1 2 9 0

UNIVERSIDADE D
COIMBRA

Pedro Nuno Cazegas Pimenta de Sá

FRAUD DETECTION WITH ALGORITHMS FOR TABULAR DATA

Dissertation in the context of the Master in Data Science and Engineering, advised by Professor Jorge Henriques and Susana Brandão and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTMENT OF INFORMATICS ENGINEERING

Pedro Nuno Cazegas Pimenta de Sá

Fraud Detection with Algorithms for Tabular Data

Dissertation in the context of the Master in Data Science and Engineering, advised by Prof. Jorge Henriques and Susana Brandão and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE Ð
COIMBRA

Departamento de Engenharia Informática

Pedro Nuno Cazegas Pimenta de Sá

Fraud Detection with Algorithms for Tabular Data

Dissertação no âmbito do Mestrado em Engenharia e Ciência de Dados, orientada pelo Professor Jorge Henriques e Susana Brandão e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2023

Acknowledgements

I hereby express my deepest gratitude to all the people that positively influenced my journey. Foremost, to my thesis advisors Prof. Jorge Henriques and Susana Brandão for the unmatched expertise and countless lessons they provided along the way. They are indeed a testament to the quality of my work. Also, to Feedzai for the opportunity to work in a project that delighted me in every way possible.

To my family for the unwavering support and for being the driving reason for my academic journey: to make my parents proud. Thank you, for everything.

To my dear friends for the relentless motivation and always opportune advice. I wholeheartedly cherish all the good times we spent together and hope for more to come.

To Sofia, for being everything I need and more.

Finally, I dedicate this work to my grandfather for being the person I always look up to.

Abstract

The massive breakthrough in the world's technological landscape has encouraged companies and businesses to move to the digital medium. This is especially evident in the payment industry, considering the popularity of online payments and cardless transactions has increased over the years. Although there is a certain appeal towards automated and digital payment infrastructure, this also provides new ventures for criminal activity. Financial fraud is a paramount concern for financial institutions, and the innovations in the consolidation of prevention systems are rapidly surpassed by smarter strategies for performing fraud. Financial fraud has registered worldwide losses exceeding one billion dollars [Bank, 2021] – which represents a major liability for financial entities.

Manual systems for detection of fraud are becoming obsolete, as they fail to keep up with smarter criminals and big data. Naturally, Machine Learning stands as a potential candidate for dealing with this problem provided its automating and intelligent capabilities, namely, on the detection of patterns from data. The relevant literature highlights that both tree-based and Deep Learning approaches are widely used in fraud detection, despite an emerging debate on why tree-based algorithms consistently outperform Deep Learning on tabular data.

In this thesis, we study the performance gap between tree-based and Deep Learning algorithms for tabular data, with a focus on fraud detection. We iterate through tree-based methods, such as Gradient Boosting Decision Trees, and recent Deep Learning algorithms for tabular data. We explore possible root causes for this gap by applying several transformations to real data from the payments industry so as to widen (or shorten) the gap. Our results suggest that the performance gap may *generally* stem from a disagreement between the prior assumptions of Deep Learning algorithms and the properties of tabular data: (i) neural networks misrepresent irregular patterns in tabular data; (ii) in tabular data, the target is usually a function of just a small subset of features. Amongst the more recent algorithms, we show that TabNet and FT-Transformer share some similarities with tree-based methods that allow them to learn representations that better align with the properties of tabular data.

Keywords

fraud detection, tabular data, gradient boosting decision trees, deep learning

Resumo

A constante inovação no panorama tecnológico mundial motiva as empresas e instituições a cimentarem-se no meio digital. Este movimento torna-se evidente na indústria dos pagamentos, dado o recente aumento na popularidade de compras online e transações *cardless*. Apesar de existir um certo apelo à adoção the infraestruturas digitais e automatizadas para pagamentos, essa adoção disponibiliza, também, novos meios para atividade criminosa. Fraude financeira é uma preocupação fulcral para instituições financeiras, e as recentes inovações na consolidação dos sistemas de prevenção são rapidamente ofuscadas por esquemas fraudulentos mais inteligentes: a fraude financeira tem registados perdas, a nível mundial, superiores a um bilião de dólares [Bank, 2021], o que representa uma vulnerabilidade de maior importância para instituições financeiras.

Sistemas manuais para deteção de fraude estão a tornar-se obsoletos por não conseguirem acompanhar as vagas de criminosos mais inteligentes e o movimento da *big data*. Naturalmente, Machine Learning destaca-se como um potencial candidato para lidar com este problema, pelas suas capacidades de automação e inteligência, nomeadamente, na deteção de padrões a partir de dados. A literatura destaca que tanto métodos à base de árvores, como Deep Learning, são bastante utilizados na deteção de fraude, apesar da existência de um debate sobre o porquê dos métodos à base de árvores serem consistentemente melhores que Deep Learning em dados tabulares.

Nesta tese, investigamos a diferença de desempenho entre algoritmos à base de árvores e Deep Learning em dados tabulares, com especial foco na deteção de fraude. Iteramos sobre métodos baseados em árvores, tais como Gradient Boosting Decision Trees, e algoritmos recentes de Deep Learning para dados tabulares. Exploramos possíveis causas para esta diferença de desempenho através da aplicação de transformações sobre dados reais da indústria de pagamentos, de forma a alargar (ou encurtar) a diferença de desempenho. Os resultados sugerem que a diferença de desempenho terá origem no desacordo entre os pressupostos dos algoritmos de Deep Learning e as propriedades dos dados tabulares: (i) as redes neuronais deturpam os padrões irregulares presentes em dados tabulares; (ii) em dados tabulares, o *target* é geralmente uma função de apenas um pequeno grupo de *features*. De entre os algoritmos mais recentes, demonstramos que o TabNet e o FT-Transformer partilham algumas semelhanças com métodos à base de árvores que possibilitam a aprendizagem the representações melhor alinhadas com as propriedades dos dados tabulares.

Palavras-Chave

deteção de fraude, dados tabulares, gradient boosting decision trees, deep learning

Contents

Acronyms

AML	Anti Money Laundering.
CNN	Convolutional Neural Network.
CNNs	Convolutional Neural Networks.
CNP	Card-Not-Present.
CP	Card-Present.
DL	Deep Learning.
DNN	Deep Neural Network.
DNNs	Deep Neural Networks.
ECB	European Central Bank.
EFB	Exclusive Feature Bundling.
EMV	Europay, Mastercard and Visa.
FC	Fully Connected.
FFN	Feed Forward Neural Network.
FNN	Fully Connected Neural Network.
GAN	Generative Adversarial Network.
GBDT	Gradient Boosting Decision Tree.
GBDTs	Gradient Boosting Decision Trees.
GLU	Gated Linear Unit.
GOSS	Gradient One-Side Sampling.
MHSA	Multi-Head Self-Attention.
ML	Machine Learning.
MLP	Multi Layer Perceptron.
ODT	Oblivious Decision Tree.
ODTs	Oblivious Decision Trees.

OHE One-Hot Encoding.

PCA Principal Component Analysis.

PIN Personal Identification Number.

POS Point-of-Sale.

ReGLU REctified Gated Linear Unit.

ReLU REctified Linear Unit.

RNNs Recurrent Neural Networks.

SBG Stochastic Gradient Boosting.

SEPA Single Euro Payments Area.

SGD Stochastic Gradient Descent.

SVD Singular Value Decomposition.

TPE Tree-structured Parzen Estimator.

UT unitary transformation.

UTs unitary transformations.

List of Figures

List of Tables

Chapter 1

Introduction

Fraud is an act of "deceiving or misrepresenting" often to "intentionally (...) induce another to part with something of value" [Dictionary, 2023]. Although fraud itself spans many distinct yet intertwining domains of everyday life (e.g. telecommunications, healthcare and insurance fraud), the focus of this thesis lies solely on financial fraud.

Financial fraud takes place when a perpetrator – also known as the *fraudster* – exploits a vulnerability in payment infrastructure to unknowingly and unlawfully acquire money. This form of criminal activity actively represents a major liability for financial institutions, as the potential losses far exceed the billion dollar mark [Bank, 2021]. Indeed, it constitutes a legal obligation to protect its clients and very often to reimburse them.

The major turnover in the general technological scenery has motivated the large corpora of financial institutions to "go digital". However, the digitalization of the payment industry compels criminals to also move to the digital medium, which gives birth to new vulnerabilities and smarter fraud initiatives. Such a trend revealed that the manual and orthodox systems for fighting financial fraud are seldom capable of coping with the increase in criminal activity. Furthermore, Machine Learning (ML) plays a key role in the adoption of intelligent and automated systems for dealing with fraudulent activity: a major contender in many other industries (e.g. healthcare, autonomic vehicles and facial recognition), ML stands out as a potential candidate for all-encompassing solutions for tackling financial fraud that can fairly handle the recurrent criminal paradigm shift and the underlying technical complexities.

This thesis describes the general problem of fraud detection and delves into a well-documented performance gap between tree-based methods, such as Gradient Boosting Decision Trees (GBDTs), and Deep Learning (DL). The aim of this work is to uncover the possible root causes for the existence of the gap, more so relevant in a liable domain such as fraud detection.

1.1 Fraud Detection

According to [Bolton and Hand, 2002], *fraud detection* is the process of identifying fraud once it has been committed, ideally, as soon as possible. Fraud detection is commonly associated with its counterpart, *fraud prevention*, which deals with the design of measures for preventing acts of fraud, such as PIN-protected cards and EMV chips. However, such methods are not foolproof and thus fraud detection is much needed as a mechanism to detect the presence of fraud when prevention fails.

The landscape in fraud activity indicates that the technological growth fueling the development of better detection mechanisms and payment systems is not decreasing the incidence of fraud – *per se* – as much as it is shifting the paradigm for criminals. With new payment schemes and credit card systems also come new types of fraud, as criminals are forced to adapt and design innovative strategies. According to [Report, 2022], over \$32 billion was lost to fraud worldwide in 2021, for which $\approx 36.8\%$ of these losses occurred in the United States. The European Central Bank (ECB) reported that in 2019: (i) the total value of overall card transactions using cards issued within the Single Euro Payments Area (SEPA) and acquired worldwide increased by 6.5% compared to 2018, whereas corresponding fraud grew by 3.4%; (ii) the total value of fraudulent transactions using cards issued within SEPA and acquired worldwide amounted to 1.87 billion EUR [Bank, 2021]. Therefore, rather than relying on the consolidation of the payment industry and the (unlikely) withdrawal of adversaries from criminal activity, financial entities ought to devise state-of-the-art fraud detection mechanisms that encompass a wide range of rapidly evolving fraud scenarios.

1.1.1 Verticals

Although financial fraud can itself span a considerable set of subdomains, only four use cases are considered in this work:

- **Transaction Fraud** is the most common variant and occurs when fraudsters perform a transaction with a card that doesn't belong to them, such as stolen credit cards. Formally, a transaction can either be Card-Present (CP), when the card is physically present at the merchant, or Card-Not-Present (CNP) when the payment is performed by providing card details without physically using the card (e.g. over the internet). CNP fraud gives fraudsters more flexibility and less exposure, whilst CP forces fraudsters to be physically present at a merchant's site;
- **Anti Money Laundering (AML)** is a task concerned with fighting money laundering: the process of concealing the origins of illegally-obtained money by passing it through a complex sequence of forged transfers or commercial transactions. Given its nature, it's extremely hard to detect;
- **Account Takeover** occurs when the fraudster illegally accesses an individual's bank account to undergo credit transfers, perform payments or withdraw money;

- **Account Opening Fraud**, also known as "new account fraud", occurs when the fraudster opens a bank account under an individual's name by providing their personal information. It can happen unknowingly to the individual or through "friendly fraud": when an individual purposely gives personal information to the fraudster to make fake purchases on the account and later argue that they don't own the account.

Across every industry and sector, any business engaged in the sale of goods or services that provides a Point-of-Sale (POS) terminal accepting credit cards as a form of payment (i.e., a merchant) is vulnerable to endure fraudulent activity in many forms. Therefore, every business should be concerned with fraud detection under the aforementioned use cases to protect its assets and maintain a good reputation within its client base.

1.1.2 Data Mining

Fraud detection is not an easy task, as it involves processing and analyzing massive amounts of data. For instance, the credit card issuer VISA processed around 226 billion purchase transactions worldwide in 2021 [Report, 2022]. Roughly speaking, this averages to a total of around 620 million transactions processed per day – a number so large, that it becomes inconceivable to manually inspect every transaction. Not only that, but it's decisive that the response time to fraud alerts is as brief as possible: the limitations of analyzing around 430 000 transactions p/ minute in a short frame are unquestionable. Moreover, considering the constantly evolving fraud scenarios and the user behaviour shifting as the payment industry itself changes, data mining emerges as a strong candidate for tackling the fraud detection problem in a fast and efficient manner.

The use of data mining for fraud detection concerns the identification of patterns or anomalies, typically, on data from credit transactions. This is data in tabular format, where rows represent transactions, and columns comprise details about a transaction (e.g. merchant code, transaction amount and cardholder ID). Supplementary statistical information is often coupled with the available data, such as behaviour profiles (e.g. the average amount spent by the cardholder in the past 3 weeks). In real-world scenarios, the data is seldom labelled, which partially explains why fraud detection is hard to achieve in practice: without the "ground truth" about a given transaction (whether it's fraudulent or legitimate), one becomes limited by the methods (supervised or unsupervised) suitable for this type of data.

The purpose of data mining for fraud detection is not necessarily to play a leading role in decision-making. Following the principles of *machine-in-the-loop* learning, i.e. to incorporate machines in decision-making processes under a supporting role, one *should* harness the power of data mining as an instrument to fraud analysts. A fraud detection system is responsible for analyzing the transactions and flagging suspicious ones, which can then be manually inspected by analysts. As such, combining the domain expertise of fraud analysts with the automated processing capacity of data mining represents a suitable path towards detecting fraud more efficiently and further reducing examination costs.

As highlighted by the relevant literature on the subject, practitioners often rely on machine learning methods to tackle the fraud detection problem and steer away from rigid rule-based systems [Bhattacharyya et al., 2011; Bolton and Hand, 2002]. A common approach is to exploit the data in ways that off-the-shelf[1] methods (e.g. Neural Networks and Decision Trees) perform better and better, whereas more recent works have started using DL for its superior predictive ability. Regardless, fraud detection is bound by a set of challenges that render its deployment cumbersome. For instance, the very availability of the fraud labels may restrict the set of algorithms to supervised and/or unsupervised approaches.

1.1.3 Challenges

The fraud detection domain itself raises many challenges, both in theory and in practice. Foremost, it greatly suffers from **class imbalance**, as fraudulent transactions usually account for only $\approx 1\%$ of the whole data, which often entails the usage of sampling techniques for balancing out the class distributions [Bolton and Hand, 2002]. Indeed, such a condition significantly impairs the ability to learn from data – more so, on domains whose concept is difficult to model, such as fraud detection [Japkowicz and Stephen, 2002].

Because criminals are compelled to change their approach as payment schemes themselves are evolving and detection systems are improving, fraudulent activity may not look the same now as it did ten years ago (from a data-driven standpoint). The characteristics of the data change over time (and so its statistical properties), which quickly renders fraud detection models virtually obsolete – this is called **concept drift**.

The need for model **interpretability** stems from an undesired incompleteness in problem formalization, which happens to be a particularity of fraud detection: the legal and financial implications of correct predictions entail the need for an explanation of how a model came to such predictions, given that correct predictions only partially solve the problem (e.g. "transaction XYZ is fraudulent, but why?") [Molnar, 2022]. Not only that, but in the fraud detection domain, predictive models cannot afford to be mistaken, at the expense of losing credibility w.r.t the clients: interpretability can also be leveraged for properly diagnosing faulting models and correcting them. As such, the fraud domain often demands interpretable models to comply with transparency and trust guidelines.

Finally, fraud detection is an unpredictable, **fast-paced environment**. As such, the slow training times of overly complex and heavily-parameterized models should often be traded for simplicity and speed at the cost of lower predictive capacity. However, this trade-off is not always straightforward, and every situation calls for a different approach.

[1]Ready to use; sparing the need for expert, manual engineer

1.2 Motivation

The characterization of the fraud detection domain above revealed suitable paths for improvement. Foremost, one understands the financial implications of fraud: businesses and card issuers can (and will) register billions in losses if not properly protected against fraud from a diverse set of attack vectors. Not only that, but the domain itself presents many challenges that render practical implementations somewhat difficult.

The relevant literature suggests that DL approaches have emerged as a strong candidates in fraud detection solutions, despite the well-studied underperformance when compared against tree-based methods (based on decision/regression trees) in tabular data (such as the data highly present in the fraud detection domain). Namely, GBDTs have been extensively used for tabular data problems and remain the long-established top contenders. This performance gap has been studied w.r.t closing the gap, but no substantial works have focused on why the performance gap exists [Borisov et al., 2021; Grinsztajn et al., 2022].

In sum, we highlight a need for a well-rounded, extensive study on why there is a performance gap between tree-based and DL algorithms for tabular data, with a special interest in the fraud detection domain, for which the development of better DL algorithms can substantially improve the performance of fraud detection systems and thus contribute to reducing the losses of financial institutions, merchants and card issuers.

1.3 Objectives

This thesis aims to explore the properties of GBDTs and DL algorithms that cause a performance gap in learning with tabular data. More specifically, the objectives are three-fold:

- To provide a scoped review of the relevant literature on fraud detection, GBDTs and DL for tabular data;
- To conduct experiments that exploit transformations on real data from the payments industry to shorten (or widen) the performance gap between GBDTs and DL algorithms;
- To shed more light into why GBDTs *still* outperform DL on tabular data and how is this relevant in the fraud detection domain.

1.4 Outline

This thesis is organized in the following manner: Chapter 2 presents a review of the relevant literature on fraud detection, GBDTs and DL for tabular data. Chapter

3 describes the fundamental concepts highlighted in the literature review. Chapter 4 describes the research design and methods employed in the experimental work. Chapter 5 details the experiment work and discusses the results. At last, Chapter 6 presents a summary of the key findings and suggests future work on the relevant research topics.

1.5 Summary

In this chapter, we provided the problem statement alongside an introduction of the fraud detection paradigm and its key challenges. Next, we presented the motivation for this thesis, built upon the problem description. At last, we enumerated the objectives for this thesis, which contemplates the design of an experimental study contributing to a better understanding of a known performance gap between GBDTs and DL algorithms in learning with tabular data.

Chapter 2

State of the Art

In following chapter, we provide a comprehensive review of the state-of-the-art on the topic of fraud detection with a special focus on approaches for tabular data. The scope of this literature review encompasses both the earlier works and the most recent ones for aided perspective. In total, 59 papers were screened and 41 were reviewed for this chapter (the remainder was excluded for being out-of-context).

2.1 Literature Review

Fraud Detection Fraud detection has been receiving attention from the scientific community for a long time, since the earlier works of [Aleskerov et al., 1997; Brause et al., 1999; Chan et al., 1999; Dorronsoro et al., 1997; Ghosh and Reilly, 1994] on credit card fraud detection, for which they became a known reference in the literature. In [Aleskerov et al., 1997; Brause et al., 1999; Dorronsoro et al., 1997; Ghosh and Reilly, 1994], the authors follow a neural network based approach, whereas in [Chan et al., 1999], the authors propose a distributed method for aggregating classifiers trained (in parallel) on subsets of the data by a variation of the AdaBoost (see [Friedman et al., 2000]) algorithm that minimizes a more suitable cost function. On the other side of the spectrum, in [Bolton and Hand, 2001], the authors lay the groundwork for research on unsupervised fraud detection with peer-group analysis.

These early works fostered further research on the topic, much of which focused on the challenges highlighted by them (e.g. class imbalance). Among the most common approaches are tree-based methods [Bhattacharyya et al., 2011; Pozzolo et al., 2018; Taha and Malebary, 2020; Wei et al., 2013]. In these works, the authors introduce new and innovative methods, in the form of either feature engineering (e.g. the average amount spent over 3 months [Bhattacharyya et al., 2011] or behavioural patterns [Wei et al., 2013]) and/or combination of other methods (e.g. majority voting of a cost-sensitive neural network, Random Forest and contrast pattern mining [Wei et al., 2013] or aggregation of two random forests trained on recent labeled transactions and previous non-disputed transactions, respectively [Pozzolo et al., 2018]). In [Taha and Malebary, 2020], the authors employ a Bayesian hyperparameter scheme on a Gradient Boosting Decision Tree (GBDT), the LightGBM (see [Ke et al., 2017]).

Other works follow different paths that don't fit into the usual approaches: in [Duman and Ozcelik, 2011], the authors use genetic algorithms for hyperparameter searching and in [Carcillo et al., 2019], the authors use a clustering algorithm to calculate outlier scores at different granularities to generate new features. The works in [Li et al., 2021; Makki et al., 2019] tackle the problem of class imbalance by training a classifier on a highly overlapping subset to learn a better decision boundary (in the former), and by comparing classical approaches, such as Cost-Sensitive models (in the latter).

A promising candidate is, however, the Deep Learning (DL) approach; in [Fu et al., 2016], the authors use a Convolutional Neural Network (CNN) on a time-lagged feature matrix to derive meaningful data representations of transactions; in [Fiore et al., 2017], the authors use a Generative Adversarial Network (GAN) to generate synthetic samples of fraud transactions for dealing with class imbalance; in [Wang et al., 2019], the authors follow a semi-supervised graph-based approach based on social relations; in [Zhang et al., 2019], the authors use behaviour analysis to derive new features, which are then trained by a Deep Neural Network (DNN).

In line with the *No Free Lunch Theorem* [Wolpert and Macready, 1997], the main takeaway is that there is no single method that outperforms every other in every possible scenario and under every constraint. Recently, DL is making a surge in the fraud detection research corpora – mostly motivated by the remarkable predictive ability across other tasks (e.g. image, text and audio). However, tree-based models are usually preferred for their compliance with the fraud detection domain constraints (e.g. explainability and low training times) and natural suitability for tabular data [Borisov et al., 2021; Grinsztajn et al., 2022; Shwartz-Ziv and Armon, 2022].

Tabular Data Under the domain of fraud detection, typical approaches (like the ones previously mentioned) involve learning from data related to transactions and cardholders – tabular data. Tabular data is structured, heterogeneous data organized in rows (representing observations or samples) and columns (representing features or dimensions). In tabular data, features are usually of mixed types and often represent some quality about the data subject (e.g. weight, age, income). The relevant literature frequently portrays the superior ability of GBDTs to learn from tabular data, despite the continuous efforts of research on DL to try and close this well-known performance gap [Borisov et al., 2021; Gorishniy et al., 2021; Grinsztajn et al., 2022; Shwartz-Ziv and Armon, 2022]. However, a relevant question still remains in the literature: *why does this gap exist?*

In [Grinsztajn et al., 2022], the authors provide a comprehensive benchmark and empirically derive possible causes for the existence of the gap, such as neural network's inability to properly learn irregular functions[1] of the target space: tree-based models learn piece-wise constant functions, which partially explains why they are generally better at tabular learning. Additionally, the works under [Borisov et al., 2021; Kadra et al., 2021] suggest that such gap may be caused by DNNs being hy-

[1]Piece-wise, nowhere-differentiable functions

persensitive towards improper regularization. Yet, even with so little known about the true, fundamental causes for this gap, the main line of research seems to be one that proposes progressively more complex DL architectures to handle the particularities of tabular data, whilst there still lacks a deeper understanding of *why* there is a performance gap consistent across several works.

Gradient Boosting Decision Trees Initially proposed by [Friedman, 2001], GB-DTs are predominantly praised in the relevant literature as the *de facto* methods for handling tabular data [Borisov et al., 2021; Gorishniy et al., 2021; Grinsztajn et al., 2022; Shwartz-Ziv and Armon, 2022]. Fast training times, low complexity, interpretability and on-par performance across several benchmarks are some of the reasons they are preferred over DL algorithms, which are known to be architecturally complex and taking longer, sometimes infeasible, times to train. Among the most popular works, stands out XGBoost [Chen and Guestrin, 2016], LightGBM [Ke et al., 2017] and CatBoost [Dorogush et al., 2018]. Notwithstanding, there are some disadvantages associated with these tree-based methods, such as poor scalability and relying on proper feature engineering to perform well (which itself demands domain knowledge about the problem). The state-of-the-art performance on established benchmarks across other tasks and the suitability for deployment in federated and online learning scenarios are grand motivations for the adaption of DL methods to tabular data – specially – in the domain of fraud detection, in which the predictive performance is as crucial as the ability to learn models in a sequential fashion [Borisov et al., 2021; Sahoo et al., 2017]. Perhaps most importantly, DL's inherent feature learning relieves the burden of performing feature engineering by encouraging the learning of compact representations from tabular data [Bengio et al., 2013, 2009; Borisov et al., 2021].

Deep Learning DL for tabular data has been a popular research topic for quite some time, namely, on the design of novel architectures. Some works address the issue of non-differentiability in decision trees by proposing a smooth decision function in the internal nodes, allowing them to be trained with gradient descent [Katzir et al., 2020; Kontschieder et al., 2015; Popov et al., 2019; Yang et al., 2018]. Other works adopt the attention mechanism (see [Vaswani et al., 2017]) to incorporate attention modules in tabular-specific DNN architectures, which enables the learning of meaningful representations from data [Gorishniy et al., 2021; Huang et al., 2020; Somepalli et al., 2021; Song et al., 2019]. Other lines of reasoning include heavy regularization of a Multi Layer Perceptron (MLP) [Kadra et al., 2021; Shavitt and Segal, 2018], numerical feature embeddings [Gorishniy et al., 2022] and Generative Adversarial Network (GAN) based synthetic data modeling [Xu et al., 2019]. A well-known work is that of [Arik and Pfister, 2021], wherein the authors leverage sequential attention to perform feature selection with DNN blocks.

In another line of work, the authors in [Badirli et al., 2020] unify both approaches and employ gradient boosting with shallow neural networks as weak learners. However, we argue gradient boosting (in itself) doesn't hold any property that makes it more suitable for tabular data: the gap is most likely caused by internal inductive biases[2]

[2]The prior assumptions of a learning algorithm

of the core algorithms, i.e. trees and neural networks, than by the inductive biases of gradient boosting – that a concept can be learned by sequential approximation of weak learners. Even going further and increasing the capacity of the weak learner (e.g. by using a DNN) not only goes against the virtue of (gradient) boosting but renders training computationally intractable by yielding overly-parameterized models.

Notwithstanding, no substantial work on uncovering the useful inductive biases for learning data tabular has been reported [Grinsztajn et al., 2022]. A deeper understanding of GBDTs, DL algorithms and the performance gap in tabular data may drive practitioners into designing better algorithms that can leverage the representational potential of DL to further shorten the gap.

The literature gap From the reviewed literature, some common pitfalls are raised: (i) only a portion of the works (50%) tackle the fraud detection from the domain perspective and consider appropriate, industry-related evaluation metrics. The fraud domain entails the need to measure performance under a tolerance on the amount of false positives, as these can incur significant losses to the card issuers; (ii) only a portion of the works (16%) provide results under a comprehensive benchmark, whilst the majority evaluate on only one dataset. This selection bias leads to results that may not be representative of the fraud detection domain; (iii) a portion of the works (16%) learn the algorithms on a commonly used dataset[3], for which the numerical features are transformed via Principal Component Analysis (PCA), which compromises interpretability and prevents tree-based methods from performing well.

Extending the performance gap debate to the fraud detection domain is much needed. Indeed, we highlight the need for a comprehensive study that: i) evaluates on several heterogeneous, real-world tabular datasets from the payments industry ii) evaluates performance on domain-appropriate metrics.

Bearing this in mind, by applying several data transformations to shorten (or widen) the performance gap, we ought to empirically derive some of the inductive biases behind this gap. Understanding (and bridging) such a gap is of the utmost importance, either under an academic context or under a critical, fast-paced industry such as fraud detection.

Therefore, the following work hypotheses are proposed:

1. Considering the representational capacity of neural networks, we explore the representations learned by DL algorithms. From a high-level standpoint, DL algorithms first perform a sequence of arbitrary transformations to data – to learn compact representations – and later apply a linear classifier. In line with [Borisov et al., 2021], we argue these learned representations lose information w.r.t the original data. As opposed to tree-based methods, neural networks learn representations via linear and non-linear relationships that may misrepresent the relationships in tabular data.

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

We argue that the misrepresentation of tabular data as compact representations contributes to the performance gap between tree-based methods and DL algorithms. To that extent, we evaluate tree-based methods on the representations learned by DL algorithms to understand how are the hidden layers manipulating the data and how it affects the performance gap.

2. We explore feature informativeness in tabular data as a favorable property for tree-based methods. Tabular data often contains redundant and irrelevant features, wherein the target function is usually modeled by just a subset of relevant features. Tree-based methods' inductive biases render them ideal for this type of data, whereas neural networks are not robust to uninformative features [Grinsztajn et al., 2022; Ng, 2004].

   Tree-based methods often require prior feature engineering to uncover relevant features as they are unable to model implicit relationships between features. Neural networks, on the other hand, do this automatically as part of learning compact representations. We argue that tree-based methods benefit from rich features and that the performance gap can be increased by aggregating features of varying degrees of complexity. Indeed, explicitly representing linear and non-linear relationships in tabular data *should* help tree-based methods to leverage all the implicit information in the raw input and thus improve performance.

2.2 Summary

In this chapter, we presented the state-of-the-art on fraud detection and learning with tabular data. We covered both the tree-based (e.g., GBDTs) and the DL approaches for learning with tabular data. We highlighted there's a need for a comprehensive study that considers real-world tabular data from the payments industry and domain-appropriate metrics. Next, we underlined the lack of substantial works on finding the root causes for the performance gap between tree-based and DL algorithms for tabular data. Finally, we proposed a work hypothesis focused on extending the study of the performance gap to the fraud domain.

Chapter 3

Background

In the following chapter, we provide a comprehensive description of the background on tabular data, gradient boosting, Deep Learning (DL) and state-of-the-art methods to be used in the experimental work.

Notation In this thesis, the focus lies on supervised classification tasks for tabular data. For that, let's assume a dataset $\mathcal{D}$ of N observations x expressed as a d-dimensional vector of d features $\{x_i\}_{i=1}^{d}$, where x_i is the N-dimensional vector of the i-th feature, and a target variable $y \in \{0,1\}$. The goal of a learning algorithm is to learn a parameterized function $f(x;\theta)$ of the input space $\mathcal{X}$, such that $f : \mathcal{X} \to \mathcal{Y}$, subject to minimizing an arbitrary loss function $\mathbf{L}\left(y, f(x)\right)$ over the joint distribution of all possible values.

We use ρ for probability, $\circ$ for the Hadamard product, $\tilde{\mu}$ for population median, $\mathcal{U}$ for a uniform distribution and $\mathcal{N}$ for a normal distribution.

Definition 1 (Inductive Bias [Battaglia et al., 2018])**.** An inductive bias is a prior assumption about the concept and/or data intrinsic to a learning algorithm. An inductive bias naturally constraints the space of solutions to a specific class of functions. Generally, one aims to align a problem with the inductive biases of a learning algorithm, as strong (or inadequate) inductive biases unsuitably constraint the space of solutions and lead to poor generalization. Tree-based methods, e.g., hold the inductive bias that the target function is described by only a subset of relevant features and constraint decision boundaries to be parallel to the axes (e.g., to compute $x_1 > x_2$, we must transform $x_1 - x_2 > 0$).

3.1 Tabular Data

Tabular data consists of data that is presented in tabular form, where each row is an instance (object) described by several features (in columns). As opposed to image or text data, tabular data doesn't exhibit spatial and semantic relationships across instances and/or features (e.g. there is no contextual and local structure in neighbouring data points[1]). Tabular data is heterogeneous by nature: comprising features of mixed types, such as continuous and discrete values, and categorical variables, which belong to a nominal (no intrinsic ordering) or ordinal scale. In tabular data, each feature has a different meaning and encodes some quality about the data subject (e.g. weight, age, income).

Table 3.1: Common tabular data example in fraud detection

account ID	timestamp	amount	transaction type	...	is fraud?
21454	125424542	5.99	`TYPE_CARD`	...	0
10433	125436436	86.2	`TYPE_QR`	...	0
46584	125476421	1500.0	`TYPE_CARD`	...	1
98437	125488793	46.0	`TYPE_TRANSFER`	...	0

Figure 3.1 illustrates a common example of tabular data in fraud detection. There is a mix of continuous and discrete values, and categorical features. An evident property is that of *semantic constraint*, when the range of a given feature depends on the value of another feature. Indeed, the presence of categorical features can induce multimodal distributions on numerical features. This also entails the existence of relevant and redundant features: the target function can highly depend on the values of feature a and not change at all for the values of features b and c. Indeed, tabular data often exhibits irregular patterns, i.e. piece-wise nowhere-differentiable functions, in the feature space. Finally, and especially in the fraud domain, features can take up different values across a temporal axis; transactions reflect user behaviour, and user behaviour changes along time causing a *distribution drift*.

Most algorithms are not natively equipped with categorical support, hence the usual need for encoding categorical features into a numerical representation. A natural approach is to perform One-Hot Encoding (OHE) of categorical features, which leads to highly sparse feature vectors, and is further aggravated by features with high cardinality. Encoding methods, such as ordinal encoding, force an ordinal sequence that may not adequately represent the nature of feature: encoding `transaction type` into an ordinal is misleading as there is no inherent ordering.

3.2 Gradient Boosting Decision Trees

At the core of Gradient Boosting Decision Trees (GBDTs) lies the decision tree. Tree-based methods (see [Hastie et al., 2009; James et al., 2013]) partition the feature space into disjoint regions (or leafs) and model the target variable by the most occurring class in a given region[2]. At each internal node, the tree algorithm searches for the best splitting feature and value such that the resulting regions minimize a given cost function (e.g. the *Entropy*). This is done successively, in a top-down approach, until some stopping criterion is met[3]. Tree-based methods are known to

[1]i.e. pixels in an image or words in a sequence

[2]In a classification setting

[3]For sake of simplicity, we refer to binary splitting

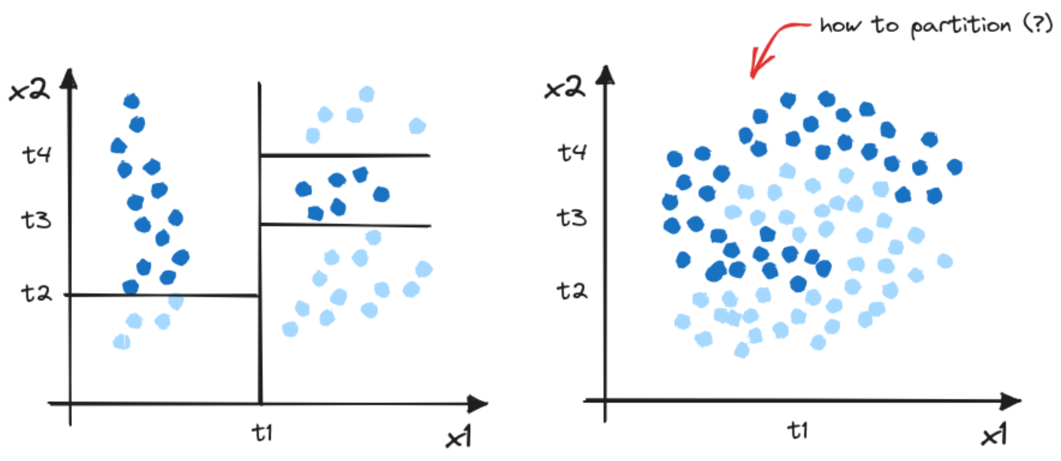


Figure 3.1: Tree decision manifolds for irregular data (left) and non-linear, smooth data (right). $t_1, \cdots, t_4$ are the threshold values for features $\{x_1, x_2\}$.

be reasonably accurate and natively interpretable, at the expense of being sensitive to small changes in data (which leads to high variance), being non-differentiable and quickly growing the number of leafs as depth increases.

In essence, a tree partitions the feature space into regions by performing axis-aligned splits such that the class overlap in each region is as small as possible. The ability to learn piece-wise constant functions allows tree-based methods to capture the irregular patterns common to tabular data. Figure 3.1 illustrates the decision manifold of a tree, showcasing its hard decision boundaries. Naturally, tree-based methods tend to perform worse when the feature space is highly non-linear as they can neither trace affine hyperplanes nor perform orientations of the feature space. Indeed, tree-based methods being good feature selection algorithms is linked to its sensitivity to the orientation of the data [Ng, 2004], which partially explains its suitability to tabular data: tree-based methods perform well well in settings where there's only a subset of relevant features. Therefore, trees' inductive biases align well with the properties of tabular data.

3.2.1 Boosting

In line with [Hastie et al., 2009], boosting is regarded as a general procedure for sequentially combining many *weak learners* (an algorithm that performs only slightly better than random guessing) into a powerful one – usually refered to as the "committee". Historically, this committee was built by taking a majority voting of the individual weak learners, although decade-old developments have pushed boosting to greater capabilities that have evolved well beyond past that.

The rationale behind boosting is that the error associated with any weak learner can be substantially reduced by forcing it to look to data on which it's expected to perform poorly [Schapire and Freund, 2012]. This assumption alone seems enough to prove boosting's ability to resist overfitting, although there is no concrete theoretical proof that such is *always* true. In fact, the practical implications of boosting are far more impressive than the underlying theory would otherwise imply [Friedman et al., 2000].

In line with [Friedman et al., 2000], boosting is a stagewise procedure for fitting a linear combination of basis functions, i.e. it approximates a decision boundary in a forward-stagewise fashion by iteratively fitting a weak learner to the residuals (error) of the previous weak learner. In the virtue of boosting, one should seek to use very simple and naive algorithms as the weak learner, which comprises the surplus benefit of trading complexity for simplicity and feasible computation. For this study, we restrict the weak learner to be the (decision) tree.

3.2.2 Gradient Boosting

The idea behind gradient boosting is to fit the weak learner to the gradient of the previous one to correct its mistakes [Friedman, 2001]. This represents a more attractive approach, as the tree parameters cannot be optimized with traiditional optimization techniques. Instead, computing the gradient w.r.t. to the tree parameters yields the step size on the direction of minimizing the loss function. Hence, a tree is first fit to the error – the pseudo-residuals – and the linear combination coefficients are learned by minimizing an arbitrary loss function in each tree region [Hastie et al., 2009]. In sum, gradient boosting performs an optimization in the tree's parameter space followed by optimization in gradient (or function) space.

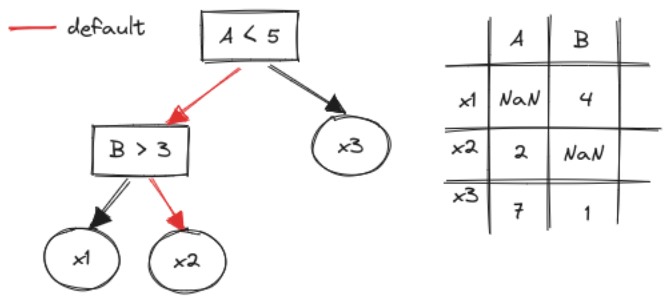
Algorithm 1: Gradient Tree Boosting [Friedman, 2001; Hastie et al., 2009]

```
   Input: input $(x, y)$, no. of iterations $M$, shrinkage $\nu$
1  begin
2      Initialize $f_0(x) = \arg\min_\gamma \mathbf{L}(y, \gamma)$
3      for $m = 1$ to $M$ do
4          Compute pseudo-residuals $r_m = -\left[\frac{\partial \mathbf{L}(y, f(x))}{\partial f(x)}\right]_{f=f_{m-1}}$
5          Fit tree to the pseudo-residuals $r_m$ yielding $L$ regions/leafs $\{R_{m,l}\}_1^{L_m}$
6          for $l = 1$ to $L_m$ do
7              $\gamma_{m,l} = \arg\min_\gamma \sum_{x \in R_{m,l}} \mathbf{L}(y, f_{m-1}(x) + \gamma)$
8          end
9          $f_m(x) = f_{m-1}(x) + \nu \sum^{L_m} \gamma_{m,l} 1(x \in R_{m,l})$
10     end
11     $f_M(x) \longleftarrow$ Final ensemble model
12 end
```

Algorithm 1 illustrates the general case of gradient tree boosting, where $1(\cdot)$ represents the indicator function and ν is the shrinkage parameter which acts as learning rate. For (binary) classification, the loss function is usually the negative binomial log-likelihood (or log loss) [Friedman et al., 2000]:

$$\mathbf{L}(y, f(x)) = -y \log(f(x)) - (1 - y) \log(1 - f(x))$$

Indeed, gradient boosting allows trees to learn some of the even more irregular patterns in data by approximating the decision boundaries. There is an implicit



	A	B
x1	NaN	4
x2	2	NaN
x3	7	1

Figure 3.2: XGBoost's default decision diagram. XGBoost avoids needless computations and decides on the default direction for missing values (adapted from [Chen and Guestrin, 2016]).

inductive bias here: a function can be approximated by taking simple guesses and improving by taking short steps at a time, i.e. the space of solutions is constrained to that of simple and naive models.

In particular, gradient boosting doesn't seem to leverage any inductive bias that is particularly useful for learning over tabular data – (decision) trees are naturally adequate for settings wherein few features are relevant and the feature space is separable by piece-wise constant functions. Therefore, any *strong* inductive bias in gradient boosting resides in the choice of the weak learner. The suitability of gradient boosting to tabular data seems to stem from the ability to reduce bias.

3.2.3 Known implementations

Under the umbrella of GBDT, the following implementations dominate the relevant research papers and competitions, consistently achieving optimal results in tabular data tasks [Shwartz-Ziv and Armon, 2022]. A key role of these implementations is to improve some aspect of GBDT (e..g reducing overfitting) by improving the split-search algorithms, appending regularization techniques and/or sampling methods.

XGBoost XGBoost [Chen and Guestrin, 2016] is a GBDT which comprises a weighted quantile sketch for learning the tree structure and a sparsity-aware algorithm. The authors also propose a new regularized loss function that penalizes the complexity of the model by encouraging the preference of shallow learners (small number of leaves). XGBoost improves the default greedy strategy for finding the optimal splitting points by defining candidate splitting points and binning continuous features according to candidate points – this technique shares some similarities with histogram-based splitting [Ke et al., 2017]. The candidate points are proposed via a quantile sketch algorithm: an approximate histogram is formed by computing the quantiles for random subsets of the data and the quantiles are used as (candidate) splitting points. The sparsity-aware algorithm incorporates a default decision for every internal node for dealing with sparse data, as seen in Figure 3.2; this effectively reduces computational effort towards sparse features, which are largely common in tabular data.

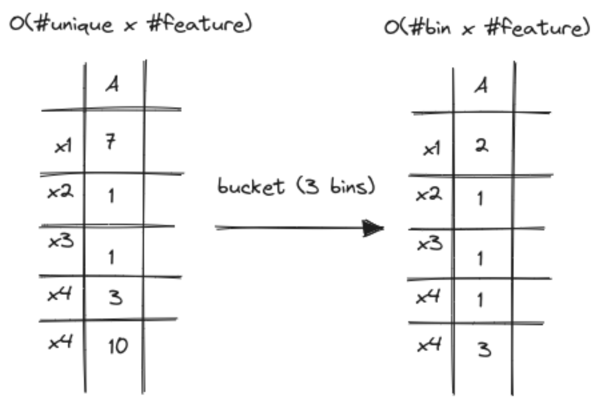


Figure 3.3: LightGBM's histogram building. Continuous features are bucketed into n bins. $O(\cdot)$ is effectively reduced as the number of bins is usually far smaller than the number of unique values in continuous features.

LightGBM LightGBM [Ke et al., 2017] replaces the greedy exact algorithm for optimal split point finding with histogram-based splitting, which reduces $O(\cdot)$ of finding the splitting points. Similarly to XGBoost, the histogram-based algorithm buckets continuous features and uses the bins to build the feature histograms, as seen in Figure 3.3.

Additionaly, it employs Gradient One-Side Sampling (GOSS), a technique that keeps instances with larger gradients and randomly downsamples instances with smaller gradients; naturally, observations with larger gradients are more under-trained, and thus contribute more to the splitting criterion. GOSS simultaneously samples the data without skeweing the distribution and provides superior computational characteristics.

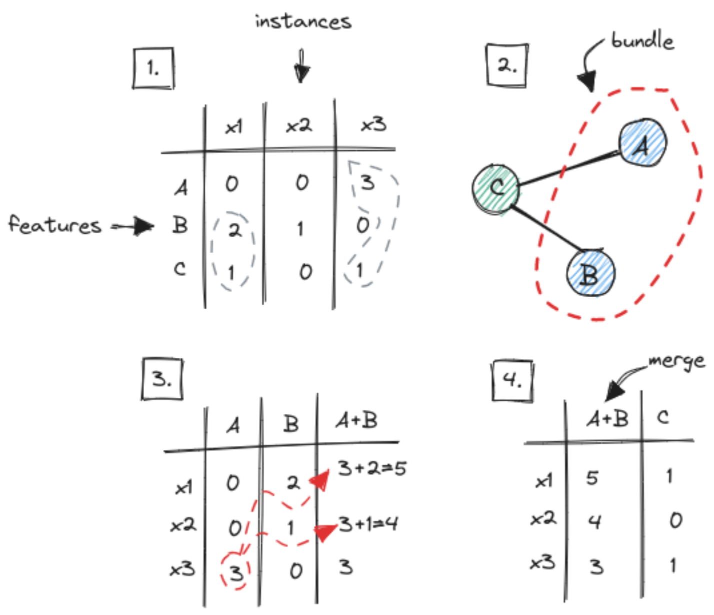
LightGBM incorporates bundling mutually exclusive features[4] into individual features; this is achieved by expressing the feature bundling as a graph coloring problem (where features are vertices and edges are added for every two non-mutually exclusive features) and solving the graph via a greedy algorithm. After building the feature bundles, these are merged into individual features by transforming feature values through an offset (the upper bound of each feature) so that different features can reside in different discrete bins. Figure 3.4 illustrates how Exclusive Feature Bundling (EFB) works. EFB is useful for reducing dimensionality in settings dominated by redundant features – such as tabular data.

LightGBM also includes a native technique for encoding categorical features built upon the idea of grouping (or encoding) categorical values for minimum variance within groups [Fisher, 1958]. LightGBM then sorts each categorical feature histogram according to its accumulated values[5] and finds the best split on the sorted histogram.

CatBoost CatBoost [Dorogush et al., 2018] is a GBDT that uses Oblivious Decision Tree (ODT)s [Kohavi, 1994] as the weak learners. An ODT uses the same

[4]Features taking non-zero values simultaneously

[5]Ration between the sum of the gradient and the sum of the hessian



	x1	x2	x3
A	0	0	3
B	2	1	0
C	1	0	1

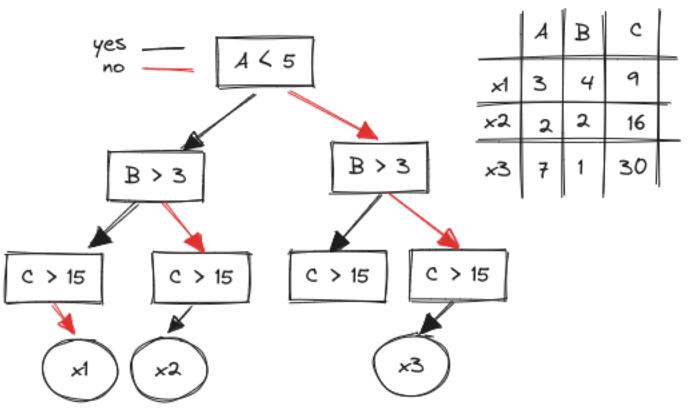
	A	B	A+B
x1	0	2	3+2=5
x2	0	1	3+1=4
x3	3	0	3

	A+B	C
x1	5	1
x2	4	0
x3	3	1

Figure 3.4: LightGBM's Exclusive Feature Bundling diagram. LightGBM forms a graph coloring problem where features are vertices and edges are traced between non mutually exclusive features – unconnected vertices (features) are bundled together. LightGBM merges bundles by placing features in different bins.

splitting criterion across the internal nodes of a given level, which renders more balanced (symmetric across a vertical axis) trees and combats overfitting. Figure 3.5 illustrates an ODT. Similarly to Stochastic Gradient Boosting (SBG) [Friedman, 2002], CatBoost incorporates bagging into the boosting procedure by generating random permutations of the data and computing the pseudo-residuals on "new" instances.

CatBoost provides native support for categorical features that is similar to target encoding (replacing a category with the average target value for that category). CatBoost goes beyond by leveraging the data permutations of the dataset and applying target encoding using only instances placed before a given instance (which reduces



	A	B	C
x1	3	4	9
x2	2	2	16
x3	7	1	30

Figure 3.5: Oblivious Decision Tree diagram. Internal nodes at the same depth use the same splitting feature and point.

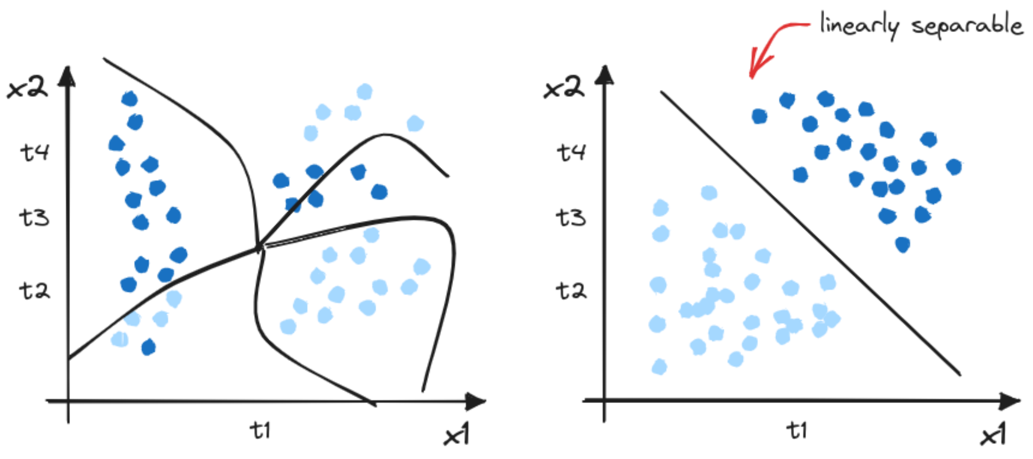


Figure 3.6: Neural Network decision manifold for tabular data (left) and after an (ideal) transformation through the hidden layers (right). $t_1, \cdots, t_4$ are the threshold values for features $\{x_1, x_2\}$.

overfitting).

3.3 Deep Learning

DL [Goodfellow et al., 2016; Russell, 2010] is built upon the idea of learning a complex concept by expressing it in terms of simpler concepts. Indeed, DL is deeply connected with learning representations from data, as opposed to traditional machine learning which often requires some sort of manual feature engineering: DL affords the possibility to save labour and learn compact representations of features directly from raw data.

At the core of DL lies the neural network. The atomic building block of a neural network is the *neuron*, a mathematical unit that receives some input, performs a computation, and outputs a real value based on an arbitrary activation function – a measure of neuron "strength" or "relevance". A Feed Forward Neural Network (FFN) connects neurons in a forward fashion, so that each neuron is a function of its current input only and forwards its activation downstream to the next neurons. These neurons are grouped into a series of layers chained together, starting on the input layer and ending on the output layer. Every layer in-between is refered to as an *hidden layer*, and these receive inputs from the preceding layer and output an activation onto the next layer. The hidden layers are defined by weights that connect neurons between layers and an additional bias term.

In the context of supervised learning, a FFN is trained by minimizing a loss function via gradient descent through backpropagation, i.e. propagating the error backwards through the hidden layers. The weights are adjusted by the negative gradient of the loss function w.r.t to the weights, so neurons with larger gradients (errors) have larger weights. Neural networks usually require some form of regularization to avoid overfitting, which is the case of the learning rate which controls the step size (alongside the negative gradient).

The power of the hidden layers resides in the ability to manipulate and transform data such that it becomes linearly separable, as illustrated in Figure 3.6. The output layer of a neural network is essentially a linear classifier that traces an hyperplane. However, on the original input, its decision boundaries appear non-linear as a result of backtracking all the transformations and non-linearities performed by the hidden layers. Nonetheless, tracing non-linear boundaries for data that is best described by piece-wise constant functions seldom yields the best result – at least, without proper regularization [Kadra et al., 2021; Shavitt and Segal, 2018]. Indeed, we hint at a possible cause for the performance gap: the neural network's inductive bias stating that "data exhibits linear relationships" doesn't hold for tabular data wherein relationships between features manifest as irregular functions.

3.3.1 Deep Neural Networks

Deep Neural Networks (DNNs) [Goodfellow et al., 2016; Russell, 2010] are neural networks with multiple hidden layers, whose practical usefulness (among many) resides in learning non-linear representations. Utilizing deep architectures encodes a very general inductive bias in DNNs: the problem to be learned is complex but can be expressed as a composition of more abstract and simpler concepts.

DNNs are usually regarded as a general concept and serve as basis for the development of complex architectures with different types of layers (e.g. convolutional and recurrent layers), which compose the building blocks of many widely used deep architectures in modern applications, such as Convolutional Neural Networks (CNNs) (e.g. image processing), Recurrent Neural Networks (RNNs) (e.g. modelling sequential data) and Transformers (e.g. natural language processing).

Multi Layer Perceptron The simplest example is the Multi Layer Perceptron (MLP): a DNN composed of linear hidden layers. A linear layer performs an affine transformation via a learnable weight matrix followed by a non-linear "squashing" function:

$$\begin{aligned} \texttt{Linear}(x) &= \mathbf{W}x + \mathbf{b} \\ \phi(x) &= a(\texttt{Linear}(x)) \end{aligned} \tag{3.1}$$

This type of DNN block is also known as a Fully Connected Neural Network (FNN) or Fully Connected (FC) layer, as $\texttt{Linear}(x)$ is a function of all the inputs, i.e. all neurons are connected. The activation function $a(\cdot)$ is a matter of design choice, being the most common the REctified Linear Unit (ReLU):

$$\texttt{ReLU}(x) = \max(0, x)$$

Training A MLP is usually trained via backpropagation with Stochastic Gradient Descent (SGD). Training a MLP (and other neural network blocks for that matter) is difficult, given that vast number of parameters (weights and biases[6]) one usually faces. Training is usually performed across multiple batches of the input data, which is usually a matter of finding a good balance between using enough samples for accurate estimates and keeping computational overhead low.

Normalization Because optimization is done via SGD, neural networks usually get stuck in *plateaus* and local minima derived from the gradients approaching very small values – this is known as the *vanishing gradient*. Practitioners often employ normalization layers to address the vanishing gradient problem. Among many, a common approach is to use a `Batch Normalization` layer between hidden layers to normalize data over the mini-batches.

Regularization Neural networks are prone to overfitting if encouraging some weights to take large values or to saturate[7]. Practitioners often employ regularization techniques to help neural networks avoid overfitting. Among many, a common approach is to use a `Dropout` layer after activation functions; `Dropout` randomly "zeroes" elements of the input space with probability ρ. Early stopping is also commonly used and consists of stopping training if `patience` (a threshold) epochs without improvement on the validation set have passed.

Non-linear Activations Non-linear activation functions allow neural networks to model non-linear interactions in data by "squashing" neuron outputs. $\texttt{ReLU}(x) = \max(0, x)$ is amongst the most widely used: although `ReLU` generally learns highly sparse representations (by pushing negative neurons towards saturation), it also solves the problem of the vanishing gradient.

Recent advancements extend activation functions to actual DNN blocks, such as the Gated Linear Unit (GLU) [Dauphin et al., 2017] and the REctified Gated Linear Unit (ReGLU) [Shazeer, 2020]:

$$\begin{aligned}\texttt{GLU}(x) =& \sigma(\texttt{Linear}(x)) \circ \texttt{Linear}(x) \\ \texttt{ReGLU}(x) =& \texttt{ReLU}(\texttt{Linear}(x)) \circ \texttt{Linear}(x)\end{aligned}$$

Amongst the relevant works, TabNet uses the `GLU` activation and FT-Transformer uses `ReGLU`.

Initialization Schemes Initialization of the neural networks parameters, particularly of weights, is itself an active field of study. Notably, improper initialization may lead towards *plateaus* and local minima. Typically, the weights are initialized by a random uniform distribution, although some novel schemes such as Xavier [Glorot and Bengio, 2010] and Kaiming [He et al., 2015] remedy some issues of DNNs:

[6]This refer to the bias term in a linear layer

[7]Approach values ≈ 0

$$
\begin{aligned}
\text{(Xavier) } \mathbf{W} &\sim \mathcal{U}\left[-\frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j+n_{j+1}}}\right] \\
\text{(Kaiming) } \mathbf{W} &\sim \mathcal{U}\left[-\frac{\sqrt{3}}{\sqrt{n_j}}, \frac{\sqrt{3}}{\sqrt{n_j}}\right]
\end{aligned}
$$

where $\mathbf{W}$ is the weight matrix and n_j is the number of neurons in the j-th layer. Indeed, Xavier initialization alleviates saturation of the top layers for sigmoidal activations. However, Xavier initialization fails to converge for deeper networks and doesn't take into account the non-linearities of `ReLU` activations (and its variants). TabNet uses Xavier initialization and FT-Transformer uses Kaiming initialization.

Embeddings An embedding is a vector representation of another vector, commonly used in DL to encode certain variables. Embeddings can have arbitrary sizes, ranging from one-dimensional embeddings – scalars – to n-dimensional embeddings. At its core, an embedding layer is a learnable lookup table for storing vector representations, e.g. of categorical features.

An embedding layer differs from a linear layer in the sense that an embedding layer doesn't perform any operation on the input (it works as lookup table), whereas a linear layer performs matrix multiplication. Thus, embeddings are generally used for encoding categorical features and the weights of the embedding (the vector representation) are learned via backpropagation. Algorithms such as TabNet and FT-Transformer leverage embedding layers to encode categorical features.

3.3.2 Known Implementations

The published works proposing novel architectures can be categorized into 3 distinct approaches: i) attention-based ii) transformer-based and iii) differentiable trees. We chose an implementation from each category based on algorithm intricacy and the recent popularity among research works about the performance gap between GBDT and DL.

Multi Layer Perceptron Regarding the MLP, we adopt an architecture identical to the one formalized in [Gorishniy et al., 2021]:

$$
\begin{aligned}
\texttt{MLP}(x) &= \texttt{Linear}(\texttt{MLPBlock}(...(\texttt{MLPBlock}(x)))) \\
\texttt{MLPBlock}(x) &= \texttt{Dropout}(\texttt{ReLU}(\texttt{Linear}(x)))
\end{aligned}
\tag{3.2}
$$

In a `MLPBlock`, the `Linear` layer performs an affine transformation as in Equation 3.1 and the `Dropout` layer acts as a regularization mechanism.

Fundamentally, each `MLPBlock` is performing some form of non-linear transformation over x of the form $\phi : \mathbb{R}^m \to \mathbb{R}^n$, whilst the last `Linear` layer – referred to as the *head* – performs a transformation of the form $\phi : \mathbb{R}^m \to \mathbb{R}^1$ (in a binary classification task). Conceptually, we can think of the MLP as a sequence of transformations that manipulate the input space x, where each `MLPBlock` is learning a different representation of x and the *head* is essentially performing logistic regression on $\mathbb{R}^n$.

In a MLP, the sequence of `MLPBlock`s manipulate the input space into one wherein data is linearly separable and the *head* traces a hyperplane in $\mathbb{R}^n$ space, just as the one portrayed in Figure 3.6. Geometrically, these blocks perform a change of basis by representing the same set of data points on a different coordinate system – matrix $\mathbf{W}$ is the coordinate system matrix for which the i-th column is the basis vector of the i-th dimension.

Naturally, the representations learned at intermediate layers may lack the information content initially present in the original coordinate space. This is especially the case for tabular data, wherein features carry individual meaning (e.g. `transaction type`) and exhibit different statistical properties, which are not preserved in these representations. Indeed, some representations may go as far as to misrepresent the irregular patterns in tabular data via linear (and non-linear) dependencies. This is not necessarily bad, albeit not ideal for tabular data wherein features usually follow multi-modal distributions and generally encode prior knowledge about the concept – homogeneous data (e.g images) largely benefit from representation learning by encouraging the learning of latent features.

TabNet TabNet [Arik and Pfister, 2021] is a DNN adaptation for tabular data based on sequential top-down attention for reasoning which features to use at each decision step. TabNet performs instance-wise feature selection by learning a sparse mask (selecting just a subset of features) which enables the focus on specific parts of the input sequence at each decision step – TabNet's sequential processing architecture allows for a greater representational capacity.

Part of TabNet's novel approach stems from the aggregation of decision outputs at each decision step to build tree-like decision manifolds: a linear transformation defines boundaries via a bias factor and a ReLU activation "zeroes" the regions. Indeed, TabNet claims to trace *almost* parallel splits via approximation of hyperplane boundaries.

TabNet employs a learnable embedding layer to natively encode categorical features. TabNet claims one-dimensional embedding (i.e. scalars) to suffice, although higher-dimensional embeddings may improve performance at the expense of interpretability.

We adopt the architecture formalized in the original paper [Arik and Pfister, 2021]:

$$\begin{aligned} \texttt{TabNet}(x) &= \texttt{Encoder}(\texttt{Embedder}(x)) \\ \texttt{Embedder}(x) &= \left[x^{num}, \left[\texttt{Embedding}_1(x_1^{cat}), ..., \texttt{Embedding}_t(x_t^{cat})\right]\right] \\ \texttt{Encoder}(x) &= \texttt{Step}(...(\texttt{Step}((\texttt{FT}(\texttt{BN}(x)))) \\ \texttt{Step}(x) &= \texttt{FT}(x \circ \texttt{AT}(x))) \end{aligned} \tag{3.3}$$

Figure 3.7 illustrates TabNet's architecture. The `Embedder` learns an embedding

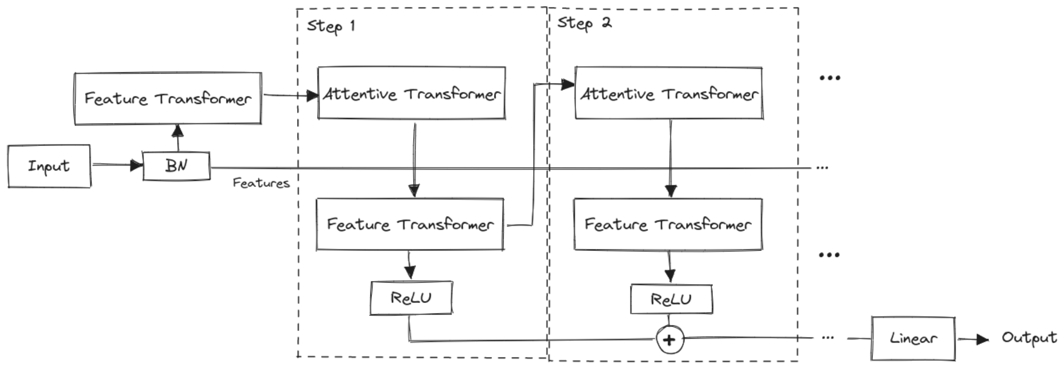


Figure 3.7: TabNet's architecture. All features are considered across decision steps, but AT learns a sparse mask for selecting just a subset of features which are processed by FT and yield the decision output. The decision output is aggregated at each step and passed through a Linear layer to compute the decision (adapted from [Arik and Pfister, 2021]).

layer Embedding for each categorical feature and numerical features are kept raw. BN stands for Batch Normalization, FT stands for Feature Transformer and AT stands for Attentive Transformer[8]. The decision output is built as follows:

$$\begin{aligned} \texttt{Out} &= \sum [\texttt{ReLU}(\texttt{Step}_1), ..., \texttt{ReLU}(\texttt{Step}_n)] \\ \texttt{Prediction} &= \texttt{Linear}(\texttt{Out}) \end{aligned} \tag{3.4}$$

In a similar fashion to the sequence of MLPBlocks in the MLP, the Encoder in TabNet's architecture is manipulating the input space x by performing a mapping $\phi : x \to x'$ wherein data becomes linearly separable. TabNet leverages a sequential architecture to reason different parts of the input at different steps: at each $\texttt{Step}_i$, the FT processes the features selected at $\texttt{Step}_{i-1}$ by the AT. The AT learns a multiplicative sparse mask that performs instance-wise sparse feature selection, which encourages the focus on a select subset of features:

$$\begin{aligned} \texttt{AT}(x) &= \texttt{sparsemax}(\texttt{BN}(\texttt{Linear}(x)) \circ \mathbf{P}) \\ \texttt{FT}(x) &= [[\texttt{Block}_1, ..., \texttt{Block}_{\texttt{n\_shared}}], [\texttt{Block}_1, ..., \texttt{Block}_{\texttt{n\_independent}}]] \\ \texttt{Block}(x) &= \texttt{GLU}(\texttt{BN}(\texttt{Linear}(x))) \end{aligned} \tag{3.5}$$

The sparse mask is balanced by the scale term $\mathbf{P}$ which weights how much a feature has been used. sparsemax [Martins and Astudillo, 2016] extends *softmax* by allowing sparse probabilities[9] which in turn encourages sparse selection of relevant features. TabNet concatenates layers shared across all steps and also step-specific layers to prevent poor generalization and hyper-focusing on certain parts of the input. The output of the FT is non-linearly transformed by a ReLU activation and is further

[8]the "Transformer" designation has no relation with the widely-known Transformer architecture [Vaswani et al., 2017]

[9]In *softmax*, probabilities never reach 0, only ≈ 0

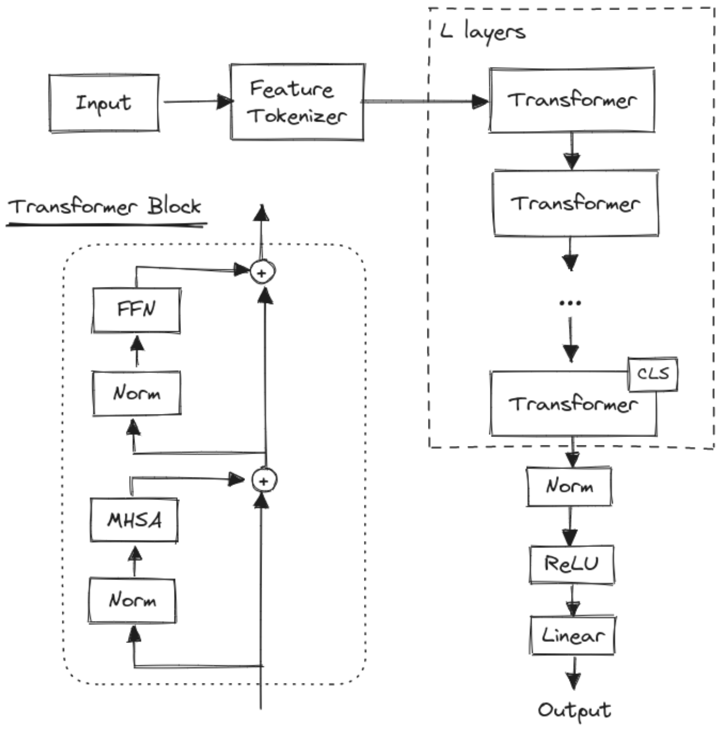


Figure 3.8: FT-Transformer's architecture. `FeatureTokenizer` learns an embedding and appends a [`CLS`] token to the sequence. L Transformer layers are applied to the data. The [`CLS`] token at the final Transformer layer is used for the output. (left) Transformer block architecture (adapted from [Gorishniy et al., 2021]).

aggregated across every `Step` before passing through a `Linear` layer which produces a tree-like decision manifold.

The presence of a feature selection mechanism hints at a possible inductive bias of TabNet similar to one held by tree-based methods: the data is dominated by redundant features and the target function depends only on a subset of relevant features. Indeed, using `sparsemax` and sparse regularization of the learnable mask encourages sparse selection which in turn favors the inductive bias above.

FT-Transformer FT-Transformer [Gorishniy et al., 2021] is an adaption of the Transformer architecture to tabular data that exploits DNN blocks in the context of tabular data. In a two-step process, FT-Transformer implements a learnable embedding layer for both numerical and categorical features so all features can be utilized by the Transformer layers [Gorishniy et al., 2022], and then applies a stack of Transformers to the processed input. The Transformer architecture here is key to focusing on different parts of the input sequence and modelling high-order feature interactions, due to the Multi-Head Self-Attention (MHSA) modules commonly used in language modelling tasks [Vaswani et al., 2017].

We adopt the architecture formalized in the original paper [Gorishniy et al., 2021]:

$$
\begin{aligned}
\texttt{FT-Transformer}(x) &= \texttt{Block}(...(\texttt{Block}(T_0))) \\
T_0 &= [[\texttt{CLS}]\,, \texttt{FeatureTokenizer}(x)] \\
\texttt{Block}(x) &= \texttt{Residual}(\texttt{FFN}, \texttt{Residual}(\texttt{MHSA},\ x)) \\
\texttt{Residual}(\texttt{Module},\ x) &= x + \texttt{Dropout}(\texttt{Module}(\texttt{LayerNorm}(x))) \\
\texttt{FFN}(x) &= \texttt{Linear}(\texttt{MLPBlock}(x))
\end{aligned}
\tag{3.6}
$$

Figure 3.8 illustrates FT-Transformer's architecture. Here, the `FeatureTokenizer` transforms the raw features into a d-dimensional embedding space and the [`CLS`] token is inserted into the embedded feature vector[10].

The `FeatureTokenizer` applies a transformation $\phi : \mathbb{R}^m \to \mathbb{R}^{m \times d}$ where m is the number of features and d is the size of the embedding layer. The embedding function is calculated as follows,

$$
\begin{aligned}
\phi(x_i) &= b_i + \begin{cases} x_i \circ W_i & \text{if } x_i \text{is numerical} \\ \texttt{OHE}(x_i) T_i & \text{if } x_i \text{is categorical} \end{cases} \\
\phi(x) &= [\phi(x_1), \ldots, \phi(x_m)] \in \mathbb{R}^{m \times d}
\end{aligned}
$$

where `OHE` outputs a one-hot vector for a given feature. The embedding of numerical features (further discussed in [Gorishniy et al., 2022]) enables its interaction with categorical ones inside the `MHSA` layers. In a similar way to [Song et al., 2019], the simple embedding scheme linearly transforms the numerical features (with the addition of a bias element) by a learnable, non-shareable matrix W_i and learns an embedding layer $T_i \in \mathbb{R}^{k \times d}$ where k is the cardinality of the i-th feature (with the addition of a bias element).

After that, L Transformer (`Block`) layers are applied. `FFN` consists of a traditional `MLPBlock` with ReGLU activation [Shazeer, 2020] instead of ReLU followed by a `Linear` layer. Only the final representation from the [`CLS`] token is considered for the output:

$$
\begin{aligned}
T_L &= \texttt{FT-Transformer}^{[\texttt{CLS}]}(x) \\
\texttt{Out} &= \texttt{ReLU}(\texttt{LayerNorm}(T_L) \\
\texttt{Prediction} &= \texttt{Linear}(\texttt{Out})
\end{aligned}
\tag{3.7}
$$

The Transformer blocks rely on the attention mechanism; the ability to focus on different parts of the input is key to learning meaningful representations that also provide good generalizations. TabNet uses sequential attention (focusing on different parts of the input at each decision step via sparse soft selection of relevant features) whilst FT-Transformer leverages adapted Transformer units (and the MHSA mechanism itself) to model high-order feature interactions.

[10]A token used to describe the end of a sequence in Transformer tasks

The key power of the FT-Transformer seems to reside on the embedding layer that transforms the raw input in a way that makes it suitable for the Transformer layers: as hinted by the authors in [Borisov et al., 2021], the underperformance in DL for tabular data may reside in inadequate encodings for the raw input features. There is no clear inductive bias in the FT-Transformer architecture; indeed, Transformers do not hold strong inductive bias, which relaxes any constraints on the space of solutions: this is a desired property as the absence of *strong* biases towards sub-optimal (or even bad) solutions allows for better generalization [Battaglia et al., 2018]. The lack of strong inductive bias could partially explain why FT-Transformer consistently performs better than its counter-parts [Gorishniy et al., 2021; Grinsztajn et al., 2022]. Notwithstanding, the lack of *appropriate* inductive bias could also explain it's underperformance compare to GBDT: tree-based methods are inherently biased towards data that is structurally similar to tabular data, thus the constraints over the space of solutions allow for *desirable* generalization whereas the constraints imposed by FT-Transformer prevent *poor* generalization.

NODE NODE [Popov et al., 2019] is a DNN architecture for tabular data based on differentiable ODTs trained via backpropagation. The core building block is the NODE layer, which consists of a set of differentiable ODTs that use an *entmax* (a generalization of *sparsemax*) transformation at the internal nodes, thus replacing a *hard* decision (i.e., smaller or greater) with soft sparse feature selection.

For each ODT of d depth, NODE selects only d features for partitioning and thus only learns d split features and thresholds. The tree output is defined as:

$$h(x) = R\left[\mathbb{H}(f_1(x) - b_1), \cdots, \mathbb{H}(f_2(x) - b_2)\right]$$

where $\mathbb{H}(\cdot)$ is the Heaviside step function and R is the d-dimensional matrix of responses containing all 2^d possible leafs or outcomes. Therefore, the response is either yes/1 (if $f_i(x) \geq b_i$) or no/0 (if $f_i(x) < b_i$).

However, as the output is not differentiable, NODE employs a differentiable feature choice and comparison operator. Indeed, NODE replaces $f_i(x)$ by a weighted sum of all m features wherein the weights are learned via a feature selection matrix $F \in \mathbb{R}^{d\times m}$ passed through `entmax`:

$$f_i(x) = \sum^{m} x_j \times \texttt{entmax}_\alpha(F_{ij})$$

where α is a parameter of `entmax` controlling sparsity. The feature selection matrix F encodes how much of a feature should be considered for each feature choice $f_i(x)$ – its values (weights) are learned via SGD which renders "feature selection" differentiable. The `entmax` transformation produces sparse probabilities which enables selecting only a subset of features.

NODE relaxes the Heaviside step function as the two-class `entmax` $\texttt{entmax}_\alpha([x, 0])$ which yields a "soft" decision. NODE computes the a choice vector as the soft decision function applied to the comparison operator scaled by a learnable parameter τ_i:

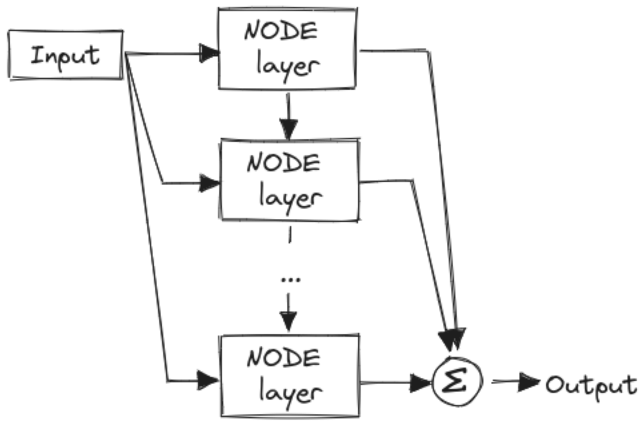


Figure 3.9: NODE's architecture diagram. NODE concatenates the input with the output of previous layers and feeds it into each layer. The output is an aggregation of the output of all layers. (adapted from [Popov et al., 2019]).

$$c_i(x) = \texttt{entmax}_\alpha \left(\left[\frac{f_i(x) - b_i}{\tau_i}, 0 \right] \right)$$

NODE computes a choice matrix $C(x) \in \mathbb{R}^d$ as the outer product of all $c_i(x)$ and yields the output as a linear combination of the response vector R weighted by the entries of the choice matrix $C(x)$. Finally, NODE concatenates the output of k trees to yield the output of the NODE layer.

Structure-wise, NODE implements a sequence of NODE layers wherein the input of each layer is a concatenation of the output of previous layers which allows the learning of shallow and deep representations of the input data. Figure 3.9 illustrates the architecture implemented in NODE. The presence of a differentiable function at the internal nodes allows NODE to be trained via SGD. NODE doesn't provide native support for categorical features and (internally) treats every feature as numeric.

The presence of a feature selection mechanism hints at a possible inductive bias of NODE similar to one held by tree-based methods: the data is dominated by redudant features and the target function depends only a subset of relevant features.

3.4 Summary

In this chapter, we presented the fundamental background concepts. We iterated over the foundational blocks such as decision trees, boosting and neural networks, delving deeper into GBDTs, one of the main family of algorithms currently popular in tabular data problems. We also described the general innerworkings of neural networks and DL. Finally, we described some known implementations of both GBDTs and DL algorithms that are commonly used in the most recent research works and competitions.

Chapter 4

Methodology

In the following chapter, we provide a description of the methods used in this research study that address the work hypotheses referred in chapter 2.

4.1 Data

We perform a study across 4 tabular datasets that span the 4 industry-specific use cases: (i) account takeover (ii) Anti Money Laundering (AML) (iii) transfer fraud and (iv) account opening. Although fairly flexible on the fraud rate, some restrictions to the dataset selection process are imposed, such as: (i) heterogeneous features (ii) labeled datasets (iii) documented datasets. Table 4.1 briefly describes the datasets selected and Table B.1 in Appendix B provides finer details about them.

Table 4.1: Datasets brief description: *subject* designs what each dataset row (instance) represents. Rich features are features incorporated via feature engineering processes.

Dataset	Use Case	Subject	Description	Fraud Rate
BB1	Account Takeover	Transfers and entities	Raw data + rich features	8%
RS1	Transfer Fraud	Transfers	Raw data + rich features	1%
CO1	Account Opening	Transfers and entities	Raw data + rich features	1%
WB1	AML	Transfers and entities	Raw data + rich features	3%

Although tabular in nature, every dataset is different and represents a different concept and a different use case under the domain of fraud detection. CO1 represents a transfer with the addition of some client data (e.g. age and credit risk score) and incorporates aggregate statistics and one-hot encoded features, i.e. dense features represented as sparse vectors. RS1 and WB1 contain records of transfers with the aggregation of statistics and behaviour profiles. BB1 comprises raw transfer data for which we aggregate profiling features of varying complexity. We split the BB1 dataset into multiple versions of profiling features with different degrees of complexity for evaluating the impact of feature engineering, as described in Chapter 5. Table B.2 in Appendix B provides finer details about the BB1 feature subsets.

The aforementioned profiling features were computed by domain experts in Feedzai under the scope of their respective projects. Nonetheless, these rich datasets still require some form of preprocessing before performing the experiments, which we describe in the next section.

4.2 Data Preprocessing

To avoid *boilerplating*[1], we developed a general and reusable workflow for data preprocessing, comprehending common processes such as cleaning, sampling and processing. Although feature engineering, i.e. computing statistical features across time windows and generating behaviour profiles, is commonly employed in fraud detection, the available datasets already endured feature engineering to some degree and we feel this kind of data modelling falls outside of the purpose of thesis.

Data Cleaning Data undergoes several cleaning steps, such as: i) validating the data schema (structure-wise cleaning); ii) removing samples/features with (many) missing values; iii) removing duplicates; iv) removing inconsistencies in data (value-wise cleaning).

Sampling Data is often very large in quantity and greatly imbalanced, which hinders the training process: stratified sampling is used to reduce dataset size (drop a portion of the legitimate transactions) whilst generally maintaining the original data distribution and retaining core valuable information.

Processing Data is often not in the desired format w.r.t the algorithms. The processing step involved: (i) type-setting; (ii) removing categorical features with high cardinality ($>$ 30 features); (iii) imputing missing values to -1; (iv) replacing infrequent categories with a placeholder category (see section 5.1 in Chapter 5 for further details); (v) encoding features. Encoding categorical features is algorithmic-dependent, whereas numerical features were normalized via a normally-distributed quantile transformation by scikit-learn's `QuantileTransformer`. To deal with the high number of features in the RS1 and WB1 datasets, we retrieve 30 features in decreasing order of feature importance (calculated by a LightGBM), for which the details can be found the Appendix B.

Encoding categorical features Most algorithms require categorical features to be encoded as numerical values, such as neural networks. Some algorithms require

[1]Needlessly repeating processes that can become modular

different encoding methods to provide reasonable results. To that extent, we further explore encoding methods for categorical features by evaluating algorithms and comparing respective native categorical support *vs.* an encoding method we propose. We describe this minor experiment in the next section and in more detail in Chapter 5.

Splitting Data is split into a train and evaluation set. CO1, RS1 and WB1 datasets were already split into train and evaluation sets. For BB1, we sort instances by the transaction timestamp in descending order and split 80/20. For tuning, we further split the train set into train and validation sets with a 80/20 ratio.

4.3 Experimental Approach

4.3.1 Algorithms

Regarding the algorithms to be evaluated, three Gradient Boosting Decision Trees (GBDTs) were selected: XGBoost [Chen and Guestrin, 2016], LightGBM [Ke et al., 2017] and CatBoost [Dorogush et al., 2018]. From the Deep Learning (DL) family, four algorithms were selected: a Multi Layer Perceptron (MLP), TabNet [Arik and Pfister, 2021], FT-Transformer [Gorishniy et al., 2021] and NODE [Popov et al., 2019]. The latter three algorithms span the different branches of recent breakthroughs in DL for tabular data, whilst MLP was chosen to provide a strong baseline.

4.3.2 Experiments

We further describe the experiments introduced under the work hypothesis in Chapter 2. We also describe the minor experimental work we perform prior to the core experiments.

Minor Experiments Foremost, we perform four minor experiments:

1. We compare aggregating results across multiple seeds *vs.* across multiple random samples of the test sets: we seek to validate if the results obtained are not coincidences of particularly good models and/or easy datasets;

2. We assess tuning times to understand if the experiments are feasible considering the available time and resource budgets;

3. We evaluate categorical encoding methods: we compare the native categorical support of some algorithms *vs.* an encoding method we propose;

4. We evaluate all algorithms across all datasets and report overall results.

Experiment 1. Considering the representational capacity of neural networks, we exploit the learned representations by DL algorithms to uncover possible inductive biases. In line with [Borisov et al., 2021], we argue that the representation learning performed by the hidden layers of Deep Neural Networks (DNNs) produces lossy representations – in the sense that information content is lost as a result of arbitrary transformations. To that extent, we seek to understand how these transformations are manipulating the data.

From an high-level standpoint, we can separate a DL into two distinct blocks: a representation learner and a linear classifier. Whilst the representation learner is tasked to learn compact representations of the data, the linear classifier traces an hyperplane to separate the data. We argue that the representations might lack some information present in the original input as a result of the transformations performed, either by compression (if the hidden layers reduce dimensionality) or by some other form. Lossy representations would partially explain the performance gap: whilst representation learning is useful for homogeneous data, it may be detrimental for tabular data by producing lossy representations of the input data. To that extent, we train DL algorithms and retrieve intermediate representations at different layers. Next, we evaluate tree-based methods on the learned representations to understand their behaviour: we expect tree-based methods to perform worse on all representations and possibly converge to the performance of DL algorithms on the representations fed to the linear classifier (the output layer of DL algorithms).

We delve further into this experiment in chapter 5 and discuss the obtained results.

Experiment 2. Considering the sensitivity of tree-based methods to feature engineering, we exploit the use of profiling features of varying complexity to uncover possible inductive biases. Tree-based methods often require some sort of feature engineering to encode linear and non-linear relationships in data: trees cannot compute $a > b$ and usually require a transformation such as $a - b > 0$.

Neural networks do not rely as much on feature engineering given their representational capacity and the ability to learn representations – to learn features. Indeed, we argue tree-based methods to be more sensitive to feature engineering in comparison to neural networks. To that extent, we compare tree-based methods with DL algorithms under different feature sets with varying degrees of complexity to understand their behaviour: we expect the performance gap to be smaller in the absence of rich features and larger when leveraging rich features. Additionally, we expect feature engineering to elevate the performance of all algorithms.

We delve further into this experiment in chapter 5 and discuss the obtained results.

4.4 Tuning

Regarding training, we use Optuna[2] – an out-of-the-box hyperparameter optimization framework – to perform hyperparameter tuning for every dataset-model pair.

[2]https://optuna.org/

We use the Tree-structured Parzen Estimator (TPE) [Bergstra et al., 2011] optimization algorithm which is proven to be substantially more efficient than random search. We set the tuning budget by the number of optuna trials at 100 for tree-based algorithms and 50 for DL algorithms. We provide the hyperparameter search spaces for all algorithms in Appendix C.

4.5 Evaluation

Regarding evaluation, we use "Recall @ n-th percentile" considering the low target densities present in the available datasets. In short, it predicts the positive class for the n-th percentile of the highest positive-class model scores. The n-th percentile largely depends on the dataset and the target density.

1. Sort the positive-class model scores p in descending order;
2. Predict 1 for the first $\frac{n}{100} \times |p|$ elements and 0 for the remaining elements.

Using "Recall @ n-th percentile", where n = Fraud Rate, affords the possibility to see how far we stand from a perfect model; a perfect model would predict 1 for $\rho \geq n \times 0.01$ and 0 elsewhere: "Recall @ n-th percentile" would be 1 for this perfect model. If we use $n <$ Fraud Rate, a perfect model would have a "Recall @ n-th percentile" lower than 1, e.g. if n = Fraud Rate = 0.5, then the "Recall @ n-th percentile" would be 0.5; by fixing the n-th percentile at the value of the fraud rate, we know that any departure from 1 is due to the model performance, and not due to changes in the fraud rate in different datasets.

Outcomes For each tuned configuration, we evaluate multiple times to guarantee the outcomes are not lucky coincidences of particularly good models and/or easy datasets. We delve into this issue in chapter 5 by comparing two strategies: i) reporting results across multiple seeds ii) reporting across multiple random samples of the test. To model the underlying error associated with experimentation, we present results using boxplots, wherein the error bars also largely depend on the dataset and the target density itself.

Statistical Testing For statistical testing, we use the `Mann-Whitney U` test for evaluating the distributions of our results, i.e. Recall. The reasons are 4-fold: i) the groups are independent ii) the observations are independent iii) the observations are ordinal in scale iv) the sample size is small. The `Mann Whitney U` [Hart, 2001] test is a non-parametric test that compares two groups and indicates whether one group tends to have higher values than the other; if the groups happen to exhibit similarly shaped distributions, we can also infer if the median of one group is higher than the other.

Following a general formulation, we have:

$$H_0 \longrightarrow X \text{ and } Y \text{ have identical distributions}$$
$$H_1 \longrightarrow X \text{ and } Y \text{ are not identical}$$

We report the population median $\tilde{\mu}$ and p-value; if the p-value falls bellow the critical point (we use 0.05 under a 95% confidence bound for a sample size of ten) the results are statistically significant, i.e. one group tends to produce higher values.

Additionally, we also use the `Kruskal-Wallis` test. The `Kruskal-Wallis` test extends `Mann-Whitney U` to multiple population groups and tests whether *at least* one group tends to produce higher (or lower) values. We use the `Kruskal-Wallis` test in settings where we want to compare two or more samples. The formulation is similar to that of `Mann-Whitney U`:

$$H_0 \longrightarrow \text{all groups have identical distributions}$$
$$H_1 \longrightarrow \text{at least one group's distribution is different}$$

We report the population median $\tilde{\mu}$ and p-value; if the p-value falls bellow the critical point (we use 0.05 under a 95% confidence bound for a sample size of ten) the results are statistically significant, i.e. at least one group has a different distribution.

4.6 Summary

In this chapter, we detailed the underlying methodology to this thesis, which included a description of the available data and preprocessing tasks. Further on, we described the experimental work to be conducted alongside the algorithms to be evaluated. Finally, we iterated over the details about tuning, evaluation and statistical testing.

Chapter 5

Experiments

In the following chapter, we provide a description of the experiments and respective results.

5.1 Minor Experiments

Foremost, we performed some minor experiments to understand the available data, guarantee there is statistical evidence for the experiments, assess computational feasibility and report overall results. A full description of the software and hardware employed is included in Appendix A.

Seeds *vs* samples To assure that we work with adequate models, we need to validate if the attained results are not coincidences of particularly good models and/or easy datasets. Indeed, we ought to exploit the main source of variance in the results – either the data or the models. To that extent, we compare two strategies: i) tuning an algorithm ten times with different initial seeds, and reporting performance on the test set across the ten seeds ii) tuning an algorithm once on one seed, and reporting performance across ten disjoint samples drawn at random from the test set. The latter corresponds to drawing equal-sized, disjoint samples at random from the test set, whilst still maintaining the same target density across samples. For that, we consider LightGBM and XGBoost and the dataset CO1. For both algorithms, we perform hyperparameter tuning with ten different seeds (the hyperparameter search space can be found in Appendix C). We let training run until `early_stopping_rounds` without improvement on the validation set have passed. For each, we evaluate i) on the whole test set, across ten different seeds and ii) on one seed, across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

As seen in Figure 5.1, there is a greater variance when reporting results across disjoint samples drawn at random from the test set compared to reporting across different seeds: this suggests that varying the initial seed has no significant bearing on the performance model, as the dispersion of the results is fairly low – the algorithms are able to achieve similar results despite the initial conditions. As previously

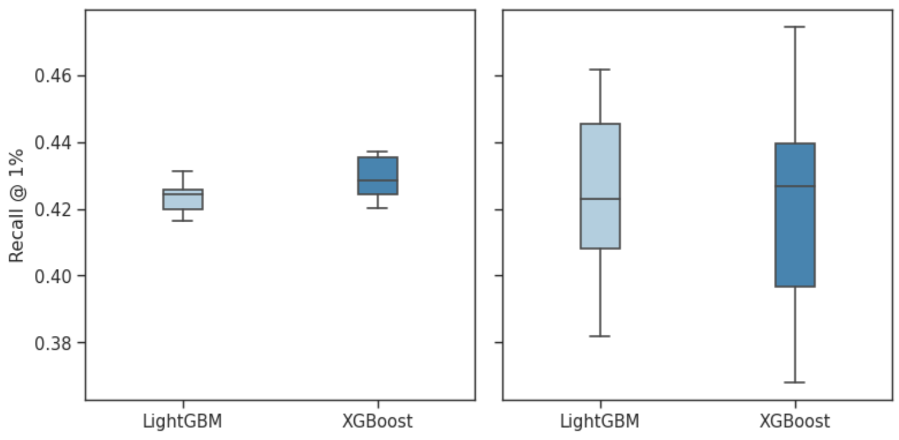


Figure 5.1: Performance of LightGBM and XGBoost (CO1 dataset, Recall @ 1%) reported across ten different seeds on the whole test set (left), and on one seed across ten disjoint samples drawn at random from the test set (right).

mentioned, the dispersion of the distribution depends on the target density of the dataset ($\approx 1\%$ for the CO1 dataset). We observed that the target density is similar across the random samples with only minor fluctuations; these minor fluctuations can, however, induce high variations in performance due to the different degree of class imbalance across all samples.

The strategy that yields greater variance is the most adequate for us as we are most likely dealing with the worst case scenario, i.e. the distribution is ample enough as to encompass most variation in the models. Because the variance across seeds is so low (compared to the variance across samples), we need not to tune the algorithms several times. As such, and for the remainder of the experimental work, we resort to reporting results across disjoint samples drawn at random from the test as a measure of dispersion.

Training Budget Initially, we evaluate training times to understand the feasibility of the algorithms, supported on the following criteria: i) training is efficient/fast ii) training overhead is small iii) training is sustainable on CPU-only hardware (a full description of the software and hardware used can be found in Appendix A).

We consider all algorithms and the dataset CO1. We perform hyperparameter tuning for all algorithms except for TabNet, FT-Transformer and NODE, for which we configure the starting parameters to the "default" parameters provided in the respective papers (and make some adjustments to comply with our computational resources). The hyperparameter search space can be found in Appendix C. Due to time limitations, we are not able to extensively tune TabNet, FT-Transformer and NODE, so we set a training budget of ten hours for each, and record the best trial under this budget. We let training run until `patience` epochs (for Deep Learning (DL)) or `early_stopping_rounds` (for tree-based) without improvement on the validation set have passed. For each, we evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$ and respective training times.

As seen in Figure 5.2, tree-based methods achieve better performance under a con-

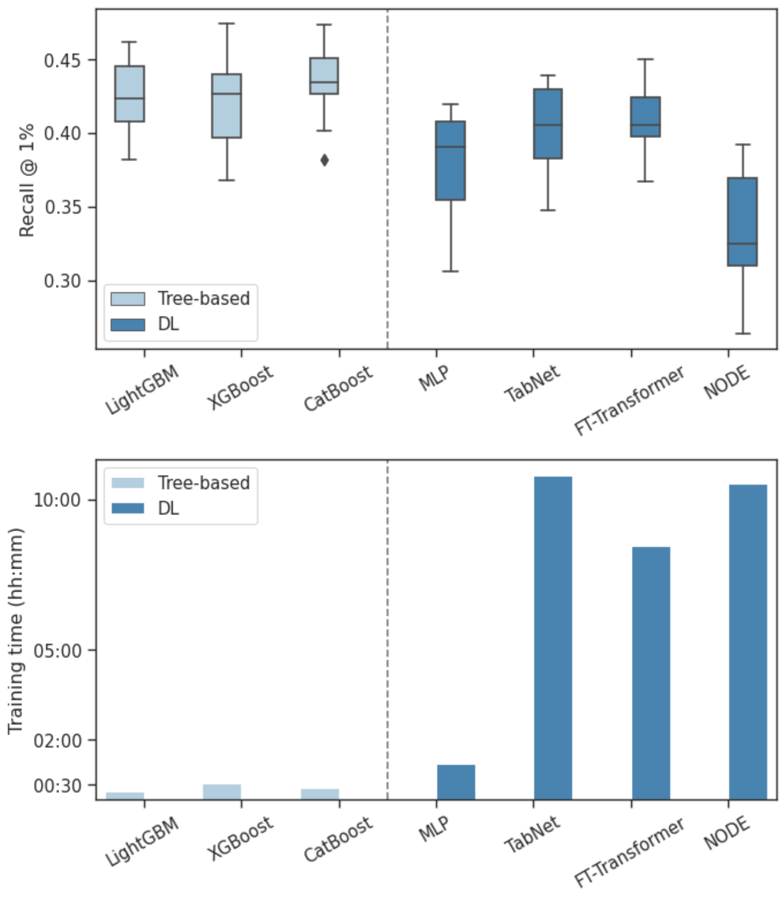


Figure 5.2: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set (top) and respective training times (bottom).

strained time budget. Only the tree-based methods and the Multi Layer Perceptron (MLP) were tuned until completion, whilst TabNet, FT-Transformer and NODE finished tuning before reaching 100 trails, ending at trials 25, 1 and 18, respectively. Naturally, these algorithms are not able to converge under the limited time budget of ten hours as we are not able to extensively tune them; on top of that, the dataset at issue (CO1) comprises 96 features, which considerably slows down training. Considering DL algorithms largely benefit from proper hardware and comfortable training budgets, we suspect that, given enough time, DL algorithms would achieve more competitive results with the tree-based methods. Indeed, the authors in [Gorishniy et al., 2021; Popov et al., 2019] claim the implementations of FT-Transformer and NODE to be memory-inefficient and to require substantial resources to present reasonable results.

Notwithstanding, in [Grinsztajn et al., 2022], the authors conduct a similar study to ours and show that tree-based models consistently outperform DL algorithms independently of the training budget, further suggesting that tight training budgets have no bearing on the performance gap.

Categorical Support Several of the selected algorithms provide native support for categorical features such as: LightGBM, CatBoost, TabNet and FT-Transformer. For these, we evaluate if the native categorical support is superior to traditional encoding methods, such as the one we employ for the remaining algorithms (XGBoost, MLP and NODE). For algorithms which don't natively support categorical features, we propose `Top-N OHE`, a variant of One-Hot Encoding (OHE):

Algorithm 2: Top-N OHE

```
Input: input $X$, min_freq and list of categorical columns cols
begin
    threshold ⟵ min_freq $\times$ len($X$);
    categories ⟵ {};
    for $col \in cols$ do
        counts ⟵ count_unique_values($X$, col);
        frequent ⟵ counts if counts $>$ threshold;
        categories ⟵ categories + (frequent, 'other');
    end
    for $col \in cols$ do
        known_cat = unique($X$, col);
        not_seen ⟵ known_cat $\notin$ categories;
        $X$[col] ⟵ replace(not_seen, 'other');
    end
    $X$ ⟵ OHE($X$);
end
```

This variant of `OHE` first replaces infrequent categories by a placeholder "other", where infrequent is defined by `min_freq` (e.g. if `min_freq` $= 0.2$, then all categories with less than 20% frequency are replaced by the "other" category). Unseen categories – categories not seen in the training data – are also replaced by the "other" category. After that, `OHE` is applied.

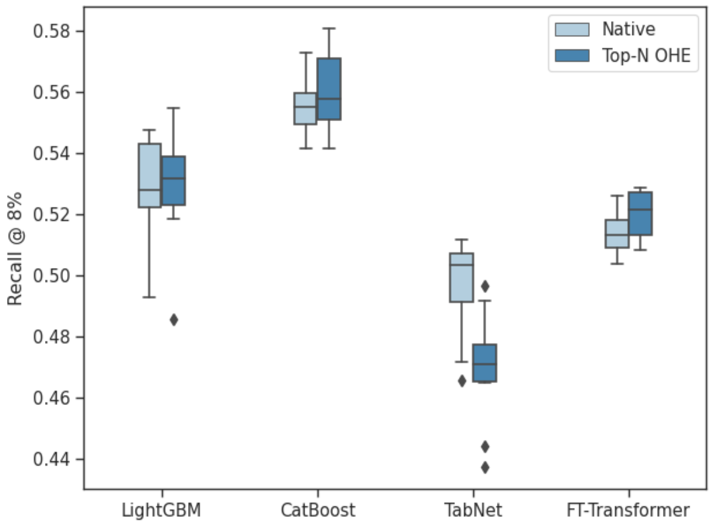


Figure 5.3: Performance (BB1 dataset, Recall @ 8%) reported across ten disjoint samples drawn at random from the test set. Results are shown for native categorical support and `Top-N OHE`.

Therefore, we consider the LightGBM, CatBoost, TabNet and FT-Transformer algorithms and the BB1 dataset. For each, we evaluate two configurations: i) native categorical support ii) `Top-N OHE`. We perform hyperparameter tuning for all algorithms (the hyperparameter search space can be found in Appendix C) and let training until `patience` epochs (for DL) or `early_stopping_rounds` (for tree-based) without improvement on the validation set have passed. For each, we evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

Figure 5.3 indicates that `Top-N OHE` can provide competitive results. LightGBM and CatBoost's native methods seem to provide as good results as `Top-N OHE`. TabNet seems to perform better with natively encoded categoricals whereas FT-Transformer seems to perform worse. TabNet employs a single-scalar learnable embedding (by default) which suggests TabNet is transforming categorical features into ordinal variables: embeddings are learned in a way that places similar categories close together in the latent space. FT-Transformer, however, applies a higher-dimensional embedding to categorical features and concatenates with an equal-sized embedding of numerical features. The key differences between TabNet and FT-Transformer lie in the dimensionality of the embedding and in the embedding of numerical features: FT-Transformer essentially applies a `Linear` layer.

Although both TabNet and FT-Transformer employ embedding modules for categorical features, only TabNet seems to benefit from this mechanism. One-hot encoded features regularize the number of categories per features and increase the degrees of freedom describing the input space, which increases overall sparsity. As FT-Transformer considers one-hot encoded features to be numeric, the embedding may be acting as dropout by forcing values to be 0 along some directions of the input space (regularization). Additionally, the size of the embedding layer (e.g. 128 for the FT-Transformer with native categorical support) may encourage the model to have high capacity and, thus, to overfit. On the other hand, the FT-Transformer with `Top-N OHE` uses an embedding layer of size 64.

Nonetheless, we argue results could vary if we left all categories instead of replacing the infrequent ones. Indeed, we consider a more thorough exploration of categorical features as valuable future work in Section 6.2. Forthcoming, we use encode categorical features with `Top-N OHE` variant for XGBoost, MLP and NODE, but consider native categorical support for LightGBM, CatBoost, TabNet and FT-Transformer.

Overall results Moving forward, we evaluate all algorithms on all datasets considering the conditions previously mentioned above and present overall results. We consider the raw version only for the BB1 dataset, for which the details can be found in Table B.2 in Appendix B.

Table 5.1: Overall performance for all algorithms. We report median Recall at the n-th percentile and interquartile range (in parenthesis) across ten disjoint samples drawn at random from the test set. The n-th percentile is different for each dataset and thus is stated in parenthesis next to the dataset name. For each dataset, the best result is highlighted in **bold**. We considered the raw version for the BB1 dataset.

	BB1 (8%)	RS1 (1%)	CO1 (1%)	WB1 (3%)
LightGBM	0.528 (0.021)	0.549 (0.093)	0.423 (0.037)	0.390 (0.017)
XGBoost	**0.558** (0.011)	**0.613** (0.205)	0.427 (0.043)	**0.405** (0.025)
CatBoost	0.555 (0.010)	0.464 (0.135)	**0.435** (0.024)	0.384 (0.011)
MLP	0.512 (0.016)	0.286 (0.108)	0.390 (0.053)	0.341 (0.016)
TabNet	0.503 (0.016)	0.394 (0.134)	0.405 (0.047)	0.344 (0.014)
FT-Transformer	0.513 (0.009)	0.444 (0.252)	0.406 (0.026)	0.367 (0.008)
NODE	0.447 (0.015)	0.321 (0.121)	0.325 (0.059)	0.340 (0.025)

As seen in Table 5.1 and Figure 5.4, tree-based methods consistently outperform DL algorithms across all datasets. As previously mentioned, such might be due to an underlying inability to properly deal with tabular data (at least, as well as tree-based methods deal). Nonetheless, a poorer performance among DL models seems to go in line with the relevant papers in the area see [Borisov et al., 2021; Grinsztajn et al., 2022].

The MLP proves to be an adequate baseline among DL models, as it's relatively fast to train and is able to provide a reasonable performance. FT-Transformer consistently outperforms other DL algorithms, sometimes, by a significant margin. TabNet represents the best trade-off between performance and computational overhead, although one questions the true potential of FT-Transformer if given enough resources and time. NODE is the lowest performing algorithm, which may seem counter-intuitive considering it's the only DL algorithm to benefit from an ensemble-like architecture: as is the case for FT-Transformer, NODE is very expensive to train, and the available training budgets may not be enough to infer proper conclusions about their performance. Nonetheless, the results go in line with [Grinsztajn et al., 2022], which shows tree-based methods to perform consistently better, and [Gorishniy et al., 2021], which shows FT-Transformer to be provide the best results across DL models.

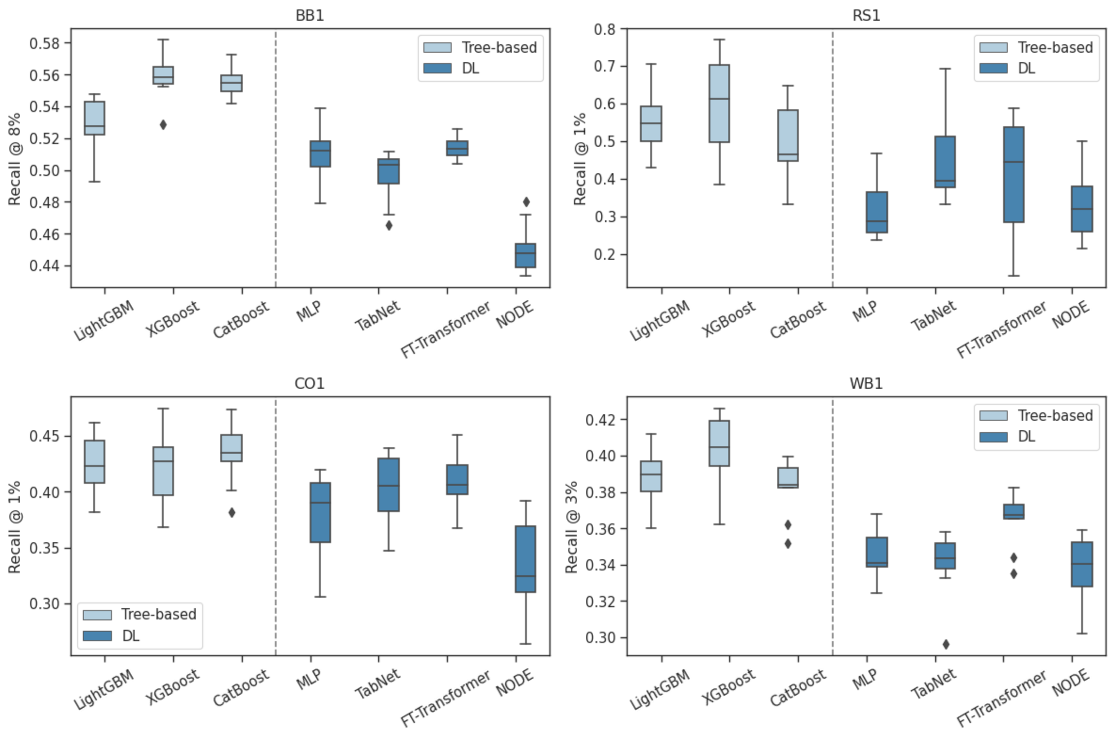


Figure 5.4: Overall performance for tree-based (in light blue) and DL (in dark blue) algorithms. The n-th percentile is different for each dataset.

5.2 Experiment 1

As part of the first experiment, we explore the representational capacity of neural networks in DL. Following the rationale in [Borisov et al., 2021], representation learning performed by the hidden layers of Deep Neural Networks (DNNs) encodes representations wherein some information might have been in lost the process.

To that extent, we explore the representations by the MLP, TabNet and FT-Transformer. First, we evaluate tree-based algorithms on the learned representations to investigate if a possible compression of the original input into a dimensional space with less relevant features causes performance to drop. Secondly, we compare the transformations performed by the sequence of DNN blocks of these algorithms with unitary transformations (e.g. rotations and/or reflections) to disprove these blocks are performing distance-preserving transformations, i.e. isometries. Finally, we relate the learned representations with the innerworkings of DL algorithms to uncover possible inductive bias of these algorithms causing the performance gap between tree-based and DL algorithms.

5.2.1 Multi Layer Perceptron

Formally, we recall the aforementioned architecture of a MLP and search for a possible compression of the original input in the intermediate representations. Effectively, compression occurs when there is a reduction in dimensionality performed by lower-dimensional hidden layers; transformation by higher-dimensional hidden layers do

Table 5.2: Best MLP model parameters on the CO1 dataset.

Parameters	
no. of layers	2
layers size	$16, 8$
dropout	≈ 0.0004

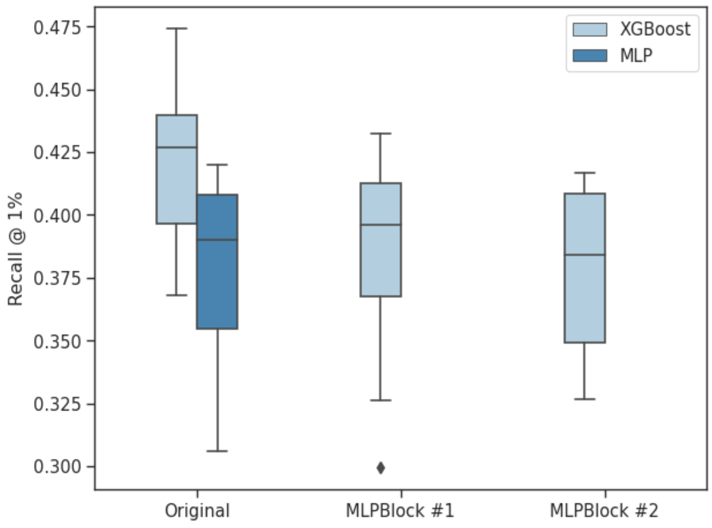


Figure 5.5: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. MLP and XGBoost performance on the original dataset (left). XGBoost performance on the representations retrieved from the first (middle) and second `MLPBlocks` (right).

not perform compression. For the MLP, we focus on the case when the hidden layers are of lower dimension than the original input. We evaluate XGBoost in the learned representations and compare results: we argue there will be a performance degradation pronounced by the amount of reduction in the size of each layer, i.e. greater compression implies greater performance degradation. Additionally, we argue the performance on the last representation matches the performance of the MLP in the original input.

We consider the best-performing MLP on the CO1 dataset and retrieve its representations: the hyperparameters can be found in Table 5.2. We perform hyperparameter tuning for XGBoost (the hyperparameter search space can be found in Appendix C) and let training run until `early_stopping_rounds` without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

According to Figure 5.5, performance seems to degrade within each representation, which goes in line with our initial guess: the learned representations are *likely* lossy. It is also evident that the greater degradation in performance happens at the first block: the original input contains 96 features and the first representation contains only 16. On the other hand, it's unclear if performance converges.

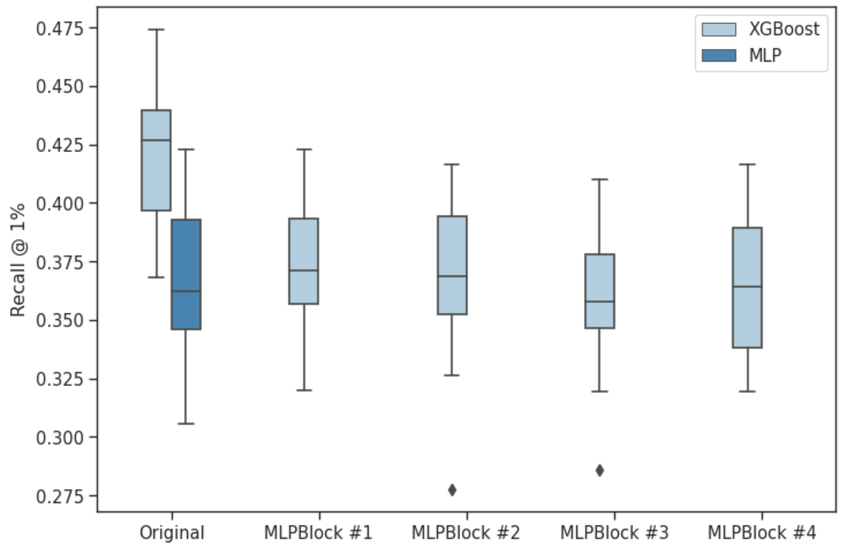


Figure 5.6: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. MLP and XGBoost performance on the original dataset (left). XGBoost performance on the representations retrieved from the first, second, third and fourth `MLPBlocks` (from left to right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the second representation ("Block #2") is identical to MLP performance on the original input.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.384, 0.390)$ p-value $\approx 0.850 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance doesn't converge.

The results above are inconclusive, so to further understand the impact of compression, we train a deep one-shot[1] MLP with four hidden layers of size 64, 32, 16 and 8, respectively, and without dropout. We repeat the same steps as we did for the shallow MLP above.

Figure 5.6 shows that performance degradation is much more pronounced in the first representation, further suggesting that the amount of degradation is influenced by the amount of dimensionality reduction. This seems to be the case for the first representation, as the following ones don't present further noticeable degradation: the first representation is a transformation of the original input, whilst the following ones "work" over an already transformed space and thus are somewhat similar between each other. Also, it's unlikely that performance degradation is a function of compression, as we can't confidently state that performance degrades from $\texttt{MLPBlock}_1$ to $\texttt{MLPBlock}_2$. Not only that, but the performance on the $\texttt{MLPBlock}_4$ seems to be slightly better than on the $\texttt{MLPBlock}_3$.

[1]Tuned for 1 trial only

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the original input is identical to XGBoost's performance on the first representation ("Block #1").

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.427, 0.371)$ p-value $\approx 0.007 \longrightarrow$ we reject the null hypothesis as results are significant. This suggests we have strong statistical evidence that performance degrades, i.e. that performance on the first representation is worse.

On the other hand, it's unclear whether performance degrades within each representation despite the compression of the space of representation.

Kruskal-Wallis Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on each representation ("Block #1" to "Block #4") are identical to each other.

$H_1 \longrightarrow$ At least one distribution of performance is different.

Test result: $\tilde{\mu} \approx (0.371, 0.369, 0.358, 0.364)$ p-value $\approx 0.781 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that XGBoost performs better (or worse) in a given representation, i.e. that performance degrades within each representation.

The results further reiterate that it's unlikely that compression *always* causes performance degradation. The hidden layers perform transformations to make data linearly separable – in this case, they find a coordinate system or basis in which the data is linearly separable. This kind of representation (or orientation) is not favorable for tree-based methods: as opposed to neural networks, tree-based methods can't draw hyperplanes; instead, tree-based methods build a decision manifold by tracing axis-aligned splits. The (non-linear) transformations in the intermediate layers are breaking the original orientation (which is a favorable one in the case of tabular data) of the data, and compelling trees to learn over a feature space wherein there is no longer a subset of revelant features: this partially explains why the first representation accounts for the largest degradation in performance and why performance doesn't degrade in the further representations.

Figure 5.7 illustrates this concept in $\mathbb{R}^2$ space. Trees are good at learning piecewise functions of the input space, made up by thresholding certain features. Such partitioning becomes challenging after an arbitrary unitary transformation[2] – such as a rotation – of the data, as trees cannot trace linear decision boundaries. Although it's still possible to define a set of regions such that it performs well, a tree will not perform as well as it does under the original orientation.

[2]Distance-preserving transformation

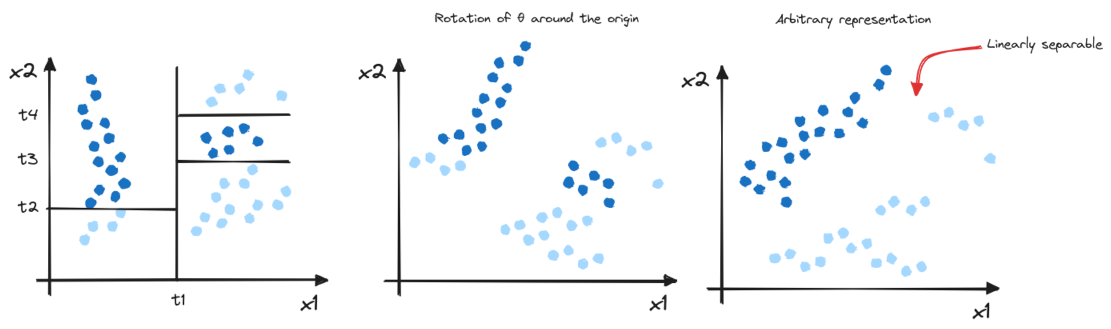


Figure 5.7: Examples of representations. This representation is common among tabular data and is easy for trees to trace axis-aligned splits (left). A representation after a rotation of θ^{o} around the origin (centre). A representation where data is linearly separable (right) .

Unitary Transformations To further understand these transformations, we apply random unitary transformations (UTs) to the data to understand if the effect is comparable to that of the hidden layers of a MLP. Furthermore, we apply the same random UTs to the learned representations and evaluate tree-based methods. In line with [Grinsztajn et al., 2022], we expect tree-based methods to perform worse on data that endured UTs, and even worse on the learned representations after the UTs. In this section, we explore this class of transformations for the MLP and later for TabNet and FT-Transformer in their respective sections.

To that extent, we consider the dataset CO1 and the representations retrieved from the shallow MLP. We generate a random matrix $Q \in \mathbb{R}^{n\times n}$ drawn from an uniform distribution and obtain a unitary matrix $R \in \mathbb{R}^{n\times n}$ via Singular Value Decomposition (SVD):

$$U\Sigma V^T = Q \tag{5.1}$$

$$R = UV^T \tag{5.2}$$

Under these conditions, it's guaranteed that R is an unitary (orthogonal) matrix and thus any transformation of the kind $\phi(x) = xR$ is guaranteed to preserve the inner product before and after the transformation. Therefore, we apply a UT by matrix R to the original input and to the learned representations. We further emphasize that we first retrieve the learned representations and then apply a UT; applying a UT, training and then retrieving the learned representations is fundamentally different and disregarded for this experiment. We perform hyperparameter tuning for XGBoost (the hyperparameter search space can be found in Appendix C and let training run until `early_stopping_rounds` without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

Figure 5.8 suggests that an arbitrary UT reverses the performance gap between XGBoost and a MLP. This goes in line with [Ng, 2004] that describes the MLP as *rotationally invariant*, and later with [Grinsztajn et al., 2022] which confirms tree-based methods to be non-rotationally invariant: although the unitary matrix

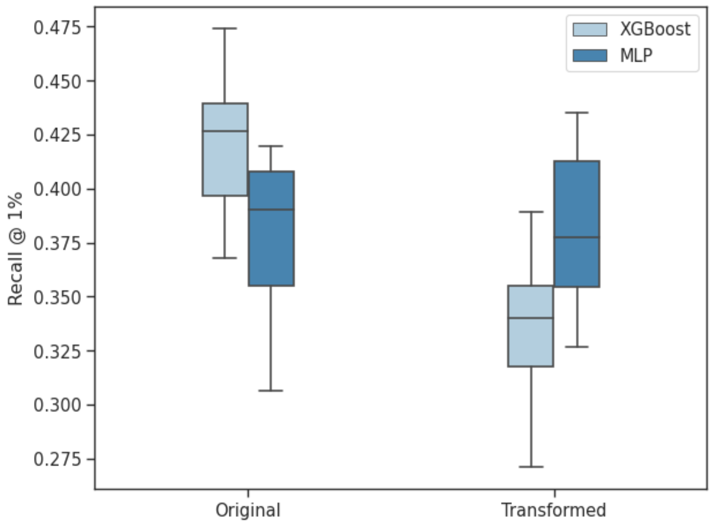


Figure 5.8: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. Performance on the original data (left). Performance on the data after a random UT (right).

R doesn't necessarily perform a rotation, a *rotation* matrix (as postulated by [Ng, 2004]) is by definition a unitary matrix – this partially explains why the MLP is invariant to rotations. This class of transformations break the axis-aligned decision boundaries of tree-based methods, which in turn partially explains why tree-based methods are non-rotationally invariant.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ MLP performance on the original input is identical to MLP performance on the original input after a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.390, 0.377)$ p-value $\approx 0.762 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance degrades after a UT.

Figure 5.9 shows surprising results. Not only does performance increase as of the first transformed representation – which is the reverse of the non-transformed representation –, but the learned representations *seem* to be invariant by rotations. The results indicate that XGBoost's performance doesn't degrade when applying a UT to the learned representation: this suggests the existence of an underlying property of the intermediate representations that breaks the non-rotationally invariance of tree-based methods – the learned representations seem to be rotationally invariant.

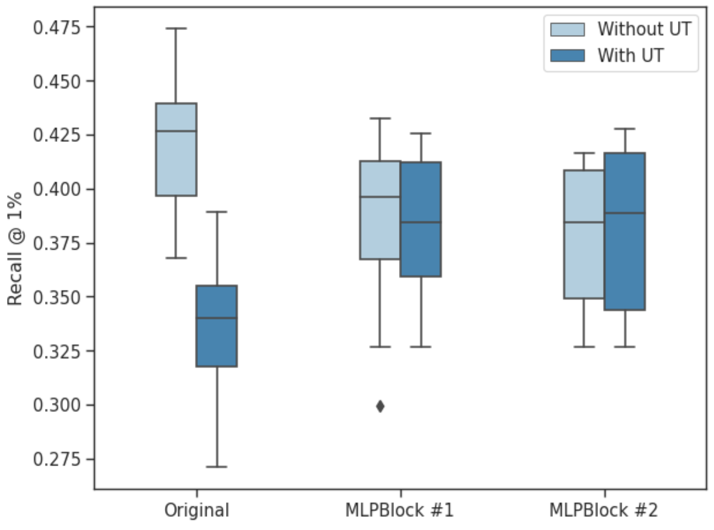


Figure 5.9: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. XGBoost performance on the original data (left), first representation (middle) and second representation (right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the first representation ("Block #1") without a UT is identical to XGBoost's performance on the first representation ("Block #1") with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.396, 0.384)$ p-value $\approx 0.880 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the first representation – or that the first representation is non-rotationally invariant.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the second representation ("Block #2") without a UT is identical to XGBoost's performance on the second representation ("Block #2") with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.384, 0.388)$ p-value $\approx 0.733 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the second representation – or that the second representation is non-rotationally invariant.

Key Takeaways There is strong evidence that, although the first hidden layer of the MLP compresses the data (because said layer outputs a representation with less degrees of freedom), the degradation in XGBoost's performance may stem from the inability of tree-based methods to work over the learned representations and not from the compression itself; these representations comprise linear and non-linear relationships between features that the axis-aligned splits of tree-based methods cannot accurately model. A key functionality of neural networks is the ability to learn compact representations, and hence to capture highly non-linear relationships in the original input. These non-linearities are not particularly useful for tree-based methods that perform axis-aligned partitions of the input space (this goes in line with [Grinsztajn et al., 2022]): the compact representations produced by neural networks exhibit local smoothness that might be preventing tree-based methods from selecting the most relevant features – a key reason behind the success of tree-based methods for tabular data is their ability to model the sensitivity of the target function to small changes in some input features [Bengio et al., 2013; Shavitt and Segal, 2018].

There is also strong evidence indicating, not only, that these representations are *not* a result of an arbitrary UT, but it also seems that the learned representations are breaking the non-invariance by UTs of tree-based methods, although there is a more obvious conclusion: the learned representation is a fundamentally different feature space; tree-based methods cannot find the original orientation of the data, and thus an arbitrary UT will not cause a performance degradation – only the data is becoming non-invariant by UTs. In fact, the very absence of semantic constraints seems to be a potential reason why tree-based methods fail, and why UTs do not cause a performance degradation.

5.2.2 TabNet

We evaluate XGBoost on the learned representations and compare results: we argue there will be a slight performance degradation (considering the superior representational capacity w.r.t. to the MLP). We consider the best-performing TabNet on the CO1 dataset and retrieve the representation learned by the `Encoder`: the TabNet model hyperparameters can be found in Table 5.3. We perform hyperparameter tuning for XGBoost (the hyperparameter search space can be found in Appendix C) and let training run until `early_stopping_rounds` without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

Table 5.3: Best TabNet model parameters on the CO1 dataset.

Parameters	
no. of decision steps	3
decision layer size	8
gamma	≈ 1.296
momentum	≈ 0.0155
lambda sparse	≈ 0.032

Figure 5.10 suggests performance seems to slightly degrade when evaluating the `Encoder` representation.

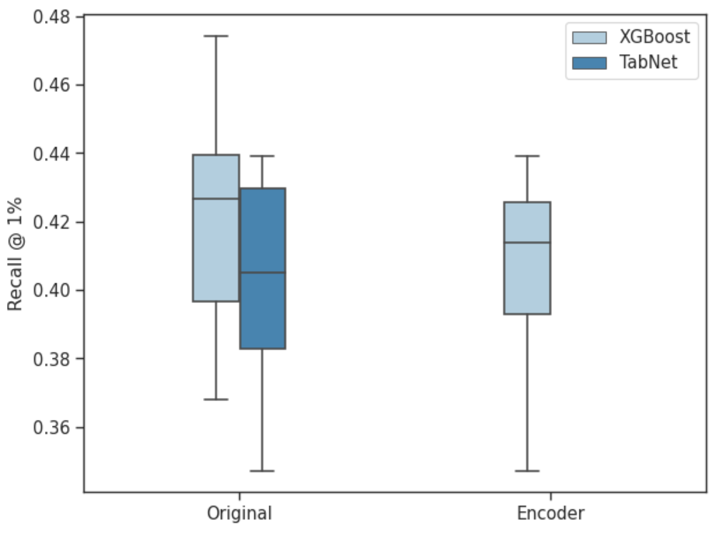


Figure 5.10: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. XGBoost and TabNet performance on the original dataset (left). XGBoost performance on the `Encoder` representation (right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the original input is identical to XGBoost's performance on the `Encoder` representation.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.427, 0.414)$ p-value $\approx 0.385 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance degrades on the `Encoder` representation.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the `Encoder` representation is identical to TabNet's performance on the original input.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.414, 0.405)$ p-value $\approx 0.940 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance doesn't converge.

These results suggest the following: the representation learned by the `Encoder` can capture some of the more complex patterns in the original input which suggests a property of TabNet that allows it to better handle tabular data. Deeper architectures generally afford greater representational capacity, although previous results (refer to Figures 5.5 and 5.6) suggest deeper models are more prone to overfitting. Indeed, the TabNet model at issue consists of only three decision steps and uses the lowest recommended dimensionality of the decision and embedding layers – which prevents

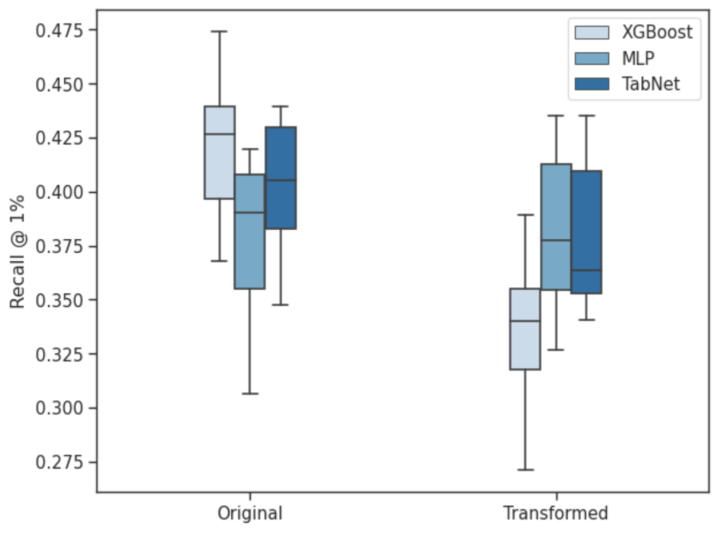


Figure 5.11: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. Performance on the original data (left). Performance on the data after a UT (right).

the model from having high capacity.

Unitary Transformations To further understand these transformations, we apply random UTs to the data to understand if the effect is comparable to that of the `Encoder`. Furthermore, we apply the same random UTs to the representation learned by the `Encoder`. In line with [Grinsztajn et al., 2022], we expect tree-based methods to perform worse on data that endured UTs, and even worse on the learned representation after the UTs.

To that extent, we consider the dataset CO1 and the representation retrieved from TabNet. We generate the unitary matrix according to 5.1 and apply a UT to the original input and the `Encoder` representation. We perform hyperparameter tuning for XGBoost (the hyperparameter search space can be found in Appendix C) and let training run until `early_stopping_rounds` without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ n%.

Figure 5.11 displays the previous results about UTs displayed in Figure 5.8 with the addition of TabNet. The results further confirm our initial claims that performance reverses after applying a random u.t. to the original input (which goes in line with [Grinsztajn et al., 2022]), but we go further and suggest that TabNet's performance seems to change.

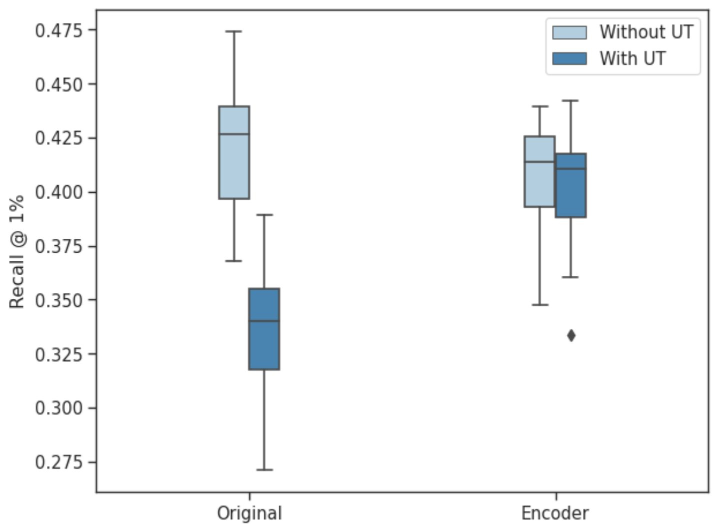


Figure 5.12: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. XGBoost performance on the original data (left) and `Encoder` representation (right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ TabNet performance on the original input is identical to TabNet performance on the original input after a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.405, 0.363)$ p-value $\approx 0.104 \longrightarrow$ we fail to reject the null hypothesis as results are not significant under a 95% confidence bound; however, we can reject the null hypothesis as results are significant under a 89% confidence bound. With less certainty, this suggests we have statistical evidence that TabNet's performance degrades after a UT.

The results above suggest TabNet is non invariant by UTs, indicating there is a key component to TabNet's architecture that breaks UT invariance.

Figure 5.12 suggests XGBoost's performance on the learned representation that doesn't significantly degrade after a UT. More striking, TabNet's `Encoder` seems to produce rotationally invariant representations as well.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the `Encoder` representation without a UT is identical to XGBoost's performance on the `Encoder` representation with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.414, 0.411)$ p-value $\approx 0.705 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the first representation – or that the `Encoder` representation is non-rotationally invariant.

Key Takeaways Nonetheless, these results are striking and indicate the possibility that TabNet is non invariant by UTs: there is an evident connection between non-rotationally invariant algorithms and the ability to perform feature selection (well) [Ng, 2004]. TabNet's instance-wise feature selection might break the invariance to UTs by encouraging the selection of salient features: when the target function only depends on a subset of features, TabNet can ignore the irrelevant features and focus only on the relevant ones [Arik and Pfister, 2021]. There is also an implicit connection with the way TabNet formulates a decision: the way the aggregation of decision outputs at different time steps – that encourages the learning of the most salient features – is performed, allows the mimic of tree-like decision manifolds which suggests TabNet can somewhat approximate the axis-aligned decision boundaries usually produced by tree-based methods.

5.2.3 FT-Transformer

We evaluate XGBoost on the learned representations and compare results: we expect performance to degrade only slightly, considering the representational capacity of FT-Transformer. We consider the best-performing FT-Transformer on the CO1 dataset and retrieve the representations learned by: i) the `FeatureTokenizer` ii) each of the Transformer `Blocks` and iii) the `Res`; the FT-Transformer model hyperparameters can be found in Table 5.4. We perform hyperparameter tuning for XGBoost (the hyperparameter search space can be found in Appendix C) and let training run until `early_stopping_rounds` without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

As previously mentioned, the [`CLS`] token is deliberately placed at the beginning of the input sequence for it comprises all the relevant information related to the learned representation. Therefore, we retrieve the [`CLS`] token from the representations at the Transformer `Blocks`, except for the `Res` layer (which is the token itself) and the `FeatureTokenizer`: given that the token is initialized as a one-vector[3], we perform average pooling of T_0 (without the token) over the m dimension to get the relevant

[3]A vector filed with 1s

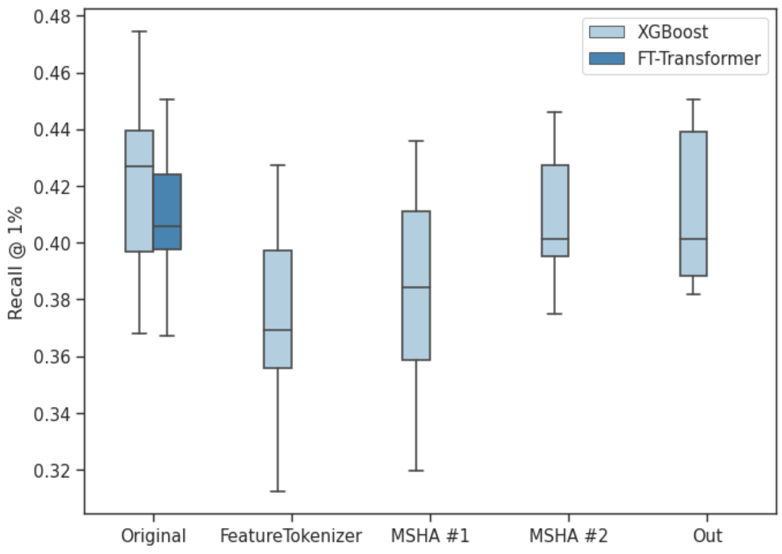


Figure 5.13: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test. XGBoost and FT-Transformer performance on the original data (left). XGBoost performance on the `FeatureTokenizer`, first and second `MHSA` blocks and `Out` representation (from left to right).

information.

Table 5.4: Best FT-Transformer model parameters on the CO1 dataset.

Parameters	
no. of layers	2
embedding layer size	256
no. of MHSA modules	8
attention dropout	≈ 0.2
`FFN` dropout	≈ 0.1

Figure 5.13 reveals a novel outcome: performance drops at the first representation but systematically improves within each representation until the top (last) layers.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the original input is identical to XGBoost's performance on `FeatureTokenizer` representation.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.426, 0.369)$ p-value $\approx 0.009 \longrightarrow$ we reject the null hypothesis as results are significant. This suggests we have strong statistical evidence that performance degrades, i.e. that performance on the `FeatureTokenizer` is worse.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the original input is identical to XGBoost's performance on the first `MHSA` representation.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.426, 0.384)$ p-value $\approx 0.054 \longrightarrow$ we fail to reject the null hypothesis as results are not significant under a 95% confidence bound; however, we can reject the null hypothesis as results are significant under a 90% confidence bound. With less certainty, this suggests we have statistical evidence that performance degrades, i.e. that performance on the first `MHSA` representation is worse.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the original input is identical to XGBoost's performance on the second `MHSA` representation.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.426, 0.401)$ p-value $\approx 0.521 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance degrades, i.e. that performance on the second `MHSA` representation is worse.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the original input is identical to XGBoost's performance on the `Out` representation.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.426, 0.401)$ p-value $\approx 0.623 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance degrades, i.e. that performance on the `Out` representation is worse.

These results suggest the representations learned by the `FeatureTokenizer` and the first `MSHA` are uninformative. Unlike the previous algorithms, the lower layers (the `FeatureTokenizer` and the first `MSHA` module) are working over a higher-dimensional embedding space. Although such might be ultimately useful for finding an optimal basis wherein data is linearly separable, one incurs the risk of increasing sample complexity.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the second `MHSA` representation is identical to FT-Transformer's performance on the original input.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.405, 0.401)$ p-value $\approx 0.650 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance doesn't converge.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the `Out` representation is identical to FT-Transformer's performance on the original input.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.405, 0.401)$ p-value $\approx 0.970 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that performance doesn't converge.

Certainly, loss of information is unlikely happening, and a much more obvious conclusion is that trees struggle to perform well in such settings wherein features are of similar relevance, further aggravated by the higher-dimensional space. The top (last) layer are working over an already transformed high-dimensional, however, its performance increases: this suggests that the top layers are learning representations that favor the inductive biases of tree-based methods.

Unitary Transformations To further understand these transformations, we apply random UTs to the data to understand if the effect is comparable to that of the hidden layers of an FT-Transformer. Furthermore, we apply the same random UTs to the representations learned by the FT-Transformer. In line with [Grinsztajn et al., 2022], we expect tree-based methods to perform worse on data that endured UTs, and even worse on the learned representations after UTs.

To that extent, we consider the CO1 dataset and apply a random u.t. to the original input and the learned representations. For XGBoost we perform hyperparameter tuning (the hyperparameter search space can be found in Appendix C) and let training run until `early_stopping_rounds` without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ $n\%$.

Figure 5.14 displays the previous results about unitary transformations displayed in Figure 5.11 but with the addition of FT-Transformer. The results further confirm our initial claims that performance reverses after applying a random u.t. to the original input (which goes in line with [Grinsztajn et al., 2022]), but we go further and suggest that FT-Transformer's performance seems to change.

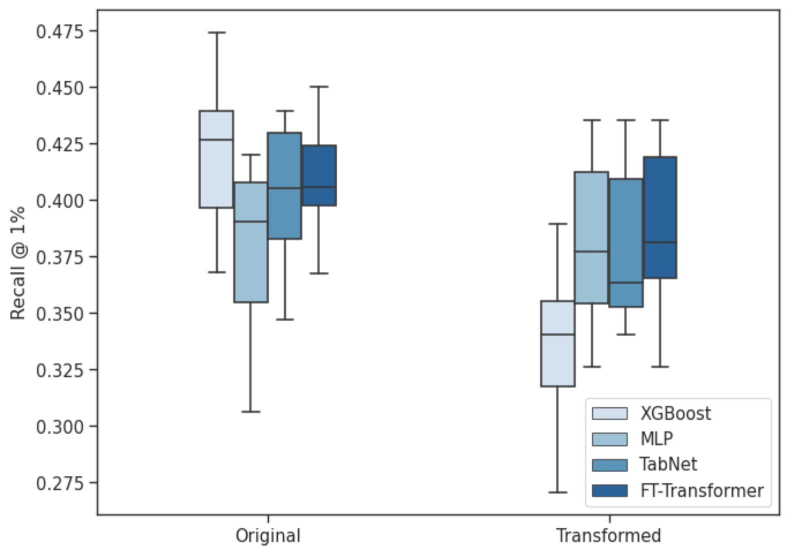


Figure 5.14: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. Performance on the original data (left). Performance on the data after a UT (right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ FT-Transformer performance on the original input is identical to FT-Transformer performance on the original input after a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.405, 0.381)$ p-value $\approx 0.186 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that FT-Transformer's performance degrades after a UT.

Although the statistical test suggests there is no meaningful performance gap between FT-Transformer's performance before and after a UT, one must consider that FT-Transformer was poorly tuned. In [Grinsztajn et al., 2022], the authors provide empirical evidence that FT-Transformer is indeed non-rotationally invariant, so we attribute the disagreement in our results to a poorly tuned model. Notwithstanding, we still found FT-Transformer to be the best DL model across our results which hints at the its promising potential as a robust architecture for DL in tabular data.

Figure 5.15 suggests that, again, XGBoost's performance doesn't degrade that much after a random u.t. when the prior input is a learned representation. More striking, is that FT-Transformer's layers seem to produce representations that are invariant to UTs.

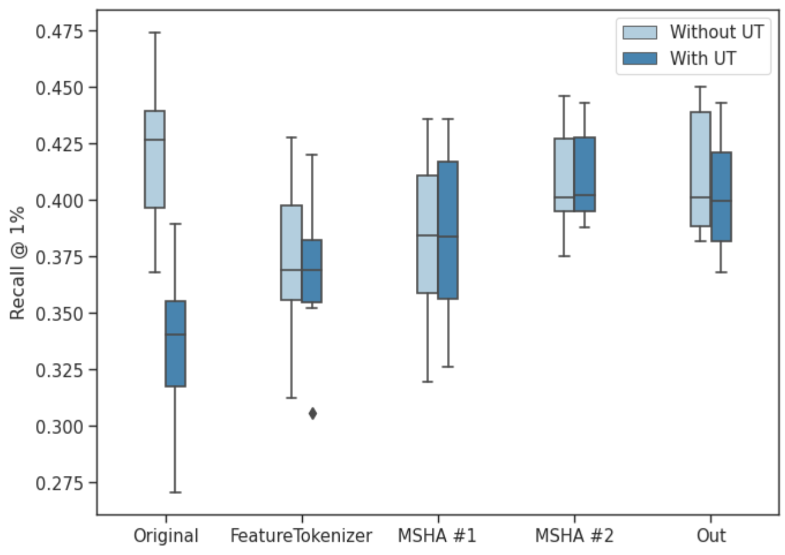


Figure 5.15: Performance (CO1 dataset, Recall @ 1%) reported across ten disjoint samples drawn at random from the test set. XGBoost performance on the original data (left), and on the `FeatureTokenizer`, first and second `MHSA` blocks and `Out` representations (left to right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the `FeatureTokenizer` representation without a UT is identical to XGBoost's performance on the `FeatureTokenizer` representation with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.369, 0.369)$ p-value $\approx 0.910 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the `FeatureTokenizer` – or that the `FeatureTokenizer` representation is non-rotationally invariant.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the first `MHSA` representation without a UT is identical to XGBoost's performance on the first `MHSA` representation with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.384, 0.384)$ p-value $\approx 1.0 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the first `MHSA` – or that the first `MHSA` representation is non-rotationally invariant.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the second `MHSA` representation without a UT is identical to XGBoost's performance on the second `MHSA` representation with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.401, 0.402)$ p-value $\approx 0.880 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the second `MHSA` – or that the second `MHSA` representation is non-rotationally invariant.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance on the `Out` representation without a UT is identical to XGBoost's performance on the `Out` representation with a UT.

$H_1 \longrightarrow$ The distributions of performance are not identical.

Test result: $\tilde{\mu} \approx (0.401, 0.399)$ p-value $\approx 0.364 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that UTs hurt performance for the `Out` – or that the `Out` representation is non-rotationally invariant.

Key Takeaways Results regarding FT-Transformer are also quite revealing, by demonstrating that the Transformer architecture can be properly adapted to tabular data if a proper embedding scheme is employed. In line with [Grinsztajn et al., 2022], the usage of certain embeddings seems to key for breaking the invariance to UTs of DL models [Gorishniy et al., 2022]. As stated in [Borisov et al., 2021], a key obstacle in asserting DL models as consistent contenders for tabular data problems is the lack of lossless encoding schemes and theoretical exploration of such. Unlike TabNet, that relies on a learnable sparse mask for soft feature selection, FT-Transformer leverages the usage of self-attention (as part of the Transformer layers) to attend to different parts of the input – this also highlights a possible link between good feature selection methods and non-rotationally invariance [Ng, 2004].

Finally, it seems that the true power of FT-Transformer also lies in Transformer layers, as evidenced by other Transformer-based models for tabular data that provided competitive results in relevant tasks [Huang et al., 2020; Somepalli et al., 2021; Song et al., 2019].

5.2.4 Takeaways

Regarding this experiment, we are able to derive some conclusions:

- Although DL produces compact representations, the compression (performed by lower-dimensional layers) is detrimental. Not only that, but the performance gap between tree-based and DL algorithms is likely not a result of compressing the input in the hidden layers.

- The DNN blocks in the DL algorithms do not perform arbitrary UTs. Not only that, but whilst tabular data is generally non invariant by UTs, the representations learned by DL algorithms seem to be invariant to UTs. Learning compact representations potentially mixes features with different statistical properties: the subset of relevant features present in the original input is lost. We show that tree-based methods trained on the learned representations have similar results to the linear classifier in DL algorithms.

- We show that the MLP is invariant to UTs (in line with [Grinsztajn et al., 2022]) and that TabNet and FT-Transformer are non-invariant to UTs: the soft feature selection mechanism of TabNet and the embedding layer of FT-Transformer seem to break this invariance.

Such results highlight for additional strong inductive biases of DL algorithms unsuitable for tabular data. Although the approximation of axis-aligned decision boundaries via hyperplanes (performed by TabNet) and contextual embeddings (employed by FT-Transformer) seem to be key for improving suitability to tabular data, the hidden layers still apply non-linear transformations to the original input; this class of transformations erase the semantic constraints of tabular data and learn compact representations, i.e. with less degrees of freedom, which may misrepresent the piece-wise relationships in tabular. Indeed, the whole point of neural networks is to encode data into a latent space where most variations within data are kept without loss of relevant information. However, the irregular patterns present in tabular data and the sensitivity of the target function to a small subset of features remain considerable obstacles to neural networks.

5.3 Experiment 2

As part of the second experiment, we explore varying degrees of feature engineering and their impact on the performance of tree-based and DL algorithms. Rich features – such as profiling features – explicit describe implicit interactions between features in tabular data; these rich features are key to tree-based methods which cannot perform such linear and non-linear combinations natively. Neural networks, on the other hand, benefit from hidden layers to learn compact representations that capture such interactions between features.

To that extent, we explore the impact of rich features of varying complexity on XGBoost and MLP. We leverage the rich BB1 dataset and split by different feature subsets. We evaluate XGBoost and the MLP on each subset to understand to what degree do tree-based and DL algorithms benefit from rich features. Finally, we relate their behaviour across the multiple feature subsets to uncover possible inductive biases behind the performance gap.

Rich Profiling Features Rich features explicitly describe implicit feature interactions, which is indeed useful for tree-based methods which cannot model linear and non-linear interactions. Not only that, but rich features can be quite complex (in terms of calculation) and represent interactions which cannot be modeled via non-linear transformations of neural networks' hidden layers.

We argue that incorporating feature engineering is ultimately beneficial for both tree-based and DL algorithms, however, we argue that tree-based methods benefit more from rich features than DL algorithms, given their inability to uncover implicit interactions in data.

Table 5.5: Examples of feature engineering on the BB1 dataset.

Type	Example
probability	log-probability of average transaction amount across one-hour windows
ratio	ratio between average transaction amounts across one-day windows
sum	total transaction amount across one-hour windows
average	average transaction amount across one-month windows

We consider the BB1 dataset and split the feature set into five subsets with different features of varying degrees of complexity: i) raw ii) raw + probabilities iii) raw + ratios iii) raw + sums iv) raw + averages. Table 5.5 provides some insight into what some of these features look like and what they represent. We evaluate XGBoost and the MLP on every feature subset. We perform hyperparameter tuning for XGBoost and the MLP (the hyperparameter search space can be found in Appendix C and let training until `early_stopping_rounds` (for XGBoost) or `patience` epochs (for the MLP) without improvement on the validation set have passed. We evaluate across ten disjoint samples drawn at random from the test set and report Recall @ n-%.

Figure 5.16 indicates XGBoost to be superior in every setting and both methods to benefit from rich features. Indeed, the MLP evaluated over the rich features subset outperforms XGBoost over the raw features for each feature subset. Additionally, results suggest that rich features widen the performance gap to some extent: there is a larger performance gap between XGBoost and the MLP for all feature subsets, except for the "sums" subset, w.r.t to the performance gap with just raw features.

Figure 5.16 displays the performance delta Δ across multiple levels of feature engineering: performance delta Δ is calculated as the performance difference between the raw version and a rich-features subset S; e.g., the higher the delta Δ, the larger the difference between the raw subset and S, i.e. subset S improves performance to a greater extent.

Foremost, the absence of negative values indicate rich features always improve the performance for both tree-based and DL algorithms. Additionally, ratio-related and sum-related features improve the performance substantially. Secondly, incorporating probabilities, ratios and averages into the raw data benefits more XGBoost than it benefits the MLP. On the other hand, leveraging sums seems to benefit more the MLP than XGBoost.

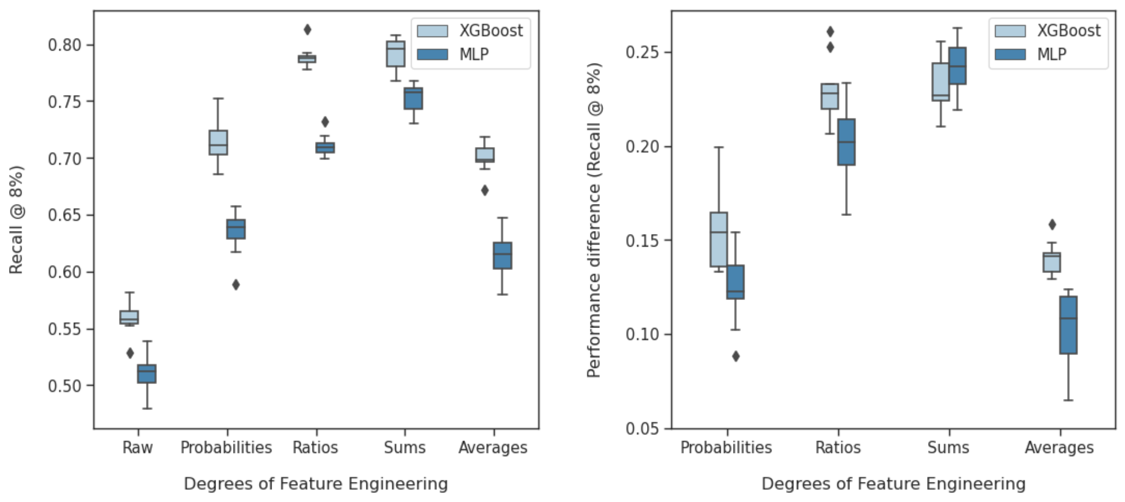


Figure 5.16: Performance (BB1 dataset, Recall @ 8%) (left) and performance delta (BB1 dataset, Recall @ 8%) reported across ten disjoint samples drawn at random from the test set. Performance across different sets of rich features (left). Performance delta is computed as the difference in performance between the raw set and a given rich set of features (right).

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance delta is identical to MLP performance delta for probability-related features.

$H_1 \longrightarrow$ The distributions of performance deltas are not identical.

Test result: $\tilde{\mu} \approx (0.154, 0.123)$ p-value $\approx 0.006 \longrightarrow$ we reject the null hypothesis as results are significant. This suggests we have strong statistical evidence that there is a performance gap.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance delta is identical to MLP performance delta for ratio-related features.

$H_1 \longrightarrow$ The distributions of performance deltas are not identical.

Test result: $\tilde{\mu} \approx (0.228, 0.202)$ p-value $\approx 0.006 \longrightarrow$ we reject the null hypothesis as results are significant. This suggests we have strong statistical evidence that there is a performance gap.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance delta is identical to MLP performance delta for sum-related features.

$H_1 \longrightarrow$ The distributions of performance deltas are not identical.

Test result: $\tilde{\mu} \approx (0.227, 0.242)$ p-value $\approx 0.151 \longrightarrow$ we fail to reject the null hypothesis as results are not significant. This suggests we don't have strong statistical evidence that there is a performance gap.

Mann-Whitney U Statistical Test

$H_0 \longrightarrow$ XGBoost's performance delta is identical to MLP performance delta for average-related features.

$H_1 \longrightarrow$ The distributions of performance deltas are not identical.

Test result: $\tilde{\mu} \approx (0.142, 0.108)$ p-value $\approx 0.000 \longrightarrow$ we reject the null hypothesis as results are significant. This suggests we have strong statistical evidence that there is a performance gap.

Statistical evidence suggests that trees benefit more from rich features w.r.t the MLP , except for sum-related features where there is no strong statistical evidence that the MLP benefits more or less from sum-related features.

Feature Importance Figure 5.17 displays XGBoost feature importances for the different sets of rich features. Feature importance is calculated as the average gain across splits for a given feature. The information gain often provides a good measure of the relevance of a given feature. Results suggest the rich features are indeed relevant considering these represent the majority of features that contribute the most to the model decision.

Additionally, the relevance of these rich features partially explains why the MLP performs better: the MLP struggles to learn in settings wherein only a subset of features are relevant [Ng, 2004]. This goes inline with [Grinsztajn et al., 2022], wherein the authors report MLP-like neural networks to be vulnerable to uninformative features and to also benefit from relevant features.

5.3.1 Takeaways

The results obtained support our initial claim that rich features help tree-based methods and DL algorithms: explicitly describing complex interactions between features affords trees to leverage all the implicit information in data. Neural networks are usually free of explicit feature engineering due to learning compact representations internally. However, neural networks might benefit from rich explicit features in tabular data tasks. In line with [Ng, 2004], neural networks usually struggle in settings wherein only a portion of the features are relevant. Feature engineering

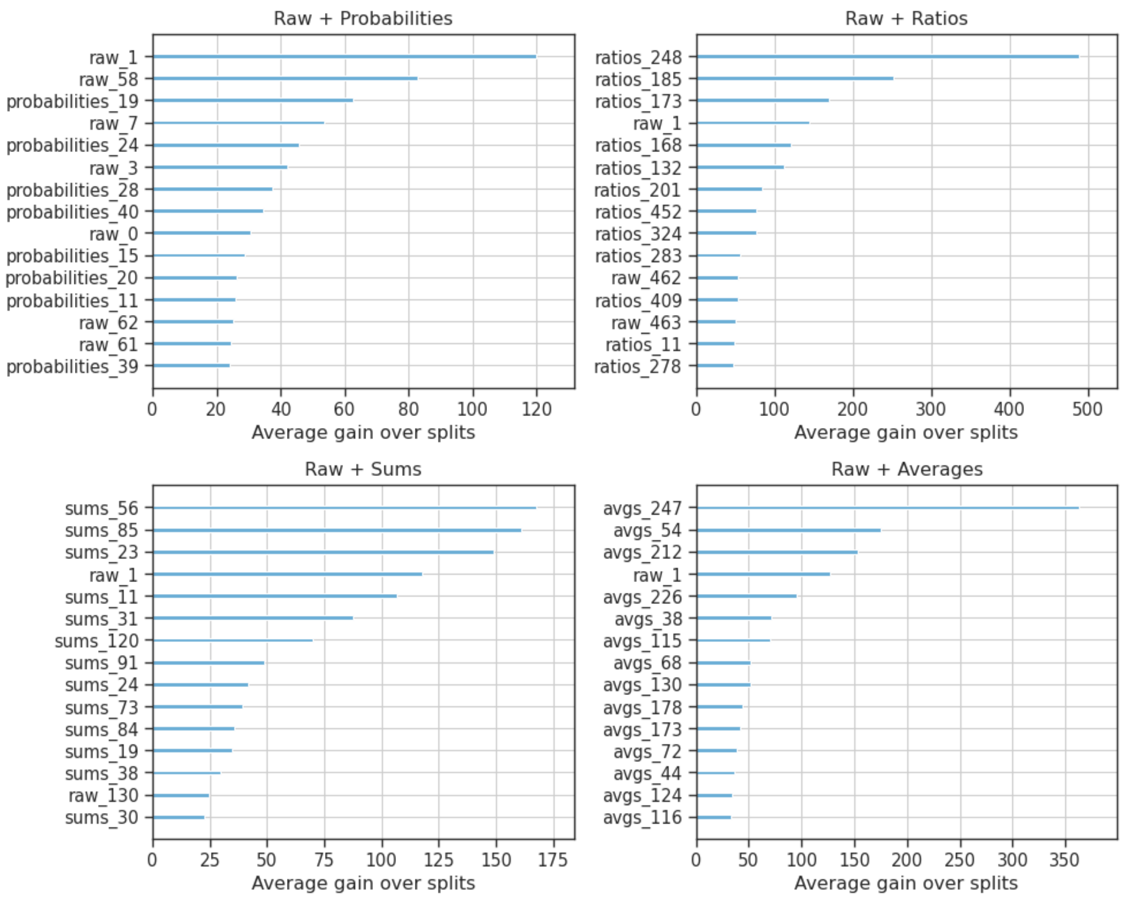


Figure 5.17: Feature importance computed by XGBoost models for feature subsets with different levels of feature engineering. We sort features by descending order of feature importance and display the top ten features. The feature importance score is computed as the average gain across splits for a given feature.

helps by increasing the amount of relevant features *a priori* and incorporating some of the most complex feature interactions that neural networks may fail to capture.

Nonetheless, the performance gap may not stem from a lack (or presence) of feature engineering. The presence of rich features improves the performance of both tree-based and DL algorithms; at most, the presence of feature engineering *widens* the performance gap between these algorithms. The relevance of such a conclusion is questionable, given that most tabular datasets commonly used in the relevant literature don't comprise rich features and are frequently dominated by redundant and uninformative features.

5.4 Summary

In this chapter, we presented the experimental work. We detailed four minor experiments we performed. Next, we delved into an experiment exploring the representational capacity of DNNs and it's relation to tabular data. Finally, we conducted an experiment evaluating the sensitivity of tree-based and DL algorithms to feature engineering.

Chapter 6

Conclusion and Outlook

In this chapter, we provide the final conclusions regarding our experimental work and hint at the relevant future work.

6.1 Conclusion

In this thesis, we uncovered possible causes for the performance gap between tree-based and Deep Learning (DL) algorithms in tabular data, under the scope of fraud detection. Foremost, we show tree-based methods outperform DL algorithms for our tabular datasets. We demonstrate the performance gap is likely not caused by compression of information or arbitrary unitary transformations in the hidden layers, nor by feature engineering. Indeed, we demonstrate it likely stems from a disagreement between the inductive biases of DL algorithms and the properties of tabular data: i) DL algorithms may misrepresent irregular patterns in tabular data via linear and non-linear dependencies in the representational space ii) invariance to UTs is an undesirable property that leads to poor performance in settings with few relevant features.

Our results reveal the Multi Layer Perceptron (MLP) to be invariant to unitary transformations (UTs) and showed TabNet and FT-Transformer to be non invariant to UTs: such is likely due to the feature selection mechanism in TabNet and to the embedding layer in FT-Transformer. These components encode the appropriate inductive biases that better align with the inductive biases of tree-based methods which are particularly useful for tabular data.

Finally, we show that feature engineering can be beneficial for both tree-based and DL algorithms. Indeed, explicitly describing complex interactions in tabular data affords the possibility to leverage all the implicit information that is difficult to model by trees and neural networks. Our results indicate that the performance gap may not stem from a lack of feature engineering, despite observing that rich features *widen* the performance gap: most tabular datasets commonly used in the relevant literature don't comprise rich features and are naturally dominated by redundant and uninformative features.

In sum, and in line with several works on the area, TabNet and FT-Transformer incorporate inductive biases that better align with the core properties of tabular data: features interact with themselves via irregular patterns and the target function usually responds to only a subset of relevant features. Such is likely evidenced by the ability of tree-based methods to perform reasonably well in the representations learned by such algorithms: TabNet and FT-Transformer learn representations that are more "GBDT-friendly".

We believe such study is relevant in the fraud domain: we showed that some DL algorithms are able to provide competitive results, which hints at the possibility of soon incorporating DL into fraud detection mechanisms. Notwithstanding, tree-based methods are still the favorite approach given their attractive characteristics much valued in the fraud domain: native interpretability and low computational overhead.

6.2 Outlook

We understand there still exists relevant work to pursue in the area. Most importantly, we foster for more theoretical results of TabNet and FT-Transformer as favorable routes of research: understanding the theory behind them seems to be key to close the performance gap.

In line with our work, we consider the exploration of higher-dimensional hidden layers and how does loss of information occur in representations with more degrees of freedom. To the best of our knowledge, validating the amount of degrees of freedom in representations as a possible cause for the performance gap is indeed relevant. Additionally, we consider the lack of appropriate encoding methods that cater to the properties of categorical features, such as multi-modal distributions and semantic constraints. Furthermore, we consider a more thorough exploration of the role of categorical features, specially in TabNet and FT-Transformer.

Moreover, we consider the exploration of more intricate embedding layers to be crucial for devising strategies that break the rotation invariance of DL algorithms. In line with [Gorishniy et al., 2022], embeddings for numerical features are an under-explored venue of research in tabular data: the authors find that embedding numerical features allows simpler architectures (such as the MLP) to perform on par with Transformer-based algorithms. Additionally, most architectures are built upon the pre-made conception that the same embedding transform (e.g. function) is applied to all features: such choice may limit the representational capacity of DL algorithms. Employing different embeddings schemes for different features may afford the possibility to learn encodings that more precisely represent each feature in the latent space.

6.3 Summary

In this chapter, we drew the final takeaways regarding this thesis. We detailed the key results of the experimental work and derived meaningful conclusions. Finally, we provided possible venues for future work.

References

Emin Aleskerov, Bernd Freisleben, and Bharat Rao. Cardwatch: A neural network based database mining system for credit card fraud detection. In *Proceedings of the IEEE/IAFE 1997 computational intelligence for financial engineering (CIFEr)*, pages 220–226. IEEE, 1997.

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6679–6687, 2021.

Sarkhan Badirli, Xuanqing Liu, Zhengming Xing, Avradeep Bhowmik, Khoa Doan, and Sathiya S Keerthi. Gradient boosting neural networks: Grownet. *arXiv preprint arXiv:2002.07971*, 2020.

European Central Bank. Seventh report on card fraud, 10 2021. URL `https://www.ecb.europa.eu/pub/cardfraud/html/ecb.cardfraudreport202110~cac4c418e8.en.html`.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

Siddhartha Bhattacharyya, Sanjeev Jha, Kurian Tharakunnel, and J Christopher Westland. Data mining for credit card fraud: A comparative study. *Decision support systems*, 50(3):602–613, 2011.

Richard Bolton and David Hand. Unsupervised profiling methods for fraud detection. *Conference on Credit Scoring and Credit Control*, 7, 2001.

Richard J Bolton and David J Hand. Statistical fraud detection: A review. *Statistical science*, 17(3):235–255, 2002.

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

Rüdiger Brause, T Langsdorf, and Michael Hepp. Neural data mining for credit card fraud detection. In *Proceedings 11th International Conference on Tools with Artificial Intelligence*, pages 103–106. IEEE, 1999.

Fabrizio Carcillo, Yann-Aël Le Borgne, Olivier Caelen, Yacine Kessaci, Frédéric Oblé, and Gianluca Bontempi. Combining unsupervised and supervised learning in credit card fraud detection. *Inf. Sci.*, 557:317–331, 2019.

Philip K Chan, Wei Fan, Andreas L Prodromidis, and Salvatore J Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems and Their Applications*, 14(6):67–74, 1999.

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

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

Merriam Webster Dictionary. Fraud, 2023.

Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

Jose R Dorronsoro, Francisco Ginel, C Sgnchez, and Carlos S Cruz. Neural fraud detection in credit card operations. *IEEE transactions on neural networks*, 8(4): 827–834, 1997.

Ekrem Duman and M Hamdi Ozcelik. Detecting credit card fraud by genetic algorithm and scatter search. *Expert Systems with Applications*, 38(10):13057–13063, 2011.

Ugo Fiore, Alfredo De Santis, Francesca Perla, Paolo Zanetti, and Francesco Palmieri. Using generative adversarial networks for improving classification effectiveness in credit card fraud detection. *Inf. Sci.*, 479:448–455, 2017.

Walter D Fisher. On grouping for maximum homogeneity. *Journal of the American statistical Association*, 53(284):789–798, 1958.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

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

Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378, 2002.

Kang Fu, Dawei Cheng, Yi Tu, and Liqing Zhang. Credit card fraud detection using convolutional neural networks. In *International Conference on Neural Information Processing*, 2016.

Sushmito Ghosh and Douglas L Reilly. Credit card fraud detection with a neural-network. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 3, pages 621–630. IEEE, 1994.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Yura Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. *arXiv preprint arXiv:2203.05556*, 2022.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data? *arXiv preprint arXiv:2207.08815*, 2022.

A Hart. Mann-Whitney test is not just a test of medians: differences in spread can be important. *BMJ*, 323(7309):391–393, August 2001.

Trevor Hastie, Robert Tibshirani, and J. H. Friedman. *The elements of statistical learning: data mining, inference, and prediction.* Springer, New York, second edition, corrected 7th printing edition, 2009. ISBN 978-0-387-84858-7. OCLC: 405547558.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*, volume 103 of *Springer Texts in Statistics*. Springer New York, New York, NY, 2013. ISBN 978-1-4614-7137-0 978-1-4614-7138-7. doi: 10.1007/978-1-4614-7138-7.

Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent data analysis*, 6(5):429–449, 2002.

Arlind Kadra, Marius Lindauer, Frank Hutter, and Josif Grabocka. Well-tuned simple nets excel on tabular datasets. *Advances in neural information processing systems*, 34:23928–23941, 2021.

Liran Katzir, Gal Elidan, and Ran El-Yaniv. Net-dnf: Effective deep modeling of tabular data. In *International Conference on Learning Representations*, 2020.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Ron Kohavi. Bottom-up induction of oblivious read-once decision graphs. In *Machine Learning: ECML-94: European Conference on Machine Learning Catania, Italy, April 6–8, 1994 Proceedings 7*, pages 154–169. Springer, 1994.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.

Zhenchuan Li, Mian Huang, Guanjun Liu, and Changjun Jiang. A hybrid method with dynamic weighted entropy for handling the problem of class imbalance with overlap in credit card fraud detection. *Expert Syst. Appl.*, 175:114750, 2021.

Sara Makki, Zainab Assaghir, Yehia Taher, Rafiqul Haque, Mohand-Saïd Hacid, and Hassan Zeineddine. An experimental study with imbalanced classification approaches for credit card fraud detection. *IEEE Access*, 7:93010–93022, 2019.

Andre Martins and Ramon Astudillo. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning*, pages 1614–1623. PMLR, 2016.

Christoph Molnar. *Interpretable Machine Learning.* 2 edition, 2022. URL `https://christophm.github.io/interpretable-ml-book`.

Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78, 2004.

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

Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29:3784–3797, 2018.

Nilson Report. Card and Mobile Payment Industry Statistics | Nilson Report Archive of Charts Graphs. URL `https://nilsonreport.com/publication_chart_and_graphs_archive.php?1=1&year=2017`.

Nilson Report. Card and Mobile Payment Industry Statistics | Nilson Report Archive of Charts Graphs, 11 2022. URL `https://nilsonreport.com/publication_chart_and_graphs_archive.php?1=1&year=2022`.

Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

Doyen Sahoo, Quang Pham, Jing Lu, and Steven CH Hoi. Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*, 2017.

Robert E. Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. The MIT Press, May 2012. doi: 10.7551/mitpress/8291.001.0001.

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

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

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

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

Weiping Song, 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*, pages 1161–1170, 2019.

Altyeb Altaher Taha and Sharaf Jameel Malebary. An intelligent approach to credit card fraud detection using an optimized light gradient boosting machine. *IEEE Access*, 8:25579–25587, 2020.

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

Daixin Wang, Yuan Qi, Jianbin Lin, Peng Cui, Quanhui Jia, Zhen Wang, Yanming Fang, Quan Yu, Jun Zhou, and Shuang Yang. A semi-supervised graph attentive network for financial fraud detection. *2019 IEEE International Conference on Data Mining (ICDM)*, pages 598–607, 2019.

Wei Wei, Jinjiu Li, Longbing Cao, Yuming Ou, and Jiahang Chen. Effective detection of sophisticated online banking fraud on extremely imbalanced data. *World Wide Web*, 16(4):449–475, 2013.

David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using conditional gan. *Advances in Neural Information Processing Systems*, 32, 2019.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Xinwei Zhang, Yaoci Han, Wei Xu, and Qili Wang. Hoba: A novel feature engineering methodology for credit card fraud detection with a deep learning architecture. *Inf. Sci.*, 557:302–316, 2019.

Appendices

Appendix A

Software and Hardware

We implement a re-usable workflow for experimentation that consists of the following:

1. Specify a configuration file for the tuple dataset-model
2. Tune the model on a dataset according to configuration file
3. Evaluate model on a dataset according to configuration file

This workflow was implemented in Python 3.7[1]. All experiments were conducted under the same hardware setup, consisting of a single Intel Xeon Gold 5120 CPU with 12 cores and 40GB of RAM.

[1]https://www.python.org/

Appendix B

Datasets

We describe all the datasets used throughout the experimental work and provide the batch size used for DL algorithms.

Table B.1: Datasets description. Batch size displays the batch size for every DDL algorithm: (A) is for the MLP, (B) for TabNet (C) for FT-Transformer and (D) for NODE.

Name	Numerical	Categorical	Train	Val	Test	Batch size
BB1	1287	6	343894	68778	85974	(A,D) 1024 (B,C) 2048
RS1	236	57	294062	58812	1M	(A,B,C,D) 2048
CO1	96	0	426013	85202	109859	(A) 1024 (B,C) 2048 (D) 512
WB1	100	1	540883	108176	135221	(A,D) 1024 (B,C) 2048

As previously mentioned in Chapter 4, we split the BB1 dataset into five subsets with different features. Table B.2 details each subset.

Table B.2: BB1 feature subsets description.

Name	Numerical	Categorical
Raw	8	6
Raw + Probabilities	56	6
Raw + Ratios	457	6
Raw + Sums	128	6
Raw + Averages	271	6

Considering the high amount of features, we tuned LightGBM on the RS1 and

WB1 datasets and retrieve 30 features in decreasing order of feature importance; we consider these reduced datasets for all experiments/results.

Appendix C

Hyperparameters

We detail the hyperparameter search space for the several algorithms used in this thesis' practical work. We perform the same hyperparameter search for all datasets but provide enough parameter range and Optuna trials for arbitrary sub-optimal parameters to be found, whilst still maintaining realistic training times. Furthermore, the only parameter we change across datasets is the batch size that we set manually for each dataset, as it largely depends on the volume and dimensionality of the dataset.

For the tree-based algorithms (LightGBM, XGBoost and Catboost) we perform 100 trials, and for DL algorithms (MLP, TabNet, NODE and FT-Transformer) we perform 50 trials. As finetuning can be quite expensive for DL algorithms, we limit the training budget to 10 hours.

For tree-based algorithms we fix the `early_stopping_rounds` $= 50$ and for DL algorithms we fix the `epochs` $= 1e9$.

LightGBM We follow the open source implementation[1] and fix the following parameters:

1. `boosting`: gbdt

We fix the boosting mechanism and use traditional GBDT which yields slightly better results than GOSS at the expense of slightly longer training times for which the trade-off is almost negligible.

XGBoost We follow the open source implementation[2] and fix the following parameters:

1. `booster`: `gbtree`
2. `tree_method`: `hist`

[1]https://lightgbm.readthedocs.io/en/v3.3.2/
[2]https://xgboost.readthedocs.io/en/stable/

Table C.1: Hyperparameter space for LightGBM

Parameter	Space
no. of estimators	uniform: $[100, 500]$
no. of leaves	uniform: $[10, 100]$
max depth	uniform: $[3, 7]$
learning rate	log uniform: $[1e-2, 1.0]$
subsample	uniform: $[0.5, 1.0]$
col. sample by tree	uniform: $[0.5, 1.0]$
L1 regularization	log uniform: $[1e-8, 1.0]$
L2 regularization	log uniform: $[1e-8, 1.0]$

Table C.2: Hyperparameter space for XGBoost

Parameter	Space
no. of estimators	uniform: $[100, 500]$
max. depth	uniform: $[3, 7]$
learning rate	log uniform: $[1e-3, 1.0]$
subsample	uniform: $[0.5, 1.0]$
col. sample by tree	uniform: $[0.5, 1.0]$
L1 regularization	log uniform: $[1e-8, 1.0]$
L2 regularization	log uniform: $[1e-8, 1.0]$

We fix the boosting mechanism and use only GBDT (`gbtree`). In both XGBoost and LightGBM we refrained from using DART as it was in our best interest to also limit the degrees of freedom in the hyperparameter search. We also fix the method for building the trees, which is a histogram-based greedy algorithm similar to LightGBM's.

Catboost We follow the open source implementation[3] and don't fix any specific parameter. We don't delve to deep into the parameterization of Catboost as it's training is quite expensive and the default parameters already provide a reasonable baseline across several datasets.

[3]https://catboost.ai/

Table C.3: Hyperparameter space for Catboost

Parameter	Space
no. of estimators	uniform: $[100, 500]$
learning rate	uniform: $[1e-3, 1e-2]$
depth	uniform: $[3, 7]$
leaf estimation iterations	log uniform: $[1, 5]$
leaf L2 regularization by tree	log uniform: $[1.0, 10.0]$
bagging temperature	uniform: $[0.0, 1.0]$

Table C.4: Hyperparameter space for MLP

Parameter	Space
no. of layers	uniform: $[1, 8]$
layers size	uniform: $\{8, 16, 32, 64, 128, 256\}$
dropout	uniform: $[0.0, 0.2]$

MLP We provide a PyTorch[4] implementation based on [Gorishniy et al., 2021] and fix the following parameters:

1. `optimizer`: `Adam` with `learning_rate`$= 1e - 2$
2. `patience`: 10
3. `fixed_layer_size`: `false`

We fix the `patience` at 10 to reduce training times. We also fix the optimizer method and parameters to avoid extensive parameter searches. Finally, we don't constrain the model to have fixed-sized hidden layers.

TabNet We follow the open source implementation[5] and fix the following parameters:

1. `optimizer`: `AdamW` with `learning_rate`$= 2e - 2$ and `weight_decay`$= 1e - 3$
2. `patience`: 6
3. `mask_type`: `sparsemax`

We fix `patience` at 6 to reduce training times. We also fix the mask to `sparsemax`, as it outputs sparse probability distributions which aligns well with the goal of sparse feature selection [Arik and Pfister, 2021; Martins and Astudillo, 2016]. Our hyperparameter space is defined according to general guidelines provided in the original paper [Arik and Pfister, 2021] and we always start from a default configuration (an "educated" guess of sorts). As advised by the authors, the width of the attention embedding N_a should be equal to the width of the decision layer N_d, therefore we only search for the optimal value for N_d and set $N_a = N_d$.

FT-Transformer We follow the open source implementation[6] and fix the following parameters:

1. `optimizer`: `AdamW` with `learning_rate`$= 1e - 3$ and `weight_decay`$= 1e - 5$
2. `patience` $= 6$

[4]https://pytorch.org/

[5]https://dreamquark-ai.github.io/tabnet/

[6]https://github.com/Yura52/tabular-dl-revisiting-models

Table C.5: Hyperparameter space for TabNet

Parameter	Space
no. of decision steps	uniform: $[3, 10]$
decision layer size	uniform: $8, 16, 32, 64, 128$
gamma	uniform: $[1.0, 2.0]$
momentum	log uniform: $[1e-2, 4e-1]$
lambda sparse	uniform: $[1e-5, 1e-1]$

Table C.6: Hyperparameter space for FT-Transformer

Parameter	Space
no. of layers	uniform: $[1, 3]$
embedding layer size	uniform: $\{4, 8, 16, 32, 64, 128, 256\}$
no. of MHSA modules	uniform: $\{4, 8\}$
attention dropout	uniform: $[0.0, 0.3]$
`FFN` dropout	uniform: $[0.0, 0.3]$

3. `activation`: `ReGLU`
4. `d_ffn_factor`: $\frac{4}{3}$
5. Residual dropout: 0.0
6. Initialization: `kaiming`
7. Prenormalization: `true`

We fix `patience` at 6 to reduce training times given the available hardware setup. Our hyperparameter space is defined according to general guidelines provided in the original paper ([Gorishniy et al., 2021]): we use the `PreNorm` variant (instead of `PostNorm`) and use `kaiming` initialization ([He et al., 2015]). As for the activation functions, we use `ReGLU` for the `FFN` modules and `ReLU` in the final layer. We keep the hyperparameter search space relatively simple as the FT-Transformer is very resource-demanding.

NODE We follow the open source implementation[7] and fix the following parameters:

1. `optimizer`: `AdamW` with `learning_rate`$= 2e-2$ and `weight_decay`$=1e-3$
2. `patience` $= 6$

We fix `patience` at 6 to reduce training times. Our hyperparameter space is defined according to general guidelines provided in the original paper [Popov et al., 2019]. We keep the hyperparameter search space relatively simple as the NODE implementation available is very memory inefficient.

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

Table C.7: Hyperparameter space for NODE

Parameter	Space
layer size	uniform: $\{128, 256, 512\}$
tree output dimension	uniform: $\{2, 3\}$
no. of layers	uniform: $[1, 6]$
tree depth	uniform: $[3, 7]$