Deep Learning vs Classical ML on Tabular Datasets.

# Introduction

## 1.1 The importance of tabular data

Heterogeneous tabular data remain the most widely utilized form of data in many industries, even with the latest advancements in large language models and generative artificial intelligence technologies. These advancements have notably revolutionized fields like computer vision and natural language processing, but their impact on handling heterogeneous tabular data deserves deeper exploration. In this context, every business finds itself with the possibility and often the necessity of storing and managing information in a tabular fashion.

Tabular data can be formally defined as a type of data organized in rows and columns, where each row represents an individual observation or record, and each column represents a specific attribute or variable. This structured format offers an intuitive and efficient way to store and analyze data. The data in each column is usually of the same data type, such as numerical or categorical, and each cell of the table contains a single value.

Industries such as finance, healthcare, retail, and many others heavily rely on tabular data to store and analyze crucial information. In finance, for instance, transaction records and investment portfolios are commonly stored in tabular formats for easy tracking and analysis. Similarly, healthcare institutions utilize tabular data to manage patient records, treatment histories, and medical outcomes. This prevalence stems from the ease of organizing and analyzing large sets of data in a structured manner, facilitating informed decision-making.

Despite the emergence of cutting-edge data technologies, the challenges tied to managing heterogeneous tabular data persist. Data inconsistency, missing values, and the need for integrating data from diverse sources are persistent issues. Ensuring data quality, governance, and security becomes paramount, especially when sensitive or confidential information is involved. These challenges underscore the importance of effective data management strategies in maximizing the value derived from heterogeneous tabular data.

The evolution of data technologies is noteworthy in this context. While advancements in large language models and generative AI technologies have significantly influenced computer vision and natural language processing domains, their application to heterogeneous tabular data is an avenue ripe for exploration. Just as these technologies can interpret images and texts, they hold potential for deriving

insights from tabular data, automating tasks, and enhancing decision-making processes.
Considering the growth of data across industries, the importance of tabular data is expected to further magnify. The ability to effectively leverage and harness the value within tabular data relies on a multitude of factors. Data quality, integration, governance, analysis, visualization, and security collectively shape the reliability and actionability of insights and technical solutions derived from the data. Once these foundational aspects are addressed, the pivotal decision arises: which statistical models or techniques should be employed to address specific business needs.

## 1.2 Predictive Tasks on Tabular Data

Often we are interested in solving a business problem which involves the goal of predicting an aleatory phenomenon, such as customer churn, or risk of credit default. In the realm of data science we aggregate such problems in three main categories which are easy to comprehend and formally define.

**Classification Problems**

In classification problems, the goal is to categorize instances into predefined classes or labels based on their features. This is particularly useful for scenarios like customer churn prediction, fraud detection, sentiment analysis, and image classification. The outcome is a decision boundary that separates different classes in the feature space. Common algorithms for classification include logistic regression, decision trees, random forests, support vector machines, and neural networks. Binary classification involves prediction of a binary variable which can only take one of two values, whereas multiclass classification involves a target variable with more than two possible distinct values

**Regression Problems**

Regression problems involve predicting a continuous numerical value based on input features. For example, predicting house prices, stock prices, or demand forecasting fall into this category. In these cases, the goal is to establish a relationship between input features and the target value. Algorithms like linear regression, polynomial regression, decision trees, random forests, and various types of neural networks are commonly used for regression tasks.

**Clustering Problems**

Clustering problems aim to group similar instances together in an unsupervised manner. This is useful for tasks such as customer segmentation, anomaly detection, and recommendation systems. Unlike classification or regression, clustering doesn't involve predefined labels; instead, the algorithm discovers the inherent structure in the data. K-means clustering, hierarchical clustering, DBSCAN, and Gaussian mixture models are popular techniques for clustering.

Each of these categories presents its own set of challenges and requires specific techniques for analysis and modeling. Before choosing a statistical model or technique, it's important to consider factors like the nature of the data, the problem's context, available computational resources, and the interpretability of the chosen method.
For the scope of this analysis on deep learning techniques on tabular data we will be restricting our exploration to binary classification, multiclass classification and regression problems.

## 1.3 Contributions

Currently there are still a number of open, unexplored or partially explored research questions in the field of benchmarking machine learning architectures on tabular data. Authors in the main research papers on this topic, which we will cite and summarize in the next sections, all leave some hints on some of the areas that could be worth exploring. In this research I attempt to explore more in depth a few open questions and contribute towards existing research by:

1. Contributing on benchmarking the algorithms in the small dataset regime (<5k Observations) and medium-large dataset regime (100-500k observations).

2. Exploring the performance on highly unbalanced binary classification tasks.

3. Focusing on a more complete set of evaluation metrics, using the most adequate metrics to evaluate performance of the algorithms. Most research papers will only use a couple of aggregate metrics like accuracy or log-loss regardless of the dataset intrinsic characteristics. In this study the evaluation will consider a much wider range of evaluation metrics trying to focus on the ones more relevant to the task at hand.

4. Providing a framework to easily test and perform hyperparameter optimizations on new models and datasets.

5. Putting to the test a number of Deep Learning architectures that have not yet been benchmarked against each other, including:
    a. ResNet + IGTD
    b. Category Embedding
    c. AutoInt
    d. GATE
    e. GANDALF

# 2 Deep Learning vs Classical Machine Learning on Tabular Data

## 2.1 Machine Learning Techniques for Tabular Data

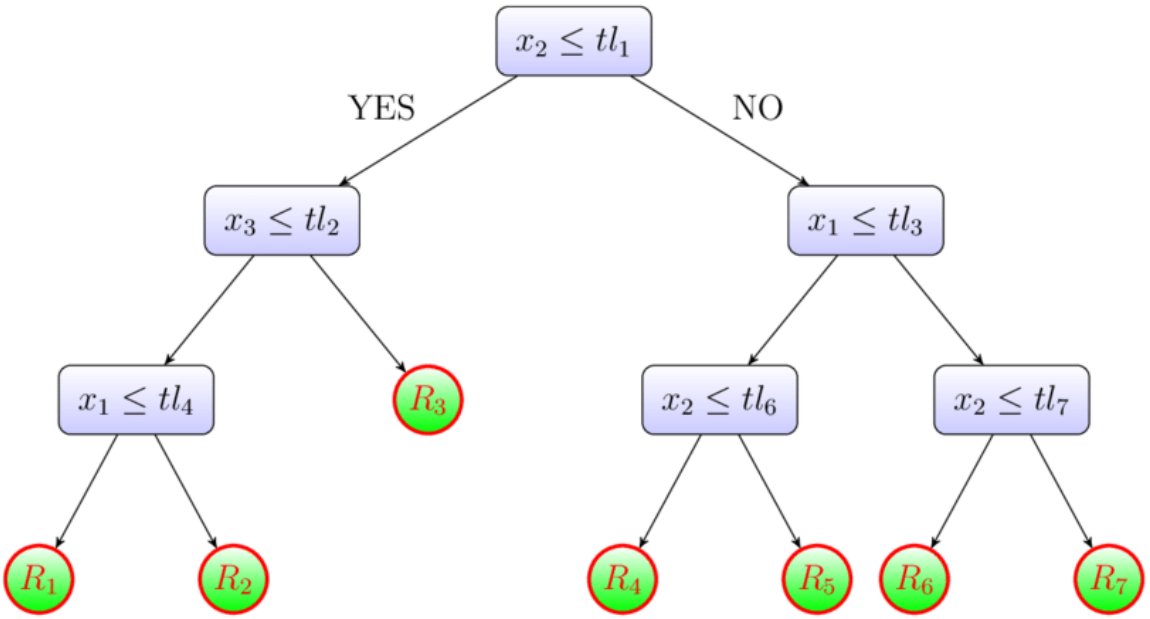
There are a number of machine learning techniques that can extract patterns and insights from large, complex datasets.

### Decision Trees

Decision trees are a simple yet powerful model architecture that can be used to classify data based on a series of binary decisions. These models provide an intuitive framework for analyzing and predicting data by recursively partitioning it into distinct subsets based on sequential binary decisions.
Unraveling the earliest roots of decision trees presents a non trivial challenge beyond this study's scope. Nonetheless, we can track a few key milestones which were pivotal in the development of decision tree architectures used today.

1963: The AID project by Morgan and Sonquist [1] introduces the first decision tree regression. Its impurity measure and recursive data splitting lay foundational groundwork.
1966: Hunt's publication at the Poznań University of Technology [2] pioneers the decision tree model's concept. This application in human learning models underscores its significance in psychology and algorithm development.
1972: Messenger and Mandell's THAID [3] project pioneers classification trees, optimizing mode category sums through data division.
1984: Official publication of the CART [4] decision tree software triggers an algorithmic revolution. CART attains prominence and remains pivotal in decision tree analytics.
1986: Quinlan's innovation [5] introduces multi-answer decision trees. ID3's gain ratio impurity criterion births C4.5, offering comprehensive questioning and node propositions.
Decision tree evolution spans decades of contributions and advancements, yielding pivotal algorithms like CART and C4.5, shaping the landscape of data analytics.

At the core of a decision tree lies its hierarchical structure, composed of nodes that serve distinct purposes. Internal nodes represent decision points, where data is split into subsets by evaluating a specific feature's value. In contrast, leaf nodes, often referred to as terminal nodes, signify the ultimate outcomes of the decision process. These leaf nodes assign class labels in classification tasks or predicted values in regression scenarios.




A decision tree consists of several essential components, here we will briefly summarize the ones used in the CART implementation.

**Root Node**

The root node is the starting point of the decision tree, where input observations are initially passed. The entire data set is considered as the root initially, and then methods are utilized to break or divide the root using decision nodes into subtrees.

**Decision Nodes**

Decision nodes are internal nodes of the tree where decisions or splits are made based on specific feature conditions.

**Splitting**

Splitting refers to the process of dividing a parent node into multiple child nodes based on specific feature conditions. It is how the tree hierarchically structures its decisions. There are a number of techniques for decision tree splitting, the main ones are Gini Impurity, Information Gain and Chi-Square.
Impurity Measures

CART algorithm typically uses Gini Impurity for classification and Mean Squared Error (MSE) for regression as impurity measures. These measures are able to quantify the quality of a split and enable the decision tree algorithm to choose the best splits given the training data.
The Gini Impurity, denoted as $G$, quantifies the probability of incorrectly classifying a randomly chosen element in a dataset if it were randomly labeled according to the class distribution in that dataset. Mathematically, it is calculated as:

$$G = \sum_{i=1}^{C} p(i) \cdot (1 - p(i))$$

where:

- $C$represents the number of classes in the dataset.
- $p(i)$ is the probability of randomly selecting an element of class

A low Gini Impurity indicates that most of the elements in the node belong to a single class (high purity), while a high Gini Impurity suggests that the elements are distributed across multiple classes (low purity). Therefore the goal here is to find splits that minimize the impurity, an impurity of 0 would mean the split perfectly divides the classes.
In decision tree regression, the impurity measure used is MSE which measures the average squared difference between the predicted values and the actual target values for the data points within a node.

**Leaf Node (Terminal Node)**

A leaf node, also known as a terminal node, does not split further and represents the final output or classification. Each leaf node corresponds to a specific category or value.

**Branch (Sub-tree)**

A subsection of a decision tree that includes nodes, splits, and leaf nodes is called a branch or sub-tree. It represents a path of decisions leading to a particular outcome.

**Pruning**

Pruning involves removing nodes (branches) from the tree to improve its generalization performance on unseen data, simplifying the model and reducing complexity. in order to perform pruning the algorithm typically assigns a validation set and bases the pruning strategy on its performance. Pruning in CART is based on a technique called "Cost-Complexity Pruning" or simply "Minimal Cost Complexity Pruning." The goal of pruning is to find a balance between the tree's complexity (number of nodes) and its ability to fit the data. A complexity parameter, denoted as $\alpha$ is introduced to control the trade-off between tree complexity and accuracy.

## Ensemble Methods

Decision trees, while potent in their own right, often play a pivotal role in ensemble methods.

An ensemble method is a powerful technique in machine learning that combines multiple individual models, often of different types or trained on different subsets of the data, to create a stronger and more accurate overall prediction. The core idea behind ensemble methods is to leverage the diversity of multiple models to collectively make better predictions than any single model could achieve on its own. Formally, an ensemble method can be defined as follows:

Given a training dataset $(D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\})$ where each $(x_i)$ is an input feature vector and $(y_i)$ is the corresponding target or label, an ensemble method combines $(M)$ individual models $(h_1, h_2, ..., h_M)$ to create a final prediction function $(H)$ that combines the outputs of the individual models:

$$[H(x) = \text{Combination}(h_1(x), h_2(x), \ldots, h_M(x))]$$

The process of combining the outputs of individual models can vary depending on the ensemble method. Common techniques include:

1. Voting: In classification tasks, ensemble methods often use majority voting to make a final prediction. Each individual model predicts a class, and the class that receives the most votes among the ensemble members is chosen as the final prediction.

2. Averaging In regression tasks, ensemble methods might average the predictions of individual models to obtain a more stable and accurate prediction.

3. Weighted Combination: Some ensemble methods assign different weights to individual models based on their performance on the training data. The final prediction is then a weighted combination of the individual model predictions.

4. Stacking: Stacking involves training a meta-model that learns to combine the predictions of individual models based on their performance and the input data. This meta-model effectively learns a higher-level strategy for combining the strengths of the individual models.

Ensemble methods, such as Random Forests, Gradient Boosting, and AdaBoost, have proven to be highly effective in improving predictive performance and reducing overfitting. By aggregating the knowledge of multiple models, ensemble methods harness the diversity and expertise of each constituent model, leading to improved generalization and robustness on a wide range of tasks.

**Random Forest**

Random Forest is an ensemble method with a number of key features which allows it to be one of the best performing and long-lasting machine learning methods, often performing on par with more complex ensemble methods.

The inception of Random Forest can be attributed to Breiman's work in 2001 [6], wherein he introduced this ensemble learning method as a way to enhance the predictive accuracy of decision trees. By combining the predictions of multiple decision trees, Random Forest addressed the overfitting tendencies of individual trees and offered improved generalization. Its effectiveness was validated through applications in various domains, including bioinformatics and image analysis. Random Forest is built upon a number of key components:

### Bagging

One of the fundamental techniques used in Random Forest is Bootstrap Aggregating or bagging. Bagging involves creating multiple subsets (or bags) of the original training data through a process called bootstrapping. In bootstrapping, data points are randomly sampled with replacement from the training data, resulting in each subset having a unique composition of data points.

### Random Feature Subsampling

Another crucial aspect of Random Forest is the random subsampling of features. When constructing each individual decision tree, Random Forest does not consider all available features at each node. Instead, it selects a random subset of features. This feature subsampling introduces diversity among the trees and prevents the model from becoming overly biased towards the most influential features.

### Independent Decision Trees

For each bootstrapped dataset and the associated random feature subset, an individual decision tree is constructed. These trees are trained independently of each other. Because of the randomness introduced during both data sampling and feature selection, each tree can have a different structure and make distinct predictions.

### Combining Predictions

Once all the individual decision trees are built, Random Forest combines their predictions to make a final prediction. The aggregation strategy depends on whether it's a classification or regression task. In classification tasks, the most common aggregation method is majority voting. Each tree "votes" for a class, and the class with the most votes across all trees becomes the final predicted class. In regression tasks, the individual predictions from each tree are averaged. This averaging process results in a more stable and accurate prediction.

## Gradient Boosting Decision Trees (GBDT)

Gradient Boosting Decision Trees (GBDT) is another ensemble method that combines the predictions of multiple weak learners, typically decision trees, to create a stronger predictive model. GBDT works by sequentially adding trees to the ensemble, with each new tree focusing on correcting the errors made by the previous

ones. GBDT emerged as a great advancement in boosting algorithms. The development of boosting methods can be attributed to the foundational work by Schapire (1990) [7], who introduced the concept of boosting weak learners into a strong learner. The introduction of GBDT was largely influenced by Friedman's work (2001) [8], where he combined boosting with decision trees to create an ensemble method capable of reducing bias and variance, resulting in enhanced predictive performance.

We will briefly summarize the algorithm’s main components:

**Decision Trees**

Each decision tree in GBDT is a weak learner, meaning it's not very accurate on its own, but it contributes to the overall model's strength when combined with others.GBDT builds decision trees one after another, each tree aiming to correct the errors of the previous ensemble. Trees are added sequentially, and each new tree is designed to predict the residuals (differences) between the target values and the predictions made by the existing ensemble. The predictions of all trees are then combined to form the final prediction.

**Gradient Boosting**

Gradient boosting is an ensemble learning technique where multiple weak learners are combined to create a strong learner. GBDT builds decision trees sequentially. It starts with a single decision tree, which is typically a shallow tree or a single node representing the initial prediction, typically this corresponds to the mean of the target values for regression or the majority class for classification.After the first tree is built, GBDT calculates the errors or residuals between the predicted values and the actual target values for each data point.

**Learning Rate**

GBDT includes a learning rate parameter that controls the contribution of each tree to the overall ensemble.

When the learning rate is higher, each tree is attempting to quickly correct the errors made by the existing ensemble. While this can lead to faster convergence and potentially accurate predictions, it also increases the risk of overfitting. Trees may

end up fitting noise or outliers in the training data, leading to a less generalized model.

A lower learning rate has the effect of slowing down the learning process. Each new tree's contribution has less impact on the overall ensemble, and the model takes smaller steps towards the optimal solution. This cautious approach reduces the risk of overfitting. By gradually combining the predictions of many trees, the model builds a more robust representation of the underlying patterns in the data.

The final prediction from the GBDT ensemble is the weighted sum of the predictions from all the trees. As more trees are added, the ensemble iteratively reduces the errors, leading to improved predictions.

**Neural Networks**

Neural networks have their origins in pioneering works from the mid-20th century. McCulloch and Pitts (1943) [9] proposed a foundational mathematical model of artificial neurons, which laid the groundwork for subsequent developments. Building on this, Rosenblatt's perceptron model (1958)[10] emerged as a significant advancement, demonstrating the potential of artificial neural networks in mimicking certain aspects of biological neurons. Additionally, Werbos (1974) [11] introduced the concept of backpropagation for training multi-layer networks, a crucial step towards enabling deep learning architectures.

At the core of a neural network is the neuron, which takes in inputs, applies weights to them, and produces an output. The weighted sum of inputs is then passed through an activation function to introduce non-linearity into the model. The general equation for a neuron's output can be expressed as:

$$\text{Output} = \text{Activation}\left(\sum_{i=1}^{n} \text{Weight}_i \times \text{Input}_i + \text{Bias}\right)$$

Here, $(n)$ represents the number of input connections, $(\text{Weight}_i)$ are the corresponding weights, $(\text{Input}_i)$ are the input values, and the bias accounts for any shift in the activation function.

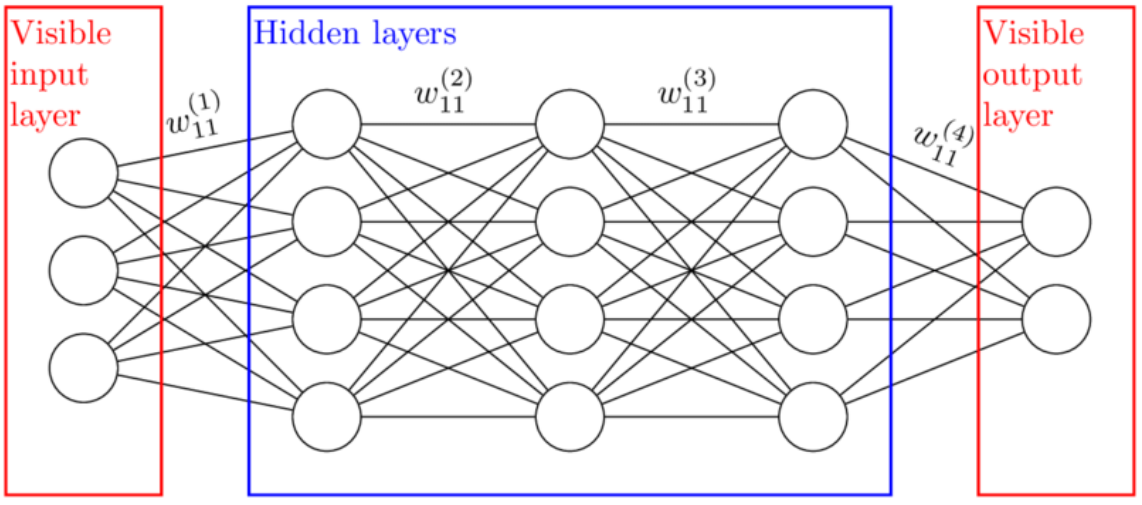
Activation functions introduce non-linearity to the neuron's output. Common activation functions include Sigmoid, Tanh, and ReLU (Rectified Linear Unit). The choice of activation function impacts the network's ability to capture complex patterns and its training dynamics.

Neural networks consist of multiple layers of interconnected neurons. The most common types of layers are the input layer, hidden layers, and output layer. Hidden layers allow the network to learn increasingly complex features and relationships in the data. The architecture of a neural network refers to the arrangement of these layers and the number of neurons in each layer.

During what is known as the feedforward process, input data is passed through the network layer by layer, and the final prediction is obtained. The difference between the predicted output and the actual target is quantified using a loss function. Backpropagation is then used to calculate the gradients of the loss with respect to the network's weights. This information guides the optimization algorithm (for example gradient descent) to adjust the weights and minimize the loss.

Diving into the the primary components of a neural network that were just mentioned
we can focus on the specific role and function of each component.

*Figure(1)*




**Input Layer**

The input layer receives the raw data that the network will process. Each neuron (or node) in this layer corresponds to a feature or input variable. For tabular data the input layer typically receives a vector which represents the features for a single observation, or a matrix which is a concatenation of these feature representations.

**Hidden Layers:**

Between the input and output layers, neural networks typically have one or more hidden layers. These layers are where the network learns and captures complex patterns and relationships in the data. Each hidden layer comprises multiple neurons, and the number of hidden layers and neurons per layer can vary depending on the network's architecture. In Figure (1) for example we can see how each neuron of the hidden layer takes as input 3 input values coming from the input layer and performs this computation $y = f\left(\sum_{i=1}^{3} x_i * w_i + b\right)$ where $x_i$ are the input values, $w_i$ the weights, $b$ the bias and $f()$ an activation function. Then, the neurons of the second hidden layer will take as input the outputs of the neurons of the first hidden layer and so on.

**Neurons (Nodes):**

Neurons are the fundamental processing units within a neural network. They receive input, perform a computation, and pass the result to the next layer. Each neuron applies a weighted sum of its inputs, adds a bias term, and then applies an activation function to produce an output.

**Weights and Biases:**

Weights and biases are the parameters that neurons use to make decisions. Weights determine the strength of connections between neurons, indicating how much influence one neuron has on another. Biases help shift the output of a neuron. These parameters are learned during training to minimize the network's error in making predictions.

**Activation Function:**

Activation functions introduce non-linearity to the neural network. They help the network learn complex relationships in the data. Common activation functions include:

- Sigmoid, which maps values to a range between 0 and 1.
- ReLU (Rectified Linear Unit): Outputs the input if it's positive; otherwise, it outputs zero.
- TanH (Hyperbolic Tangent): Maps values to a range between -1 and 1.
- The Softmax activation function takes in a vector of raw outputs of the neural network and returns a vector of probability scores.

**Output Layer:**

The output layer produces the final predictions or outputs of the neural network. The number of neurons in this layer depends on the problem type. For example, in binary classification, there might be one output neuron (0 or 1), while in multi-class classification, there would be as many output neurons as there are classes.Typically the activation function of the output layer is a Linear one for regression, a Sigmoid for binary classification and a Softmax for multiclass classification.

**Loss Function:**

The loss function (or cost function) measures the difference between the predicted outputs and the actual target values. It quantifies how well or poorly the network is performing on the given task. The goal during training is to minimize this loss function. Each type of task has a set of appropriate loss functions that are typically used.

Binary Cross-Entropy Loss also known as Log Loss is used for binary classification tasks.

$$[\text{Binary Cross-Entropy Loss (Log Loss)} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p(y_i)) + (1-y_i)\log(1-p(y_i))]]$$

- $N$ is the number of data samples.
- $y_i$ is the true label (0 or 1) for the $i'th$ sample.
- $p(y_i)$ is the predicted probability that the $i'th$ sample belongs to class 1.

Categorical Cross-Entropy Loss: This loss function is used for multiclass classification problems, where there are more than two classes.

$$[\text{Categorical Cross-Entropy Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(p_{ij})]$$

- $N$ is the number of data samples.
- $C$ is the number of classes.
- $y_{ij}$ is a binary indicator (0 or 1) of whether class $j$ is the correct classification for sample $i$.
- $p_{ij}$ is the predicted probability that sample $i$ belongs to class $j$.

Mean Squared Error (MSE) Loss: This loss function is commonly used for regression tasks, where the goal is to predict continuous values.

$$[\text{MSE Loss} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2]$$

- $N$ is the number of data samples.
- $y_i$ is the true target value for the $i$-th sample.
- $\hat{y}_i$ is the predicted target value for the $i$-th sample.

**Optimizer**:

The optimizer is responsible for updating the weights and biases of the network to minimize the loss function. Gradient-based optimization algorithms, such as stochastic gradient descent (SGD) and Adam, are commonly used for this purpose. The primary goal is to find the optimal set of parameters that result in the lowest possible loss. Optimizers use gradients of the loss function with respect to the model parameters to update them iteratively.

SGD: It updates model parameters iteratively to minimize the loss function.
This update of the model parameters $\theta$ is performed using the gradient of the loss function with respect to $\theta$ and a fixed learning rate $\eta$ as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla L(\theta_t, \text{mini-batch})$$

In the context of a neural network we can think of $\theta$ as a parameter vector containing the model's parameters, weights and biases.

$$[\theta = \left(W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}, \ldots, W^{(L)}, b^{(L)}\right)]$$

In this representation:

- $L$ represents the total number of layers in the neural network.
- $\theta$ is a vector that contains all the weights and biases of the network, organized as indicated.

The gradient $\nabla L(\theta_t, \text{mini-batch})$ measures how the loss changes with respect to the model parameters.
SGD updates $\theta$ in the direction that reduces the loss, scaled by the learning rate.

Adam: Adam maintains two moving averages: the first moment estimate ($m_t$) and the second moment estimate ($v_t$) of gradients. It combines them to update $\theta$ as follows:

$$\begin{aligned}
m_t &= \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \nabla L(\theta_t, \text{mini-batch}) \\
v_t &= \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot (\nabla L(\theta_t, \text{mini-batch}))^2 \\
\hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\
\theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t
\end{aligned}$$

$\beta_1$ and $\beta_2$ are smoothing parameters for the first and second moments; they control the decay rates of the moving averages ($m_t$ and $v_t$). They determine how quickly or slowly the optimizer forgets past gradient information, influencing the adaptability of the optimization process.
$\eta$ is the learning rate, and $\epsilon$ is a small constant to prevent division by zero.

Adam therefore adapts learning rates for each parameter based on the magnitudes of gradients.

$m_t$ and $v_t$ are moving averages that help adjust learning rates, they provide a smoothed and more stable estimate of the gradient statistics, reducing the noise and variance in gradient updates. By averaging gradients over time, the optimizer is less sensitive to the noisy variations in individual gradient values.
It ensures smoother parameter updates, which can help convergence, especially when dealing with noisy data. These moving averages adaptively adjust the learning rates for each parameter based on the historical gradient information, improving the optimization process.

The final update combines these estimates to update $\theta$ in a direction that reduces the loss efficiently.

The process of adjusting the weights based on the gradients is referred to as training. Neural networks learn by iteratively updating the weights to minimize the loss function. This process involves finding a balance between fitting the training data well and avoiding overfitting, so typically the dataset is divided into a training and validation set and the true goal is the one of optimal generalization capabilities of the model and not merely the one of minimizing the loss on the training set.

Neural networks with multiple hidden layers are known as deep neural networks and therefore fall into the category of Deep Learning methods . Deep learning leverages the depth of networks to automatically extract hierarchical representations of data. It has led to breakthroughs in various domains due to its ability to learn complex patterns from large amounts of data.

The components just described are the core components that all neural networks share, indeed there is a very large literature which contributed to creating much more complex and convoluted neural network architectures and techniques which were proven to increase the potential predictive performance of neural networks.

The undisputed leaders in the tabular domains are tree-based models: Random Forests, Gradient Boosted Trees to name the most popular implementations which build upon the concept of a decision tree. On the other hand Neural Network architectures in other domains such as computer vision and natural language processing have accomplished astounding achievements, particularly in the last few years.

The development of deep learning architectures tailored for tabular data has been the subject of intense research.The possibility to dethrone tree-based models is particularly appealing. Tree-based models also lack differentiability, making it difficult to incorporate them into joint training with other deep learning components. Tree-based models lack differentiability because they are based on a series of binary decisions that are not easily differentiable. Each decision in a tree-based model splits the data into two or more subsets based on a threshold value, creating a hierarchy of nodes and leaves. This structure is difficult to differentiate because the output of each node only depends on the input variables and the threshold value, rather than a continuous function. On the contrary, neural networks can be deployed for multimodal learning problems where tabular data can be a component alongside other data types such as text and images.

The lack of differentiability in tree-based models can make it difficult to understand exactly why the model arrives at its predictions. Since each decision in the tree is based on a simple threshold value, it can be difficult to determine which variables and combinations of variables are most important in making the prediction. Additionally, the hierarchical structure of the tree can make it difficult to visualize and interpret the model's decision-making process. This can be problematic in situations where interpretability and transparency are important, such as in healthcare or finance, where decisions made by machine learning models can have significant consequences. In these cases, it is important to be able to understand which variables and features are most important in making the prediction, and how changes in the input data can affect the output.There are a number of methods which allow us to explain the model’s decisions such as feature importance methods. However, these methods have limitations, including potentially missing important interactions between features, being biased towards certain types of features, and being sensitive to hyperparameters.

Deep learning techniques, on the other hand, are highly differentiable and can be trained using backpropagation to optimize the model parameters. This makes it, depending on the type of architecture used, potentially easier to understand how the model arrives at its predictions, and to identify which input features are most important in making the prediction. Additionally, deep learning architectures can be explicitly designed to be more interpretable, such as by using attention mechanisms or visualizing feature activations.

## 2.2 Deep Learning Strategies for Tabular Data

The focus of this section is to highlight the different groups of methods which are used to create neural network architectures applied to tabular data. We are going to briefly mention and describe some of the most well known as well as recent research papers proposing novel neural network architectures for tabular data tasks.
There are several promising deep learning architectures that have been proposed recently for tabular data. In this section I attempt to list a number of ones which caught the public's attention for their state-of-the-art performance in some of their respective applications, providing a brief summary of their architecture and the datasets they were evaluated on. It is evident that these methods were all evaluated on different datasets and using different evaluation schemes and metrics, not all of these papers compare their novel architecture to state-of-the-art tree based methods and not all of them provide code to reproduce and expand their results. These architectures were constructed to try and combat the inherent challenges of tabular data, which are characterized by feature heterogeneity, mixed input types, and varying levels of reliability and integrity. Tabular data also presents challenges such as feature inter-correlations and redundancy, missing values, poor scaling, and noisy measurement. These challenges often lead to data situations with low signal-to-noise ratio, making it difficult to devise a model that will easily perform well.
In their paper [14] the authors categorize tabular deep learning methods into three groups: data transformations, specialized architectures, and regularization models. Most of the successful architectures usually use a combination of two or more of these techniques.

**Data Transformation Methods**

The first group, data transformation methods, transform categorical and numerical data to enable deep neural network models to extract information signals better. These methods do not require new architectures or adaptations of the existing data processing pipeline but increase the preprocessing time. The area can be further subdivided into Single-Dimensional and Multi-Dimensional Encodings.
Single-dimensional encoding involves representing categorical variables using a single value. This value is typically a scalar, number, or code that is associated with each category. This approach simplifies the categorical data by converting it into a numerical format that can be directly input into a neural network. Examples of

single-dimensional encoding methods include ordinal encoding, one-hot encoding, binary encoding, and leave-one-out encoding.
In ordinal encoding category is mapped to a unique integer value. However, this method introduces an inherent order among categories, which may not always be appropriate. One-Hot Encoding transforms each category into a binary vector, where each element corresponds to the presence or absence of the category. It's a widely used method, but it can lead to high-dimensional and sparse feature vectors.
In binary encoding categories are first assigned unique integer values and then converted to their binary representations. This reduces dimensionality compared to one-hot encoding.
In Leave one out encoding categories are replaced with the mean of the target variable for that category while excluding the current instance to avoid overfitting.

Multi-dimensional encoding, on the other hand, aims to create a multi-dimensional representation for categorical variables. Instead of representing a category using a single value, each category is mapped to a multi-dimensional vector. This approach attempts to capture more complex relationships and interactions between categories. One example of multi-dimensional encoding is the Variational Implicit Matrix Estimation (VIME) [31] approach, which transforms categorical and numerical features into a new homogeneous representation using a self-supervised deep learning framework.

**Specialized Architectures**

Specialized architectures constitute a significant group of deep learning approaches designed explicitly for heterogeneous tabular data. This group can be categorized into two sub-groups: Hybrid models and transformer-based models.
Hybrid Models combine data transformation techniques with deep neural networks and successful classical machine learning approaches, especially decision trees. They can be classified into fully differentiable and partly differentiable models.
Fully Differentiable Models allow end-to-end deep learning using gradient descent optimizers. Examples include the Regularization Learning Network (RLN) [35], ensemble of differentiable oblivious decision trees (NODE) [19], and models like soft decision trees (SDT) [36] for improved interpretability.
Partly Differentiable Models blend non-differentiable approaches (like decision trees) with deep neural networks. Examples include DeepGBM [33], TabNN [37], and Boosted Graph Neural Network (BGNN) [38].

**Transformer-based Models**

Inspired by the success of transformers in text and visual data, researchers have proposed using attention mechanisms for tabular data.
The transformer mechanism was introduced in a paper called "Attention is All You Need" [39] and it consists of self-attention layers that allow the model to selectively focus on different parts of the input sequence. This attention mechanism allows the model to capture long-range dependencies and relationships between different parts of the input sequence, which is particularly useful in natural language processing tasks.
Transformer-based architectures have become popular in recent years, achieving state-of-the-art performance in various natural language processing tasks such as language translation, question-answering, and sentiment analysis and this led to the question of assessing their usability also in the tabular data domain.

These specialized architectures cater to the unique challenges of tabular data and offer various trade-offs between accuracy, interpretability, and performance. They incorporate techniques such as decision trees, attention mechanisms, and hybrid combinations to extract valuable insights and patterns from tabular data.

**Regularization Methods**

The third category of approaches focuses on addressing the high flexibility of deep learning models for tabular data by applying strong regularization to improve overall performance.
In the context of neural networks, regularization involves a range of techniques used to address overfitting, a situation where a model becomes too specialized in the training data and struggles with new, unseen data. These methods aim to ensure that the model captures meaningful patterns in the data while avoiding learning noise. L1 and L2 regularization approaches introduce additional terms to the loss function, encouraging the model to have smaller weights and less dense weight distributions. The Dropout technique randomly deactivates certain neurons during training to promote adaptability. Data augmentation introduces diversity by introducing slight variations into the training data.
Batch normalization works to stabilize training by standardizing activations in each layer. Weight constraints help to prevent weights from becoming excessively large.

Regularization loss terms, such as penalties for sparsity, guide the learning process. Ensemble methods combine multiple models to improve overall generalization. Regularization essentially finds a balance between closely fitting the data and preventing overfitting. Its effectiveness varies based on the specific problem, dataset, and architecture, influencing both the model's learning process and its ability to generalize to new data.

An early method in this group is RLN [35]. The RLN employs trainable regularization coefficients for each weight in a neural network, effectively reducing the sensitivity of the network. These coefficients are determined using a novel "Counterfactual Loss" in the learning process, resulting in a sparse network that highlights the importance of remaining input features.
In a similar study, “Regularization is All You Need” [40], authors emphasize that well-regularized simple multilayer perceptrons can outperform complex algorithms on tabular data. They propose a series of thirteen different regularization techniques. Despite the benefits of improved performance, the extensive per-data set regularization and hyperparameter optimization incur higher computational costs compared to tree-based methods like gradient boosting decision trees.
These methods support the idea that strong regularization of deep neural networks can offer advantages for tabular data.

## 2.3 Benchmarking Performance of DL vs Classical ML on Tabular Data

### Existing Literature

Several papers compare Tabular Deep Learning Methods with Tree-Based methods, the general consensus is that the second group of methods outperforms the first. In the publicly available literature on the comparison between these fundamentally different machine learning architectures, the authors' selection criteria for models, datasets and methodology are varied. For reasons such as computational capacity as well as arbitrary choices made by the authors, assumptions are made to make the problem more manageable. There are a number of assumptions made regarding which datasets to use, how to treat and transform the data, how to train and evaluate the models that leave a wide space for future research to explore.
While many recent publications assert that their models outperform or match tree-based models, doubts have been raised about the validity of these claims. For instance, a basic Resnet architecture has been shown to perform competitively with

some of these new models, as shown by “Revisiting deep learning models for tabular data” [12]. Often the claims in the relative research papers are then questioned by further studies who seek to put these novel architectures to the test and benchmark them against, not surprisingly, state of the art decision tree based models.

The absence of a standardized benchmark for tabular data learning allows researchers more flexibility in evaluating their methods. However, this also means that there is no widely accepted standard for comparing different approaches. Another challenge in evaluating tabular data methods is that most publicly available datasets are comparatively small in scale when compared to benchmarks in other machine learning domains such as ImageNet [13]. This can result in increased variability in evaluation metrics and make it more difficult to draw definitive conclusions about the comparative performance of different methods.

Some research papers have attempted to tackle these issues by structuring an analysis of comparison between tabular models and tree based models to evaluate their performance in a more complete and fair fashion.

In “Deep Neural Networks and Tabular Data: A Survey” [14], published in 2021, the authors systematically explore the application of deep neural network approaches to heterogeneous tabular data. The authors acknowledge the prominence of tabular data in critical applications and computational tasks. While deep neural networks have excelled on homogeneous datasets, adapting them to tabular data for inference or generation tasks remains a complex endeavor. The work contributes by categorizing deep learning methods for tabular data into three groups: data transformations, specialized architectures, and regularization models. This categorization offers a comprehensive overview of the main approaches in each group. The paper also delves into deep learning methods for generating tabular data and strategies for explaining deep models on such data. This work bridges a gap by addressing existing research streams, methodologies, challenges, and open questions. Furthermore, the paper conducts an empirical comparison of traditional machine learning methods and eleven deep learning approaches on five real-world tabular datasets of varying sizes and learning objectives. The publicly available results indicate that gradient-boosted tree ensembles continue to outperform deep learning models in supervised learning tasks, signaling a stagnation in competitive deep learning progress for tabular data. This comprehensive survey provides an unprecedented overview of deep learning methods for heterogeneous tabular data. It establishes a unified taxonomy for categorizing these methods and offers insights into the challenges and advances related to modeling, generating, and explaining

tabular data. The authors anticipate their work will guide researchers and practitioners seeking to navigate the complex landscape of deep learning applications in tabular data scenarios.
The paper “Why do tree-based models still outperform deep learning on tabular data?” [15] , published in 2022, contributes by conducting extensive benchmarking, comparing 4 Deep Learning learning techniques with established tree-based models like XGBoost and Random Forests. The authors curate a standardized set of 45 datasets characterized by typical tabular data traits. The methodology covers model fitting and hyperparameter optimization, revealing that tree-based models maintain their advantage in medium-sized data situations, even without considering computational efficiency.The research delves deeper, seeking to explain the performance difference between tree-based models and Neural Networks (NNs). Empirical investigations uncover distinctive inductive biases exhibited by these two approaches. This leads to the authors identifying three key challenges that researchers should consider when designing Neural Networks for tabular data: handling uninformative features, preserving data orientation, and efficiently learning irregular functions. To encourage further exploration, the paper offers a standardized benchmarking platform with code available on github to replicate their results and shares raw data used for baseline performance. Additionally, it provides insights into the substantial computational effort invested in hyperparameter searches, serving as a valuable resource for fellow researchers.

In “Tabular Data Deep Learning is not all you need” [16], published in 2021, the authors conduct a thorough comparison between various deep learning models and XGBoost across different datasets.The study conducted experiments using 11 diverse tabular datasets, representing both classification and regression problems. These datasets vary in terms of the number of features (ranging from 10 to 2,000), classes (1 to 7), and samples (7,000 to 1,000,000). The datasets also exhibit differences in the composition of numerical and categorical features. Some datasets include "heterogeneous" features with varying units of measurement, while others contain "homogeneous" features like pixels or words. The datasets were sourced from various papers and sources, including TabNet [17], DNF-Net [18], NODE [19], and Kaggle [20]. The data preprocessing and training followed the procedures outlined in the original papers. Standardization was applied to the data to have zero mean and unit variance, with statistics computed from the training data. They focus not only on performance but also on aspects such as tuning requirements and computation. The study demonstrates that XGBoost [21] consistently outperforms

the deep models on a range of datasets, including those where the deep models were initially proposed to excel. The authors propose an ensemble approach that combines XGBoost with deep models, showing that this ensemble performs better than individual models or classical ensembles.

The authors emphasize that despite the advancements in deep models for tabular data, they do not surpass the performance of XGBoost on the explored datasets. The authors suggest that further research is needed in this domain, urging future studies to systematically evaluate performance on diverse datasets. They also point out the importance of considering factors beyond just performance, such as the ease of hyperparameter optimization. The paper suggests that while XGBoost might achieve the best results within time constraints and ease of optimization, an ensemble approach that includes deep models could potentially lead to even better performance. The paper concludes that deep learning for tabular data still requires more investigation, including the development of new deep models that are easier to optimize and can compete with XGBoost in terms of performance.

The issue of improper baselines in the deep learning for tabular data literature is discussed specifically in [12] and the authors propose two strong baselines: a ResNet-like architecture and a transformer-based architecture called FT-Transformer (Feature Tokenizer + Transformer). The FT-Transformer outperforms other deep tabular methods in six out of 11 cases and has the best overall rank across all 11 datasets considered in the study. The most competitive deep tabular method is NODE, which outperforms other methods in four out of 11 cases. When compared with gradient boosted trees such as XGBoost and CatBoost [22] , the FT-Transformer outperforms the former in seven out of 11 cases, leading the authors to conclude that there is no universally superior method.

## Why do Tree Based Models Perform Better?

### Data Quality

A first explanation is based on the fact that tabular datasets frequently contain a multitude of features that lack substantial informational content. Tabular data often suffer from issues such as missing values, outliers, erroneous data, inconsistency, and class imbalance, which affect the performance of machine learning algorithms. Decision tree-based methods are better equipped to handle these issues. Authors in [15] investigate this hypothesis by systematically removing features based on their importance, as determined by the feature importance of a Random Forest model

trained on the same task. The removal of up to half of the least important features minimally affects the classification accuracy of GBTDs. Conversely, MLP based architectures exhibit heightened sensitivity to the presence of uninformative features. In their results, the removal of uninformative features reduces the performance gap between neural networks and MLP based Neural Networks.

We can identify other characteristics which we can place into the realm of Data Quality issues, and those are dependency on preprocessing and importance of single features. Deep learning on homogeneous data often requires minimal preprocessing due to implicit representation learning. However, tabular data often need extensive preprocessing, especially for handling categorical features, which can lead to information loss and affect predictive performance. Small changes in categorical or binary features can significantly impact predictions in tabular data, the smallest possible change of a categorical (or binary) feature can entirely flip a prediction [35], unlike decision tree based models, which typically require coordinated changes in many features. Decision tree algorithms excel at handling varying feature importance by selecting a single feature and threshold values, effectively "ignoring" the rest of the potentially misleading data.

**Neural Networks Favor Overly Smooth Solutions**

A second explanation that is also intrinsically tied to the first, is that neural networks tend to favor overly smooth solutions. Tabular data, especially real world large datasets contain noise and irregularities and Neural Networks struggle to generalize. Authors in [15], investigate this hypothesis they modified each training dataset by applying a Gaussian Kernel smoothing technique with various length-scale values for the kernel on the target function. The higher the intensity of the smoothing the more the accuracy of tree-based models was reduced, whereas the impact on the performance of neural networks was much smaller. A finding that is also in line with Rahaman et al.'s "On the Spectral Bias of Neural Networks" [41] findings, which indicate that neural networks have a bias towards low-frequency functions.

**Spatial Dependencies**

Spatial dependencies refer to the relationships or similarities between neighboring data points within a two-dimensional space. In the context of images, adjacent pixels often share certain visual attributes, such as color or texture. This spatial correlation is exploited by convolutional operations in algorithms like CNNs to capture meaningful patterns and features. However, in tabular datasets, where features are

columns without a natural spatial arrangement, this kind of correlation doesn't naturally exist. Feature dependencies can be complex and irregular. This makes it challenging for deep learning models, which rely on inductive biases not suited for tabular data.

## 2.4 Novel Tabular Deep Learning Architectures

In this section I will briefly summarize the works of some of the most recent and well known novel deep learning architectures. It is not surprising that most if not all of the research in this field explicitly compares the performance of their model to the performance of a GBDT algorithm. As mentioned previously, often the reported findings of superiority to the novel architectures are either not replicable or challenged by future research.

GATE (Gated Additive Tree Ensemble) [23], published in 2022, is a deep learning approach for tabular data that utilizes a hierarchical stacking of decision tree stumps. It is based on differentiable decision trees but also includes self-attention for reweighting the outputs and is inspired by GRU gating mechanisms [24]. GATE was evaluated on 5 large datasets (3 classification and 2 regression datasets) with up to 800 million training examples, and achieved the highest average rank (tied with LightGBM). However, the lack of available code effectively limits the replicability of the results.

In "Revisiting Pretraining Objectives for Tabular Deep Learning" [25] published in 2022, the authors investigate different pretraining strategies for deep tabular methods. Unlike most tree-based methods, deep neural networks support pretraining and can be trained iteratively. The authors study various pre-training schemes for multilayer perceptrons, including self-prediction versus contrastive methods, masked-based self-prediction versus reconstruction, and target-aware versus target-agnostic. They use the target dataset for pretraining instead of additional data and evaluate their approach on 11 different datasets with varying characteristics. The study finds that numerical embedding-based multilayer perceptrons with target-aware mask-based (self-prediction) pretraining perform best in most cases, with only two datasets where gradient boosting methods outperform the pretrained MLPs.

"Neural Additive Models: Interpretable Machine Learning with Neural Nets" [26] published in 2020 proposes Neural Additive Models (NAMs), which are essentially an ensemble of multilayer perceptrons (MLPs) where each MLP is used per input feature. The output values are summed and passed to a logistic sigmoid function for

binary classification, and the method is easy to interpret as each input feature is handled independently by a different neural network. The NAM was evaluated on four datasets (two classification and two regression datasets) and performed slightly worse than XGBoost across all four datasets but better than Explainable Boosting Machines on two out of four datasets.

In "Scalable Interpretability via Polynomials" [27] published in 2022, the authors propose Scalable Polynomial Additive Models (SPAM), which is a type of generalized additive model that prioritizes interpretability. Unlike the Neural Additive Model (NAM), which requires an individual neural network for each feature, SPAM is easier to scale. The authors evaluate their approach on three tabular datasets and find that the SPAM-Neural model, which shares the same multilayer perceptron architecture as NAM, outperforms NAM on all three datasets and outperforms XGBoost in two out of three cases.

"On Embeddings for Numerical Features in Tabular Deep Learning" [28] published in 2022 explores embedding methods for tabular data, specifically a piecewise linear encoding of scalar values and periodic activation-based embeddings, instead of designing new architectures for end-to-end learning. The experiments demonstrate that the embeddings are beneficial not only for transformers but also for other methods, such as multilayer perceptrons, which are competitive with transformers when trained on the proposed embeddings. Using the proposed embeddings, ResNet, multilayer perceptrons, and transformers outperform CatBoost and XGBoost on several (but not all) datasets. However, a control experiment where CatBoost and XGBoost are trained on the proposed embeddings would have provided a more comprehensive comparison.

"SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training" [29] published in 2021 proposes the SAINT hybrid architecture based on self-attention that applies attention across both rows and columns, along with a self-supervised learning technique for pre-training under scarce data regimes. The proposed SAINT method tends to outperform gradient-boosted trees when looking at the average performance across all nine datasets, which ranged from 200 to 495,000 examples.

"TabTransformer: Tabular Data Modeling Using Contextual Embeddings" [30] published in 2020 proposes a transformer-based architecture based on self-attention that can be applied to tabular data. The authors also propose a semi-supervised approach leveraging unsupervised pre-training. However, the official implementation is not available, and only open-source implementations are available

on GitHub. Looking at the average AUC across 15 datasets, the proposed TabTransformer performs on par with gradient-boosted trees.
"VIME: Extending the Success of Self- and Semi-supervised Learning to Tabular Domain"[31] published in 2020 introduces VIME (Value Imputation and Mask Estimation), which includes self- and semi-supervised learning frameworks for tabular data. The authors provide good ablation studies demonstrating that the semi-supervised learning variant of VIME outperforms the supervised-only and self-supervised-only variants. The best VIME variant uses both self- and semi-supervised learning and outperforms XGBoost on all five datasets used for comparison.
"Neural Oblivious Decision Ensembles for Deep Learning on Tabular Data" [19] published in 2019 proposes the Neural Oblivious Decision Ensembles (NODE) method, which combines decision trees and deep neural networks to be trainable in an end-to-end fashion. The method is based on oblivious decision trees (ODTs), a type of decision tree that uses the same splitting feature and splitting threshold in all internal nodes of the same depth. The experiments were conducted on six large datasets, ranging from 400K to 10.5M training examples. NODE slightly outperforms XGBoost on all six datasets when default hyperparameters are used; with tuned hyperparameters, NODE outperforms XGBoost in four out of six cases. However, the training speed of NODE is reported to be slower than XGBoost, taking about seven times longer to train and two times longer for inference.
"TabNet: Attentive Interpretable Tabular Learning" [17] published in 2019 proposes the TabNet architecture based on a sequential attention mechanism that shows self-supervised learning with unlabeled data can improve performance over purely supervised training regimes in tabular settings. Across six synthetic datasets, TabNet outperforms other methods on three out of six cases, with extremely randomized trees as the tree-based reference method. However, XGBoost was omitted from the comparison. Across four KDD datasets, TabNet ties with CatBoost and XGBoost on one dataset and performs almost as well as the gradient-boosted tree methods on the remaining three datasets.
"ARM-Net: Adaptive Relation Modeling Network for Structured Data" [32] published in 2021 proposes a method that transforms inputs into an exponential space and uses a sparse attention method to generate interaction weights to obtain cross-features for prediction. This approach is essentially a transformer-inspired embedding method followed by a multilayer perceptron for prediction. However, besides logistic regression, the paper does not include comparisons with

conventional machine learning methods, which may limit the ability to generalize the results.
There are also some interesting implementations which try to incorporate tree based structures and their structural advantages and combine them with deep learning methods.

"DeepGBM" [33] published in 2020 proposes a hybrid architecture that combines gradient boosting machines with deep neural networks to leverage both the strengths of tree-based models and deep learning methods. The authors use a novel layer-wise training strategy to train the deep neural network component, which helps to avoid overfitting and improve generalization performance. The gradient boosting component is used to generate high-quality predictions that are fed into the deep neural network component to further refine the predictions. The experiments show that DeepGBM outperforms both gradient boosting machines and deep neural networks on a variety of benchmark datasets, including the Higgs and Microsoft Malware Classification challenges. In essence, DeepGBM aims to combine the interpretability and efficiency of gradient boosting machines with the flexibility and expressiveness of deep neural networks.

"XBNet: An Extremely Boosted Neural Network" [34] published in 2021 proposes a method that initializes the weights of a multilayer perceptron using feature importances derived from XGBoost models. After initialization, the method trains an additional XGBoost model for each intermediate layer, with the feature importances used to update the neural network weights during backpropagation. The XGBnet method was evaluated on eight small datasets, including the classic Iris and Wine datasets, and outperformed XGBoost on three out of the eight datasets.

In all the existing research on this topic of comparing architectures and their performances in the tabular data domain the choice of datasets is key, datasets are intrinsically different with respect to data heterogeneity, dimensionality, and real-world relevance, so the choice inherently influences the results.

The preprocessing steps also introduce assumptions. Removing missing data, balancing classes, and handling categorical features impact model performances. Preprocessing decisions might favor certain models over others due to their inherent characteristics. For instance, handling missing data can be approached differently by tree-based models and neural networks, potentially biasing results.

In [15], the removal of datasets with missing data or high cardinality categorical or numerical features would effectively exclude many real-world datasets that contain a number of high frequency categorical variables. Having missing data is extremely common in real world datasets. An organization who is thinking of developing or already has machine learning models in production could not afford to simply drop missing data or exclude high frequency categorical features which might have good predictive power on the task they are interested in. They also remove datasets where too little information is available, once again this has its limitations, in Big Data a large issue is documentation. It is common to have to use datasets and features for which not much information is present. What is also particularly interesting is their choice of restricting the train set size to 50,000 samples when studying "large" datasets. It is also very common, especially for relatively large enterprises to have datasets with millions of observations. In their findings they report that, "in most cases, increasing the train set size reduces the gap between neural networks and tree-based models" and "leave a rigorous study of this trend to future work." Their work also chooses to handle class imbalance in an arbitrary way.
The authors highlight these limitations and state that there is still large room for further investigation and improvement, particularly with regards to the application of deep learning models to real-world large datasets.
Similarly the authors in [14] make use of 5 popular open source datasets to perform the comparison and only 1 of these datasets has more than 1 million rows, a size which is extremely common in many real world applications. They also choose to fill missing values with 0s and use ordinal encoding for all categorical variables if the model does not accept categorical values natively.

Investigating how both tree-based models and neural networks address specific challenges, such as missing data, high-cardinality categorical features, and applications to large datasets are all potential topics to explore more in depth.

Additionally, evaluating the effectiveness of a model's predictions is critical in understanding its performance and suitability for specific applications. However, the choice of evaluation metric can significantly impact the interpretation of a model's performance, especially in different problem domains. In many instances, the current literature tends to employ certain metrics that are commonly associated with particular types of tasks. For classification tasks, the metrics most frequently utilized include Accuracy and Logloss, which measure the overall correctness of predictions and the quality of predicted probabilities, respectively. Similarly, for regression

tasks, Mean Squared Error (MSE) and R2 Score are commonly adopted to assess the accuracy and goodness-of-fit of predicted values against actual outcomes.

However, the choice of these metrics might not always be appropriate for all scenarios, particularly in classification tasks. While Accuracy provides a straightforward measure of the proportion of correct predictions, it may be misleading in cases involving imbalanced datasets, where the class distribution is significantly skewed. In such situations, where the number of instances in one class greatly outweighs the other, Accuracy might appear high simply due to the dominance of the majority class. This could potentially mask the model's inability to effectively identify instances from the minority class, which is often of more significant interest. Most papers also resort to the AUC metric, the Area Under the ROC Curve can be optimistic under a severe class imbalance, especially when the number of examples in the minority class is small.

For example in a highly unbalanced setting such as identifying credit default risk, accuracy and AUC, used by [14] might not be the best metrics to analyze and we could be more interested in a different metric such as precision, or lift score.

It is important to recognize that there is no universally "best" metric for all scenarios. The choice of metric depends on the specific characteristics of the task, dataset, and the objectives of the analysis. Researchers must carefully consider the nature of the problem, the class distribution, and the potential implications of misclassification before deciding on an appropriate metric. By doing so, the evaluation process becomes more accurate and reflective of the true performance of the model, ultimately leading to more informed decisions in practical applications.

While prevailing consensus leans towards the superiority of tree-based methods, the intricacies of comparison methodologies continue to raise questions for further investigation. The assumptions made in dataset selection, preprocessing, and evaluation represent a frontier where future research can deepen our understanding of the capabilities and limitations of these distinct machine learning paradigms.

# 3 Methodology

## 3.1 Models

### Decision Tree Based Architectures Tested

#### Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting (XGBoost) emerged as an evolution of traditional Gradient Boosting algorithms, particularly Gradient Boosting Decision Trees (GBDT). It was developed to address the limitations of its predecessors and further enhance predictive modeling. XGBoost was created by Tianqi Chen, a machine learning researcher, and it gained significant popularity in the machine learning community for its remarkable performance and flexibility. XGBoost builds upon the principles of GBDT, inheriting the concept of boosting, which involves combining multiple weak learners (typically shallow decision trees) to create a strong predictive model. However, XGBoost introduces several innovations and optimizations that set it apart:

**Regularized Objective Function:**

XGBoost uses a regularized objective function that combines a loss function (e.g., mean squared error for regression or log loss for classification) with a regularization term. This helps prevent overfitting by penalizing complex models.

**Efficient Tree Pruning:**

XGBoost incorporates a mechanism for pruning trees during their construction, eliminating splits that do not contribute significantly to reducing the objective function. This makes the algorithm more efficient and prevents deep, overly complex trees.

**Handling Missing Data:**

XGBoost has built-in support for handling missing data. It can automatically learn how to treat missing values during training and prediction, reducing the need for preprocessing.

**Parallel and Distributed Computing:**

XGBoost is highly efficient and can leverage parallel and distributed computing to accelerate training on multicore CPUs and distributed computing clusters.

**Customizable Loss Functions:**
Users can define custom loss functions, opening up the possibility of solving a wide range of machine learning problems.

**Feature Importance Analysis:**

XGBoost provides tools for feature importance analysis, helping users identify which features are most influential in making predictions.

**Early Stopping:**

It supports early stopping, allowing the training process to stop when the model's performance on a validation set ceases to improve, preventing overfitting.

XGBoost's combination of speed, efficiency, and predictive accuracy has made it a go-to choice for data scientists and machine learning practitioners across various domains. Its origins in the pursuit of enhancing gradient boosting techniques have led to a versatile and widely adopted algorithm that consistently delivers state-of-the-art results in machine learning competitions and real-world applications.

## Categorical Boosting (CatBoost)

CatBoost, short for "Categorical Boosting," is another powerful gradient boosting algorithm designed to tackle challenges associated with categorical features in machine learning. Developed by Yandex researchers, it has gained widespread popularity for its ability to handle categorical data seamlessly while delivering high

predictive performance. CatBoost builds upon the principles of gradient boosting, similar to GBDT and XGBoost, with a particular focus on categorical feature support and automatic handling. CatBoost was developed by Yandex, a Russian multinational IT company, and was first introduced in 2017. It was created to address the limitations of existing gradient boosting algorithms when dealing with datasets containing a large number of categorical features. The "Cat" in CatBoost stands for "Categorical," highlighting its primary innovation: efficient and automated categorical feature handling. CatBoost is built upon a number of key components:

**Oblivious Decision Tree (Decision Table)**

At the very core of CatBoost is the Oblivious Decision Tree. An oblivious decision tree, also known as an oblivious decision table, is a type of decision tree structure with a specific constraint: it uses the same splitting feature and splitting threshold for all internal nodes at the same depth level. This constraint ensures that all nodes at a particular depth level make their decisions based on the same feature and threshold, regardless of their position in the tree. In an oblivious decision tree, each internal node at a particular depth level employs a fixed feature and threshold for splitting. This means that all data samples reaching a particular depth level are split using the same criterion, which is determined during training. Due to the uniform splitting, an oblivious decision tree can be represented as a table with $2^d$ entries, where $d$ is the depth of the tree. Each entry corresponds to a combination of binary splits (e.g., 0 or 1) at each depth level. These tables are precomputed during training and are used for efficient inference. Oblivious decision trees are considered weaker learners compared to unconstrained decision trees because they have limited flexibility in terms of feature selection and thresholding. However, their strength lies in their resistance to overfitting when used within ensemble methods.
Inference with oblivious decision trees is highly efficient. Because all nodes at a given depth level use the same feature and threshold, the computations for multiple splits can be carried out in parallel, resulting in rapid predictions.Oblivious decision trees are often used as base learners in ensemble methods, such as gradient boosting. When combined with boosting, these trees can contribute to improved predictive performance.

**Categorical Feature Support:**

CatBoost is specially designed to work seamlessly with categorical variables. It employs a technique called "ordered boosting" that treats categorical features differently during tree construction, reducing overfitting and improving predictive accuracy.

**Optimized Tree Growth**:
CatBoost uses an ordered boosting scheme combined with a specialized algorithm for selecting the optimal splits in a tree, leading to more efficient and less overfit models.

**Automatic Handling of Missing Data:**

CatBoost can automatically handle missing values in both categorical and numerical features, reducing the need for extensive data preprocessing.

**Built-in Regularization**:
The algorithm includes built-in L1 and L2 regularization to control model complexity and prevent overfitting.

**Support for Custom Loss Functions:**
Users can define custom loss functions for specific problem domains.

**Efficient Parallel Processing:** CatBoost is designed for efficient multicore processing, accelerating the training process.

**Feature Importance Analysis:**
CatBoost provides tools for interpreting feature importance, helping users understand which features have the most significant impact on predictions.

## Deep Learning Architectures Tested

### Image Generator for Tabular Data (IGTD)

IGTD falls into the branch of data transformation techniques which try to represent tabular data as images, and then apply popular state of the art computer vision

techniques, or variations of them, to solve the task at hand. IGTD is one of the existing methods which aims to transform tabular data into a more homogeneous format that can be used with convolutional neural networks (CNNs). The IGTD framework introduced in "Converting tabular data into images for deep learning with convolutional neural networks" [42], published in 2021 proposes a metho which transforms tabular datasets as 2D images to be used as input to convolutional networks. This approach is similar to the earlier TAC (Tabular Convolution) [43] method. The CNN trained on IGTD-based images outperforms XGBoost and LightGBM.

The key steps and features of the IGTD algorithm are:

1. Feature-to-Pixel Assignment: IGTD assigns each feature from the tabular data to a specific pixel position in the generated image. This assignment process is crucial as it determines how features are spatially represented in the image.

2. Pixel Intensity Encoding: Once the features are assigned to pixels, the pixel intensity in the image reflects the value of the corresponding feature in the data sample. This encoding ensures that the image retains the original data information.

3. Optimization for Feature Placement: IGTD uses an optimization process to determine the best assignment of features to pixels. It achieves this optimization by minimizing the difference between two rankings:
    a. The ranking of pairwise distances between features.
    b. The ranking of pairwise distances between their assigned pixels in the image.

    By minimizing this difference, IGTD ensures that similar features are placed close to each other in the image, and dissimilar features are placed farther apart. This process enhances the preservation of feature neighborhood structure, which is essential for CNN-based modeling.

4. Iterative Swapping: The optimization process involves an iterative swapping of pixel assignments for features. During each iteration, the algorithm identifies the feature that has not been considered for swapping for the

longest time and seeks the most favorable feature swapping that minimizes the difference in rankings.

By applying IGTD to tabular data, researchers can generate compact image representations that capture the spatial relationships between features. These image representations are better suited for CNN-based modeling compared to traditional tabular data. In the study, IGTD was specifically used to transform gene expression profiles of cancer cell lines and molecular descriptors of drugs into image representations. The CNNs trained on IGTD-generated images demonstrated superior performance in predicting anti-cancer drug responses compared to alternative transformation methods or models trained directly on the original tabular data.

The outstanding performance in this study may, in part, be attributed to the inherent structure of the data employed, which primarily consists of biological information. Biological data often exhibit complex relationships and dependencies among features, given the intricate nature of biological systems. These relationships may include gene interactions, molecular pathways, or cellular functions that can be effectively captured by IGTD's transformation into image representations. By encoding this inherent structure into the images, IGTD allows CNNs to exploit the spatial relationships between features, thereby yielding improved predictive performance.

While the Image Generator for Tabular Data (IGTD) has demonstrated remarkable efficacy in domains characterized by the intricate structure of data, such as bioinformatics and medicine, it is essential to recognize that its performance may not be universally applicable to all domains. In fields where the data exhibit different structural characteristics or where spatial relationships between features are less prominent, IGTD may not perform optimally. The algorithm's effectiveness hinges on the ability to encode meaningful relationships between features into the image representations, and this capability may vary depending on the nature of the data. Therefore, when applying IGTD to other domains, researchers should carefully assess the suitability of the algorithm and consider alternative approaches tailored to the specific data structures and requirements of their respective fields.

## ResNet

The success of deep neural networks in various computer vision tasks has been nothing short of remarkable. However, as network depth increases, the training of very deep networks becomes increasingly challenging due to the vanishing gradient problem. This limitation hindered the development of deeper architectures and hindered further improvements in model performance.
In 2015, the introduction of the ResNet (Residual Network) architecture by Kaiming He et al. [44] presented a groundbreaking solution to this challenge. ResNet not only allowed for the training of networks with hundreds or even thousands of layers but also achieved state-of-the-art results in image recognition tasks.

At the core of the architecture are a number of key components:

### The Residual Block

This block is the fundamental building unit of the architecture and addresses the vanishing gradient problem with a clever structural modification. Instead of relying solely on the output of convolutional layers, the residual block introduces skip connections.
Each residual block contains two convolutional layers, each followed by batch normalization and ReLU activation functions. Importantly, the output of the block is the sum of its input and the output of the convolutional layers. This skip connection provides a direct path for gradients during backpropagation, enabling the training of very deep networks.

### Deep Layer Stacking

ResNet architectures stack multiple residual blocks on top of each other, creating a hierarchy of layers. Deeper layers capture increasingly abstract and complex features, which is critical for image recognition tasks. This deep layer stacking is a key contributor to the architecture's success.

### Bottleneck Architectures

In deeper ResNets, bottleneck blocks are employed to reduce computational complexity. These blocks incorporate convolution layers, reducing the number of

parameters and computation. This modification ensures that extremely deep networks remain computationally efficient.

ResNet architectures have consistently achieved state-of-the-art results in various image classification tasks, including the prestigious ImageNet Large Scale Visual Recognition Challenge [13]. The architecture's performance is further enhanced by variants like ResNet-50, ResNet-101, and ResNet-152, each with a different depth, catering to various task complexities.

Unfortunately this type of architecture is not natively suited to tabular data as it takes as input 2D images. However it can be coupled with methods such as IGTD in order to transform the vector of features into a compatible format, either grayscale single channel or RGB three channel. As mentioned previously the performance of such methods is very dependent on the task at hand and has been shown to perform well in domains where features are inherently related to each other such as the biological and medical field.

### TabTransformer

The first transformer-based architecture on tabular data was introduced by Huang et al. (2020) in their TabTransformer: Tabular Data Modeling Using Contextual Embeddings paper [30].
The main idea in the paper is that the performance of regular Multi-layer Perceptron (MLP) can be significantly improved if we use Transformers to transform regular categorical embeddings into contextual ones.
Categorical embeddings are a classical way to use categorical features in deep learning models. Each categorical value is assigned a unique dense vector representation that can be passed on to the next layers. These embeddings are then concatenated with numerical features and used as input to the MLP. However, categorical embeddings lack the context and do not encode any interactions or relationships between the categorical variables. To address this, the authors propose using transformers, which are currently used in NLP for the same purpose. The transformers can contextualize the embeddings, allowing them to capture the relationships and interactions between the categorical variables. The trained contextual embeddings enable the model to place categorical values close to each other in the vector space, even if they come from different variables. This context makes the embeddings more useful and cannot be achieved using simple categorical

embeddings. The authors also claim and provide evidence regarding the architecture's increased robustness to noisy and missing data.

The TabTransformer architecture comprises a column embedding layer, a stack of N Transformer layers, and a multi-layer perceptron. Each Transformer layer [39] consists of a multi-head self-attention layer followed by a position-wise feed-forward layer.
In the column embedding layer, categorical features are associated with an embedding lookup table $e_{\phi_i}(.)$ . For features with $d_i$ classes, the embedding table contains $d_i$ + 1 embeddings, with one additional embedding reserved for representing missing values. The encoding for a particular value $x_i$ where xi belongs to the set [0, 1, 2, .., di], is defined as $e_{\phi_i}(j) \quad = [c_{\phi_i}, w_{\phi_i j}$ where $c_{\phi_i} \quad \in \mathbb{R}^l$ and $w_{\phi_i j} \quad \in \mathbb{R}^{d-l}$ .

The hyper-parameter $l$ determines the dimension of $c_{\phi_i}$. The unique identifier $c_{\phi_i} \quad \in \mathbb{R}^l$ distinguishes the classes within column i from those in other columns. This unique identifier is a novel concept, specifically designed for tabular data, as opposed to language modeling where embeddings are typically combined with positional encodings. Since tabular data lacks inherent feature ordering, positional encodings are not utilized.
To evaluate different embedding strategies, including choices for hyperparameters '`' and 'd,' as well as the use of unique identifiers and feature-value specific embeddings, the authors perform an ablation study and evaluate it on a list of binary classification datasets.Through experiments on fifteen publicly available datasets, the authors show that the TabTransformer outperforms the state-of-the-art deep learning methods for tabular data by at least 1.0% on mean AUC, and matches the performance of tree-based ensemble models

## TabNet

The TabNet architecture [17] is based upon a number of key components: adaptive feature selection, sequential decision-making, enhanced learning capabilities via non-linear processing, and a design reminiscent of ensemble methods. These elements collectively contribute to its success in handling complex tasks.

The architecture employs a method , Sparse Instance-Wise Feature Selection, for selecting important features on a per-instance basis, which means it adapts to the specific characteristics of each input. This ensures that computational resources are focused on relevant features and not wasted on irrelevant ones.
It then creates a multi-step structure where each step in the sequence contributes directly to the final decision-making process based on the features that have been selected and also contributes to the feature representation constructed in the next step. This sequential approach allows the model to make decisions progressively, refining its understanding at each step.
To boost its ability to learn complex patterns, the architecture introduces non-linear processing for the selected features. This means that the model can capture intricate relationships and dependencies within the data, which may be missed by linear methods. It emulates the benefits of ensemble learning by incorporating higher-dimensional representations and increasing the number of decision-making steps. This ensemble-like approach helps improve the model's overall performance and robustness.
TabNet’s encoding strategy for tabular data involves processing raw numerical features and mapping categorical features with trainable embeddings. It does not rely on global feature normalization except for the input layer but instead employs batch normalization. TabNet's encoding is based on sequential multi-step processing, with each step making decisions about which features to use and contributing to the overall decision.

### Neural Oblivious Decision Ensembles (NODE)

NODE [19] is inspired and developed by a number of researchers who also developed CatBoost, which leverages oblivious decision trees (decision tables) during gradient boosting, resulting in efficient inference and resistance to overfitting. NODE, in essence, generalizes CatBoost and makes the feature choice and decision tree routing differentiable, allowing it to be fully integrated into existing deep learning frameworks like TensorFlow or PyTorch. Moreover, NODE can create multi-layer architectures that resemble "deep" gradient boosting decision trees trained end-to-end, a concept not previously explored. Key design choices within NODE include the use of oblivious decision tables and the incorporation of the entmax transformation for "soft" splitting feature choices.

The entmax transformation [45] is a crucial component of NODE that converts a vector of real-valued scores into a discrete probability distribution. This transformation represents a generalization of the conventional softmax and an alternative known as sparsemax [46], which enforces sparsity in probability distributions. The sparsemax has found applications in diverse fields, including probabilistic inference, topic modeling, and neural attention mechanisms [47] [48]. The authors discuss how entmax is particularly noteworthy because it can generate sparse probability distributions where the majority of probabilities are precisely 0. In their research, they argue that incorporating entmax into the model serves as a valuable inductive bias. It enables the creation of differentiable split decisions within the internal nodes of decision trees. Essentially, entmax can learn to make splitting decisions based on a small subset of data features, similarly to classical decision trees where only one feature is considered for each split. This approach mitigates the undesirable influence of irrelevant features. Additionally, utilizing entmax for feature selection offers computational advantages, enhancing the efficiency of the model.
The authors claim through their experimental analysis on various datasets that NODE demonstrates consistent outperformance of leading GBDT implementations.

### Soft-Ordering One-Dimensional CNN

Kaggle user Baosenguo [48] in a competition [49] aimed at predicting the biological activity of molecules based on their genetic and cellular responses managed to achieve second place with this architecture [50]. The standout feature of the second-place solution was the 1D-CNN model, characterized by its straightforward yet highly effective design. The model builds on the fundamental concept of CNNs, which are extensively used in tasks involving computer vision due to their ability to capture spatial patterns within data. In the context of images, CNNs leverage convolutional kernels to extract distinct features while considering the arrangement of pixels in the data.

However, a challenge arises when applying CNNs to tabular datasets, where the absence of inherent spatial correlations between individual features poses an issue.

To address this challenge, the article introduces the concept of "soft-ordering." This entails rearranging the features within the tabular data to introduce some form of spatial correlation, enabling the subsequent use of 1D convolutional layers.

The architecture of the 1D-CNN model involves a series of steps. Initially, a fully connected (FC) layer is employed to expand the dimensions of the features. This layer serves a dual purpose: it provides an adequate amount of data for processing and generates ordered features that hold meaning within the context of the problem. The FC layer takes the original 937 features and increases them to 4096, and this output is then reshaped into 256 channels. Each channel encapsulates a set of 16 ordered features.

What's important to note is that the values present in these 16-size signals are not direct replicas of the original features. Instead, they represent a non-linear combination of the initial features. This particular characteristic is what led to the term "soft-ordering" 1-dimensional CNN. Following this stage, a series of 1D convolutional layers are applied to the reshaped features and skip-connections are used. Skip connections, also known as residual connections, are a technique commonly used in neural network architectures. They involve connecting the output of one layer directly to a later layer in the network, effectively "skipping" over one or more intermediate layers. This approach aims to mitigate the vanishing gradient problem and facilitate the flow of gradients during training. In the context of the 1D-CNN model, skip connections are employed between certain 1D convolutional layers, allowing information from earlier layers to directly influence the learning process of later layers. This can lead to improved training and the extraction of more complex features.

## Gated Additive Tree Ensemble (GATE)

GATE [23] incorporates a feature selection mechanism inspired by Gated Recurrent Units. It combines this mechanism with a set of differentiable non-linear decision trees, enhanced with straightforward self-attention weighting, to predict target outputs effectively. The author’s claim that, through rigorous experimentation on various datasets, GATE's has a competitive edge over state-of-the-art methods like GBDTs, NODE, and FT Transformers.

The GATE model incorporates several key components and techniques to improve performance in this challenging domain.

### Gating Mechanism and Feature Selection

GATE employs a gating mechanism inspired by Gated Recurrent Units (GRUs) for feature representation learning. It uses this mechanism to select and learn relevant features from the input data. Feature selection is critical for tabular data where some features may be irrelevant or redundant, this method typically used for natural language processing is applied for tabular data for the first time as claimed by the authors.

**Differentiable Non-Linear Decision Trees (DNDT)**

DNDTs are the core of GATE's architecture. These non-linear decision trees aim to capture the patterns and relationships in tabular data. Unlike traditional decision trees, these trees are differentiable, making them suitable for training with gradient-based optimization methods.

**Self-Attention**

GATE also incorporates self-attention layers between the outputs of the decision trees. In GATE they help weigh the importance of different tree outputs when making predictions.

**Ensemble of Trees**

GATE combines the predictions from multiple DNDTs to improve performance. This ensemble approach is a common practice in machine learning, where multiple models are combined to produce more accurate predictions.

GATE is designed to be parameter-efficient, meaning it can achieve good performance with fewer model parameters. This is essential for practical applications, as it reduces the computational resources required for training and inference. Overall, GATE aims to bridge the performance gap between traditional gradient boosted decision trees (GBDTs) and deep learning models in the context of tabular data.

## GANDALF

The Gated Adaptive Network for Deep Automated Learning of Features (GANDALF) [42] is closely related to GATE. It is a specialized architecture designed

for optimal performance in handling tabular datasets, focusing on feature selection and feature engineering. GANDALF is authored by the same researchers who developed GATE and the two architectures share a number of similarities. GANDALF primarily consists of two stages. In the first stage, it utilizes Gated Feature Learning Units (GFLUs) to process input features through a series of hierarchical layers. These GFLUs perform automated feature selection and engineering, learning the best representation for the input features. Then, the learned representation is passed through a standard Feed Forward Network to generate the final predictions.

GFLUs are inspired by gating mechanisms in Gated Recurrent Units (GRUs), with key adaptations for tabular data. These units serve two main functions: feature selection and feature engineering. By stacking multiple GFLUs, GANDALF enables hierarchical learning of an optimal feature representation for the specific task.
The critical distinction between GFLUs and GRUs lies in how they utilize learnable feature masks to softly select subsets of features as input. This soft selection makes the inputs to different GFLUs, which are stacked atop each other, distinct from each other. Essentially, GFLUs decide which information to use from raw features, adapt, and create the most suitable set of features for the task.
The gating mechanism in GFLUs includes reset and update gates, which determine how much information is used for updating internal feature representations. The update gate  and reset gate are computed using sigmoid activations and learnable parameters. These gates facilitate the flow of information through the GFLU hidden states.
In summary, GANDALF consists of a stack of N GFLUs, each contributing to the hierarchical feature learning process. The final learned representation (H) is then fed into a standard K-layer Feed Forward Network with non-linear activations (ReLU) for prediction.

### FT-Transformer

The FT-Transformer [12], which stands for Feature Tokenizer + Transformer, is another adaptation of the Transformer architecture specifically designed for handling tabular data. It aims to transform both categorical and numerical features into meaningful embeddings and then applies a stack of Transformer layers to these embeddings.

Here's a breakdown of the key components and steps in the FT-Transformer architecture:

**Feature Tokenizer**

The Feature Tokenizer module is responsible for transforming input features (both numerical and categorical) into embeddings. For each feature its embedding is computed using a bias term and a feature-specific function. The function depends on the type of feature. For numerical features, the function is implemented as an element-wise multiplication with a learnable weight vector. For categorical features, it is implemented as a lookup operation using a categorical weight matrix. Finally, all feature embeddings are stacked together.

**Transformer**

After feature embedding, the classification token's embedding is appended to it, and a number of Transformer layers are applied successively. The Transformer layers apply self-attention and feed-forward operations to the embeddings. The final prediction is based on the representation of the classification token from the last Transformer layer. This representation is passed through a linear layer followed by a ReLU activation and LayerNorm. The output of this process is the predicted target variable. The author's highlight as potential limitations of the FT-Transformer its complexity. Since it may require significant computational resources and time for training, making it less suitable for cases with a very large number of features. The quadratic complexity of the Multi-Head Self-Attention (MHSA) mechanism can be a limiting factor. To address this, efficient approximations of MHSA can be used.

The conclusions derived from the experimental results in which the authors test a number of deep learning methods on tabular data is that the FT-Transformer performs best on most tasks and becomes a new powerful solution for the field.

## Automatic Feature Interaction Learning via Self-Attentive Neural Networks (AutoInt)

The AutoInt [43] architecture is designed for learning feature interactions automatically and was first applied in the context of click-through rate (CTR) prediction. CTR prediction is a process used in online advertising and

recommendation systems to estimate the likelihood that a user will click on a specific item, such as an ad or a recommended product, when presented with it. AutoInt automatically learns high-order feature interactions by mapping input features into a common low-dimensional space using embeddings and then using multi-head self-attention mechanisms to model complex relationships between different feature fields. This approach eliminates the need for manual feature engineering and allows the model to capture intricate patterns in the data.

AutoInt aims to map a high-dimensional and sparse feature vector into a lower-dimensional space while simultaneously modeling high-order feature interactions. This is accomplished through several layers of computation.

**Input Layer**

The input consists of a feature vector $x$, which is a concatenation of all feature fields. Each feature representation $x_i$ can be either a one-hot vector (for categorical features) or a scalar value (for numerical features).

**Embedding Layer:**

To handle categorical features, which are typically sparse and high-dimensional, AutoInt uses an embedding layer. Each categorical feature $x_i$ is mapped to a low-dimensional dense vector $e_i$ using an embedding matrix $V_i$.Numerical features are also represented in the same low-dimensional feature space.

**Interacting Layer:**

This is perhaps the most crucial component of AutoInt, the key objective of the interacting layer is to model high-order combinatorial features. AutoInt employs a multi-head self-attentive neural network for this purpose. It's used to determine which feature combinations are meaningful. Features from different fields are combined through an attention mechanism, with different combinations evaluated using multiple attention heads. The output of this layer represents the high-order combinatorial features.

**Output Layer:**

The output of the interacting layer consists of feature vectors, including both raw individual features and learned combinatorial features.

### Category Embedding Model

This is perhaps the simplest Deep Learning architecture tested. The architecture is based on a Feed Forward Network with the categorical features passed through a learnable embedding layer. The model is a good starting point for any tabular dataset and serves as a  good baseline to compare against other models.

## 3.2. DATASETS

In total this study utilizes 10 datasets, of which 7 are binary classification tasks, 2 are multiclass classification and one regression. The datasets vary in size, class balancing, number of categorical and numerical features and overall size.
Here is a view of some structural features of all datasets. A number of these datasets have been used as part of existing benchmarks in the literature and it will be interesting to compare these results with the literature.

| Dataset | Total Samples | Total Features | Target Type | Num Classes | Pos. Class Ratio | Num Categorical | Num Numerical |
|---|---|---|---|---|---|---|---|
| iris | 150 | 4 | Classification | 3 | NaN | 0 | 4 |
| breastcancer | 569 | 30 | Classification | 2 | 62.74 | 0 | 30 |
| titanic | 891 | 12 | Classification | 2 | 38.38 | 5 | 7 |
| ageconditions | 1234 | 57 | Classification | 2 | 17.5 | 0 | 57 |
| heloc | 10459 | 24 | Classification | 2 | 52.19 | 0 | 24 |
| housing | 20640 | 10 | Regression | None | None | 0 | 10 |
| adult | 32561 | 15 | Classification | 2 | 24.08 | 8 | 7 |
| diabetes | 101766 | 47 | Classification | 2 | 11.16 | 39 | 8 |
| covertype | 281011 | 54 | Classification | 7 | None | 0 | 54 |
| creditcard | 284807 | 30 | Classification | 2 | 0.17 | 0 | 30 |

**Titanic Dataset [44]**

This dataset is very simple and used in many basic machine learning competitions. The goal is to predict which passengers survived the Titanic shipwreck.

**Iris Dataset [45]**

This, like the titanic dataset, is also what could be defined as a "toy" dataset, it is a very simple multiclass classification task in which the task is to predict 3 different types of irises' (Setosa, Versicolour, and Virginica) with 4 features: Sepal Length, Sepal Width, Petal Length and Petal Width.

**Breast Cancer Dataset [46]**

The Breast Cancer Wisconsin (Diagnostic) Data Set is the third and final relatively simple "toy" dataset in this study and contains medical information on patients computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. These attributes describe the characteristics of the cell and nuclei present in the image. The final data structure is of course tabular.

**Age Conditions Dataset [47]**

The AgeConditions dataset is designed for predicting whether a person has one or more of three specified medical conditions (Class 1) or none of these conditions (Class 0). This prediction is based on measurements of health characteristics. The dataset aims to streamline the process of diagnosing these medical conditions, offering a more efficient and private approach compared to traditional, invasive data collection from patients.
The dataset's context highlights the significance of addressing health issues associated with aging, and it underscores the potential of data science and predictive modeling to contribute to the field of bioinformatics. Current models, such as XGBoost and random forest, have faced challenges in achieving reliable and consistent predictions in cases where lives are at stake. The competition is hosted by InVitro Cell Research, LLC (ICR), a company focused on regenerative and preventive personalized medicine, with the goal of advancing research into aging-related health problems and improving existing diagnostic methods. Participants in the competition are tasked with creating models to predict these medical conditions based on health characteristic data, aiming to enhance the field of bioinformatics and develop innovative solutions for complex healthcare challenges.

**CoverType Dataset [48]**

The CoverType dataset is a commonly used dataset in machine learning for classification tasks. It contains cartographic variables as features that describe geographic attributes of forested areas, and the goal is to predict the type of forest cover present. There are seven possible forest cover types in the dataset. This dataset is often used to practice and demonstrate classification algorithms and techniques.

**HELOC Dataset [49]**

The HELOC (Home Equity Line of Credit) dataset contains financial and credit-related information about individuals who have applied for a home equity line of credit. The dataset can include features such as credit scores, loan amounts, debt-to-income ratios, employment information, and more. This type of dataset is often used for credit risk assessment and predicting the likelihood of loan default.

**Credit Card Fraud Detection Dataset [50]**

The Credit Card Fraud Detection dataset is used for the task of detecting fraudulent credit card transactions. It contains transactions made by credit cards and includes features like transaction amount, time of the transaction, and various anonymized numerical features. The dataset is highly imbalanced, with a small proportion of fraudulent transactions compared to legitimate ones. This dataset is commonly used to develop fraud detection models. A single derived feature is added in order to provide a reasonable shape for the dataset to be converted in image format and is available for use by all other models for consistency. This feature is a simple addition of two credit utilization features (NetFractionRevolvingBurden + NetFractionInstallBurden).

**California Housing Prices Dataset** [51]

The California Housing Prices dataset is often used as a benchmark in regression tasks. It contains features related to housing districts in California, such as median income, housing median age, average rooms, etc. The target variable is typically the median house value for California districts. This dataset is used to build regression models to predict housing prices based on various features. A single derived feature is added in order to provide a reasonable shape for the dataset to be converted in

image format and is available for use by all other models for consistency. This feature is a simple division of the population feature by the average rooms feature.

**Adult Dataset** [52]

The Adult dataset, also known as the "Census Income" dataset, is often used for classification tasks related to income prediction. It includes features like age, education level, marital status, occupation, and more. The goal of this dataset is to predict whether an individual's income exceeds a certain threshold (e.g., $50,000 per year).

**Diabetes 130-US  Dataset [53]**

The dataset comprises a decade's worth of clinical records (1999-2008) from 130 US hospitals and integrated delivery networks. Each entry pertains to the hospital documentation of patients diagnosed with diabetes, encompassing their laboratory results, medication history, and stays of up to 14 days. The primary objective is to ascertain whether patients are readmitted within 30 days of their discharge. This problem holds significant importance for several reasons. Despite substantial evidence demonstrating enhanced clinical outcomes when diabetic patients receive preventive and therapeutic interventions, many patients do not benefit from these measures. One contributing factor is the inconsistent management of diabetes in hospital settings, often overlooking glycemic control. Failing to deliver adequate diabetes care not only inflates healthcare costs due to patient readmissions but also elevates the risk of complications and mortality for these patients, exacerbating the challenges associated with diabetes management.

## 3.3 DATA TRANSFORMATION AND PREPROCESSING

All datasets are preprocessed in a coherent way depending on whether the model is a tree based or deep learning model and on native support for categorical features. If the model natively supports categorical features as input no prior transformation is applied, the models that allow this are CatBoost, GATE, GANDALF, Category Embedding, FT Transformer, TabNet, AutoInt, TabTransformer and NODE.
For MLP, XGBoost, ResNet and S1DCNN the categorical variables are one hot encoded.
The missing values for all datasets are imputed with the median.

For all Neural Network based models features are scaled with the StandardScaler, also known as Z-score normalization. This method scales each feature by subtracting the mean and dividing by the standard deviation of that feature, transforming the data into a distribution with a mean of 0 and a standard deviation of 1.

The datasets are mostly left untouched and fed directly into the machine learning algorithms. Sometimes a few simple derived features are added that are created from the original features, this is typically done in order to be able to reach a reasonable shape for the algorithms that transform the tabular data into an image format, for coherence and consistency all algorithms will be able to utilize the derived features. The specific details of created features are contained in the datasets description section.
No outlier handling technique is utilized.

## 3.4 Hyperparameter Optimization

Hyperparameter optimization in machine learning aims to find the best hyperparameters for a given algorithm to achieve the best performance on a validation set.
Evaluating hyperparameter configurations is computationally expensive, especially with complex models, and we could with confidence say that in the context of tabular data the model's benchmarks are quite complex and computationally demanding. Therefore the technique used for hyperparameter optimization is critical especially considering the computational constraint. Grid and random search are more efficient than manual tuning but still not ideal because they don't use past evaluations to inform future choices.
A branch of optimization methods, Bayesian optimization methods, maintain a probabilistic model of the objective function based on past evaluations. They choose the next hyperparameters to evaluate by selecting those that are likely to yield the best results based on the surrogate model. Bayesian methods aim to become "less wrong" with more data, updating the surrogate model after each evaluation. Sequential Model-Based Optimization (SMBO) formalizes Bayesian optimization and involves running trials sequentially, improving hyperparameter choices with each iteration. SMBO involves a domain of hyperparameters, an objective function to minimize, a surrogate model, a selection function to choose the next hyperparameters, and a history of past evaluations.

The objective function computes a score to minimize, such as MSE or Accuracy or any appropriate scoring function and then a probability model, represented by what is known as a surrogate function is nothing more than a probability representation of the objective function built using past evaluations.
Various forms of surrogate functions exist, such as Gaussian Processes, Random Forest regressions, and Tree-structured Parzen Estimators (TPE).
Bayesian optimization typically looks to model $P(y|x)$ , which is the probability of an objective function value $(y)$, given hyperparameters $(x)$. TPE looks at the function from the opposite side, using Bayes theorem, it looks to model $P(x|y)$, which is the probability of the hyperparameters $(x)$, given the objective function value $(y)$. The selection function, often referred to as "Expected Improvement," guides the choice of the next set of hyperparameters to evaluate based on the surrogate model. It balances exploration (seeking unexplored regions) and exploitation (choosing known promising regions). For the scope of this experimental analysis, we use a TPE. TPE builds a probabilistic model using Bayes' rule and focuses on maximizing the ratio of probabilities under the surrogate function. It draws sample hyperparameters from the distribution that yielded lower scores than a threshold. The algorithm maintains a history of score and hyperparameter pairs, updating the surrogate model based on these records.
This method is theoretically more efficient than random or grid search because it is expected to propose better candidate hyperparameters for evaluation. There is an overhead of selecting the next hyperparameters but this is negligible when compared to the actual computational time of trials. In general a sequential model based optimization strategy is expected to find better hyperparameters in fewer evaluations compared to random search.
To implement this the Hyperopt open-source library is used. Each model is allowed 100 search iterations on each dataset. The actual parameter spaces for each model can be found in Appendix A. For all neural network based models there are a set of common hyperparameters that are worth mentioning. The choice of optimizer and learning rate scheduler are both hyperparameters, as well as the specific parameters associated with the optimizer and scheduler. The choices of optimizers are Adam and AdamW, whereas for schedulers they are a ReduceLROnPlateau, ExponentialLR and StepLR as implemented by torch. In terms of the objective function used to evaluate the parameters MSE is used for regression, Accuracy for multiclass classification and F1 Measure for binary classification with the exception of those datasets where there is a decisive class imbalance, identified by a minority class with a frequency of less than 20%.

Early stopping was used on all models with a criterion set on the validation loss not decreasing for a number of iterations. The number of iterations is set to 5 on all deep learning models and is 30 for CatBoost and XGBoost. This could potentially introduce some positive bias on the tree based models but the values were chosen also considering the computational time. With a more powerful hardware setting, with multiple GPUs it would be fair to test all models with the same early stopping condition. However, extensive trials were performed to determine a reasonable value, considering the tradeoff between a potential loss in accuracy and compute time. The value was decided by setting the parameter as a hyperparameter in an initial exploratory search. The choice was then dictated by common sense considering the aforementioned tradeoff and for all neural network based models the early stopping criterion is set to 5 epochs. For the tree based models where each iteration is typically less demanding the value is higher, 30 iterations.

**Loss Functions**

Different loss functions are selected based on model and the problem type. ResNet and S1DCNN architectures use Binary Cross Entropy for binary classification and Cross Entropy for multiclass. GATE, GANDALF, AutoInt, TabNet, TabTransformer, NODE, Category Embedding and FT Transformer use Cross Entropy for both binary and multiclass classification. This choice has no inherent meaning other than simplicity of implementation and native support of the open-source libraries used to implement the architectures. All neural network based models use MSE for regression. For all Cross Entropy based losses a parameter of class weights is passed and will serve as a weighting factor, useful when dealing with unbalanced training datasets, the individual losses will be scaled with respect to their prevalence in the dataset, in this way the model learns to account for class imbalance and avoid bias towards the majority classes. Then standard objective functions are used when training XGboost and CatBoost.

## 3.5 Evaluation Strategy

In terms of evaluating the performance of the models a KFold strategy with 5 folds was used, for binary and multiclass problems the strategy was stratified. The average and standard deviation of results on the validation folds are reported.

To evaluate the models there is not a single metric of choice, instead the aim is to evaluate performance on a larger set of evaluation metrics in order to have a more robust and complete point of view. Furthermore each dataset and problem task will be logically evaluated based on it's inherent characteristics. For highly unbalanced tasks such as credit fraud detection the metrics that are more appropriate will be used, for example the Lift score and F1 Measure, since relying on a simple accuracy or AUC metric might be deceiving.

The metrics considered for evaluation for regression are Mean Squared Error, Root Mean Squared Error and R2 Score.

**Mean Squared Error (MSE)**

The Mean Squared Error measures the average squared difference between the predicted values and the actual target values. It gives higher weight to larger errors.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Where:

- $(n)$ is the number of data points.
- $(y_i)$ is the actual target value for the \(i\)-th data point.
- $(\hat{y}_i)$ is the predicted value for the \(i\)-th data point.

**Root Mean Squared Error (RMSE)**

The Root Mean Squared Error is the square root of the Mean Squared Error. It provides a measure of the average magnitude of errors in the same units as the target variable.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

**R-squared (R2) Score (Coefficient of Determination)**

The R-squared score measures the proportion of the variance in the dependent variable (target) that is predictable from the independent variables (predictions). It ranges from 0 to 1, with 1 indicating a perfect fit.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

Where:

- $(n)$ is the number of data points.
- $(y_i)$ is the actual target value for the \(i\)-th data point.
- $(\hat{y}_i)$ is the predicted value for the \(i\)-th data point.
- $(\bar{y})$ is the mean of the actual target values.

For binary classification the metrics that are used are: Accuracy, Precision, Recall, F1, AUC, AUPR and Lift Score.

**Lift**

The lift metric is a valuable performance measure in the context of binary classification, particularly for evaluating models that aim to target a specific subset of the population, such as marketing campaigns or fraud detection. It provides insights into the effectiveness of a model or strategy in identifying the positive cases (e.g., responders, fraud cases) compared to a random or baseline approach.
To calculate the lift, first the prediction probability scores for the minority class are sorted in descending order. Then, a certain percentage (e.g., top 10%) of the data is selected based on these sorted predictions. The lift is the ratio of the actual number of positive cases in this selected subset to the number of positive cases in the filtered dataset.

The formula for lift is:

$$\text{Lift} = \frac{\text{Actual Positives in Subset}}{\text{Expected Positives in Subset}}$$

The lift metric is intriguing and rarely considered when benchmarking classification datasets. It is interesting especially in highly unbalanced problems because it quantifies the improvement achieved by a predictive model or strategy over random selection.
It is particularly useful in a number of real world applications where there is a large number of data points and only a few actually are expected to satisfy a target condition. In such cases aggregated metrics like accuracy which is sensitive to relative class frequencies, but even F1 Measure which is more suited to such cases, fall short. They fall short in those cases where an aggregate evaluation metric on the whole dataset is not that meaningful due to the inherent characteristics of the business or otherwise logical purpose of the predictive task. We can make a number of examples where this is the case.

Lift helps to assess the effectiveness of marketing campaigns. For example, in direct marketing, it measures how much better a model can target potential customers compared to randomly selecting individuals in a specific subset of the population. Imagine a company with millions of customers who is trying to predict which customers are better suited to adhere to a campaign, the rate of acceptance in such cases is known to be extremely low and also the actual capability of delivering the advertised product to customers is limited by the cost of delivering such a campaign. If the company is able to send out 50,000 communications per campaign and the population size is 5,000,000, a lift metric of the first 10% is very suited to determining the performance of the predictive model, as it will give a sense of how the model has performed with respect to a random prediction. This metric will be able to discriminate between models in a much more precise way for all problems of this nature whereas an aggregate metric on the 5 million customers is not as significant. The case can be made also for fraud detection, risk assessment, disease prevention and others. A lift greater than 1 signifies that the model or strategy is effective, while a lift less than 1 indicates that random selection would be just as effective, or even better.

**Accuracy**

Accuracy measures the proportion of correctly predicted instances (both true positives and true negatives) out of the total instances.

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total Instances}}$$

**Precision**

Precision quantifies the number of true positive predictions relative to the total number of positive predictions made by the model. It assesses the model's ability to avoid false positives.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

**Recall (Sensitivity, True Positive Rate)**

Recall measures the proportion of true positive predictions relative to the actual total positive instances. It assesses the model's ability to identify all positive cases.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

**F1 Score**

The F1 score is the harmonic mean of precision and recall. It balances the trade-off between precision and recall, providing a single metric that considers both false positives and false negatives.

$$\text{F1 Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

**AUC (Area Under the ROC Curve)**

The AUC measures the area under the receiver operating characteristic (ROC) curve, which plots the true positive rate (recall) against the false positive rate at various thresholds. It provides an overall measure of the model's ability to discriminate between positive and negative cases.

**AUPR (Area Under the Precision-Recall Curve)**

The AUPR measures the area under the precision-recall curve, which plots precision against recall at various thresholds. It assesses the model's performance when the focus is on positive cases.

Having considered these metrics we will proceed to evaluate models on all of them in order to have a much more complete view of the model's performance than if we were using a single metric.

# 4 Results

## 4.1 Performance with Default Hyperparameters

In a first step, all models where trained with default hyperparameters, By using default hyperparameters, we create a benchmark that reflects the "out-of-the-box" performance of each model. This allows us to assess the models in their initial configurations, which is a common starting point for many practitioners. It also gives an insight into the practical facility of using each model.
In the tables that follow the results for each dataset are ordered with respect to the evaluation metric that was optimized in the hyperparameter search. However we will also underline models which won based on other evaluation criteria.

## Titanic

This dataset serves primarily for a sanity check of the model's ability to learn but also as an insight on model performance on very small datasets. For the titanic dataset the hyperparameter optimization was based on the roc_auc metric, with the default hyperparameters CatBoost is the winner in terms of roc_auc lif and area_under_pr. XGBoost wins in terms of accuracy and f1 score. Interestingly ResNet 18 is the most precise model and the Category Embedding model the one with the highest accuracy. In this scenario it would be reasonable to conclude that the tree based models have slightly outperformed the most competitive deep learning models.

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| catboost | 83.61 ± (2.64) | 88.11 ± (1.78) | 25.45 ± (0.9) | 76.91 ± (4.1) | 71.34 ± (5.09) | 83.69 ± (4.76) | 85.71 ± (1.63) |
| tabtransformer | 80.02 ± (1.51) | 84.3 ± (1.45) | 25.44 ± (0.76) | 74.05 ± (1.95) | 74.26 ± (2.06) | 73.87 ± (2.49) | 82.93 ± (2.06) |
| autoint | 79.12 ± (1.93) | 86.17 ± (1.62) | 25.14 ± (0.86) | 74.04 ± (2.83) | 77.75 ± (4.2) | 70.72 ± (2.08) | 83.0 ± (1.48) |
| gandalf | 81.81 ± (3.46) | 86.88 ± (2.39) | 25.14 ± (0.86) | 76.34 ± (5.03) | 76.86 ± (6.5) | 75.91 ± (3.99) | 84.79 ± (2.16) |
| xgb | 83.84 ± (1.66) | 87.43 ± (2.04) | 24.83 ± (1.19) | 77.79 ± (2.73) | 73.97 ± (4.03) | 82.11 ± (1.61) | 84.73 ± (2.94) |
| resnet | 79.13 ± (2.93) | 84.18 ± (2.82) | 24.53 ± (2.47) | 67.43 ± (5.38) | 56.77 ± (6.88) | 83.99 ± (5.78) | 80.86 ± (5.03) |
| mlp | 79.8 ± (1.38) | 83.45 ± (1.24) | 24.52 ± (1.43) | 71.11 ± (2.31) | 64.91 ± (3.49) | 78.76 ± (1.8) | 80.0 ± (1.14) |
| s1dcnn | 80.92 ± (2.38) | 83.78 ± (2.73) | 22.98 ± (2.11) | 72.4 ± (3.43) | 65.2 ± (3.35) | 81.55 ± (4.99) | 78.1 ± (5.1) |
| fttransformer | 77.77 ± (8.58) | 78.89 ± (16.91) | 22.05 ± (7.21) | 61.15 ± (30.85) | 61.59 ± (31.1) | 60.83 ± (30.8) | 75.53 ± (19.52) |
| gate | 73.16 ± (7.74) | 78.0 ± (10.72) | 21.75 ± (4.11) | 54.8 ± (28.39) | 54.62 ± (28.6) | 55.13 ± (28.43) | 72.43 ± (13.38) |
| categoryembedding | 71.87 ± (16.72) | 78.59 ± (14.34) | 21.48 ± (7.72) | 71.05 ± (8.03) | 81.82 ± (9.92) | 65.94 ± (13.82) | 74.66 ± (18.18) |
| tabnet | 73.29 ± (7.18) | 78.04 ± (10.78) | 20.53 ± (4.87) | 52.91 ± (26.46) | 48.84 ± (26.94) | 65.15 ± (16.41) | 70.68 ± (13.75) |

## Iris

This dataset also serves as a sanity check and a baseline comparison of the models in a multiclass classification setting, where the models were optimized with respect to accuracy. Particularly for this dataset with only 4 features, we could expect the more complicated deep learning models to struggle as many of their default configurations are vastly overparameterized for this task. Interestingly AutoInt and the FT Transformer models are the clear winners with default parameters.

| model | accuracy | f1 |
|---|---|---|
| autoint | 98.0 ± (2.67) | 98.0 ± (2.67) |
| fttransformer | 98.0 ± (2.67) | 97.98 ± (2.69) |
| catboost | 96.0 ± (2.49) | 95.98 ± (2.51) |
| tabtransformer | 96.0 ± (4.9) | 95.99 ± (4.91) |
| xgb | 96.0 ± (2.49) | 95.98 ± (2.51) |
| gandalf | 94.67 ± (5.42) | 94.63 ± (5.46) |
| s1dcnn | 93.33 ± (4.71) | 93.3 ± (4.7) |
| gate | 91.33 ± (12.58) | 89.11 ± (16.96) |
| resnet | 91.33 ± (8.06) | 91.27 ± (8.08) |
| categoryembedding | 90.67 ± (10.62) | 90.14 ± (11.48) |
| tabnet | 70.67 ± (8.27) | 68.74 ± (8.72) |
| mlp | 66.67 ± (8.16) | 64.08 ± (9.61) |

## Breast Cancer

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| gate | 97.89 ± (0.89) | 99.61 ± (0.59) | 15.94 ± (0.1) | 98.34 ± (0.69) | 98.88 ± (1.04) | 97.83 ± (1.84) | 99.73 ± (0.42) |
| gandalf | 97.54 ± (1.29) | 99.54 ± (0.7) | 15.94 ± (0.1) | 98.05 ± (1.03) | 98.32 ± (1.38) | 97.8 ± (1.62) | 99.64 ± (0.59) |
| fttransformer | 97.54 ± (0.65) | 99.57 ± (0.61) | 15.94 ± (0.1) | 98.04 ± (0.51) | 98.04 ± (0.67) | 98.04 ± (0.67) | 99.7 ± (0.46) |
| catboost | 97.37 ± (0.78) | 99.35 ± (0.67) | 15.94 ± (0.1) | 97.91 ± (0.61) | 98.32 ± (1.05) | 97.53 ± (1.58) | 99.49 ± (0.58) |
| tabtransformer | 96.49 ± (1.36) | 99.53 ± (0.43) | 15.94 ± (0.1) | 97.2 ± (1.08) | 97.21 ± (1.97) | 97.25 ± (1.91) | 99.7 ± (0.28) |
| xgb | 96.13 ± (2.12) | 99.35 ± (0.54) | 15.94 ± (0.1) | 96.97 ± (1.62) | 98.04 ± (1.42) | 96.0 ± (3.21) | 99.58 ± (0.37) |
| resnet | 94.9 ± (1.72) | 99.35 ± (0.94) | 15.94 ± (0.1) | 96.0 ± (1.22) | 96.65 ± (2.87) | 95.59 ± (4.19) | 99.55 ± (0.68) |
| mlp | 93.67 ± (3.11) | 98.35 ± (0.86) | 15.94 ± (0.1) | 94.86 ± (2.68) | 94.13 ± (5.07) | 95.75 ± (1.15) | 98.92 ± (0.53) |
| s1dcnn | 80.85 ± (12.39) | 99.09 ± (0.48) | 15.94 ± (0.1) | 86.68 ± (7.28) | 93.82 ± (9.28) | 83.17 ± (15.0) | 99.36 ± (0.39) |
| categoryembedding | 86.31 ± (22.15) | 95.94 ± (7.28) | 15.06 ± (1.7) | 81.83 ± (32.19) | 80.3 ± (35.24) | 93.91 ± (8.27) | 96.74 ± (5.93) |
| autoint | 84.38 ± (23.43) | 99.25 ± (0.71) | 15.94 ± (0.1) | 77.51 ± (38.79) | 78.04 ± (39.04) | 77.01 ± (38.58) | 99.51 ± (0.48) |
| tabnet | 68.72 ± (19.17) | 79.08 ± (17.55) | 12.19 ± (4.48) | 69.27 ± (21.99) | 63.08 ± (24.98) | 79.66 ± (15.04) | 82.82 ± (14.79) |

In the Breast Cancer dataset the choice of metric to optimize was the f1 score. It can be argued that this is for this particular task which involves disease prediction one of the most meaningful metrics since we are interested in both high precision and high recall. In any case the deep learning model GATE is the clear winner. In this case all models that share the same lift managed to precisely identify all true positive cases in the first 10% of scores.

## Age Conditions

In the Age Conditions dataset the setting is very similar to the Breast Cancer dataset and so the choice of metric to optimize was the f1 score. The S1DCNN was best in terms of overall accuracy, f1 and recall and CatBoost came in at a close second being more precise overall and also in the top 10% of scores as measured by the lift metric.

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| s1dcnn | 94.73 ± (1.06) | 96.06 ± (1.34) | 50.0 ± (5.97) | 85.28 ± (2.92) | 87.05 ± (3.72) | 83.69 ± (3.78) | 83.48 ± (5.94) |
| catboost | 94.16 ± (2.09) | 96.7 ± (2.45) | 52.35 ± (4.96) | 81.62 ± (6.4) | 74.07 ± (6.3) | 91.4 ± (9.12) | 88.86 ± (8.08) |
| xgb | 91.57 ± (2.17) | 95.72 ± (3.2) | 52.35 ± (2.36) | 73.29 ± (5.61) | 65.71 ± (6.41) | 84.68 ± (11.5) | 87.76 ± (6.28) |
| gandalf | 88.5 ± (5.43) | 94.74 ± (3.72) | 47.67 ± (5.78) | 72.83 ± (9.81) | 84.29 ± (8.57) | 65.37 ± (12.91) | 82.56 ± (7.83) |
| gate | 86.88 ± (4.0) | 91.2 ± (3.25) | 46.61 ± (5.06) | 68.35 ± (5.46) | 78.7 ± (2.24) | 61.06 ± (8.29) | 73.12 ± (7.75) |
| autoint | 87.85 ± (1.77) | 86.26 ± (7.17) | 46.62 ± (4.19) | 63.77 ± (13.78) | 68.79 ± (23.8) | 70.7 ± (15.71) | 72.65 ± (9.28) |
| tabtransformer | 78.58 ± (16.16) | 82.01 ± (16.05) | 38.92 ± (12.43) | 59.07 ± (17.06) | 74.03 ± (13.92) | 51.32 ± (18.21) | 63.57 ± (21.67) |
| categoryembedding | 85.09 ± (4.2) | 78.61 ± (20.41) | 38.85 ± (14.91) | 54.58 ± (24.25) | 60.74 ± (28.05) | 52.61 ± (19.69) | 64.78 ± (25.51) |
| mlp | 87.34 ± (5.13) | 86.45 ± (3.38) | 40.0 ± (9.32) | 53.49 ± (23.98) | 46.23 ± (23.29) | 65.29 ± (21.83) | 64.82 ± (13.73) |
| tabnet | 39.76 ± (22.65) | 57.49 ± (9.9) | 15.21 ± (6.88) | 29.53 ± (3.97) | 70.69 ± (28.52) | 22.41 ± (10.32) | 25.91 ± (7.39) |
| fttransformer | 82.98 ± (4.17) | 69.96 ± (27.78) | 31.26 ± (15.32) | 27.23 ± (34.0) | 38.1 ± (46.75) | 21.39 ± (27.14) | 50.42 ± (27.41) |
| resnet | 83.47 ± (1.32) | 77.74 ± (7.07) | 38.02 ± (7.13) | 15.16 ± (13.51) | 9.26 ± (8.33) | 42.0 ± (36.0) | 55.87 ± (12.52) |

## Adult

| model | accuracy | roc_auc | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|
| catboost | 87.42 ± (0.3) | 92.97 ± (0.12) | 71.48 ± (0.63) | 65.45 ± (0.82) | 78.75 ± (1.07) | 83.21 ± (0.36) |
| xgb | 87.33 ± (0.31) | 92.85 ± (0.17) | 71.35 ± (0.75) | 65.5 ± (0.96) | 78.36 ± (0.85) | 83.0 ± (0.47) |
| categoryembedding | 80.18 ± (1.77) | 90.77 ± (0.37) | 67.82 ± (1.41) | 86.49 ± (2.41) | 55.9 ± (2.78) | 76.2 ± (1.24) |
| mlp | 85.19 ± (0.22) | 90.74 ± (0.34) | 66.47 ± (0.57) | 60.97 ± (1.48) | 73.13 ± (1.24) | 76.79 ± (1.0) |
| gandalf | 80.49 ± (1.5) | 90.73 ± (0.41) | 67.84 ± (0.8) | 85.41 ± (5.05) | 56.55 ± (2.99) | 76.5 ± (1.0) |
| resnet | 82.98 ± (0.79) | 90.45 ± (0.38) | 49.41 ± (4.65) | 34.82 ± (5.05) | 86.75 ± (2.29) | 76.61 ± (0.94) |
| tabtransformer | 78.56 ± (3.1) | 88.79 ± (3.45) | 65.59 ± (4.1) | 84.52 ± (3.86) | 53.63 ± (4.03) | 71.95 ± (7.88) |
| s1dcnn | 81.73 ± (1.1) | 87.4 ± (1.25) | 66.6 ± (0.72) | 75.63 ± (3.64) | 59.72 ± (2.64) | 66.9 ± (1.08) |
| gate | 79.2 ± (2.55) | 87.31 ± (5.35) | 62.12 ± (9.05) | 75.74 ± (21.79) | 57.19 ± (6.56) | 70.59 ± (8.35) |
| fttransformer | 58.83 ± (28.38) | 82.43 ± (14.69) | 57.06 ± (14.91) | 90.4 ± (8.07) | 45.02 ± (17.18) | 65.09 ± (19.63) |
| tabnet | 77.75 ± (4.62) | 79.65 ± (15.88) | 52.59 ± (24.89) | 65.12 ± (31.68) | 47.85 ± (13.85) | 58.87 ± (21.85) |
| autoint | 77.18 ± (2.35) | 58.24 ± (16.11) | 14.2 ± (26.98) | 16.39 ± (32.05) | 31.81 ± (41.06) | 34.45 ± (20.17) |

In the Adult dataset the optimization was performed with respect to the roc_auc metric. CatBoost is a clear winner and XGBoost comes in at a close second. The Deep Learning models seem to be significantly worse at this task.

## Housing

| model | mse | r2_score | rmse |
|---|---|---|---|
| catboost | 0.21 ± (0.01) | 84.54 ± (0.85) | 0.45 ± (0.01) |
| xgb | 0.21 ± (0.01) | 83.96 ± (0.63) | 0.46 ± (0.01) |
| mlp | 0.3 ± (0.02) | 77.65 ± (1.12) | 0.55 ± (0.02) |
| s1dcnn | 0.3 ± (0.01) | 77.41 ± (0.66) | 0.55 ± (0.01) |
| resnet | 0.31 ± (0.01) | 76.95 ± (0.73) | 0.55 ± (0.01) |
| gandalf | 0.34 ± (0.02) | 74.44 ± (1.95) | 0.58 ± (0.02) |
| categoryembedding | 0.35 ± (0.01) | 73.55 ± (1.0) | 0.59 ± (0.01) |
| gate | 0.39 ± (0.03) | 71.07 ± (1.99) | 0.62 ± (0.02) |
| fttransformer | 0.39 ± (0.02) | 70.69 ± (1.71) | 0.62 ± (0.02) |
| autoint | 0.41 ± (0.02) | 69.53 ± (1.14) | 0.64 ± (0.01) |
| tabnet | 0.42 ± (0.03) | 68.49 ± (2.03) | 0.65 ± (0.02) |
| tabtransformer | 0.54 ± (0.03) | 59.2 ± (1.84) | 0.74 ± (0.02) |

Perhaps the most obvious and clear performance gap is displayed in the regression task on the housing dataset. CatBoost and XGBoost outperform the best Neural Network based model, the simple MLP, by nearly 30% in mse and rmse and by about 10% in terms of r2_score.

## Diabetes

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| xgb | 88.86 ± (0.04) | 67.86 ± (0.47) | 24.71 ± (0.67) | 3.19 ± (0.87) | 1.65 ± (0.46) | 52.09 ± (5.33) | 23.08 ± (0.62) |
| catboost | 88.86 ± (0.03) | 68.11 ± (0.59) | 24.54 ± (0.86) | 2.52 ± (1.01) | 1.29 ± (0.53) | 52.03 ± (8.94) | 23.42 ± (0.93) |
| gate | 62.4 ± (3.35) | 67.8 ± (0.58) | 24.43 ± (0.4) | 27.12 ± (0.69) | 62.51 ± (3.56) | 17.35 ± (0.84) | 22.46 ± (1.02) |
| s1dcnn | 79.25 ± (0.93) | 65.33 ± (0.95) | 22.97 ± (0.86) | 26.36 ± (0.61) | 33.28 ± (1.54) | 21.87 ± (0.81) | 19.6 ± (0.73) |
| gandalf | 66.2 ± (5.88) | 65.93 ± (3.26) | 22.82 ± (2.5) | 26.14 ± (2.28) | 54.31 ± (12.83) | 17.51 ± (1.27) | 20.87 ± (2.28) |
| mlp | 88.77 ± (0.07) | 65.22 ± (1.2) | 22.68 ± (0.93) | 3.77 ± (0.66) | 1.97 ± (0.35) | 43.65 ± (5.89) | 20.54 ± (1.08) |
| tabtransformer | 65.64 ± (4.8) | 65.94 ± (2.31) | 22.45 ± (2.59) | 26.37 ± (1.31) | 55.22 ± (8.17) | 17.48 ± (1.04) | 20.68 ± (2.01) |
| resnet | 88.84 ± (0.0) | 65.2 ± (0.86) | 22.26 ± (1.15) | 0.0 ± (0.0) | 0.0 ± (0.0) | 0.0 ± (0.0) | 20.56 ± (1.13) |
| fttransformer | 53.16 ± (21.14) | 63.96 ± (6.99) | 21.08 ± (5.31) | 25.7 ± (2.84) | 68.35 ± (16.11) | 16.23 ± (2.6) | 19.83 ± (4.39) |
| tabnet | 68.55 ± (8.29) | 62.47 ± (2.6) | 20.86 ± (1.97) | 24.74 ± (1.18) | 46.64 ± (13.73) | 17.84 ± (2.74) | 18.02 ± (1.76) |
| categoryembedding | 53.23 ± (21.1) | 63.09 ± (6.59) | 20.17 ± (5.41) | 25.39 ± (2.71) | 67.32 ± (16.48) | 16.05 ± (2.5) | 19.14 ± (4.15) |
| autoint | 72.67 ± (10.06) | 62.95 ± (6.78) | 19.96 ± (5.39) | 21.05 ± (10.65) | 40.97 ± (24.02) | 14.75 ± (7.46) | 18.78 ± (3.96) |

The Diabetes dataset is particularly interesting because it is quite a complex task, with a high number of categorical features. In terms of optimization metric here it Lift was the metric used. The reasoning is that this dataset contains information on patients that were readmitted to the hospital in a short time frame after their first admission. It would be logical to think that in a real world scenario this model could be used to potentially get in contact with patients who are deemed to be at higher risk of readmission but also there could be a constraint on the number of patients who can be contacted. XGBoost and CatBoost are the most precise models overall and also in the top 10% of predictions, so they are the winners here.

## Creditcard

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| catboost | 99.96 ± (0.01) | 98.54 ± (0.85) | 96.34 ± (1.52) | 86.94 ± (2.26) | 79.88 ± (2.39) | 95.4 ± (2.59) | 85.92 ± (2.32) |
| xgb | 99.96 ± (0.01) | 97.91 ± (0.68) | 94.11 ± (1.62) | 86.49 ± (2.12) | 79.27 ± (1.64) | 95.19 ± (3.26) | 85.6 ± (2.22) |
| mlp | 99.95 ± (0.01) | 96.99 ± (0.69) | 93.91 ± (1.7) | 82.97 ± (1.77) | 77.23 ± (2.33) | 89.77 ± (3.38) | 82.59 ± (2.81) |
| s1dcnn | 97.44 ± (1.31) | 94.2 ± (1.78) | 92.48 ± (2.3) | 13.35 ± (6.12) | 89.01 ± (3.15) | 7.37 ± (3.71) | 11.42 ± (4.96) |
| resnet | 99.83 ± (0.0) | 96.17 ± (0.99) | 91.05 ± (2.39) | 0.0 ± (0.0) | 0.0 ± (0.0) | 0.0 ± (0.0) | 69.7 ± (6.45) |
| gate | 99.15 ± (0.54) | 89.9 ± (13.2) | 80.43 ± (18.89) | 20.35 ± (13.09) | 67.6 ± (34.38) | 12.27 ± (8.4) | 52.68 ± (25.71) |
| gandalf | 95.64 ± (4.34) | 80.9 ± (34.28) | 75.73 ± (36.87) | 10.23 ± (5.48) | 72.66 ± (33.34) | 5.52 ± (3.01) | 58.58 ± (29.28) |
| fttransformer | 98.91 ± (1.08) | 67.18 ± (37.3) | 57.38 ± (44.77) | 15.34 ± (19.47) | 52.71 ± (43.1) | 10.22 ± (14.3) | 39.66 ± (32.48) |
| autoint | 92.35 ± (5.45) | 78.67 ± (13.76) | 47.98 ± (28.64) | 7.48 ± (8.18) | 54.51 ± (31.32) | 4.18 ± (4.66) | 15.25 ± (27.34) |
| categoryembedding | 91.45 ± (14.57) | 64.0 ± (17.45) | 35.11 ± (31.38) | 1.25 ± (2.15) | 32.08 ± (39.29) | 0.64 ± (1.11) | 11.82 ± (21.16) |
| tabnet | 59.65 ± (0.6) | 40.78 ± (7.47) | 30.11 ± (7.07) | 0.32 ± (0.06) | 37.63 ± (7.31) | 0.16 ± (0.03) | 4.65 ± (2.04) |
| tabtransformer | 46.1 ± (0.3) | 13.81 ± (1.6) | 6.3 ± (1.96) | 0.09 ± (0.01) | 14.43 ± (1.51) | 0.05 ± (0.0) | 0.13 ± (0.03) |

The creditcard is another highly unbalanced dataset where looking at just accuracy and roc_auc can be highly misleading. Although with default hyperparameters the tree based models are the clear winners in most metrics.
A note must be made on the ResNet model and a few of the deep learning models that really struggled with the high class imbalance and therefore predicted very few positive cases. This can be quite easily dealt with by setting a different prediction probability threshold  and does not necessarily invalidate the models as we can see explained by the lift metric, most of them were able to learn something about the function and concentrate the positive target observations in the first top 10% of predictions. This is likely how a credit risk model is applied in a real world scenario. An exception is indeed the TabTransformer network which is the only one who

exhibited worse performance than a random prediction in the top 10% of its positive class scores.

## Heloc

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| catboost | 72.31 ± (0.75) | 79.8 ± (0.65) | 17.2 ± (0.29) | 74.36 ± (0.49) | 76.9 ± (0.53) | 71.99 ± (1.07) | 79.53 ± (0.9) |
| fttransformer | 72.12 ± (0.68) | 79.17 ± (0.25) | 17.12 ± (0.23) | 73.64 ± (0.45) | 74.59 ± (1.0) | 72.73 ± (1.16) | 79.08 ± (0.59) |
| mlp | 71.79 ± (0.65) | 78.87 ± (0.67) | 16.96 ± (0.3) | 73.7 ± (0.51) | 75.75 ± (0.89) | 71.78 ± (0.92) | 78.65 ± (0.83) |
| gandalf | 72.1 ± (0.38) | 79.29 ± (0.49) | 16.94 ± (0.31) | 73.6 ± (0.99) | 74.63 ± (3.29) | 72.75 ± (1.43) | 78.96 ± (0.95) |
| gate | 72.13 ± (0.76) | 79.24 ± (0.46) | 16.9 ± (0.46) | 74.13 ± (0.58) | 76.48 ± (0.36) | 71.92 ± (0.89) | 79.0 ± (0.73) |
| xgb | 72.11 ± (0.85) | 78.97 ± (0.89) | 16.89 ± (0.15) | 74.34 ± (0.65) | 77.41 ± (0.95) | 71.53 ± (1.12) | 78.63 ± (0.77) |
| categoryembedding | 71.78 ± (0.63) | 78.6 ± (0.45) | 16.89 ± (0.34) | 72.93 ± (1.1) | 72.93 ± (2.53) | 72.99 ± (0.47) | 78.5 ± (0.54) |
| tabtransformer | 71.41 ± (0.59) | 77.89 ± (0.63) | 16.87 ± (0.34) | 72.42 ± (1.2) | 72.01 ± (2.95) | 72.93 ± (0.86) | 77.54 ± (0.72) |
| resnet | 71.27 ± (0.69) | 78.38 ± (0.54) | 16.78 ± (0.37) | 73.89 ± (0.54) | 77.91 ± (1.9) | 70.32 ± (1.29) | 78.14 ± (0.62) |
| s1dcnn | 69.55 ± (0.84) | 78.18 ± (0.72) | 16.56 ± (0.41) | 66.65 ± (1.97) | 58.45 ± (3.48) | 77.75 ± (1.04) | 77.65 ± (1.07) |
| tabnet | 70.73 ± (1.22) | 77.79 ± (1.23) | 16.3 ± (0.27) | 71.08 ± (0.93) | 68.9 ± (1.7) | 73.49 ± (2.13) | 77.27 ± (1.32) |
| autoint | 67.17 ± (8.35) | 77.81 ± (1.94) | 16.02 ± (1.84) | 62.81 ± (21.41) | 63.51 ± (26.21) | 69.68 ± (3.35) | 76.71 ± (3.08) |

The logical task used here is similar to the Credit Card dataset, we are in a binary classification setting aimed at assessing credit risk. However the dataset is extremely well balanced with the positive class ratio at 52%, which makes this task quite a bit easier and allows us to use accuracy as the most suitable evaluation metric. CatBoost once again wins but the margin is definitely not as large with the FT Transformer coming in close second place.. MLP is worth mentioning coming in third place in both credit risk tasks.

## Covertype

| model | accuracy | f1 |
|---|---|---|
| resnet | 92.06 ± (0.78) | 92.26 ± (0.75) |
| catboost | 89.25 ± (0.08) | 89.19 ± (0.08) |
| gandalf | 87.9 ± (1.75) | 88.24 ± (1.66) |
| s1dcnn | 87.4 ± (5.11) | 87.84 ± (4.78) |
| xgb | 86.9 ± (0.04) | 86.84 ± (0.05) |
| mlp | 85.81 ± (0.31) | 85.64 ± (0.33) |
| categoryembedding | 74.2 ± (5.35) | 75.46 ± (4.87) |
| gate | 68.0 ± (3.49) | 69.79 ± (3.06) |
| tabtransformer | 59.21 ± (2.83) | 62.22 ± (2.2) |
| tabnet | 59.11 ± (11.99) | 57.34 ± (19.25) |
| fttransformer | 46.69 ± (28.71) | 39.68 ± (33.39) |
| autoint | 24.27 ± (23.67) | 18.95 ± (24.29) |

In the largest dataset used in this study, the clear winner with default hyperparameters is ResNet18. It is also evident that some of the deep learning models achieve at best mediocre results on this task with FT-Transformer and AutoInt failing largely at correctly classifying the target.

### Insights from Default Hyperparameters

Out of the 10 datasets tested, tree based models won in 6 cases. In the category of very small datasets, with less than 5,000 observations, the Deep Learning based models perform better and win 3 out of 4 times. In the medium dataset regime (5,000 to 100,000 observations), the tree based models win in all 3 datasets. In the medium-large dataset regime (100,000 to 1,000,000) observations tree based models win 2 out of 3 times.
Looking at the results from a task perspective, in the only regression task analyzed the tree base models win by a very large margin. In binary classification tasks overall the tree based models take the first place 5 times out of 7. If we focus on the tasks exhibiting high class imbalance (defined here by the positive class ratio being less than 20%), tree based models win in 2 out of the 3 datasets.
In the multiclass classification realm, perhaps the least explored in current literature, Deep Learning based models win in both of the datasets.

## 4.2 Performance with Hyperparameter Optimization

## Titanic

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| xgb | 83.16 ± (2.57) | 88.65 ± (2.70) | 25.45 ± (1.35) | 76.54 ± (3.70) | 71.64 ± (4.29) | 82.31 ± (4.49) | 85.80 ± (3.16) |
| catboost | 81.82 ± (2.91) | 88.37 ± (2.12) | 24.83 ± (0.69) | 74.76 ± (4.56) | 70.46 ± (5.47) | 79.70 ± (3.54) | 85.23 ± (2.32) |
| node | 82.16 ± (2.62) | 88.17 ± (2.19) | 25.44 ± (0.82) | 77.04 ± (3.49) | 78.06 ± (4.60) | 76.26 ± (4.52) | 85.50 ± (2.07) |
| fttransformer | 81.70 ± (2.02) | 87.88 ± (2.45) | 25.44 ± (0.82) | 76.06 ± (3.13) | 76.00 ± (4.90) | 76.30 ± (2.78) | 85.67 ± (2.40) |
| categoryembedding | 81.14 ± (1.82) | 87.84 ± (2.70) | 25.75 ± (0.68) | 75.67 ± (2.66) | 76.59 ± (4.14) | 74.89 ± (2.58) | 85.84 ± (2.27) |
| gandalf | 80.92 ± (2.53) | 87.69 ± (2.24) | 24.53 ± (1.08) | 75.45 ± (3.59) | 76.58 ± (4.66) | 74.42 ± (3.18) | 84.86 ± (2.35) |
| tabnet | 79.23 ± (1.69) | 86.87 ± (2.23) | 25.45 ± (0.90) | 72.96 ± (2.06) | 73.38 ± (8.36) | 73.90 ± (6.64) | 83.88 ± (1.30) |
| gate | 81.14 ± (2.01) | 86.82 ± (2.77) | 24.83 ± (0.69) | 75.57 ± (3.18) | 76.29 ± (4.62) | 74.93 ± (1.88) | 84.00 ± (3.04) |
| autoint | 79.68 ± (1.36) | 86.72 ± (1.94) | 25.44 ± (0.82) | 74.76 ± (1.40) | 78.35 ± (2.44) | 71.60 ± (2.75) | 84.36 ± (2.26) |
| resnet | 80.92 ± (2.05) | 86.16 ± (1.57) | 25.44 ± (0.76) | 71.33 ± (3.98) | 62.30 ± (6.29) | 84.44 ± (5.43) | 83.83 ± (1.87) |
| s1dcnn | 81.37 ± (3.10) | 85.95 ± (2.28) | 24.83 ± (0.69) | 73.47 ± (4.45) | 67.25 ± (4.79) | 81.24 ± (6.02) | 82.97 ± (2.55) |
| mlp | 82.38 ± (2.82) | 85.90 ± (0.44) | 25.13 ± (0.71) | 75.25 ± (3.35) | 69.59 ± (3.71) | 82.45 ± (7.10) | 84.13 ± (2.05) |
| tabtransformer | 78.79 ± (2.01) | 85.56 ± (1.56) | 25.14 ± (1.34) | 71.62 ± (3.42) | 70.14 ± (6.45) | 73.61 ± (3.54) | 82.69 ± (1.34) |

The titanic dataset results do not vary greatly as a result of hyperparameter optimization, all models do gain some predictive quality. Assigning a winner here is not very interesting but also not exactly obvious. In terms of accuracy XGB takes the first place but there are 2 Deep Learning models which are superior in terms of f1 and area_under_pr, which are perhaps equally if not more relevant evaluation metrics in this setting.

## Iris

The situation here remains mostly unchanged with the AutoInt model maintaining its first place. XGB and CatBoost are 10th and last in the scoreboard. However the performance is near perfect in all cases and it is very likely that if the number of hyperparameter search iterations were increased we would see all models achieving perfect accuracy.

| model | accuracy | f1 |
|---|---|---|
| autoint | 98.67 ± (1.63) | 98.66 ± (1.64) |
| tabtransformer | 98.00 ± (4.00) | 97.99 ± (4.01) |
| fttransformer | 98.00 ± (2.67) | 97.98 ± (2.69) |
| gandalf | 98.00 ± (2.67) | 98.00 ± (2.67) |
| node | 98.00 ± (2.67) | 98.00 ± (2.67) |
| resnet | 97.33 ± (3.89) | 97.33 ± (3.90) |
| s1dcnn | 97.33 ± (3.89) | 97.33 ± (3.90) |
| tabnet | 97.33 ± (3.27) | 97.32 ± (3.28) |
| categoryembedding | 97.33 ± (2.49) | 97.33 ± (2.50) |
| xgb | 97.33 ± (2.49) | 97.32 ± (2.52) |
| gate | 96.67 ± (4.22) | 96.66 ± (4.22) |
| mlp | 96.67 ± (4.22) | 96.65 ± (4.23) |
| catboost | 96.00 ± (3.89) | 95.98 ± (3.90) |

## Breast Cancer

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| gate | 98.95 ± (1.02) | 99.62 ± (0.75) | 15.94 ± (0.10) | 99.17 ± (0.81) | 99.72 ± (0.56) | 98.63 ± (1.23) | 99.73 ± (0.55) |
| node | 98.59 ± (1.19) | 99.43 ± (0.63) | 15.91 ± (0.12) | 98.89 ± (0.93) | 99.44 ± (0.68) | 98.36 ± (1.58) | 99.59 ± (0.49) |
| gandalf | 98.59 ± (0.70) | 99.58 ± (0.61) | 15.94 ± (0.10) | 98.89 ± (0.54) | 99.44 ± (0.69) | 98.37 ± (1.57) | 99.71 ± (0.45) |
| categoryembedding | 98.59 ± (0.70) | 99.68 ± (0.37) | 15.94 ± (0.10) | 98.88 ± (0.56) | 99.16 ± (0.69) | 98.63 ± (1.23) | 99.79 ± (0.26) |
| fttransformer | 98.59 ± (0.70) | 99.64 ± (0.60) | 15.94 ± (0.10) | 98.88 ± (0.55) | 99.16 ± (0.68) | 98.61 ± (0.87) | 99.74 ± (0.44) |
| tabtransformer | 98.42 ± (0.66) | 99.67 ± (0.49) | 15.94 ± (0.10) | 98.75 ± (0.51) | 99.44 ± (0.69) | 98.08 ± (1.04) | 99.77 ± (0.35) |
| mlp | 98.07 ± (1.16) | 99.66 ± (0.46) | 15.94 ± (0.10) | 98.47 ± (0.92) | 99.16 ± (1.12) | 97.81 ± (1.38) | 99.77 ± (0.32) |
| s1dcnn | 98.07 ± (0.66) | 99.62 ± (0.48) | 15.94 ± (0.10) | 98.46 ± (0.53) | 98.60 ± (1.54) | 98.37 ± (1.57) | 99.74 ± (0.34) |
| autoint | 98.07 ± (0.65) | 99.47 ± (0.55) | 15.94 ± (0.10) | 98.46 ± (0.52) | 98.60 ± (0.88) | 98.34 ± (1.03) | 99.65 ± (0.39) |
| tabnet | 97.37 ± (1.11) | 99.29 ± (0.95) | 15.94 ± (0.10) | 97.91 ± (0.88) | 98.04 ± (1.42) | 97.80 ± (1.37) | 99.39 ± (0.92) |
| catboost | 97.19 ± (1.40) | 99.53 ± (0.53) | 15.94 ± (0.10) | 97.78 ± (1.09) | 98.60 ± (1.52) | 97.03 ± (2.28) | 99.68 ± (0.38) |
| resnet | 97.02 ± (1.62) | 99.07 ± (0.82) | 15.94 ± (0.10) | 97.64 ± (1.25) | 98.04 ± (1.68) | 97.33 ± (2.78) | 99.06 ± (0.90) |
| xgb | 96.31 ± (1.29) | 99.09 ± (0.66) | 15.94 ± (0.10) | 97.08 ± (0.99) | 97.48 ± (1.06) | 96.73 ± (2.35) | 99.39 ± (0.46) |

GATE is the clear winner being superior in all but roc_auc, losing to Category Embedding. Overall we could confidently say that while all models are nearly perfect on this task, the Deep Learning based algorithms have an advantage.

## Age Conditions

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| node | 91.96 ± (3.68) | 95.59 ± (2.77) | 51.88 ± (4.07) | 79.67 ± (9.33) | 88.83 ± (8.71) | 72.70 ± (11.76) | 85.87 ± (8.27) |
| catboost | 93.57 ± (1.59) | 96.36 ± (2.53) | 54.70 ± (1.72) | 79.60 ± (5.04) | 72.29 ± (5.50) | 88.67 ± (4.96) | 89.15 ± (5.29) |
| xgb | 93.57 ± (1.44) | 96.25 ± (3.04) | 53.76 ± (3.39) | 79.13 ± (5.72) | 71.17 ± (7.94) | 89.88 ± (6.07) | 88.68 ± (8.27) |
| gandalf | 91.16 ± (2.64) | 94.19 ± (1.84) | 50.94 ± (5.41) | 77.06 ± (6.28) | 84.29 ± (3.63) | 71.15 ± (8.28) | 81.43 ± (6.92) |
| s1dcnn | 90.51 ± (2.51) | 92.70 ± (3.21) | 46.12 ± (4.31) | 73.16 ± (6.74) | 74.11 ± (6.06) | 72.28 ± (7.52) | 72.87 ± (9.70) |
| mlp | 91.31 ± (2.37) | 92.37 ± (1.55) | 50.90 ± (6.65) | 72.69 ± (8.30) | 67.71 ± (10.94) | 79.52 ± (6.84) | 80.40 ± (7.67) |
| autoint | 89.40 ± (2.82) | 91.57 ± (3.06) | 46.17 ± (4.75) | 72.51 ± (6.57) | 79.70 ± (5.99) | 66.80 ± (8.10) | 75.25 ± (5.83) |
| categoryembedding | 88.92 ± (3.53) | 91.74 ± (4.10) | 46.20 ± (7.37) | 72.34 ± (6.04) | 81.56 ± (5.62) | 65.79 ± (9.09) | 74.34 ± (9.15) |
| gate | 89.76 ± (4.36) | 92.60 ± (4.76) | 48.25 ± (4.48) | 72.33 ± (10.96) | 76.84 ± (13.05) | 69.38 ± (11.99) | 80.63 ± (8.36) |
| fttransformer | 89.08 ± (3.40) | 91.51 ± (6.42) | 48.10 ± (7.42) | 72.05 ± (8.42) | 80.74 ± (10.75) | 65.45 ± (8.33) | 79.00 ± (10.06) |
| tabtransformer | 87.78 ± (1.17) | 89.51 ± (2.80) | 43.22 ± (4.47) | 70.09 ± (2.60) | 82.42 ± (3.33) | 61.00 ± (2.47) | 69.97 ± (8.05) |
| tabnet | 85.21 ± (5.42) | 87.51 ± (3.60) | 40.41 ± (8.28) | 65.08 ± (9.04) | 75.84 ± (3.97) | 58.26 ± (13.41) | 64.77 ± (9.36) |
| resnet | 86.81 ± (2.14) | 85.27 ± (5.28) | 42.27 ± (5.76) | 53.46 ± (13.98) | 47.32 ± (17.44) | 69.45 ± (9.79) | 64.80 ± (11.48) |

NODE, the model which could not be tested with default hyperparameters due to GPU memory constraints, seems to perform extremely well on these small datasets. In this case it is also not trivial to assign a clear winner between NODE, CatBoost and XGB. It really depends on the metric that we want to prioritize, it could be argued that in the task of identifying age related disease conditions a model needs to focus on both a high recall and a high precision. My personal opinion is that there is no clear winner here.

## Adult

In the run with default hyperparameters a number of Deep Learning models were struggling with this task. We see a large increase in performance subsequent to trying different hyperparameter combinations. However the situation at the top is unchanged here with CatBoost and XGBoost being the clear winners with quite a significant margin.

| model | accuracy | roc_auc | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|
| catboost | 87.50 ± (0.22) | 93.11 ± (0.10) | 71.73 ± (0.42) | 65.83 ± (0.65) | 78.80 ± (0.94) | 83.49 ± (0.31) |
| xgb | 87.36 ± (0.39) | 92.91 ± (0.11) | 71.51 ± (0.89) | 65.90 ± (0.91) | 78.17 ± (0.99) | 83.06 ± (0.41) |
| fttransformer | 81.39 ± (0.28) | 91.71 ± (0.21) | 69.09 ± (0.41) | 86.34 ± (0.76) | 57.59 ± (0.43) | 79.71 ± (0.52) |
| autoint | 81.30 ± (0.70) | 91.67 ± (0.22) | 69.12 ± (0.72) | 86.86 ± (0.97) | 57.41 ± (1.13) | 79.30 ± (0.65) |
| gate | 80.96 ± (0.75) | 91.57 ± (0.23) | 68.76 ± (0.64) | 86.94 ± (1.16) | 56.89 ± (1.28) | 79.29 ± (0.41) |
| node | 81.37 ± (0.46) | 91.57 ± (0.19) | 69.11 ± (0.49) | 86.49 ± (0.76) | 57.55 ± (0.75) | 79.11 ± (0.64) |
| gandalf | 81.24 ± (0.44) | 91.50 ± (0.20) | 68.90 ± (0.60) | 86.28 ± (0.39) | 57.35 ± (0.66) | 78.93 ± (0.57) |
| categoryembedding | 80.83 ± (0.60) | 91.44 ± (0.25) | 68.68 ± (0.55) | 87.28 ± (0.73) | 56.63 ± (0.94) | 78.60 ± (0.79) |
| tabnet | 80.55 ± (1.31) | 91.34 ± (0.23) | 68.27 ± (1.13) | 86.75 ± (1.49) | 56.35 ± (2.13) | 78.42 ± (0.62) |
| mlp | 85.17 ± (0.30) | 90.89 ± (0.27) | 67.20 ± (1.27) | 63.16 ± (2.57) | 71.89 ± (0.82) | 77.27 ± (0.94) |
| tabtransformer | 81.15 ± (0.34) | 90.73 ± (0.23) | 68.50 ± (0.37) | 85.14 ± (1.38) | 57.32 ± (0.64) | 76.56 ± (0.75) |
| resnet | 81.83 ± (1.13) | 90.43 ± (0.35) | 42.56 ± (7.48) | 28.52 ± (6.65) | 88.72 ± (3.64) | 76.16 ± (0.79) |
| s1dcnn | 83.52 ± (0.39) | 90.08 ± (0.26) | 68.79 ± (0.57) | 75.41 ± (0.36) | 63.24 ± (0.80) | 73.67 ± (1.23) |

## Housing

| model | mse | r2_score | rmse |
|---|---|---|---|
| catboost | 0.18 ± (0.01) | 86.43 ± (0.59) | 0.42 ± (0.01) |
| xgb | 0.19 ± (0.01) | 85.95 ± (0.69) | 0.43 ± (0.01) |
| mlp | 0.25 ± (0.01) | 81.45 ± (0.62) | 0.50 ± (0.01) |
| s1dcnn | 0.27 ± (0.01) | 81.01 ± (2.02) | 0.51 ± (0.01) |
| resnet | 0.29 ± (0.02) | 79.34 ± (2.12) | 0.53 ± (0.01) |
| gandalf | 0.31 ± (0.02) | 77.32 ± (2.25) | 0.55 ± (0.01) |
| node | 0.31 ± (0.02) | 76.56 ± (1.46) | 0.56 ± (0.02) |
| gate | 0.34 ± (0.03) | 75.05 ± (3.32) | 0.57 ± (0.03) |
| fttransformer | 0.35 ± (0.02) | 74.56 ± (2.64) | 0.59 ± (0.01) |
| categoryembedding | 0.35 ± (0.01) | 74.28 ± (0.70) | 0.58 ± (0.00) |
| tabtransformer | 0.38 ± (0.03) | 72.08 ± (3.03) | 0.61 ± (0.02) |
| autoint | 0.45 ± (0.01) | 71.64 ± (0.85) | 0.67 ± (0.01) |
| tabnet | 0.40 ± (0.02) | 69.87 ± (1.67) | 0.63 ± (0.02) |

The situation for the only regression dataset in this study also remains very similar to the results achieved with default hyperparameters with XGB and CatBoost

winning by a great margin. Only the simplest regression model, the MLP managed to achieve similar results but is still considerably worse.

## Diabetes

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| catboost | 88.88 ± (0.01) | 68.39 ± (0.54) | 25.13 ± (0.78) | 2.28 ± (0.44) | 1.16 ± (0.23) | 58.43 ± (2.99) | 23.74 ± (0.86) |
| gandalf | 63.34 ± (2.50) | 68.15 ± (0.54) | 25.11 ± (1.00) | 27.40 ± (0.29) | 61.95 ± (3.56) | 17.62 ± (0.47) | 23.21 ± (1.06) |
| xgb | 88.88 ± (0.02) | 68.17 ± (0.55) | 24.96 ± (0.87) | 3.26 ± (0.95) | 1.68 ± (0.51) | 56.28 ± (1.41) | 23.41 ± (0.81) |
| categoryembedding | 63.58 ± (1.28) | 67.89 ± (0.56) | 24.61 ± (0.82) | 27.41 ± (0.36) | 61.61 ± (1.81) | 17.64 ± (0.34) | 22.75 ± (0.83) |
| gate | 66.2 ± (5.88) | 65.93 ± (3.26) | 24.32 ± (2.62) | 26.14 ± (2.28) | 54.31 ± (12.83) | 17.51 ± (1.27) | 20.87 ± (2.28) |
| s1dcnn | 80.71 ± (1.89) | 65.49 ± (1.20) | 24.31 ± (1.17) | 27.60 ± (0.93) | 32.98 ± (3.91) | 24.04 ± (1.90) | 20.05 ± (1.14) |
| node | 63.64 ± (2.80) | 67.79 ± (0.60) | 24.27 ± (0.73) | 27.26 ± (0.69) | 60.97 ± (3.57) | 17.59 ± (0.70) | 21.99 ± (0.90) |
| autoint | 63.56 ± (1.36) | 67.54 ± (0.78) | 23.95 ± (1.03) | 27.14 ± (0.47) | 60.80 ± (2.25) | 17.47 ± (0.36) | 21.77 ± (1.17) |
| tabnet | 63.75 ± (2.53) | 66.60 ± (0.78) | 23.94 ± (0.95) | 26.66 ± (0.46) | 59.07 ± (4.54) | 17.25 ± (0.45) | 21.22 ± (1.20) |
| mlp | 88.83 ± (0.04) | 66.32 ± (0.53) | 23.63 ± (1.07) | 2.87 ± (1.70) | 1.50 ± (0.89) | 40.68 ± (21.16) | 21.76 ± (0.80) |
| fttransformer | 63.44 ± (2.51) | 67.02 ± (1.20) | 23.17 ± (1.18) | 26.99 ± (0.74) | 60.46 ± (3.04) | 17.40 ± (0.73) | 21.56 ± (1.27) |
| resnet | 88.84 ± (0.00) | 65.70 ± (0.98) | 22.84 ± (1.00) | 0.00 ± (0.00) | 0.00 ± (0.00) | 0.00 ± (0.00) | 21.01 ± (1.15) |
| tabtransformer | 64.56 ± (5.96) | 64.53 ± (2.04) | 21.14 ± (1.26) | 25.71 ± (1.40) | 54.68 ± (7.90) | 17.00 ± (1.50) | 20.14 ± (0.82) |

Results remain mostly unvaried with a general increase in performance, tree based models are superior however GANDALF is just behind XGB.

## Creditcard

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| gandalf | 97.00 ± (1.42) | 98.79 ± (0.13) | 96.97 ± (0.82) | 12.55 ± (6.93) | 92.57 ± (0.46) | 6.90 ± (4.11) | 73.59 ± (2.43) |
| categoryembedding | 96.82 ± (1.05) | 98.61 ± (0.61) | 96.95 ± (1.29) | 9.84 ± (2.55) | 92.07 ± (2.38) | 5.22 ± (1.42) | 73.62 ± (4.33) |
| xgb | 99.96 ± (0.01) | 98.55 ± (0.70) | 96.75 ± (2.27) | 85.83 ± (2.26) | 78.87 ± (2.69) | 94.19 ± (2.35) | 85.23 ± (2.54) |
| catboost | 99.96 ± (0.01) | 98.47 ± (0.93) | 96.55 ± (1.64) | 86.66 ± (2.01) | 79.28 ± (2.54) | 95.63 ± (2.47) | 86.26 ± (1.93) |
| node | 98.15 ± (1.04) | 98.32 ± (0.57) | 95.73 ± (1.50) | 16.87 ± (5.41) | 90.04 ± (1.76) | 9.41 ± (3.22) | 72.89 ± (6.37) |
| autoint | 97.22 ± (0.92) | 98.51 ± (0.56) | 95.53 ± (1.89) | 11.09 ± (3.21) | 90.86 ± (3.46) | 5.95 ± (1.85) | 74.39 ± (1.73) |
| fttransformer | 97.25 ± (0.85) | 98.41 ± (0.78) | 95.47 ± (1.26) | 11.22 ± (2.13) | 90.71 ± (1.79) | 5.51 ± (1.85) | 74.11 ± (1.57) |
| gate | 97.18 ± (0.72) | 98.31 ± (0.65) | 95.27 ± (1.83) | 11.02 ± (3.53) | 90.46 ± (2.36) | 5.41 ± (1.85) | 73.59 ± (1.77) |
| s1dcnn | 97.11 ± (0.78) | 98.21 ± (0.78) | 95.14 ± (1.63) | 11.01 ± (2.48) | 90.21 ± (1.68) | 5.23 ± (1.65) | 73.45 ± (1.97) |
| mlp | 99.94 ± (0.01) | 97.81 ± (0.63) | 94.51 ± (1.90) | 81.77 ± (1.73) | 77.85 ± (1.66) | 86.20 ± (3.27) | 79.65 ± (4.09) |
| resnet | 99.83 ± (0.00) | 96.09 ± (0.93) | 91.23 ± (1.87) | 0.00 ± (0.00) | 0.00 ± (0.00) | 0.00 ± (0.00) | 48.46 ± (6.91) |
| tabtransformer | 75.47 ± (26.81) | 93.30 ± (6.34) | 89.81 ± (7.99) | 7.49 ± (6.77) | 89.22 ± (3.47) | 4.05 ± (3.76) | 61.81 ± (6.64) |
| tabnet | 78.45 ± (39.13) | 93.76 ± (4.21) | 87.59 ± (7.45) | 11.18 ± (7.04) | 85.93 ± (10.52) | 6.16 ± (4.03) | 42.52 ± (20.06) |

In the medium-large dataset regime this is perhaps the most interesting result. This is undoubtedly the hardest classification task due to the large class imbalance with a 0.17% positive class rate. The very recent (2022) GANDALF model displays the best lift score, being able to concentrate the highest number of true positives inside the top 10% of predictions ordered by the probability.Another Deep Learning model, one of the simplest also performs very well and this is the Category Embedding model. Tree ensembles are just behind.

## Heloc

| model | accuracy | roc_auc | lift | f1 | recall | precision | area_under_pr |
|---|---|---|---|---|---|---|---|
| fttransformer | 72.59 ± (0.35) | 79.52 ± (0.08) | 16.99 ± (0.19) | 74.45 ± (0.41) | 76.50 ± (1.56) | 72.54 ± (0.88) | 79.15 ± (0.60) |
| xgb | 72.57 ± (0.57) | 79.61 ± (0.62) | 17.25 ± (0.42) | 74.82 ± (0.36) | 78.09 ± (0.83) | 71.83 ± (0.90) | 79.37 ± (0.85) |
| gandalf | 72.53 ± (0.53) | 79.46 ± (0.53) | 17.18 ± (0.31) | 74.36 ± (0.28) | 76.30 ± (0.75) | 72.53 ± (0.92) | 79.22 ± (0.79) |
| autoint | 72.40 ± (0.56) | 79.61 ± (0.48) | 17.22 ± (0.31) | 73.77 ± (0.65) | 74.37 ± (1.46) | 73.20 ± (0.79) | 79.39 ± (0.81) |
| catboost | 72.28 ± (0.75) | 79.81 ± (0.62) | 17.38 ± (0.21) | 74.46 ± (0.48) | 77.40 ± (0.66) | 71.75 ± (1.08) | 79.62 ± (0.81) |
| mlp | 72.11 ± (0.85) | 78.96 ± (0.57) | 17.01 ± (0.22) | 74.20 ± (0.98) | 76.88 ± (2.58) | 71.77 ± (1.27) | 78.59 ± (0.78) |
| tabnet | 72.11 ± (0.55) | 79.36 ± (0.32) | 17.03 ± (0.22) | 73.54 ± (1.31) | 74.41 ± (3.77) | 72.85 ± (1.26) | 78.83 ± (0.48) |
| gate | 72.06 ± (0.60) | 79.11 ± (0.55) | 16.98 ± (0.23) | 73.47 ± (0.49) | 74.13 ± (1.33) | 72.86 ± (1.09) | 78.88 ± (0.65) |
| node | 71.83 ± (0.49) | 79.77 ± (0.42) | 17.33 ± (0.35) | 72.24 ± (0.79) | 70.25 ± (1.66) | 74.37 ± (0.49) | 79.68 ± (0.68) |
| tabtransformer | 71.47 ± (0.70) | 78.07 ± (0.51) | 16.92 ± (0.28) | 72.71 ± (1.10) | 72.89 ± (2.90) | 72.63 ± (1.29) | 77.74 ± (0.51) |
| categoryembedding | 71.33 ± (0.64) | 78.82 ± (0.44) | 17.12 ± (0.24) | 71.85 ± (1.40) | 70.25 ± (3.59) | 73.69 ± (1.31) | 78.64 ± (0.56) |
| resnet | 71.04 ± (0.95) | 77.51 ± (0.71) | 16.34 ± (0.66) | 73.78 ± (0.97) | 78.16 ± (4.09) | 70.06 ± (2.03) | 76.83 ± (1.13) |
| s1dcnn | 69.45 ± (0.79) | 78.71 ± (0.56) | 17.14 ± (0.20) | 66.15 ± (1.68) | 57.28 ± (2.88) | 78.42 ± (0.96) | 78.59 ± (0.33) |

Scores are very similar for all datasets with only a 3% gap in performance between the best performing FT Transformer and the last S1DCNN model. Once again the choice of best model here is non trivial, since a Deep Learning model achieves a mere 0.02% increase in accuracy with respect to the best Tree Based model.

## Covertype

For the covertype dataset the clear winners are tree based models. Perhaps a larger number of search iterations is needed for deep learning architectures to bridge this gap.

| model | accuracy | f1 |
|---|---|---|
| catboost | 94.20 ± (0.10) | 94.19 ± (0.10) |
| xgb | 94.09 ± (0.06) | 94.07 ± (0.06) |
| gandalf | 90.60 ± (0.73) | 90.87 ± (0.67) |
| s1dcnn | 90.26 ± (3.32) | 90.14 ± (3.43) |
| mlp | 88.92 ± (0.32) | 88.54± (0.33) |
| resnet | 87.22 ± (1.80) | 87.69 ± (1.60) |
| categoryembedding | 86.34 ± (0.93) | 86.70 ± (0.87) |
| gate | 80.38 ± (4.31) | 81.11 ± (3.99) |
| autoint | 72.96 ± (2.13) | 74.25 ± (1.90) |
| tabnet | 72.09 ± (2.90) | 73.38 ± (2.70) |
| fttransformer | 79.65 ± (1.03) | 71.06 ± (0.99) |
| node | 72.52 ± (1.14) | 75.09 ± (1.21) |
| tabtransformer | 70.82 ± (2.04) | 73.51 ± (1.61) |

## 4.3 Results Summary and Conclusions

Out of the 10 datasets tested, tree based models "won" in 5 cases if we consider the win based solely on the highest score with respect to the evaluation metric used in optimization. In this table we can view the rank achieved by the models on all datasets.

| dataset | CATBOOST | XGB | GANDALF | NODE | FTT. | CAT. EMBED | AUTOINT | GATE | S1DCNN | MLP | TABNET | RESNET | TAB. TRANS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADULT | 1 | 2 | 7 | 5 | 3 | 8 | 4 | 6 | 13 | 10 | 9 | 12 | 11 |
| AGECONDITIONS | 2 | 3 | 4 | 1 | 10 | 8 | 7 | 9 | 5 | 6 | 12 | 13 | 11 |
| BREASTCANCER | 11 | 13 | 3 | 2 | 4 | 5 | 8 | 1 | 9 | 7 | 10 | 12 | 6 |
| COVERTYPE | 1 | 2 | 3 | 12 | 11 | 7 | 9 | 8 | 4 | 5 | 10 | 6 | 13 |
| CREDITCARD | 4 | 3 | 1 | 5 | 7 | 2 | 6 | 8 | 9 | 10 | 13 | 11 | 12 |
| DIABETES | 1 | 3 | 2 | 7 | 11 | 4 | 8 | 5 | 6 | 10 | 9 | 12 | 13 |
| HELOC | 2 | 4 | 6 | 3 | 1 | 8 | 5 | 11 | 7 | 10 | 9 | 13 | 12 |
| HOUSING | 1 | 2 | 6 | 7 | 9 | 13 | 11 | 8 | 4 | 3 | 12 | 5 | 10 |
| IRIS | 13 | 6 | 2 | 2 | 2 | 6 | 1 | 11 | 10 | 11 | 6 | 6 | 2 |
| TITANIC | 2 | 1 | 6 | 3 | 4 | 5 | 9 | 8 | 11 | 12 | 7 | 10 | 13 |

In the category of very small datasets, the Deep Learning based models perform better and win 3 out of 4 times. In the medium dataset regime, the tree based models win in 2 out of the 3 datasets. In the medium-large dataset regime deep learning models win 2 out of 3 times.
Looking at the results from a task perspective, in the only regression task analyzed the tree based models win by a very large margin. In binary classification tasks overall the deep learning models take the first place 4 times out of 7. In the multiclass classification realm, the Deep Learning models really struggle with the large Covertype dataset and lose by quite a large margin. Perhaps a larger search space would be needed to close this gap.

This study tested 13 models on 10 datasets, the binary classification datasets were evaluated with respect to 7 metrics, multiclass with respect to 2 metrics and regression with respect to 3 metrics. Overall this led to a total of 56 distinct points that could be used for comparison.

In regards to analyzing model performance with respect to specific architectural components, an interesting statistic is shown by the number of times a single architecture or category of architectures bested all the other architectures considering the 56 evaluation metrics. The split is extremely balanced with both Deep Learning and Tree Based types tying with 28 metrics each. However from a single model perspective, there is no single deep learning architecture which is as consistent as XGB and CatBoost are.

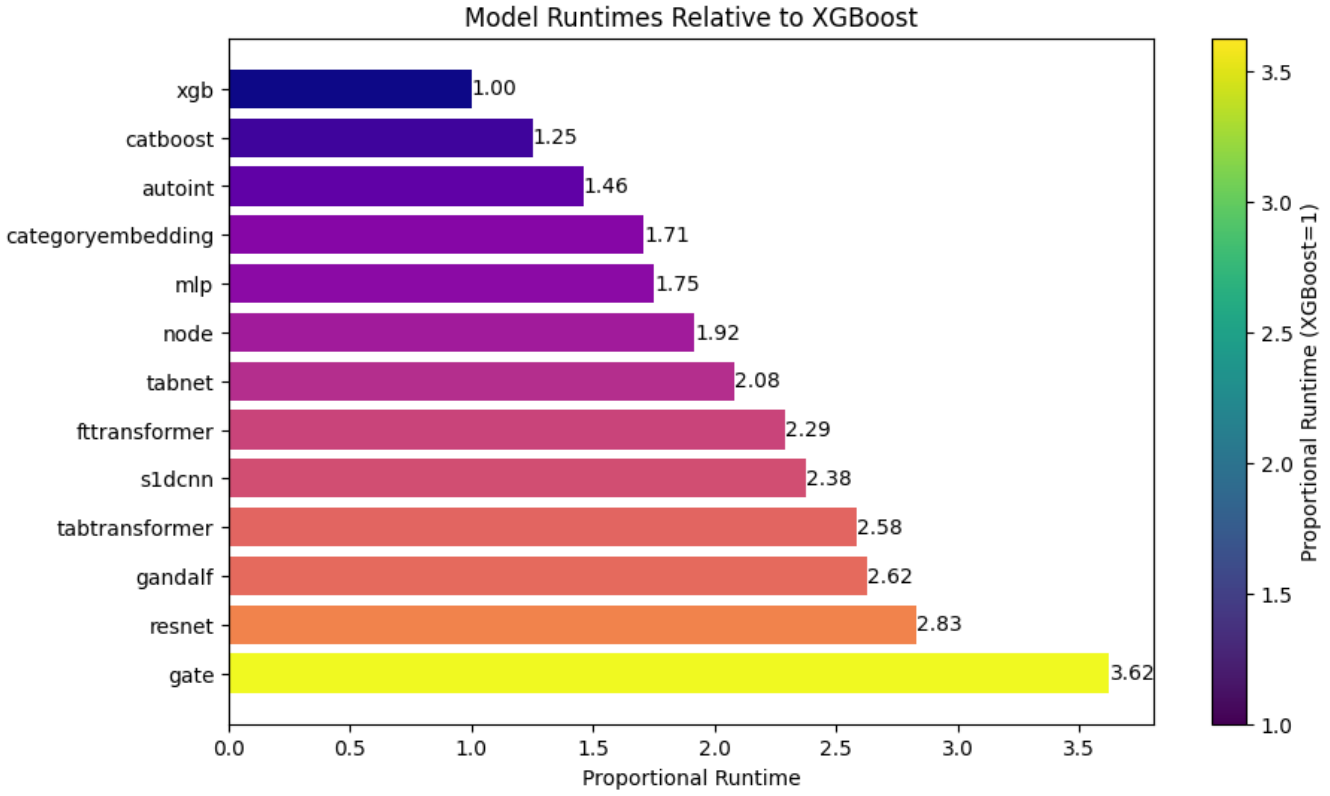
Focusing on the distinctive characteristics of Neural Network architectures tested, there were 5 models (NODE, GATE, TabTransformer, TabNet, FT Transformer which implemented some form of the Self-Attention mechanism. There were also 5 Neural Network architectures which did not implement this mechanism (MLP, Category Embedding, GANDALF, AutoInt, S1DCNN). The Transformer based architectures outperformed all others in 10 metrics while the other group won in 18. This is evidence that the attention mechanism applied to tabular data is perhaps not well suited and does not generally provide a benefit to neural network architectures in the tabular data domain. However the application of attention in this domain is indeed still in its recent phases and there is likely room for optimization and improvement.

A summary view of the average rank achieved over all tasks along with the relative standard deviation is shown below:

| | AVERAGE RANK | STD. DEV. |
|---|---|---|
| CATBOOST | 3.8 | 4.4 |
| XGB | 3.9 | 3.5 |
| GANDALF | 4.0 | 2.1 |
| NODE | 4.7 | 3.3 |
| FTT. | 6.2 | 3.9 |
| CAT. EMBED | 6.6 | 3.0 |
| AUTOINT | 6.8 | 2.9 |
| GATE | 7.5 | 3.0 |
| S1DCNN | 7.8 | 3.1 |
| MLP | 8.4 | 3.0 |
| TABNET | 9.7 | 2.2 |
| RESNET | 10.0 | 3.1 |
| TAB. TRANS | 10.3 | 3.6 |

There are 2 Deep Learning models which can be considered the most competitive overall; GANDALF which is one of the most recently published architectures (2022) achieved the most consistent results across the board with an average rank of 4, only 0.2 points below CatBoost but with a much lower standard deviation of 2.0. NODE, also is competitive with the current state of the art models. These 2 architectures exhibit some promising results and seem to be a step in the right direction.
If we focus on the tasks exhibiting high class imbalance (defined here by the positive class ratio being less than 20%), tree based models win in 2 out of the 3 datasets.

Training time was significantly higher for Deep Learning models.




Effectively this limits our ability to expand the search space which might be a larger issue for the deep learning models. These architectures are vastly less used in the industry and tree based models have the advantage of numerous tests being executed on a wide range of problems to select a more optimal search space. This in turn might further give an advantage to the tree based models which have an easier hyperparameter optimization process. Libraries such as XGBoost and CatBoost have been extensively optimized for performance, whereas it is possible that a number of these deep learning implementations still have room for performance improvements. The practice of benchmarking and testing these algorithms is of crucial importance.

The results from benchmarking could lead practitioners to better comprehend which architectures can be competitive with the state of the art models and which architectural components are more efficient both in terms of model performance and computational cost. Finding the balance between model complexity and predictive performance will be a key aspect of future developments on tabular data architectures.

## 4.4 Limitations and Questions for Future Work.

Firstly I have to comment on the overall meaningfulness of these results. It is indeed possible if not likely that for a number of these datasets, especially the larger ones, we have not explored the full capabilities of these models and a large portion of the hyperparameter search space is left unexplored. This on one hand means that we cannot conclude definitely in any of these cases that one model or category of models is superior to one another in a general sense. The search iterations were set to 50 for all models simply due to a computational time constraint, with some of the largest deep learning models needing more than 2 hours per k-fold iteration (5 folds). I also want to highlight a potential argument in terms of the evaluation strategy for these model that has not been, to the best of my knowledge, raised in the field of tabular data benchmarks. The strategy used in this study follows what has been done in [14],[15],[16] as well as most other papers benchmarking the performance of algorithms on tabular data. This consists in training the models with k-fold cross validation and then averaging performances over the folds in the validation sets. While this strategy might seem the most fair I believe there is one issue that is not addressed. Most if not all algorithms perform some sort of early stopping on the validation loss. All algorithms then are judged based on this validation loss. While this might be a perfectly good strategy for hyperparameter optimization it does suffer from an implicit bias: there is no real hold-out test set. In a real world scenario we would have a set of training data on which we can train and validate our models, but subsequently we would be using the model for inference on a new unseen set of data, otherwise known as the test set. We would not be able to use any form of information from this test set. The sole fact of using early stopping and then evaluating on the validation scores is a form of information leakage. The solution to this would require to still perform k-fold cross validation for hyperparameter optimization, however for actual model evaluation we would use a separate unseen test set on which we only perform inference and evaluation. In

terms of training time this would probably only be marginally more expensive and is an evaluation strategy that I believe should be used in future benchmarks.

## 5 Bibliography

[1] Ali, M. Abid, P. J. Hickman, and A. T. Clementson. “The Application of Automatic Interaction Detection (AID) in Operational Research.” Operational Research Quarterly (1970-1977) 26, no. 2 (1975): 243–52. https://doi.org/10.2307/3008458.

[2] Hunt, E. B., Marin, J., & Stone, P. J. (1966). Experiments in induction. New York: Academic.

[3] Messenger, Robert, and Lewis Mandell. “A Modal Search Technique for Predictive Nominal Scale Multivariate Analysis.” Journal of the American Statistical Association 67, no. 340 (1972): 768–72. https://doi.org/10.2307/2284634.

[4] Breiman, L., J. H. Freidman, Richard A. Olshen and C. J. Stone. “CART: Classification and Regression Trees.” (1984).

[5] Quinlan, J.R. Induction of decision trees. Mach Learn 1, 81–106 (1986). https://doi.org/10.1007/BF00116251

[6] Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

[7] The Strength of Weak Learnability. / Schapire, Robert E.
In: Machine Learning, Vol. 5, No. 2, 06.1990, p. 197-227.

[8] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[9] McCulloch, Warren S., and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity." The bulletin of mathematical biophysics 5 (1943): 115-133.

[10] Rosenblatt, Frank. "The perceptron: a probabilistic model for information storage and organization in the brain." Psychological review 65, no. 6 (1958): 386.

[11] Werbos, Paul. "Beyond regression: New tools for prediction and analysis in the behavioral sciences." PhD thesis, Committee on Applied Mathematics, Harvard University, Cambridge, MA (1974).

[12] Gorishniy, Yury, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. "Revisiting deep learning models for tabular data." Advances in Neural Information Processing Systems 34 (2021): 18932-18943.

[13] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In 2009 IEEE conference on computer vision and pattern recognition, pp. 248-255. Ieee, 2009.

[14] Borisov, Vadim, 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 (2022).

[15] Grinsztajn, Léo, 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 (2022): 507-520.

[16] Shwartz-Ziv, Ravid, and Amitai Armon. "Tabular data: Deep learning is not all you need." Information Fusion 81 (2022): 84-90.

[17] Arik, Sercan Ö., and Tomas Pfister. "Tabnet: Attentive interpretable tabular learning." In Proceedings of the AAAI conference on artificial intelligence, vol. 35, no. 8, pp. 6679-6687. 2021.

[18] Abutbul, Ami, Gal Elidan, Liran Katzir, and Ran El-Yaniv. "Dnf-net: A neural architecture for tabular data." arXiv preprint arXiv:2006.06465 (2020).

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

[20] https://www.kaggle.com/

[21] Chen, Tianqi, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, Kailong Chen, Rory Mitchell, Ignacio Cano, and Tianyi Zhou. "Xgboost: extreme gradient boosting." R package version 0.4-2 1, no. 4 (2015): 1-4.

[22] Prokhorenkova, Liudmila, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. "CatBoost: unbiased boosting with categorical features." Advances in neural information processing systems 31 (2018).

[23] Joseph, Manu, and Harsh Raj. "GATE: Gated Additive Tree Ensemble for Tabular Classification and Regression." arXiv preprint arXiv:2207.08548 (2022).

[24] Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. "Empirical evaluation of gated recurrent neural networks on sequence modeling." arXiv preprint arXiv:1412.3555 (2014).

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

[26] Agarwal, Rishabh, Levi Melnick, Nicholas Frosst, Xuezhou Zhang, Ben Lengerich, Rich Caruana, and Geoffrey E. Hinton. "Neural additive models: Interpretable machine learning with neural nets." Advances in neural information processing systems 34 (2021): 4699-4711.

[27] Dubey, Abhimanyu, Filip Radenovic, and Dhruv Mahajan. "Scalable interpretability via polynomials." Advances in neural information processing systems 35 (2022): 36748-36761.

[28] Gorishniy, Yury, Ivan Rubachev, and Artem Babenko. "On embeddings for numerical features in tabular deep learning." Advances in Neural Information Processing Systems 35 (2022): 24991-25004.

[29] Somepalli, Gowthami, 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).

[30] Huang, Xin, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. "Tabtransformer: Tabular data modeling using contextual embeddings. arXiv 2020." arXiv preprint arXiv:2012.06678 (2012).

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

[32] Cai, Shaofeng, Kaiping Zheng, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Meihui Zhang. "Arm-net: Adaptive relation modeling network for structured data." In Proceedings of the 2021 International Conference on Management of Data, pp. 207-220. 2021.

[33] Ke, Guolin, Zhenhui Xu, Jia Zhang, Jiang Bian, and Tie-Yan Liu. "DeepGBM: A deep learning framework distilled by GBDT for online prediction tasks." In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 384-394. 2019.

[34] Sarkar, Tushar. "XBNet: An extremely boosted neural network." Intelligent Systems with Applications 15 (2022): 200097.

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

[36] Olaru, Cristina, and Louis Wehenkel. "A complete fuzzy decision tree technique." Fuzzy sets and systems 138, no. 2 (2003): 221-254.

[37] Ke, Guolin, Jia Zhang, Zhenhui Xu, Jiang Bian, and Tie-Yan Liu. "TabNN: A universal neural network solution for tabular data." (2018).

[38] Ivanov, Sergei, and Liudmila Prokhorenkova. "Boost then convolve: Gradient boosting meets graph neural networks." arXiv preprint arXiv:2101.08543 (2021).

[39] Vaswani, Ashish, 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).

[40] Kadra, Arlind, Marius Lindauer, Frank Hutter, and Josif Grabocka. "Regularization is all you need: Simple neural nets can excel on tabular data." arXiv preprint arXiv:2106.11189 536 (2021).

[41] Rahaman, Nasim, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. "On the spectral bias of neural networks." In International Conference on Machine Learning, pp. 5301-5310. PMLR, 2019.

[42] Zhu, Yitan, Thomas Brettin, Fangfang Xia, Alexander Partin, Maulik Shukla, Hyunseung Yoo, Yvonne A. Evrard, James H. Doroshow, and Rick L. Stevens. "Converting tabular data into images for deep learning with convolutional neural networks." *Scientific reports* 11, no. 1 (2021): 11325.

[43] Song, Weiping, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. "Autoint: Automatic feature interaction learning via self-attentive neural networks." In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1161-1170. 2019.

[44] https://www.kaggle.com/c/titanic/data

[45] https://www.kaggle.com/datasets/uciml/iris

[46] https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data

[47] https://www.kaggle.com/competitions/icr-identify-age-related-conditions

[48] https://archive.ics.uci.edu/dataset/31/covertype

[49] https://www.kaggle.com/datasets/averkiyoliabev/home-equity-line-of-creditheloc

[50] https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

[51] https://www.kaggle.com/datasets/camnugent/california-housing-prices

[52] https://archive.ics.uci.edu/dataset/2/adult

[53] https://archive.ics.uci.edu/dataset/296/diabetes+130-us+hospitals+for+years+1999-2008