When Do Neural Nets Outperform Boosted Trees on Tabular Data?

Duncan McElfresh[*][1,2], **Sujay Khandagale**[3], **Jonathan Valverde**[4], **Vishak Prasad C**[5], **Ganesh Ramakrishnan**[5], **Micah Goldblum**[6], **Colin White**[1,7]

[1] Abacus.AI, [2] Stanford, [3] Pinterest, [4] University of Maryland, [5] IIT Bombay, [6] New York University, [7] Caltech

Abstract

Tabular data is one of the most commonly used types of data in machine learning. Despite recent advances in neural nets (NNs) for tabular data, there is still an active discussion on whether or not NNs generally outperform gradient-boosted decision trees (GBDTs) on tabular data, with several recent works arguing either that GBDTs consistently outperform NNs on tabular data, or vice versa. In this work, we take a step back and question the importance of this debate. To this end, we conduct the largest tabular data analysis to date, comparing 19 algorithms across 176 datasets, and we find that the 'NN vs. GBDT' debate is overemphasized: for a surprisingly high number of datasets, either the performance difference between GBDTs and NNs is negligible, or light hyperparameter tuning on a GBDT is more important than choosing between NNs and GBDTs. Next, we analyze dozens of metafeatures to determine what *properties* of a dataset make NNs or GBDTs better-suited to perform well. For example, we find that GBDTs are much better than NNs at handling skewed or heavy-tailed feature distributions and other forms of dataset irregularities. Our insights act as a guide for practitioners to determine which techniques may work best on their dataset. Finally, with the goal of accelerating tabular data research, we release the TabZilla Benchmark Suite: a collection of the 36 'hardest' of the datasets we study. Our benchmark suite, codebase, and all raw results are available at `https://github.com/naszilla/tabzilla`.

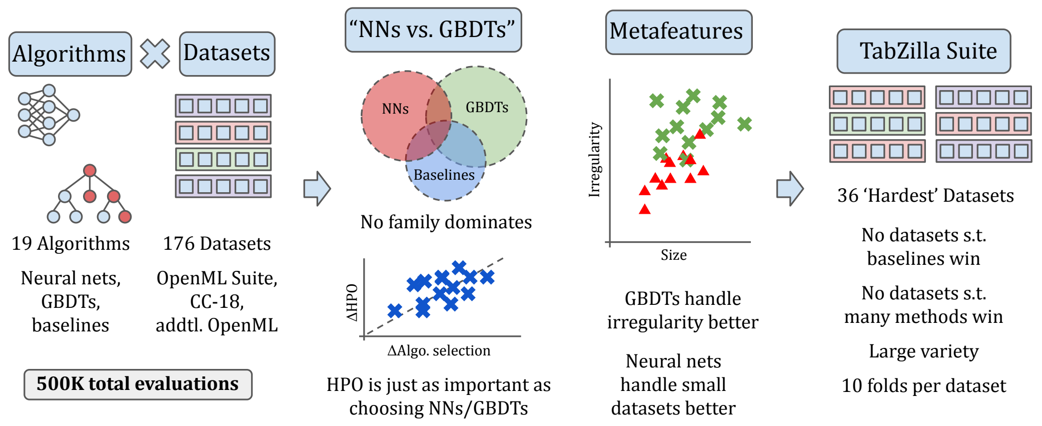
1 Introduction

Tabular datasets are data organized into rows and columns, consisting of distinct features that are typically continuous, categorical, or ordinal. They are the oldest and among the most ubiquitous dataset types in machine learning in practice [7, 55], due to their numerous applications across medicine [36, 58], finance [5, 13], online advertising [28, 45, 50], and many other areas [9, 10, 59].

Despite recent advances in neural network (NN) architectures for tabular data [4, 47], there is still an active debate over whether or not NNs generally outperform gradient-boosted decision trees (GBDTs) on tabular data, with multiple works arguing either for [4, 37, 40, 47, 51] or against [7, 26, 27, 55] NNs. This is in stark contrast to other areas such as computer vision and natural language understanding, in which NNs have far outpaced competing methods [8, 18, 19, 39, 61].

Nearly all prior studies of tabular data use fewer than 50 datasets or do not properly tune baselines [44, 57], putting the generalizability of these findings into question. Furthermore, the bottom line of many prior works is to answer the question, 'which performs better, NNs or GBDTs, in terms of the average rank across datasets' without searching for more fine-grained insights.

[*]Correspondence to: `{duncan, colin}@abacus.ai`. Work done while SK and JV were at Abacus.AI.



Algorithms
Datasets
19 Algorithms
Neural nets, GBDTs, baselines
176 Datasets
OpenML Suite, CC-18, addtl. OpenML
500K total evaluations
"NNs vs. GBDTs"
NNs
GBDTs
Baselines
No family dominates
ΔHPO
ΔAlgo. selection
HPO is just as important as choosing NNs/GBDTs
Metafeatures
Irregularity
Size
GBDTs handle irregularity better
Neural nets handle small datasets better
TabZilla Suite
36 'Hardest' Datasets
No datasets s.t. baselines win
No datasets s.t. many methods win
Large variety
10 folds per dataset

Figure 1: Overview of our work. We start by conducting the largest study on tabular data to date (left); we analyze the importance of algorithm selection ('NNs vs. GBDTs') as well as metafeatures (middle); and based on our study, we release TabZilla, a collection of the hardest tabular datasets.

In this work, we take a completely different approach by focusing on the following points. First, we question the importance of the 'NN vs. GBDT' debate, by investigating the significance of algorithm selection. Second, we analyze what *properties* of a dataset make NNs or GBDTs better-suited to perform well. We take a data-driven approach to answering these questions, conducting the largest tabular data analysis to date, by comparing **19 algorithms each with up to 30 hyperparameter settings, across 176 datasets**, including datasets from the OpenML-CC18 suite [6] and the OpenML Benchmarking Suite [25]. To assess performance differences across datasets, we consider dozens of metafeatures. We use 10 folds for each dataset to further reduce the uncertainty of our results.

We find that for a surprisingly high fraction of datasets, either a simple baseline (such as a decision tree or KNN) performs on par with the top algorithms; furthermore, for roughly one-third of all datasets, light hyperparameter tuning on CatBoost or ResNet increases performance more than choosing among GBDTs and NNs. These results show that for many tabular datasets, it is not necessary to try out many different NNs and GBDTs: **in many cases, a strong baseline or a well-tuned GBDT will suffice**. While NNs are the best approach for a non-negligible fraction of the datasets in this study, we do find that GBDTs outperform NNs on average over all datastes.

Next, we run analyses to discover what *properties* of datasets explain which methods, or families of methods, do or do not succeed. We compute the correlations of various metafeatures with algorithm performance, and we demonstrate that these correlations are predictive. Our main findings are as follows (also see Figure 5): **dataset *regularity* is predictive of NNs outperforming GBDTs** (for example, feature distributions that are less skewed and less heavy-tailed). Furthermore, GBDTs tend to perform better on larger datasets.

Finally, with the goal of accelerating tabular data research, we release the **TabZilla Benchmark Suite**: a collection of the 'hardest' of the 176 datasets we studied. We select datasets on which a simple baseline does not win, as well as datasets such that most algorithms do not reach top performance.

Our work provides a large set of tools for researchers and practitioners working on tabular data. We provide the largest open-source codebase of algorithms and datasets in one interface, together with a set of the 'hardest' datasets, and raw results (over 500K trained models) at `https://github.com/naszilla/tabzilla` for researchers and practitioners to more easily compare methods. Finally, our metafeature insights can be used by researchers, to uncover the failure modes of tabular algorithms, and by practitioners, to help determine which algorithms will perform well on a new dataset.

Our contributions. We summarize our main contributions below:

- We conduct the largest analysis of tabular data to date, comparing 19 methods on 176 datasets, with more than half a million models trained. We show that for a surprisingly high fraction of datasets, either a simple baseline performs the best, or light hyperparameter tuning of a GBDT is more important than choosing among NNs and GBDTs, suggesting that the 'NN vs. GBDT' debate is overemphasized.

- After analyzing dozens of metafeatures, we present a number of insights into the properties that make a dataset better-suited for GBDTs or NNs.
- We release the TabZilla Suite: a collection of 36 'hard' datasets, with the goal of accelerating tabular data research. We open-source our benchmark suite, codebase, and raw results.

Related work. Tabular datasets are the oldest and among the most common dataset types in machine learning in practice [7, 55], due to their wide variety of applications [5, 10, 13, 36, 50]. GBDTs [21] iteratively build an ensemble of decision trees, with each new tree fitting the residual of the loss from the previous trees, using gradient descent to minimize the losses. XGBoost [12], LightGBM [38], and Catboost [48] are three widely-used, high-performing variants. Borisov et al. described three types of tabular data approaches for neural networks [7]: data transformation methods [29, 64], architecture-based methods [4, 11, 28, 47] (including transformers [26, 34, 56]), and regularization-based methods [35, 37, 54]. Several recent works compare GBDTs to NNs on tabular data, often finding that *either* NNs [4, 26, 37, 47] *or* GBDTs [7, 26, 27, 55] perform best.

Perhaps the most related work to ours is by Grinsztajn et al. [27], who investigate why tree-based methods outperform neural nets on tabular data. There are a few differences between their work and ours. First, they consider seven algorithms and 45 datasets, compared to our 19 algorithms and 176 datasets. Second, their dataset sizes range from 3 000 to 10 000, or seven that are exactly 50 000, in contrast to our dataset sizes which range from 32 to 1 025 009 (see Table 6). Additionally, they further control their study, for example by upper bounding the ratio of size to features, by removing high-cardinality categorical feature, and by removing low-cardinality numerical features. While this has the benefit of being a more controlled study, their analysis misses out on some of our observations, such as GBDTs performing better than NNs on 'irregular' datasets. Finally, while Grinsztajn et al. focused in depth on a few metafeatures such as dataset smoothness and number of uninformative features, our work considers orders of magnitude more metafeatures. Again, while each approach has its own strengths, our work is able to discover more potential insights, correlations, and takeaways for practitioners. To the best of our knowledge, the only related work has considered more than 50 datasets is TabPFN [32], which considered 179 datasets which are size 2 000 or smaller. See Appendix C for a longer discussion of related work.

2 Analysis of Algorithms for Tabular Data

In this section, we present a large-scale study of techniques for tabular data across a wide variety of datasets. Our analysis seeks to answer the following two questions.

1. How do algorithms (and algorithm families) compare across a wide variety of datasets?
2. What properties of a dataset are associated with algorithms (and families) outperforming others?

Algorithms and datasets implemented. We present results for 19 algorithms, including popular recent techniques and baselines. The methods include three GBDTs: CatBoost [48], LightGBM [38], and XGBoost [12]; 11 neural networks: DANet [11], FT-Transformer [26], two MLPs [26], NODE [47], ResNet [26], SAINT [56], STG [63], TabNet [4], TabPFN [32], and VIME [64]; and five baselines: Decision Tree [49], KNN [16], Logistic Regression [17], Random Forest [42], and SVM [15]. We choose these algorithms because of their popularity, diversity, and strong performance.

We run the algorithms on 176 classification datasets from OpenML [60]. Our aim is to include most classification datasets from popular recent papers that study tabular data [7, 26, 37, 55], including datasets from the OpenML-CC18 suite [6], the OpenML Benchmarking Suite [25], and additional OpenML datasets. Due to the scale of our experiments (538 650 total models trained), we limit the run-time for each experiment (described below), which precluded the use of datasets of size larger than 1.1M. Table 6 shows summary statistics for all datasets. CC-18 and OpenML Benchmarking Suite are both seen as the go-to standards for conducting a fair, diverse evaluation across algorithms due to their rigorous selection criteria and wide diversity of datasets [6, 25]. To the best of our knowledge, our 19 algorithms and 176 datasets are the largest number of *either* algorithms *or* datasets (with the exception of TabPFN [32]) considered by recent tabular dataset literature, and the largest number available in a single open-source repository.

Table 1: Performance of algorithms across 104 datasets (see Table 11 for extended results). Columns show the algorithm family (GBDT, NN, or baseline), rank over all datasets, the average normalized accuracy (Mean Acc.), the std. dev. of normalized accuracy across folds (Std. Acc.), and the train time in seconds per 1000 instances. Min/max/mean/median of these quantities are taken over all datasets.

		Rank				Mean Acc.		Std. Acc.		Time /1000 inst.	
Algorithm	Class	min	max	mean	med.	mean	med.	mean	med.	mean	med.
CatBoost	GBDT	1	17	5.06	4	0.89	0.94	0.31	0.22	30.27	2.22
XGBoost	GBDT	1	16	6.38	6	0.83	0.91	0.34	0.22	0.94	0.42
ResNet	NN	1	18	6.87	7	0.77	0.84	0.31	0.21	16.07	10.04
SAINT	NN	1	18	7.15	6	0.75	0.87	0.32	0.23	168.14	144.84
NODE	NN	1	18	7.35	7	0.75	0.85	0.27	0.19	146.89	118.94
FT-Transformer	NN	1	17	7.60	7	0.76	0.82	0.32	0.21	29.47	18.63
RandomForest	base	1	18	7.66	7	0.78	0.84	0.33	0.22	0.36	0.25
LightGBM	GBDT	1	18	7.80	8	0.78	0.85	0.37	0.22	1.06	0.37
SVM	base	1	17	8.34	9	0.70	0.80	0.27	0.20	29.22	1.37
DANet	NN	1	17	8.87	9	0.75	0.80	0.33	0.22	69.29	60.15
MLP-rtdl	NN	1	18	9.59	10	0.65	0.74	0.29	0.15	14.33	7.62
STG	NN	1	18	10.88	11	0.58	0.66	0.30	0.17	18.47	16.00
DecisionTree	base	1	18	10.91	12	0.61	0.68	0.36	0.25	0.03	0.01
LinearModel	base	1	18	11.31	13	0.53	0.58	0.32	0.24	0.04	0.03
MLP	NN	1	18	11.35	12	0.57	0.57	0.30	0.19	18.61	11.92
TabNet	NN	1	18	11.87	13	0.56	0.62	0.40	0.23	35.09	30.83
KNN	base	1	18	12.96	14	0.46	0.52	0.29	0.23	0.01	0.00
VIME	NN	2	18	14.27	16	0.36	0.32	0.27	0.17	17.10	14.97

Metafeatures. We extract metafeatures using the Python library PyMFE [3], which contains 965 metafeatures. The categories of metafeatures include: 'general' (such as number of datapoints, classes, or numeric/categorical features), 'statistical' (such as the min, mean, or max skewness, or kurtosis, of all feature distributions), 'information theoretic' (such as the Shannon entropy of the target), 'landmarking' (the performance of a baseline such as 1-Nearest Neighbor on a subsample of the dataset), and 'model-based' (summary statistics for some model fit on the data, such as number of leaf nodes in a decision tree model). Since some of these features have long-tailed distributions, we also include the log of each strictly-positive metafeature in our analysis.

Experimental design. For each dataset, we use the ten train/test folds provided by OpenML, which allows our results on the test folds to be compared with other works that used the same OpenML datasets. Since we also need validation splits in order to run hyperparameter tuning, we divide each training fold into a training and validation set. For each algorithm, and for each dataset split, we run the algorithm for up to 10 hours. During this time, we train and evaluate the algorithm with at most 30 hyperparameter sets (one default set and 29 random sets, using Optuna [2]). Each parameterized algorithm is given at most two hours on a 32GiB V100 to complete a single train/evaluation cycle. In line with prior work, our main metric of interest is *accuracy*, and we report the test performance of the hyperparameter setting that had the maximum performance on the validation set. We also consider log loss, which is highly correlated with accuracy but contains significantly fewer ties. We also include results for F1-score and ROC AUC in Appendix D. Similar to prior work [20, 62], whenever we average across datasets, we use the average distance to the minimum (ADTM) metric, which consists of 0-1 scaling (after selecting the best hyperparameters, which helps protect against outliers [27]). Finally, in order to see the variance of each method on different folds of the same dataset, we report the average (scaled) standard deviation of each method across all 10 folds.

2.1 Relative Algorithm Performance

In this section, we answer the question, "How do individual algorithms, and families of algorithms, perform across a wide variety of datasets?" We especially consider whether the difference between GBDTs and NNs is significant.

No individual algorithm dominates. We start by comparing the average rank of all algorithms across all datasets, while excluding datasets which ran into memory or timeout issues on a nontrivial number of algorithms using the experimental setup described above. Therefore, we consider a set

Table 2: Performance of algorithms across 57 datasets of size less than or equal to 1250. Columns show the rank over all datasets, the average normalized accuracy (Mean Acc.), the standard deviation of normalized accuracy across folds (Std. Acc), and the train time per 1000 instances. Min/max/mean/median are taken over all datasets.

Algorithm	Rank min	max	mean	med.	Mean Acc. mean	med.	Std. Acc. mean	med.	Time /1000 inst. mean	med.
TabPFN	1	18	4.88	3	0.84	0.93	0.35	0.26	0.00	0.00
CatBoost	1	18	5.37	4	0.85	0.91	0.39	0.30	26.22	2.75
ResNet	1	19	6.75	6	0.77	0.79	0.42	0.30	23.67	13.87
RandomForest	1	18	7.65	7	0.76	0.82	0.40	0.29	0.47	0.32
SAINT	1	19	7.67	6	0.74	0.87	0.42	0.31	197.41	181.62
FTTransformer	1	18	7.93	7	0.75	0.78	0.42	0.32	32.93	26.39
XGBoost	1	17	8.30	8	0.74	0.80	0.42	0.30	0.95	0.61
NODE	1	19	8.35	8	0.73	0.75	0.36	0.28	173.55	144.45
SVM	1	18	9.54	11	0.68	0.72	0.35	0.28	23.90	0.42
MLP-rtdl	1	19	9.77	10	0.64	0.69	0.39	0.31	21.48	12.21
LightGBM	1	19	10.00	10	0.68	0.71	0.45	0.38	0.64	0.23
LinearModel	1	19	10.21	11	0.61	0.71	0.38	0.29	0.06	0.05
DANet	1	18	10.74	10	0.68	0.69	0.41	0.34	83.57	71.19
DecisionTree	1	19	11.44	13	0.60	0.67	0.45	0.32	0.02	0.01
MLP	1	19	11.49	13	0.57	0.54	0.38	0.30	27.88	16.81
STG	1	19	11.49	12	0.57	0.64	0.40	0.34	21.22	18.24
KNN	1	19	13.12	15	0.46	0.51	0.38	0.32	0.00	0.00
TabNet	3	19	14.54	16	0.42	0.40	0.52	0.49	41.83	34.35
VIME	2	19	14.88	17	0.33	0.27	0.36	0.29	18.95	16.43

of 104 datasets (and we include results on all 176 datasets in the next section and in Appendix D.2). As mentioned in the previous section, for each algorithm and dataset split, we report the test set performance after tuning on the validation set; see Table 1.

Surprisingly, *nearly every algorithm ranks first on at least one dataset and last on at least one other dataset*. As expected, baseline methods tend to perform poorly while neural nets and GBDTs tend to perform better on average. The fact that the best out of all algorithms, CatBoost, only achieved an average rank of 5.06, shows that there is not a single approach that dominates across most datasets. In Table 2, we compute the same table for the 57 datasets with size at most 1250 (training set at most 1000), so that we can include TabPFN [32] in our rankings. **We find that TabPFN achieves the best average performance of all algorithms, while also having the fastest training time.** However, with an average rank of 4.88, it still does not dominate all other approaches across different datasets. Furthermore, the *inference* time for TabPFN is higher than other algorithms.

Performance vs. runtime. In Figure 2, we plot the accuracy vs. runtime for all algorithms, averaged across all datasets. Overall, neural nets require the longest runtime, and often outperform baseline methods. On the other hand, GBDTs simultaneously require little runtime while also achieving strong performance: they consistently outperform baseline methods, and consistently require less runtime than neural nets. There is one caveat: our experiments train each neural net for a pre-determined number of epochs (100 in most cases), with early stopping if there is no improvement for 20 epochs. It is possible that these neural nets can achieve strong performance with less runtime, e.g., with a more aggressive early-stopping criterion.

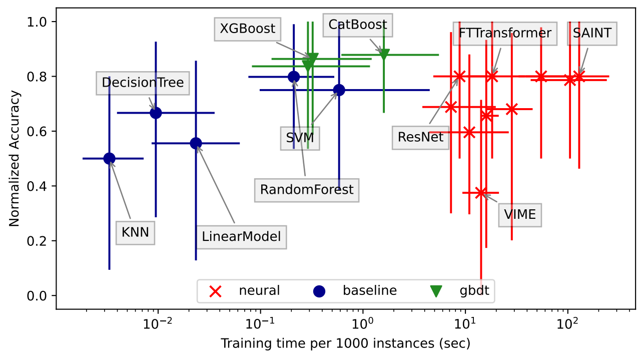


Figure 2: Median runtime vs. median normalized accuracy for each algorithm, over 104 datasets. The bars span the 20th to 80th percentile over all datasets.

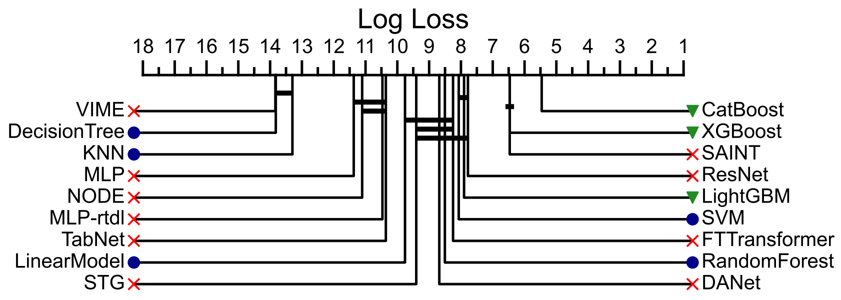
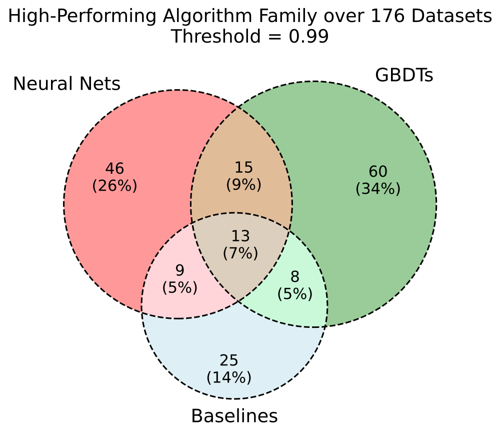


Figure 3: Critical difference plot comparing all algorithms according to their mean log loss rank over 104 datasets. Each algorithm's average rank is shown as a horizontal line on the axis. Sets of algorithms which are *not significantly different* are connected by a horizontal black bar. Algorithm family is indicated by a marker next to the algorithm name: red "X" indicates a neural net, blue circle indicates a baseline algorithm, and green triangles indicate GBDTs.



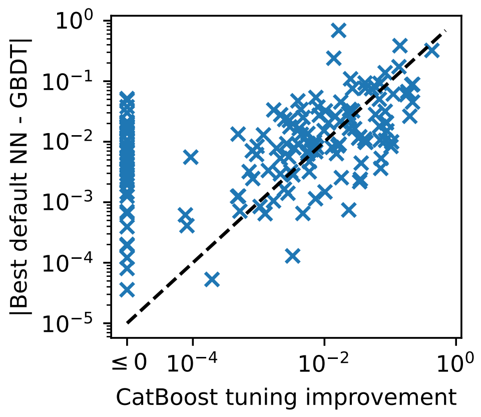


Figure 4: Left: Venn diagram of the number datasets where each algorithm is 'high-performing' for each algorithm class, over all 176 datasets. An algorithm is high-performing if its test accuracy after 0-1 scaling is at least 0.99 (we show 0.9999 in Appendix D.2). Right: the performance improvement of hyperparameter tuning on CatBoost, compared to the absolute performance difference between the best neural net and the best GBDT using default hyperparameters. Each point indicates the normalized log loss of one dataset, Points on or below the dotted line indicate that the performance improvement due to tuning is as high as the difference between NN-GBDT algorithm selection.

Statistically significant performance differences. Table 1 shows that many algorithms have similar performance. Next, we determine *statistically significant* ($p < 0.05$) performance differences between algorithms across the 104 datasets described above. First, we use a Friedman test to determine whether performance differences between each algorithm are significant [22]; we can reject the null hypothesis (p$<$ 0.05) for this test; the p-value is less than 10^{-20}. We then use a Wilcoxon signed-rank test to determine which pairs of algorithms have significant performance differences (p$<$0.05) [14]. With the Wilcoxon tests we use a Holm-Bonferroni correction to account for multiple comparisons [33]. Due to the presence of many ties in the test accuracy metric, we use the test log loss metric. See Figure 3 (and see Figure 7 for the F1 score). In these figures, the average rank of each algorithm is shown on the horizontal axis; if differences between algorithms are *not significant* (p$\geq$ 0.05), then algorithms are shown connected by a horizontal bar. We find that CatBoost outperforms all other algorithms across 104 datasets.

GBDTs vs. NNs. Although Table 11 tells us which *individual* methods perform best on average, now we consider the age-old question, 'are GBDTs better than NNs for tabular data?' We split the 19 algorithms into three *families*: GBDTs (CatBoost, XGBoost, LightGBM), NNs (the 11 listed above), and baselines (Decision Tree, KNN, LinearModel, RandomForest, SVM). We say that an algorithm is ‘high-performing’ if it achieves a 0-1 scaled test accuracy of at least 0.99, and then we determine which algorithm families (GBDTs, NNs, baselines) have a high-performing algorithm; see Figure 4. Surprisingly, the three-way Venn diagram is relatively balanced among GBDTs, NNs, and baselines, although GBDTs overall have the edge. In Appendix D.2, we run the same analysis, using a threshold of 0.9999. In this case, GBDTs are the sole high-performing algorithm family for most datasets. Since these wins are by less than 0.01%, they may not be significant to practitioners.

Algorithm selection vs. tuning. Next, we determine whether it is more important to select the best possible algorithm family, or to simply run light hyperparameter tuning on an algorithm that performs well in general, such as CatBoost or ResNet. We consider a scenario in which a practitioner can decide to *(a)* test several algorithms using their default hyperparameters, or *(b)* optimize the hyperparameters of a single model, such as CatBoost or ResNet. We compute whether *(a)* or *(b)* leads to better performance. Specifically, we measure the performance difference between the best-performing GBDT and NN using their default hyperparameters, as well as the performance difference between CatBoost with the default hyperparameters vs. CatBoost tuned via 30 iterations of random search on a validation set; see Figure 4 (right), and see Appendix D.2 for the same analysis with ResNet. Surprisingly, light hyperparameter tuning yields a greater performance improvement than GBDT-vs-NN selection for about one-third of all datasets. Once again, this suggests that for a large fraction of datasets, it is not necessary to determine whether GBDTs or NNs are better: light tuning on an algorithm such as CatBoost or ResNet can give just as much performance gain. In the next section, we explore *why* a dataset might be more amenable to a neural net or a GBDT.

2.2 Metafeature Analysis

In this section, we answer the question, “What properties of a dataset are associated with certain techniques, or families of techniques, outperforming others?” We answer this question by computing the correlation of metafeatures with three different quantities related to the performance difference between algorithm families, the performance difference between pairs of algorithms, and the relative performance of individual algorithms.

In order to assess the difference in performance between NNs and GBDTs, we calculate the difference in normalized log loss between the best NN and the best GBDT, which we refer to as $\Delta_{\ell\ell}$. We compute the correlation of $\Delta_{\ell\ell}$ to various metafeatures across all datasets; see Figure 12 and Table 14 in Appendix D.3. Next, in order to determine the individual strengths and weaknesses of each algorithm, we compute the correlation of various metafeatures to the performance of an individual algorithm relative to all other algorithms; see Table 3 and Figure 5. Finally, we compute the correlation of metafeatures to the difference in performance between pairs of the top-performing algorithms from Section 2.1: CatBoost, XGBoost, SAINT, and ResNet. See Figure 5 and Table 19.

In order to show that these metafeatures are *predictive*, we train and evaluate a meta-learning model using a leave-one-out approach: one dataset is held out for testing, while the remaining 175 datasets are used for training, averaged across all 176 possible test sets; see Appendix D.3. In the rest of this section, we state and discuss the main findings of our metafeature analysis.

Neural nets perform comparatively worse on larger datasets. Throughout our metafeature analyses, we find that GBDTs perform comparatively better than NNs and baselines with larger datasets. Figure 5 shows that XGBoost achieves top performance compared to all 19 algorithms on the seven largest datasets, and GBDTs overall perform well. In Table 3, somewhat surprisingly, dataset size is the most negatively-correlated metafeature with the relative performance of both LightGBM and XGBoost. Finally, Table 14 shows that the GBDT family’s performance is also positively correlated with the *ratio* of size to the number of features. Notably, all of these analyses are relative to the performance of all algorithms, which includes the newly released TabPFN [32], a neural net that performs remarkably well on datasets of small size, due to its carefully-designed prior. On the other hand, GBDTs excel when the ratio of dataset size to number of features is high, because all split in the decision trees are computed using more datapoints. Some of our above findings are backed up by prior work, for example, Grinsztajn et al. [27] showed that increasing the ratio of (uninformative) features to dataset size, hurts the performance of ResNet, and our results indicate the same trend (Table 20). On the other hand, NNs as a whole see the opposite trend (Table 14), which at least is shown for FTTransformer in Grinsztajn et al. [27]. We provide additional analyses in Appendix D.2.6. Note that, while the general trend shows GBDTs outperforming NNs on larger

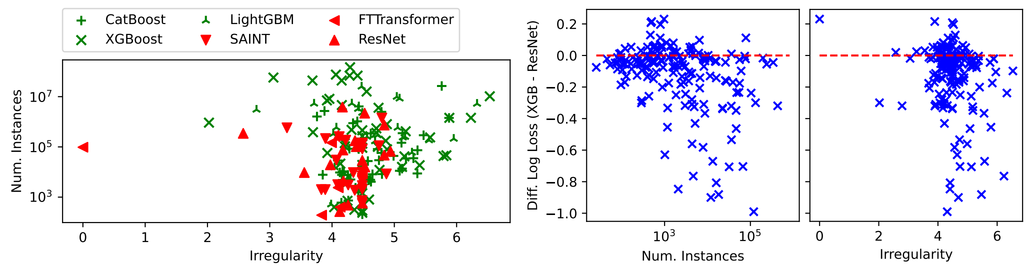


Figure 5: Left: scatterplot of the best algorithm on all 176 datasets across metafeatures. The vertical axis indicates the dataset size, and the horizontal axis combines five dataset metafeatures related to irregularity. Right: scatterplot of the difference in normalized log loss between XGBoost and ResNet, by dataset size (middle subplot) and irregularity (right subplot). The irregularity feature is a linear combination of five standardized dataset attributes: the minimum eigenvalue of the feature covariance matrix (-0.33), the skewness of the standard deviation of all features (0.23), the skewness of the range of all features (0.22), the interquartile range of the harmonic mean of all features (0.21), and the standard deviation of the kurtosis of all features (0.21).

Table 3: Metafeatures that are most correlated with the performance of each top-performing algorithm, calculated as the Pearson correlation between the metafeature and normalized log loss over 176 datasets. Metafeatures with the largest absolute Pearson correlation with algorithm performance are shown, and correlations are presented as 95% confidence intervals, calculated using the Fisher transformation.

Alg.	Metafeature Description	Corr.
CatBoost	Noisiness of the features: $(\sum_i S_i - \sum_i MI(i, y))/\sum_i MI(i, y)$, where S_i is the entropy of feature i, and $MI(i, y)$ is the mutual information between feature i and the target y.	[0.25, 0.34]
XGBoost	Log. number of instances.	[-0.27, -0.18]
LightGBM	Log. number of instances.	[-0.36, -0.28]
ResNet	Mean canonical correlation between any numeric feature and the target.	[-0.28, -0.19]
FTTransformer	Number of target classes.	[0.23, 0.32]
SAINT	Noisiness of the features.	[0.19, 0.29]

datasets, this does not imply that all GBDTs are better than all NNs on larger datasets. For example, TabPFN and TabNet are both neural nets, yet TabPFN performs particularly well on smaller datasets, and TabNet performs particularly well on larger datasets. **It is important to note that when choosing an algorithm for a new use-case, practitioners should focus on algorithm-specific analyses (such as in Table 1, Table 2, and Appendix D.2.6) rather than general 'GBDT vs. NN' trends.**

GBDTs favor irregular datasets. Another trend present throughout all three metafeature analyses is that GBDTs consistently favor 'irregular' datasets. When comparing pairs of the top GBDTs and NNs, we find that both CatBoost and XGBoost outperform ResNet and SAINT on datasets whose feature distributions are heavy-tailed, skewed, or have high variance (see Table 19 for the full details). That is, some datasets' feature distributions all have a similar amount of skewness, while other datasets' feature distributions are more irregular, with a high range of skewness. It is the latter type of datasets on which GBDTs outperform NNs. We also find that GBDTs perform better when datasets are more class imbalanced (on SAINT in particular). In Figure 5, GBDTs perform best on the most irregular datasets, computed via a linear combination of five metafeatures each measuring the skewness or kurtosis of the feature distributions.

The bottom line. Overall, we answer the title question of our paper: GBDTs outperform NNs on datasets that are more 'irregular', as well as large datasets, and datasets with a high ratio of size to number of features. When a practitioner is faced with a new dataset, based on all the analysis in Section 2, we give the following recommendation: first try simple baselines, and then conduct light hyperparameter tuning on CatBoost. Surprisingly, this often will already result in strong performance. As a next step, the practitioner can try NNs and other GBDTs that are most-correlated with strong performance based on the dataset's metafeatures, using analyses such as Table 1 and Appendix D.2.6.

Table 4: The TabZilla Benchmark Suite. Columns show the hardness metrics used as selection criteria, dataset attributes, and the top-performing algorithms. 'Std. Kurtosis' indicates the std. dev. of the kurtosis of all features. Hardness metrics that meet our selection criteria are shown in bold.

	Hardness Metrics			Dataset Attributes			Top 3 Algs.		
Dataset	base	4th-best	GBDT	N	# feats.	Std. Kurtosis	1st	2nd	3rd
credit-g	**0.26**	**0.13**	**0.12**	1 000	21	1.92	ResNet	FTTransformer	CatBoost
jungle-chess	**0.30**	**0.18**	**0.17**	44 819	7	0.08	SAINT	TabNet	LightGBM
MiniBooNE	**0.20**	**0.09**	0.00	130 064	51	12162.65	LightGBM	XGBoost	CatBoost
albert	**0.42**	**0.28**	0.00	425 240	79	1686.90	CatBoost	XGBoost	ResNet
electricity	**0.46**	**0.38**	0.00	45 312	9	2693.51	LightGBM	XGBoost	FTTransformer
elevators	**0.36**	**0.08**	0.05	16 599	19	2986.50	TabNet	XGBoost	CatBoost
guillermo	**0.35**	**0.60**	0.00	20 000	4 297	NaN	XGBoost	RandomForest	TabNet
higgs	**0.41**	**0.10**	0.07	98 050	29	15.53	ResNet	XGBoost	LightGBM
nomao	**0.22**	**0.18**	0.00	34 465	119	1100.34	LightGBM	XGBoost	CatBoost
100-plants-texture	**0.20**	**0.11**	0.00	1 599	65	17.66	CatBoost	XGBoost	ResNet
poker-hand	**0.58**	**0.98**	0.00	1 025 009	11	0.08	XGBoost	CatBoost	KNN
profb	**0.39**	**0.38**	0.00	672	10	0.95	CatBoost	DeepFM	MLP-rtdl
socmob	**0.24**	**0.10**	0.00	1 156	6	NaN	XGBoost	CatBoost	ResNet
audiology	**0.43**	0.03	0.00	226	70	NaN	STG	XGBoost	ResNet
splice	**0.30**	0.03	0.00	3 190	61	NaN	LightGBM	XGBoost	CatBoost
vehicle	0.05	**0.10**	**0.10**	846	19	15.16	TabPFN	SVM	DANet
Australian	0.15	**0.08**	0.00	690	15	2.00	CatBoost	XGBoost	TabPFN
Bioresponse	0.07	**0.07**	0.00	3 751	1 777	328.77	LightGBM	XGBoost	CatBoost
GesturePhase	0.08	**0.08**	0.00	9 872	33	52.18	LightGBM	XGBoost	CatBoost
SpeedDating	0.18	**0.14**	0.00	8 378	121	36.43	XGBoost	CatBoost	LightGBM
ada-agnostic	0.12	**0.11**	0.00	4 562	49	NaN	XGBoost	CatBoost	LightGBM
airlines	**0.20**	**0.18**	0.00	539 382	8	2.01	LightGBM	XGBoost	CatBoost
artificial-characters	0.13	**0.11**	0.00	10 218	8	0.63	XGBoost	LightGBM	CatBoost
colic	0.13	**0.11**	0.00	368	27	4.00	CatBoost	XGBoost	FTTransformer
credit-approval	0.12	**0.08**	0.00	690	16	74.77	CatBoost	TabPFN	XGBoost
heart-h	0.10	**0.07**	0.08	294	14	NaN	DeepFM	TabTransformer	NAM
jasmine	0.13	**0.13**	0.00	2 984	145	47.60	CatBoost	XGBoost	LightGBM
kc1	0.14	**0.07**	0.00	2 109	22	28.34	CatBoost	XGBoost	FTTransformer
lymph	0.14	**0.08**	0.00	148	19	17.04	XGBoost	DANet	SAINT
mfeat-fourier	0.00	**0.07**	0.07	2 000	77	0.64	SVM	SAINT	STG
phoneme	0.10	**0.15**	0.00	5 404	6	1.23	XGBoost	LightGBM	RandomForest
qsar-biodeg	0.08	**0.08**	0.05	1 055	42	93.24	TabPFN	CatBoost	SAINT
balance-scale	0.07	0.05	**0.16**	625	5	0.02	TabPFN	SAINT	MLP
cnae-9	0.11	0.04	**0.10**	1 080	857	NaN	TabTransformer	STG	MLP-rtdl
mfeat-zernike	0.00	0.04	**0.10**	2 000	48	1.42	SVM	DANet	ResNet
monks-problems-2	0.04	0.00	**0.17**	601	7	NaN	SAINT	ResNet	MLP-rtdl

3 TabZilla Benchmark Suite

In order to accelerate tabular data research, we release the TabZilla Benchmark Suite: a collection of the 36 'hardest' of the 176 datasets we studied in Section 2. We use the following three criteria.

Hard for baseline algorithms. As discussed in Section 2.1, simple baselines perform very well on a surprisingly large fraction of the datasets in our experiments. Therefore, to select our suite of hard datasets, we remove any dataset such that a baseline (as defined in Section 2) achieved a normalized log loss within 20% of the top-performing algorithm. This criterion is not perfect; for example, if a dataset is so hard that all 19 algorithms fail to reach non-trivial performance, it would not satisfy the criterion. However, the criterion is a good proxy for dataset hardness given the available information.

Hard for most algorithms. This criterion is designed to include datasets on which most algorithms were not able to reach top performance. In particular, a dataset satisfies the criterion if the fourth-best log loss out of 19 algorithms is at least 7% worse than the top log loss. In other words, this criterion will include datasets on which one, two, or three algorithms were able to stand out in terms of performance. For example, if ten algorithms were all able to achieve a performance within 7% of the top-performing algorithm, we can reasonably assume that the dataset might not be 'hard'. Interestingly, this criterion subsumes the majority of the datasets from the previous criterion.

Hard for GBDTs. The first two criteria result in datasets such that GBDTs primarily are the top-performing methods. This is not surprising in light of the overall performance of GBDTs that we showed in Section 2. However, for the field of tabular data to progress, focusing on datasets for which GBDTs already perform well would leave a blindspot on datasets for which GBDTs perform poorly. Therefore, we add all datasets for which GBDTs perform 10% worse than the top-performing algorithm, in order to achieve a greater diversity of datasets.

Table 5: Performance of algorithms across the Tabular Benchmark Suite of 36 datasets. Columns show the rank over all datasets, the average normalized accuracy (Mean Acc.), the standard deviation of normalized accuracy across folds (Std. Acc.), and the train time per 1000 instances. Min/max/mean/median of these quantities are taken over all datasets.

	Rank				Mean LL		Std. LL		Time /1000 inst.	
Algorithm	min	max	mean	med.	mean	med.	mean	med.	mean	med.
XGBoost	1	14	3.27	2.0	0.06	0.03	0.11	0.06	2.25	0.28
CatBoost	1	12	3.86	3.0	0.09	0.06	0.11	0.07	26.46	1.15
LightGBM	1	14	6.06	5.5	0.13	0.09	0.20	0.09	1.23	0.36
ResNet	1	13	6.14	5.5	0.19	0.17	0.12	0.07	8.24	5.30
SAINT	1	17	6.37	5.0	0.20	0.15	0.12	0.08	130.18	92.42
DANet	2	15	7.26	7.0	0.18	0.16	0.15	0.11	58.70	52.74
FTTransformer	2	14	7.66	7.0	0.25	0.20	0.13	0.12	17.41	12.64
RandomForest	2	16	8.67	8.0	0.29	0.28	0.22	0.08	0.42	0.25
MLP-rtdl	2	18	8.76	7.0	0.38	0.26	0.17	0.13	6.30	4.40
SVM	1	17	9.17	9.0	0.29	0.22	0.14	0.08	19.73	2.81
STG	1	17	9.28	10.0	0.32	0.24	0.11	0.06	15.97	15.33
TabNet	1	18	10.12	9.5	0.37	0.28	0.27	0.12	26.39	27.00
LinearModel	3	18	10.57	11.0	0.47	0.38	0.11	0.06	0.04	0.02
MLP	2	17	10.62	10.0	0.46	0.40	0.15	0.13	8.73	4.28
NODE	5	16	10.63	10.0	0.38	0.35	0.08	0.07	153.72	124.27
VIME	4	17	13.00	14.0	0.51	0.48	0.09	0.08	20.79	15.18
DecisionTree	5	18	13.19	14.5	0.61	0.61	0.38	0.21	0.20	0.01
KNN	3	18	14.55	16.0	0.70	0.74	0.36	0.21	0.03	0.00

TabZilla Characteristics. Table 4 shows all datasets, their statistics, and their top three algorithms. Based on the criteria alone, the dataset characteristics are diverse, with sizes ranging from 148 to over 1 million, as well as a large range of the variance of kurtosis of the features (one of our measures of irregularity). In Table 5, we compare the performance of all algorithms on the benchmark suite. The top five algorithms are XGBoost, CatBoost, LightGBM, ResNet, and SAINT, with mean ranks of 3.27, 3.86, 6.06, 6.14, and 6.37, respectively. In order to accelerate research in tabular data, we release TabZilla as a collection in OpenML, and we open-source all of our computed metafeatures and results. In Appendix B, we give the full dataset documentation, including a datasheet [23].

4 Conclusions and Future Work

In this work, we conducted the largest tabular data analysis to date, by comparing 19 approaches across 176 datasets. We found that the 'NN vs. GBDT' debate is overemphasized: for a surprisingly high number of datasets, either a simple baseline method performs on par with all other methods, or light hyperparameter tuning on a GBDT increases performance more than choosing the best algorithm. On the other hand, on average, GBDTs do outperform NNs. We also analyzed what properties of a dataset make NNs or GBDTs better-suited to perform well. For example, GBDTs are better than NNs at handling various types of data irregularity. Finally, based on our analysis, we released TabZilla, a collection of the 36 'hardest' out of the 176 datasets we studied: hard for baselines, most algorithms, and GBDTs. The goal in releasing TabZilla is to accelerate tabular data research by focusing on improving the current blind spots in the literature.

Our work provides a large set of tools to accelerate research on tabular data. For example, researchers developing a new neural net for tabular data can use our open-source repository to immediately compare their method to 19 algorithms across 176 datasets. Researchers can also use our metafeature analysis to improve the weaknesses of current or future algorithms; for example, making neural nets more robust to data irregularities is a natural next step. Furthermore, researchers studying ensemble methods can weight the models differently for each dataset based on its metafeatures. Finally, our open-source collection of extensive datasets and metafeatures can make it easier for researchers to design new meta-learned [32] or pre-trained models for tabular data [31, 41, 65]. There are several interesting ideas for extensions such as regression datasets, time-series forecasting datasets, studying uncertainty quantification, studying the effect of the percentage of categorical features on NNs, and studying more comprehensive hyperparameter optimization including regularization methods.

References

[1] Rishabh Agarwal, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E Hinton. Neural additive models: Interpretable machine learning with neural nets. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[3] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís P. F. Garcia, Jefferson T. Oliva, and André C. P. L. F. de Carvalho. Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research*, 21(111):1–5, 2020.

[4] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[5] Kumar Arun, Garg Ishan, and Kaur Sanmeet. Loan approval prediction based on machine learning approach. *IOSR J. Comput. Eng*, 18(3):18–21, 2016.

[6] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G Mantovani, Jan N van Rijn, and Joaquin Vanschoren. Openml benchmarking suites. *arXiv preprint arXiv:1708.03731*, 2017.

[7] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *arXiv preprint arXiv:2110.01889*, 2021.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Anna L Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[11] Jintai Chen, Kuanlun Liao, Yao Wan, Danny Z Chen, and Jian Wu. Danets: Deep abstract networks for tabular data classification and regression. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2022.

[12] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[13] Jillian M Clements, Di Xu, Nooshin Yousefi, and Dmitry Efimov. Sequential deep learning for credit risk monitoring with tabular financial data. *arXiv preprint arXiv:2012.15330*, 2020.

[14] William Jay Conover. *Practical nonparametric statistics*, volume 350. john wiley & sons, 1999.

[15] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 1995.

[16] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 1967.

[17] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 1958.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[19] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[20] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0: Hands-free automl via meta-learning. *The Journal of Machine Learning Research*, 2022.

[21] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[22] Milton Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 1937.

[23] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé Iii, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[24] E.S. Gelsema and L.N. Kanal. *Pattern Recognition in Practice IV: Multiple Paradigms, Comparative Studies and Hybrid Systems*. ISSN. Elsevier Science, 2014.

[25] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[26] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[27] Leo Grinsztajn, Edouard Oyallon, and Gael Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

[28] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. In *IJCAI*, 2017.

[29] John T Hancock and Taghi M Khoshgoftaar. Survey on categorical data for neural networks. *Journal of Big Data*, 7(1):1–41, 2020.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[31] Stefan Hegselmann, Alejandro Buendia, Hunter Lang, Monica Agrawal, Xiaoyi Jiang, and David Sontag. Tabllm: Few-shot classification of tabular data with large language models. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2023.

[32] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[33] Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[34] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

[35] Alan Jeffares, Tennison Liu, Jonathan Crabbé, Fergus Imrie, and Mihaela van der Schaar. Tangos: Regularizing tabular neural networks through gradient orthogonalization and specialization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[36] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[37] Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 34, 2021.

[38] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[39] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2012.

[40] Roman Levin, Valeriia Cherepanova, Avi Schwarzschild, Arpit Bansal, C Bayan Bruss, Tom Goldstein, Andrew Gordon Wilson, and Micah Goldblum. Transfer learning with deep tabular models. *ICLR*, 2023.

[41] Roman Levin, Valeriia Cherepanova, Avi Schwarzschild, Arpit Bansal, C Bayan Bruss, Tom Goldstein, Andrew Gordon Wilson, and Micah Goldblum. Transfer learning with deep tabular models. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2023.

[42] Andy Liaw, Matthew Wiener, et al. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[43] Guido Lindner and Rudi Studer. Ast: Support for algorithm selection with a cbr approach. In *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '99, page 418–423, Berlin, Heidelberg, 1999. Springer-Verlag.

[44] Zachary C Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship: Some ml papers suffer from flaws that could mislead the public and stymie future research. *Queue*, 2019.

[45] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the Annual Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1222–1230, 2013.

[46] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

[47] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2020.

[48] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Proceedings of the Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2018.

[49] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1986.

[50] Matthew Richardson, Ewa Dominowska, and Robert Ragno. Predicting clicks: estimating the click-through rate for new ads. In *Proceedings of the 16th international conference on World Wide Web*, pages 521–530, 2007.

[51] Ivan Rubachev, Artem Alekberov, Yury Gorishniy, and Artem Babenko. Revisiting pretraining objectives for tabular deep learning. *arXiv preprint arXiv:2207.03208*, 2022.

[52] Mostafa A Salama, Aboul Ella Hassanien, and Kenneth Revett. Employment of neural network and rough set in meta-learning. *Memetic Computing*, 5:165–177, 2013.

[53] Bernhard Schäfl, Lukas Gruber, Angela Bitto-Nemling, and Sepp Hochreiter. Hopular: Modern hopfield networks for tabular data. *arXiv preprint arXiv:2206.00664*, 2022.

[54] Ira Shavitt and Eran Segal. Regularization learning networks: deep learning for tabular datasets. *Advances in Neural Information Processing Systems*, 31, 2018.

[55] Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

[56] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

[57] Bojan Tunguz. Trouble with hopular, 2022.

[58] Dennis Ulmer, Lotta Meijerink, and Giovanni Cinà. Trust issues: Uncertainty estimation does not enable reliable ood detection on medical tabular data. In *Machine Learning for Health*, pages 341–354. PMLR, 2020.

[59] Christopher J Urban and Kathleen M Gates. Deep learning: A primer for psychologists. *Psychological Methods*, 2021.

[60] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 2014.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[62] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*, 2015.

[63] Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2020.

[64] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. Vime: Extending the success of self-and semi-supervised learning to tabular domain. *Advances in Neural Information Processing Systems*, 33:11033–11043, 2020.

[65] Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. Xtab: Cross-table pretraining for tabular transformers. *arXiv preprint arXiv:2305.06090*, 2023.

A Broader Societal Impact Statement

The goal of our work is to conduct an analysis of tabular data, including the significance of 'GBDTs vs. NNs', as well as to provide a large set of tools for researchers and practitioners working on tabular data. We do not see any negative broader societal impacts of our work. In fact, our work shows that on a surprisingly large fraction of datasets, it is not necessary to train a resource-intensive neural net: a simple baseline, or tuning CatBoost, is enough to reach top performance. We even predict which datasets are more amenable to GBDTs: larger datasets, datasets with a high size-to-number-of-features ratio, and 'irregular' datasets. Our hope is that our work will have a positive impact for both practitioners and researchers: by providing the largest analysis and open-source codebase to date, along with a benchmark suite of 'hard' datasets, our work can both accelerate future research and make the comparisons in future work more rigorous and comprehensive.

B Dataset Documentation

In this section, we give an overview of the documentation for our dataset. For the full details, including usage and tutorials, see `https://github.com/naszilla/tabzilla`.

B.1 Author responsibility and license

We, the authors, bear all responsibility in case of violation of rights. The license of our repository is the **Apache License 2.0**. For more information, see `https://github.com/naszilla/tabzilla/blob/main/LICENSE`.

B.2 Maintenance plan

The data is available on OpenML at `https://www.openml.org/search?type=study&study_type=task&id=379&sort=tasks_included`.

We plan to actively maintain the benchmark suite, and we welcome contributions from the community.

B.3 Code of conduct

Our Code of Conduct is from the Contributor Covenant, version 2.0. See `https://www.contributor-covenant.org/version/2/0/code_of_conduct.html`.
The policy is copied below.

> "We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, caste, color, religion, or sexual identity and orientation."

B.4 Datasheet

We include a datasheet [23] for TabZilla, which is available here and also at `https://github.com/naszilla/tabzilla`.

Motivation For Datasheet Creation

*Why was the datasheet created? (e.g., was there a specific task in mind? was there a specific gap that needed to be filled?) The goal of releasing the TabZilla Benchmark Suite is to accelerate research in tabular data by introducing a set of 'hard' datasets. Specifically, simple baselines cannot reach top performance, and most algorithms (out of the 19 we tried) cannot reach top performance. We found that a surprisingly high percentage of datasets used in tabular research today are such that a simple baseline can reach just as high accuracy as the leading methods.

*Has the dataset been used already? If so, where are the results so others can compare (e.g., links to published papers)? All of the individual datasets are already released in OpenML, and many have been used in prior work on tabular data. However, our work gathers these datasets into a single 'hard' suite.

*What (other) tasks could the dataset be used for? All of these datasets are tabular classification datasets, and so to the best of our knowledge, they cannot be used for anything other than tabular classification.

*Who funded the creation dataset? This benchmark suite was created by researchers at Abacus.AI, Stanford, Pinterest, University of Maryland, IIT Bombay, New York University, and Caltech. Funding for the dataset computation itself is from Abacus.AI.

*Any other comment? None.

Datasheet Composition

*What are the instances?(that is, examples; e.g., documents, images, people, countries) Are there multiple types of instances? (e.g., movies, users, ratings; people, interactions between them; nodes, edges) Each instance is a tabular datapoint. The makeup of each point depends on its dataset. For example, three of the datasets consist of poker hands, electricity usage, and plant textures.

*How many instances are there in total (of each type, if appropriate)? See Table 4 for a breakdown of the number of instances for each dataset.

*What data does each instance consist of ? "Raw" data (e.g., unprocessed text or images)? Features/attributes? Is there a label/target associated with instances? If the instances related to people, are subpopulations identified (e.g., by age, gender, etc.) and what is their distribution? The raw data is hosted on OpenML. In our repository, we also contain scripts for the standard preprocessing we ran before training tabular data models. The data are not related to people.

*Is any information missing from individual instances? If so, please provide a description, explaining why this information is missing (e.g., because it was unavailable). This does not include intentionally removed information, but might include, e.g., redacted text. There is no missing information.

*Are relationships between individual instances made explicit (e.g., users' movie ratings, social network links)? If so, please describe how these relationships are made explicit. There are no relationships between individual instances.

*Does the dataset contain all possible instances or is it a sample (not necessarily random) of instances from a larger set? If the dataset is a sample, then what is the larger set? Is the sample representative of the larger set (e.g., geographic coverage)? If so, please describe how this representativeness was validated/verified. If it is not representative of the larger set, please describe why not (e.g., to cover a more diverse range of instances, because instances were withheld or unavailable).

We selected the datasets for our benchmark suite as follows. We started with 176 datasets, which we selected with the aim to include most classification datasets from popular recent papers that study tabular data [7, 26, 37, 55], including datasets from the OpenML-CC18 suite [6], the OpenML Benchmarking Suite [25], and additional OpenML datasets [60]. Due to the scale of our experiments (538 650 total models trained), we limited to datasets smaller than 1.1M. CC-18 and OpenML Benchmarking Suite are both seen as the go-to standards for conducting a fair, diverse evaluation across algorithms due to their rigorous selection criteria and wide diversity of datasets [6, 25]. Out of these 176 datasets, we selected 36 datasets for our suite as described in Section 3.

*Are there recommended data splits (e.g., training, development/validation, testing)? If so, please provide a description of these splits, explaining the rationale behind them.

We use the 10 folds from OpenML, and it is recommended to report performance averaged over these 10 folds, as we do and as OpenML does. If a validation set is required, we recommend additionally using the validation splits that we used, described in Section 2.

*Are there any errors, sources of noise, or redundancies in the dataset? If so, please provide a description. There are no known errors, sources of noise, or redundancies.

*Is the dataset self-contained, or does it link to or otherwise rely on external resources (e.g., websites, tweets, other datasets)? If it links to or relies on external resources, a) are there guarantees that they will exist, and remain constant, over time; b) are there official archival versions of the complete dataset (i.e., including the external resources as they existed at the time the dataset was created); c) are there any restrictions (e.g., licenses, fees) associated with any of the external resources that might apply to a future user? Please provide descriptions of all external resources and any restrictions associated with them, as well as links or other access points, as appropriate. The dataset is self-contained.

Any other comments? None.

Collection Process

*What mechanisms or procedures were used to collect the data (e.g., hardware apparatus or sensor, manual human curation, software program, software API)? How were these mechanisms or procedures validated? We did not create the individual datasets. However, we selected the datasets for our benchmark suite as follows. We started with 176 datasets, which we selected with the aim to include most classification datasets from popular recent papers that study tabular data [7, 26, 37, 55], including datasets from the OpenML-CC18 suite [6], the OpenML Benchmarking Suite [25], and additional OpenML datasets [60]. Due to the scale of our experiments (538 650 total models trained), we limited to datasets smaller than 1.1M. CC-18 and OpenML Benchmarking Suite are both seen as the go-to standards for conducting a fair, diverse evaluation across algorithms due to their rigorous selection criteria and wide diversity of datasets [6, 25].

Out of these 176 datasets, we selected 36 datasets for our suite as described in Section 3.

*How was the data associated with each instance acquired? Was the data directly observable (e.g., raw text, movie ratings), reported by subjects (e.g., survey responses), or indirectly inferred/derived from other data (e.g., part-of-speech tags, model-based guesses for age or language)? If data was reported by subjects or indirectly inferred/derived from other data, was the data validated/verified? If so, please describe how. The datasets were selected using the three criteria from Section 3.

*If the dataset is a sample from a larger set, what was the sampling strategy (e.g., deterministic, probabilistic with specific sampling probabilities)? As described earlier, the datasets were selected using the three criteria from Section 3.

*Who was involved in the data collection process (e.g., students, crowdworkers, contractors) and how were they compensated (e.g., how much were crowdworkers paid)? The creation of the TabZilla Benchmark Suite was done by the authors of this work.

*Over what timeframe was the data collected? Does this timeframe match the creation timeframe of the data associated with the instances (e.g., recent crawl of old news articles)? If not, please describe the timeframe in which the data associated with the instances was created. The timeframe for constructing the TabZilla Benchmark Suite was from April 15, 2023 to June 1, 2023.

Data Preprocessing

*Was any preprocessing/cleaning/labeling of the data done (e.g., discretization or bucketing, tokenization, part-of-speech tagging, SIFT feature extraction, removal of instances, processing of missing values)? If so, please provide a description. If not, you may skip the remainder of the questions in this section.

We include both the raw data and the preprocessed data. We preprocessed the data by imputing each NaN to the mean of the respective feature. We left all other preprocessing (such as scaling) to the algorithms themselves.

*Was the "raw" data saved in addition to the preprocessed/cleaned/labeled data (e.g., to support unanticipated future uses)? If so, please provide a link or other access point to the "raw" data.

The raw data is available at `https://www.openml.org/search?type=study&study_type=task&id=379&sort=tasks_included`.

*Is the software used to preprocess/clean/label the instances available? If so, please provide a link or other access point.

Our README contains an extensive section on the data preprocessing, here: `https://github.com/naszilla/tabzilla#openml-datasets`.

*Does this dataset collection/processing procedure achieve the motivation for creating the dataset stated in the first section of this datasheet? If not, what are the limitations? We hope that the release of this benchmark suite will achieve our goal of accelerating research in tabular data, as well as making it easier for researchers and practitioners to devise and compare algorithms. Time will tell whether our suite will be adopted by the community.

*Any other comments None.

Dataset Distribution

*How will the dataset be distributed? (e.g., tarball on website, API, GitHub; does the data have a DOI and is it archived redundantly?) The benchmark suite is on OpenML at `https://www.openml.org/search?type=study&study_type=task&id=379&sort=tasks_included`.

*When will the dataset be released/first distributed? What license (if any) is it distributed under? The benchmark suite is public as of June 1, 2023, distributed under the Apache License 2.0.

*Are there any copyrights on the data? There are no copyrights on the data.

*Are there any fees or access/export restrictions? There are no fees or restrictions.

*Any other comments? None.

Dataset Maintenance

*Who is supporting/hosting/maintaining the dataset? The authors of this work are supporting/hosting/maintaining the dataset.

*Will the dataset be updated? If so, how often and by whom? We welcome updates from the tabular data community. If new algorithms are created, the authors may open a pull request to include their method.

*How will updates be communicated? (e.g., mailing list, GitHub) Updates will be communicated on the GitHub README `https://github.com/naszilla/tabzilla`.

*If the dataset becomes obsolete how will this be communicated? If the dataset becomes obsolete, it will be communicated on the GitHub README `https://github.com/naszilla/tabzilla`.

*If others want to extend/augment/build on this dataset, is there a mechanism for them to do so? If so, is there a process for tracking/assessing the quality of those contributions. What is the process for communicating/distributing these contributions to users? Others can create a pull request on GitHub with possible extensions to our benchmark suite, which will be approved case-by-case. For example, an author of a new hard tabular dataset may create a PR in our codebase with the new dataset. These updates will again be communicated on the GitHub README.

Legal and Ethical Considerations

*Were any ethical review processes conducted (e.g., by an institutional review board)? If so, please provide a description of these review processes, including the outcomes, as well as a link or other access point to any supporting documentation. There was no ethical review process. We note that our benchmark suite consists of existing datasets that are already publicly available on OpenML.

*Does the dataset contain data that might be considered confidential (e.g., data that is protected by legal privilege or by doctorpatient confidentiality, data that includes the content of individuals non-public communications)? If so, please provide a description. The datasets do not contain any confidential data.

*Does the dataset contain data that, if viewed directly, might be offensive, insulting, threatening, or might otherwise cause anxiety? If so, please describe why None of the data might be offensive, insulting, threatening, or otherwise cause anxiety.

*Does the dataset relate to people? If not, you may skip the remaining questions in this section. The datasets do not relate to people.

*Any other comments? None.

C Additional Related Work

Gradient-boosted decision trees. GBDTs iteratively build an ensemble of decision trees, with each new tree fitting the residual of the loss from the previous trees, using gradient descent to minimize the losses. GBDTs have been a powerful tool for modeling tabular data ever since their creation in 2001 [21], and numerous works propose high-performing GBDT variants. XGBoost (eXtreme Gradient Boosting) [12] uses weighted quantile sketching and sparsity-awareness, allowing it to scale to large datasets. LightGBM (Light Gradient Boosting Machine) [38] uses gradient-based one-sided sampling and exclusive feature bundling to create a faster and more lightweight GBDT implementation. CatBoost (Categorical Boosting) [48] introduces ordered boosting, a new method for handling categorical features, as well as better methods for handling missing values and outliers.

Neural networks for tabular data. Borisov et al. described three types of tabular data approaches for neural networks [7]. Data transformation methods [29, 64] seek to encode the data into a format that is better-suited for neural nets. Architecture-based methods design specialized neural architectures for tabular data [11, 28, 47], a large sub-class of which are transformer-based architectures [4, 26, 34, 56]. Regularization-based methods specially tailor regularizers methods to improve the performance of a given architecture [35, 37, 54]. Notably, one recent work designs a new framework for regularization in the tabular setting built on latent unit attributions [35], and another recent work shows that searching for the optimal combination/cocktail of 13 regularization techniques applied to a simple neural net achieves strong performance. While regularization was not the focus of our current work, including regularization methods would be an exciting direction for follow-up work.

GBDTs versus NNs. Several recent works compare GBDTs to NNs on tabular data, often finding that *either* neural nets [4, 26, 37, 47] or GBDTs [7, 26, 27, 55] perform best. Shwartz-Ziv and Armon compare GBDTs and NNs on 30 datasets, finding that GBDTs perform better on average, and ensembling both achieves better performance [55]. Kadra et al. compare GBDTs and NNs on 40 datasets, finding that properly-tuned neural networks perform best on average [37].

Gorishniy et al. [26] introduce a ResNet-like [30] architecture, and FT-Transformer, a transformer-based [61] architecture. Across experiments on eleven datasets, they conclude that there is still no universal winner among GBDTs and NNs. Borisov et al. [7] compare classical machine learning methods with eleven deep learning approaches on five tabular datasets, concluding that GBDTs still have the edge. In the transfer learning setting, Levin et al. [40] find that neural networks have a decisive edge over GBDTs when pre-training data is available.

Perhaps the most related work to ours is by Grinsztajn et al. [27], who investigate why tree-based methods outperform neural nets on tabular data. There are a few differences between their work and ours. First, they only consider seven algorithms and 45 datasets, compared to our 19 algorithms and 176 datasets. Second, their dataset sizes range from 3 000 to 10 000, or seven that are exactly 50 000, in contrast to our dataset sizes which range from 32 to 1 025 009 (see Table 6). Additionally, they

Table 6: Summary statistics for all 176 datasets used in our experiments. Left columns show the number of instances, number of features, number of target classes, and the ratio of the mimum frerquency of any class to the maximum frequency of any class (Min-Max Class Freq.). Right columns show the number of feature types. All statistics except for class frequency ratio are rounded to the nearest integer.

	# Inst.	# Feats.	# Classes	Min-Max Class Freq.	# Feature Types		
					Num.	Bin.	Cat.
mean	30567	223	6	0.48	206	25	17
std	106943	786	12	0.35	781	144	119
min	32	2	2	2e-05	0	0	0
25%	596	9	2	0.14	4	0	0
50%	2218	21	2	0.46	10	0	0
75%	11008	61	6	0.82	50	2	8
max	1025009	7200	100	1.00	7200	1555	1555

further control their study, for example by upper bounding the ratio of size to features, by removing high-cardinality categorical feature, and by removing low-cardinality numerical features. While this has the benefit of being a more controlled study, their analysis misses out on some of our observations, such as GBDTs performing better than NNs on ‘irregular’ datasets. Finally, while Grinsztajn et al. focused in depth on a few metafeatures such as dataset smoothness and number of uninformative features, our work considers orders of magnitude more metafeatures. Again, while each approach has its own strengths, our work is able to discover more potential insights, correlations, and takeaways for practitioners.

D Additional Experiments

We give additional experiments, including dataset statistics (Appendix D.1), additional results from Section 2.1 (Appendix D.2), and additional results from Section 2.2 (Appendix D.3).

D.1 Dataset statistics

Table 6 shows summary statistics for all 176 datasets used in our experiments. Roughly half of our datasets have a binary classification target (and as many as 100 target classes), and roughly half of all training sets have fewer than 2300 instances—though many datasets have tens of thousands of instances.

D.2 Additional experiments from Section 2.1

In this section, we give additional experiments from Section 2.1, including relative performance tables, training time analysis, and critical difference diagrams.

D.2.1 Relative performance tables

First, we show the ranking of all tuned algorithms according to each performance metric, averaged over datasets: log loss (Table 7), F1 score (Table 8), and ROC-AUC (Table 9). These are similar to Table 1, but with different metrics. Rankings are calculated by first averaging tuned performance over all 10 splits of each dataset, and then ranking each algorithm for each dataset according to their average performance. For these rankings, a tie between two algorithms is indicated by the *lowest* (best) ranking. So if multiple algorithms achieve the highest average accuracy for a dataset, they both receive ranking 1.

Some algorithms perform well even *without* hyperparameter tuning. Next, we calculate the ranking of all datasets using the same procedure as in Table 1 and the previous tables, but with their default hyperparameter set displayed as a separate algorithm. We show ranking results for test accuracy. See Table 10. Many of the best-performing algorithms, including CatBoost, XGBoost, LightGBM, and ResNet, perform fairly well both with and without hyperparameter tuning.

Table 7: Performance of algorithms according to Log Loss over 104 datasets. Columns show the rank over all datasets, the average normalized log loss (Mean LL), the standard deviation of normalized LL across folds (Std. LL), and the train time per 1000 instances. Min/max/mean/median are taken over all datasets.

	Rank				Mean LL		Std. LL		Time /1000 inst.	
Algorithm	min	max	mean	med.	mean	med.	mean	med.	mean	med.
CatBoost	1	13	4.48	3.0	0.04	0.02	0.08	0.06	15.91	1.73
XGBoost	1	14	4.76	4.0	0.04	0.02	0.08	0.06	0.90	0.39
SAINT	1	18	6.32	5.0	0.11	0.04	0.09	0.08	199.93	163.28
ResNet	1	15	7.41	7.0	0.11	0.06	0.10	0.07	16.32	9.21
SVM	1	17	7.64	8.0	0.14	0.05	0.09	0.05	55.37	1.10
LightGBM	1	18	7.67	6.5	0.12	0.07	0.21	0.09	1.13	0.29
DANet	1	17	7.76	8.0	0.11	0.08	0.12	0.08	72.05	61.35
FTTransformer	1	16	8.53	9.0	0.14	0.08	0.11	0.09	31.21	18.82
STG	1	17	8.66	8.5	0.17	0.06	0.07	0.05	18.82	16.00
LinearModel	1	18	9.73	10.0	0.24	0.10	0.10	0.06	0.04	0.03
RandomForest	1	18	9.84	10.5	0.19	0.13	0.22	0.07	0.31	0.23
NODE	1	17	11.01	11.0	0.23	0.18	0.04	0.03	206.41	187.22
MLP-rtdl	1	18	11.12	11.0	0.29	0.18	0.17	0.12	13.69	8.15
MLP	1	18	11.62	12.0	0.30	0.22	0.15	0.11	18.44	12.49
TabNet	1	18	11.92	12.5	0.38	0.24	0.36	0.17	34.41	29.69
VIME	2	18	13.54	15.0	0.41	0.37	0.10	0.07	17.15	14.72
KNN	1	18	14.41	15.0	0.51	0.42	0.38	0.20	0.01	0.00
DecisionTree	1	18	14.57	17.0	0.55	0.57	0.53	0.40	0.02	0.01

Table 8: Performance of algorithms according to F1 score over 104 datasets. Columns show the rank over all datasets, the average normalized F1 score (Mean F1), the standard deviation of normalized F1 score across folds (Std. F1), and the train time per 1000 instances. Min/max/mean/median are taken over all datasets.

	Rank				Mean F1		Std. F1		Time /1000 inst.	
Algorithm	min	max	mean	med.	mean	med.	mean	med.	mean	med.
CatBoost	1	17	5.30	4.0	0.88	0.95	0.31	0.21	29.61	2.25
XGBoost	1	16	6.30	5.0	0.83	0.91	0.34	0.21	0.96	0.43
ResNet	1	18	6.94	7.0	0.78	0.84	0.30	0.18	16.11	10.04
SAINT	1	18	7.32	7.0	0.75	0.87	0.31	0.23	170.19	143.68
NODE	1	18	7.53	8.0	0.75	0.82	0.27	0.18	149.90	120.14
FTTransformer	1	16	7.66	7.0	0.77	0.83	0.31	0.19	29.74	18.82
RandomForest	1	18	7.67	7.0	0.78	0.84	0.32	0.21	0.37	0.27
LightGBM	1	18	7.81	7.5	0.78	0.86	0.36	0.21	1.05	0.36
SVM	1	18	8.37	8.5	0.70	0.78	0.26	0.17	28.84	1.43
DANet	1	17	8.74	9.0	0.76	0.82	0.32	0.20	70.38	60.20
MLP-rtdl	1	18	9.46	10.0	0.65	0.74	0.28	0.15	14.36	7.62
DecisionTree	1	18	10.75	12.0	0.62	0.73	0.36	0.24	0.03	0.01
STG	1	18	10.90	11.0	0.59	0.68	0.29	0.17	18.44	15.93
LinearModel	1	18	11.32	13.0	0.54	0.59	0.31	0.23	0.04	0.03
MLP	1	18	11.54	13.0	0.57	0.58	0.30	0.18	18.63	11.79
TabNet	1	18	11.82	13.0	0.57	0.65	0.39	0.25	34.89	29.81
KNN	1	18	12.89	14.0	0.47	0.54	0.29	0.21	0.01	0.00
VIME	2	18	14.38	16.0	0.35	0.34	0.27	0.17	17.27	15.10

Table 9: Performance of algorithms according to ROC-AUC, over 104 datasets. Columns show the rank over all datasets, the average normalized ROC-AUC score (Mean AUC), the standard deviation of normalized ROC-AUC score across folds (Std. AUC), and the train time per 1000 instances. Min/max/mean/median are taken over all datasets.

Algorithm	Rank min	max	mean	med.	Mean AUC mean	med.	Std. AUC mean	med.	Time /1000 inst. mean	med.
CatBoost	1	17	5.12	4	0.92	0.97	0.18	0.09	31.21	1.98
XGBoost	1	16	5.66	4	0.91	0.96	0.19	0.10	0.97	0.39
SAINT	1	18	6.39	5	0.85	0.94	0.19	0.13	168.47	145.51
ResNet	1	18	6.55	6	0.85	0.93	0.19	0.11	15.99	8.97
RandomForest	1	17	6.97	6.5	0.88	0.94	0.19	0.10	0.42	0.29
DANet	1	16	7.25	7	0.84	0.93	0.19	0.08	64.75	57.39
LightGBM	1	18	8.03	7.5	0.84	0.92	0.23	0.10	1.10	0.34
NODE	1	18	8.53	9	0.81	0.90	0.20	0.13	166.63	138.37
FTTransformer	1	18	8.75	9	0.79	0.89	0.20	0.14	29.60	19.51
SVM	1	18	9.07	9	0.76	0.88	0.22	0.11	71.53	1.70
MLP-rtdl	1	18	9.62	9.5	0.71	0.84	0.21	0.13	15.06	7.46
STG	1	18	10.26	11	0.68	0.83	0.24	0.15	18.58	16.18
LinearModel	1	18	10.97	12	0.64	0.78	0.23	0.17	0.04	0.03
MLP	1	18	11.85	13	0.64	0.70	0.24	0.14	18.37	12.50
TabNet	1	18	11.93	13	0.64	0.76	0.32	0.19	35.25	29.89
DecisionTree	1	18	12.92	14	0.53	0.60	0.30	0.23	0.03	0.01
KNN	1	18	13.67	15	0.53	0.64	0.25	0.20	0.01	0.00
VIME	1	18	14.14	15	0.48	0.50	0.30	0.19	18.19	15.84

Now, we present a table similar to Table 1, but with the full set of 176 datasets. See Table 11. Note that some algorithms did not complete on all 176 datasets, but the ordering of the algorithms is nearly the same as in Table 1. We also include partial results for NAM [1], DeepFM [28], and TabTransformer [34].

D.2.2 Training time analysis

In this section we analyze the relative training time required by each algorithm. Here we only consider algorithms with their *default* hyperparameters, so no tuning is used. Table 12 shows a ranking of all algorithms, according to the average training time per 1 000 training samples; as before, we separately present results for datasets of size less than or equal to 1250 in Table 13. These rankings are calculated by first taking the average training time per 1 000 samples over all 10 folds of all 176 datasets, and then ranking each algorithm for each dataset according to this average train time.

D.2.3 HPO Plot

In Figure 4, we plotted the performance improvement of hyperparameter tuning on CatBoost, compared to the absolute performance difference between the best neural net and the best GBDT using default hyperparameters. Now, we also give the same plot for ResNet. See Figure 6.

D.2.4 Critical difference diagrams

Figure 7 shows a critical difference plot according to F1 score. Note that we repeat the Friedman test four times with rankings of the same datasets and algorithms. However it is unlikely that our findings would change, given that p-values for these tests without correction are extremely small ($p < 10^{-20}$).

We compare the performance of each algorithm *family* (GBDTs, NNs, and baselines). Here we use all 176 datasets. We use the same methodology here as in previous sections. See Figure 8 (log loss) and Figure 9 (F1 score).

D.2.5 Venn Diagrams

Recall from Section 2 that for our Venn diagram plots, we split the 19 algorithms into three *families*: GBDTs (CatBoost, XGBoost, LightGBM), NNs (listed in Section 2), and baselines (Decision Tree, KNN, LinearModel, RandomForest, SVM). To compare algorithm performance across datasets, we

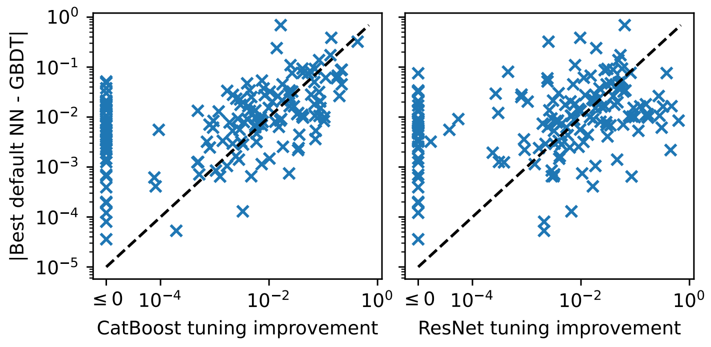


Figure 6: The performance improvement of hyperparameter tuning (horizontal axis) for CatBoost (left) and ResNet (right) compared to the absolute performance difference between the best neural net and the best GBDT, using default hyperparameters (vertical axis). Each point indicates a different dataset, and all values are in normalized log loss. Points on the dotted line indicate that the performance improvement due to hyperparameter tuning is identical to the difference due to NN-vs-GBDT algorithm selection.

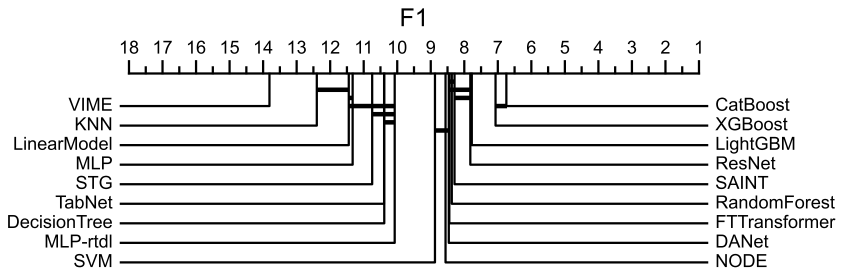


Figure 7: Critical difference diagram comparing all algorithms according to F1 score. Each algorithm's average rank is shown as a horizontal line on the axis. Algorithms which are *not significantly different* are connected by a horizontal black bar.

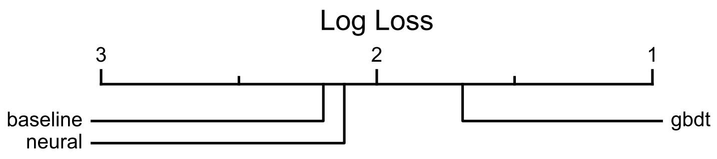


Figure 8: Critical difference plot comparing three algorithm types, according to log loss. Each algorithm's average rank is shown as a horizontal line on the axis. Algorithms which are *not significantly different* are connected by a horizontal black bar.

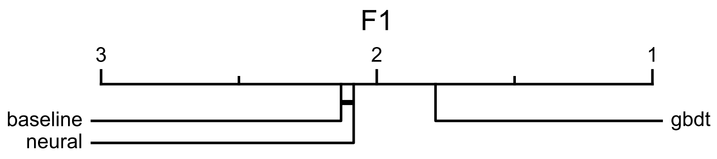


Figure 9: Critical difference plot comparing three algorithm types, according to F1 score. Each algorithm's average rank is shown as a horizontal line on the axis. Algorithms which are *not significantly different* are connected by a horizontal black bar.

Table 10: Performance of algorithms including algorithms parameterized with their default hyperparameters, according to accuracy, over 104 datasets. Columns show the rank over all datasets, the average normalized accuracy (Mean Acc.), the standard deviation of normalized accuracy across folds (Std. Acc.), and the train time per 1000 instances. Min/max/mean/median are taken over all datasets.

	Rank				Mean Acc.		Std. Acc.		Time /1000 inst.	
Algorithm	min	max	mean	med.	mean	med.	mean	med.	mean	med.
CatBoost	1	30	8.05	6.0	0.91	0.95	0.20	0.12	30.27	2.22
CatBoost (default)	1	31	10.12	7.0	0.87	0.93	0.20	0.12	21.95	1.53
XGBoost	1	30	10.77	8.5	0.87	0.93	0.22	0.13	0.94	0.42
ResNet	1	34	11.51	11.5	0.84	0.90	0.20	0.11	16.07	10.04
XGBoost (default)	1	34	12.02	10.5	0.85	0.91	0.22	0.13	1.20	0.61
NODE	1	33	12.17	12.0	0.82	0.88	0.18	0.12	146.89	118.94
SAINT	1	34	12.27	10.0	0.81	0.91	0.20	0.15	168.14	144.84
FTTransformer	1	31	12.78	13.0	0.83	0.87	0.21	0.14	29.47	18.63
RandomForest	1	33	13.00	12.0	0.83	0.88	0.21	0.15	0.36	0.25
LightGBM	1	34	13.19	12.0	0.85	0.90	0.23	0.14	1.06	0.37
ResNet (default)	1	35	13.94	14.0	0.79	0.87	0.22	0.14	15.26	8.67
SVM	1	32	14.24	16.0	0.78	0.88	0.18	0.13	29.22	1.37
LightGBM (default)	1	34	14.38	12.5	0.80	0.88	0.23	0.16	1.30	0.50
SAINT (default)	1	35	14.48	12.0	0.76	0.88	0.20	0.15	135.22	107.82
NODE (default)	1	34	14.61	15.0	0.77	0.86	0.18	0.12	67.54	50.85
DANet	1	33	15.08	15.0	0.83	0.87	0.21	0.15	69.29	60.15
RandomForest (default)	1	35	16.22	15.0	0.76	0.82	0.20	0.14	0.50	0.39
MLP-rtdl	1	34	16.28	18.0	0.74	0.83	0.18	0.11	14.33	7.62
FTTransformer (default)	1	35	18.01	20.0	0.71	0.80	0.21	0.14	25.92	15.35
STG	1	34	18.73	19.0	0.70	0.81	0.20	0.11	18.47	16.00
DecisionTree	1	35	18.93	20.0	0.73	0.79	0.23	0.18	0.03	0.01
SVM (default)	1	35	19.63	24.0	0.64	0.74	0.19	0.12	1.09	0.38
MLP	1	34	19.66	21.5	0.69	0.73	0.20	0.11	18.61	11.92
LinearModel	1	35	19.95	22.0	0.64	0.71	0.22	0.15	0.04	0.03
MLP-rtdl (default)	1	35	20.11	21.0	0.63	0.78	0.20	0.13	13.07	6.33
DANet (default)	1	33	20.19	22.0	0.71	0.74	0.21	0.15	45.04	38.53
TabNet	1	35	20.56	22.5	0.68	0.78	0.25	0.18	35.09	30.83
DecisionTree (default)	1	35	21.88	23.0	0.63	0.70	0.24	0.16	0.02	0.01
KNN	1	35	22.88	25.0	0.59	0.62	0.20	0.15	0.01	0.00
TabNet (default)	1	35	23.52	25.0	0.60	0.70	0.29	0.22	28.39	25.93
MLP (default)	1	35	23.78	27.0	0.56	0.59	0.24	0.18	17.37	11.30
KNN (default)	1	35	24.61	26.5	0.54	0.57	0.21	0.16	0.01	0.00
VIME	3	34	25.16	27.0	0.53	0.59	0.17	0.13	17.10	14.97
STG (default)	1	35	26.17	30.0	0.44	0.38	0.18	0.11	16.39	13.74
VIME (default)	6	35	30.78	33.0	0.23	0.05	0.23	0.14	15.75	14.10

use min-max scaling as described in Section 2. In Figure 4, we said that an algorithm is 'high-performing' if it achieves a scaled test-set accuracy of at least 0.99, and then we determine which algorithm families (GBDTs, NNs, baselines) have a high-performing algorithm. Now we compute the same Venn diagram, when tightening the definition of high-performing to 0.9999 scaled accuracy. See Figure 10. In this case, GBDTs are the sole high-performing algorithm family for 42% of datasets, while NNs are the sole high-performing algorithm family for 30% of datasets. However, since these differences are smaller than 0.1%, they may not be significant to practitioners.

D.2.6 Dataset size analysis

In this section, we investigate the association between dataset size and performance. In Section 2.2 we show that, when compared to NNs and baselines, GBDTs perform relatively better with larger datasets; this is based on a negative correlations between normalized log loss and dataset size. However, it is more informative to compare performance of individual algorithms rather than algorithm families. For example, Figure 11 compares the rank of three algorithms: CatBoost, SAINT, and TabNet, with dataset size. In the left panel (CatBoost minus TabNet), CatBoost outperforms TabNet for all datasets up to size roughly 1 500. For larger datasets there is little difference between CatBoost and

Table 11: Performance of 21 algorithms over all 176 datasets according to accuracy. Columns show the rank over all datasets, the average normalized accuracy (Mean Acc.), and the standard deviation of normalized accuracy across folds (Std. Acc.). Min/max/mean/median are taken over all datasets. The rightmost column show the number of datasets where each algorithm ran successfully, out of 176.

	Rank				Mean Acc.		Std. Acc.		
Algorithm	min	max	mean	med.	mean	med.	mean	med.	
CatBoost	1	18	5.21	4.0	0.88	0.94	0.22	0.11	165
XGBoost	1	19	5.60	4.0	0.88	0.96	0.24	0.12	174
ResNet	1	20	6.80	7.0	0.79	0.88	0.22	0.10	174
LightGBM	1	21	6.98	6.0	0.83	0.92	0.27	0.12	165
SAINT	1	20	7.55	7.0	0.76	0.88	0.25	0.13	138
NODE	1	20	7.57	7.0	0.76	0.84	0.23	0.14	141
RandomForest	1	20	8.10	8.0	0.77	0.84	0.22	0.10	173
FTTransformer	1	17	8.15	8.0	0.76	0.81	0.25	0.14	148
SVM	1	19	8.41	8.0	0.74	0.85	0.21	0.14	143
DANet	1	20	8.73	8.0	0.77	0.83	0.26	0.15	147
MLP-rtdl	1	19	9.57	10.0	0.67	0.75	0.20	0.09	176
DeepFM	1	21	10.89	11.5	0.64	0.74	0.26	0.19	90
TabNet	1	21	11.05	10.0	0.64	0.75	0.29	0.13	168
MLP	1	21	11.36	12.0	0.62	0.64	0.21	0.12	175
DecisionTree	1	21	11.40	12.0	0.60	0.65	0.25	0.13	175
TabTransformer	1	21	11.52	12.0	0.58	0.66	0.16	0.10	124
STG	1	21	11.55	12.0	0.60	0.69	0.22	0.11	164
LinearModel	1	21	12.30	14.0	0.51	0.57	0.23	0.13	168
KNN	1	21	12.83	14.0	0.53	0.58	0.22	0.13	167
VIME	1	21	14.58	16.0	0.41	0.40	0.21	0.12	163
NAM	1	21	15.94	17.0	0.35	0.26	0.25	0.17	80

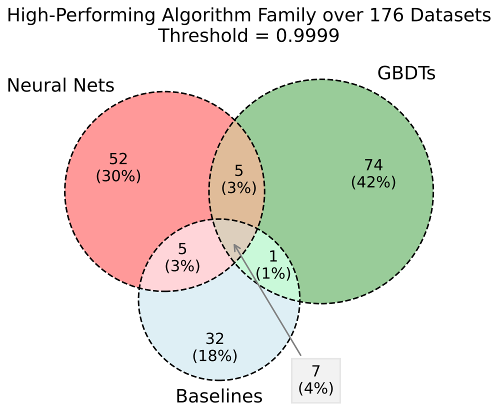


Figure 10: Venn diagram of the number of datasets where an algorithm family is ‘high-performing’, over all 176 datasets. An algorithm is high-performing if its test accuracy after 0-1 scaling is above a certain threshold. While Figure 4 used a threshold 0.99, this figure uses threshold 0.9999.

Table 12: Ranking of all algorithms over all 176 datasets, according to average training time per 1 000 samples. Lower ranks indicate lower training times. Rank columns show min, max, and mean ranks over all datasets. Right columns show average training time per 1 000 samples over all 10 training folds, and the number of datasets considered for each algorithm.

	Rank			Mean Train Time	# Datasets
Algorithm	*min*	*max*	*mean*	(s/1000 samples)	
KNN	1	5	1.66	0.04	167
DecisionTree	1	8	2.33	0.18	175
LinearModel	1	4	2.45	0.05	168
RandomForest	1	8	5.05	0.47	173
XGBoost	1	16	5.70	1.68	174
LightGBM	4	14	6.67	2.64	165
SVM	3	20	6.74	3.76	141
CatBoost	1	19	7.32	15.36	165
MLP-rtdl	2	17	9.23	9.21	176
DeepFM	7	16	9.64	5.52	90
MLP	3	17	10.43	12.19	175
ResNet	4	16	10.46	11.58	174
TabTransformer	5	19	12.79	17.28	122
VIME	8	19	13.12	18.25	156
STG	8	20	13.18	15.26	164
FTTransformer	7	18	13.47	21.97	148
TabNet	9	20	15.37	26.41	160
DANet	11	21	17.20	42.10	146
NODE	7	21	17.44	60.76	141
NAM	12	21	18.34	129.18	79
SAINT	10	21	18.38	119.49	124

TabNet; this indicates that CatBoost should be chosen over TabNet for smaller datasets, and that both algorithms are comparable for larger datasets On the other hand, the center panel (CatBoost minus SAINT), indicates that both CatBoost and SAINT have comparable performance for all datasets *up to* those with size 1 500; for larger datasets, CatBoost outperforms SAINT. This indicates that CatBoost should be chosen over SAINT for very large datasets, but for small datasets both algorithms are comparable.

The main takeaway from these findings is that practitioners should not focus on choosing an algorithm family, such as NNs or GBDTs, to focus on. For example, TabPFN and TabNet are both neural nets, but TabPFN does comparatively better on smaller datasets, while TabNet does comparatively better on larger datasets. Rather they can consult our metadataset of results to decide which algorithm is appropriate for their specific use case. General trends are helpful, but not sufficient, for selecting an effective algorithm.

D.3 Additional experiments from Section 2.2

Recall from Section 2.2 that $\Delta_{\ell\ell}$ denotes the difference in normalized log loss between the best neural net and the best GBDT method. Table 14 and Figure 12 shows the dataset properties with the largest absolute correlation with $\Delta_{\ell\ell}$.

To evaluate the predictive power of dataset properties, we train several decision tree models using the train/test procedure above, with a binary outcome: 1 if $\Delta_{\ell\ell} > 0$ (the best neural net beats the best GBDT), and 0 otherwise. Table 15 shows the performance accuracy of decision trees trained on this task, with varying depth levels; we also include an XGBoost model for comparison. Finally, we include a visual depiction of a simple depth-3 decision tree, in Figure 13. The decision tree classifies which of the top five algorithms performs the best. Note that the decision splits are based purely on maximizing information gain at that point in the tree.

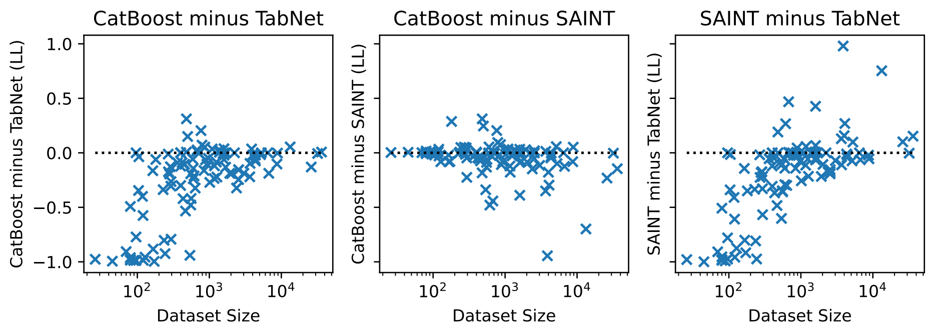


Figure 11: Difference in normalized log loss (lower is better), between three algorithms: CatBoost, SAINT, and TabNet, plotted with dataset size. Points above the dotted line indicate that the second algorithm has lower log loss, meaning better performance than the first.

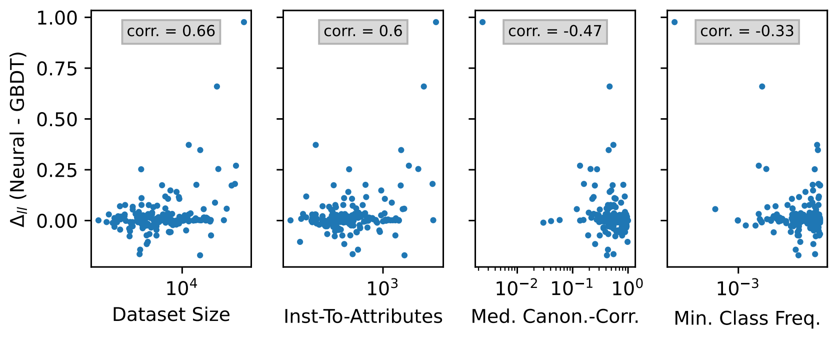


Figure 12: Difference in normalized log loss between the best NN and GBDT ($\Delta_{\ell\ell}$) by four dataset properties, for all 10 splits of all 176 datasets. Larger correlations mean that a larger value of the property corresponds to larger log loss (worse performance) for the best neural net compared to the best GBDT. All dataset properties are plotted on a log scale.

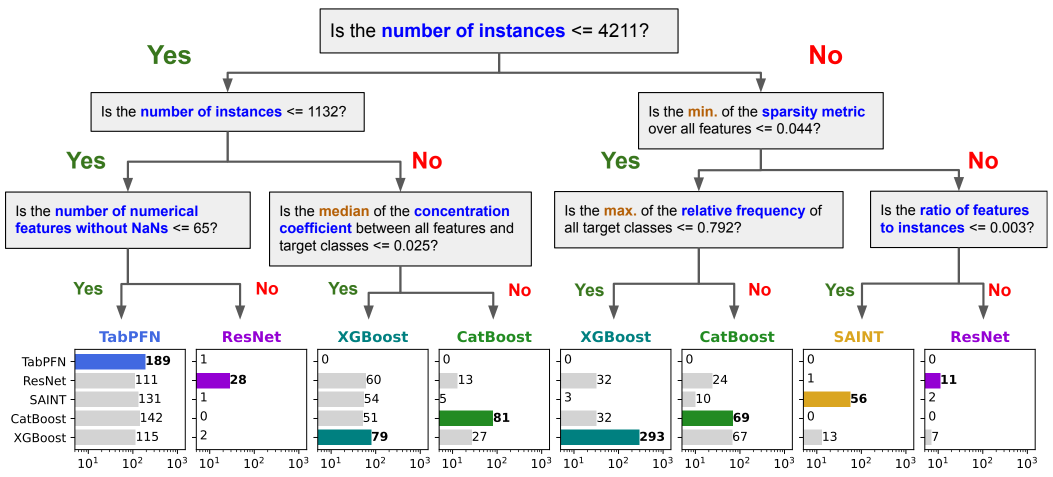

Is the number of instances <= 4211?
Yes
No
Is the number of instances <= 1132?
Is the min. of the sparsity metric over all features <= 0.044?
Yes
No
Yes
No
Is the number of numerical features without NaNs <= 65?
Is the median of the concentration coefficient between all features and target classes <= 0.025?
Is the max. of the relative frequency of all target classes <= 0.792?
Is the ratio of features to instances <= 0.003?
Yes
No
Yes
No
Yes
No
Yes
No
TabPFN
ResNet
XGBoost
CatBoost
XGBoost
CatBoost
SAINT
ResNet
TabPFN
ResNet
SAINT
CatBoost
XGBoost
189
111
131
142
115
1
28
1
0
2
0
60
54
51
79
0
13
5
81
27
0
32
3
32
293
0
24
10
69
67
0
1
56
0
13
0
11
2
0
7
10^1
10^2
10^3

Figure 13: Decision tree for picking the best algorithm, based on our experiments across 176 datasets. The decision tree chooses among the five best-performing algorithms from our experiments: ResNet, SAINT, TabPFN, CatBoost, and XGBoost. Note that the decision splits are based purely on maximizing information gain at that point in the tree, across 176 datasets. IQR denotes interquartile range, and CanCor [43] denotes canonical correlation.

Table 13: Ranking of all 22 algorithms over the 57 datasets of size less than or equal to 1250, according to average training time per 1 000 samples. Lower ranks indicate lower training times. Rank columns show min, max, and mean ranks over all datasets. Right columns show average training time per 1 000 samples over all 10 training folds, and the number of datasets considered for each algorithm.

	Rank			Mean Train Time	# Datasets
Algorithm	*min*	*max*	*mean*	(s/1000 samples)	
TabPFN	1	3	1.16	0.00	57
KNN	1	3	2.00	0.00	57
DecisionTree	1	4	2.88	0.02	57
LinearModel	3	5	4.25	0.06	57
SVM	4	8	5.23	0.22	57
LightGBM	5	12	6.47	0.85	57
RandomForest	5	9	6.84	0.70	57
XGBoost	6	9	7.35	1.25	57
CatBoost	8	19	9.67	17.94	57
MLP-rtdl	9	17	11.51	20.26	57
ResNet	10	16	12.14	22.70	57
VIME	9	16	12.49	18.03	57
STG	9	15	12.56	18.99	57
MLP	9	17	13.26	26.30	57
FTTransformer	11	18	13.86	29.09	57
TabNet	10	18	15.04	32.14	57
DANet	15	19	17.39	53.91	57
NODE	13	19	17.40	61.94	57
SAINT	16	19	18.51	155.07	57

Table 14: Selected metafeatures with the largest absolute correlation with the difference in normalized log loss between the best NN and GBDT ($\Delta_{\ell\ell}$), over all all 176 datasets. Higher correlation values indicate that *larger* values of the metafeature are associated with *worse* NN performance and *stronger* GBDT performance.

Description	**Corr. with $\Delta_{\ell\ell}$**
Log number of instances.	0.63
Ratio of the size of the dataset to the number of features.	0.55
Log of the median canonical correlation between each feature and the target.	-0.41
Log of the min. target class frequency.	-0.35

Finally, we present the metafeatures most-correlated with the difference in log loss between pairs of algorithms. We consider the two best-performing algorithms from each family: CatBoost, XGBoost, ResNet, and SAINT. See Table 19 and Table 20 for statistical and general metafeatures, respectively.

D.4 Experiments on Regression Datasets

While our experiments focus on classification datasets, the TabZilla codebase is also equipped to handle regression datasets—which have a continuous target variable rather than categorical or binary. We run experiments using 12 algorithms with 17 tabular regression datasets, using the same experiment design and parameters described in Section 2. Each algorithm is tuned for each dataset by maximizing the R-squared (R2) metric. The regression datasets used in these experiments have been used in recent studies of machine learning with tabular data [26, 34, 47, 53]; each dataset corresponds to an OpenML task, and can be preprocessed exactly like the classification datasets used in other experiments. The datasets used in these experiments are "Bank-Note-Authentication-UCI" (OpenML task 361002), "EgyptianSkulls" (5040), "Wine" (190420), "Wisconsin-breast-cancer-cytology-features" (361003), "bodyfat" (5514), "california" (361089), "chscase-foot" (5012),

Table 15: The test accuracy of tree models for predicting whether the best neural network will outperform the best GBDT model on a tabular dataset. Results are aggregated over 176 train/test splits with one dataset family held out for testing in each split. The number of dataset properties used by any model of each model type is listed in the leftmost column. Top rows include decision trees (DT$\leq n$) with maximum depth n; the bottom row is an XGBoost model for comparison.

Model	Test Accuracy (mean ± stddev)	Num. Metafeatures
DT$=1$	0.54 ± 0.28	3
DT≤ 3	0.60 ± 0.29	21
DT≤ 5	0.59 ± 0.28	183
DT≤ 7	0.64 ± 0.29	382
DT≤ 9	0.66 ± 0.27	423
DT≤ 11	0.66 ± 0.30	529
DT≤ 13	0.65 ± 0.29	561
DT≤ 15	0.68 ± 0.30	594
DT$\leq \infty$	0.67 ± 0.30	685
XGBoost	0.74 ± 0.33	675

Table 16: Performance of 12 algorithms across 17 tabular regression datasets. Columns show the rank over all datasets, the average normalized R2 (Mean R2), and the std. dev. of normalized R2 across folds (Std. R2). Min/max/mean/median of these quantities are taken over all datasets.

	Rank				Mean R2		Std. R2	
Algorithm	min	max	mean	med.	mean	med.	mean	med.
CatBoost	1.0	7.0	3.0	3.0	0.96	0.98	0.11	0.02
LightGBM	1.0	11.0	4.35	3.0	0.93	0.97	0.15	0.03
RandomForest	2.0	10.0	4.94	4.0	0.89	0.94	0.17	0.03
XGBoost	1.0	10.0	5.29	6.0	0.89	0.99	0.18	0.03
STG	1.0	11.0	6.24	7.0	0.82	0.89	0.10	0.02
LinearModel	1.0	12.0	6.71	7.0	0.75	0.89	0.21	0.04
MLP	1.0	12.0	6.82	7.0	0.78	0.95	0.23	0.04
NODE	1.0	12.0	7.56	8.5	0.47	0.5	0.16	0.16
TabNet	3.0	12.0	7.65	7.0	0.68	0.9	0.48	0.04
DecisionTree	2.0	12.0	7.94	8.0	0.7	0.84	0.30	0.07
KNN	2.0	12.0	8.06	8.0	0.64	0.88	0.11	0.03
VIME	3.0	12.0	9.18	10.0	0.62	0.77	0.15	0.07

"cleveland" (2285), "colleges" (359942), "cpu-small" (4883), "dataset-sales" (190418), "kin8nm" (2280), "liver-disorders" (52948), "meta" (4729), "mv" (4774), "pbc" (4850), and "veteran" (4828).

Table 16 shows the rankings of 12 algorithms on these 17 regression datasets, according to the R2 metric calculated on the test set. The general conclusions are similar to our findings with classification datasets: most algorithms perform well and poorly on *at least one* dataset; however, GBDTs perform particularly well, especially CatBoost.

D.5 Additional Results with Quantile Scaling

In this section, we discuss dataset preprocessing. Different papers use a variety of different preprocessing methods, and there is also a wide range in the amount of 'built-in' preprocessing techniques inside the algorithms themselves. Therefore, our main results minimize confounding factors by having a consistent, lightweight preprocessing (imputing NaN values). In this section, we compare 13 algorithms on the tabzilla benchmark suite with and without quantile scaling, one of the most popular techniques, for all continuous features. We use QuantileTransformer from scikit-learn [46]. We use the same computational setup and experiment design as in our main experiments. See Table 17. We find that quantile scaling improves the simple algorithms: decision tree, MLP, random forest, and SVM, while it has little effect on the high-performing algorithms.

Table 17: Performance of 13 algorithms on all 36 datasets in the hard dataset benchmark suite, with and without applying quantile scaling to each continuous feature. Algorithms with the suffix "-QSCALE" use quantile scaling, while those without this suffix use the raw continuous features. Algorithms are ranked according to normalized log loss, and columns show the rank, normalized log loss, and training time, similar to Table 1.

	Rank				Mean LL		Std. LL		Time /1000 inst.
Algorithm	min	max	mean	med.	mean	med.	mean	med.	mean	med.

Algorithm	Rank min	Rank max	Rank mean	Rank med.	Mean LL mean	Mean LL med.	Std. LL mean	Std. LL med.	Time /1000 inst. mean	Time /1000 inst. med.
XGBoost	1	22	4.78	3	0.07	0.04	0.10	0.06	2.02	0.28
XGBoost-QSCALE	1	24	4.83	3	0.07	0.04	0.10	0.06	1.02	0.26
CatBoost	1	19	5.60	4	0.09	0.06	0.11	0.06	26.46	1.15
CatBoost-QSCALE	1	27	6.57	4	0.12	0.08	0.10	0.06	2.40	1.19
LightGBM	1	24	9.53	8	0.13	0.09	0.19	0.08	1.23	0.36
ResNet	1	23	9.89	9	0.19	0.17	0.12	0.08	8.27	5.22
ResNet-QSCALE	1	25	10.43	9	0.22	0.17	0.13	0.09	8.63	5.74
SAINT	1	28	10.52	8	0.20	0.14	0.12	0.08	130.18	92.42
DANet	2	23	11.63	12	0.18	0.15	0.14	0.11	58.70	52.74
SAINT-QSCALE	1	24	11.74	10	0.23	0.15	0.09	0.05	90.53	42.05
FTTransformer	4	24	12.48	11	0.25	0.20	0.13	0.11	17.41	12.64
RandomForest-QSCALE	3	26	12.77	11	0.27	0.23	0.16	0.08	0.20	0.15
DANet-QSCALE	2	27	13.33	12	0.20	0.17	0.18	0.14	59.44	57.77
FTTransformer-QSCALE	5	24	13.81	12	0.27	0.22	0.14	0.09	19.00	15.93
RandomForest	4	27	14.46	14	0.28	0.27	0.22	0.09	0.35	0.24
MLP-rtdl	2	30	14.56	12	0.36	0.25	0.16	0.13	6.33	4.21
SVM	1	28	15.03	14	0.29	0.21	0.14	0.08	19.73	2.81
SVM-QSCALE	6	22	15.25	17	0.21	0.17	0.13	0.12	8.85	1.91
STG	1	28	15.61	17	0.32	0.23	0.10	0.06	15.99	15.29
MLP-rtdl-QSCALE	1	29	15.92	14.5	0.39	0.31	0.18	0.14	7.35	4.85
TabNet	1	30	16.88	16	0.37	0.27	0.26	0.12	27.02	27.10
NODE	9	27	17.50	17	0.37	0.34	0.08	0.07	153.72	124.27
NODE-QSCALE	6	26	17.76	18	0.36	0.33	0.08	0.07	171.73	147.36
MLP	3	29	17.81	18.5	0.43	0.40	0.15	0.14	8.86	4.36
LinearModel	4	29	17.97	18.5	0.47	0.37	0.11	0.07	0.04	0.02
LinearModel-QSCALE	5	30	18.06	17	0.47	0.38	0.12	0.08	0.01	0.01
VIME	6	28	21.44	24	0.49	0.48	0.09	0.08	20.79	15.18
DecisionTree-QSCALE	8	30	21.54	23	0.60	0.59	0.36	0.20	0.05	0.01
DecisionTree	9	31	21.86	23	0.62	0.62	0.38	0.20	0.11	0.01
KNN	6	31	23.61	25	0.68	0.70	0.35	0.21	0.03	0.00
KNN-QSCALE	5	30	23.94	25.5	0.70	0.75	0.34	0.23	0.01	0.00

D.6 Experiments with Additional Hyperparameter Optimization

In our main experiments, for hyperparameter optimization (HPO), we ran 30 iterations of random search for all algorithms. In this section, we test the impact of additional HPO for four algorithms: XGBoost, CatBoost, LightGBM, and RandomForest. We did not run additional HPO experiments on any neural net methods due to the substantial compute resources required.

For each algorithm, we run 100 iterations of HPO using the default Optuna [2] algorithm (tree-structured Parzen Estimator), optimizing log loss. We run these HPO experiments on all 36 datasets in the TabZilla benchmark suite. All hyperparameter ranges can be viewed in our repository in the folder `https://github.com/naszilla/tabzilla/tree/main/TabZilla/models`.

Table 18 shows the performance of these HPO experiments (algorithm suffix "(HPO)"), compared with with the performance of the default hyperparameters (suffix "(default)"), and the performance after 30 iterations of random hyperparameter search as in our main results (no suffix). As expected, additional hyperparameter tuning improves the performance of XGBoost, CatBoost, LightGBM, and RandomForest.

D.7 Forward Feature Selection for Identifying Important Dataset Attributes

In this section, we present a different method for determining which dataset attributes are related to performance differences between algorithms. Here we use greedy forward feature selection [24] to identify important dataset attributes. In these experiments, we study the problem of predicting the difference in normalized log loss between CatBoost and ResNet (two very effective GBDT and NN algorithms), using metafeatures.

Table 18: Performance of all algorithms on all 36 datasets in the hard dataset benchmark suite, including the performance after 100 iterations of HPO (algorithms with suffix "(HPO)"), and the default hyperparameters (algorithms with suffix "(default)"). Algorithms names without any suffix indicate performance after 30 iterations of random hyperparameter search, as in our main results. Algorithms are ranked according to normalized log loss, and columns show the rank, normalized log loss, and training time, similar to Table 1.

	Rank				Mean LL		Std. LL		Time /1000 inst.	
Algorithm	min	max	mean	med.	mean	med.	mean	med.	mean	med.
XGBoost (HPO)	1	30	6.28	4	0.03	0.02	0.04	0.03	6.36	1.54
XGBoost	1	28	6.83	5	0.03	0.02	0.04	0.03	2.02	0.28
CatBoost (HPO)	1	28	6.88	6	0.05	0.02	0.04	0.03	16.75	1.82
CatBoost	1	20	7.23	6	0.05	0.02	0.04	0.02	26.46	1.15
LightGBM (HPO)	1	22	7.38	4.5	0.04	0.02	0.04	0.03	0.64	0.20
XGBoost (default)	1	27	9.06	8	0.07	0.03	0.03	0.03	1.76	0.41
CatBoost (default)	1	33	11.17	10	0.10	0.04	0.03	0.02	29.53	0.97
LightGBM	1	30	11.94	11	0.06	0.03	0.07	0.04	1.23	0.36
ResNet	1	26	12.20	12	0.10	0.04	0.04	0.04	8.27	5.22
SAINT	1	34	13.33	11	0.08	0.05	0.04	0.03	130.18	92.42
LightGBM (default)	2	34	13.81	12	0.07	0.05	0.05	0.04	1.46	0.61
DANet	2	31	14.81	14	0.06	0.05	0.05	0.04	58.70	52.74
FTTransformer	2	29	15.69	15	0.10	0.06	0.04	0.03	17.41	12.64
SAINT (default)	2	37	16.18	13	0.08	0.05	0.04	0.04	111.07	83.68
RandomForest (HPO)	4	35	16.29	13	0.14	0.08	0.08	0.03	0.33	0.20
ResNet (default)	2	35	17.00	16	0.13	0.06	0.06	0.04	7.28	4.72
MLP-rtdl	2	37	17.92	16	0.18	0.09	0.06	0.04	6.33	4.21
RandomForest	5	33	18.03	16	0.15	0.10	0.08	0.03	0.35	0.24
STG	1	35	18.48	20	0.14	0.06	0.04	0.03	15.99	15.29
SVM	1	34	18.62	18	0.11	0.07	0.04	0.03	19.73	2.81
TabNet	1	37	20.30	18	0.14	0.12	0.09	0.06	27.02	27.10
RandomForest (default)	4	34	20.57	21	0.23	0.10	0.03	0.02	0.32	0.27
MLP	3	36	21.06	21.5	0.21	0.12	0.05	0.04	8.86	4.36
FTTransformer (default)	5	36	21.17	23	0.19	0.07	0.05	0.04	15.71	11.42
NODE	11	32	21.27	20	0.19	0.12	0.03	0.02	153.72	124.27
MLP-rtdl (default)	1	39	21.69	21.5	0.29	0.13	0.11	0.06	5.82	3.89
DANet (default)	6	37	21.74	23	0.12	0.09	0.08	0.05	40.59	38.95
LinearModel	8	37	22.09	22.5	0.26	0.13	0.04	0.03	0.04	0.02
NODE (default)	11	34	22.70	23	0.21	0.14	0.03	0.02	52.26	42.19
SVM (default)	2	35	23.57	26	0.15	0.08	0.04	0.02	4.19	0.80
TabNet (default)	2	38	24.94	27	0.22	0.14	0.11	0.07	24.04	23.40
MLP (default)	4	39	25.28	26	0.34	0.17	0.10	0.07	8.13	4.44
VIME	4	34	25.91	29	0.23	0.18	0.04	0.03	20.79	15.18
STG (default)	10	38	26.81	27	0.25	0.14	0.03	0.02	13.72	13.20
DecisionTree	14	37	27.29	28	0.29	0.21	0.13	0.08	0.11	0.01
KNN	6	37	29.27	31	0.28	0.26	0.12	0.09	0.03	0.00
DecisionTree (default)	15	39	31.91	35	0.60	0.67	0.30	0.15	0.12	0.02
VIME (default)	18	39	32.00	33	0.38	0.31	0.09	0.02	20.10	12.79
KNN (default)	6	39	33.48	36	0.59	0.55	0.22	0.17	0.03	0.00

At a high level, greedy forward feature selection selects metafeatures sequentially which improve the performance of the meta-model. To evaluate performance we use leave-one-dataset-out cross validation: each dataset contributes 10 folds to the overall metadataset, so each fold includes 10 instances for validation and all remaining instances for training.

In these experiments, we first use an XGB regressor fit on the entire metadataset to select 200 features for consideration. Then we use greedy forward selection, implemented in the python package mlxtend (`https://rasbt.github.io/mlxtend`), using an XGB regressor as a meta-model. The first five selected features are (in order):

1. Number of features normally-distributed, according to the Shapiro-Wilk test.
2. Median value of the minimum of all features.
3. Median value of the sparsity of all features. [52]
4. Interquartile range of the mean value of all features.
5. Mean of the harmonic mean of all features.

Table 19: Dataset metafeatures that are most correlated with the difference in normalized log loss between pairs of algorithms (the loss of Alg. 1 minus Alg. 2). Correlations are taken over all 10 splits of all 133 datasets in which CatBoost, XGBoost, ResNet, and SAINT ran successfully. The 10 dataset attributes with the largest absolute correlation are listed for each pair of algorithms. Attribute names correspond to the naming convention used by PyMFE.

Alg. 1	Alg. 2	Corr.	Attribute Name
CatBoost	ResNet	-0.25	Maximum skewness of all features.
CatBoost	ResNet	-0.24	Range of the skewness of all features.
CatBoost	ResNet	-0.23	Log of the standard deviation of the kurtosis of all features.
CatBoost	ResNet	-0.23	Log of the standard deviation of the skewness of all features.
CatBoost	ResNet	0.22	Log of the median of the absolute value of the covariance between all feature pairs.
CatBoost	ResNet	0.21	Log of the median of the standard deviation of all features.
CatBoost	ResNet	0.21	Log of the median of the variance of all features.
CatBoost	ResNet	0.20	Log of the median of the maximum value of all features.
CatBoost	ResNet	0.20	Best performance of a naive Bayes classifier trained over 10-fold CV.
CatBoost	SAINT	0.26	Best performance over all 10 folds, of 10-fold CV of a single-node decision tree fit using the least-informative feature.
CatBoost	SAINT	0.25	Average performance over 10-fold CV of a single-node decision tree fit using the least-informative feature.
CatBoost	SAINT	0.24	Median performance over 10 folds, for 10-fold CV of a single-node decision tree fit using the least-informative feature.
CatBoost	SAINT	-0.24	Log of the worst performance over 10-fold CV of a single-node decision tree fit using the most-informative feature.
CatBoost	SAINT	-0.23	Log of the kurtosis of the performance of a single-node decision tree fit using the most-informative feature, over 10-fold CV.
CatBoost	SAINT	0.23	Log of the Shannon entropy of the target.
CatBoost	SAINT	-0.23	Log of the best performance of elite-nearest-neighbor over 10-fold CV.
XGBoost	ResNet	-0.29	Maximum skewness of all features.
XGBoost	ResNet	-0.28	Range of the skewness of all features.
XGBoost	ResNet	-0.28	Log of the standard deviation of the kurtosis of all features.
XGBoost	ResNet	-0.27	Log of the standard deviation of the skewness of all features
XGBoost	ResNet	0.25	Log of the median value of the absolute covariance between all pairs of features.
XGBoost	ResNet	0.24	Log of the median standard deviation of all features.
XGBoost	ResNet	0.23	Log of the median variance of all features.
XGBoost	ResNet	0.23	Log of the median maximum-value of all features.
XGBoost	ResNet	0.22	Best performance of a naive Bayes classifier over 10-fold CV.
XGBoost	ResNet	0.22	Log of the best performance of a naive Bayes classifier over 10-fold CV.
XGBoost	SAINT	-0.23	Noisiness of the features: $(\sum_i S_i - \sum_i MI(i,y))/\sum_i MI(i,y)$, where S_i is the entropy of feature i, and $MI(i,y)$ is the mutual information between feature i and the target y.
XGBoost	SAINT	-0.22	Log of the standard deviation of the kurtosis of all features.
XGBoost	SAINT	-0.20	Log of the standard deviation of the skewness of all features.
XGBoost	SAINT	-0.20	Best performance over 10-fold CV of a single-node decision tree fit using the most-informative feature.
XGBoost	SAINT	-0.20	Maximum skewness of all features.
XGBoost	SAINT	0.19	Log of the Shannon entropy of the target.
XGBoost	SAINT	-0.19	Log of the standard deviation of the absolute correlation between all pairs of features.
XGBoost	SAINT	-0.18	Range of the skewness of all features.
XGBoost	SAINT	0.18	Log of the range of the performance of a decision tree trained on a random attribute, over 10-fold CV.

Table 20: Dataset metafeatures that are most correlated with the difference in normalized log loss between pairs of algorithms (the loss of Alg. 1 minus Alg. 2). Correlations are taken over all 10 splits of all 133 datasets in which CatBoost, XGBoost, ResNet, and SAINT ran successfully. The 10 dataset attributes with the largest absolute correlation are listed for each pair of algorithms. Attribute names correspond to the naming convention used by PyMFE.

Alg. 1	Alg. 2	Corr.	Attribute Name
CatBoost	ResNet	-0.14	Log of the size of the dataset.
CatBoost	ResNet	0.14	Log of the ratio of number of features to dataset size.
CatBoost	ResNet	-0.14	Log of the ratio of dataset size to number of features.
CatBoost	ResNet	0.06	Number of numerical features.
CatBoost	ResNet	0.06	Number of features.
CatBoost	ResNet	-0.06	Range of the relative frequency of each target class.
CatBoost	ResNet	-0.06	Log of the maximum frequency of any target class.
CatBoost	ResNet	-0.05	Interquartile range of the relative frequency of all target classes.
CatBoost	ResNet	-0.04	Maximum relative frequency of any target class.
CatBoost	SAINT	-0.19	Standard deviation of the relative frequency of all target classes.
CatBoost	SAINT	0.18	Kurtosis of the relative frequency of all target classes.
CatBoost	SAINT	-0.18	Maximum relative frequency of all target classes.
CatBoost	SAINT	-0.18	Mean relative frequency of all target classes.
CatBoost	SAINT	-0.17	Log of the median relative frequency of all target classes.
CatBoost	SAINT	0.16	Number of target classes.
CatBoost	SAINT	0.15	Skewness of the relative frequency of target classes.
XGBoost	ResNet	-0.20	Log of the dataset size.
XGBoost	ResNet	-0.20	Log of the ratio of dataset size to number of features.
XGBoost	ResNet	0.20	Log of the ratio of number of features to dataset size.
XGBoost	ResNet	0.11	Log of the minimum relative frequency of all target classes.
XGBoost	ResNet	0.08	Number of numerical features.
XGBoost	ResNet	0.08	Median relative frequency of all target classes.
XGBoost	ResNet	0.07	Minimum relative frequency of all target classes.
XGBoost	SAINT	-0.17	Standard deviation of the relative frequency of all target classes.
XGBoost	SAINT	-0.16	Log of the dataset size.
XGBoost	SAINT	-0.16	Interquartile range of the relative frequency of all target classes.
XGBoost	SAINT	0.15	Log of the ratio of dataset size to number of features.
XGBoost	SAINT	-0.15	Log of the ratio of number of features to dataset size.
XGBoost	SAINT	-0.15	Maximum relative frequency of all target classes.
XGBoost	SAINT	-0.13	Range of the relative frequency of all target classes.
XGBoost	SAINT	-0.13	Mean of the relative frequency of all target classes.
XGBoost	SAINT	-0.12	Median of the relative frequency of all target classes.