensembling helps reduce the variance.

Ensemble learning can usually be classified into two main types. The first includes techniques based on randomization, such as random forests [28], where each ensemble member has a different initial parameterization and training data. In this type of models, the base learners can be trained simultaneously without interacting with each other. In the second type of ensembles, boosting-based approaches are used to fit the ensemble base learners sequentially. In both cases, all the base learners may either use the same architecture or differ in their architecture. Achieving high performance requires the individual base learners to provide sufficiently high performance [28]. In most ensembles, decision trees are used as the base learner. Breiman [29], introduced bagging, a method that combines decision trees generated by randomly selected subsets of the training data and votes on the final outcome. [30] presented the boosting technique, in which the weights of training samples are updated after each iteration of training, and weighted voting is used to combine the classification outputs. In boosting, the new models are added to adjust the existing models' errors.

The models are added iteratively until no significant improvements are observed. [31] suggested using linear regression to combine the outputs of neural networks, which later became known as stacking.

In our study, we use five classifiers in our ensemble: TabNet, NODE, DNF-Net, 1D-CNN, and XGBoost. To construct a practical and straightforward ensemble, we suggest two different versions: (1) Treating the ensemble as a uniformly weighted mixture model and combining the predictions as

$$p(y|x) = \sum_{k=1}^{K} p_{\theta_m}(y|x, \theta_m) \quad (1)$$

(2) A weighted average of the predictions from each trained model. The relative weights of each model are defined simply by the normalized validation loss:

$$p(y|x) = \sum_{k=1}^{K} l_k^{\mathrm{val}} p_{\theta_m}(y|x, \theta_m) \quad (2)$$

where l_k^{val} is the validation loss for the kth model. Having uniform weights makes Eq. (1) a special case of Eq. (2). Some of the models above have ensembles built into their design, as mentioned above. Nevertheless, these ensembles are of the same basic submodels with different parameters, not of different types of models. Since these models typically perform better with more data, we use the entire training dataset to train each model.

3. Comparing the models

We investigate whether the proposed deep models have advantages when used for various tabular datasets. For real-world applications, the models must (1) perform accurately, (2) be trained and make inferences efficiently, and (3) have a short optimization time (fast hyperparameter tuning). Therefore, we first evaluate the performance of the deep models, XGBoost, and ensembles on various datasets. Next, we analyze the different components of the ensemble. Then, we investigate how to select models for the ensemble and test whether deep models are essential for producing good results or combining 'classical' models (XGBoost, SVM [32] and CatBoost [7]) is sufficient. In addition, we explore the tradeoff between accuracy and computational resource requirements. Finally, we compare the hyperparameter search process of the different models and demonstrate that XGBoost outperforms the deep models.

3.1. Experimental setup

3.1.1. Data-sets description

Our experiments use 11 tabular datasets that represent diverse classification and regression problems. The datasets include 10 to 2,000 features, 1 to 7 classes, and 7,000 to 1,000,000 samples (for a full description, see Table 1). Additionally, they differ in the number of numerical and categorical features. In some datasets, you can find "heterogeneous" features — which describe the physical properties of an object in different units of measurement. Other datasets contain "homogeneous" features, such as pixels for images or words for text. We use nine datasets from the TabNet, DNF-Net, and NODE papers, drawing three datasets from each paper. In addition, we use two Kaggle datasets not used by any of these papers. 1D-CNN was recently proposed in a Kaggle competition for use on one specific dataset, which we do not explore. Each dataset was preprocessed and trained as described in the original paper. Data is standardized to have zero mean and unit variance, and the statistics for the standardization are calculated based on the training data. The datasets we use are Forest Cover Type, Higgs Boson, and Year Prediction [33], Rossmann Store Sales [34], Gas Concentrations, Eye Movements, and Gesture Phase [35], MSLR [36], Epsilon [37], Shrutime [38], and Blastchar [39].

3.1.2. Implementation details

The Optimization Process. To select the model hyperparameters, we used HyperOpt [40], which uses Bayesian optimization. The hyperparameter search was run for 1,000 steps on each dataset by optimizing the results on a validation set. The initial hyperparameters were taken from the original paper. Each model had 6–9 main hyperparameters that we optimized. For the deep learning model, these include the learning rate, number of layers, and number of nodes. The full hyperparameter search space for each model is provided in Appendix B.

We split the datasets into training, validation, and test sets in the same way as in the original papers that used them. When the split was reported to be random, we performed three repetitions of the random partition (as done in the original paper), and we reported their mean and the standard error of the mean. Otherwise, we used four random seed initializations in the same partition, and we report their average.

Metrics and evaluation. For binary classification problems, we report the cross-entropy loss. For regression problems, we report the root mean square error. We ran four experiments with different random seeds for each tuned configuration, and we reported the performance on the test set.

Statistical significance test. In addition to using the RMSE or cross-entropy loss for evaluating the performance of the models, it is also necessary to assess whether these differences are statistically significant. Friedman's test [41] is a widely used nonparametric method for testing statistical significance. Its advantage is that it does not assume that the data in each group is normally distributed. Using the Friedman test, we compare the different models' errors to see whether there is a statistically significant difference between them. Friedman's hypothesis states that the performance results come from the same population. To test this hypothesis, we examine whether the rank sums of the k classifiers included in the test differ significantly. After applying the omnibus Friedman test, we compare all classifiers against each other or against a baseline classifier. The significance level (95% in our study) determines whether the null hypothesis should be rejected based on the resulting p-value for each model pair. If the p-value is greater than 0.05, it will fail to reject the null hypothesis for this pair. Otherwise, if the p-value is less than 0.05, the null hypothesis will be rejected at a confidence level of 95%.

Training. For classification datasets, we minimize cross-entropy loss, while for regression datasets, we minimize and report mean squared error. We use the term "original model" to refer to the model used on a given dataset in the paper that presented the respective model. The "unseen datasets" for each model are those not mentioned in the paper that published the respective model. Note that a model's unseen dataset is not a dataset it was not trained on, but a dataset that did not appear in its original paper. For the deep models, we follow the original implementations and use the Adam optimizer [42] without learning rate schedules. For each dataset, we also optimize the batch size for all models. We continue training until there are 100 consecutive epochs without improvement on the validation set.

See Appendix B for a detailed description of the different baseline configurations and their hyperparameter search spaces.

3.2. Results

Do the deep models generalize well to other datasets?

We first explore whether the deep models perform well when trained on datasets that were not included in their original paper and compare them to XGBoost. Table 2 presents the mean and the standard error of the mean of the performance measure for each model for each dataset (a lower value indicates better performance). Columns 1–3 correspond to datasets from the TabNet paper, columns 4–6 to the DNF-Net paper, and columns 7–9 to the NODE paper. The last two columns correspond to datasets that did not appear in any of these papers.

As mentioned above, the Friedman test, with a 95% significance level, was carried out to check statistically significant differences between susceptibility models.

We make several observations regarding these results:

Table 1
Description of the tabular datasets.

Dataset	Features	Classes	Samples	Source	Paper
Gesture Phase	32	5	9.8k	OpenML	DNF-Net
Gas Concentrations	129	6	13.9k	OpenML	DNF-Net
Eye Movements	26	3	10.9k	OpenML	DNF-Net
Epsilon	2000	2	500k	PASCAL Challenge 2008	NODE
YearPrediction	90	1	515k	Million Song Dataset	NODE
Microsoft (MSLR)	136	5	964k	MSLR-WEB10K	NODE
Rossmann Store Sales	10	1	1018K	Kaggle	TabNet
Forest Cover Type	54	7	580k	Kaggle	TabNet
Higgs Boson	30	2	800k	Kaggle	TabNet
Shrutime	11	2	10k	Kaggle	New dataset
Blastchar	20	2	7k	Kaggle	New dataset

- In most cases, the models perform worse on unseen datasets than do the datasets' original models.
- The XGBoost model generally outperformed the deep models. For 8 of the 11 datasets, XGBoost outperformed the deep models, which did not appear in the original paper. For these datasets, the results were significant ($p < 0.005$).
- No deep model consistently outperformed the others. Each deep model was better only on the datasets that appeared in its own paper. However, the performance of the 1D-CNN model may seem better since all datasets were new to it.
- The ensemble of deep models and XGBoost outperformed the other models in most cases. For 7 of the 11 datasets, the ensemble of deep models or XGBoost was significantly better than the single deep models. The p-value in these cases was less than 0.005, which indicates the null hypothesis (i.e., no difference between the performance of the tested models) is rejected.

To directly compare between the different models, we calculate for each dataset the relative performance of each model compared to the best model for that dataset. Table 3 presents the averaged relative performance per model on all its unseen datasets (geometric mean). The ensemble of all models was the best model with 2.32% average relative increase (all p-values except one were less than 0.005). XGBoost was the second best with 3.4%, 1D-CNN had 7.5%, TabNet had 10.5%, DNF-Net had 11.8%, and NODE had 14.2%.

These results are surprising. When we trained on datasets other than those in their original papers, the deep models performed worse than XGBoost. Compared to XGBoost and the full ensemble, the single deep model's performance is much more sensitive to the specific dataset. There may be several reasons for the deep models' lower performance when they are trained on previously unseen datasets. The first possibility is **selection bias**. Each paper may have naturally demonstrated the model's performance on datasets with which the model worked well. The second possibility is differences in the **optimization of hyperparameters**. Each paper may have set the model's hyperparameters based on a more extensive hyperparameter search on the datasets presented in that paper, resulting in better performance. Our results for each model on its original datasets matched those presented in its respective paper, thus excluding implementation issues as the possible reason for our observations.

Do we need both XGBoost and deep networks?

In the previous subsection, we saw that the ensemble of XGBoost and deep models performed best across the datasets. It is therefore interesting to examine which component of our ensemble is mandatory. One question is whether XGBoost needs to be combined with the deep models, or would a simpler ensemble of nondeep models perform similarly. To explore this, we trained an ensemble of XGBoost and other nondeep models: SVM [32] and CatBoost [7]. Table 2 shows that the ensemble of classical models performed much worse than the ensemble of deep networks and XGBoost. Additionally, the table shows that the ensemble of deep models alone (without XGBoost) did not provide good results. These differences were significant (all p-values were extremely low and less than 0.005). This indicates that combining both the deep models and XGBoost provides an advantage for these datasets.

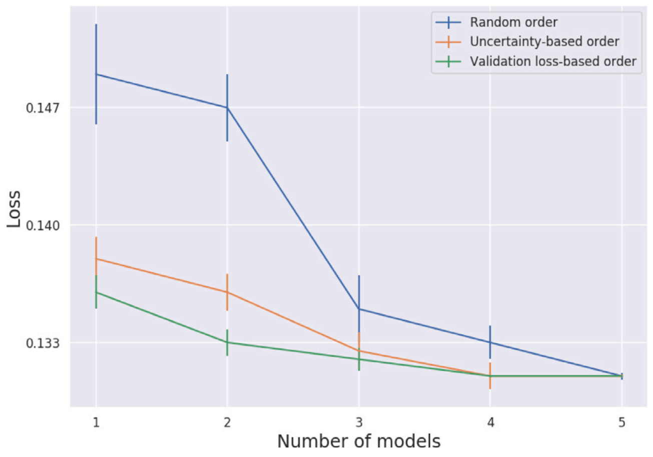

Fig. 1. The impact of selecting a subset of models in the ensemble.

Subset of models

We observed that the ensemble improved performance, but the use of multiple models also requires additional computation. When real-world applications are considered, computational constraints may affect the eventual model performance. We therefore considered using subsets of models within the ensemble to examine the tradeoff between performance and computation.

There are several ways to choose a subset from an ensemble of models: (1) based on the **validation loss**, choosing models with low validation loss first; (2) based on the models' **uncertainty for each example**, choosing the highest confidence models (by some uncertainty measure) for each example; and (3) based on a **random order**.

In Fig. 1, these methods of selecting models are compared for an example of an unseen dataset (Shrutime). The best selection approach was averaging the predictions based on the models' validation loss. Only three models were needed to achieve almost optimal performance this way. On the other hand, choosing the models randomly provided the worst choice according to our comparison. The differences in performance for the first three models were significant ($p < 0.005$).

How difficult is the optimization?

In real-life settings, there are limited time and resources to train a model for a new dataset and optimize its hyperparameters. It is therefore interesting to understand how difficult it is to do this for each model. One approach to evaluating this is to calculate the number of computations the model requires. This is typically measured in floating-point operations per second (FLOPS). Tang et al. [43] suggested that the FLOPS number is a better indicator of energy usage and latency than the number of parameters. However, each hyperparameter set has a different FLOPS number, so it is impossible to compare different models this way when optimizing the hyperparameters. Another approach is to compare the total time it takes to train and optimize the model.

Table 2
Test results on tabular datasets. Presenting the performance for each model. MSE is presented for the YearPrediction and Rossman datasets, and cross-entropy loss (with 100X factor) is presented for the other datasets. The papers that used these datasets are indicated below the table. The values are the averages of four training runs (lower value is better), along with the standard error of the mean (SEM)

Model Name	Rossman	CoverType	Higgs	Gas	Eye	Gesture
XGBoost	490.18 ± 1.19	3.13 ± 0.09	21.62 ± 0.33	2.18 ± 0.20	**56.07**±0.65	80.64 ± 0.80
NODE	488.59 ± 1.24	4.15 ± 0.13	21.19 ± 0.69	2.17 ± 0.18	68.35 ± 0.66	92.12 ± 0.82
DNF-Net	503.83 ± 1.41	3.96 ± 0.11	23.68 ± 0.83	**1.44** ±0.09	68.38 ± 0.65	86.98 ± 0.74
TabNet	**485.12**±1.93	3.01 ± 0.08	**21.14**±0.20	1.92 ± 0.14	67.13 ± 0.69	96.42 ± 0.87
1D-CNN	493.81 ± 2.23	3.51 ± 0.13	22.33 ± 0.73	1.79 ± 0.19	67.9 ± 0.64	97.89 ± 0.82
Simple Ensemble	488.57 ± 2.14	3.19 ± 0.18	22.46 ± 0.38	2.36 ± 0.13	58.72 ± 0.67	89.45 ± 0.89
Deep Ensemble w/o XGBoost	489.94 ± 2.09	3.52 ± 0.10	22.41 ± 0.54	1.98 ± 0.13	69.28 ± 0.62	93.50 ± 0.75
Deep Ensemble w XGBoost	485.33 ± 1.29	**2.99** ±0.08	22.34 ± 0.81	1.69 ± 0.10	59.43 ± 0.60	**78.93** ±0.73

TabNet DNF-Net

Model Name	YearPrediction	MSLR	Epsilon	Shrutime	Blastchar
XGBoost	77.98 ± 0.11	55.43±2e−2	11.12±3e−2	13.82 ± 0.19	20.39 ± 0.21
NODE	76.39 ± 0.13	55.72±3e−2	**10.39**±1e−2	14.61 ± 0.10	21.40 ± 0.25
DNF-Net	81.21 ± 0.18	56.83±3e−2	12.23±4e−2	16.8 ± 0.09	27.91 ± 0.17
TabNet	83.19 ± 0.19	56.04±1e−2	11.92±3e−2	14.94±, 0.13	23.72 ± 0.19
1D-CNN	78.94 ± 0.14	55.97±4e−2	11.08±6e−2	15.31 ± 0.16	24.68 ± 0.22
Simple Ensemble	78.01 ± 0.17	55.46±4e−2	11.07±4e−2	13.61±, 0.14	21.18 ± 0.17
Deep Ensemble w/o XGBoost	78.99 ± 0.11	55.59±3e−2	10.95±1e−2	14.69 ± 0.11	24.25 ± 0.22
Deep Ensemble w XGBoost	**76.19** ±0.21	**55.38**±1e−2	11.18±1e−2	**13.10**±0.15	**20.18**±0.16

NODE New datasets

Table 3
Average relative performance deterioration for each model on its unseen datasets (lower value is better).

Name	Average relative Performance (%)
XGBoost	3.34
NODE	14.21
DNF-Net	11.96
TabNet	10.51
1D-CNN	7.56
Simple Ensemble	3.15
Deep Ensemble w/o XGBoost	6.91
Deep Ensemble w XGBoost	**2.32**

Generally, we found XGBoost to be significantly faster than the deep networks in our experiments (more than an order of magnitude). However, runtime differences are significantly affected by the level of optimization of the software. Comparing a widely used library with non-optimized implementations from papers would not be fair. Thus, we used another approach: comparing the number of iterations of the hyperparameter optimization process until we reached a plateau. This can be seen as a proxy measure for the model's robustness and how far the default hyperparameters are from the best solutions. It represents an inherent characteristic of the model that does not depend on software optimization.

Fig. 2 shows the model's performance (mean and standard error of the mean) as a function of the number of iterations of the hyperparameter optimization process for the Shrutime dataset. We observe that XGBoost outperformed the deep models, converging to good performance more quickly (in fewer iterations, which were also shorter in terms of runtime). These results may be affected by several factors: (1) We used a **Bayesian hyperparameter optimization process**, and the results may differ for other optimization processes; (2) **The initial hyperparameters** of XGBoost may be more robust because they might have been originally suggested based on more datasets. Perhaps we could find some initial hyperparameters that would work better for the deep models for different datasets; and (3) The XGBoost model may have some **inherent characteristics** that make it more robust and easier to optimize. It may be interesting to investigate this behavior further.

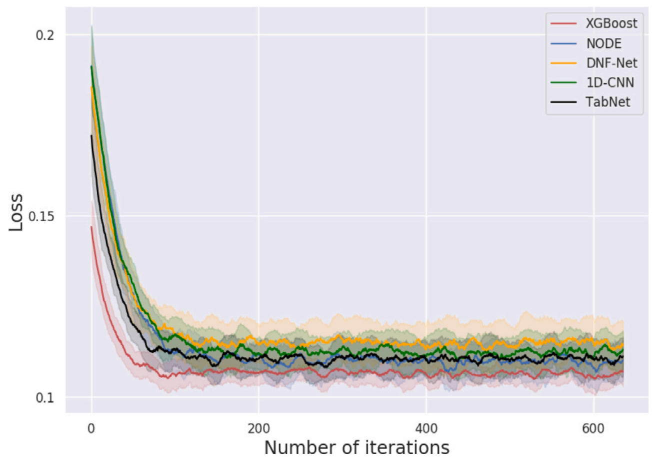

Fig. 2. The Hyper-parameters optimization process for different models.

4. Discussion and conclusions

In this study, we investigated the performance of recently proposed deep models for tabular datasets. In our analysis, the deep models were weaker on datasets that did not appear in their original papers, and they were weaker than XGBoost, the baseline model. Therefore, we proposed using an ensemble of these deep models with XGBoost. This ensemble performed better on these datasets than any individual model and the 'non-deep' classical ensemble. In addition, we explored possible tradeoffs between performance, computational inference cost, and hyperparameter optimization time, which are important in real-world applications. Our analysis shows that we must take the reported deep models' performance with a grain of salt. When we made a fair comparison of their performance on other datasets, they provided weaker results. Additionally, it is much more challenging to optimize deep models than XGBoost on a new dataset. However, we found that an ensemble of XGBoost with deep models yielded the best results for the datasets we explored.

Consequently, when researchers choose a model to use in real-life applications, they must consider several factors. Apparently, under time

Table A.4
Description of the tabular datasets.

Dataset	Features	Classes	Samples	Source	Paper	Link
Gesture Phase	32	5	9.8k	OpenML	DNF-Net	openml.org/d/4538
Gas Concentrations	129	6	13.9k	OpenML	DNF-Net	openml.org/d/1477
Eye Movements	26	3	10.9k	OpenML	DNF-Net	openml.org/d/1044
Epsilon	2000	2	500k	PASCAL Challenge 2008	NODE	https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html
YearPrediction	90	1	515k	Million Song Dataset	NODE	https://archive.ics.uci.edu/ml/datasets/yearpredictionmsd
Microsoft (MSLR)	136	5	964k	MSLR-WEB10K	NODE	https://www.microsoft.com/en-us/research/project/mslr/
Rossmann Store Sales	10	1	1018K	Kaggle	TabNet	https://www.kaggle.com/c/rossmann-store-sales
Forest Cover Type	54	7	580k	Kaggle	TabNet	https://www.kaggle.com/c/forest-cover-type-prediction
Higges Boson	30	2	800k	Kaggle	TabNet	https://www.kaggle.com/c/higgs-boson
Shrutime	11	2	10k	Kaggle	New dataset	https://www.kaggle.com/shrutimechlearn/churn-modelling
Blastchar	20	2	7k	Kaggle	New dataset	https://www.kaggle.com/blastchar/telco-customer-churn

constraints, XGBoost may achieve the best results and be the easiest to optimize. However, XGBoost alone may not be enough to achieve the best performance; we may need to add deep models to our ensemble to maximize performance.

In conclusion, despite significant progress using deep models for tabular data, they do not outperform XGBoost on the datasets we explored, and further research is probably needed in this area. Taking our findings into account, future research on tabular data must systematically check the performance on several diverse datasets. Our improved ensemble results provide another potential avenue for further research. Nevertheless, this is not sufficient. Comparing models requires including details about how easy it is to identify the most appropriate hyperparameters. Our study showed that some deep models require significantly more attempts to find the correct configuration. Another line of research could be developing new deep models that are easier to optimize and may compete with XGBoost in terms of this parameter.

CRediT authorship contribution statement

Ravid Shwartz-Ziv: Conceptualization, Methodology, Software. **Amitai Armon:** Writing – review & editing, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Tabular data-sets description

We use a wide variety of datasets, each with different characteristics, such as the number of features, the number of classes, and the number of samples. The datasets include both classification and regression tasks, including in high-dimensional space. In some datasets, you can find "heterogeneous" features — which describe the physical properties of an object in different units of measurement. Other datasets contain homogeneous features, such as pixels for images or words for text. We followed the training and prepossessing procedures outlined in the original study for each dataset (see Table A.4). The datasets we use are Forest Cover Type, Higgs Boson and Year Prediction datasets [33], Rossmann Store Sales [34], Gas Concentrations, Eye Movements and Gesture Phase [35], MSLR [36], Epsilon [37], Shrutime [38], and Blastchar [39] (see Table 1).

Appendix B. Optimization of hyperparameters

To tune the hyperparameters, we split the datasets into training, validation, and test sets in the same way as in the original papers that used them. Specifically, we performed a random stratified split of the full training data into train set (80%) and validation set (20%) for the Epsilon, YearPrediction, MSLR, Shrutime, and Blastchar datasets. For Eye, Gesture, and Year datasets, we split the full data into validation set (10%), test set (20%), and train set (70%). For Forest Cover Type, we use the train/val/test split provided by the dataset authors [44]. For the Rossmann dataset, we used the same preprocessing and data split as [7] – data from 2014 was used for training and validation, whereas 2015 was used for testing. We split $100k$ samples for validation from the training dataset, and after the optimization of the hyperparameters, we retrained on the entire training dataset. For the Higgs dataset, we split $500k$ samples for validation from the training dataset, and after the optimization of the hyperparameters, we retrained on the entire training dataset. We used the Hyperopt library to optimize the models. We selected the set of hyperparameters corresponding to the smallest loss on the validation set for the final configuration. For all models, early stopping is applied using the validation set.

B.1. CATBOOST

The list of hyperparameters and their search spaces for Catboost:

- Learning rate: Log-Uniform distribution $[e^{-5}, 1]$
- Random strength: Discrete uniform distribution $[1, 20]$
- Max size: Discrete uniform distribution $[0, 25]$
- L2 leaf regularization: Log-Uniform distribution $[1, 10]$
- Bagging temperature: Uniform distribution $[0, 1]$
- Leaf estimation iterations: Discrete uniform distribution $[1, 20]$

B.2. XGBoost

The list of hyperparameters and their search spaces for XGBoost:

- Number of estimators: Uniform distribution $[100, 4000]$
- Eta: Log-Uniform distribution $[e^{-7}, 1]$
- Max depth: Discrete uniform distribution $[1, 10]$
- Subsample: Uniform distribution $[0.2, 1]$
- Colsample bytree: Uniform distribution $[0.2, 1]$
- Colsample bylevel: Uniform distribution $[0.2, 1]$
- Min child weight: Log-Uniform distribution $[e^{-16}, e^5]$
- Alpha: Uniform choice {0, Log-Uniform distribution $[e^{-16}, e^2]$}
- Lambda: Uniform choice {0, Log-Uniform distribution $[e^{-16}, e^2]$}
- Gamma: Uniform choice {0, Log-Uniform distribution $[e^{-16}, e^2]$}

B.3. NODE

The list of hyperparameters and their search spaces for NODE:

- Learning rate: Log-Uniform distribution $[e^{-5}, 1]$
- Num layers: Discrete uniform distribution $[1, 10]$
- Total tree count: $\{256, 512, 1024, 2048\}$
- Tree depth: Discrete uniform distribution $[4, 9]$
- Tree output dim: Discrete uniform distribution $[1, 5]$
- Learning rate: Log-Uniform distribution $[e^{-4}, 0.5]$
- Batch size: Uniform choice {512, 1024, 2048, 4096, 8192}

B.4. TabNet

The list of hyperparameters and their search spaces for TabNet:

- Learning rate: Log-Uniform distribution $[e^{-5}, 1]$
- feature dim: Discrete uniform distribution $[20, 60]$
- output dim: Discrete uniform distribution $[20, 60]$
- n steps: Discrete uniform distribution $[1, 8]$
- bn epsilon: Uniform distribution $[e^{-5}, e^{-1}]$
- relaxation factor: Uniform distribution $[0.3, 2]$
- Batch size: Uniform choice {512, 1024, 2048, 4096, 8192}

B.5. DNF-Net

The list of hyperparameters and their search spaces for DNF-Net:

- n. formulas: Discrete uniform distribution $[256, 2048]$
- Feature selection beta: Discrete uniform distribution $[1e^{-2}, 2]$
- Learning rate: Log-Uniform distribution $[e^{-4}, 0.5]$
- Batch size: Uniform choice {512, 1024, 2048, 4096, 8192}

B.6. 1D-CNN

The list of hyperparameters and their search spaces for 1D-CNN:

- Hidden layer sizes: Discrete uniform distribution $[100, 4000]$
- Number of layers: Discrete uniform distribution $[1, 6]$
- Learning rate: Log-Uniform distribution $[e^{-4}, 0.5]$
- Batch size: Uniform choice {512, 1024, 2048, 4096, 8192}

References

[1] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, 2019.

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[3] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu, WaveNet: A generative model for raw audio, in: Proc. 9th ISCA Workshop on Speech Synthesis Workshop (SSW 9), 2016, p. 125.

[4] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[5] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.

[6] J.H. Friedman, Greedy function approximation: a gradient boosting machine, Ann. Statist. (2001) 1189–1232.

[7] L. Prokhorenkova, G. Gusev, A. Vorobev, A.V. Dorogush, A. Gulin, CatBoost: unbiased boosting with categorical features, in: 32nd Conference on Neural Information Processing Systems (NeurIPS), 2018.

[8] S. Arik, T. Pfister, Tabnet: Attentive interpretable tabular learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, (8) 2021, pp. 6679–6687, URL https://ojs.aaai.org/index.php/AAAI/article/view/16826.

[9] L. Katzir, G. Elidan, R. El-Yaniv, Net-DNF: Effective deep modeling of tabular data, in: 9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021, OpenReview.net, 2021, URL https://openreview.net/forum?id=73WTGs96kho.

[10] S. Popov, S. Morozov, A. Babenko, Neural oblivious decision ensembles for deep learning on tabular data, in: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020, OpenReview.net, 2020, URL https://openreview.net/forum?id=r1eiu2VtwH.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, L. Fei-Fei, Imagenet: A large-scale hierarchical image database, in: 2009 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2009, pp. 248–255.

[12] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, S. Bowman, GLUE: A multi-task benchmark and analysis platform for natural language understanding, in: Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP, Association for Computational Linguistics, Brussels, Belgium, 2018, pp. 353–355, http://dx.doi.org/10.18653/v1/W18-5446, URL https://aclanthology.org/W18-5446.

[13] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems, Vol. 30, Curran Associates, Inc., 2017, URL https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[14] L. Prokhorenkova, G. Gusev, A. Vorobev, A.V. Dorogush, A. Gulin, Catboost: Unbiased boosting with categorical features, in: Proceedings of the 32nd International Conference on Neural Information Processing Systems, in: NIPS'18, Curran Associates Inc., Red Hook, NY, USA, 2018, pp. 6639–6649.

[15] Y. Zhao, G. Chetty, D. Tran, Deep learning with XGBoost for real estate appraisal, in: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2019, pp. 1396–1401.

[16] S. Ramraj, N. Uzir, R. Sunil, S. Banerjee, Experimenting XGBoost algorithm for prediction and classification of different datasets, Int. J. Control Theory Appl. 9 (2016) 651–662.

[17] S. Badirli, X. Liu, Z. Xing, A. Bhowmik, K. Doan, S.S. Keerthi, Gradient boosting neural networks: Grownet, 2020, arXiv preprint arXiv:2002.07971.

[18] H. Hazimeh, N. Ponomareva, P. Mol, Z. Tan, R. Mazumder, The tree ensemble layer: Differentiability meets conditional computation, in: International Conference on Machine Learning, PMLR, 2020, pp. 4138–4148.

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

[20] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing neural networks, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, 2017, pp. 972–981.

[21] Baosenguo, Baosenguo/kaggle-moa-2nd-place-solution, 2021, URL https://github.com/baosenguo/Kaggle-MoA-2nd-Place-Solution.

[22] P. Kontschieder, M. Fiterau, A. Criminisi, S.R. Bulò, Deep neural decision forests, in: 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1467–1475, http://dx.doi.org/10.1109/ICCV.2015.172.

[23] G. Somepalli, M. Goldblum, A. Schwarzschild, C.B. Bruss, T. Goldstein, Saint: Improved neural networks for tabular data via row attention and contrastive pre-training, 2021, arXiv preprint arXiv:2106.01342.

[24] A. Kadra, M. Lindauer, F. Hutter, J. Grabocka, Regularization is all you need: Simple neural nets can excel on tabular data, 2021, arXiv preprint arXiv:2106.11189.

[25] I. Shavitt, E. Segal, Regularization learning networks: deep learning for tabular datasets, in: 32nd Conference on Neural Information Processing Systems (NeurIPS), 2018.

[26] A. Beutel, P. Covington, S. Jain, C. Xu, J. Li, V. Gatto, E.H. Chi, Latent cross: making use of context in recurrent recommender systems, in: WSDM 2018: The Eleventh ACM International Conference on Web Search and Data Mining, 2018.

[27] R. Caruana, A. Niculescu-Mizil, G. Crew, A. Ksikes, Ensemble selection from libraries of models, in: Proceedings of the Twenty-First International Conference on Machine Learning, 2004, p. 18.

[28] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.

[29] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.

[30] Y. Freund, R.E. Schapire, et al., Experiments with a new boosting algorithm, in: Icml, Vol. 96, Citeseer, 1996, pp. 148–156.

[31] D.H. Wolpert, Stacked generalization, Neural Netw. 5 (2) (1992) 241–259.

[32] C. Cortes, V. Vapnik, Support-vector networks, Mach. Learn. 20 (3) (1995) 273–297.

[33] D. Dua, C. Graff, UCI machine learning repository, 2017, URL http://archive.ics.uci.edu/ml.

[34] Kaggle, Rossmann store sales, 2019, URL https://www.kaggle.com/c/rossmann-store-sales.

[35] J. Vanschoren, J.N. Van Rijn, B. Bischl, L. Torgo, OpenML: networked science in machine learning, ACM SIGKDD Explor. Newsl. 15 (2) (2014) 49–60.

[36] T. Qin, T. Liu, Introducing LETOR 4.0 datasets, 2013, CoRR http://arxiv.org/abs/1306.2597.

[37] Pascal, Pascal large scale learning challenge, 2008, URL https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[38] Kaggle, Churn modelling, 2019, URL https://www.kaggle.com/shrutimechlearn/churn-modelling.

[39] IBM, Telco customer churn, 2019, https://community.ibm.com/community/user/businessanalytics/blogs/steven-macko/2019/07/11/telco-customer-churn-1113.

[40] J. Bergstra, B. Komer, C. Eliasmith, D. Yamins, D.D. Cox, Hyperopt: a python library for model selection and hyperparameter optimization, Comput. Sci. Discov. 8 (1) (2015) 014008.

[41] M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, J. Amer. Statist. Assoc. 32 (200) (1937) 675–701, http://dx.doi.org/10.1080/01621459.1937.10503522, http://arxiv.org/abs/https://www.tandfonline.com/doi/pdf/10.1080/01621459.1937.10503522.

[42] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, in: 8th International Conference on Learning Representations, ICLR 2020, 2015.

[43] R. Tang, W. Wang, Z. Tu, J. Lin, An experimental analysis of the power consumption of convolutional neural networks for keyword spotting, in: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2018, pp. 5479–5483.

[44] R. Mitchell, A. Adinets, T. Rao, E. Frank, Xgboost: Scalable GPU accelerated learning, 2018, arXiv preprint arXiv:1806.11248.