International Journal of Data Science and Analytics (2023) 16:85–100
https://doi.org/10.1007/s41060-022-00350-z

REGULAR PAPER

DeepTLF: robust deep neural networks for heterogeneous tabular data

Vadim Borisov[1] · Klaus Broelemann[2] · Enkelejda Kasneci[1] · Gjergji Kasneci[1,2]

Received: 8 February 2022 / Accepted: 16 July 2022 / Published online: 23 August 2022

Abstract
Although deep neural networks (DNNs) constitute the state of the art in many tasks based on visual, audio, or text data, their performance on heterogeneous, tabular data is typically inferior to that of decision tree ensembles. To bridge the gap between the difficulty of DNNs to handle tabular data and leverage the flexibility of deep learning under input heterogeneity, we propose *DeepTLF*, a framework for deep tabular learning. The core idea of our method is to transform the heterogeneous input data into homogeneous data to boost the performance of DNNs considerably. For the transformation step, we develop a novel knowledge distillations approach, *TreeDrivenEncoder*, which exploits the structure of decision trees trained on the available heterogeneous data to map the original input vectors onto homogeneous vectors that a DNN can use to improve the predictive performance. Within the proposed framework, we also address the issue of the multimodal learning, since it is challenging to apply decision tree ensemble methods when other data modalities are present. Through extensive and challenging experiments on various real-world datasets, we demonstrate that the DeepTLF pipeline leads to higher predictive performance. On average, our framework shows 19.6% performance improvement in comparison to DNNs. The DeepTLF code is publicly available.

Keywords Deep neural networks · Heterogeneous data · Tabular data · Tabular data encoding · Multimodal learning

1 Introduction

Tabular data is the most commonly used form of data, and it is ubiquitous in various applications [1], such as medical diagnosis based on patient history [2], predictive analytics for financial applications [3], cybersecurity [4]. Although deep neural networks (DNNs) perform outstandingly well on homogeneous data, e.g., visual, audio, and textual data [5], heterogeneous, tabular data still pose a challenge to these models [1,6].

We hypothesize that the moderate performance of DNNs on tabular data comes from two major factors. The first is the inductive bias(es) [7,8]; for example, convolutional neural networks (CNNs) assume that specific spatial structures are present in the data, recurrent neural networks (RNNs) assume that a temporal relationship between data points exists, whereas tabular data do not have any spatial or temporal connections. The second reason is the high information loss during the data preprocessing step since tabular input data need to undergo cleansing (dealing with missing, noisy, and inconsistent values), uniform discretized representation (handling categorical and continuous values together), and scaling (standardized representation of features) steps. Along with these feature-processing steps, important information contained in the data may get lost, and hence, the preprocessed feature vectors[1](especially when one-hot encoded) may negatively impact training and learning effectiveness [9]. As reported in [10], an efficient transformation of categorical data for training DNNs is still a significant challenge. Furthermore, a work [11] shows that the embeddings (transformations) for numerical features can be also beneficial for DNNs.

Typically, when heterogeneous tabular data is involved, the first choice across all machine learning (ML) algorithms is ensemble models based on decision trees [12], such as random forests (RF) [13], or gradient-boosted decision trees (GBDT) [14]. Since the inductive bias(es) of the methods based on decision trees are well suited to non-spatial heterogeneous data, the data preprocessing step is reduced to a

✉ Vadim Borisov
vadim.borisov@uni-tuebingen.de

1 The University of Tübingen, Tübingen, Germany

2 SCHUFA Holding AG, Wiesbaden, Germany

[1] In this work, we define a *feature vector* as an n-dimensional vector of numerical and categorical features that represent a data object.

minimum. In particular, the most common implementations of the GBDT algorithm—XGBoost [15], LightGBM [16], and CatBoost [17]—handle the missing values internally by searching for the best approximation of missing data points.

However, the most significant computational disadvantage of the decision tree-based methods is while training the need to store (almost) the entire dataset in memory [8]. Furthermore, in the multimodal datasets in which different data types are involved (e.g., visual and tabular data), decision tree-based models are not able to provide state-of-the-art results, whereas DNNs models allow for batch-learning (no need to store the whole dataset), and for those multimodal data tasks, DNNs demonstrate state-of-the-art performance [18].

Towards the goal of significantly boosting DNNs on tabular data, we propose DeepTLF, a novel deep tabular learning framework that exploits the advantages of the GBDT algorithm as well as the flexibility of DNNs. The key element of the framework is a novel encoding algorithm, TreeDrivenEncoder, which transforms the heterogeneous tabular data into homogeneous data by distilling knowledge from nodes of trained decision trees. Thus, DeepTLF can preserve most of the information that is contained in the original data and encoded in the structure of the decision trees and benefit from preprocessing power of decision tree-based algorithms.

Through experiments on various freely available real-world datasets, we demonstrate the advantages of such a composite learning approach for different prediction tasks. We argue that by transforming heterogeneous tabular data into homogeneous vectors, we can drastically improve the performance of DNNs on tabular data.

The main contributions of this work are: (I) We propose a deep tabular learning framework—DeepTLF—that combines the *preprocessing strengths* of GBDTs with the *learning flexibility* of DNNs. (II) The proposed framework builds on a generic approach for transforming heterogeneous tabular data into homogeneous vectors using the structure of decision trees from a gradient boosting model using a novel encoding function—TreeDrivenEncoder. Hence, the transformation approach can also be used independently from the presented deep learning framework. (III) In extensive experiments on eight datasets and compared with state-of-the-art ML approaches, we show that the proposed framework mitigates well-known data-processing challenges and leads to unprecedented predictive performance, outperforming all the competitors. (IV) For multimodal settings with tabular data, we demonstrate the robust performance of our deep tabular learning framework. (V) We provide an open-source implementation of proposed algorithm and published it online https://github.com/unnir/DeepTLF.

2 Related work

In recent years, deep neural networks on tabular data have received much attention from the machine learning and data science communities [1,8,19–32]. The existing approaches can be grouped into two broad categories— *architecture-based* and *data transformation-based* models.

Architecture-based models This group aims at developing new deep learning architectures for heterogeneous data [19,20,23,24,26]. For example, the authors of [23] proposed distinct neural network architecture for reducing the preprocessing and feature engineering effort by introducing a data sharing strategy between a deep and a wide network so that low- and high-level interactions between the inputs can be learned simultaneously, based on the ideas of factorization machines (FM) proposed in [33]. The work [34] extended the sharing strategy using the FM for structured data further. In [24], the authors propose an integrated solution by introducing two special neural networks, one for handling categorical features and another for numerical data. However, for mentioned approaches [23,24,34], it is not clear how other data-related issues, such as missing values, different scaling of numeric features, and noise, influence the predictions produced by the models.

Another line of research in this group tries to combine the advantages of decision trees and neural networks. For example, the authors of [35] introduced the neural decision forest algorithm, an ensemble of neural decision trees, where split functions in each tree node are randomized multilayer perceptrons (MLPs). Another approach [36] presented a strategy for selecting paths in a neural directed acyclic graph to produce the prediction for a given input. Hence, the selected neural paths are specialized to specific inputs. In [37], the authors empirically showed that neural networks with random forest structure could have better generalization ability across various input domains.

A fully differentiable architecture for deep learning, which generalizes ensembles of oblivious decision trees on tabular, is introduced in [20]. Their architecture (coined NODE) employs the entmax transformation [38] and thus maps a vector of real-valued scores to a discrete probability distribution. Furthermore, the work [8] promotes localized decisions that are taken over small subsets of the features.

Other approaches focus on architectures that build on attention-based (deep transformers) mechanisms [39]. For example, the authors of [19] and [22] propose an attentive transformer architecture for deep tabular learning. Their architecture also offers the possibility to interpret the input features; however, for reliable performance, a large amount

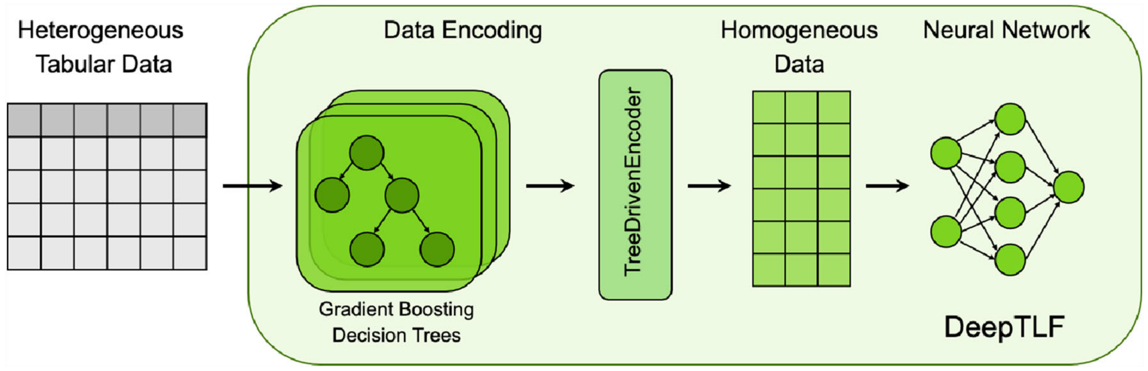


Fig. 1 A data pipeline for the DeepTLF framework. First, the *train* data is used to train a gradient-boosted decision trees (GBDT) model. The heterogeneous data is transformed by exploiting the structures of the decision trees in the ensemble. More specifically, the *TreeDrivenEncoder* algorithm distills information from trained decision trees of the GBDT model to produce homogeneous binary vectors. These vectors are then used to train a DNN. Note that DeepTLF does not require data preprocessing, such as normalization, handling missing values, and encoding categorical features; therefore, in total, it dramatically speeds up the *data preprocessing* time. Note that the test data are not used to train the GBDT algorithm

of training data is needed. Another drawback is that the attention mechanism is only applied to categorical data. Hence, the continuous data do not throw the self-attention block, meaning that correlations between categorical and continuous features are dropped. The work [27] proposes a variation of a transformer and offers semi-supervised learning. However, no clear statements can be drawn for all methods described so far regarding the relationship between data heterogeneity and prediction quality (especially robustness under noisy data or labels). Moreover, many of the solutions in this line of research are quite challenging from a practical perspective since it is often unclear which architectural choices should be employed in realistic scenarios.

These architecture-based approaches generally rely on novel neural network architectures, which are difficult to (re-)implement and optimize for specific real-world use cases. Especially for critical, data-intensive applications, e.g., data streaming, large-scale recommendation systems [40], and many more, it is not always clear what additional adjustments to the working pipeline are needed.

Data transformation-based models Another way to improve the predictive quality in the presence of tabular data is to transform heterogeneous data into homogeneous feature vectors. The transformation can range from simple data preprocessing, such as the normalization of numerical variables or binary encoding of categorical variables, to linear or nonlinear embedding schemes (e.g., generated by advanced autoencoders) [9,10]. The advantage of such data transformation approaches is that they do not require adapting the deep learning architecture. However, they may reduce the information content by smoothing critical values that might have been highly relevant for the final prediction.

Independent works [41,42] demonstrate that data can be encoded using the RF algorithm by accessing leaf indices in the decision trees. The idea was also utilized by [25], where trees from a GBDT model are used for the categorical data encoding instead of the RF model. These works show that the decision trees are a powerful and convenient way to implement nonlinear and categorical feature transformations for heterogeneous data. The DeepGBM framework [24] further evolved the idea of distilling knowledge from the decision tree leaf index by encoding them using a neural network for online learning tasks. Overall, the leaf embedding approach received much attention; however, *the leaf indices from a decision tree embedding do not fully represent the whole decision tree structure*. Thus, each boosted tree is treated as a new *meta categorical feature*, which might be an issue for the DNNs [10].

In contrast to related methods, our aim is to *holistically* distill the information from decision trees by utilizing the whole decision tree, not only the output leaves. The DeepTLF combines the advantages of GBDT (such as handling missing values and categorical variables) with the learning flexibility of DNNs to achieve superior and robust prediction performance. Also, [43] demonstrates that a DNN trained using distilled data can outperform models trained on the original data.

Other approaches such as NODE [20] and Net-DNF [8] try to mimic the decision trees using DNNs. Also, the work [44] proposes a gradient-descent-based strategy that exploits the decision tree structure to propagate gradients in the learning process. Our approach is different because DeepTLF is more robust to data inconsistencies and does not require new DNN architectures. Hence, it is straightforward to use.

Furthermore, the observation that local Boolean features from decision trees model can be informative for global modeling is also reported in [45], where the authors exploit sparse local contrastive explanations of a black-box model to obtain

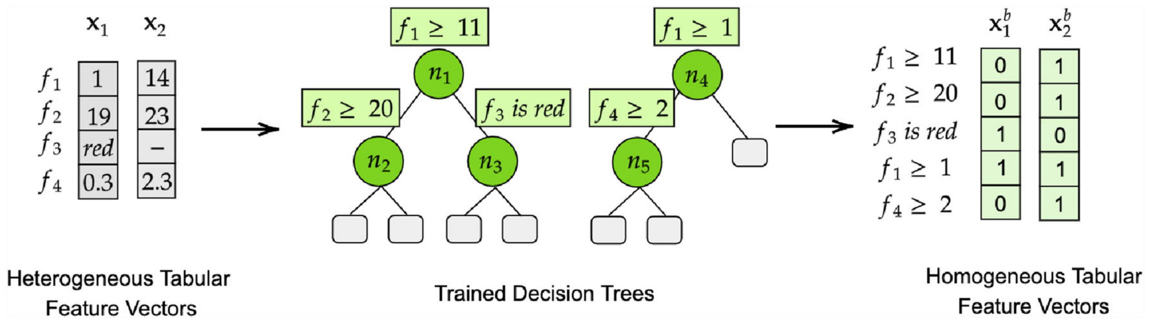


Fig. 2 A toy example of the proposed data transformation performed by the *TreeDrivenEncoder* algorithm. On the left, we see two heterogeneous input feature vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ from the original dataset $\mathcal{D}$, where $\mathbf{x}_i \in \mathbb{R} \times \mathbb{R} \times \{red, -\} \times \mathbb{R}$; f_1, f_2, f_4 are numerical features and f_3 is a categorical feature in a tabular dataset. Note there is also a missing value $-$ for the feature f_3 in $\mathbf{x}_2$. To encode the input data, we use two trained decision trees on the dataset $\mathcal{D}$, with 5 inner nodes in total. By evaluating the Boolean function in each inner node for a given input vector, we construct two homogeneous feature vectors $\mathbf{x}_1^b$ and $\mathbf{x}_2^b$, where a component of these vectors is set to 1 if the corresponding Boolean function evaluates to true and 0 otherwise (colour figure online)

custom Boolean features. A globally transparent model is then trained on the Boolean features; empirically, the global model shows a predictive performance that is slightly worse than state-of-the-art approaches.

In summary, in contrast to state-of-the-art methods that exploit decision tree structures and mainly focus on leaf indices, DeepTLF *utilizes the whole decision tree structure* from a GBDT model, and it furthermore *considers the representation of each feature independently* in the information distillation process. Our framework combines the advantages of gradient-boosted trees (such as handling different scales, different attribute types, missing values, outliers, and many more) with the learning flexibility of neural networks to achieve excellent predictive performance.

3 DeepTLF: deep tabular learning framework

In this section, we present the main components of our DeepTLF framework. As it is depicted in Fig. 1, DeepTLF consists of three major parts: (1) an ensemble of decision trees (in this work, we utilize the GBDT algorithm), (2) a TreeDrivenEncoder that performs the transformation of the original data into homogeneous, binary feature vectors by distilling the information contained in the structures of the decision trees through the TreeDrivenEncoder algorithm, and (3) a deep neural network model trained on the binary feature vectors obtained from the TreeDrivenEncoder algorithm. We will describe the details of each component in the following subsections.

3.1 Gradient-boosted decision tree

For the data encoding step, we selected one of the most powerful algorithms on tabular data, namely the gradient-boosted decision trees (GBDT) algorithm [14]. GBDT is a well-known and widely used ensemble algorithm for tabular data both in research and industrial applications [15] and is particularly successful for tasks containing heterogeneous features, small dataset sizes, and “noisy” data [12]. Especially when it comes to handling variance and bias, gradient boosting ensembles show highly competitive performance in comparison with state-of-the-art learning approaches [12,14]. In addition, multiple evaluations have empirically demonstrated that the *decision trees of a GBDT ensemble preserve the information* from the original data and can be used for further data processing [24,25].

The key idea of the GBDT algorithm is to construct a strong model by iterative addition of weak learners. The set of weak learners $\mathcal{H}$ is usually formed by shallow decision trees, which are directly trained on the original data. Consequently, almost no data preparation is needed, and the information loss is minimized. We denote a GBDT model as a set of decision trees:

$$\mathcal{T}_{\text{GBDT}} = \{T_1, T_2, \ldots, T_k\},$$

where k is the number of estimators in the GBDT algorithm.

The formal definition of the GBDT algorithm is in Appendix A.

3.2 Knowledge distillation from decision trees

The trained GBDT model provides structural data information, which also encodes dependencies between the input features with respect to the prediction task. In order to distill the knowledge from a tree-based model, we propose a novel data transformation algorithm – *TreeDrivenEncoder*. For every input vector from the original data, the proposed encoding method maps all features occurring in the decision trees of the GBDT ensemble to a binary feature vector $\mathbf{x}^b$.

This has the advantage that the neural network in the final component can form its own feature representations from *homogeneous data*. In Fig. 2, we illustrate the transformation obtained by applying the TreeDrivenEncoder algorithm on a toy example. There we have two input feature vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ with categorical and numerical values that are encoded into corresponding homogeneous binary feature vectors $\mathbf{x}_1^b$ and $\mathbf{x}_2^b$.

To formally describe the TreeDrivenEncoder algorithm, we first need a definition of the decision trees:

Definition 1 [Decision Tree] Let T be a structure $T = (V, E, \mu)$, where V is a set of nodes, $E \subseteq V \times V$ is a set of edges and $\mu = (\mu_v)_{v \in V}$ is a sequence of mapping functions $\mu_v : \mathbb{R}^{d \to V \cup \{\emptyset\}}$ that map input vectors to (child) nodes. We call T a (binary) decision tree if it satisfies the following properties:

1. (V, E) is a directed acyclic graph
2. There is exactly one designated node $v_r \in V$, called the root, which has no entering edges, i.e., for a node $v \in V$:

   $$v = v_r \Leftrightarrow \forall w \in V : (w, v) \notin E.$$

3. Every node $v \in V \backslash \{v_r\}$ has *exactly one* entering edge with the parent node at its other end:

   $$w \in V : (w, v) \in E \Leftrightarrow w = parent(v)\,.$$

4. Each node has either two or zero outgoing edges. We call the nodes with two outgoing edges inner nodes and all others nodes leaves. We denote the sets of inner nodes and leaves with V_I and V_L, respectively.
5. μ_v maps feature vectors from inner nodes to their child nodes and from leaves to Ø.

$$v \in V_I \Rightarrow \forall \mathbf{x} \in \mathbb{R}^d : (v, \mu_v(\mathbf{x})) \in E, \qquad (1)$$

$$v \in V_L \Rightarrow \forall \mathbf{x} \in \mathbb{R}^d : \mu_v(\mathbf{x}) = \emptyset. \qquad (2)$$

In the following, we denote the number of inner nodes as $|T| = V_I$. Furthermore, we assume that the child nodes can be identified as left or right child. For each inner node $v \in V_I$, we use a modified mapping function $\tilde{\mu}_v : \mathbb{R}^d \to \{0, 1\}$ (i.e., a Boolean function) where 0 encodes the left child and 1 encodes the right child.

For an input vector $\mathbf{x} \in \mathbb{R}^d$, we exploit the structure of T to derive a binary vector of length $|T|$. To this end, as shown in Alg. 1, we employ a breadth-first-search approach on the nodes of T. More specifically, for every feature that is evaluated at an inner node v of T, we retrieve the corresponding value from $\mathbf{x}$ and evaluate that value at v based on the associated Boolean function. Note that other node visiting strategies (e.g., depth-first search) can be used as well. It is only important that the order of $\tilde{\mu}_v$ is the same.

Finally, we concatenate all the vectors generated from the single decision trees of the ensemble $\mathcal{T}$ on the input vector $\mathbf{x}$, which gives us the final binary representation $\mathbf{x}^b$ of $\mathbf{x}$. We summarize the full algorithm in Alg. 1.

For mathematical completeness, the mapping obtained by applying TreeDrivenEncoder is formalized as follows. Given the feature vector $\mathbf{x}$ that represents an instance from the training dataset $\mathcal{D}$ and a trained decision tree ensemble $\mathcal{T}$ (i.e., a collection of decision trees) on the same dataset, we exploit the structure of each tree $T \in \mathcal{T}$ to produce a binary feature vector for the original feature vector $\mathbf{x} = (x_1, \ldots, x_d)^\top$ and employ a transformation function:

$$map_T : \mathbb{R}^d \to \{0, 1\}^{|T|}, \qquad (3)$$

$$map_T : \mathbf{x} \mapsto (\tilde{\mu}_v(\mathbf{x}))_{v \in V_I}, \qquad (4)$$

where V_I again represents the inner nodes in a well-defined order and $|T|$ their number. The mapping is performed such that at an inner node v of T, the corresponding component x_j of $\mathbf{x}$ *is mapped to 1 if the Boolean function at v evaluates to true for x_j and 0 otherwise*. Note that we apply the transformation function to each node in the decision tree T, even if a node does not belong to the *decision path* of $\mathbf{x}$; hence, it holds that $map_T(\mathbf{x}) \in \{0, 1\}^{|T|}$.

For the multiple decision trees $T_1, ..., T_k$, we construct a function:

$$\texttt{TreeDrivenEncoder} : \mathbb{R}^d \to \{0, 1\}^{\sum_{i=1}^k |T_i|}, \qquad (5)$$

with

$$\texttt{TreeDrivenEncoder}(\mathbf{x}) = \big(map_{T_1}(\mathbf{x}), ..., map_{T_k}(\mathbf{x})\big)^\top. \qquad (6)$$

3.3 Deep learning models for encoded homogeneous data

After the data distillation by the TreeDrivenEncoder algorithm, the new binary representations of the feature vectors are used to train and validate a chosen neural network.

A deep neural network defines a mapping function $\hat{f}$:

$$\hat{y} = f(x^b) \approx \hat{f}(x^b; W), \qquad (7)$$

where $\hat{y}$ is the output of the deep tabular learning framework, x^b is a homogeneous tabular data transformed using TreeDrivenEncoder, and W are learning parameters of the deep learning model.

Depending on a downstream task, the deep learning architecture of the proposed framework should be selected. The

Algorithm 1 For a GBDT model $\mathcal{T}$ and an instance $\mathbf{x}$ from the underlying dataset, the *TreeDrivenEncoder* procedure visits the inner nodes of each $T \in \mathcal{T}$ (in a breadth-first search manner) and exploits their Boolean functions to construct a binary vector according to the feature values of $\mathbf{x}$.

```
1: procedure TreeDrivenEncoder($\mathbf{x}$, $\mathcal{T}$))
2:     $\mathbf{x}^b$ vector of length 0
3:     for tree $T \in \mathcal{T}$ do
4:         $\mathbf{u}$ vector of length $|T|$        ▷ binary vector we aim to construct
5:         $i := 0$                                  ▷ position index in the binary vector
6:         $Q := \emptyset$ an empty queue
7:         $Q$.enqueue(T.root)
8:         while $Q$.notEmpty do
9:             $v$ := Q.dequeue()
10:            $x$ := getFeatureValue($\mathbf{x}$, $v$)   ▷ get from $\mathbf{x}$ the value of the feature that is evaluated at $v$
11:            if $v$.evaluate($x$) == true then      ▷ evaluate $x$ at $v$
12:                add $\mathbf{u}(i) = 1$
13:            else
14:                add $\mathbf{u}(i) = 0$
15:            end if
16:            i++
17:            for all children $v'$ of $v$ do
18:                if $v'$ is an inner node ($v' \in V_I$) then
19:                    $Q$.enqueue($v'$)
20:                end if
21:            end for
22:        end while
23:        $\mathbf{x}^b$ = concat($\mathbf{x}^b$, $\mathbf{u}$)
24:    end for
25:    return $\mathbf{x}^b$
26: end procedure
```

flexibility of the proposed framework allows to utilize almost any existing types of the DNNs.

3.4 DeepTLF and multimodal data

The proposed deep tabular learning framework can be employed for multimodal learning problems [18,46], where multimodal data involve both tabular and other data sources (e.g., text, image, or sound) in an integrated manner while achieving a robust performance. Our multimodal strategy is decoupled from any particular artificial neural network architecture, and thus, it can be easily integrated into an existing multimodal pipeline. Practically multimodal learning is done utilizing different data fusion strategies, i.e., early fusion, middle fusion, and late fusion [47–49].

In early fusion, input data samples can be directly concatenated. Formally, given two feature vectors from modalities I and II, $\boldsymbol{x}^{\mathrm{I}} \in \mathbb{R}^n$ and $\boldsymbol{x}^{\mathrm{II}} \in \mathbb{R}^m$, where n and m are numbers of variables in modalities I and II, respectively. Then, we can define the concatenation as $\mathbb{R}^n \oplus \mathbb{R}^m \mapsto \mathbb{R}^{n+m}$, by the map $(\boldsymbol{x}^{\mathrm{I}}, \boldsymbol{x}^{\mathrm{II}}) \mapsto (x_1^{\mathrm{I}}, \ldots, x_n^{\mathrm{I}}, x_1^{\mathrm{II}}, \ldots, x_m^{\mathrm{II}})$. The concatenation procedure can be accordingly further scaled for more then two modalities.

The middle fusion, sometimes also referred to as intermediate fusion in the literature, is typically used when data from different modalities come with different structures and dimensionalities which are homogeneous for each modality but heterogeneous across modalities, e.g., a multi-dimensional dataset with visual and audio data along with single-dimensional tabular data. Thus, it is challenging to directly concatenate the input data upfront. The middle fusion is done by utilizing the multi-input deep neural network architecture with two types of inputs: single-dimensional input (fully connected layer, recurrent layer, 1D CNN layer) or multi-dimensional layers (e.g., CNN layer). Then, the concatenation of the data signal can be done in the middle of the DNN. Similar to the middle fusion method, the late fusion combines data signals using the last layers.

The choice of the data fusion strategy depends on the modality of the dataset, downstream task, and hardware. In our experiments, we observe that the middle fusion performs better on the modalities. However, further research on data fusion is needed.

4 Experiments

To evaluate the performance of DeepTLF against state-of-the-art models, we employ eight real-world heterogeneous datasets of varying sizes from different application domains.

4.1 Experimental settings

4.1.1 Datasets

For the evaluation of DeepTLF, we used six heterogeneous and two multimodal dataset from different domains as described in Table 1; each dataset was previously featured in multiple published studies. The web access points and description of each dataset are in Appendix C.1. The data is preprocessed in the same way for each experiment; we do normalization and missing values subsection steps, except for GBDT and DeepTLF; since these approaches can handle missing values independently.

4.1.2 Baseline models

For the baseline models, we select the following algorithms: LR, linear or logistic regression models; k-Nearest Neighbors (kNN) [50] is a nonparametric machine learning method; Random Forest (RF) [13]; for GBDT [14], we utilize the XGBoost implementation [15]; DNN, A deep neural network with four fully connected layers and two DropOut layers [51]; Leafs+LR, A hybrid model, combining leaf index from a trained GBDT model and generalized linear models proposed in [25]; RLNs [26], Regularization Learning Networks

Table 1 Details of the datasets used in the experimental evaluations

	Dataset	#Samples	#Num	#Cat	Task
D1	HIGGS	11,000,000	28	0	Classification
D2	Default of clients	30,000	14	9	Classification
D3	Telecom churn	51,047	38	18	Classification
D4	Zillow	167,888	31	27	Regression
D5	Avocado prices	18,249	8	3	Regression
D6	California housing	20,640	8	0	Regression
D7	E-commerce clothing reviews	23,486	6	4	Classification
D8	PetFinder adoption prediction	14,993	3	14	Classification

#Sample is the number of data points, #Num is the number of numerical variables, and #Cat is the number of categorical variables in a dataset

(RLNs) is a dedicated to tabular learning DNN, which uses the counterfactual loss to tune its regularization hyperparameters efficiently. TabNet [19] is a deep tabular data learning architecture, which uses sequential attention to choose which features to reason from at each decision step; neural oblivious decision ensembles (NODE) [20] is a deep tabular data learning architecture, which generalizes ensembles of oblivious decision trees, but benefits from both end-to-end gradient-based optimization and the power of multilayer hierarchical representation learning; DeepGBM [24], a deep learning framework distilled by the GBDT algorithm; Net-DNF [8]; VIME [27], a self-supervised learning framework for tabular data; TabTransformer [22], a framework built using self-attention transformers; lastly, DeepTLF (*the proposed algorithm*), consisting of a four fully connected layers with the two DropOut layers to lower the overfitting effect, the full architecture is presented in Table 2. We *deliberately* select a relatively simple neural network model without advanced layers such as the batch normalization or attention (transformer) to demonstrate the power of our approach. By applying more sophisticated DL techniques, the model performance can be further improved.

Table 2 The DL architecture of the DeepTLF model

Layer	Parameters
Fully connected	#weights = 384
SWISH activation	
DropOut	$p = 0.23$
Fully connected	#weights = 64
SWISH activation	
DropOut	$p = 0.23$
Fully connected	#weights = 32
SWISH activation	
Fully connected	#weights = 1 or 2

4.2 Performance evaluation

Main benchmark In our performance evaluation, we partitioned each of the datasets using *(stratified) fivefold cross-validation*. Our quality measures are cross-entropy loss for classification and mean-squared error (MSE) for regression tasks. Results are reported in terms of mean and standard deviation values in Table 3. Furthermore, we conduct the 5×2 CV paired statistical t-test [52] to compare the proposed framework and GBDT model for all datasets from Table 3. Under the null hypothesis ($\mathbf{H}_0$) that the *GBDT and DeepTLF models have equal performance*, we also set the significance level to 0.05 ($\alpha = 0.05$), i.e., the critical region for a significant statistical difference between our model and the comparison methods. The results are presented in Table 4.

Corrupted data We also compare the performance of DeepTLF with a plain DNN and GBDT under corrupted data to verify the *robustness* of our deep tabular learning framework in scenarios of noisy labels, noisy data, and missing values in the training data (Fig. 3).

Noisy training data and labels We use two different setups: noisy training labels and noisy training data. We artificially corrupted the customer churn dataset by introducing random noise either to the training labels (labels were shuffled) and the training dataset. Note that for validation purposes, the test dataset was not corrupted. A distinguishing strength of the DeepTLF framework compared to other state-of-the-art approaches in the field is that it can handle missing values internally through the proposed gradient-boosting embeddings.

Missing values experiment Figure 9 in Appendix shows the performance of DNN, GBDT, and DeepTLF models with different proportions of missing in the training dataset. As we can see, the performance of the DNNs drops drastically, while DeepTLF shows stable performance.

Sensitivity to hyperparameters This experiment demonstrates how the GBDT hyperparameters contribute to the final performance of the DeepTLF such as the number of decision

Table 3 Experimental results based on (stratified) fivefold cross-validation

	Classification datasets			Regression datasets		
	D1	D2	D3	D4	D5	D6
LR	0.637 ± 0.001	0.470 ± 0.001	0.584 ± 0.001	0.028 ± 0.001	0.966 ± 0.001	0.552 ± 0.063
KNN [50]	Out-of-Memory	0.467 ± 0.003	0.600 ± 0.003	0.029 ± 0.001	0.038 ± 0.001	0.408 ± 0.011
RF [13]	0.502 ± 0.001	0.444 ± 0.007	0.564 ± 0.003	0.028 ± 0.001	0.025 ± 0.001	0.254 ± 0.008
GBDT [14]	0.498 ± 0.001	0.429 ± 0.006	0.559 ± 0.003	0.027 ± 0.003	0.026 ± 0.003	0.217 ± 0.021
Leafs+LR [25]	0.659 ± 0.001	0.453 ± 0.002	0.580 ± 0.002	0.029 ± 0.001	0.105 ± 0.036	0.358 ± 0.032
Deep neural network models						
DNN	0.511 ± 0.001	0.437 ± 0.005	0.579 ± 0.002	0.028 ± 0.001	0.069 ± 0.002	0.339 ± 0.122
RLN [26]	0.507 ± 0.002	0.433 ± 0.051	0.599 ± 0.001	0.399 ± 0.042	0.275 ± 0.244	0.947 ± 0.228
DeepGBM [24]	0.487 ± 0.001	0.457 ± 0.023	0.589 ± 0.001	**0.026 ± 0.001**	0.038 ± 0.045	0.299 ± 0.017
TabNet [19]	0.503 ± 0.001	0.447 ± 0.001	0.591 ± 0.005	0.049 ± 0.001	0.073 ± 0.002	0.455 ± 0.106
VIME [27]	0.514 ± 0.001	0.453 ± 0.006	0.593 ± 0.002	0.030 ± 0.003	0.120 ± 0.016	0.684 ± 0.023
TabTransformer [22]	0.581 ± 0.002	0.515 ± 0.003	0.650 ± 0.021	0.029 ± 0.001	0.073 ± 0.002	0.994 ± 0.501
Net-DNF [8]	0.561 ± 0.001	0.512 ± 0.001	0.594 ± 0.003	0.027 ± 0.001	0.321 ± 0.093	2.491 ± 0.051
NODE [20]	0.489 ± 0.006	0.458 ± 0.006	0.598 ± 0.001	0.028 ± 0.001	0.104 ± 0.030	0.722 ± 0.052
DeepTLF (ours)	**0.483 ± 0.001**	**0.427 ± 0.006**	**0.557 ± 0.003**	**0.026 ± 0.001**	**0.021 ± 0.005**	**0.215 ± 0.012**

We use the same fold splitting strategy for every dataset. The cross-entropy measure (lower is better) is selected for classification tasks and MSE measure (lower is better) is selected for regression problems, respectively. The top results for each dataset are marked in bold

Table 4 Results of the 5 × 2 CV paired statistical t-test between GBDT and DeepTLF models (where * means $p < 0.05$, ** means $p < 0.01$)

	D1	D2	D3	D4	D5	D6
p-value	0.022*	0.003**	0.027*	0.805	0.044*	0.002**

trees (Fig. 4) and learning rate (Fig. 10). For comparison purposes, we also add the GBDT baseline to the figures. It can be seen that DeepTLF does not require extensive hyperparameter tuning, since it reaches the saturation level.

t-SNE visualizations We also compare t-SNE visualizations [53] of the default of the clients dataset and a TreeDrivenEncoder encoded version of the same dataset; the results are shown in Fig. 5. It can be seen that TreeDrivenEncoder indeed preserves valuable information from the trained decision trees.

Multimodal data In this experiment, we demonstrate how the proposed framework performs on multimodal data. For that purpose, we select two multimodal datasets e-commerce clothing reviews [54] and PetFinder adoption prediction [55]. There e-commerce clothing reviews dataset consists of textual and tabular data modalities, and PetFinder adop-

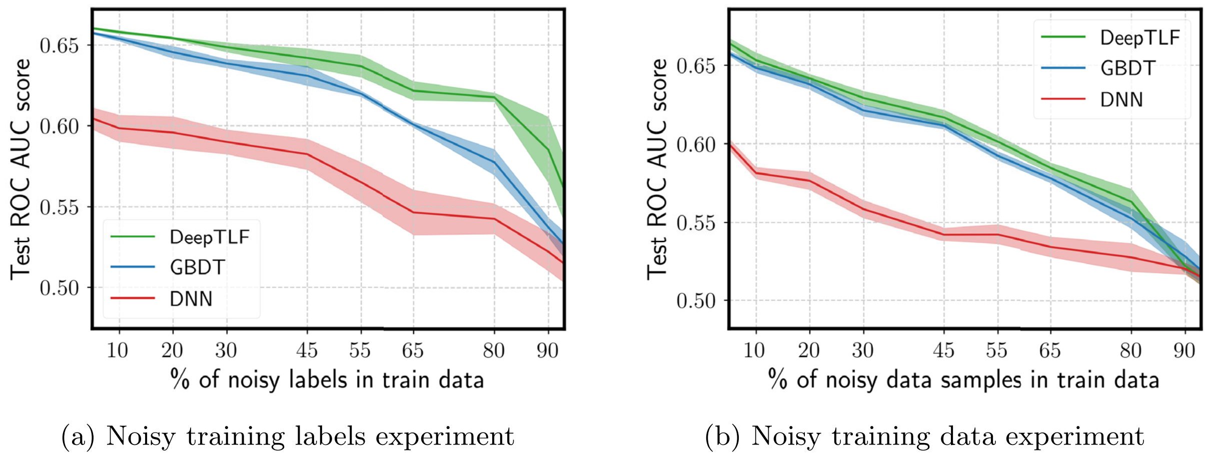


Fig. 3 The DNN here is identical to the DL part in the DeepTLF. Note the only the train data is corrupted, test data has the original values. We report the ROC AUC value (higher is better). Results are averages over five trials for the telecom churn (D3) dataset

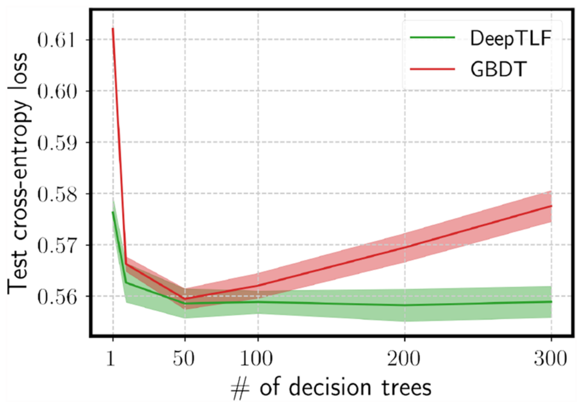
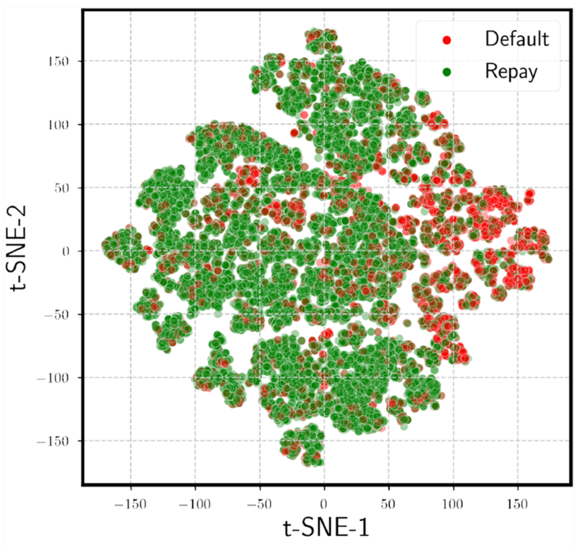


Fig. 4 A relationship between a number of trees in the GBDT and the final performance of the proposed DeepTLF framework. The precise same GBDT model is used for the data encoding in the DeepTLF. Results are averaged over five trials for the D3 dataset



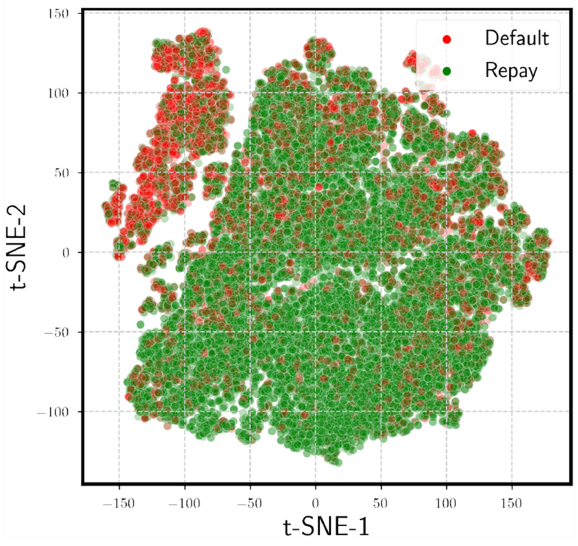


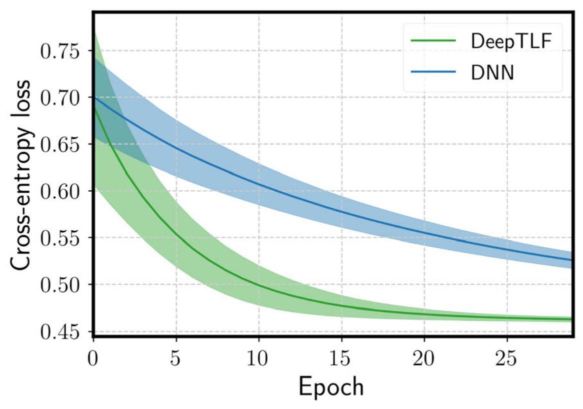
Fig. 5 t-SNE visualizations of original heterogeneous tabular default of clients dataset (top), and the same dataset after the TreeDrivenEncoder transformation (bottom)

tion prediction dataset has visual (images) and tabular data modalities. We compare DNN and DeepTLF models on unseen validation data (Fig. 6) using the middle-fusion strategy (Sect. 3.4) for both datasets. Tabular data representation is the only difference between DeepTLF and DNN baselines in this experiment, for DNN it is the original heterogeneous dataset after the normalization step, where the proposed framework utilizes TreeDrivenEncoder for the data transformation step. The results demonstrate the efficiency of our framework in the multimodal setting.

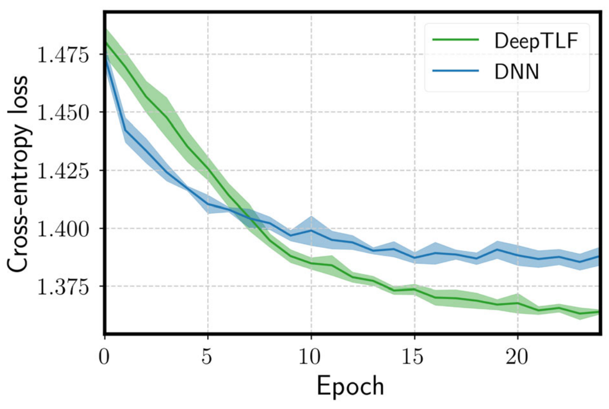
Training/Inference Runtime Comparison Finally, we compare the runtime performance between several DL-based algorithms with GBDT (XGBoost [15]). Table 5 summarizes our results. To make a fair comparison, we used the latest available versions of the corresponding implementations. Also, we utilize the same DL framework, PyTorch [56], and the same number of epochs as well as the batch size, for each DL-based baseline. One of the possible reasons for the gap between the proposed method and other DL-based approaches is that DeepTLF utilizes a simple deep neural network, whereas other approaches apply transformer networks or specialized decision tree-like layers. We also report the data preprocessing time for each baseline. The time cost of DeepTLF is increased compared with GBDT in the inference phase, due to the fact that the GBDT model is a well-optimized framework and written in C++, where DeepTLF is not yet fully optimized in terms of time efficiency and mostly written in Python; however, we do utilize the CUDA acceleration for training and inference steps.

5 Discussion

Empirical evaluations We can derive the following observations from experiments of the study: Our framework, DeepTFL, combines the preprocessing strengths of gradient-enhanced decision trees with the learning flexibility of deep neural networks. It can handle heterogeneity in the data very well and hence shows to be highly efficient. Also, the DeepTLF shows a stable performance irrespective of data size. On a large dataset, the DeepTLF approach demonstrates more than 3% improvement over the GBDT algorithm. We hypothesize that the improvement comes from the fact that deep neural networks perform better when a high number of data samples are available since DL models have more learnable parameters and, as a consequence, are more flexible than decision trees. Finally, with regard to data quality issues (noisy data and labels, missing values), our approach clearly outperforms the DNN and GBDT models, thus showing a robust performance under data quality challenges and



(a) D7 - E-commerce clothing reviews



(b) D8 - Petfinder adoption prediction

Fig. 6 We compare the performance of DNN and DeepTLF models using textual and tabular modalities from the D7 dataset and visual and tabular modalities from the D8 dataset, with identical DL architectures and training setups. The only difference between DeepTLF and DNN models in the experiment is the tabular data representation. Results are averaged over five trials

Table 5 A comparison of the training and inference runtime for selected models from different categories on the whole Zillow dataset (167,888 samples)

Model	Training time (s)	Inference time (s)	Data preprocessing time (s)	#Learning parameters
GBDT (CPU)	13.5	0.5	0	200 trees, depth 4
GBDT (GPU)	3.1	0.3	0	200 trees, depth 4
DNN (GPU)	10.1	0.64	0.3	53,551 weights
RLN (GPU)	10.4	2.22	0.3	60,355 weights
DeepGBM (GPU)	23.9	5.2	0.3	222,548 weights
TabNet (GPU)	79.1	2.2	0.3	584,832 weights
VIME (GPU)	30.2	5.5	0.3	99,732 weights
NODE (GPU)	310.2	15.5	0.3	27,105,922 weights
DeepTLF (GPU)	15.1	3.2	0	80,351 weights

The results related to the training, inference and preprocessing time are averages over five runs over the whole dataset for training and inference tests. The data preprocessing step includes: data scaling and handling missing values

it can *be applicable to many real-world applications where data loss occurs frequently.*

Decision tree model choice Noteworthy, the proposed prediction approach can use any decision tree ensemble as a basic algorithm; in this work, we adopt the GBDT method because of its well-known superior performance on heterogeneous tabular data and its robust feature handling capacities. In addition, the GBDT algorithm sequentially constructs the trees; at each step, the next tree maximally reduces the loss given the current loss. Thus, *there are conditional dependencies between the trees* in the GBDT ensemble, and as a consequence, they provide *adequate coverage of the data distribution*.

Hyperparameter selection for DeepTLF In our experiments, we demonstrate that the DeepTLF framework does not require extensive tinning for the decision tree ensemble part (Figs. 4 and 8); after reaching the saturation level, the number of trees does not have significant effect the performance of proposed framework.

Tabular data encoding Besides constructing a new homogeneous representation for the heterogeneous, tabular data, TreeDrivenEncoder encodes information about the whole dataset, as represented by the structures of the decision trees, which can be seen as a *local feature selection* (and feature engineering).

Furthermore, in terms of efficient representation, the encoded binary data has a drastically smaller size than the original heterogeneous data, since real-valued features are typically represented as 32-bit float types. In contrast, a binary vector can be efficiently represented by a sequence of Boolean values (i.e., 1 bit per value). *This allows for efficient training in the final component for the DeepTLF model.*

The TreeDrivenEncoder algorithm can be used for efficient categorical data encoding. In comparison with leaf-based encoding in DeepGBM [24,25], our transformation scheme utilizes the whole decision tree and produces binary features, whereas leaf-based encoding creates meta categorical features (indexing the leaves).

A comparison to Transformer-based models Most of the current state-of-the-art methods for deep learning on tabular data require an explicit definition of the categorical variables for a dataset, which might bring issues in the online setting, especially for environments with an increasing feature space. Moreover, a drawback of transformer-based methods is that the attention mechanism is applied to categorical values only, implying that the possible correlation between categorical and continuous variables is not taken into account. Furthermore, transformer-based approaches are learning representations for each category, which might be an issue for categorical variables with high cardinality. The proposed approach utilized the power of the decision trees encodes the all type of data together, therefore not suffering from aforementioned drawbacks.

Future work and limitations We see further potential in improving the efficiency of DeepTLF by replacing the decision trees with an efficient neural transformation layer, thus achieving an end-to-end deep learning mechanism for heterogeneous and multimodal data. However, with replacing the GBDT algorithm, the proposed framework loses the preprocessing powers essential for the tabular format [1]. Further improvements of our approach could be the usage of more advanced deep learning architectures such as convolution or attention-based neural networks [39].

Furthermore, an unsupervised training approach is desirable for the self-supervised learning techniques [57]. A possible way to do that is to use multiple variables as targets for the GBDT algorithm after the obtained feature vectors can be stacked into a single *meta* feature vector. Alternatively, the isolated forest algorithm [58] can be utilized for the first stage of the DeepTLF model. Also, the tabular data generation task is challenging due to its heterogeneous nature; however, with our proposed technique, which allows converting heterogeneous data into homogeneous, we see a lot of potentials.

Lastly, further analysis is needed to investigate the performance of DeepTLF in online learning scenarios. The goal would be to develop feature transformation mechanisms that can dynamically adjust to the data distribution's temporal changes and dimensionality. With regard to the GBDT algorithm, deep-learning-based algorithms allow efficient online training. However, DeepTLF works in a hybrid setting; therefore, for the next step, the gradient boosting decision trees might be replaced with a deep learning-based solution; this will lower the training and inference time.

6 Conclusion

In this work, we discussed the challenge of learning from heterogeneous tabular data with deep neural networks. The challenge stems from the concurrent existence of numerical and categorical feature types, complex, irregular dependencies between the features, and other data-related issues such as scales, outliers, and missing values. To address the challenge, we proposed DeepTLF, a framework that exploits the decision trees' structures from an ensemble model to map the original data into a homogeneous feature space where deep neural networks can be effectively and robustly trained. This allows DeepTLF to distill and conserve relevant information in the original data and utilize it in the deep-learning process. Furthermore, the distillation step reduces the required preprocessing to a minimum and can mitigate the mentioned data-related issues by exploiting decision trees' data-processing advantages (internal handling, missing values, and data scaling). Our extensive empirical evaluation on real-world datasets of different sizes and modalities convincingly showed that DeepTLF consistently outperforms the evaluated competitors, which are state-of-the-art approaches in this field. Also, the proposed framework showed robust performance on corrupted data (noisy labels, noisy data, and missing values). Compared to most approaches in this field, DeepTLF is easy to use and does not require changes to existing ML pipelines, which is essential for many practical applications. Moreover, we provide an open-source implementation of DeepTLF which can be used researchers and practitioners for various learning tasks on heterogeneous or multimodal tabular data.

Funding Open Access funding enabled and organized by Projekt DEAL.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Appendix A: Background: gradient boosting decision trees

Formally, at each iteration k of the gradient boosting algorithm, the GBDT model φ can be defined as:

$$\varphi^k(\mathbf{x}) = \varphi^{k-1}(\mathbf{x}) + \lambda\, h^k(\mathbf{x}), \tag{A1}$$

where $\mathbf{x}$ is an input feature vector, φ^{k-1} is the strong model constructed at the previous iteration, h is a weak learner from a family of functions $\mathcal{H}$, and λ is the learning rate.

$$h^k = \underset{h \in \mathcal{H}}{\arg\min} \sum_i \left(-\frac{\partial L(\varphi^{k-1}(\mathbf{x}_i), y_i)}{\partial \varphi^{k-1}(\mathbf{x}_i)} - h(\mathbf{x}_i) \right)^2. \tag{A2}$$

More specifically, a pseudo-residual $-\frac{\partial L(\varphi^{k-1}(\mathbf{x}_i), y_i)}{\partial \varphi^{k-1}(\mathbf{x}_i)}$ should be approximated as well as possible by the current weak model $h(\mathbf{x}_i)$. The gradient w.r.t. the current predictions indicates how these predictions should be changed in order to minimize the loss function. Informally, gradient boosting can be thought of as performing gradient descent in the functional space.

Appendix B: Additional experiments

In the following section, we provide further experimental results to support the proposed deep neural network model on heterogeneous tabular data.

Validation loss curves We examine DNNs and our DeepTLF model separately using only the validation (unseen) data. To enable a fair comparison, the deep learning part in DeepTLF is identical to the DNN we used. The results are presented in Fig. 7.

Is there a correlation between GBDT's performance and the final performance of the DeepTLF? In this experiment, we examine the correlation between the performance of GBDT (which is used for data encoding) and DeepTLF (Figs. 8, 9 and 10). In other words, we want to demonstrate that if the performance of the GBDT improves, the performance of the DeepTLF rises. Figure 11 presents the results of the experiments; as it can be observed, there is indeed a high positive correlation between the performance of the GBDT and DeepTLF. It is also noticeable that the DeepTLF performance in many cased the GBDT results.

A "sanity check" experiment In this simple experiment, we want to verify that the proposed encoding function distills knowledge better than a random "encoding function". In order to do so, we design a random function such that it extracts a random feature f with random splitting value. The experiment confirms that indeed the proposed encoder performs better than a random set of Boolean rules for a given dataset (Fig. 12).

Appendix C: Reproducibility details

We use the following implementation of baseline ML models: kNN, RF, LR, algorithms are from the widely used open-source machine learning python library Scikit-Learn [59], for GBDT we select the python version of distributed gradient boosting library XGBoost [15], RLN, we use the official TensorFlow implementation from the GitHub repository[2], we use well-tested PyTorch implementation of TabNet[3]. We use the official implementation of Net-DNF[4]. We use the official PyTorch implementation of NODE from the GitHub repository[5], we adapt the official implementation of DeepGBM[6] to our need using the PyTorch framework. VIME[7] and TabTansformer[8] implementations are from official GitHub repositories. DeepTLF, for the encoding part, we employ the XGBoost[9] implementation of the GBDT algorithm; for the deep learning part, PyTorch [56] is used. Additionally, we will provide a TensorFlow implementation. Note, other implementations of GBDT can be used as well. We select the AdaBelief Optimizer [60] for proposed framework.

The DNN architecture Table 2 presents the architecture of the DL part of the DeepTLF model, which is identical to the architecture of the DNN baseline. We utilize the three common used neural network layers: fully connected (dense), DropOut [51], and activation layers. For activation, we select the SWISH function proposed in [61]. For the hyper-parameter selection task for all baseline, we apply the tree-structured parzen estimator (TPE) optimization algorithm using the HyperOpt library [62].

t-SNE Experiment For the t-SNE experiments, we scaled the original datasets by applying the z-score normalization. We did not preprocessed an encoded homogeneous dataset after the TreeDrivenEncoder transformation.

Computing infrastructure Out experimental setup for all experiments has two RTX2080Ti GPUs and a single CPU AMD 3960X 24-Core with the Ubuntu 20.04 operation system.

[2] https://github.com/irashavitt/regularization_learning_networks

[3] https://github.com/dreamquark-ai/tabnet

[4] https://github.com/amramabutbul/DisjunctiveNormalFormNet

[5] https://github.com/Qwicen/node

[6] https://github.com/motefly/DeepGBM

[7] https://github.com/jsyoon0823/VIME

[8] https://github.com/lucidrains/tab-transformer-pytorch

[9] https://xgboost.readthedocs.io/en/latest/

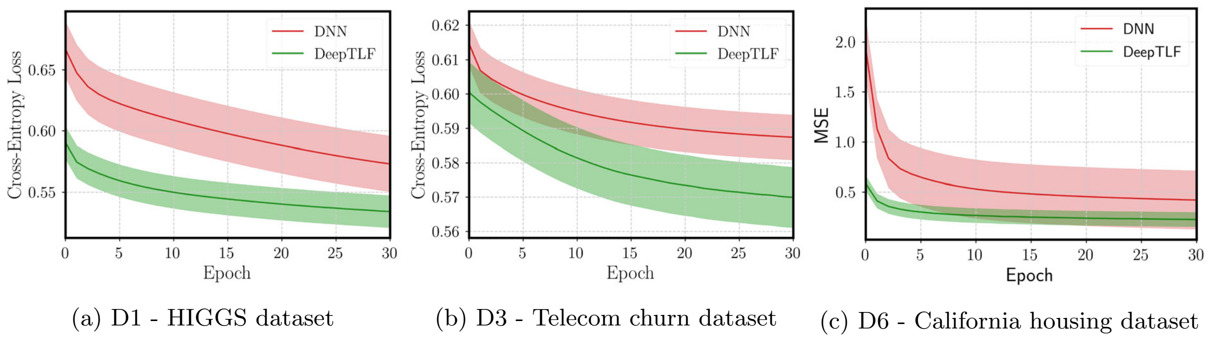


Fig. 7 The comparison of the DeepTLF (a green line) and the deep neural model (DNN) (a red line) models on validation (unseen) data. DeepTLF and DNN models have the exact same architecture. The results are computed over ten runs with different random seeds (colour figure online)

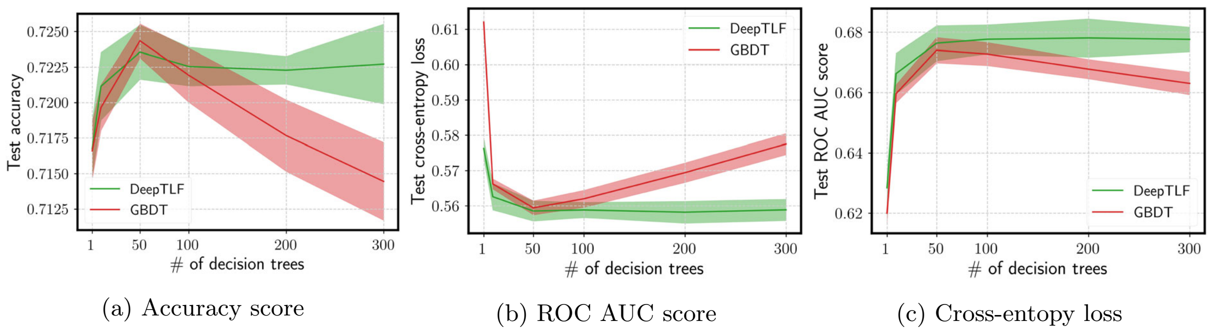


Fig. 8 A relationship between number of decision trees and the DeepTLF performance. The accuracy score (higher is better), ROC AUC score (higher is better), cross-entropy loss (lower is better) metrics for the same experiment. The exact same GBDT model is used for the data encoding in the DeepTLF. The results are averages over five trials for the telecom churn (D3) dataset

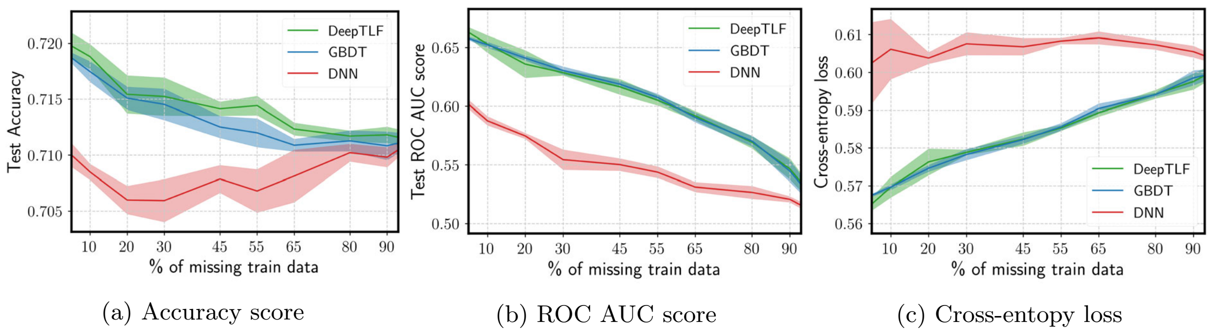
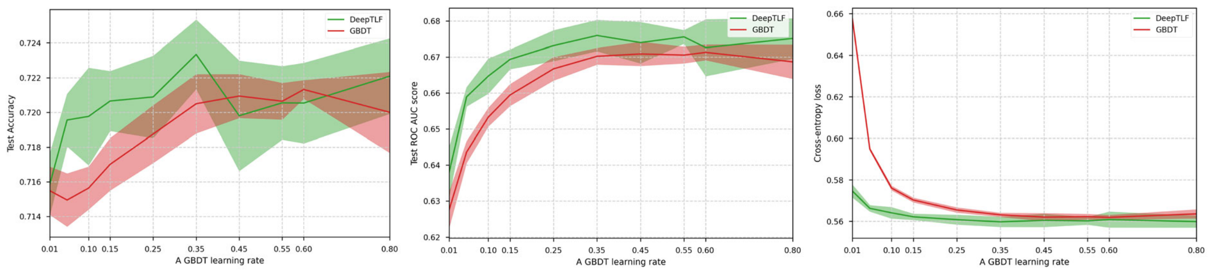


Fig. 9 The missing data experiment. The accuracy score (higher is better), ROC AUC score (higher is better), cross-entopy loss (lower is better) metrics for the same experiment. The exact same GBDT model is used for the data encoding in the DeepTLF. The DNN model is identical in training and architecture to the DeepTLF's DNN part. The results are averages over five trials for the telecom churn (D3) dataset

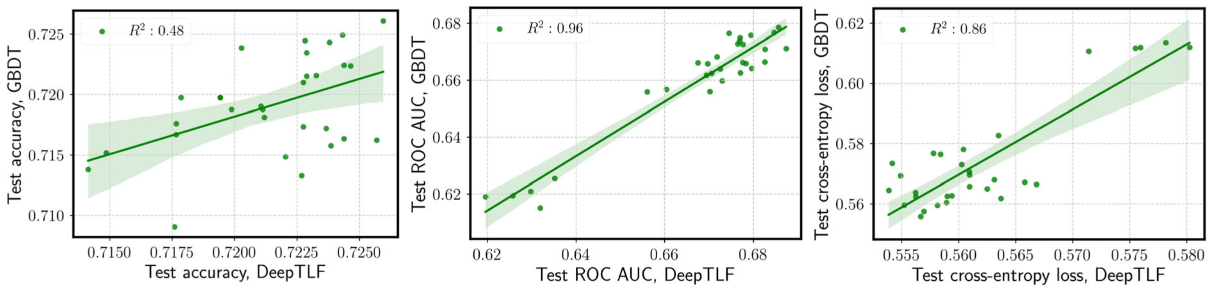
C.1: Datasets description

Among these, the *HIGGS* dataset, which stems from experimental physics, is the largest dataset in our evaluation. As an exemplary dataset from the financial industry, we include the dataset *defaults of clients*, which contains information on default payments, demographic factors, credit data, history of payment, and bill Statements of credit card clients in Taiwan from April 2005 to September 2005. In addition, the *Zillow* dataset represents typical heterogeneous data from the real estate sector. It is important to emphasize that in this dataset around 47 % of the data inputs are missing values. The avocado dataset is another representative of tabular datasets, which provides historical data on avocado prices. The *telecom churn* dataset presents customer data of different feature types with the goal to estimate the behavior of a customer. The *California housing* dataset which contains information about house pricing in 1990. Lastly, we employ two mul-



(a) Accuracy score (b) ROC AUC score (c) Cross-entopy loss

Fig. 10 A relationship between the GBDT learning rate and the DeepTLF performance. In this experiment, we want to show the performance changes of the DeepTLF model by varying the learning rate parameter in the GBDT algorithm. The results are averaged over five trials for the telecom churn (D3) dataset



(a) Accuracy score (b) ROC AUC score (c) Cross-entopy loss

Fig. 11 Correlation plots for different quality measurements. The exact same GBDT model is used for the data encoding in the DeepTLF. The results demonstrate that there is indeed a high positive relationship between the performance of GBDT and DeepTLF. Thus, the proposed data distillation algorithm can successfully distill the knowledge from trees. The results are averaged over five trials for the telecom churn (D3) dataset

timodal datasets: *E-commerce clothing reviews* dataset [54] with text and tabular data, and the *PetFinder adoption prediction* dataset [55] with visual and tabular data, it consists of information on cats and dogs with associated images.

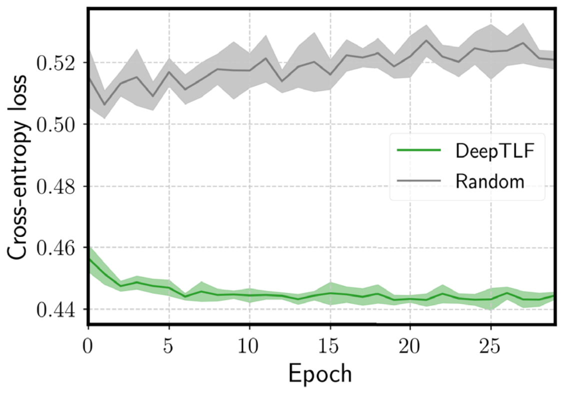


Fig. 12 A "sanity check" experiment. A comparison of the TreeDrivenEncoder and random encoding functions. The random encoding function mimics the TreeDrivenEncoder, but it selects a random feature and splitting value. The experiment verifies that the TreeDrivenEncoder is able to distill the knowledge using trained decision trees in a GBDT algorithm. The results are averaged over five trials for the telecom churn (D3) dataset

All these datasets are collected from real-world problems and contain numerical as well as categorical data. Moreover, these datasets are freely available online and common in tabular data processing: each dataset was previously featured in multiple published studies. We deliberately chose these eight datasets to cover different domain areas (web, natural sciences, etc.), tasks (classification and regression), different dataset sizes, and various data modalities.

Table 6 presents positive and negative class ratios for the classification datasets of this study. The online links to each dataset are provided in Table 7.

We prepossessed the data in the same way for every baseline model by applying standard normalization. For the linear regression, logistic regression, and models based on neural networks, the missing values were substituted with zeros since these methods cannot handle them otherwise.

Table 6 Positive and negative class ratios for binary classification datasets used in the study

	Dataset	Negative class (0) %	Positive class (1) %
D1	HIGGS	0.47	0.53
D2	Default of Clients	0.778	0.221
D3	Telecom churn	0.712	0.288
D4	E-commerce clothing reviews	0.178	0.822

Table 7 URLs for datasets of the study

	Dataset	URL
D1	HIGGS	https://archive.ics.uci.edu/ml/datasets/HIGGS
D2	Default of clients	https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset
D3	Telecom churn	https://www.kaggle.com/c/zillow-prize-1
D4	Zillow	https://www.kaggle.com/neuromusic/avocado-prices
D5	Avocado prices	https://www.kaggle.com/blastchar/telco-customer-churn
D6	California housing	https://www.kaggle.com/camnugent/california-housing-prices
D7	E-commerce clothing reviews	https://www.kaggle.com/nicapotato/womens-ecommerce-clothing-reviews
D8	PetFinder adoption prediction	https://www.kaggle.com/competitions/petfinder-adoption-prediction/data

References

1. Borisov, V., Leemann, T., Seßler, K., Haug, J., Pawelczyk, M., Kasneci, G.: Deep neural networks and tabular data: a survey. arXiv preprint arXiv:2110.01889 (2021)
2. Fatima, M., Pasha, M., et al.: Survey of machine learning algorithms for disease diagnostic. J. Intell. Learn. Syst. Appl. **9**(01), 1 (2017)
3. Dastile, X., Celik, T., Potsane, M.: Statistical and machine learning models in credit scoring: a systematic literature survey. Appl. Soft Comput. **91**, 106263 (2020)
4. Buczak, A.L., Guven, E.: A survey of data mining and machine learning methods for cyber security intrusion detection. IEEE Commun. Surv. Tutor. **18**(2), 1153–1176 (2015)
5. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning (2016). http://www.deeplearningbook.org
6. Shwartz-Ziv, R., Armon, A.: Tabular data: deep learning is not all you need (2021)
7. Mitchell, B.R., et al.: The spatial inductive bias of deep learning. PhD thesis, Johns Hopkins University (2017)
8. Katzir, L., Elidan, G., El-Yaniv, R.: Net-DNF: effective deep modeling of tabular data. In: International Conference on Learning Representations (2020)
9. García, S., Luengo, J., Herrera, F.: Data preprocessing in data mining, vol. 72. Springer, Cham, Switzerland (2015)
10. Hancock, J.T., Khoshgoftaar, T.M.: Survey on categorical data for neural networks. J. Big Data **7**, 1–41 (2020)
11. Gorishniy, Y., Rubachev, I., Babenko, A.: On embeddings for numerical features in tabular deep learning. arXiv preprint arXiv:2203.05556 (2022)
12. Nielsen, D.: Tree boosting with xgboost-why does xgboost win "every" machine learning competition? Master's thesis, NTNU (2016)
13. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
14. Friedman, J.H.: Stochastic gradient boosting. Comput. Stat. Data Anal. **38**(4), 367–378 (2002)
15. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, pp. 785–794 (2016)
16. Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.-Y.: Lightgbm: a highly efficient gradient boosting decision tree. In: Advances in Neural Information Processing Systems, pp. 3146–3154 (2017)
17. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: CatBoost: unbiased boosting with categorical features. In: Advances in Neural Information Processing Systems, pp. 6638–6648 (2018)
18. Gu, K., Budhkar, A.: A package for learning on tabular and text data with transformers. In: Proceedings of the Third Workshop on Multimodal Artificial Intelligence, pp. 69–73 (2021)
19. Arik, S.O., Pfister, T.: TabNet: attentive interpretable tabular learning. arXiv preprint arXiv:1908.07442 (2019)
20. Popov, S., Morozov, S., Babenko, A.: Neural oblivious decision ensembles for deep learning on tabular data. arXiv preprint arXiv:1909.06312 (2019)
21. Yin, P., Neubig, G., Yih, W.-T., Riedel, S.: Tabert: pretraining for joint understanding of textual and tabular data. arXiv preprint arXiv:2005.08314 (2020)
22. Huang, X., Khetan, A., Cvitkovic, M., Karnin, Z.: Tabtransformer: tabular data modeling using contextual embeddings. arXiv preprint arXiv:2012.06678 (2020)
23. Guo, H., Tang, R., Ye, Y., Li, Z., He, X.: DeepFM: a factorization-machine based neural network for CTR prediction. arXiv preprint arXiv:1703.04247 (2017)
24. Ke, G., Xu, Z., Zhang, J., Bian, J., Liu, T.-Y.: DeepGBM: a deep learning framework distilled by GBDT for online prediction tasks. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 384–394 (2019)
25. He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers, S., et al.: Practical lessons from predicting clicks on ads at facebook. In: Proceedings of the Eighth International Workshop on Data Mining for Online Advertising, pp. 1–9 (2014)
26. Shavitt, I., Segal, E.: Regularization learning networks: deep learning for tabular datasets. In: Advances in Neural Information Processing Systems, pp. 1379–1389 (2018)
27. Yoon, J., Zhang, Y., Jordon, J., van der Schaar, M.: Vime: extending the success of self-and semi-supervised learning to tabular domaindim. Adv. Neural Inf. Process. Syst. **33**, 11033–11043 (2020)

28. Padhi, I., Schiff, Y., Melnyk, I., Rigotti, M., Mroueh, Y., Dognin, P., Ross, J., Nair, R., Altman, E.: Tabular transformers for modeling multivariate time series. arXiv preprint arXiv:2011.01843 (2020)
29. Levy, E., Mathov, Y., Katzir, Z., Shabtai, A., Elovici, Y.: Not all datasets are born equal: on heterogeneous data and adversarial examples. arXiv preprint arXiv:2010.03180 (2020)
30. Ballet, V., Renard, X., Aigrain, J., Laugel, T., Frossard, P., Detyniecki, M.: Imperceptible adversarial attacks on tabular data. arXiv preprint arXiv:1911.03274 (2019)
31. Akrami, H., Aydore, S., Leahy, R.M., Joshi, A.A.: Robust variational autoencoder for tabular data with beta divergence. arXiv preprint arXiv:2006.08204 (2020)
32. Gupta, K., Pesquet-Popescu, B., Kaakai, F., Pesquet, J.-C.: A quantitative analysis of the robustness of neural networks for tabular data. In: ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 8057–8061 (2021). IEEE
33. Rendle, S.: Factorization machines. In: 2010 IEEE International Conference on Data Mining, pp. 995–1000 (2010). IEEE
34. Lian, J., Zhou, X., Zhang, F., Chen, Z., Xie, X., Sun, G.: xDeepFM: combining explicit and implicit feature interactions for recommender systems. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1754–1763 (2018)
35. Rota Bulo, S., Kontschieder, P.: Neural decision forests for semantic image labelling. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 81–88 (2014)
36. Denoyer, L., Gallinari, P.: Deep sequential neural network. arXiv preprint arXiv:1410.0510 (2014)
37. Wang, S., Aggarwal, C., Liu, H.: Using a random forest to inspire a neural network and improving on it. In: Proceedings of the 2017 SIAM International Conference on Data Mining, pp. 1–9 (2017). SIAM
38. Peters, B., Niculae, V., Martins, A.F.: Sparse sequence-to-sequence models. arXiv preprint arXiv:1905.05702 (2019)
39. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. arXiv preprint arXiv:1706.03762 (2017)
40. Baylor, D., Breck, E., Cheng, H.-T., Fiedel, N., Foo, C.Y., Haque, Z., Haykal, S., Ispir, M., Jain, V., Koc, L., et al.: Tfx: a tensorflow-based production-scale machine learning platform. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1387–1395 (2017)
41. Moosmann, F., Triggs, B., Jurie, F.: Fast discriminative visual codebooks using randomized clustering forests. In: Advances in Neural Information Processing Systems, pp. 985–992 (2007)
42. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Mach. Learn. **63**(1), 3–42 (2006)
43. Medvedev, D., D'yakonov, A.: New properties of the data distillation method when working with tabular data. arXiv preprint arXiv:2010.09839 (2020)
44. Bruch, S., Pfeifer, J., Guillame-bert, M.: Learning representations for axis-aligned decision forests through input perturbation. arXiv preprint arXiv:2007.14761 (2020)
45. Pedapati, T., Balakrishnan, A., Shanmugam, K., Dhurandhar, A.: Learning global transparent models consistent with local contrastive explanations. Adv. Neural Inf. Process. Syst. **33**, 3592–3602 (2020)
46. Ngiam, J., Khosla, A., Kim, M., Nam, J., Lee, H., Ng, A.Y.: Multimodal deep learning. In: ICML (2011)
47. Boulahia, S.Y., Amamra, A., Madi, M.R., Daikh, S.: Early, intermediate and late fusion strategies for robust deep learning-based multimodal action recognition. Mach. Vis. Appl. **32**(6), 1–18 (2021)
48. Ma, M., Ren, J., Zhao, L., Testuggine, D., Peng, X.: Are multimodal transformers robust to missing modality? In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 18177–18186 (2022)
49. Nagrani, A., Yang, S., Arnab, A., Jansen, A., Schmid, C., Sun, C.: Attention bottlenecks for multimodal fusion. Adv. Neural Inf. Process. Syst. **34**, 14200–14213 (2021)
50. Fix, E.: Discriminatory analysis: nonparametric discrimination, consistency properties (1951)
51. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**(56), 1929–1958 (2014)
52. Dietterich, T.G.: Approximate statistical tests for comparing supervised classification learning algorithms. Neural Comput. **10**(7), 1895–1923 (1998)
53. Van der Maaten, L., Hinton, G.: Visualizing data using t-SNE. J. Mach. Learn. Res. **9**(11), 2579–2605 (2008)
54. Brooks, N.: Women's E-commerce clothing reviews. Data retrieved from Kaggle, https://www.kaggle.com/datasets/nicapotato/womens-ecommerce-clothing-reviews (2018)
55. PetFinder.my: PetFinder.my adoption prediction. data retrieved from Kaggle, https://www.kaggle.com/competitions/petfinder-adoption-prediction (2019)
56. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS-W (2017)
57. Jaiswal, A., Babu, A.R., Zadeh, M.Z., Banerjee, D., Makedon, F.: A survey on contrastive self-supervised learning. Technologies **9**(1), 2 (2021)
58. Liu, F.T., Ting, K.M., Zhou, Z.-H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining, pp. 413–422 (2008). IEEE
59. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
60. Zhuang, J., Tang, T., Ding, Y., Tatikonda, S., Dvornek, N., Papademetris, X., Duncan, J.S.: Adabelief optimizer: adapting stepsizes by the belief in observed gradients. arXiv preprint arXiv:2010.07468 (2020)
61. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2017)
62. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. Neural Inf. Process. Syst. Found. **24**, 2546–2554 (2011)