UNIVERSITE DE LIEGE
FACULTE DES SCIENCES APPLIQUEES

Can Deep Learning models improve performance on tabular data ?

BAMBONEYEHO Sonny

Thesis presented to obtain the degree of :
Master of Science in Data Science Engineering

Thesis supervisor :
GEURTS Pierre

Academic year: **2024 - 2025**

Acknowledgments

First and foremost, I would like to thank God for his constant presence, wisdom, and peace throughout the completion of this thesis. Thanks to him, I was able to advance despite moments of doubt, anxiety, and fatigue.

I would also like to thank my family and friends for their support throughout the process of writing this thesis. I thank them for their moral support and encouragement during this period.

I am deeply grateful to my supervisor, Professor Geurts. I thank him for his precious advice, his availability, and feedback throughout the development of this thesis.

I would also like to thank the Grammarly tool that I used to correct grammatical errors and to proofread this thesis.

Finally, I am grateful for all those who contributed directly or indirectly to the successful completion of this thesis.

Summary

This thesis examines the capacity of deep learning models to handle tabular data and whether they can surpass traditional methods, such as XGBoost or Random Forest. Despite the development of deep learning, applying it to tabular data has been less explored. Also, tabular data generation has been investigated to try to improve deep learning performance on tabular data.

A related work chapter has been written to explain what has already been done for deep learning on tabular data. Then, key concepts such as tabular data, interpretability, and deep learning have been defined to clarify the theoretical framework. A special focus has been made on the limitations of deep learning on tabular data, such as dealing with missing values or the presence of uninformative features.

The datasets in this thesis are all designed for regression tasks with numerical features. The methodologies include the explanation of each deep learning model chosen for this thesis, the interpretability tool SHAP (Shapley Additive Explanations), and the tabular data generation method using Large Language Models (LLMs).

Models were ranked for each dataset, and an average rank across datasets was obtained. The results proved that the traditional methods outperformed the deep learning models. However, the ranking of some deep learning models was not too far from them, indicating that deep learning models have the potential to perform well on tabular data. The SHAP analysis provided insights into the performance of the models by highlighting which features contributed most to their predictions. Generating tabular data with LLMs has been tested. The results are dependent on the dataset used, meaning that performance can improve or deteriorate.

To conclude, this thesis demonstrated that deep learning can be a good alternative to traditional methods, as long as proper tuning is performed. However, deep learning presents limitations, especially computational ones. Indeed, those models require much more computational resources as well as more training time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past decade, deep learning has shown great success in unstructured data such as images or texts [3]. However, its application to tabular data, which is certainly the most common form of data in the world, has been less explored [5]. Tabular data consists of structured data arranged in rows and columns.

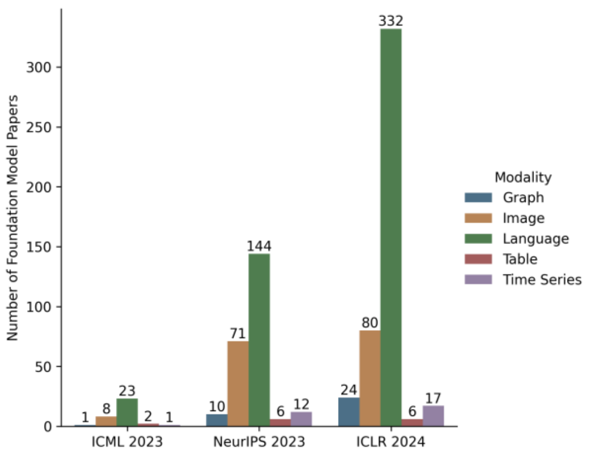


Figure 1.1: Representation of different modalities in foundation model research reproduced from Van Breugel [1]

The application of deep learning on tabular data remains unexplored, as can be seen in Figure 1.1. Yet, tabular data plays a crucial role in various domains. In finance, structured datasets include transaction and account features, which enable tasks such as risk analysis and portfolio management [6]. In healthcare, data contains patient records, laboratory results, and diagnostic information. These datasets enable applications such as predictive diagnostics, patient outcome prediction, or patient treatment [7]. Similarly, in cybersecurity, real-time data analysis is essential for detecting unusual activities and maintaining network security by detecting anomalies and unauthorized access. Fraud detection plays a crucial role in financial transaction systems, such as banks or online purchases. Transactions are registered as tabular data and then analyzed to detect whether a transaction is fraudulent or not [8]. Anomaly detection, also named outlier detection, refers to detecting data points that deviate from the norm, which is useful across fields where unusual data patterns can signal critical events [9]. To summarize, tabular data is present in many fields, as explained before, so investigating more structured data could lead to progress in many areas.

Deep neural networks face significant challenges with tabular data due to the heterogeneous nature of tabular data. Indeed, tabular data combine sparse categorical features with numerous numerical features, whose correlations tend to be weaker than those found for images or text, making the task more challenging for DNNs [10]. Deep learning models are more prone to overfitting because of their large number of parameters.

Gradient Boosting Decision Trees (GBDT) and Random Forests (RF) became the state of the art in working with tabular data [11], [12], [13], [14]. Both are ensemble methods that extend traditional decision trees(DT). Their dominance is further proved by numerous victories of GBDT methods on Kaggle Machine Learning competitions, where they generally outperform DNN on tabular data [15]. A main strength of DTs lies in their ability to pick global features efficiently with the highest information gain [16].

However, investigating this topic could be worthwhile for several reasons. The first and most obvious reason is the potential for improvement of the DNN method's performance on tabular data. Moreover, DNNs facilitate end-to-end learning via gradient descent, which brings some advantages [3]

- The ability to handle different data types, such as images, together with tabular data.
- Reducing the need for manual feature engineering, which is the process of transforming raw data into features. This is an important step for the effectiveness of tree-based methods. Such models can struggle if they encounter unprocessed data.
- Learn from streaming data
- Many appplications scenarios including data-efficient domain adaptation [17], generative modeling [18] and semi-supervised learning [19]

Another motivation to explore deep learning for tabular data is that training tree-based methods requires storing almost the entire dataset in memory [20]. Moreover, decision trees can degrade when dealing with large datasets [21].

The question of the thesis is to investigate the effectiveness of applying deep learning models on tabular data, a domain dominated by traditional models like XGBoost, RandomForest, or GradientBoosting. While classic methods have been explored and optimized for tabular data, the potential of deep learning for this data has been less explored. This research has the goal of providing a view of how deep learning performs and comparing it with traditional methods. Additionally, the thesis aims to see where deep learning models can offer an advantage over classic models and where it is limited. This thesis focuses exclusively on regression tasks involving numerical features. Furthermore, it explores an approach to enhancing the performance of deep learning models on tabular data, which involves using Large Language Models to generate synthetic data to expand the datasets.

The method is motivated by the observation that deep learning methods seem to perform better with larger datasets [22]. The structure of the thesis is as follows: Related Work, Definitions, Issues, Data Collection and Preparation, Methodologies, and Results. Related Work will explore why tabular data should be a priority, why tree-based models outperform deep learning, different types of deep learning models that have been tested on tabular data, tabular data generation, and interpretability. The definitions chapter will define the key concepts around this thesis, such as tabular data, interpretability, and deep learning. The issues chapter will explain why tabular data is a problem for deep learning and the inherent problems of deep learning for tabular data. The Data Collection and Preparation chapter will talk about which datasets were selected for this thesis, as well as how they have been prepared. The Methodologies part will detail every deep learning model that will be used in this thesis, the method that will be used to generate tabular data (GReaT), and the tool that will be used for interpretability (SHAP). Finally, the results chapter will show the results of the model comparison, explain the models' predictions with interpretability, and if the generation of tabular data helped to enhance the performance of deep learning.

Chapter 2

Related Work

In this section, existing work on deep learning applied to tabular data, comparative studies, and tabular data generation methods will be reviewed.

Tabular Data should be a priority Tabular data is omnipresent in many fields, as said in the Introduction. Boris van Breugel & Mihaela van der Schaar [1], in their paper, aim to redirect researchers to tabular data. They proposed the Large Tabular Models (LTM), which are foundation models but in the tabular data field. LTMs could be a tool for preprocessing, cleaning, and finding relevant datasets. Moreover, it could be useful for data augmentation and automated analysis. Tabular Foundation Models (FMs) have been overlooked because of 3 reasons: data, tabular machine learning difficulty, and human perception.

Investigation on why tree-based models outperform deep learning on tabular data The lack of a clear benchmark hinders progress in the field of deep learning with tabular data. This lack of standardization creates flexibility in dataset selection and the choice of methods. Shwartz-Ziv and Armon [23] highlight this by noticing that deep learning methods benchmarked perform better on datasets they selected themselves than on other datasets. Furthermore, this flexibility can lead to unequal hyperparameter tuning [24] and ignoring statistical uncertainty [25]. To solve these issues, Léo Grinsztajn et al. [13] created a tabular data benchmark. The goal of this paper is to investigate different assumptions that a model makes to learn (*i.e.* inductive biases) from the data between tree-based methods and deep learning methods. The paper raises questions, such as "which inductive biases explain the performance of tree-based methods on tabular data ?". The paper provides explanations for why tree-based methods are better for tabular data. Indeed, their superiority is explained by tabular data features including irregular patterns in the target function (*i.e.* the relation between the input data and target is complex). Additionally, tabular data often contains uninformative features and is non-rotationally invariant. Moreover, this benchmark can be used to compare new architectures with those explored in the paper.

Deep Learning on Tabular Data Vadim Borisov et al. [5] present a survey that provides an overview of the application of deep learning to tabular data. As mentioned in the previous paper (Léo Grinsztajn et al. [13]), there are a variety of benchmarks, but they cannot be compared. This paper aims to bring a comparison between deep learning inference methods and classical methods such as XGBoost. That paper leads to different conclusions. Firstly, tree-based methods continue to outperform deep learning methods. Secondly, variations in dataset selection and hyperparameter choices can lead to different outcomes, especially given the large number of datasets possible for benchmarking. Consequently, there is a need for a standardized benchmarking procedure. Thirdly, the preprocessing of tabular data is important for deep learning because this kind of data is heterogeneous (sparse and categorical values). Transforming it into a homogeneous form will make it more compatible with deep learning methods and consequently improve the performance of these methods on tabular data. Architecture-wise, transformers offer multiple advantages over standard neural network methods, such as learning attention over both categorical and numerical features. Performance-wise, hybrid methods, which are methods that unify classical ones with neural network ones, and transformer-based methods achieve superior performance to plain deep neural networks. The paper also concludes that regularization reduces the sensitivity to small changes in input data, thereby enhancing the overall performance of deep neural networks on tabular data. David Holzmüller et al. Yury Gorishniy et al. [26] provided an overview comparing simple deep learning architectures, such as Resnet, and more complex including a transformer-based architecture they implemented, named FT-Transformer. ResNet was chosen for its simplicity and is recommended as a baseline for future benchmarks. The tests were conducted on a diverse set of tasks. NODE [14] achieved high performance but was not the best model across all datasets. It got beaten by FT-Transformer and ResNet, despite its simplicity.

David Holzmüller et al. [27] introduced a method called RealMLP, which is claimed to be an upgrade from MLP. The model incorporates new components such as robust scaling or smooth clipping. RealMLP is reported to be competitive with GBDT methods, and it outperforms other NN architectures with a moderate runtime.

Deep Learning is not all you need Some deep learning models recently proposed for tabular data claimed to be better than XGBoost for some use-cases. Ravid Shwartz-Ziv & Amitai Armon [23] explored whether those models are effective for tabular data. The paper evaluates the methods proposed by other researchers on two criteria: accuracy on datasets different from those used in the original paper, and the time required for training and hyperparameter tuning. The conclusion was that those deep learning models did not outperform XGBoost mainly because the paper did not use the same datasets adopted in the original papers of each deep learning method.

Hybrid models for Deep Learning on Tabular Data Hybrid models can be a solution to make deep learning models work on tabular data. For instance, there is a method introduced by Sergei Popov et al. [14]: Neural Oblivious Decision Ensembles. This method, according to its authors, is the first DNN method capable of outperforming GBDTs in machine learning on tabular data. They provided evidence for this claim through an extensive experimental evaluation on a large number of datasets. This paper concludes that, despite taking more time to train and to infer than established competitors such as CatBoost or XGBoost, NODE offers superior accuracy. A possible research direction is to incorporate a NODE layer inside multi-modal models. These models are built to process different data types, such as images, sequences, and tabular data. While CNNs process images and RNNs handle sequences, the aim is to add a NODE layer to process tabular data. Another example of a hybrid model was proposed by Nathan Lay et al. [28], called Random Hinge Forest. The authors of the paper aimed to address the limitations of random forests by making them differentiable. Their method is trainable end-to-end, similarly to deep neural networks. The result of their experiments demonstrated that the Random Hinge Forest outperforms Random Forest.

Moreover, it performed well on tasks not suited for differentiable (gradient-based) learning. Thomas M. Hein et al. [21] proposed an approach named E2EDT using the same idea. Nevertheless, Borisov et al. [20] combined both methods differently. Indeed, they proposed an approach called DeepTLF. It is based on an encoding algorithm, TreeDrivenEncoder, which turns heterogeneous data into homogeneous data by extracting (distilling) knowledge from nodes of trained decision trees. The output of this process is more suitable for deep learning methods. By transforming tabular data, it improves the predictive quality. This method offers several advantages. First, the distillation step reduces the required processing and mitigates the data-related issues by exploiting the strengths of decision trees. Moreover, it demonstrated robust performance when dealing with corrupt data. Finally, it is easy to use. In addition, attention can be added to hybrid models. This is what Chen et al. [29] have suggested. Their model is called QuantumForest. The authors claim that applying attention could improve the performance of the model. Through their experiments, it was observed that, on the datasets used in the paper, QuantumForest performs better than XGBoost and CatBoost. That means that these new methods have the potential to be better than the state of the art in the domain.

Tabular Data generation The interest in the generation of synthetic, yet realistic, rows of tabular data has increased [5], [30]. This interest comes from the fact that LLMs have shown remarkable performance [31]. A proof of the growing interest in LLMs is the emergence of different LLMs, namely ChatGPT [32] or Grok. Before exploring the LLMs for tabular data generation, other methods have been investigated, such as generative adversarial networks [30] or variational autoencoders (VAE) [33]. To assess an image generation method, synthetic images are generated and then visualized to see if they seem good or not. However, in the context of tabular data, it is very difficult to claim that the generated rows seem to be real or not. To evaluate it, the "train on synthetic and test on real" (TSTR) approach is applied. The better the score, the more realistic the synthetic dataset is. The method that is most used in tabular data generation is the generative adversarial network. Nonetheless, LLM-based methods have been proposed, and they offer some advantages. First, heavy data preprocessing, including encoding categorical data or normalizing numerical features, is not necessary. These steps bring information loss and artificial introduction of noise. Secondly, tabular data is represented as text, enabling the capture of context knowledge between features. Dang Nguyen et al. [34] provide a LLM-based method to produce tabular data. Thanks to some improvements, their method can generate realistic samples. The method has been evaluated in depth with 20 datasets and 10 state-of-the-art baselines. As a result, their method achieves superior performance.

Generative Adversarial Network (GAN) models have shown great results on diverse tasks in machine learning. Even in tabular data, some models were built, including CTGAN and TableGAN. They had enhanced performance in generating synthetic tabular data. However, these methods presented weaknesses. Yishuo Zhang et al. [35] proposed a novel GAN model to generate tabular data. They shifted from the classic GAN formulation using a deep adversarial neural network by using a Bayesian Network. Their method is a Generative Adversarial Network modelling inspired by Naive Bayes and Logistic Regression's relationship (GANBLR). The results showed that GANBLR fake datasets had the best results, meaning that the method was the best among the methods they used in their experiment.

A Bayesian extension of the Variational Auto-Encoder (VAE) framework has been proposed by Patricia A. Apellániz et al. [36]. The method incorporates a Bayesian Gaussian mixture model inside a VAE. This method outperforms state-of-the-art methods such as CTGAN and VTAE. This is explained by the fact that the model captures the distribution of the features individually and across features. Furthermore, the model handles various data types.

Diffusion models have become the leading model in order to generate images or perform other computer vision tasks. High-quality generative models are needed for tabular data for privacy reasons, since personal data cannot be shared, while data generated by a model can be published. Nevertheless, training a diffusion model on tabular data is more challenging because of the heterogeneity of the features and the fact that tabular datasets are mostly small. Despite these two drawbacks, Akim Kotelnikov et al. [37] proposed TabDDPM, a diffusion model capable of processing tabular data. They evaluated the performance by comparing this method with similar methods such as GAN and VAE on a variety of datasets. As a result, it has been found that TabDDPM produces more realistic feature distributions than other methods, which means that it produces higher-quality synthetic data. This result proves the domination of diffusion models in the data generation field. The result is that TabDDPM is less private than GAN and VAE. Since the synthetic data it generates is more similar to real data, it increases the risk of leaking sensitive information about individuals. By contrast, GAN and VAE produce less realistic samples, which makes them more private than TabDDPM.

Transformer for Tabular Data It's unclear which deep learning model could outperform state-of-the-art models for tabular data since there is no standardized benchmark. The lack of clarity is increased since there is a large number of deep learning architectures. Moreover, ensemble methods such as GBDTs, including libraries such as XGBoost [11] or LightGBM [38], are still the state-of-the-art method for tabular data [39]. Since attention-based methods had a lot of success in other domains [40], [41] Yury Gorishniy et al. [26] introduce the FT-transformer, which is a deep learning architecture using transformers for tabular data. It seems to be the universal deep learning (DL) method for tabular data because it performs well on a wider range of tasks than other DL models. However, their method requires more resources for training and is not scaled for large datasets.

Somepalli et al. [42] proposed an approach called Self-Attention and Intersample Attention Transformer(SAINT) using self-attention to train tabular data. It projects all features into a dense vector and then passes it into an attention encoder. Attention is used in two ways in this method. First, "self-attention" which is focused on individual features in each row in the data. Secondly, "intersample attention" is concentrated on the classification of a row by linking it to other rows in the table. Through comparisons with deep learning architectures and tree-based methods, it results that SAINT performs better than boosted trees, including XGBoost and LightGBM.

Convolutional Neural Network(CNN) for Tabular Data Several methods to process tabular data into CNNs have emerged. Methods to perform this can be divided into two parts: domain-specific and generic methods. Domain-specific methods are relevant for the domain they are in, but useless outside of it [43]. For instance, Lyu and Haque used a method to classify tumor types [44]. On the other hand, generic methods are those that can be applied to multiple domains. The main idea is to transform tabular data into an image to be an input for a CNN. Sharma et al. [45] proposed such a method to turn non-image data into an image for a CNN to receive it. Dimension reduction methods such as t-distributed Stochastic Neighbor Embedding (t-SNE) or kernel Principal Component Analysis (kPCA) have been used for high-dimensional data. Amit Damri et al. [43] presented a new generic method named Feature Clustering-Visualization. This technique can be utilized for both data visualization and data classification. FC-Viz performs as a state-of-the-art CNN method for tabular data while reducing the size of resulting images, training time, and inference time.

Self-supervised Learning Collecting large labeled datasets is hardly possible, especially in the medical domain. Generally, those large datasets contain plenty of unlabeled samples. These datasets are better suited for self- or semi-supervised learning. Yet, this type of learning is not efficient with tabular data since it relies on the spatial or semantic structure of an image or a text. Self-supervised learning(SSL) has already been applied to tabular data. Indeed, in the Denoising Autoencoder [46] or Context Encoder [47], the pretext task, which is an SSL task to help the model learn features from unlabeled data, was to recreate the original dataset from a corrupted one. But as can be seen, none of these cases applied SSL to train on tabular data. To solve this issue, Yoon et al. [48] brought self- and semi-supervised frameworks for tabular data. In their method, the pretext task is to recover the mask vector, which is a vector indicating the part of the input data that is missing. The goal of this pretext task is to locate the areas where data is corrupted or masked. They proposed a new method to augment tabular data to broaden SSL to tabular data. Their model is named VIME. It performs better on the datasets they worked with, proving that making proper pretext tasks as well as proper data augmentation techniques for tabular data can lead to good results. Sercan O. Arık and Tomas Pfister [3] propose a new DNN architecture named TabNet. It receives tabular data without any preprocessing and trains using gradient descent. TabNet focuses on an attention mechanism to select important features for each decision step. The paper shows that TabNet outperforms previous approaches on the topic. Additionally, unsupervised pre-training, a technique where a model first learn useful features and is then fine-tuned with labeled data [49], enables fast adaptation and better performance.

Interpretability in Tabular Data Performance is often seen as the main indicator of how good a model is, but if a model cannot be trusted, it will not be used. Therefore, establishing trust in individual predictions of any classifier or regressor is essential. In risk domains such as medicine or terrorism, decisions cannot be accepted blindly. Marco Tulio Ribeiro et al. [50] introduced an algorithm named LIME designed to explain any individual decision of a model. They generalized this method to apply that to the overall behaviour of the model, under the name of SP-LIME. They proved through experiments that providing explanations for model decisions improves user interactions with machine learning models. For both expert and non-expert users, explanations helped to build trust and improve confidence in models that seem untrustworthy. Concerning neural networks, there is a method called DeepLIFT proposed by Avanti Shrikumar et al. [51]. It assigns a score to each feature, determining its influence on the final prediction. This is done by comparing how the network reacts to a real input versus a simple "reference" input. The need to understand why a model makes a certain decision has increased, especially with datasets containing personal data. Therefore, it is crucial to identify harms, such as discrimination introduced by decision-making models. Moreover, algorithmic transparency can help to detect input data that leads to negative outcomes, such as denial of insurance due to incorrect information in the user's profile. Finally, it can explain why a wrong decision was made and also guide how to reverse this decision. However, the research on this topic is still limited. To solve this issue, Anupam Datta et al. [52] brought some advancements in this field. Transparency reports are analyses that explain how a model behaves given different inputs. They formalize them with Quantitative Input Influence (QII), which measures the influence of inputs on outputs. It answers questions about fairness and the influence of individual features, even if features are correlated. However, it is still unclear which of the previously mentioned [50], [52] is better. To address this, Scott M. Lundberg and Su-In Lee [53] developed a unified approach to interpret model predictions named SHAP. Previously proposed DNN approaches for tabular data were not well-suited for this task. They often stacked convolutional layers or multi-layer perceptrons, making the models overparametrized. Without appropriate assumptions, they failed to find optimal solutions for tabular decision-making. Another deep learning method that incorporates interpretability is Table2Image introduced by Seungeun Lee et al. [54]. The method combines tabular data with an image representation of it. The reason for this combination is to have the advantages of both approaches. Tabular structure provides feature-level information while image representation captures more complex patterns. By applying this dual perspective, the model offers a better understanding of its decisions.

Chapter 3

Definitions

To assure clarity and precision in the discussion that will follow, this chapter provides definitions of key terms that will be essential within the context of the thesis.

3.1 Tabular Data

As said briefly in the introduction, tabular data consists of data organized in rows and columns, with each column representing a specific feature. There are several characteristics of tabular data: [55]

1. Heterogeneity: tabular data may contain different kinds of features, including numerical, categorical, and binary.

2. Sparsity: Real-world tabular datasets often present imbalanced data classes and missing values, leading to long-tailed distributions.

3. Preprocessing-dependent: Data preprocessing is crucial in tabular data. For numerical features, examples of preprocessing are normalization or scaling, handling of missing values, and removal of outliers. For categorical features, common techniques for data preprocessing are label encoding and one-hot encoding. A bad or a lack of preprocessing might lead to information loss, a sparse matrix, or even multicollinearity in the case of one-hot encoding, for instance.

4. Correlation in tabular data: In tabular data, variables are correlated. If we take an example of a table containing the temperature and ice cream sales. It's obvious that if the temperature increases, the ice cream sales will increase too.

5. Order invariant: Unlike text-based or image-based data, which are linked to the position of the word or the pixel, tabular data is order-invariant. This property hardens the task of dealing with tabular data for position-based methodologies such as Convolutional Neural Networks(CNN).

6. Lack of prior knowledge: For image and audio data, there is generally prior knowledge about the spatial or temporal structure of the data. However, in tabular data, this knowledge is often lacking. As a result, the model has more difficulty in capturing the relationships between features.

3.2 Interpretability

The definition of interpretability from [56] is: "The model interpretability is the ability (of the model) to explain or to present in understandable terms to a human." While model performance is often the primary criterion for evaluation, understanding how and why a model makes a prediction, which is interpretability, is equally important. It encourages user trust and provides useful insights on how the model can be improved. Interpretability is especially important for domains such as finance and health. Indeed, a finance expert or a doctor should be able to explain the model's decision. For instance, a client has been denied a loan. Interpretability enables the analyst to explain why the loan was rejected. Simple models, such as linear models, are easier to interpret, but with the growth of big data and the increasing use of complex models, interpretability has become a necessity. Indeed, the best explanation of a model is the model itself; however, complex models are not easy to understand. To overcome this limitation, interpretable approximations of complex models are often employed. Interpretability can be categorized into two types: by-design interpretability, where the model itself is inherently understandable, and post-hoc interpretability, which consists of analyzing a trained model to explain its behaviour [57]:

3.2.1 Interpretability by design

The models that fit this category are also called intrinsically or inherently interpretable models. Examples of interpretable models are :

- Linear Regression: The model is simple and produces coefficients that can be easily interpretable.
- Decision trees: They make decisions by splitting data at different nodes; as a result, the decision-making process can be followed by traversing the tree.
- ProtoViT [58]: It is a neural network classifying prototypes. It is trained so that image classification results as a weighted sum of prototypes, similarly to linear regression.
- Yang et al. [59] proposed interpretable tree ensemble methods, which are boosted trees such as XGBoost. This method is a mix of both types of interpretability.
- Boosting [60]: the trained models are a weighted sum of weak learners.

There are different scopes in intrinsically interpretable models :

- Entirely interpretable: it is only the case for the simplest models, namely linear regression with too few coefficients or a small decision tree.
- Parts of the model are interpretable. For instance, in regression, individual coefficients can be interpreted, but not necessarily all coefficients as a whole.
- Predictions of the model are interpretable.

Models in this category of interpretability are easier to debug and improve since information on their inner processing is available. Moreover, they explain their model better and better outputs because the reasons for making certain predictions are often directly understandable from the model.

3.2.2 Post-hoc interpretability

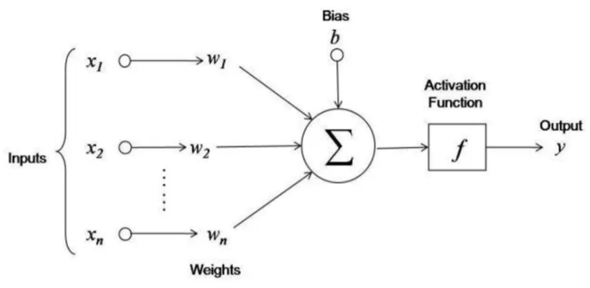
It separates into two types: model-agnostic, where the focus is only on how the model output changes depending on changes to the input, or model-specific, where parts of the model are analyzed for better comprehension.

Model-agnostic These methods follow the SIPA principle: **S**ample from the data, perform an **I**ntervention, get the **P**redictions of the manipulated data, and **A**ggregate the results. One can take a data sample, intervening on it by permuting values from one feature, getting the predictions on the new model, computing error loss, and comparing it to the original loss. This enables the interpretability of the model concerning its features. Model-agnostic methods separate model training and interpretability, contrary to by-design interpretability. It has advantages [50] such as flexibility in the choice of the model and the interpretation method. An example of model-agnostic post-hoc interpretability is SHAP (Shapley Additive Explanations), but it will be discussed more in Chapter 6.

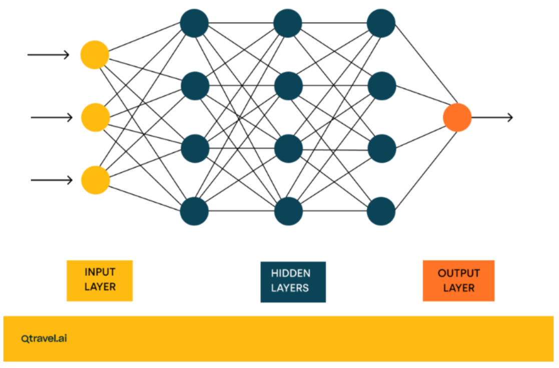
Model-specific Post-hoc model-specific methods are applied after model training and are applied to a certain range of machine learning methods. For predictions in a neural network, the input data has possibly been through a lot of layers, and, as a consequence, the connection between the input and the final prediction is complex. Indeed, millions of weights would have to be considered to understand the model prediction. Interpreting the behaviour and predictions of neural networks requires specialized interpretation methods. However, gradient information from neural networks could be used to develop computationally efficient methods.

3.3 Deep Learning

Deep learning is one of the most explored topics in machine learning because of its ability to simulate the decision-making power of the human brain [61].



(a) Activation Function Process image from [62]



(b) Neural Network image from [63]

Figure 3.1: Neural Network components

From Figure 3.1b, it can be seen that a neural network is made up of layers of elements called neurons. Each neuron receives an input, processes it, and sends it to the next layer. The neural network is divided into 3 parts [17].

- Input layer: receives raw data.
- Hidden layers: layers that are between the input and output layer. Their role is to detect patterns.
- Output layer: produces the output of the process.

From Figure 3.1a, it can be seen that the input of an activation function is a combination of inputs multiplied by weights and bias. Then, this combination is processed into a function and then passed to the next layer, except when the process occurs in the output layer. In that case, the result of the activation function serves as the network's output. Activation functions can be either linear or not, but non-linear ones introduce additional complexity in neural networks and ease the ability for them to learn and approximate a much wider range of functions. Furthermore, activation functions enable mapping unknown distributions into a known one. For instance, the sigmoid function maps any real-valued input to an output in the range [0,1]. This helps stabilize the neural network [64].

Neural network training has the goal of finding weights that minimize the best prediction loss. The weights are generally initialized randomly. Then, inputs and weights are multiplied and passed to the forward layers. This phenomenon is called the forward pass. Following that, the error is computed based on the result of the output layer and the actual output. As mentioned before, the goal is to minimize the error; then, since the actual output is constant, the result of the output layer has to be modified. By decomposing the prediction of the neural network, it is found that weights are the variable components of it. Backpropagation aims to update network weights using gradient descent. It computes the gradient of the error prediction with respect to the weights. This computation is applied backwards through the network [64]. There are different kinds of DNNs that are explained below.

3.3.1 Fully-Connected Neural Networks (FCNN)

These are the simplest kind of neural network. Each neuron of one layer is connected to every neuron in the next layer. An example of FCNN is represented in 3.1b.

3.3.2 Convolutional Neural Networks (CNN)

They are mostly used in pattern recognition tasks for images. The input of a CNN is an image. This network is composed of three types of layers. First, convolution layers compute the output of neurons by focusing on local regions of the input. It is applied by multiplying the weights of the neurons with the values linked to the specific local region and summing them up (convolution operation). Secondly, the pooling layer will reduce the size of a given input image. Finally, the fully-connected layers act the same as layers in more classical neural networks [65]. CNNs are particularly successful at computer vision tasks such as face detection, handwritten character recognition, or X-ray image analysis.

3.3.3 Attention and Transformers

Visual attention seeks the answer to two questions: where and what to look at. The studies around that subject inspired the artificial intelligence field. The attention idea came from [66]. Attention has an encoder-decoder structure. This mechanism seeks to focus on specific parts of the input. It assigns different weights, determining the importance of each part of the input. This results in the model assessing and focusing on the most relevant information [41]. It is especially used for tasks such as natural language processing or computer vision, but less for tabular data. A particular example of attention is transformers, which employ self-attention and are especially used for sequential data [67], but as seen in the Related Work section, it has been tested on tabular data. Transformers have multiple real-world applications, including machine translation, question answering, and text generation.

Chapter 4

Issues with Tabular Data

Deep Learning models underperform compared to traditional machine learning models when applied to tabular data. This section highlights the characteristics of tabular data that make it challenging for deep neural networks.

4.1 Data-Related Issues

Tabular data contains missing values, outliers, or erroneous data. All of these are issues for machine learning algorithms, but traditional algorithms such as Random Forest or XGBoost can handle them [5].

The structured and abstract nature of tabular data makes it less interpretable for humans. As a consequence, domain expertise is required to make sense of the data. For instance, recognizing a dog with an image is intuitive, but doing that with some variables from a structured dataset is more challenging [54].

Neural networks need complete data. Handling missing values for NN implies filling values with imputation methods such as mean or k-NN(nearest neighbour) imputation, contrary to traditional methods, which have incorporated systems to deal with missing data. Moreover, these imputation methods can become computationally expensive, particularly when there is a lot of missing data. It highlights that handling missing data for deep learning remains a significant challenge. This is proven by Marek Smieja et al [68], who introduced a methodology to input missing data into neural networks. Indeed, through experiments, their methodology performed the best among other imputation methods, which proves that just filling in missing data is not the best approach. Building a model able to train even with tabular data is a benefit. As said in the Related Work in Self-supervised part, the pretext task of their model was to reconstruct missing points, which helps the model to better understand the patterns of tabular data, even if tabular data contains missing values. The model performed better than XGBoost on the datasets chosen in the paper [48].

Tabular data can often be imbalanced, *i.e.*, a class is more present. This results in always outputting the majority class to achieve high performance, but as measured by accuracy, this results in a useless model predicting only one class. Moreover, through an experiment of Han-Jia Ye [69], the models that were able to handle class imbalance performed better.

In tabular data, there are relationships between attributes. Understanding and modelling these dependencies is crucial but challenging because the relations between features are not always visible from observed data [70]

4.2 Architectural Limitations of Neural Networks

Additionally, deep neural networks (DNNs) face challenges when learning from dense numerical attributes because of the complexity of optimization hyperplanes, which are high-dimensional spaces where each dimension represents a model feature. Consequently, the risk of converging towards a local minimum is increasing. [70]

- The weight Matrix represents a linear transformation, which preserves convexity when applied to convex functions.
- Convex Nonlinear Element: non-linear activation functions such as ReLU or tanh are locally convex (*e.g.* convex in a certain region).

While each operation may be convex or locally convex, the composition of these operations across multiple layers leads to a non-convex function. The combination of non-linear activation functions with linear operations results in a complex, non-linear output. The more the depth increases, the more complex the network becomes. This results in a complex landscape with multiple local minima and saddle points, which are critical points that are neither maxima nor minima.

Moreover, optimization hyperplanes become more complex as the number of numerical features increases, since they are a high-dimensional space where each dimension is a numerical feature. Additionally, features could be correlated, and that should be taken into account in the computation of the hyperplanes. Thus, it could be challenging because the relation between some features is not directly visible from observed data; the combination of all creates more local minima and hardens the path to the solution [71].

MLP(Multi-Layer perceptron)-like neural networks are more affected by uninformative features than traditional algorithms. Indeed, removing uninformative features reduces the performance gap between MLPs and other methods, but adding more increases this gap. This proves that MLP-like neural networks are less robust to uninformative features [13]. That is problematic since the presence of uninformative features is quite common in tabular datasets.

When rotating the features of both training and testing sets, the training process that trains an MLP is unchanged. This is a characteristic of the rotationally invariant method [72]. Moreover, Ng [72] shows that any rotationally invariant training process has a worst-case complexity that rises at least linearly in the number of irrelevant features. When features are rotated, a rotationally invariant algorithm can't directly tell if a feature is important or not based on its direction. Removing uninformative features requires a rotationally invariant algorithm to find the original directions of features and then remove the least informative ones [13]. This is problematic in high-dimensional datasets because many features could be irrelevant.

Some papers claimed their deep learning model outperformed SOTA methods, namely XGBoost. But other papers proved that these methods performed worse, especially on datasets not chosen in the papers. This can be explained by the fact that hyperparameter optimization is different from papers based on one DNN method and papers trying to do a benchmark. Papers on one DNN method should have a broader hyperparameter search. Moreover, in real-life applications, it is needed that the optimization time is not too long, and XGBoost does not take too much time to converge towards good performance [23].

Also, tree-based methods require fewer computational resources than deep learning methods [13].

The choice of the activation function plays a crucial role, as it impacts the performance of a deep learning model. Indeed, it has been found that the ReLU activation function leads to the best performances [73].

In summary, while deep learning presents great promise in general, its application to tabular data is complicated due to several challenges ranging from data quality issues to architectural limitations.

Chapter 5

Data Collection and Preparation

This section presents the datasets chosen for this thesis, as well as how they have been preprocessed and prepared before model training.

5.1 Datasets

For an empirical study, those regression datasets have been selected [74].

Dataset name	Number of samples	Number of features
cpu_act	8192	21
pol	15000	26
elevators	16599	16
wine_quality	6497	11
Ailerons	13750	33
yprop_4_1	8885	42
houses	20640	8
houses_16H	22784	16
delays_zurich_transport	5465575	8
diamonds	53940	6
Brazilian_houses	10692	8
Bike_Sharing_Demand	17379	6
nyc-taxi-green-dec-2016	581835	9
house_sales	21613	15
sulfur	10081	6
medical_charges	163065	3
MiamiHousing2016	13932	13
superconduct	21263	79

Table 5.1: Datasets used for the thesis

The datasets follow criteria [13] :

- Not high-dimensional: The number of dimensions should be much smaller than the number of observations.

- Undocumented datasets: Datasets with too little information are removed.
- Independent and Identically Distributed Data
- Real-world data: Artificial datasets are removed, but some simulated datasets are kept because studying them can be beneficial.
- Not too small: Datasets with too few observations (< 3000) and too few dimensions (< 4) are discarded.
- Not deterministic: Datasets for which the target is a deterministic function of the data. This relates to poker or chess datasets.

5.2 Preprocessing

The same preprocessing as in the benchmark [13] has been applied to compare the methods in this paper with their results.

- Medium-sized training set: The training set is truncated to 10,000 samples.
- No missing data: first, columns where there is too much missing data are removed, then when a row contains missing data, it is deleted too.
- High cardinality numerical features: Features with less than 10 unique values are removed.

5.3 Data preparation

- For NN models, the features are Gaussianized with the QuantileTransformer from Scikit-Learn. In other words, the features are scaled in a way that makes the data look like a normal distribution.
- Target variables are log-transformed when their distributions are heavy-tailed. To check that, two values have been computed. First, the skewness, which measures how far our distribution is from the normal distribution. A positive skewness indicates that the right tail is longer than the left, which means that values are mainly on the left side of the mean. Conversely, a negative skewness indicates that the left tail is longer than the right, which means that values lie more on the right side of the mean. The first condition to verify is that the absolute value of the skewness exceeds a chosen value. The peak of the tail has to be checked as well. This is possible with the kurtosis measure. The skewness of X is measured as

$$skew(X) = \mathbb{E}\left[\left(\frac{X-\mu}{\sigma}\right)^3\right]$$

  A distribution will be heavy-tailed if the kurtosis is bigger than a certain threshold and the absolute value of the skewness exceeds another threshold, as said before. The kurtosis of X is measured as

$$kurt(X) = \mathbb{E}\left[\left(\frac{X-\mu}{\sigma}\right)^4\right]$$

In the implementation, to be heavy-tailed, the skewness has to exceed 0.5 and the kurtosis 3.

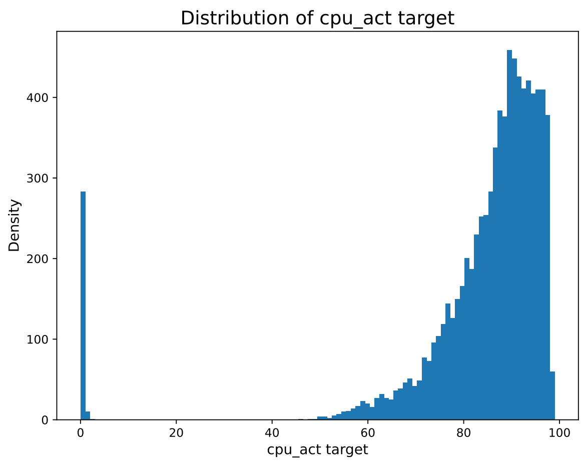


Figure 5.1: Histogram of the target variable in the *cpu_ act* dataset

As can be seen from Figure 5.1, the distribution of the target variable is heavy-tailed. Moreover, this is confirmed by the skewness $= -3.4$ and the kurtosis $= 12.7$.

Chapter 6

Methodologies

This section presents the methodology to compare various approaches on tabular data. The models considered are a mix of SOTA (state-of-the-art) tree-based, namely XGBoost and RandomForest, which have been discussed in the Introduction Section, and some deep learning models. The deep learning models are LassoNet, MLP, RealMLP, FT-Transformer, and TabNet. LassoNet was selected because it uses feature selection, MLP because it is one of the simplest and most widely used deep learning models, RealMLP because the developer of the model stated that this model could compete with tree-based models when applied to tabular data, FT-Transformer because it leverages self-attention mechanisms to model complex features, and TabNet was selected because it combines competitive performance with interpretability. Beyond model comparison, LLMs will be used for tabular data generation will be also explored, as well as interpretability with SHAP. Tree-based methods are not described in detail, as the primary focus lies on deep learning.

6.1 Compared Methods

6.1.1 Random Forest

Random Forests employ the concept of bagging, creating multiple datasets with random sampling with replacement, known as bootstrap sampling. For each training dataset, a decision tree is built. Each node is split using a different subset of features to reduce correlation between trees. When a new data point is met, it is passed through all trees of the forest. For classification tasks, predictions are made by majority voting among the trees. While, for a regression task, the averaged prediction between all trees in the forest is computed. RFs reduce overfitting by aggregating decision trees, which results in lower variance but not bias. A small number of decision trees can lead to high variance because the model may overfit the training data, including noise. Increasing the number of decision trees improves the stability of the forest. Additionally, RFs can deal with missing data using surrogate splits: if a tree splits on a feature (primary split) but some samples have missing points on that feature, the tree will find an alternative feature split (surrogate split) which separates the data similarly, and the tree will use this split only for samples where the primary split is missing [75].

6.1.2 XGBoost

XGBoost is a widely used implementation of GBDT that combines multiple weak learners, which are generally decision trees, to form a strong predictive model for tabular data. GBDTs are built sequentially, with each tree trained to correct the errors made by the previous one. This iterative process progressively reduces errors, thereby improving model performance. In assembling weak learners, a strong learner is constructed that performs well and is robust [10]. One of the most well-known models in the Gradient Boosting Decision Trees (GBDT) family is XGBoost. Dealing with missing data could be challenging, but XGBoost demonstrated robustness against it. It eliminates the need to handle missing data manually. Indeed, this machine learning algorithm learn from missing values. XGBoost includes built-in mechanisms to detect overfitting by implementing early stopping criteria [75].

6.1.3 LassoNet

The first method explored is the LassoNet, which is a combination of Lasso (Least Absolute Shrinkage and Selection Operator) regression and feature sparsity to feed-forward neural networks. This method is designed for the network to use only a subset of features [2]. Similar to Lasso regression, LassoNet assigns zero weights to irrelevant features. Feature selection is achieved by adding an input-to-output skip-layer (residual), where a feature participates in any hidden layer only if its skip-layer representative is active. In other words, if it's active, the features can bypass the hidden layers and directly impact the output.

This method enables feature selection. High-dimensional datasets often include irrelevant and redundant features that can negatively impact the performance of a machine learning process. Identifying meaningful features requires domain knowledge or the help of an expert, but for large datasets, this process becomes impractical. As a result, a more systematic approach is needed. By filtering out the irrelevant features, the learning speed will increase, it will reduce overfitting, and create simpler models [76].

Architecture

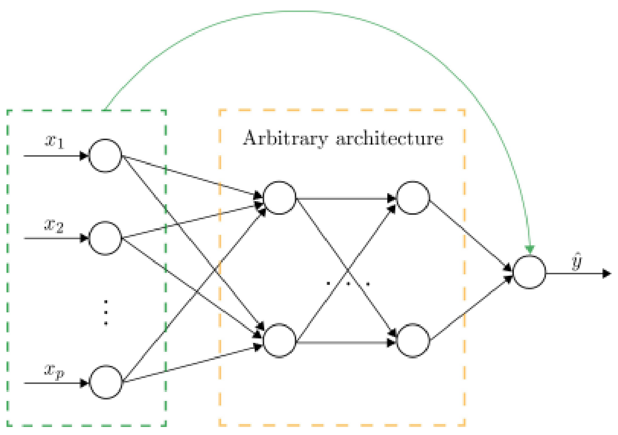


Figure 6.1: LassoNet Architecture: This consists of a single residual connection from the inputs to the output, as shown in green, and an arbitrary feed-forward network [2].

Problem Formulation

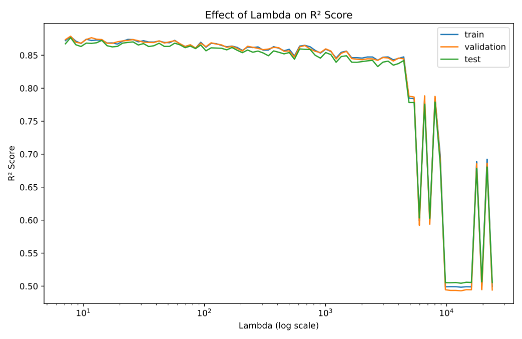
The general architecture of LassoNet is presented in Figure 6.1, the objective function is defined as [2], where $L(\boldsymbol{\theta}, W)$ is the empirical loss in the training data, $\lambda\|\boldsymbol{\theta}\|_1$ is the l_1-penalty on $\boldsymbol{\theta}$. $\boldsymbol{\theta}$ is a vector of skip-layer weights that encode the linear contribution of each feature to the output. If $\theta_j = 0$, feature j will not contribute to the output. W is a weighting matrix in a deep neural network. $W_j^{(1)}$ denotes the weights for feature j in the first hidden layer.

$$\underset{\boldsymbol{\theta}, W}{min}\ L(\boldsymbol{\theta}, W) + \lambda\|\boldsymbol{\theta}\|_1, \text{ subject to } \|W_j^{(1)}\|_\infty \leq M|\theta_j| \tag{6.1}$$

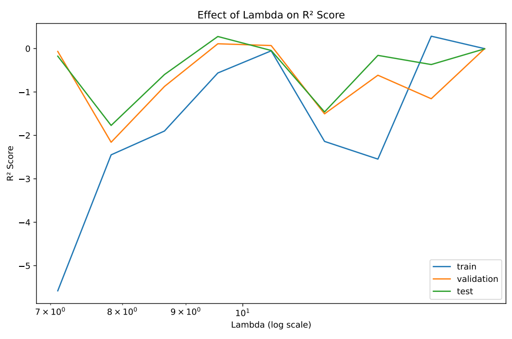
In practice, the vector of skip-layer weights $\boldsymbol{\theta}$ as well as the network weight matrix W are learned through training by minimizing the objective function 6.1.

Hyperparameter optimization

As mentioned in [2], the hyperparameters to optimize are the l_1-penalty coefficient, λ, and the hierarchy coefficient, M.



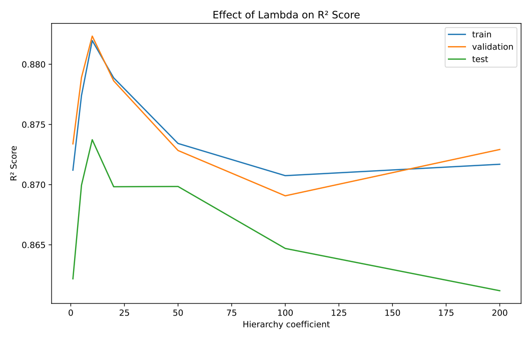
(a) Lambda effect on R^2 score on the *Ailerons* dataset



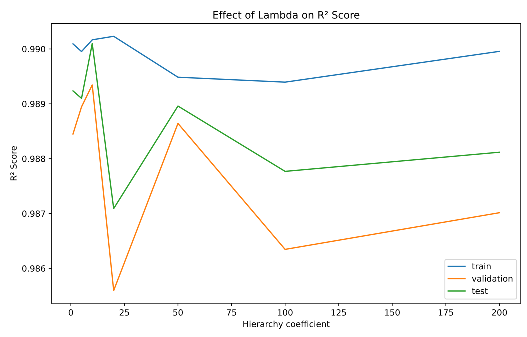
(b) Lambda effect on R^2 score on the *yprop_4_1* dataset

Figure 6.2: Lambda effect on R^2 score

In Figure 6.2a, it can be seen that the R^2 score decreases as the lambda increases. This behaviour is justified by the fact that the model becomes sparser (*e.g.*, contains fewer variables) as the l_1-penalty coefficient grows.



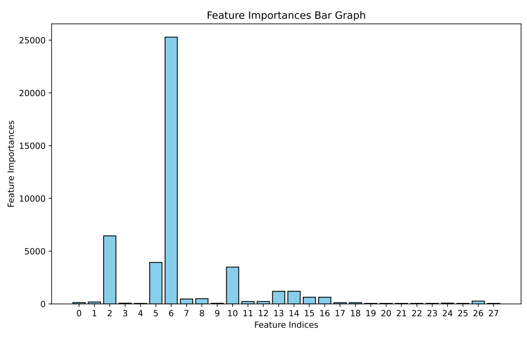
(a) Hierarchy coefficient (M) effect on R^2 score on the *Ailerons* dataset



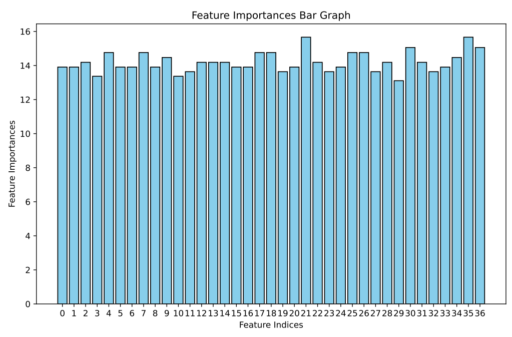
(b) Hierarchy coefficient (M) effect on R^2 score on the *yprop_4_1* dataset

Figure 6.3: Hierarchy coefficient (M) effect on R^2 score

It can be seen from Figure 6.3 that the hierarchy coefficient has its impact too. Those figures prove that those 2 hyperparameters have to be tuned to have the best result.



(a) Importance of features on the *Ailerons* dataset



(b) Importance of features on the *yprop_4_1* dataset

Figure 6.4: Importance of features

An interesting property of this method is that it allows the importance of the features to be computed through the weights $\boldsymbol{\theta}$. In the *Ailerons* dataset, there is a variable that dominates the model, as can be seen in Figure 6.4a. Thanks to this image, the behavior of the graph in Figure 6.2a is logical. The plot remains steady until it reaches a threshold, at which it declines because the most important variable is removed from the model. Figure 6.4b describes another case in which all variables have the same importance. As a result, when the penalty coefficient is over the importance, all variables will be removed from the model. This explains the chaotic behavior of the graph presented in Figure 6.2b.

6.1.4 Improved MLP

This method has been introduced by David Holzmüller et al. [27]. Numerical features are processed as follows : Let $x_1, \ldots, x_n \in \mathbb{R}$ be the values of a column, q_p be the p-quantile of $(x_1, \ldots, x_n)$ where $p \in [0, 1]$

$$x_{j,processed} := f(s_j \cdot (x_j - q_{\frac{1}{2}})), \quad f(x) = \frac{x}{\sqrt{1+\left(\frac{x}{3}\right)^2}}$$

$$s_j := \begin{cases} \frac{1}{q_{\frac{3}{4}} - q_{\frac{1}{4}}} & \text{if } q_{\frac{3}{4}} \neq q_{\frac{1}{4}} \\ \frac{2}{q_1 - q_0} & \text{if } q_{\frac{3}{4}} = q_{\frac{1}{4}} \text{ and } q_1 \neq q_0 \\ 0 & \text{otherwise} \end{cases}$$

In practice, using scikit-learn [77], this process is similar to applying RobustScaler for the first case or MinMaxScaler for the second case. The output of the function f lies within the range $[-3, 3]$. These kinds of functions are called smoothing clipping functions, and they have been applied by David Holzmüller et al. [78] and Hafner et al. [79]. Smooth clipping prevents large outliers from influencing the result too badly, while robust scaling enables the inlier points not to be affected by the outliers. Then, periodic bias linear DenseNet (PBLD) embeddings combine the original numerical feature with the PL embeddings introduced by Gorishniy et al. [4] and add a different embedding with biases inspired by Huang et al. [80] and Rahimi and Recht [81]. In PBLD embeddings, each feature is passed through a 2-layer MLP, where $\mathbf{w}_{emb}^{(1,i)}, \mathbf{b}_{emb}^{(1,i)} \in \mathbb{R}^{16}$, $\mathbf{b}_{emb}^{(2,i)} \in \mathbb{R}^3$, $\mathbf{W}_{emb}^{(2,i)} \in \mathbb{R}^{3\times16}$, i denotes the feature, 1 or 2 indicates the layer number.

$$\left(x_i, \mathbf{W}_{emb}^{(2,i)} \cos\left(2\pi \mathbf{w}_{emb}^{(1,i)} x_i + \mathbf{b}_{emb}^{(1,i)}\right) + \mathbf{b}_{emb}^{(2,i)}\right) \in \mathbb{R}^4$$

In Python, and for this thesis, this method will be named RealMLP since it is the name of the model in the Python package.

6.1.5 TabNet

This method was proposed by Arık et al. [3]. TabNet has raw tabular data as input. In other words, the input dataset is the actual dataset without preprocessing, such as normalization or missing value removal. Moreover, it is trained using gradient descent and employs sequential attention to choose which features to use at each decision step. Sequential attention is better than naive feature selection methods using attention. Those naive proceedings can select redundant features or can miss features that are only meaningful with others [82]. It enables interpretability, as features used at each decision step are visible, and helps the model to focus on relevant features, which can improve learning efficiency. The feature selection is instance-wise, meaning that it can differ for each input. It can be said that TabNet acts the same way as a decision tree, with the fact that features are chosen for each decision step. Raw tabular data is often sparse, which makes it difficult to process. However, TabNet uses masking to reduce the dimensionality of the data in order to solve the sparsity issue. The method uses unsupervised pre-training, which was proved by Erhan et al. [49], to help deep learning. Indeed, unsupervised pre-training leads to better starting points, resulting in better generalization. It also helps neural networks avoid poor local minima, whose probability is increasing, as well as the depth of the network. Unsupervised pre-training acts in a unique form of regularization by guiding the model to regions of parameter space that generalize better. TabNet is an encoder-decoder architecture.

Encoder

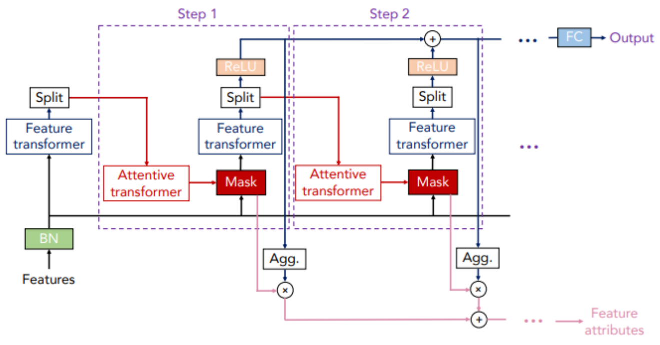


Figure 6.5: TabNet encoder reproduced from Arik [3]

The encoder described in Figure 6.5 contains two types of transformers and a mask. There are the feature transformer and the attentive transformer. The output of the feature transformer is split in half. One part is for the fully-connected (FC) layer, further, and the other part is for the attentive transformer. The FC layer concatenates all splits from the feature transformer and outputs them. Concerning the attentive transformer, it generates mask values that are then used to select the most relevant subset of features at each step. By combining mask values, some features can be prioritized in the training based on their importance. This methodology enhances the performance of the model as well as its interpretability. The feature transformer is trained on features that were filtered by the mask. The role of the transformer is to turn an input space into an embedding space. Since there are decision steps in TabNet, it is logical that the architectures will contain as many encoders as decision steps. All these encoders are stacked and passed through an FC Layer. The attentive transformer computes the weights representing the importance of each part of the input, which helps the model to focus only on the most important parts. Thanks to this process, the next encoder selects the most useful parts made possible by the attention weights from the previous embedding space. This process is applied in series.

$$M[i] = sparsemax(P[i-1] \cdot h_i(a[i-1])) \tag{6.2}$$

$$P[i] = \sum_{j=1}^{i} (\gamma - M[j]) \tag{6.3}$$

Equation 6.2 expresses how the current-step mask is computed from the previous step. $P[i]$ is the prior scale term, which denotes how a feature has been used previously. This can be computed using equation 6.3. $a[i-1]$ is the output of the attention transformer at the previous step, and h_i is a trainable function. Sparsemax is an activation function similar to softmax, but it can output sparse probabilities($P = 0$), whereas softmax cannot, since it always returns probabilities bigger than zero. γ in equation 6.3 is a parameter called the relaxation parameter. The more it increases, the more a feature can be used in decision steps.

Decoder

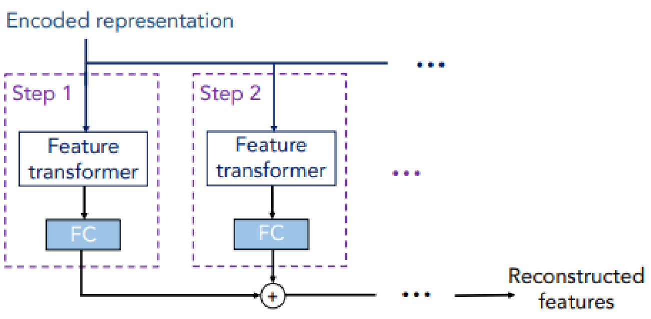


Figure 6.6: TabNet decoder reproduced from Arik [3]

The decoder shown in Figure 6.6 is employed to reconstruct the original features from the encoded representations. During the reconstruction process, a reconstruction loss is computed between the input data and the output of the decoder. The decoder is composed of layers that are similar to the feature transformer layers in the encoder. The reconstruction loss encourages the encoder to capture informative and compressible representations of input data. Instead of selecting which features to focus on for decision-making, the decoder uses a reserved-to-the-encoder attention mechanism to recover the input data. This directs the decoder to prioritize reconstructing the most important features.

6.1.6 FT-Transformer

FT-Transformer (Feature Tokenizer + Transformer) is an adaptation of the transformer architecture for tabular data. This model converts all features into embeddings and applies a stack of transformer layers to them [4].

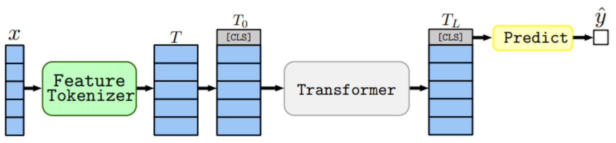


Figure 6.7: FT-Transformer architecture reproduced from Gorishniy [4]

Figure 6.7 describes the key features of the FT-Transformer model. It can be seen from this figure that it features [CLS] tokens. This type of token is put at the beginning of the sequence, and its purpose is to summarize the input sequence. It is also used for other transformers such as BERT [83] or SAINT [42].

Feature Tokenizer The feature tokenizer proceeds as follows: [4]. It transforms input features x into embeddings T. The embedding for a given feature (x_j):

$$T_j = b_j + f_j(x_j) \in \mathbb{R}^{k \times d}$$

k denotes the number of features, b_j denotes the feature-j bias, f_j represents the element-wise multiplication with the weight matrix $W_j^{(num)}$.

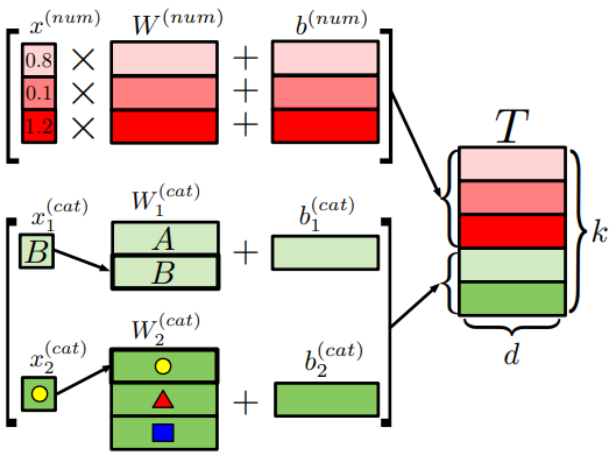


Figure 6.8: Feature Tokenizer reproduced from Gorishniy [4]

$$T_j = b_j^{(num)} + x_j^{(num)} \cdot W_j^{(num)} T = stack\left[T_1^{(num)}, \ldots, T_k^{(num)}\right]$$

The equations above describe what happens in the Feature Tokenizer represented in Figure 6.8. Although the feature tokenizer can also process categorical features, the focus will be only on numerical features.

Transformer After feature tokenization, the $[CLS]$ token is added at the beginning to form a bigger embedding. After that, it is passed through N transformer layers $F_1, F_2, \ldots, F_N$.

As it can be seen from Figure 6.7, the final $[CLS]$ token is used for the final prediction

$$\hat{y} = Linear(ReLU(LayerNorm(T_L^{[CLS]})))$$

$$T_0 = stack\,[CLS, T] \quad T_i = F_i(T_{i-1})$$

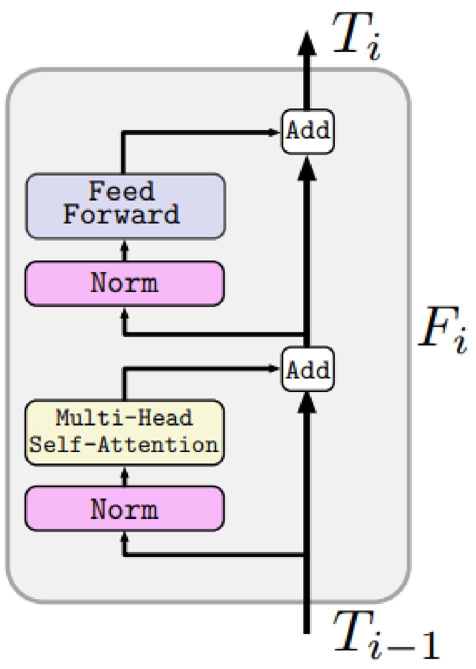


Figure 6.9: Transformer Layer reproduced from Gorishniy [4]

Figure 6.9 illustrates a transformer layer. Multi-head self-attention extends self-attention by performing multiple attention operations in parallel. Each self-attention run is called a self-attention head, which captures different relationships in the input. The output of all the heads is then concatenated to produce the final prediction [84].

6.1.7 Hyperparameter tuning

For hyperparameter tuning, random search has been used. Indeed, parameter grids were constructed for each model, and a fixed set of 50 configurations was built for each model. This number offers a good coverage of the hyperparameter space while keeping the computation cost reasonable. Moreover, cross-validation was applied following the approach of [13]. The dataset was first split into training (70%) and testing (30%) sets. The testing set was then split into validation (30% of the testing set) and the remaining forms the final testing set. For larger datasets, the training set is truncated to 10,000 samples for computation efficiency. The folds are created before training to ensure that all models are trained and tested on the exact splits. The results from multiple folds are then averaged to obtain a stable estimate of model performance. The number of folds is dependent on the size of the testing set

- If there are 6000 or more samples, 1 fold
- If there are between 3000 and 6000 samples, 2 folds
- If there are between 1000 and 3000 samples, 3 folds
- If there are fewer than 1000 samples, 5 folds

The range of hyperparameters explored is provided in the Appendix at the end of the thesis.

6.2 Large Language Models (LLM)

One potential approach to improve the performance of deep learning models on tabular data (TD) could be to use the combination of an LLM and a deep neural network. The lack of large tabular datasets [1] may be a reason why deep learning methods fail to outperform traditional methods, as deep learning requires a lot of data to be effective [85]. A possible solution is to first process TD through an LLM for data augmentation [55], generating new rows based on the current dataset. The augmented dataset can then be used as an input for a DNN to assess whether performance improves with the amplification of the dataset. However, there are three drawbacks of LLMs for tabular data [1].

- LLMs are not good at modeling continuous variables. Indeed, it is due to the tokenization of numerical values [86]. For instance 3.86 will be tokenized as 3 $\rightarrow$. $\rightarrow$ 86. 3.86 and 3.87 should have about the same probability to be generated, contrary to 3.86 and 116.86, but in both cases, only one token differs. For that reason, unrealistic outputs appear [87].
- Studies have proven that LLMs produce outputs whose confidence levels do not match the true underlying distributions. In other words, when an LLM assigns a probability to a prediction, this probability does not consistently match the actual probability of the event. They struggle with sampling theoretical distributions, including uniform distribution, for example [88]. This raises their inability to model more complex distributions.
- LLMs are computationally expensive. Tabular data often contains even a hundred or more features, which can result in slow, inefficient, and expensive training and inference. Nevertheless, there exists a solution to improve efficiency. For instance, smaller transformers may suffice since tabular data is highly structured [1].

Tabular data offers some challenges for LLMs [89]:

- An expensive preprocessing is required to generate tabular data. More precisely, it includes data normalization, handling of categorical values, handling missing values, and removing outliers. Lossy preprocessing can occur because it can cause the loss of important information about the original dataset.
- Context between features has to be taken into account when realizing synthetic data from tabular data, commonly referred to as the contextual knowledge problem.
- Currently, the majority of the models have to be retrained for each set of arbitrary features to be conditioned on. In other words, if "Age" and "Weight" are the conditioned features, and after that it has been decided that this will be "Height" and "Name", the model has to be trained again from nothing. When a model is flexible, independent of a set of features, it is said that the model enables arbitrary conditioning.

6.2.1 Method

The method to generate tabular data used for this thesis is the same as [34]

1. The encoding is done simply. The combination of x_i, where x represents the input variables and i the row index, and y_i, where y represents the target variable, is turned into a sentence. Indeed, the sentence for the row i is : $s_i =$ "X_1 *is* $a_{i,1}$, X_2 *is* $a_{i,2}\ldots$, X_M *is* $a_{i,M}$, Y *is* y_i". $\{X_1, X_2, \ldots, X_M\}$ are feature names, $a_{i,j}$ is the value from row i and feature j and Y is the target variable name. By including feature names in the encoding, it enables to better understanding of the relationships between the features, addressing the context knowledge problem mentioned previously [89].

2. LLM methods permute both features and target for arbitrary conditioning. It denotes the ability to produce data based on an arbitrary set of variables. However, in this context, this could be a drawback. In fact, in the attention matrix of LLM, each token only refers to tokens before it, but not after it. So, if the target variable is put, by shuffling, at the first positions of the sentence, it will not be able to learn from all the input features. Therefore, permuting both the target variable and the features will make the LLM fail to identify the relationship between X and Y. A solution to this is to only shuffle the features and to always put the target variable at the end. Both the initial encoding and the permuted encoding are put in the LLM.

3. Fine-tuning: The model is trained to predict the next word (or token) in a sequence, given all the words that came before it. This is called an auto-regressive approach. First, each sentence is turned into a sequence of tokens:

   $$tokenize(s_i) = \{t_1, t_2, \ldots, t_m\}$$

   In an auto-regressive model, the probability of the entire sequence is expressed as the product of the probabilities of each token given the previous tokens.

   $$p(s_i) = p(t_1, t_2, \ldots, t_m) = \prod_{k=1}^{m} p(t_k|t_1, \ldots, t_{m-1})$$

   The model learns to predict each token based on all prior tokens in the sequence.

4. Sampling $\hat{x}$: To generate a synthetic row $\hat{x}$ using the LLM, feature-conditional sampling is used. It exists different techniques to perform it.

   - Feature Name Preconditioning: The LLM is only fed with feature names in order. The LLM autoregressively generates values for each feature one by one. The model predicts a value for the first feature, then moves to the second feature, generating the second value using the previously predicted value as context. This process continues until all values of features have been generated.
   - Name-Value Pair Preconditioning: For each feature in the dataset, a value is drawn from its distribution and used as a condition to generate a row. For example, if the feature is Age and then the value drawn from the distribution is 35, then the condition for the LLM becomes *Age is* 35, similar to the encoding phase.
   - Multiple Name-Value Pair Preconditioning: It is the same as Name-Value Pair Preconditioning except it is performed with multiple name-value pairs as a total generation condition.

5. Querying $\hat{y}$: After generating $\hat{x}_i$, it is encoded and used as a prompt to sample $\hat{y}$. By applying this approach, the LLM can use all the attention links between all features to generate more accurate target results. Furthermore, the LLM acts as a predictive model as $\hat{y}$ is generated by a conditional distribution $p(\hat{y}|\hat{x})$.

6.3 SHAP

The unified method for interpreting the model's output is based on [53]. SHAP (SHapley Additive exPlanations) values are a method to explain any machine learning outcome. It assigns values corresponding to the importance of each feature in the final output. These values indicate how each feature affects model decisions as well as the significance of each feature compared to others. SHAP values are model-agnostic, meaning that they are usable for any kind of model: Decision trees, Linear regression, Random Forests, GBDT, and Neural Networks. In this paper, complex methods are presented, meaning that the best explanation of these models cannot be themselves, contrary to simple models such as linear models. For these complex models, it is required to use simpler explanation models that denote any interpretable approximation of the complex model. f is the original model and g the explanation model. Explanation models often use simplified inputs x', which are generally binary. SHAP is an additive feature attribution method

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i$$

where ϕ_i denotes the effect of feature i on the model and z' is a binary simplified input vector $z' \in \{0,1\}^M$ and M the number of simplied input features. The z' values are on/off switches for each feature. If $z'_i = 1$, it means that the feature i is included in the explanation; if $z'_i = 0$, the feature i is excluded. z' vectors are created by turning each feature on and off in all possible combinations, so the effect of including or excluding each feature on the model's prediction can be observed.

SHAP has properties that are explained below [53]

- Local accuracy: Local accuracy requires the explanation model g to match the output of the original model f for the simplified input x' where x is the original input.

$$f(x) = g(x') = \phi_0 + \sum_{i=1}^{M} \phi_i x'_i$$

- Missingness: The missingness property implies that missing features in the original input do not affect the final output. In this case, simplified inputs correspond to feature presence, meaning if $x'_i = 0$ then feature i is missing in the original input.

$$x'_i = 0 \quad \rightarrow \phi_i = 0$$

- Consistency: SHAP values provide a good overview of the model's behaviour even if the parameters or the architecture change.

However, computing SHAP values is challenging, but they can be approximated by using information gained from additive feature attribution methods. There exist different types of SHAP depending on the model used. While Kernel SHAP is adapted for any machine learning model, there exist more model-specific SHAP

- Linear SHAP for linear models
- Tree SHAP for tree-based models [90]
- Deep SHAP for neural networks

Chapter 7

Results

7.1 Model comparison

The model comparison has been conducted on seven models: Random Forest, XGBoost, MLP, RealMLP, FT-Transformer, TabNet, and LassoNet. The models are ranked by their R^2 test values. To evaluate the performance, the coefficient of determination (R^2) is used. This metric measures the proportion of variance in the predicted variable that can be explained by the inputs. R^2 effectively summarizes the model performance and can be used to compare multiple methods on the same dataset. Finally, it is widely used across datasets. The test value has been chosen because it demonstrates the model's ability to generalize with unseen data. Rank 1 is assigned to the model with the highest R^2 value and Rank 7 to the lowest. Ranks are averaged over datasets to obtain average ranks for each model. Below, the practical implementations of the methods are described

- LassoNet: Implemented with LassoNetRegressor from the LassoNet Github : (insert link)
- RealMLP: Implemented with RealMLP_TD_Regressor from the pytabkit package.
- MLP: Implemented with MLPRegressor from scikit.
- RandomForest: Implemented with the RandomForestRegressor from scikit.
- XGBoost: Implemented with the XGBoostRegressor from scikit.
- TabNet: Implemented with the TabNetPretrainer for unsupervised training, and TabNetRegressor from the pytorch_tabnet package.
- FT-Transformer: Implemented with FTTransformer from rtdl_revisiting_models. The model was trained using a created function because it is the only model that does not have a built-in fit function. The parameters of the model have been set with the help of (link)

	RandomForest	XGBoost	RealMLP	FT-Transformer	MLP	TabNet	LassoNet
cpu	0.983 (2)	0.984 (1)	0.983 (2)	0.976 (5)	0.975 (6)	0.977 (4)	0.968 (7)
pol	0.988 (3)	0.988 (4)	0.995 (1)	0.99 (2)	0.948 (5)	0.928 (6)	0.92 (7)
ele	0.883 (6)	0.885 (1)	0.881 (2)	0.872 (3)	0.867 (4)	0.859 (5)	0.572 (7)
wine	0.547 (1)	0.463 (2)	0.423 (4)	0.419 (5)	0.428 (3)	0.4 (7)	0.417 (6)
Ail	0.83 (3)	0.819 (7)	0.834 (1)	0.83 (2)	0.828 (4)	0.828 (5)	0.827 (6)
ypr	0.052 (1)	0.018 (2)	-0.000 (3)	-0.004 (5)	-0.055 (6)	-0.001 (4)	-0.203 (7)
hou	0.844 (1)	0.841 (2)	0.841 (3)	0.837 (4)	0.819 (5)	0.806 (6)	0.801 (7)
h_H	0.567 (1)	0.524 (2)	0.211 (4)	0.142 (6)	0.218 (3)	0.034 (7)	0.205 (5)
del	-0.012 (7)	0.03 (1)	0.024 (3)	0.024 (4)	0.019 (6)	0.025 (2)	0.02 (5)
dia	0.943 (5)	0.945 (1)	0.945 (2)	0.945 (3)	0.944 (4)	0.941 (7)	0.943 (6)
Braz	0.993 (6)	0.994 (5)	0.996 (3)	0.997 (2)	0.997 (1)	0.992 (7)	0.995 (4)
Bik	0.679 (4)	0.688 (1)	0.681 (2)	0.681 (3)	0.663 (5)	0.65 (7)	0.662 (6)
nyc	0.54 (1)	0.513 (2)	0.486 (3)	0.454 (6)	0.464 (4)	0.454 (7)	0.459 (5)
sales	0.875 (5)	0.887 (2)	0.888 (1)	0.883 (3)	0.872 (6)	0.876 (4)	0.87 (7)
sul	0.859 (2)	0.889 (1)	0.751 (6)	0.773 (4)	0.79 (3)	0.772 (5)	0.715 (7)
med	0.978 (7)	0.979 (5)	0.98 (1)	0.98 (2)	0.98 (3)	0.979 (6)	0.979 (4)
Mia	0.918 (4)	0.922 (2)	0.926 (1)	0.921 (3)	0.914 (5)	0.91 (7)	0.914 (6)
sup	0.909 (1)	0.905 (2)	0.9 (3)	0.892 (5)	0.9 (4)	0.867 (7)	0.889 (6)
avg rank	3.333	2.389	2.556	3.722	4.278	5.722	6.0

Table 7.1: Model performance by dataset

The values from Table 7.1 have been rounded to 3 decimals for the sake of readability, but the ranking is based on the exact values. The values in brackets in the table are the rank of the method for a dataset. It can be seen from Table 7.1 that XGBoost is the best-performing model across all datasets, with RealMLP close behind. RandomForest is third but very close to FT-Transformer. After that, the ranking is MLP, then TabNet, and finally LassoNet. From the ranking, it can be seen that a deep learning model (RealMLP) is outperforming an SOTA model (RandomForest) in dealing with tabular data. Moreover, its performance is close to XGBoost, meaning that this model is promising. FT-Transformer also looks encouraging; however, for MLP, LassoNet, and TabNet, they do not look close to approaching the overall performance of SOTA methods.

Once the average ranks are obtained, the Friedman test is applied to them following the guidelines in [91]. It will be used in this case to see whether there is a significant difference between the models. The null hypothesis states that all algorithms are equivalent. As a consequence, rejecting the null hypothesis means that at least one algorithm performs significantly better than others. The Friedman test value is computed as follows.

$$\frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right]$$

where N is the number of datasets, R_j is the average rank for the j-th model, and k is the number of models. In this case, since N and k are big enough (N > 10 and k > 5), the Friedman test value is distributed following a chi-squared distribution with $k-1$ degrees of freedom. The null hypothesis is rejected if the Friedman test value is bigger than the critical chi-squared value for the given significance level α and degrees of freedom. If the null hypothesis is rejected, a post-hoc test is done: The Nemenyi test. The performance of two models differs significantly if the absolute difference between average ranks is bigger than the critical difference (CD). It is computed as

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}}$$

where q_α critical value based on the Studentized range distribution for k algorithms and significance level α (usually found in Nemenyi test tables) [91].

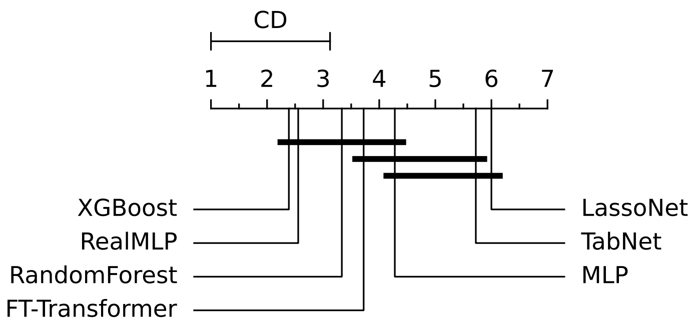


Figure 7.1: Critical Difference Diagram

The diagram in Figure 7.1 is the result of the post-hoc tests done on the ranks from Table 7.1. The models on the left are the best-performing models, and on the right are the worst-performing models. They are placed based on their average rank. The horizontal line at the top represents the critical difference. The thick horizontal lines regroup models with differences being smaller than the critical difference, meaning that the models are not significantly different. In this case, RealMLP, XGBoost, RandomForest, FT-Transformer, and MLP are connected. TabNet and LassoNet are connected but far from the best models.

The results are mostly similar to the benchmark [13]. The common models are XGBoost, RandomForest, and FT-Transformer. The best-performing models' order in the benchmark is XGBoost, RandomForest, and FT-Transformer, which is the same order as in this thesis. However, in the benchmark, the gap between RandomForest and FT-Transformer is bigger than in this thesis. The results for RealMLP in this thesis confirm what Holzmüller et al. [27] stated, which is that this model can compete with tree-based models.

It is also crucial for models to run for a reasonable amount of time. Fast models bring the possibility to do more experiments, such as hyperparameter tuning. Moreover, these kinds of models consume less computational power, and training on bigger datasets is much more possible.

	RandomForest	XGBoost	RealMLP	FT-Transformer	MLP	TabNet	LassoNet
fit_time	95.4	0.327	55.723	102.059	59.892	180.626	125.827

Table 7.2: Fit time of each model in seconds

It has to be noted that the *fit_time* values presented in Table 7.2 are fit time values averaged over all datasets for one iteration of random search. It can be observed from this table that XGBoost is undoubtedly the fastest model by a large margin. The MLP models are the second and third-fastest. The combination of the model analysis and the fit times proves that RealMLP and XGBoost are the two best models of the experiment, as they are both performant and fast. The two slowest models are TabNet and LassoNet, but there is a reason for this. It is because they have to run two functions to be able to train. For LassoNet, the path must be run to select the features that will be used further, depending on the λ parameter. For TabNet, as explained in Section 6, the model uses unsupervised pretraining, followed by supervised training.

7.2 SHAP analysis

In order to better understand how models make their decisions and highlight the differences in how the features are used depending on the model, a SHAP analysis is conducted.

The SHAP analysis is not performed on all datasets used in this thesis, as doing so would be both redundant and useless. Instead, a subset of three datasets is selected based on performance diversity, allowing for a meaningful interpretation of model behavior.

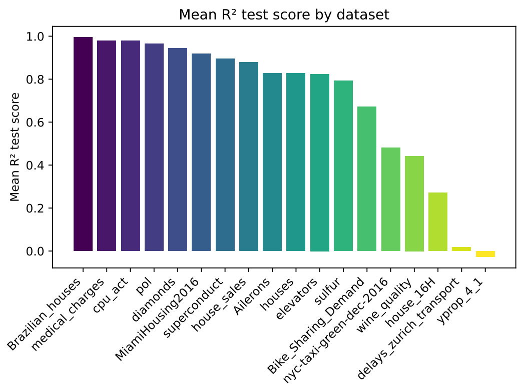


Figure 7.2: Mean best iteration score: for each model, a value of validation R^2 is linked to a random search iteration. Therefore, the best value among iterations is selected for each model and then averaged.

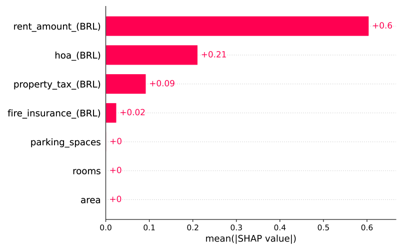
Figure 7.2 represents the mean best iteration R^2-score depending on the dataset. This barplot will be used to select datasets to work with for the SHAP analysis. The corresponding mean R^2 value is in brackets. The three datasets chosen are:

- *Brazilian_houses*: High-performing dataset (nearly 1)
- *nyc-taxi-green-dec-2016*: Medium-performing dataset (0.5)
- *house_16H*: Low-performing dataset (0.2)

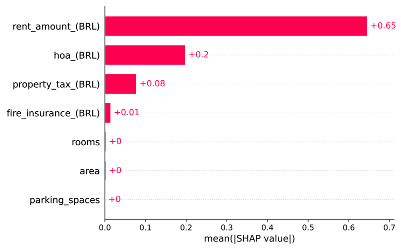
Additionally, for the same reasons as datasets, only a subset of models has to be taken into account in this study. In this case, it will be XGBoost, RealMLP, and FT-Transformer.

SHAP plots help to understand which features are the most important for each model. Global interpretation enables the description of the overall behaviour of a model. SHAP plots were done using the shap package on Python.

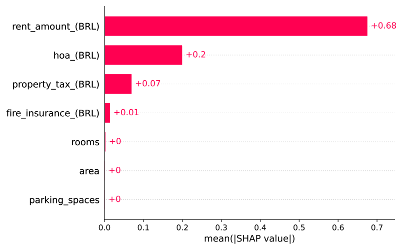
Brazilian_houses



(a) SHAP bar plot for RealMLP (R^2 = 0.996)



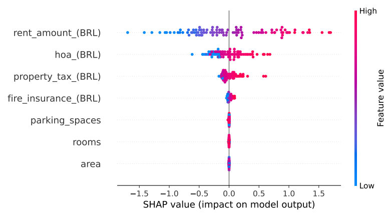
(b) SHAP bar plot for XGBoost (R^2 = 0.994)



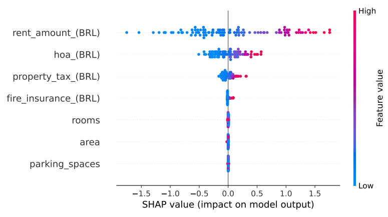
(c) SHAP bar plot for FT-Transformer (R^2 = 0.997)

Figure 7.3: SHAP bar plots for RealMLP, XGBoost, and FT-Transformer for the *Brazilian_houses* dataset

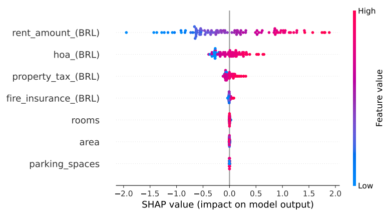
SHAP bar plots, as shown in Figure 7.3, represent the mean absolute SHAP value for each feature of the dataset. This measures the impact (positive or negative) of each feature on the final prediction. The features are ranked from top to bottom by their mean absolute SHAP values for the entire dataset, indicating their relative importance.



(a) SHAP Beeswarm plot for RealMLP



(b) SHAP Beeswarm plot for XGBoost



(c) SHAP Beeswarm plot for FT-Transformer

Figure 7.4: SHAP Beeswarm plots for RealMLP, XGBoost, and FT-Transformer for the *Brazilian_houses* dataset

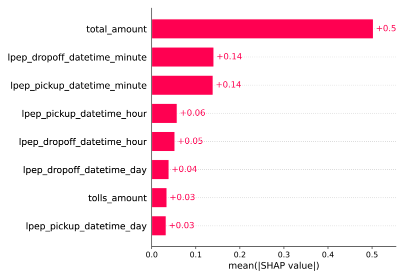
SHAP Beeswarm plots, as shown in Figure 7.4, offer information about the importance of each feature, such as bar plots. Additionally, they offer information about the relationships between the features and the predicted output. A point corresponds to a row of the dataset. If a point is blue, it means that the value of the variable in a row is low, and if it is red, it means that the value is high, as seen on the color bar at the right of each beeswarm plot. This precision is necessary to avoid confusion with SHAP values.

In Figure 7.3, it can be seen that the top 3 features ((*rent_amount_(BRL)*, *hoa_(BRL)*, *property_tax_(BRL)*)) are the same for the 3 models. This indicates that those features have a strong relationship with the target variable. Figure 7.4 confirms this, which means that the top 3 features have the same relationships with the predicted outcome. In this case, the relation is: a high feature value leads to a higher-value prediction.

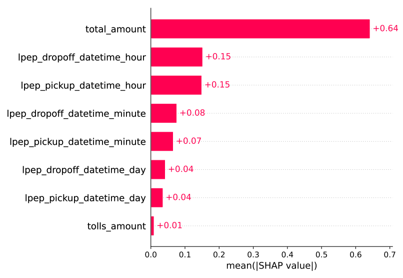
The order for the remaining features is different for each model. It indicates that each model captures different aspects of the data. However, those remaining features have a low impact on the prediction of the output for the three models.

To conclude, for this dataset, it is observed that all the models recognize the main features, and the effect of the remaining features is negligible.

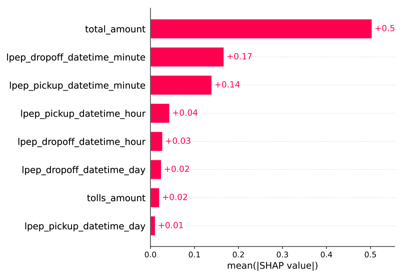
nyc-taxi-green-dec-2016



(a) SHAP bar plot for RealMLP (R^2 = 0.486)

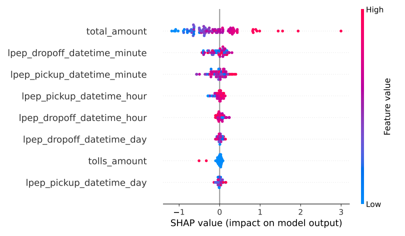


(b) SHAP bar plot for XGBoost (R^2 = 0.513)

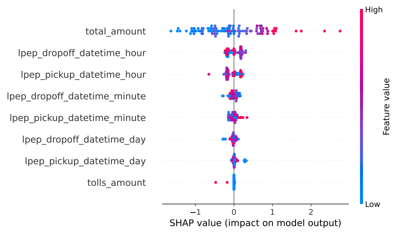


(c) SHAP bar plot for FT-Transformer (R^2 = 0.454)

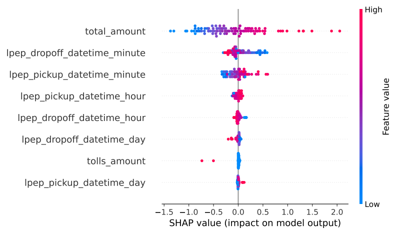
Figure 7.5: SHAP bar plots for RealMLP, XGBoost, and FT-Transformer for the *nyc-taxi-green-dec-2016* dataset



(a) SHAP Beeswarm plot for RealMLP



(b) SHAP Beeswarm plot for XGBoost

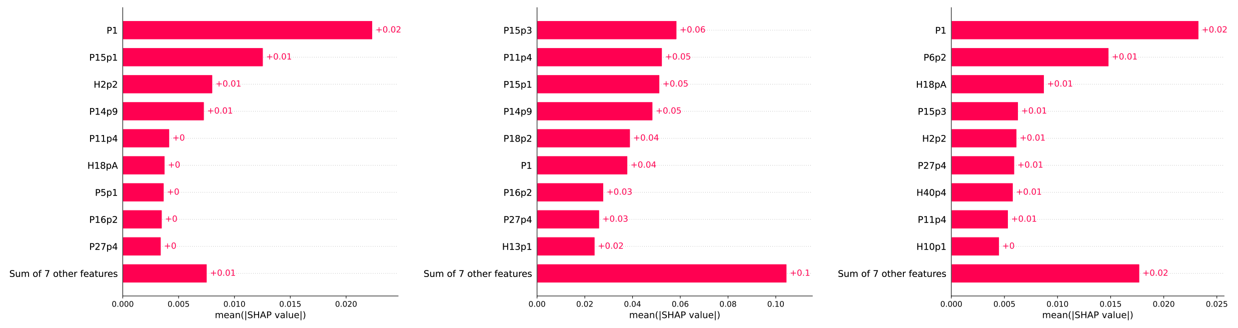


(c) SHAP Beeswarm plot for FT-Transformer

Figure 7.6: SHAP Beeswarm plots for RealMLP, XGBoost, and FT-Transformer for the *nyc-taxi-green-dec-2016* dataset

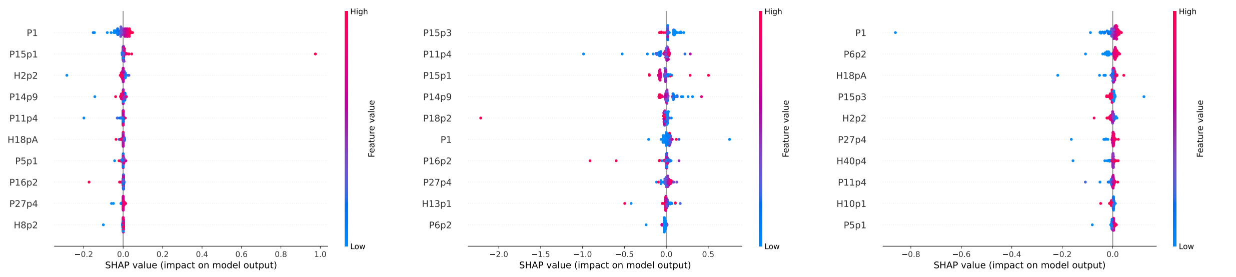
As seen in Figure 7.5, the 3 models agree on the top feature *total_amount*, meaning that this feature has a strong relationship with the target feature. The beeswarm plots confirm that higher values of *total_amount* lead to higher predicted values. RealMLP and FT-Transformer identify *lpep_dropoff_datetime_minute* and *lpep_pickup_datetime_minute* as the second and the third most important features, whereas XGBoost considers *lpep_dropoff_datetime_hour* and *lpep_pickup_datetime_hour* as the second most important feature. Although RealMLP and FT-Transformer agree on the second-best feature, the relationship between this feature and the target is different. Indeed, for FT-Transformer, the relationship is straightforward, meaning that the relationship can easily be seen from the Beeswarm plot. However, the relationship between *lpep_dropoff_datetime_minute* feature and the target is less straightforward, as the beeswarm plot shows a complex group of points along the feature's axis, indicating a more complex interaction. For the other feature *lpep_pickup_datetime_minute*, the relationships are roughly the same for both models. The two features after the top three are the ones not considered in the top 3 for the models. So for XGBoost, it is *lpep_dropoff_datetime_minute* and *lpep_pickup_datetime_minute*, while for RealMLP and FT-Transformer, it is *lpep_dropoff_datetime_hour* and *lpep_pickup_datetime_hour*. After those five features, the 3 models agree that the effect of the remaining features is minor.

house_16H



(a) SHAP bar plot for RealMLP (R^2 = 0.211) (b) SHAP bar plot for XGBoost (R^2 = 0.524) (c) SHAP bar plot for FT-Transformer (R^2 = 0.142)

Figure 7.7: SHAP bar plots for RealMLP, XGBoost, and FT-Transformer for the *house_16H* dataset



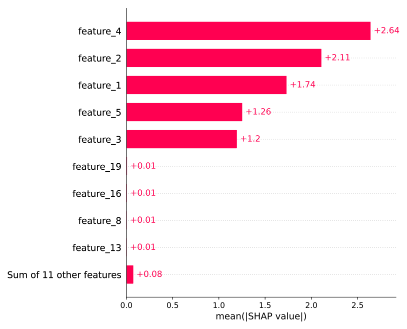
(a) SHAP Beeswarm plot for RealMLP (b) SHAP Beeswarm plot for XGBoost (c) SHAP Beeswarm plot for FT-Transformer

Figure 7.8: SHAP Beeswarm plots for RealMLP, XGBoost, and FT-Transformer for the *house_16H* dataset

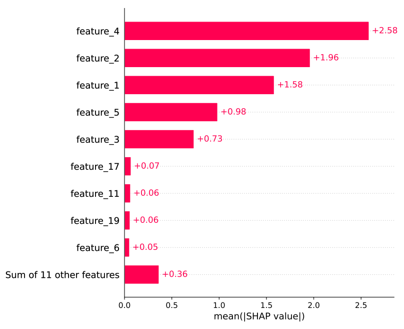
As can be seen from Figure 7.7, contrary to the two other datasets, all the models do not agree on the top feature. RealMLP and Ft-Transformer have the same top-ranked feature, and the relationship between this feature and the target is the same, as it can be seen from Figure 7.8. Moreover, they do not possess the same features as the top-ranked features, indicating that, in this case, the choice of important features is model-dependent. This disagreement shows that the predictive relationships within the dataset are weak, unclear, or complex. The SHAP beeswarm plots in Figure 7.8 reinforce this observation as inconsistent patterns are found in feature importance across models.

Friedman dataset

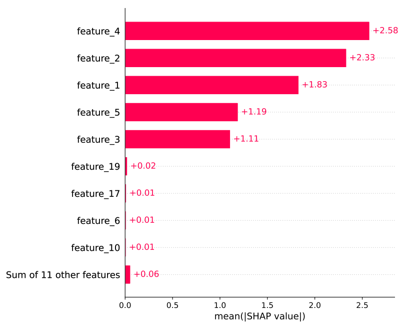
In the previous dataset, the truly relevant features are unknown. To evaluate if the models can identify the real features of a dataset, a test can be made using 5 features and noise features. These are the characteristics of the *friedman* dataset from sklearn. The example below has been made with 20 total features, including five real features and the other 15 being noise.



(a) SHAP bar plot for RealMLP



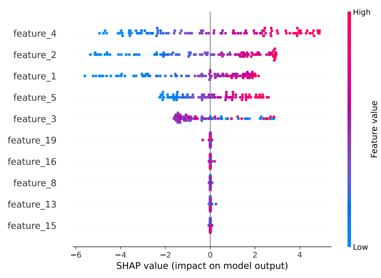
(b) SHAP bar plot for XGBoost



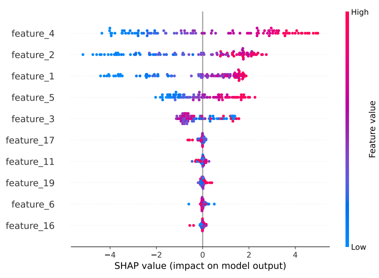
(c) SHAP bar plot for FT-Transformer

Figure 7.9: SHAP bar plots for RealMLP, XGBoost, and FT-Transformer for the *friedman* dataset with 20 features

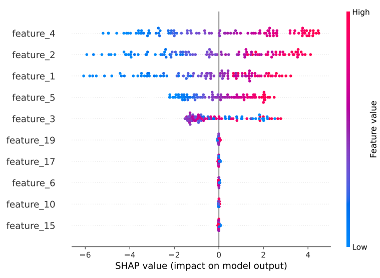
It can be seen from Figure 7.9 that all three models have identified the real features. Moreover, the models have the same feature ranking.



(a) SHAP Beeswarm plot for MLP



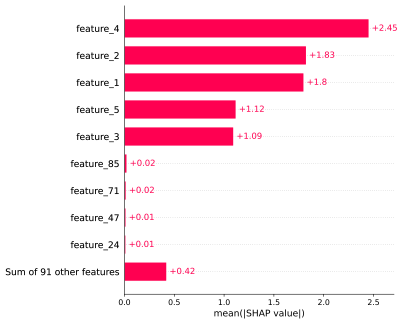
(b) SHAP Beeswarm plot for XGBoost



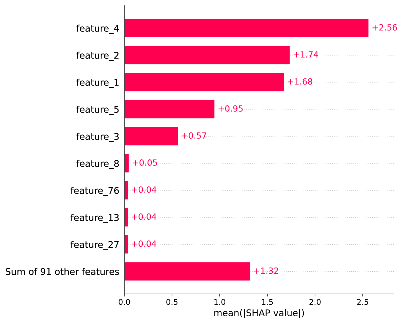
(c) SHAP Beeswarm plot for FT-Transformer

Figure 7.10: SHAP Beeswarm plots for MLP, XGBoost, and FT-Transformer for the *friedman* dataset with 20 features

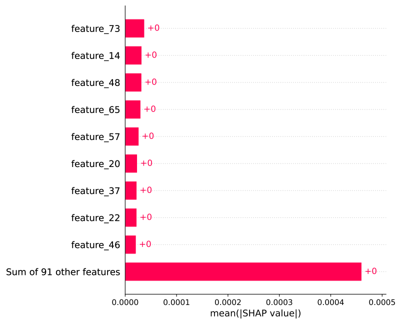
It can be observed from Figure 7.10 that all models captured the same relationship between the real features and the target variable, meaning that these features have a strong relationship with the target.



(a) SHAP bar plot for RealMLP

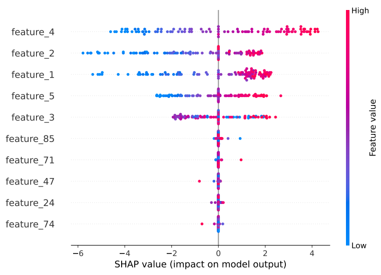


(b) SHAP bar plot for XGBoost

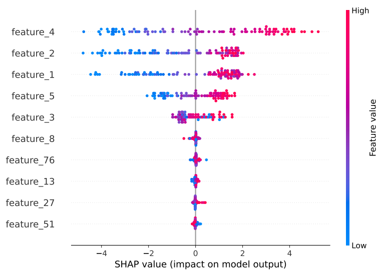


(c) SHAP bar plot for FT-Transformer

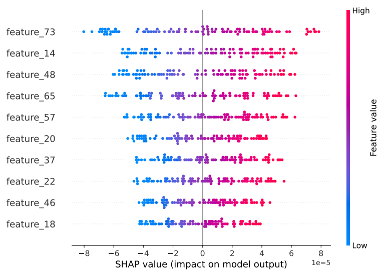
Figure 7.11: SHAP bar plots for RealMLP, XGBoost, and FT-Transformer for the *friedman* dataset with 100 features



(a) SHAP Beeswarm plot for RealMLP



(b) SHAP Beeswarm plot for XGBoost



(c) SHAP Beeswarm plot for FT-Transformer

Figure 7.12: SHAP Beeswarm plots for RealMLP, XGBoost, and FT-Transformer for the *friedman* dataset with 100 features

In Figure 7.11 and 7.12, the number of total features has been set to 100. The reason is to determine if the models can still identify the main features despite the presence of many additional noise features. RealMLP and XGBoost were able to identify the real features and their relationships with the target value. However, FT-Transformer

7.3 LLM generation results

To generate synthetic data, the Generation of Realistic Tabular data (GReaT) Python framework [89] has been used. Similarly to SHAP analysis, the tabular data generation will only be used on carefully selected datasets.

- *elevators*: Its overall performance can be improved
- *Bike_Sharing_Demand*: Its overall performance can be improved
- *sulfur*: It has one of the smallest number of samples across all datasets (10081 samples)
- *wine_quality* : Same reason as *sulfur* (6497 samples)

A straightforward idea to use synthetic data with tabular data could be to add them into the original dataset to see if the performance of a deep learning model can improve with more data. To see if synthetic data has an effect, the number of samples that is generated is twice the number of samples of the original dataset.

Nevertheless, it is not guaranteed that the distribution of synthetic data matches the distribution of original data. Mixing the entire dataset with the original data will decrease the model's performance, as it introduces noise. A possible solution is to filter the synthetic data based on the original data to have better quality data, and then merge the filtered data with the original dataset.

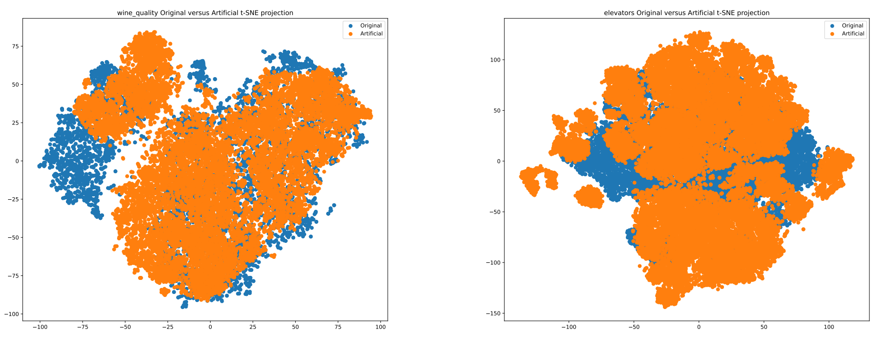
Dataset	MLP (original)	MLP (augmented)	MLP (filtered 90%)
ele	0.899	-48.686	0.855
Bik	0.672	0.686	0.684
wine	0.355	0.366	0.336
sul	0.782	0.559	0.778

Table 7.3: Artificial data evaluation using R^2 as metric

MLP (original) is the MLP trained on the original dataset, MLP (augmented) is the one trained on the original + synthetic dataset, and MLP (filtered) is the same as the previous except that the synthetic data is filtered. The percentages indicate the amount of synthetic data that is kept. For filtering the data, the nearest neighbours method will be used. The algorithm has been fit on the original data to enable the filtering of the artificial data.

It can be seen from Table 7.3 that adding synthetic data to the original dataset can be beneficial or detrimental depending on the dataset. For the *Bike_Sharing_Demand* and *wine_quality*, the addition is beneficial since the MLP performance improves, contrary to the *elevators* and

To explain the performance of the augmented datasets, notably the strange result of the *elevators* dataset when all the synthetic dataset is stacked with the original one. There is a need to visualize the datasets; however, the datasets include many features. The issue is solved using t-distributed Stochastic Neighbor Embedding (t-SNE), which reduces the dimensionality of the data to facilitate visualization.



(a) *wine_quality* t-SNE projection on original data and synthetic data (b) *elevators* t-SNE projection on original data and synthetic data

Figure 7.13: Visualization of *wine_quality* and *elevors* datasets using dimension reduction method t-SNE

Figure 7.13b explains the catastrophic augmented dataset performance. Indeed, it can be seen from this figure that synthetic data contains some outliers, making the result of stacking the original dataset and the artificial bad. It also confirms why filtering at 90% enables having more reasonable results.

It can be seen from Figure 7.13a that synthetic data mostly agrees with the original dataset, explaining that adding it fully improves the performance of the MLP on the dataset.

Chapter 8

Conclusion

This thesis aimed to determine whether deep learning models could outperform traditional algorithms for tabular data. One important contribution of this thesis is that tabular data can be well-modeled by deep learning methods, not just by classical machine learning algorithms.

The results of this thesis prove that investing time in deep learning models on tabular data can be worthwhile because two deep learning models were close to the traditional methods, but not better. Secondly, interpretability techniques, such as SHAP, help for understanding why a model performs better than another based on feature importance. Then, the computational cost of training deep learning models should be taken into account. Indeed, it can take a long time to train and requires significantly more resources. As a result, it is better to favor simpler models when limited by resources and time. Finally, adding synthetic data to augment an original dataset can improve the performance of a deep learning model, but it improves slightly.

The models have been trained with only 50 random search iterations. Therefore, the findings cannot be generalized. As a result, it is suggested that future research train models on a larger number of random search iterations. This thesis is also limited by a small number of datasets. Despite this limitation, it makes an important contribution, as the selected datasets provide some diversity. While SHAP was used to interpret models, such methods give approximations rather than a complete understanding of model decision-making. Future research should explore a wider range of datasets as well as additional techniques for interpretability. Moreover, the datasets used in this thesis contained only numerical values. For future research, categorical values should be included to explore tabular data in all of its forms. Concerning tabular data generation, this thesis has been limited to the method used to perform this; however, there is great potential. For future research, tabular data generation will have to be further investigated, and other methods should be used.

Overall, this thesis highlights the potential of deep learning on tabular data and serves as a basis for further exploration in both performance and interpretability.

Bibliography

[1] Boris Van Breugel and Mihaela Van Der Schaar. Position: Why tabular foundation models should be a research priority. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 48976–48993. PMLR, 21–27 Jul 2024.

[2] Ismael Lemhadri, Feng Ruan, Louis Abraham, and Robert Tibshirani. Lassonet: A neural network with feature sparsity. *Journal of Machine Learning Research*, 22(127):1–29, 2021.

[3] Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

[4] Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35:24991–25004, 2022.

[5] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 35(6):7499–7519, June 2024.

[6] 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.

[7] Meherwar Fatima and Maruf Pasha. Survey of machine learning algorithms for disease diagnostic. *Journal of Intelligent Learning Systems and Applications*, 09:1–16, 01 2017.

[8] Francesco Cartella, Orlando Anunciação, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. Adversarial attacks for tabular data: Application to fraud detection and imbalanced data. 01 2021.

[9] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM Computing Surveys*, 54(2):1–38, March 2021.

[10] A Yarkın Yıldız and Asli Kalayci. Gradient boosting decision trees on medical diagnosis over tabular data. *arXiv preprint arXiv:2410.03705*, 2024.

[11] 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*, KDD '16, page 785–794. ACM, August 2016.

[12] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

[13] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems*, 35:507–520, 2022.

[14] Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

[15] Vasyl Harasymiv. Lessons from 2 million machine learning models on kaggle, 2015.

[16] Krzysztof Grabczewski and Norbert Jankowski. Feature selection with decision tree criterion. In *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 6–pp. IEEE, 2005.

[17] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[18] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[19] Zihang Dai, Zhilin Yang, Fan Yang, William W Cohen, and Russ R Salakhutdinov. Good semi-supervised learning that requires a bad gan. *Advances in neural information processing systems*, 30, 2017.

[20] Vadim Borisov, Klaus Broelemann, Enkelejda Kasneci, and Gjergji Kasneci. Deeptlf: robust deep neural networks for heterogeneous tabular data. *International Journal of Data Science and Analytics*, 16, 08 2022.

[21] End-to-End Learning of Decision Trees and Forests. *International Journal of Computer Vision (IJCV)*, 128:997–1011, 2020.

[22] Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, Mar 2021.

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

[24] 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*, 17(1):45–77, 2019.

[25] Xavier Bouthillier, Pierre Delaunay, Mirko Bronzi, Assya Trofimov, Brennan Nichyporuk, Justin Szeto, Naz Sepah, Edward Raff, Kanika Madan, Vikram Voleti, Samira Ebrahimi Kahou, Vincent Michalski, Dmitriy Serdyuk, Tal Arbel, Chris Pal, Gaël Varoquaux, and Pascal Vincent. Accounting for variance in machine learning benchmarks, 2021.

[26] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in neural information processing systems*, 34:18932–18943, 2021.

[27] David Holzmüller, Léo Grinsztajn, and Ingo Steinwart. Better by default: Strong pre-tuned mlps and boosted trees on tabular data. *Advances in Neural Information Processing Systems*, 37:26577–26658, 2024.

[28] Nathan Lay, Adam P Harrison, Sharon Schreiber, Gitesh Dawer, and Adrian Barbu. Random hinge forest for differentiable learning. *arXiv preprint arXiv:1802.03882*, 2018.

[29] Yingshi Chen. Attention augmented differentiable forest for tabular data. *arXiv preprint arXiv:2010.02921*, 2020.

[30] Edward Choi, Siddharth Biswal, Bradley Malin, Jon Duke, Walter F Stewart, and Jimeng Sun. Generating multi-label discrete patient records using generative adversarial networks. In *Machine learning for healthcare conference*, pages 286–305. PMLR, 2017.

[31] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent abilities of large language models, 2022.

[32] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, Zihao Wu, Lin Zhao, Dajiang Zhu, Xiang Li, Ning Qiang, Dingang Shen, Tianming Liu, and Bao Ge. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology*, 1(2):100017, September 2023.

[33] Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[34] Dang Nguyen, Sunil Gupta, Kien Do, Thin Nguyen, and Svetha Venkatesh. Generating realistic tabular data with large language models. In *2024 IEEE International Conference on Data Mining (ICDM)*, pages 330–339. IEEE, 2024.

[35] Yishuo Zhang, Nayyar A Zaidi, Jiahui Zhou, and Gang Li. Ganblr: a tabular data generation model. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 181–190. IEEE, 2021.

[36] Patricia A Apellániz, Juan Parras, and Santiago Zazo. An improved tabular data generator with vae-gmm integration. In *2024 32nd European Signal Processing Conference (EUSIPCO)*, pages 1886–1890. IEEE, 2024.

[37] Akim Kotelnikov, Dmitry Baranchuk, Ivan Rubachev, and Artem Babenko. Tabddpm: Modelling tabular data with diffusion models. In *International conference on machine learning*, pages 17564–17579. PMLR, 2023.

[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. *Advances in neural information processing systems*, 30, 2017.

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

[40] 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. *arXiv preprint arXiv:2010.11929*, 2020.

[41] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[42] 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.

[43] Amit Damri, Mark Last, and Niv Cohen. Towards efficient image-based representation of tabular data. *Neural Computing and Applications*, 36(2):1023–1043, 2024.

[44] Boyu Lyu and Anamul Haque. Deep learning based tumor type classification using gene expression data. In *Proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics*, pages 89–96, 2018.

[45] Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith Boroevich, and Tatsuhiko Tsunoda. Deepinsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific Reports*, 9, 08 2019.

[46] Pascal Vincent, Hugo Larochelle, Y. Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. pages 1096–1103, 01 2008.

[47] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544, 2016.

[48] Jinsung Yoon, Yao Zhang, James Jordon, and Mihaela van der Schaar. Vime: Extending the success of self- and semi-supervised learning to tabular domain. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 11033–11043. Curran Associates, Inc., 2020.

[49] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(19):625–660, 2010.

[50] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery.

[51] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMIR, 2017.

[52] Anupam Datta, Shayak Sen, and Yair Zick. Algorithmic transparency via quantitative input influence: Theory and experiments with learning systems. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 598–617, 2016.

[53] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[54] Seungeun Lee, Il-Youp Kwak, Kihwan Lee, Subin Bae, Sangjun Lee, Seulbin Lee, and Seungsang Oh. Table2image: Interpretable tabular data classification with realistic image transformations. *arXiv preprint arXiv:2412.06265*, 2024.

[55] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. Large language models(llms) on tabular data: Prediction, generation, and understanding – a survey, 2024.

[56] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[57] Christoph Molnar. *Interpretable Machine Learning*. 3 edition, 2025.

[58] Chiyu Ma, Jon Donnelly, Wenjun Liu, Soroush Vosoughi, Cynthia Rudin, and Chaofan Chen. Interpretable image classification with adaptive prototype-based vision transformers. *Advances in Neural Information Processing Systems*, 37:41447–41493, 2024.

[59] Zebin Yang, Agus Sudjianto, Xiaoming Li, and Aijun Zhang. Inherently interpretable tree ensemble learning. *arXiv preprint arXiv:2410.19098*, 2024.

[60] Peter Bühlmann and Torsten Hothorn. Rejoinder: Boosting algorithms: Regularization, prediction and model fitting. *Statistical Science*, 22(4):516–522, 2007.

[61] Iqbal Sarker. Deep learning: A comprehensive overview on techniques, taxonomy, applications and research directions. *SN Computer Science*, 2, 08 2021.

[62] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022.

[63] Katarzyna Rojewska. What are neural networks and what are their applications? `https://www.qtravel.ai/blog/what-are-neural-networks-and-what-are-their-applications/`, 2023.

[64] Osval Montesinos-López, Abelardo Montesinos, and Jose Crossa. *Fundamentals of Artificial Neural Networks and Deep Learning*, pages 379–425. 01 2022.

[65] Keiron O'shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[66] Derya Soydaner. Attention mechanism in neural networks: where it comes and where it goes. *Neural Computing and Applications*, 34(16):13371–13385, May 2022.

[67] Saidul Islam, Hanae Elmekki, Ahmed Elsebai, Jamal Bentahar, Nagat Drawel, Gaith Rjoub, and Witold Pedrycz. A comprehensive survey on applications of transformers for deep learning tasks. *Expert Systems with Applications*, 241:122666, 2024.

[68] Marek Śmieja, Łukasz Struski, Jacek Tabor, Bartosz Zieliński, and Przemysław Spurek. Processing of missing data by neural networks. *Advances in neural information processing systems*, 31, 2018.

[69] Han-Jia Ye, Si-Yang Liu, Hao-Run Cai, Qi-Le Zhou, and De-Chuan Zhan. A closer look at deep learning methods on tabular datasets. *arXiv preprint arXiv:2407.00956*, 2024.

[70] Weijieying Ren, Tianxiang Zhao, Yuqing Huang, and Vasant Honavar. Deep learning within tabular data: Foundations, challenges, advances and future directions. *arXiv preprint arXiv:2501.03540*, 2025.

[71] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial intelligence and statistics*, pages 192–204. PMLR, 2015.

[72] Andrew Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. *Proceedings of the Twenty-First International Conference on Machine Learning*, 09 2004.

[73] Marc Schmitt. Deep learning vs. gradient boosting: Benchmarking state-of-the-art machine learning algorithms for credit scoring. *arXiv preprint arXiv:2205.10535*, 2022.

[74] Gaël Varoquaux. Léo Grinsztajn, Edouard Oyallon. Datasets for benchmark. https://huggingface.co/datasets/inria-soda/tabular-benchmark, 2021.

[75] Sana Fatima, Ayan Hussain, Sohaib Amir, Syed Haseeb Ahmed, and Syed Aslam. Xgboost and random forest algorithms: An in depth analysis. *Pakistan Journal of Scientific Research*, 3:26–31, 10 2023.

[76] Mustafa BÜYÜKKEÇECİ and Mehmet Okur. A comprehensive review of feature selection and feature selection stability in machine learning. *GAZI UNIVERSITY JOURNAL OF SCIENCE*, 36, 09 2022.

[77] 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.

[78] David Holzmüller, Viktor Zaverkin, Johannes KÃstner, and Ingo Steinwart. A framework and benchmark for deep batch active learning for regression. *Journal of Machine Learning Research*, 24(164):1–81, 2023.

[79] Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*, 2023.

[80] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks (2016). *arXiv preprint arXiv:1608.06993*, 7, 2018.

[81] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. *Advances in neural information processing systems*, 20, 2007.

[82] Taisuke Yasuda, MohammadHossein Bateni, Lin Chen, Matthew Fahrbach, Gang Fu, and Vahab Mirrokni. Sequential attention for feature selection. *arXiv preprint arXiv:2209.14881*, 2022.

[83] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[84] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. Multi-head attention: Collaborate instead of concatenate. *arXiv preprint arXiv:2006.16362*, 2020.

[85] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[86] Avijit Thawani, Jay Pujara, Pedro A Szekely, and Filip Ilievski. Representing numbers in nlp: a survey and a vision. *arXiv preprint arXiv:2103.13136*, 2021.

[87] Georgios P Spithourakis and Sebastian Riedel. Numeracy for language models: Evaluating and improving their ability to predict numbers. *arXiv preprint arXiv:1805.08154*, 2018.

[88] Aspen K Hopkins, Alex Renda, and Michael Carbin. Can LLMs generate random numbers? evaluating LLM sampling in controlled domains. In *ICML 2023 Workshop: Sampling and Optimization in Discrete Space*, 2023.

[89] Vadim Borisov, Kathrin Seßler, Tobias Leemann, Martin Pawelczyk, and Gjergji Kasneci. Language models are realistic tabular data generators. *arXiv preprint arXiv:2210.06280*, 2022.

[90] Jilei Yang. Fast treeshap: Accelerating shap value computation for trees, 2022.

[91] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, December 2006.

Appendix

The hyperparameter ranges of all models except TabNet are based on the Grinsztajn benchmark [13].

Parameter	Range
Number of layers	UniformInt [1, 9]
Layer size	UniformInt [16, 1025]
Batch size	[256, 512, 1024]
Learning rate	LogUniform [1e-4, 1e-2]

Table 1: MLP hyperparameters space

Parameter	Range
Number of layers	UniformInt [1, 9]
Layer size	UniformInt [16, 1025]
Batch size	[256, 512, 1024]
Learning rate	LogUniform [1e-4, 1e-2]
Lambda	UniformInt[10, 600, 10]
Hierarchy coefficient M	[10, 50, 100]

Table 2: LassoNet hyperparameters space

Parameter	Range
Max depth	UniformInt [1, 12]
Num_estimators	UniformInt[10, 2000]
Min child weight	LogUniformInt [1, 100]
Learning rate	LogUniform[1e-5, 0.7]
Gamma	LogUniform[1e-8, 7]
Lambda	LogUniform[1, 4]
Alpha	LogUniform[1e-8, 1e2]

Table 3: XGBoost hyperparameters space

Parameter	Range
Max depth	[None, 2, 3, 4] with probability [0.7, 0.1, 0.1, 0.1]
Criterion	['squared_error', 'absolute_error']
Min child weight	LogUniformInt [1, 100]
Max features	[sqrt, sqrt, log2, None, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
Min samples split	[2, 3] with probability [0.95, 0.05]

Table 4: RandomForest hyperparameters space

Parameter	Range
Number of layers	UniformInt [1, 9]
Layer size	UniformInt [16, 1025]
Batch size	[256, 512, 1024]
Learning rate	LogUniform [1e-4, 1e-2]

Table 5: RealMLP hyperparameters space

Parameter	Range
Number of layers	UniformInt [1, 6]
Residual dropout	[0, 0.5]
Attention dropout	[0, 0.1, 0.2, 0.3, 0.4, 0.5]
Learning rate	LogUniform [1e-5, 1e-2]
Batch size	[64, 256]

Table 6: FT-Transformer hyperparameters space

For TabNet, the hyperparameter space is based on `https://dreamquark-ai.github.io/tabnet/generated_docs/README.html`

Parameter	Range
num_steps	UniformInt [3, 8]
n_d	[8, 16, 24]
n_a	[8, 16, 24]
Gamma	[1.0, 1.2, 1.5, 2.0]
Learning rate	LogUniform [1e-4, 1e-2]
Lambda_sparse	[0, 0.0001, 0.001, 0.01, 0.1]
Learning rate	[0.005, 0.01, 0.02, 0.025]
Batch size	[64, 256]

Table 7: TabNet hyperparameters space