

Mixed data regression for price prediction

Alexander Larionov

Submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in the Department of Mathematics, Faculty of Natural Sciences,
Imperial College London.
March 2025

Abstract

There are many applications of price regression: to manage cash flows for business, balance supply and demand for energy, or quantify risks for insurance. Historically, regression models – be they linear or complex neural networks – have mainly used ‘tabular’ data composed of numeric or nominal categorical variables. However, as companies and institutes increasingly collect more text data, and language models substantially improve every year, there is growing interest in how to leverage text for regression. This thesis studies using text-only data as well as combining text *and* tabular data for regression, applied to price prediction.

We begin by discussing an often overlooked aspect of *tabular* modelling: missing value handling. A large, real UK insurer dataset is used to study the impact of missing value handling on the performance of gradient boosted trees and deep learning models, two contemporary tabular modelling approaches. In doing so we establish Catboost as a strong tabular model, relevant for benchmarking joint text-tabular models, but emphasize the importance of missing values in doing so.

We then conduct a study of text-only regression. Text-only regression requires a pipeline of preprocessing, tokenization, featurization and modelling steps. We rigorously analyze choices for these steps, where the literature lacks systematic work. Among other results, for the first time, our study quantifies the relative importance of the model choice compared to choice of steps before modelling. We find that choice of the upstream steps contributes variance in performance comparable to model choice.

We conclude with a study of joint text-tabular modelling: studying an ensemble of a tabular-only Catboost and a text-only neural network. This ensemble is compared to existing methods. Overall, we conclude that although this ensemble does not consistently have the best accuracy, it is simple, robust to data with low signal in the text modality and achieves good accuracy.

Publications

Chapter 3 is based on the publication:

Larionov, Alexander, Niall M. Adams, and Kevin N. Webster. **Investigating the Impact of Missing Value Handling on Boosted Trees and Deep Learning for Tabular Data: A Claim Reserving Case Study**. Transactions on Machine Learning Research, 3 September 2024.

Statement of originality

I, Alexander Larionov, certify this thesis and the work it refers to is the output of research conducted by me, unless otherwise stated. Any work of others is fully acknowledged with standard referencing practices.

Signed: Alexander Larionov

Date: 14 March 2025

Copyright

The copyright of this thesis rests with the author. Unless otherwise indicated, its contents are licensed under a Creative Commons Attribution-Non Commercial 4.0 International Licence (CC BY-NC). Under this licence, you may copy and redistribute the material in any medium or format. You may also create and distribute modified versions of the work. This is on the condition that: you credit the author and do not use it, or any derivative works, for a commercial purpose. When reusing or sharing this work, ensure you make the licence terms clear to others by naming the licence and linking to the licence text. Where a work has been adapted, you should indicate that the work has been changed and describe those changes. Please seek permission from the copyright holder for uses of this work that are not included in this licence or permitted under UK Copyright Law.

Acknowledgements

I would like to express my gratitude for the consistent support of my two supervisors, Niall Adams and Kevin Webster. Our regular weekly meetings, for 4 years, helped me to find my way through the fog of research. To Niall: I am thankful for your insightful critiques that were designed to challenge my assumptions, train my critical thinking and grow as an independent researcher. Your practical advice on how to ask and answer *research* questions (and how to write them up!) without kidding myself were invaluable throughout the my studies. To Kevin: I am also thankful for your steadfast support throughout, with fast feedback whenever I needed it. I have fond memories of the time spent learning the fundamentals of DL using your course. I truly believe without it I would have been lost at sea early on.

I wish to also acknowledge my industry partners, Tesco Underwriting and Dunnhumby, for their funding and professional assistance. Their role was vital in making the work contained in this thesis a reality, and allowing me to study interesting real world problems.

To my parents: it is you who have always supported me to put the work in to pursue challenging goals, and from whom I learned to resolutely persevere in the face of difficulty.

I also wish to say thanks to the friends along the way. Xing, I was lucky to have you in the office. You are a brilliant mind, with a gift for clear thought that often helped me. Andrea, the regular weekend activities and lunchtime statistics puzzles were a great joy. Jose and Mike, beyond the regular coffees and chats, the Bocconi summer school will always be a great memory. Tresnia, I feel lucky to have you as a friend. Your advice on life, and opportunities to enjoy teaching that you gave me, were invaluable.

Finally, I also want to thank my partner Rachel, who I suspect was a philosopher in a previous life. You have a unique and wise outlook on life that has made the daily task of enjoying life always easier with you. Your reservoir of insights and support has lifted me up and pushed me to be the person I want to be.

Contents

1	Introduction	15
1.1	Thesis outline	17
2	Background	19
2.1	Natural language processing	19
2.1.1	Tokenization	20
2.1.2	Featurization	22
2.2	Deep learning	24
2.2.1	Fundamentals	24
2.2.2	Layers	27
2.2.3	Architectures	31
2.2.4	BERT training	33
2.2.5	Other training techniques	35
2.3	Gradient boosted decision trees	40
2.4	Catboost	41
2.4.1	Ordered boosting procedure	42
2.4.2	Categorical handling	43
2.4.3	Further Catboost specifics	44
2.5	Alternative statistical approaches	46
3	Missing values in tabular modelling	47
3.1	Introduction	47
3.2	Background	49
3.2.1	Data description	51
3.3	Models	53
3.3.1	Catboost model	53
3.3.2	ResNet MLP	54
3.3.3	TabNet model	56

3.4	Experimental method	57
3.4.1	Hyperparameter tuning	58
3.4.2	Investigating different preprocessing schemes	61
3.4.3	Effects of imputation scheme on other datasets	63
3.5	Results and discussion	63
3.6	Conclusion	66
4	NLP Pipelines for textual regression	68
4.1	Introduction	68
4.2	Background	69
4.2.1	Preprocessing	70
4.2.2	Tokenization	72
4.2.3	Featurization	72
4.2.4	Modeling	74
4.2.5	‘Traditional’ and ‘Modern’ approaches	75
4.3	Related work	75
4.4	Method	76
4.5	Results and Discussion	77
4.5.1	Contextual word embeddings outperform non-contextual word embeddings	78
4.5.2	Finetuning is required for BERT methods to outperform bag of words . .	78
4.5.3	Bag of words approaches are still relevant for resource constrained applications	81
4.5.4	Pipeline steps upstream from modelling have performance impact comparable to model choice	81
4.5.5	Vector embeddings are not suited to regression with GBDTs	85
4.6	Conclusion	86
5	Evaluating ensembling for text and tabular modelling	88
5.1	Introduction	88
5.2	Related work	90
5.2.1	Terminology	90
5.2.2	An example of ‘ad hoc’ text-tabular modelling in medicine	91
5.2.3	Methodological text-tabular studies	92
5.2.4	Gap in the literature: simple ensembles	97
5.3	Method	99

5.3.1	Ensemble model	99
5.3.2	Experimental procedure	101
5.4	Results and Discussion	104
5.5	Conclusion	106
6	Conclusion	108
6.1	Future Work	110
	Bibliography	113
A	Appendices	123
A.1	Missing values in tabular modelling	123
A.1.1	Data properties	123
A.1.2	Training speed	126
A.1.3	Optuna HPO Results	126
A.1.4	Impact of imputation scheme on HPO	126
A.1.5	TabZilla replication	128
A.1.6	Combining missing value handling strategies	129
A.2	NLP pipelines for textual regression	130
A.2.1	Pipeline component coverage	130
A.2.2	Dataset properties	132
A.3	Evaluating ensembling for text and tabular modelling	134
A.3.1	Results including LANISTR	134

List of Tables

3.1	RMSE results for different imputation schemes	64
4.1	Testing relative BERT and GloVE accuracy	78
4.2	Testing finetuned, not finetuned and bag of words models	80
4.3	Ten best pipelines for obl	82
4.4	Ten best pipelines for jcp	82
4.5	Testing finetuned and bag of words runtime differences	82
4.6	Maximum reduction in mean accuracy for given model choices	83
4.7	Results of variance allocation to model or upstream	85
4.8	Testing accuracy of trees with vector embeddings	85
5.1	Dataset properties	102
5.2	Summary comparing studied methods	103
A.1	Missingness distribution by column, to nearest %	123
A.2	Missingness distribution by row, to nearest %	124
A.3	RMSE results for different imputation schemes with Optuna	126
A.4	Comparison of optimal grid search hyperparameters by imputation	127
A.5	Comparison of optimal Optuna hyperparameters by imputation	128
A.6	Results impact of optimal Optuna hyperparameters by imputation	128
A.7	Tabzilla replication results with missing completely at random (MCAR) injection	129
A.8	RMSE performance of a mixed imputation strategy	130
A.9	Component coverage across random pipelines for the obl dataset.	131
A.10	Component coverage across random pipelines for the jcp dataset.	132
A.11	Samples of obl	133
A.12	Samples of jcp	133

List of Figures

2.1	Illustration explaining batching	25
2.2	Illustration of backpropagation	26
2.3	Illustration of embedding layers	31
2.4	An example residual connection	32
2.5	Pooling mechanisms for transformers	34
2.6	How magnitude of learning rate impacts training	37
2.7	Catboost feature combinations	44
3.1	ResNet MLP model architecture	55
3.2	Dataset partitioning for hyperparameter tuning	57
3.3	Binarize transformation	61
4.1	NLP pipeline summary	70
4.2	Accuracy of finetuned, not finetuned and bag of words models	80
4.3	Illustrations of upstream and model groups.	84
4.4	Boxplots of trees with and without vector embeddings	86
5.1	Differences between fused-representation and ensembling	91
5.2	Example text and tabular application from medicine	93
5.3	Text and tabular model from Gu and Budhkar, 2021	94
5.4	Text and tabular model from Shi et al., 2021	94
5.5	Different ensembling approaches from Shi et al., 2021	95
5.6	LANISTR architecture	97
5.7	LANISTR pretraining	98
5.8	A simple ensemble	100
5.9	Response distribution across datasets	101
5.10	Normalized accuracy boxplots for studied models	104

A.1	Mean standardized claim value against time	124
A.2	Normalized accuracy boxplots, including LANISTR	134

Chapter 1

Introduction

This thesis investigates the inclusion of textual data in regression problems, focusing on applications in price prediction. By price prediction we refer to the regression of prices for goods and services, e.g. consumer goods or expected vehicle repair costs for a car insurer. There are a myriad of regression modelling approaches and regression problems are found in both industry and academia. However most applications use regression models on **tabular** or **structured** data (McElfresh et al., 2023). Tabular data can be thought of as data that fits in a spreadsheet: a set of observations (rows) and for each observation there is a fixed size feature vector, where each feature (column) is typically either numeric or a nominal categorical. Numeric data does not usually require any preprocessing as most machine learning models expect numeric input. While categorical variables must be encoded numerically to act as model inputs, there are some established methods for doing so: one hot encoding (Murphy, 2022, page 23), target mean encoding (Micci-Barreca, 2001) and categorical vector embeddings (Guo and Berkhahn, 2016). However, there is information in other, **unstructured**, data modalities (text, audio, images etc.) that could improve regression models. For example, Seinen et al., 2022 identify that use of textual data in electronic health records significantly improves prognostic classifier accuracies across over 100 medical applications. The use of these unstructured data modalities for regression is relatively under-explored. This thesis will empirically study the extension of regression to specifically textual data: both **text-only** and **text-tabular** (i.e. text with

complementary tabular data). For instance, the prediction of stock price movement from *only* company reports could be considered text-only regression. Alternatively, the prediction of cost to settle a car insurance claim given tabular details of the vehicles (e.g. make and model) and a textual description of the accident could be considered text-tabular regression.

We begin this work by first establishing rigorous baselines with the incumbent tabular-only approaches, to understand how to benchmark text-tabular approaches. There is literature (Gorishniy et al., 2021; Grinsztajn, Oyallon, and Varoquaux, 2022; McElfresh et al., 2023; Tschalzev et al., 2024) comparing gradient boosted decision trees (GBDTs) and deep learning, explained in Chapter 2, that are the two leading approaches in tabular-only modelling. This literature suggests that GBDTs are on average the most accurate approach. However, when it comes to training these tabular-only models on real data it immediately becomes unclear how important certain practical details are, as these are often glossed over if mentioned at all. In particular, missing data is a common issue of real datasets, and it is this problem that forms the first of our research questions: **How much does the choice of missing value handling procedure influence the relative performance of GBDT and deep learning (DL) tabular-only approaches?**

Having established tabular-only baselines we proceed to studying text-only regression. Text-only refers to problems where we seek to predict a continuous value output from *only* a textual value input, e.g. predicting the cost of a home from its listing description. A standard approach to text processing involves the choice of sequential processing steps, forming a pipeline. However, typically these components are chosen without much justification, following a ‘traditional’ (Joshi et al., 2010; Tsai and Wang, 2014) or ‘modern’ (Bitvai and Cohn, 2015; Caron and Müller, 2020) approach, that mirrors the evolution of the field before and after deep learning approaches began to dominate. The lack of systematic evaluation of these choices makes it difficult to know which steps in the pipeline are important, high variance, choices. This knowledge of relative step variance could guide time spent optimizing choices to ones that are more likely to improve performance. Furthermore, the lack of systematic study has left some combinations of choices understudied. This leads us to our second research question: **Which text processing pipeline steps are associated with the greatest impact on performance?**

With baselines in text-only and tabular-only models established we finish by studying text-tabular models, i.e. models that use both data modalities. Text-tabular modelling is a quickly growing field (Seinen et al., 2022) with most studies applying custom approaches to achieve good performance for a specific application. Some work has also begun to study text-tabular modelling as a problem class (Gu and Budhkar, 2021; Shi et al., 2021), instead of through specific applications. There are studies on models that jointly train neural networks with text *and* tabular data in a ‘fused-representation’ approach, along with complex ensembles of such models. However, there is a notable gap in the literature: the study of an ensemble using a specialized text-only and tabular-only model is absent. This is our third and final research question: **How do ensembles of text-only DL and tabular-only GBDT models compare to fused-representation approaches?**

1.1 Thesis outline

The thesis is organized as follows:

- Chapter 2 introduces a technical background for the thesis. The background includes an introduction to natural language processing (NLP) terminology, deep learning, and gradient boosted trees.
- Chapter 3 addresses the first research aim. It presents a case study on a previously unpublished large real insurance dataset. The case study is used to illustrate the importance of missing value handling in benchmarking GBDTs and DL methods for tabular-only modelling. Furthermore, it provides insights for insurance claim reserving. The work in this chapter is adapted from the author’s publication (Larionov, Adams, and Webster, 2024).
- Chapter 4 addresses the second research aim. It introduces a systematic study of NLP pipeline components. In doing so we identify interesting aspects of pipeline construction and study which parts of the pipeline are associated with high variance in model performance.

- Chapter 5 addresses the third research aim. It compares an ensemble using text-only DL and tabular-only GBDT models to other prominent approaches for text-tabular data. We find that the ensemble achieved competitive performance. However, overall, a joint-fusion approach demonstrates more promise for text-tabular modelling.
- Chapter 6 concludes this thesis and discusses directions for future research.

Chapter 2

Background

In this chapter we will cover general background knowledge necessary to understand this thesis. Background that is only relevant to one chapter, and not relevant outside that chapter, is left to said chapter. In Section 2.1 we introduce terminology for natural language processing. In Section 2.2 we cover the deep learning background required to follow this thesis. In Section 2.3 we briefly introduce the gradient boosted decision tree algorithm. In Section 2.4 we go into detail on the Catboost algorithm (Prokhorenkova et al., 2018), a specific GBDT that is used extensively in this work. In Section 2.5 we briefly outline alternative, primarily statistical, methods for mixed data regression not used in this thesis, to contextualize the work. Terms that will be used in the thesis to refer to a specific technical concept are initially written in **bold**.

2.1 Natural language processing

This thesis is concerned with the inclusion of more data modalities when performing regression. While use of tabular data is commonplace in regression tasks, we wish to explore how best to include natural language. Natural language is the language used in day to day communication, e.g. English, that has evolved without rigid logic along with civilization. This natural evolution of language makes it very difficult to describe and model language with a consistent set of unambiguous rules (Bird, 2006). **Natural language processing (NLP)** is the field of research

concerned with using data driven algorithms to automatically gain insight from natural language data.

To use text with modern regression methods we are required to somehow convert the text to numeric features, and then train a model on the numeric features that can ultimately make a prediction. This whole series of steps from raw text to prediction can be broken down into sequential steps, called an NLP **pipeline**. This NLP pipeline comprises four steps:

- **Preprocessing**: maps text to text. The aim is to remove information which is irrelevant and make relevant information easier to extract. Multiple different preprocessors may be used in the same pipeline.
- **Tokenization**: maps text to a sequence of ‘tokens’, which are subsections of the text that will be used in the featurization algorithm.
- **Featurization**: maps a sequence of tokens to a numeric representation.
- **Modeling**: maps a numeric representation to a prediction.

In this section we give a basic background on **tokenization** and **featurization** to familiarize the reader with these core concepts. As specific choices for **preprocessing** and **modelling** are relevant to different chapters we leave them to be explained as and when they are relevant. Specifically, we note Chapter 4 contains more comprehensive introduction on the preprocessing, tokenization, featurization and modelling used in that chapter.

2.1.1 Tokenization

The goal of tokenization algorithms is to choose how to split a given textual input into chunks, or **tokens**, that will be processed by the featurization algorithm in the most effective manner for modelling. The collection of tokens a given tokenization algorithm can produce is called a **vocabulary** of tokens.

On one extreme, one could tokenize character by character, creating a vocabulary of letters and punctuation. However the downstream pipeline would then need to implicitly learn combinations of characters that correspond to a meaning relevant for the task. For example, say we are predicting the price of a house and we have a sentence ‘*this house has three bedrooms*’. Tokenized character by character this would become `t,h,i,s,_,h,o,u,s,e,_,h,a,s,_,t,h,r,e,e,_,b,e,d,r,o,o,m,s` (where `_` is an explicit space character). Then the downstream featurizer would need to convert this tokenized sequence into some numeric form. However, the downstream pipeline would need to learn that when the token `t` is followed by `h,r,e,e,_` then we are discussing a quantity, and when that is followed by `b,e,d,r,o,o,m` then the information is likely highly relevant for the house price. Conversely, when `t` is followed by `h,i,s,_` then it is not always relevant to predicting the house price. To do this for the whole of the english language is a very challenging combinatorial problem, especially with it’s variety of contextual meanings for words. However, we note that there are approaches in NLP that take this character level approach (Xue et al., 2022).

On the other extreme, one could take each unique input and tokenize it as one token, i.e. the sentence ‘*this house has three bedrooms*’ is one token all by itself, essentially treating the text as a categorical feature. The issue with this approach is that due to the variety of natural language this often quickly results in both a large number of unique tokens, and very few examples *per token* that can be used to refine the token’s featurization. Additionally, when running the pipeline on unseen data it is likely that there will be new tokens that do not exactly match those in the vocabulary. As these tokens are not in the vocabulary they must either be ignored or treated as a special ‘unknown’ token.

A compromise between every character and every input is tokenizing every word. The question then becomes how to define a word programmatically. Even excluding languages without spaces (e.g. Mandarin Chinese) it is surprisingly difficult to separate words purely on spaces. For instance, punctuation seems intuitive to remove before splitting on whitespace. However, this can conflate plurals and ownership in the case of apostrophes *John’s* vs *Johns*. Alternatively, periods may be used both as sentence termination markers and in acronyms (e.g. Dr.Smith), so it is unclear how to make a distinct rule to split on sentence termination while retaining

contractions. Word level tokenization can also suffer from declensions and conjugations, which transform the same root word into multiple distinct forms. For example, in English, the verb ‘run’ can appear as ‘runs’, ‘ran’ and ‘running’. Since each variation appears less frequently in training data, the featurization and model have fewer examples to learn the underlying concept.

In general these comparisons strike at a core trade off in tokenization algorithms: the larger your tokens the larger your vocabulary and the less the downstream pipeline needs to combine them to form meaningful units for modelling. However, the larger the tokens the less examples per token you will have to learn from and the more likely you are to encounter unknown tokens at test time.

Instead of attempting to make a rule motivated by some higher level notion of language divisibility (by trying to find meaningful word boundaries or always splitting by character) modern NLP research in DL has attempted to strike a balance by *learning* tokenization algorithms. These learned algorithms, like Byte-Pair Encoding (Sennrich, Haddow, and Birch, 2016), SentencePiece (Kudo and Richardson, 2018) and WordPiece (Devlin et al., 2019), learn tokens of varying sizes. Notably these tokens need not be mutually exclusive, for example the vocabulary may include ‘a’, ‘n’ and ‘an’. Furthermore, they can also learn *subword* tokens such as common prefixes and suffixes. For example, ‘*this house has three bedrooms*’ could be tokenized **this_**, **house_**, **has_**, **three_**, **bed**, **rooms**. As the tokens are not mutually exclusive, there is many ways one could tokenize a given sentence. Correct partitioning of the input text is hence done algorithmically. Typically, this is a greedy algorithm that iteratively merges smaller tokens into longer ones, aiming to maximize resulting token lengths, i.e. tokenizing ‘*an apple*’ as ‘**an**, **apple**’ instead of ‘**a**, **n**, **app**, **le**’.

2.1.2 Featurization

Given a sequence of tokens from a tokenizer, a featurizer needs to map the sequence to a numeric representation. In this section we will briefly explain and contrast the two main approaches for featurization: **bag of words** and **vector embeddings**.

A **bag of words** requires a vocabulary of possible tokens, $\mathcal{V} = \{v_0, v_1, v_2, \dots, v_V\}$. Given a sequence of tokens, $\mathcal{T} = \{t_0, \dots, t_T\}$, for a specific input, bag of words will generate a feature vector $f_i = \sum_j^T \mathbb{I}(t_j = v_i)$ for $i \in \{1 \dots V\}$, where $\mathbb{I}(t_j = v_i)$ is an indicator variable that takes the value 1 if $t_j = v_i$ and 0 otherwise. Plainly put, a bag of words featurization is a vector where each element is the count of occurrences a token, for every token in the vocabulary. As most words do not occur in most samples of text, for large vocabularies, bag of words (BoW) can produce highly sparse vectors, i.e. $\sum_j^T \mathbb{I}(t_j = v_i)$ is 0 for most v_i if $\mathcal{V}$ is sufficiently large.

Instead of the count, some BoW implementations can be a 0/1 indicator of whether the vocabulary token is in the sequence at all. Normalization can also be applied to the counts to down-weight frequent yet less informative words (e.g. ‘*the* and *an*’). An example of such an approach used in this thesis is TF-IDF weighting (Salton, 1991). We note that each *sequence* of tokens is always mapped to a *single* vector of size V , the vocabulary size.

Alternatively, **vector embeddings**, initially proposed by (Mikolov et al., 2013), learn a vector of arbitrary fixed dimension *per token*, for all tokens. In practice, this is done through use of embedding layers, deep neural networks and language modelling tasks. Embedding layers and an example language modelling task are explained in Section 2.2, which covers deep learning background. These vector embeddings can be further **contextual** and **non-contextual**. Non-contextual vector embeddings are essentially lookup tables that always map the same input token to the same vector representation. Contextual vector embeddings, as the name suggests, will have different representations for a given token conditional on context. Typically, this is achieved by applying a deep learning model to a sequence of non-contextual vector embeddings. This model is then trained by some language modelling task to adapt the embeddings to make them contextual. For example, the word ‘bank’ may be featurized differently in the different contexts of rivers and finance. Importantly, unlike bag of words approaches vector embeddings are *per token*. This means there is a vector *per token* in the sequence and the overall sequence embeddings is of *variable size*, matching the token sequence, and often needs to be pooled into a fixed size representation prior to modelling.

2.2 Deep learning

Deep learning (DL) is a vast and rapidly evolving field of machine learning. As such, this background does not attempt to be comprehensive, instead the intent is to provide a self-contained introduction to follow the details of this thesis. For a comprehensive introduction, C. M. Bishop and H. Bishop, 2024 is a recent textbook that is both accessible and detailed.

Section 2.2.1 covers the relevant fundamentals and basic terminology of deep learning. Deep learning models can be considered as a composition of functions, or **layers**, and Section 2.2.2 covers the layers relevant to this thesis. Section 2.2.3 covers the architectures that are mentioned in this thesis. Section 2.2.4 covers details of training a specific architecture, BERT (Devlin et al., 2019), used throughout this thesis. Section 2.2.5 covers other generic training techniques used in this thesis.

2.2.1 Deep learning fundamentals

As with most machine learning problems, the task is to find a function f with a set of parameters Θ that allows us to model data. The distinct nature of DL lies in the choice of function and how the parameters are optimized to fit the data. In DL, f can be very complicated but is typically constructed from a directed acyclic graph (DAG) of relatively simple *differentiable* functions, also called **layers**. This DAG is also often referred to as an **architecture**, and the function f as a **neural network**. The parameters, Θ , for this network are typically randomly initialized (the specifics of which are important but outside the scope of this thesis). These parameters are then optimized to fit the data in a procedure referred to as **training**, after which the model can be used for prediction.

Before training, the training data is commonly shuffled and then partitioned into sequential chunks of fixed size called **batches**. This allows training on datasets that are too large to fit into memory, as well as providing a degree of randomness in the training that can be beneficial, discussed later in this section. The process of batching is illustrated in Figure 2.1.

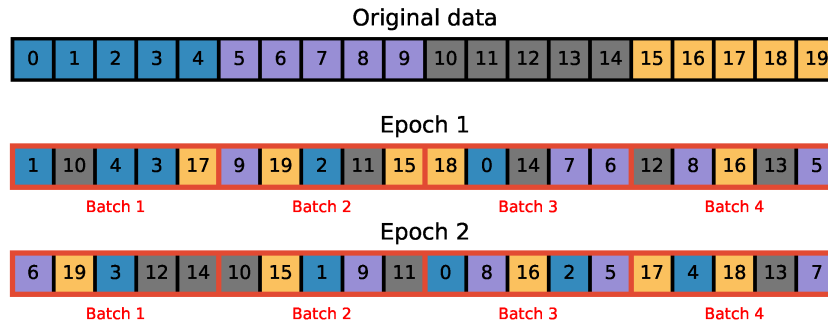


Figure 2.1: Randomized batching of dataset over epochs. Coloring of original dataset is added to aid in visualization of the shuffling.

Given the randomly initialized f , we pass our batched data through f to generate batches of predictions $\hat{y}$. A differentiable loss function, $L(y, \hat{y})$, measures the goodness of fit for the predictions against the true labels y . Then the gradient of the loss with respect to any parameter, $\theta \in \Theta$, can be found by using the chain rule. The DAG nature of our function f allows for efficient calculation of this gradient through an algorithm called backpropagation, briefly explained in Figure 2.2. See C. M. Bishop and H. Bishop, 2024, page 233 for more details on backpropagation. In practice, programming frameworks such as PyTorch (*PyTorch* 2025), Tensorflow (*TensorFlow* 2025) or JAX (*JAX* 2025) are used to automatically apply backpropagation to any arbitrary DAG of differentiable functions, which is important considering the large and intricate architectures used in modern DL.

Given gradients of the loss with respect to any arbitrary θ the network is trained by applying a gradient descent procedure to gradually adapt θ for a good fit. The specific gradient descent procedure is often called an **optimizer**, the most simple of which is stochastic gradient descent (SGD),

$$\theta_{t+1} = \theta_t - \eta \frac{\partial L}{\partial \theta} \Big|_t,$$

where η is a **learning rate** that controls the size of the updates. The smaller η is the less likely SGD is to diverge, however the slower the training and the more likely you are to get trapped in local minima. Local minima are bad for two reasons: i) the loss may be higher than otherwise achievable or, ii) the local minima could represent very good performance on only the training data and for samples outside the training data we would observe poor performance. This latter phenomena is called **overfitting**. We note that the ‘stochastic’ in SGD is due to the random

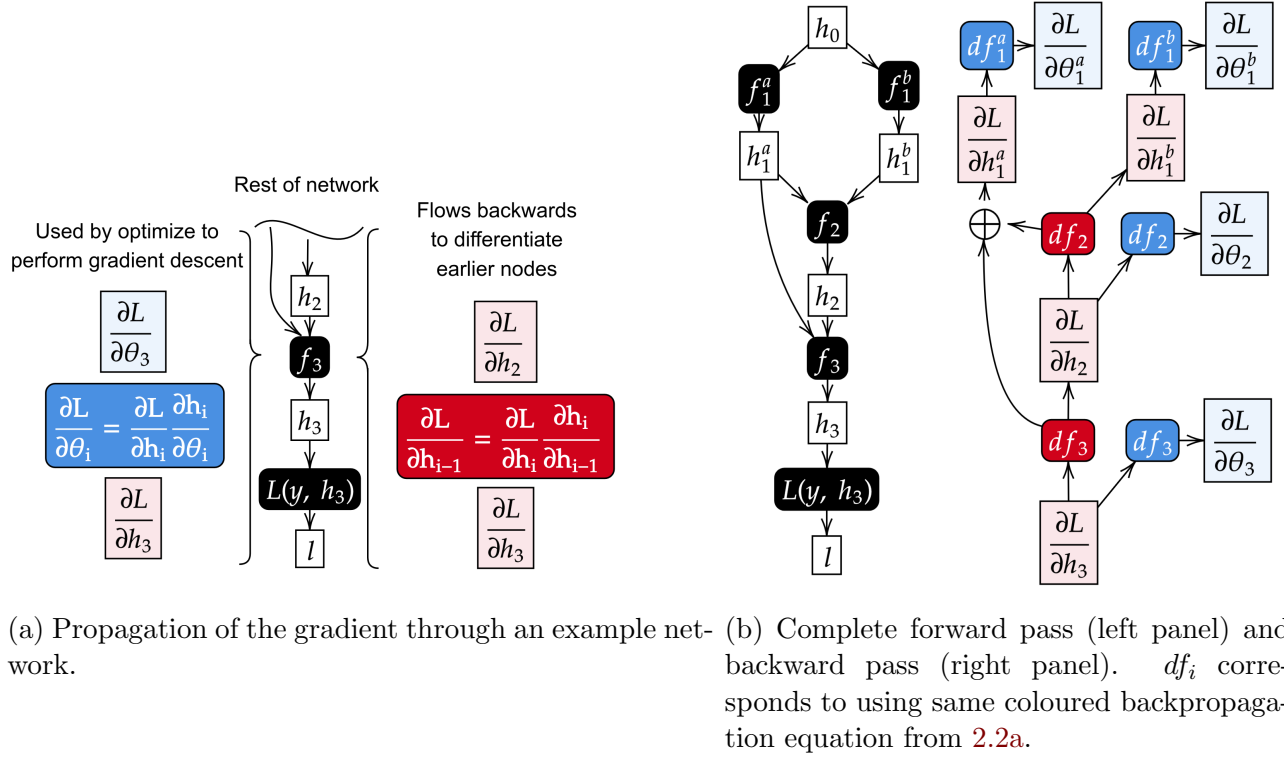


Figure 2.2: How gradients are propagated through a deep learning model in order to calculate $\frac{\partial L}{\partial \theta_i}$. Inputs/outputs are illustrated in square boxes, rounded edges are used to illustrate functions. Chosen architecture uses skip connections (from h_1^a to f_3) to illustrate how backpropagation works with more complex DAGs. h_0 can be considered the input, h_3 can be considered the output. l is the loss obtained from $L(y, h_3)$ where y is the true label. Note $\frac{\partial h_i}{\partial h_{i-1}}$ and $\frac{\partial h_i}{\partial \theta_i}$ are just properties of function $f_i(\mathbf{h}_{i-1})$. Note the summation on the backward pass is due to definition of the total derivative.

nature of the batches used. The random nature of the batches aids in avoiding overfitting by causing stochastic gradients, which can push the parameters out of local minima.

Most optimizers are variants of SGD and are generally of the form,

$$\theta_{t+1} = \theta_t - \eta g \left(\frac{\partial L}{\partial \theta} \Big|_0, \dots, \frac{\partial L}{\partial \theta} \Big|_t, \theta_t \right),$$

where $\frac{\partial L}{\partial \theta} \Big|_0, \dots, \frac{\partial L}{\partial \theta} \Big|_t$ denotes that gradient information from any of the prior steps and g is some function designed to improve the speed, stability or ability to find regions of lower loss. Notably, this still uses a learning rate η and choosing η is a vital part of training a DL model that we cover in more detail in Section 2.2.5. The whole process of evaluating the loss, backpropagating the gradient and using an optimizer to change Θ is called a training **step**. The first training step uses the first batch, the second training step uses the second batch and so on.

A complete pass through the whole dataset, i.e. using every batch, is called a training **epoch**. After every epoch the training data is reshuffled and repartitioned into new batches, the first training step of the new epoch using the first new random batch and so on. Training of the overall DL model is terminated when a stopping criterion is reached. The most common stopping criterion are either a pre-determined number of training epochs or **early stopping**, which will be discussed later (Section 2.2.5).

2.2.2 Layers

In this thesis we will refer to the following layers: **dense**, **ReLU**, **softmax**, **BatchNorm**, **dropout**, **GLU**, **attention** and **embedding** layers. These are described below. Unless otherwise specified, we assume each layer takes a real vector, $\mathbf{h}$, referred to as the input. As layers are composable, this input may not be the global input, i.e. the data, but rather the output from a prior layer.

Dense

A layer that performs affine transformation: $f(\mathbf{h}) = W\mathbf{h} + \mathbf{b}$, where W is a real matrix and $\mathbf{b}$ is a real vector. Both W and $\mathbf{b}$ are trained parameters. This layer is typically followed by a non-linear **activation** layer, to allow for the overall network to be non-linear.

Activations

ReLU and **Softmax** are the main activations used in this thesis. ReLU is a layer which calculates $\max(\mathbf{h}, 0)$ element-wise. This layer has no trained parameters. Softmax is a layer which calculates $\text{softmax}(\mathbf{h})_i = \frac{e^{h_i}}{\sum_j e^{h_j}}$. This layer also has no trained parameters. Notably, softmax maps to the simplex, i.e. has outputs $[0, 1]$ that sum to 1.

BatchNorm

BatchNorm (Ioffe and Szegedy, 2015) is a layer that controls the mean and variance of layer inputs per feature across the batch. This has been empirically shown to stabilize model training. BatchNorm is done in two steps. First, each input feature in $\mathbf{h}$ is standardized for a given batch, such that across the batch a given feature has zero mean and unit variance. Let the standardized $\mathbf{h}$ be written $\mathbf{h}^*$. Second, these standardized inputs are given scale and location through $f(\mathbf{h}^*) = \zeta \odot \mathbf{h}^* + \beta$, where $\odot$ is an element-wise product and ζ, β are vectors of trained parameters and are used across batches. BatchNorm layers operate differently during training and prediction. During *prediction*, means and variances computed cumulatively over the training are used in the standardization process, instead of batch statistics. This ensures deterministic outputs. For details, see Ioffe and Szegedy, 2015.

Dropout

During training, **Dropout** (G. E. Hinton et al., 2012) uses an independent Bernoulli distribution, with probability p , to stochastically set features of the layer inputs, $\mathbf{h}$, to zero. For dense layers, this process is equivalent to setting the associated column of the weight matrix, W , to zero with probability p . This is done independently for each sample during training.

The goal of this layer is a form of training regularization and was originally designed to prevent overfitting by preventing ‘co-adaptation’ (G. E. Hinton et al., 2012). The rationale being to make weights learn features that are useful without over-reliance on other weights. Empirically, dropout has been shown to be an effective form of regularization. During *prediction*, Dropout does not set any layer inputs to zero but instead multiplies outputs by $(1 - p)$. This has the effect of making the layer deterministic while keeping the prediction scale similar to training.

Gated Linear Unit (GLU)

A GLU layer (Dauphin et al., 2017) is a variant of the Dense layer that has the form $f(\mathbf{h}) = (W\mathbf{h} + \mathbf{b}) \odot \sigma(V\mathbf{h} + \mathbf{c})$, where σ is the sigmoid function and $\odot$ is elementwise multiplication. The real matrices W, V and vectors $\mathbf{b}, \mathbf{c}$ are trainable parameters of this layer. Essentially, a GLU layer learns a scaling of the output of a Dense layer; using the output of a sigmoid which is bounded $[0, 1]$. This is intended to ‘gate’ the output of a Dense layer: if the sigmoid is near 0 or 1 it will prevent or allow, respectively, an output to propagate. The original work proposing GLU suggests this form of scaling helps to improve propagation of gradients and empirically demonstrates that it can improve training speeds.

Embeddings

An **embedding** layer is mostly used to represent nominally discrete features as a numeric vector. An embedding does so by mapping each unique level of the discrete feature to a separate vector of trainable parameters. To map K unique levels each to a vector of size m , a matrix, $E \in \mathbb{R}^{K \times m}$, is randomly initialized. Each level is then assigned a unique index, so that the level is represented using the indexed *row* of E . If a row of E is used as an embedding input to the network, gradients are calculated for each of the parameters in the row. When an optimizer updates these parameters it will only update the parameters in rows of E corresponding to inputs, adjusting the parameters to improve the loss and therefore learning a vector representation of the feature. There is no explicit structure imposed on these embeddings, and any structure or relation between them is learned through optimizer updates. This process is illustrated in Figure 2.3. This is an important layer type for DL NLP, as it is

used by DL approaches to featurize tokens into vector inputs.

Attention

A key component of transformers (discussed in Section 2.2.3) is the attention layer. We start with ‘source’ and ‘target’ sequences of vectors, $\{v_{1,0}, \dots, v_{1,S_1}\}$ and $\{v_{2,0}, \dots, v_{2,S_2}\}$, of length S_1, S_2 indexed by i and j respectively. For example, these may be the vector embeddings of text tokens produced by an embedding layer. Assuming these vectors are of consistent dimensionality, $v_{1,i}, v_{2,j} \in \mathbb{R}^d$, we can represent each sequence with matrices $X_1 \in \mathbb{R}^{S_1 \times d_1}$, $X_2 \in \mathbb{R}^{S_2 \times d_2}$. The attention operation is then expressed as:

$$\text{Attention}(X_1, X_2) = \text{softmax}_{\text{rows}} \left(\frac{(X_1 W_q)(X_2 W_k)^T}{\sqrt{d_k}} \right) (X_2 W_v), \quad (2.1)$$

where $W_q, W_k \in \mathbb{R}^{d \times d_k}$, $W_v \in \mathbb{R}^{d_v}$ and $\text{softmax}_{\text{rows}}(\cdot)$ indicates a softmax which is applied separately for each row of the input, $(\cdot)$. Attention aims to produce representations of each source vector that take into account pairwise interactions with target vectors. The query and key matrices, W_q and W_k , project the source and target vectors into a space where the dot product allows matching of ‘relevant’ sources and targets. This means if $(v_{1,i}^T W_q)(v_{2,j}^T W_k)^T$ is large for some i, j then the j th vector in the target is relevant for updating the i th input of the source. The output of the query-key dot product, $(X_1 W_q)(X_2 W_k)^T$, is an ‘attention’ matrix, $A \in \mathbb{R}^{S_1 \times S_2}$, which is normalized by $\sqrt{d_k}$ for training stability. The attention matrix is also normalized across rows using the softmax layer. This makes the resulting rows of A have elements $[0, 1]$ that sum to 1, i.e. lie in the $(S_2 - 1)$ simplex. Finally, W_v is a matrix that projects the input to some ‘value’ space. So $A(X_2 W_v)$ finds a weighted combination (where the weights are in the simplex) for value vectors, $v_{2,j}$, for each given input vector, $v_{1,i}$. An important case of attention is when $X_1 = X_2$; i.e. attention is applied with the source and target sequence the same. This is called self-attention.

Often attention is used in a format called **multi-head attention**. Multi-head attention uses multiple parallel attention layers and projects the concatenated outputs of the attention heads.

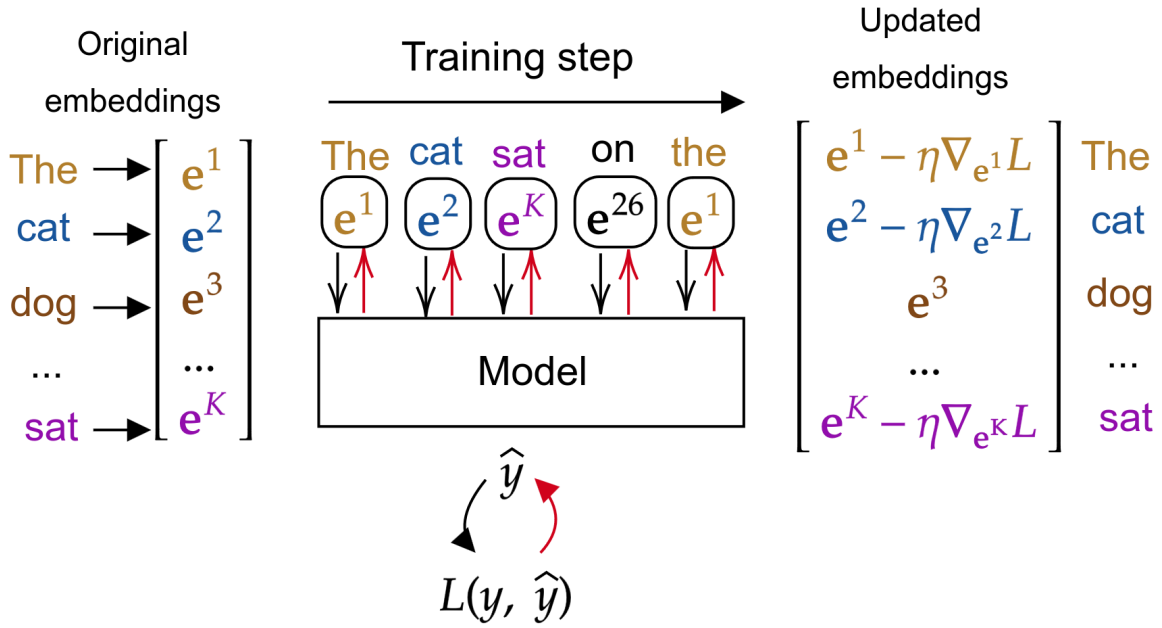


Figure 2.3: Embedding layers and how a training step updates representations associated with specific inputs. Black lines in training step represent a forward pass, red lines represent back-propagation and optimizer gradient descent. *Every element* of the embedding matrix is a trainable parameter, thus learning representations for the input. However, only ‘the’ ‘cat’ ‘sat’ and ‘on’ are updated in this example training step. Note, contribution to gradient of ‘the’ embedding accumulates from *two* uses in the input.

If the multi-head attention uses H attention ‘heads’ then the operation would be

$$\text{MHA}(X_1, X_2) = [\text{Attention}_0(X_1, X_2), \dots, \text{Attention}_H(X_1, X_2)]W_o,$$

where MHA is the multi-head attention function, $[\dots]$ denote concatenation of terms inside the brackets, and $W_o \in \mathbb{R}^{d_v \times d}$ is a learned projection matrix. Note, each head has independently initialized and trained parameters, with the aim of multi-head attention to learn different notions of similarity for each head.

2.2.3 Architectures

In this section we introduce the architectures that we refer to in this thesis. Specifically the **multi layer perceptron (MLP)**, **ResNet** and **transformer** architectures.

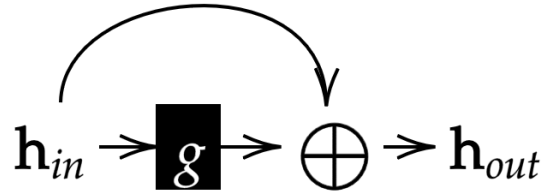


Figure 2.4: A single residual connection, ‘skipping’ the layer g . Where $\oplus$ indicates element-wise addition. Therefore dimension of input and out of g must be the same.

The sequential composition of alternating dense layers and non-linear activation layers (e.g. ReLU) is often referred to as an **MLP**. The phrase ‘N layer’ MLP refers to an MLP with N dense layers.

A residual connection is a general purpose deep learning architectural modification that includes additive skip connections to the network DAG. That is for an input $\mathbf{h}$ the residual connection computes $\mathbf{h} + g(\mathbf{h})$ where g is a layer, or layers, that is ‘skipped’ across, as depicted in Figure 2.4. A **ResNet**, or residual network, is an architecture that makes extensive use of these residual connections. Originally studied for image applications (He et al., 2016a) residual connections have been empirically observed to significantly improve the stability of training, allowing for very deep networks (i.e. many composed layers) to be trained that would otherwise have unstable training dynamics (C. M. Bishop and H. Bishop, 2024, page 275).

Transformers are a class of models originally proposed in Vaswani et al., 2017 that leverage multi-head attention for sequence to sequence modelling. These models have gained widespread popularity due to their empirically excellent performance in both text and image related tasks. The original transformer is comprised of two main components: an ‘encoder’ and a ‘decoder’. It is worth noting that references to a ‘transformer’ in the literature can refer to an encoder-only, a decoder-only or a joint encoder/decoder architecture. Occasionally even architectures with little relation to Vaswani et al., 2017 will be called transformers for the appeal of the nomenclature. For clarity we will use the term transformer to refer exclusively to encoder-only architectures that use self-attention throughout.

The technical details of how transformers work are somewhat involved and not central to this thesis. Although we described attention in Section 2.2.2, the reader can treat them as a

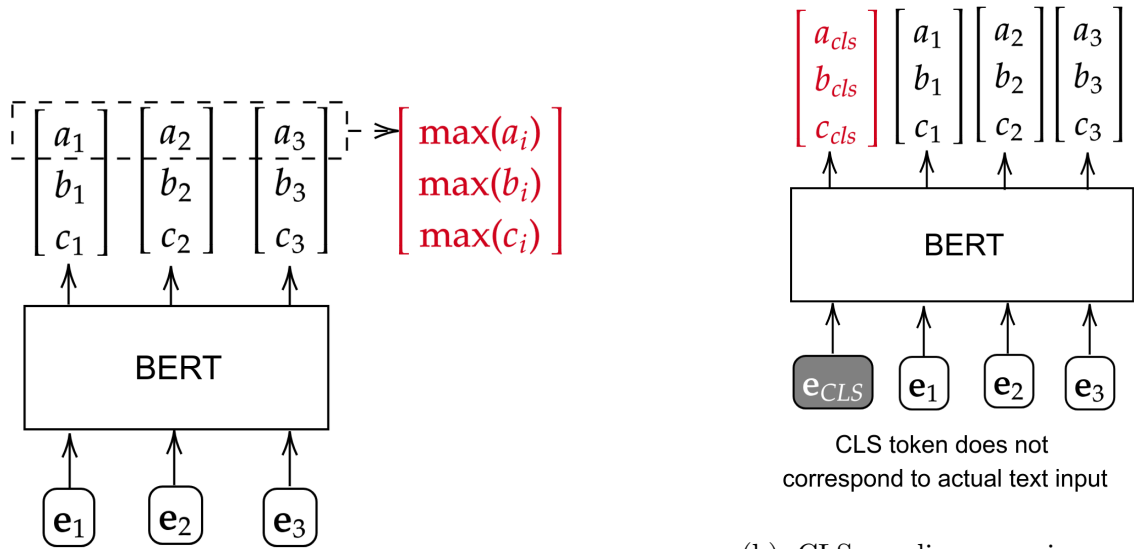
complicated black-box layer with many parameters. For further technical detail, transformers are well described in the original work by Vaswani et al., 2017. However, what is relevant is that encoder-only transformers are sequence-to-sequence models, meaning they take in a sequence of vectors and output an aligned sequence of vectors. In this work we mainly use BERT (Devlin et al., 2019) (or variants thereof) as an NLP transformer for processing textual inputs. BERT takes embeddings of tokens (discussed in Section 2.1) as inputs and produces output vectors trained to predict inputs that were intentionally dropped, or ‘masked’. This training procedure is called masked language modelling and is covered in more detail in Section 2.2.4.

The goal of NLP transformer training is to create a general purpose language model which allows for output vectors to contextually represent input tokens, i.e. there is an output vector *per input token* that changes with context. However, for tasks such as regression we typically want an overall prediction taking into account *every token*. To convert the variable length sequence of vectors representing each token to a fixed size representation of the whole text input, a ‘pooling’ operation is typically performed. Common pooling operations are illustrated in Figure 2.5. One such pooling operation is an elementwise mean or elementwise max across vector outputs, as shown in Figure 2.5a. However specifically BERT, and derivative architectures, also have the option to use ‘CLS’ pooling. CLS pooling involves prepending a so called ‘CLS’ token to each text input and training the output corresponding to this token to represent the overall sequence as shown in Figure 2.5b. The specifics of this CLS token and its training are discussed in Section 2.2.4.

2.2.4 BERT training

The BERT, or **B**idirectional **E**ncoder **R**epresentations from **T**ransformers, architecture was proposed in Devlin et al., 2019, along with two novel training tasks that were simultaneously used for training. These tasks were **masked language modelling** and **next sentence prediction**.

Masked language modelling involves randomly replacing a proportion of input tokens with a so called ‘mask’ token. Notably, as we are just removing words present in existing text it is



(a) Elementwise pooling. Diagram uses max for example but the mean is also commonly used.

(b) CLS pooling, requires use of BERT or a derivative architecture. CLS output is trained to represent the overall input sequence.

Figure 2.5: Common pooling mechanisms for transformers to convert sequence outputs to fixed size representations. Pooled representation for sequence in red. Input token vector embeddings are represented with $\mathbf{e}$.

easy to generate huge amounts of training data for this task just given sufficient arbitrary text. Then the network is trained to predict the word in the masked position at the output. This is done by passing each output vector through a classification MLP and then evaluating whether the masked tokens are classified as their original token. The idea behind this training is that if the model can predict masked tokens then it has some implicit general language modelling ability. Additionally, to perform well at predicting masked tokens, this implicit model should be capable of representing input text in a contextual manner. By ‘contextual’ we mean the model produces different output vectors for the same input token given context that alters the token’s interpretation. For example, the output vector for the input token ‘bank’ should be different in contexts about rivers or finance.

In the **next sentence prediction** task, two sentences are sampled from the data, either a pair that actually follow each other or two disconnected sentences. Importantly, it is easy to generate huge amounts of training data for this task because of the simplicity of the sampling. Then a so called CLS token is prepending to the concatenated sentences. Ultimately, using just the output associated with the artificial CLS token, an MLP classification layer is trained to

predict whether the second sentence follows the first. The idea of the next sentence prediction task is to learn an aggregation procedure that captures sentence level representations instead of token level representations, and encode this in the output associated with the CLS token. We also note the CLS token is always prepended, irrespective of the task, to every input and thus often used as a pooled output for tasks other than next sentence prediction.

The masked language modelling and next sentence prediction tasks are examples of so called **unsupervised**, or ‘self-supervised’ training. Unsupervised training does not use expensive manual labels, and instead the labels are automatically generated depending on the task. This automatic label generation allows training to leverage huge amounts of text data. However, training on such huge datasets is expensive in both time and compute. To save on training costs, research groups such as those that created BERT (Devlin et al., 2019) will release **pre-trained** models. These pretrained models have already been trained on unsupervised tasks. This pretraining can have gone through potentially millions of training steps on datasets with hundreds of thousands, millions, or billions of tokens. Downstream users of these pretrained models typically discard the MLPs used for the pretraining objectives and then adapt, or **fine-tune**, the remaining models to other tasks, such as regression. Finetuning will be described in more detail in Section 2.2.5.

2.2.5 Other training techniques

The overall training procedure (random initialization, backpropagation and optimizer updates until a stopping criterion) outlined in Section 2.2.1 is often modified to improve the quality of the resulting model. This section will discuss the training techniques used in this thesis.

Early stopping

Early stopping is a procedure for determining when to terminate the training. Periodically the model is evaluated on a withheld ‘validation’ dataset that is disjoint from the training data.

The period between evaluations is typically one epoch. The best validation performance across these epochs is stored and training terminated if it is not improving. Concretely, early stopping uses a counter that is reset to zero whenever a new best validation performance is achieved. After each epoch, if the best performance is not beaten the counter increments by one. Once the counter crosses a threshold, called the ‘patience’ of the early stopping, then the training is terminated as the model has not improved for a given number of epochs. Often the parameters of the model are also reset to the parameters that achieved the prior best performance, which was not surpassed. This termination procedure aims to reduce overfitting and can significantly speed up training relative to fixed epoch termination criteria.

Weight regularization

Weight regularization involves including the norm of certain weights in the loss function. For example, the phrase ‘L2 regularization of kernel weights’ refers to inclusion of the parameters from the weight matrix for a dense layer in the loss. Concretely, if the original loss function is L then the loss function of a network with an L2 regularized dense layer would become

$$L^*(y, \hat{y}, \theta) = L(y, \hat{y}) + \lambda \sum_{\theta: \theta \in W} |\theta|^2,$$

where W is the matrix used in the dense layer being regularized and λ controls the strength of regularization and the 2 in L2 describes the exponent in the sum. The higher λ is, the more the training procedure will learn lower valued parameters even at the expense of worse performance, as measured by the unregularized loss L . Weight regularization is a common technique to reduce the ‘complexity’ of the model, by penalizing high value parameters. This in turn is done with the goal of reducing overfitting (C. M. Bishop and H. Bishop, 2024, page 253).

Learning rate scheduling

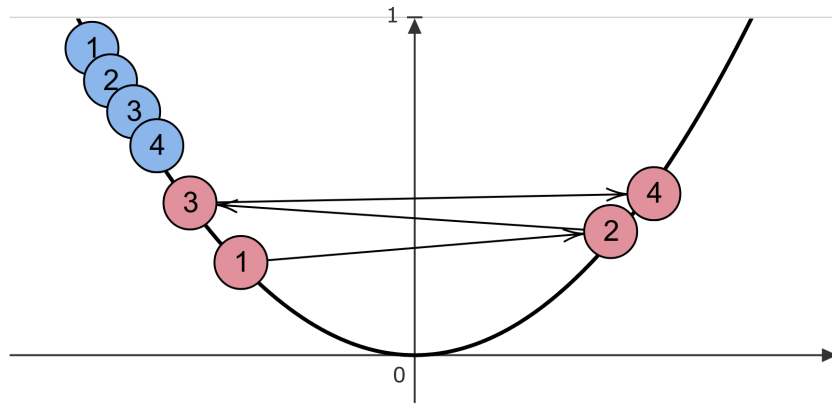


Figure 2.6: Illustration of sequential steps following small and large learning rates in a simple 1D quadratic gradient descent. Notice that if we start far away from the optimum, with a small learning rate (blue steps) we will take a long time to reach the optimum at $x = 0$. Alternatively, if we are close and the learning rate is too large (red steps) we will be unable to step into the region of better loss.

As described in Section 2.2.1, almost all optimizer algorithms have a notion of a **learning rate** that controls the relative size of weight updates for each training step. In practice, at the beginning of a training run the random initialization may be far from a ‘good’ set of parameters, and therefore it can be advantageous to take larger steps. However, later in the training run once the model is near a ‘good’ set of parameters taking a step that is too large may prevent descent down a more narrow loss valley, illustrated in Figure 2.6. Therefore, it is common practice to use a **learning rate schedule** that changes the learning rate throughout the training algorithm. This learning rate schedule typically starts out high and decreases throughout the training, for example following an exponential or linear decay every certain number of steps. However, sometimes the learning rate schedule can follow an oscillatory pattern. For example, the recommendations of Howard and Ruder, 2018 for adapting pretrained language models suggest following a triangular learning rate. This triangular rate linearly increases from zero to some prescribed maximum value over a number of *warmup* epochs and then linearly decreases the rate back to zero over a prescribed number of epochs. The initial learning rate of the schedule can have a large impact on the training dynamics of the network. It is typically found through empirical trial-and-error, as part of the broader process of **hyperparameter optimization**.

Hyperparameter optimization

Deep learning architectures, along with the inclusion or exclusion of training tricks, often lack theoretical justification for specific choices. For example, the output shape of the weight matrix in a dense layer; regularization λ parameters; the initial learning rate and learning rate schedule are all important choices in the design and training of a network but have no theoretical justification in their selection. As such, in practice these choices are often chosen through a meta optimization procedure, called **hyperparameter optimization (HPO)**. HPO involves subjectively selecting a range of ‘reasonable’ values for each hyperparameter that is deemed important and then searching through the resulting possible combinations. This can be done with a grid search, purely randomly, or through more elaborate methods (Akiba et al., 2019). In practice, HPO is an important element of any deep learning model and can be just as, or more, important than choosing the right ‘family’ of model, e.g. transformer, MLP, ResNet (Kadra et al., 2021).

Unsupervised and supervised finetuning

As described in Section 2.2.3, large transformers for language modelling such as BERT will typically be released in a pretrained state. However, the pretrained model may not achieve good performance for tasks other than those it was pretrained on. As such, if we wish to perform a task such as regression these models need to be **finetuned**. Finetuning typically comprises two stages (Howard and Ruder, 2018): unsupervised finetuning and supervised finetuning.

Unsupervised finetuning trains the model using the same objectives as the pretraining tasks but on specifically the text used for the new application. This unsupervised finetuning is used to adapt the implicit language representation of the model to better model the specific language of your problem. This step of finetuning has been empirically validated and is especially important if your textual data is different in style or distribution to the original pretraining text. This difference in token distribution is common in practice, unless you are performing your task on

the internet forums, books and Wikipedia articles used in the transformer’s pretraining.

Supervised finetuning on the other hand introduces a layer (often an MLP, also called a **head**) that acts on the, typically pooled, output of the transformer. The model can then be trained to predict the labels associated with each input for your application. For example, the price associated with a textual description in an auction (regression) or the sentiment associated with a review (classification). Howard and Ruder, 2018 also propose other training tricks for finetuning models which we describe below.

Layerwise learning rate decay

Howard and Ruder, 2018 propose using **layerwise learning rate decay** when performing both unsupervised and supervised finetuning. The core idea is that the learning rate should vary not only with epoch, following a learning rate schedule, but should also decrease as the weights you alter get closer to the input. This aims to reduce the magnitude of changes that are applied to the encoder model, especially in earlier layers which are hypothesized to have extracted generic linguistic features that we do not wish to destroy by updating too much during finetuning.

Freezing and thawing

Howard and Ruder, 2018 also propose a gradual **unfreezing** training procedure. When performing *supervised* finetuning we should initially freeze the transformer encoder, only training the task specific head of the model. To **freeze** a layer or layers means to not apply any optimizer updates to it’s parameters. Freezing is done as the initially random task specific head will likely predict very badly, creating large errors that when backpropagated could destroy the implicitly learned linguistic features of the encoder. Instead, under initial frozen training, only parameters in the randomly initialized head are changed until some stopping criterion, then the rest of the layers can be **thawed** (i.e. begin to receive optimizer updates). In this thesis we will refer to these two supervised finetuning stages as **frozen supervised finetuning** and **un-**

frozen supervised finetuning. Howard and Ruder, 2018 originally proposed incrementally thawing one more layer per finetuning epoch, starting from the head. However, in this thesis, we initially freeze everything except the randomly initialized head until a stopping criteria is triggered and then thaw *everything* (Keras Team, 2024).

2.3 Gradient boosted decision trees

In this section we will present a high level description of gradient boosted decision trees followed by a more detailed treatment that is required to understand the novel elements of Catboost.

Boosted models learn an additive ensemble of ‘weak learner’ models. **Gradient boosted models** learn an additive ensemble that represents a gradient descent in a functional space. **Gradient boost *decision trees*** are gradient boosted models where the model class trained for inclusion in the ensemble is a **decision tree** (Breiman et al., 1984).

More precisely, if T is the total number of weak learners we want to use, a boosted model would be:

$$F_T(\mathbf{x}) = G_0(\mathbf{x}) + \sum_{t=1}^T \eta G_t(\mathbf{x}), \quad (2.2)$$

where $G_t(\mathbf{x})$ is the t th ‘weak learner’, η is a weighting factor, and $G_0(\mathbf{x})$ is an initial estimate, such as the mean response of the training set.

Usually boosted models are built in a sequential fashion, e.g. the i th model incorporating i weak learners would be $F_i = F_{i-1} + G_i$ for $i = 1, \dots, T$. The sequential construction of the model enables the procedure to be terminated early, i.e. return $F_i(\mathbf{x})$ with $i < T$, if validation performance is not improving.

For *gradient* boosting, the summands $G_t(\mathbf{x})$, $t \in \{1, \dots, T\}$, are chosen from within a hypothesis class of functions $\mathcal{G}$ to approximate $-\frac{\partial L}{\partial F}(y, F(\mathbf{x}))|_{F_{t-1}}$, the negative functional derivative of the loss. This evaluation of the negative functional derivative of the loss is also called a pseudo-residual and denoted $r_{t-1}(y, \mathbf{x})$ (Friedman, 2001).

To evaluate $r_{t-1}(y, \mathbf{x})$ requires knowledge of both $\mathbf{x}$ and y . As y is unavailable outside the training set, $r_{t-1}(y, \mathbf{x})$ can only be evaluated on the training data. However we can approximate r_{t-1} with a given summand $G_t(\mathbf{x})$ and measure of function fit, $L'(r_{t-1}(y, \mathbf{x}))$:

$$G_t = \arg \min_{G \in \mathcal{G}} \sum_{(\mathbf{x}, y) \in \mathcal{D}} L'(r_{t-1}(\mathbf{x}, y), G(\mathbf{x})). \quad (2.3)$$

In practice finding the true arg min is infeasible so G_t is some approximation learned following a standard algorithm to minimize L' .

L' can be different from L as it is used to fit $G_t(\mathbf{x})$ to $r_{t-1}(y, F(\mathbf{x}))$ to enable derivative evaluation on data outside the training set. It is the addition of G_t to the ensemble that contributes to the minimization of L given a small enough step size η .

This results in the boosted ensemble approximating a gradient descent of the loss functional (in the space of linear combinations of $\mathcal{G}$) with constant learning rate η :

$$F_t = F_{t-1} + \eta G_t \approx F_{t-1} - \eta \left. \frac{\partial L}{\partial F} \right|_{F_{t-1}} \quad (2.4)$$

In the context of GBDTs; the weak learner is a decision tree (Breiman et al., 1984).

2.4 Catboost

Catboost is a GBDT that i) proposes novel alterations to the learning algorithm to reduce overfitting; ii) proposes a novel categorical handling scheme and iii) implements many engineering tricks to improve the speed of fitting. For the self-contained nature of this thesis, and as the original work (Prokhorenkova et al., 2018) can be difficult to follow, this section will provide a detailed account of the algorithm. Some of these details are not present in the original manuscript and are only described in the package documentation (Catboost, 2024b) or not at all and must be reverse engineered through the code (Catboost, 2024a).

A key feature of the Catboost algorithm is that it relies on an arbitrary ordering of the data.

We denote this ordering through the subscripts on $\mathbf{x}$ and y , for example $\{\mathbf{x}_j : j < i\}$ would denote the set of feature vectors, $\mathbf{x}$, that are before index i in the ordering.

2.4.1 Ordered boosting procedure

When fitting G_t , for a given $(\mathbf{x}_i, y_i)$ and some step t , traditional GBDT methods will fit r_{t-1} evaluated on *all* datapoints, including $(\mathbf{x}_i, y_i)$. If we include the training data dependence in our notation of F_t as $F_t|\mathcal{D}$, we can write the next residual r_t as:

$$r_t(\mathbf{x}_i, y_i) = - \left. \frac{\partial L}{\partial F_t} (y_i, F_t(\mathbf{x}_i)) \right|_{F_t|\mathcal{D}}.$$

Prokhorenkova et al., 2018 note that $r_t(\mathbf{x}_i, y_i)$ will be a biased estimate of the gradient as it will be evaluated on F_t which is itself trained on $(\mathbf{x}_i, y_i)$. Loosely speaking, the reason for this bias is similar as to why performance evaluation should not be done on the training set, and needs to be done on an separate dataset. A more rigorous proof of the existence of this bias is present in Prokhorenkova et al., 2018.

Catboost proposes ‘Ordered boosting’ to remove this bias by evaluating r_t on different data from that which the F_t are trained on, specifically on $\{\mathbf{x}_j, y_j : j < i\}$ which naturally excludes $(\mathbf{x}_i, y_i)$:

$$r_t(\mathbf{x}_i, y_i) = - \left. \frac{\partial L}{\partial F_t} (y_i, F_t(\mathbf{x}_i)) \right|_{F_t|\{\mathbf{x}_j, y_j : j < i\}}.$$

To train a separate GBDT model for each of the n datapoints has a complexity of $O(nm)$ where m is the complexity of the GBDT training algorithm, which is itself *more* than linear in the data and with dependence on the number of features. Hence, the overall complexity is *more* than $O(n^2)$, which can quickly get computationally expensive with dataset size. In practice, the number of models trained is substantially decreased, as described in Section 2.4.3.

2.4.2 Categorical handling

The Catboost categorical encoding algorithm is an adaptation of target mean encoding (Micci-Barreca, 2001). Typically, for a categorical feature, x' , of $\mathbf{x}$, a target mean encoding represents all occurrences of category c_k with e_k ,

$$e_k = \frac{\sum_i^N y_i \mathbb{I}(x'_i = c_k)}{\sum_i^N \mathbb{I}(x'_i = c_k)},$$

the mean response value for all responses that are associated with c_k . Where $\mathbb{I}(p)$ is an indicator variable - taking the value 1 if p is true, and 0 otherwise.

Catboost amends this to an ‘Ordered’ target mean encoding,

$$e_{k,j} = \frac{\sum_i^{j-1} [y_i \mathbb{I}(x'_i = c_k)] + ap}{\sum_i^{j-1} [\mathbb{I}(x'_i = c_k)] + a},$$

note the change in the upper limit of the sum, where $e_{k,j}$ is the encoding of category c_k for datapoint with index j , p is a ‘prior’ value that the encoding will take if the categorical variable has not been seen before and a is a weighting term used to control the influence of this ‘prior’. The larger a , the closer the encoding will be to p . p is typically the global mean. The encoding $e_{k,j}$ represents the mean response value for all responses that are associated with c_k *for data before* $(\mathbf{x}_j, y_j)$, similar to the ordered boosting procedure. The dependence of e on j means the categorical encoding could be different for each datapoint.

Much like the ordered boosting procedure, ordered target mean encoding also has complexity $O(nm)$ where instead m is the complexity of training a mean encoding. As $O(m)$ will be linear in the data - with further dependence on the number of unique categories - ordered categorical encoding will be at least $O(n^2)$. How this complexity is controlled is described in Section 2.4.3.

Catboost further adapts categorical handling through a process named ‘Feature Combinations’. At any given decision tree split, if a categorical feature, x' , is used for the split then all subsequent split decisions will use an altered dataset. Specifically, all categorical variables will be concatenated with the previously used feature and the categorical encodings for each feature

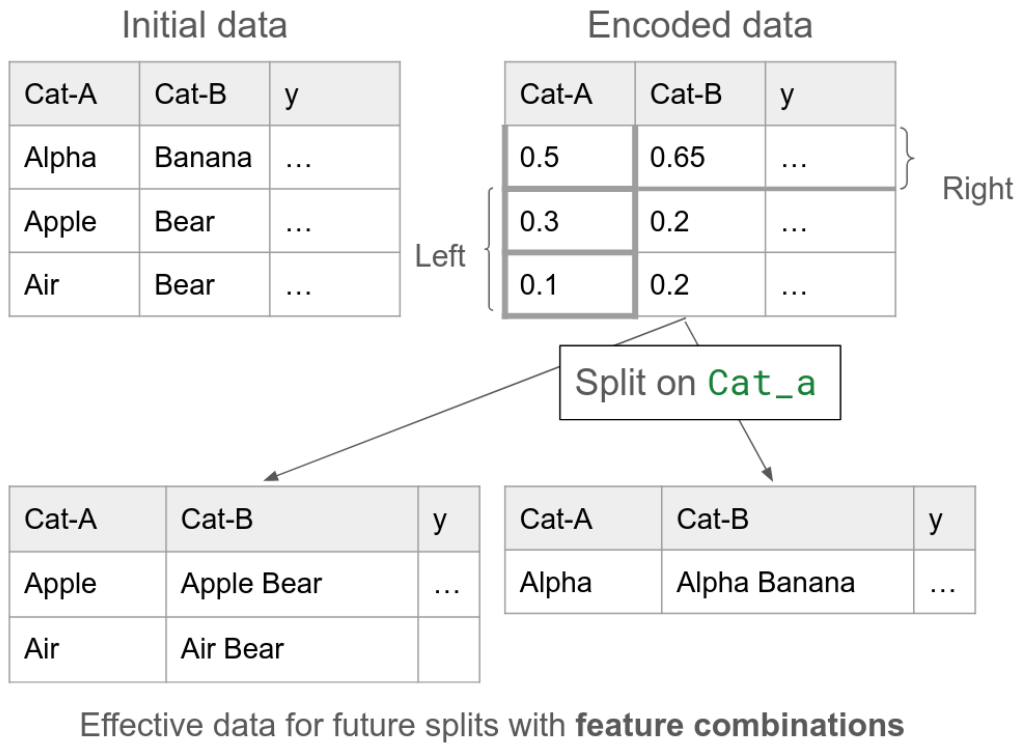


Figure 2.7: Example of how feature combinations affect data considered for decision tree growth. Note how what was previously identical categories ‘Bear’ in **Cat-B** are now two separate categories due to an upstream split on **Cat-A**

recomputed. This typically results in higher cardinality categorical features. This is shown in Figure 2.7.

2.4.3 Further Catboost specifics

In this section we describe how the ordered boosting and ordered target mean encoding are made to scale better computationally. We also describe a label quantization strategy used prior to target mean encoding for categorical variables.

In order to attain a feasible computational complexity for ordered boosting and ordered target mean encoding, Catboost does not train a separate G_t or $e_{k,j}$ for *every* datapoint. Instead, the training data is partitioned into chunks of exponentially expanding size, e.g. the first chunk contains the first datapoint, the second chunk contains the next 2 datapoints, the third chunk contains the next 4 datapoints etc. up to $\log_2(n)$ chunks. Then when evaluating the ordered r_t for boosting or $e_{k,j}$ for categorical encoding, all of the points in a given chunk only use data

up to the boundary of the previous chunk. So if we denote $(\mathbf{x}_i, y_i)$ as belonging to chunk with index $z : z(i) = \lfloor \log_2(i) \rfloor$ then we get ordered boosting:

$$r_t(\mathbf{x}_i, y_i) = - \left. \frac{\partial L}{\partial F_t}(y_i, F_t(\mathbf{x}_i)) \right|_{F_t|\{(\mathbf{x}_j, y_j) : z(j) < z(i)\}},$$

meaning we only train $\log(n)$ separate models for evaluating r_t . This in turn reduces the complexity of the ordering procedures to $O(\log(n)m)$ where m is the complexity of training a GBDT for ordered boosting. For categorical encoding the formula then becomes:

$$e_{k,j} = \frac{\sum_{i:z(i) < z(j)} [y_i \mathbb{I}(x'_i = c_k)] + ap}{\sum_{i:z(i) < z(j)} [\mathbb{I}(x'_i = c_k)] + a},$$

also meaning that only $\log(n)$ target encoding means need to be calculated. This in turn reduces the complexity of the ordering procedures to $O(\log(n)m)$ where m is the complexity of target mean encoding for categorical encoding.

However, whilst the complexity is decreased by chunking the data, it is partially increased to reduce dependence on the arbitrary ordering. The ordering of the datapoints is permuted randomly a number of times, and the ordered boosting gradients and ordered target mean encodings for a given datapoint are averaged over those calculated in each of the data permutations. This is to reduce the variance associated with learning G_t or $e_{k,j}$ that are derived from a low number of prior datapoints, for example calculated on the chunks of size 1, 2 etc.

Beyond adapting the data used for the training algorithm and calculation of categorical features, Catboost alters the definition of a target mean encoding specifically for regression problems, which are the only problems considered in this thesis. Prior to calculations of the target mean encoding, the response y is ‘quantized’, i.e. discretized, into B bins, where B defaults to 2. The bin locations are chosen to maximize the log-sum of unique value counts in each bin, $\sum_b \sum_{u(b)} \log(n_{u(b)})$ where $u(b)$ are the unique values in bin b and $n_{u(b)}$ is the respective count of the unique value. Price responses are usually unique in every y , and for unique responses this amounts to evenly dividing the datapoints into B bins. For each bin, b , a binary response is created and an encoding $e_{k,j,b}$ is performed using this binary response. The overall the target

mean encoding is then made a vector of binarized target mean encodings representation for each category, $e_{k,j} \in \mathbb{R}^b$. This discretized target mean encoding is still ‘ordered’ and so respects the partitions made for ordered target mean encoding and ordered boosting. It is not explained in the paper, documentation or code why this quantization of the response is done prior to target mean encoding.

2.5 Alternative statistical approaches

This thesis uses relatively ‘modern’ approaches to study the combination of textual data with tabular data for price regression, mainly focusing on ‘machine learning’ approaches such as GBDTs and DL. Although machine learning was built upon concepts initially developed in statistics (e.g. concept of regression), it has somewhat diverged and now is a field of algorithmic and computation-first approaches. We note that there is also a strong wealth of literature on *statistical* models for tabular data. Generalised linear models (GLMs) (McCullagh, 2019) and support vector machines (SVMs) (Hearst et al., 1998) are extensively studied approaches to regression in the statistical field. These modelling algorithms could be extended to handle text through concatenating BoW, or pooled vector embeddings, to the tabular data. For example, there is work studying use of SVMs with BoW (Soguero-Ru  z et al., 2016) or use of GLMs with vector embeddings (Lee, Manski, and Maiti, 2020). Furthermore, there exists kernels defined on text (e.g. string subsequence kernels, proposed by Lodhi et al., 2002) and through combination of these kernels with more ‘typical’ kernels for numeric data (e.g. RBF, linear and periodic), one can apply any kernel regression approach (e.g. Gaussian processes, SVMs). However, we opt to focus this thesis on GBDT and DL approaches as these are the state-of-the-art approaches for tabular data (McElfresh et al., 2023). Furthermore, DL specifically has caused a paradigm shift and step change in performance for many NLP tasks (e.g. translation in Vaswani et al., 2017 and perhaps most notably language generation in Radford, Wu, et al., 2019) making it a reasonable candidate for detailed study in text-tabular regression.

Chapter 3

Investigating the impact of missing value handling on Boosted trees and Deep learning for Tabular data

This chapter is an adaptation of the author’s published work: Larionov, Adams, and Webster, [2024](#).

3.1 Introduction

Many machine learning problems involve regressing or classifying with *tabular* or *structured* data. Since their introduction, GBDTs have performed well on such tabular data (Friedman, [2001](#)). Meanwhile, DL has become the state of the art in many problems that involve *unstructured* data, e.g. images (He et al., [2016b](#); Simonyan and Zisserman, [2015](#); Tao, Sapra, and Catanzaro, [2020](#)), audio (Ao et al., [2022](#)), text (Baktha and Tripathy, [2017](#); Touvron et al., [2023](#); Ziegler et al., [2019](#)), and their combinations (Radford, Kim, Hallacy, et al., [2021](#); Ramesh et al., [2021](#); Rombach et al., [2022](#)).

Naturally, with the rise of research in DL, architectures have been proposed that claim to

outperform GBDTs on tabular data (Huang et al., 2020; Kadra et al., 2021; Shavitt and Segal, 2018; Somepalli et al., 2021). However, there is a lack of consensus on whether these architectures really are more accurate. Large studies (Borisov et al., 2022; Grinsztajn, Oyallon, and Varoquaux, 2022; McElfresh et al., 2023) have compared DL to GBDTs across many datasets, many tasks and different computational budgets and suggest that GBDTs are on average the more accurate model for tabular data. The proposed reasons for the performance edge of GBDTs in these large studies remains an area of active research. Theories include the ability of GBDTs to ignore irrelevant variables and model discontinuous functions (Grinsztajn, Oyallon, and Varoquaux, 2022). Importantly, neither the proposed DL architectures nor the large studies investigate the impact of missing values. As missing values are common in real tabular data (Van Ness et al., 2023), and missing value handling can significantly impact the results of analyses (Jin et al., 2021), this is a significant gap in the literature.

In the remainder of this paper we use a claim reserving application with data from a large UK car insurer to shed light on the comparison of GBDTs to DL, with a particular focus on the impact of missing value handling. We specifically study this real-world dataset because it is characterized by extensive missing values that are explicitly *not* missing completely at random (MCAR) – offering a more realistic scenario than injection of MCAR missing values into previously studied complete datasets.

We investigate Catboost (Prokhorenkova et al., 2018) and three DL architectures: TabNet (Arik and Pfister, 2021) and two ResNet-inspired MLPs. TabNet was chosen as a specialized tabular DL architecture with prior validation in insurance (McDonnell et al., 2023). Catboost and ResNets were chosen as the respective best GBDT and DL model from McElfresh et al., 2023.

In Section 3.2 we present background on our application as well as describe the car insurance dataset that we analyze. In Section 3.3 we give a more detailed description of the modelling strategies we compare. In Section 3.4 we cover three elements of methodology. Section 3.4.1 covers our approach to hyperparameter tuning, where extra care was taken to avoid bias. Section 3.4.2 covers the experiments used to investigate the impact of missing value handling.

Section 3.4.3 briefly covers experiments studying injection of MCAR missing values into other previously studied complete datasets. In Section 3.5 we present and discuss the results on our insurance dataset, not only finding Catboost is the more accurate model for our data, but also highlighting the importance of missing value handling in model accuracy.

3.2 Background

Car insurance is an important financial service with a 2024 global value of over 1.9 trillion USD, which is estimated to reach over 2 trillion USD by 2028 (Statista, 2024). Car insurance works on the principle that insurers charge customers a *premium* in return for obligations to provide financial support in the event of contractually agreed risks. Accurate pricing is vital for both the sustainable profit of the insurer and fair prices for customers. The process of determining a price for a prospective customer in car insurance is complex (Olivieri and Pitacco, 2015; Werner and Modlin, 2010). It comprises of three core steps i) estimating the expected value of payments to the customer over the duration of the contract ii) estimating current liabilities for claims that are *reported but not settled (RBNS)* and iii) somehow sensibly combining the two prior estimates into a price. The first step typically comprises finding a model for *claim frequency* and a model for *claim severity*; the latter estimating the cost of a claim conditional on an accident. The second step comprises modelling the cost of claims conditional on them having already occurred and is called *claim reserving*. The third step combines the claim frequency, claim severity and claim reserve estimates using risk models and business considerations such as profit margins, legal requirements, risk appetite and operational costs.

The focus within this work will be on the second step: claim reserving. Specifically, we focus on *outstanding claim reserve* modelling which is the process of predicting costs for claims that have been RBNS.

Typically, outstanding claim reserve modelling is mainly done on a portfolio level. In other words, insurance companies predict the overall reserve requirement for a given time period, say a quarter, across all customers. Importantly, these forms of claim reserve modelling use no

individual claim information, instead using historic portfolio-level settlement aggregates. This is done with deterministic algorithms such as run-off triangles, the chain ladder (CL) method and the Bornhuetter-Ferguson algorithm (Bornhuetter and Ferguson, 1972); or stochastic extensions of said algorithms.

We focus instead on *individual claim reserve modelling*, or *micro-level reserving*, an alternative method of reserving. Individual claim reserve modelling predicts portfolio reserves from aggregating estimates per incident. There is not yet a consensus that individual claim reserving is more or less accurate than aggregate modelling. Still, the hypothesized benefits of micro-level reserving are: greater insight into exposure profiles within a portfolio; more signal (i.e. relevant covariates) should produce more accurate models; and the ability to adapt to trends that can be captured by covariates (Blier-Wong et al., 2021; Delong and Wüthrich, 2020; Lopez, Milhaud, and Thérond, 2019).

There is literature investigating the use of older machine learning (ML) algorithms such as CART (Breiman et al., 1984) and generalized linear models for individual claim reserving (De Felice and Moriconi, 2019; Lopez, Milhaud, and Thérond, 2019; Taylor, McGuire, and Sullivan, 2008; Wüthrich, 2018). Newer ML methods, such as neural networks (DeLong, Lindholm, and Wüthrich, 2022; Delong and Wüthrich, 2020; Kuo, 2020) and GBDTs (Duval and Pigeon, 2019) have also had some limited research. These works analyzing micro-level reserving strategies broadly conclude that their respective models are either on par or better than an aggregate CL method, validating micro-level reserving in principle. However, there are only a few such works; their insurance fields vary; they use small sets of covariates and some use simulated data. This makes it difficult to know whether the results are relevant to car insurance micro-level reserving. Furthermore, of considerable practical importance is that missing data is endemic to real insurance data (Fauzan and Murfi, 2018; Hanafy and Ming, 2021) and none of these works give any special focus to missing data. Finally, these works often report benchmarks against the CL method instead of overall accuracy which complicates the interpretation of results as disagreement with CL could be the consequence of more accurate modelling. To our knowledge, also noted by the survey of Blier-Wong et al., 2021, none compare modern ML methods directly to each other on real data. The lack of direct comparison means no conclusion can be drawn

about the relative performance of newer ML methods for claim reserving.

Ultimately, both research in insurance and ML more broadly paints a blurry picture on the relative merits of GBDTs and DL for micro-level reserving using tabular data. Furthermore, treatment or influence of missing values on accuracy is not investigated when comparing the methods. Although missing value handling has been shown to be important in other fields (Herring, Ibrahim, and Lipsitz, 2004) and as such could be important to reserving. This leaves reserving actuaries dealing with tabular data unclear on whether it is worth the investment to investigate and deploy these more modern ML algorithms, nor the impact of missing data for said algorithms.

The most relevant work to ours, comparing DL to GBDTs in insurance, is McDonnell et al., 2023. They compare the DL architecture TabNet (Arik and Pfister, 2021) to the GBDT implementation XGBoost. They model discretized *claim severity* classification on a dataset with hundreds of thousands of claims. They find TabNet to be comparable to XGBoost, with marginally better F1 score. Although this is modelling claim *severity*, not claim *reserving*, we note that claim severity and claim reserving both model costs conditional on an accident occurring. However, claim severity is estimated *before* the accident occurs and claim reserve *after*. From the perspective of regression, the only difference between micro-level reserving and claim severity modelling is the number of covariates - as more is known after an accident occurs. In the work of McDonnell et al., 2023, although TabNet performs comparatively well to XGBoost, the models were evaluated on synthetic data generated using a neural network (So, Boucher, and Valdez, 2021), hence potentially biasing performance towards DL as the model class was more likely to be correct. Furthermore, the casting of the regression problem into discretized classification and lack of missing data makes the findings less interpretable.

3.2.1 Data description

The tabular data we model in Section 3.4 consists of many hundreds of thousands of insurance claim feature vectors as rows, with hundreds of features as columns. This dataset has never been previously studied in any published work. The data is a combination of information available

at policy issue (e.g. make and model of the car) and information available just after the time of claim reporting (e.g. accident date). The settlement value (SV) variable gives how much the insurer paid overall to settle a claim; inclusive of vehicle, personal and property damage. We aim to accurately predict SV for each claim to build a micro-level reserve, as described in Section 3.2. As we use supervised ML methods we only consider closed claims, i.e. there exists a SV to be used as a label.

Commercial confidentiality prevents us from giving a detailed description of the data. However, we present the missing data properties in this section. Other data characteristics and their implications for modelling and data processing are presented in Appendix A.1.1; this includes time varying properties and handling of high cardinality categoricals, such as postcode information.

Missing data

The dataset we study has extensive missing values. Over 50% of features contain missing values, therefore ignoring all features with missing values would drop the number of features by over half. This could drop highly informative features, e.g. details of additional drivers on a policy, which are missing in the majority of claims.

Furthermore, due to the interaction of missing values in multiple features there is *no complete feature vectors*, i.e. every row has at least one missing value. Therefore if we directly apply a strategy such as complete-case analysis (Little and Rubin, 2019, p. 47), where any row with missing values is dropped, the whole dataset would be dropped. Instead, we can first drop features that are missing in more than a certain proportion of cases and then run a complete-case analysis. This latter approach is also used to deal with missing values by Grinsztajn, Oyallon, and Varoquaux, 2022, one of the broad comparative studies mentioned in Section 3.1. We explore this method, along with alternative imputation approaches, calling this missing value handling strategy **Drop** in Section 3.4.

Beyond the extent of missing values, the data presents a dependence of the response, SV, on the missing value structure. This can be shown by a large shift in the mean and standard

deviation of the SV when using **Drop** at various missing value proportion thresholds. Smaller proportions of missing values in a feature vector are associated with substantially higher SV. This suggests that the data is not-MCAR (Little and Rubin, 2019, pages 13-23). Therefore, fitting a model under a **Drop** strategy will result in biased predictions, above and beyond any bias introduced by the model or training algorithm. This bias also means the accuracy a model fit on **Drop** is not comparable to that of a model fit on imputed data.

To summarize, missing values represent a large portion of our insurance dataset and are *not* MCAR as the SV varies substantially conditional on the missing value structure. This highlights the importance of investigating and choosing appropriate missing value handling strategies.

3.3 Models

In this section we briefly give background on the models used: i) Catboost, a GBDT model; ii) two ResNets, a general purpose feed forward DL model: ours and that of Gorishniy et al., 2021; and iii) TabNet, a DL architecture specifically designed to accommodate tabular data. Within this section we do not aim to provide comprehensive details. Instead we aim to describe methods in sufficient detail to follow the hyperparameters tuned in Section 3.4.1.

3.3.1 Catboost model

Catboost (Prokhorenkova et al., 2018) is a GBDT (Friedman, 2001) with a special procedure for categorical encoding and gradient estimation designed to reduce overfitting. Detailed background on the algorithm was given in Chapter 2. Note that the Catboost implementation is complex and has many configurable options. For removal of ambiguity, as the default behavior can vary depending on the execution hardware, we used defaults for running on a CPU opposed to a GPU.

3.3.2 ResNet MLP

As discussed in Chapter 2, a ResNet is a feed-forward neural network with additive *residual* connections that skip layers. Without additional knowledge about the underlying structure of the data, an MLP is a simple general purpose DL architecture; and the residual connections make training more stable (C. M. Bishop and H. Bishop, 2024, page 275).

We implement a ResNet MLP by using residual connections across building blocks, along with residual connections to the output. We use a building block layout of BatchNorm, ReLU and dense layers as in He et al., 2016b. We add dropout following the example of Gorishniy et al., 2021 – which proposed the best performing DL model (a ResNet) from McElfresh et al., 2023. For the sake of clarity, our ResNet MLP is *not* identical in architecture to Gorishniy et al., 2021, and by extension McElfresh et al., 2023. To contextualize our findings we also present results using the ResNet architecture from Gorishniy et al., 2021, referring to it as ‘RTDL ResNet’. There are a few key differences between Our ResNet MLP and the RTDL ResNet MLP:

- Our ResNet MLP HPO tunes more HPs; McElfresh et al., 2023 only tunes learning rate whereas our ResNet uses an HPO grid containing learning rate, weight decay, regularization coefficient, learning rate schedule and optimizer.
- Our ResNet MLP does not attempt to embed categorical features, as target mean encoding was used to convert all categories into scalar representations.
- Our ResNet MLP has residual connections from the output of each block to the overall model output; whereas in McElfresh et al., 2023 residual connections only skip blocks.
- We use three repeating blocks prior to the regression head layer; McElfresh et al., 2023 use two.
- Our ResNet MLP does not vary dimensionality of the hidden state - we keep constant dimensionality of 256, in comparison to McElfresh et al., 2023 who step down to 128 and back up to 256 throughout the network. Ultimately, this and the above point result in our model having approximately 2.25x the parameters of McElfresh et al., 2023.

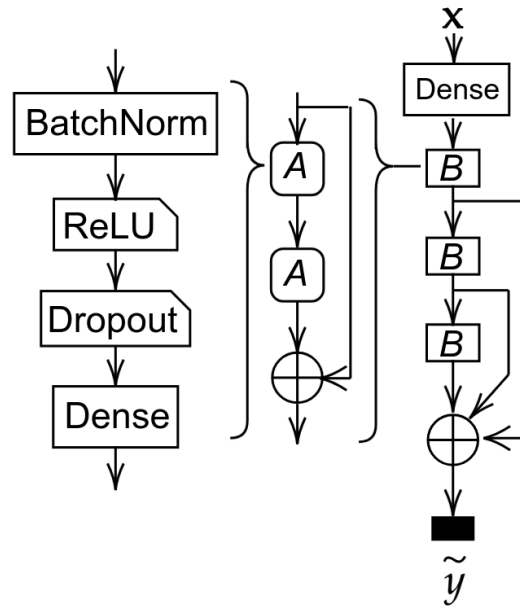


Figure 3.1: ResNet MLP model architecture. The architecture consists of a number of blocks, B . The skip connections in the diagram indicate that the outputs from each block B are all summed together and passed through a final Dense layer to produce the scalar output $\tilde{y}$. Each layer B consists of three sub-blocks denoted as A in the diagram. Each sub-block A consists of feed forward BatchNorm, ReLU, dropout and dense layers.

Figure 3.1 shows the layout of our ResNet MLP layers on the far left, with their combination into a sub-block denoted by A . The ResNet sub-block, A , is repeated in a residual pattern to form a high level block B – shown on the right of Figure 3.1. This higher level block B is in turn composed using residual connections into an overall model. Each block has independently trainable parameters. In the context of deep learning, choosing the architecture size (such as number of blocks, size of dense layers in units etc.) is a part of the broader problem of hyperparameter tuning. Our approach involved choosing the number of B blocks to vary depth; and choosing the number of units used in every dense layer to vary the width of the network. This tuning is further described in Section 3.4.1. We use a grid search, described in Section 3.4.1, to select the optimizer used, the initial learning rate of the optimizer, learning rate schedule, dense layer weight regularization strategy and regularization intensity.

3.3.3 TabNet model

TabNet (Arik and Pfister, 2021) is a DL architecture specifically designed for tabular data. TabNet works by learning a *step* that multiplies a subset of features by zero, conditional on the input. Then TabNet uses DL layers on the remaining non-zero parts to produce an intermediate *decision* vector. The architecture sequentially applies multiple steps. Each step can determine a different subset of features to set to zero – so a feature that is set to zero for one step does not need to be zero for the following steps. In fact, the hyperparameters described below control how many distinct features can be selected and their potential for reuse across steps. As each step can select different subsets, each step can produce a different decision vector. Finally, the decisions from all steps are combined through a DL layer into a prediction.

TabNet is a complex architecture for which we defer the detailed description to the original paper (Arik and Pfister, 2021). However, there are some key hyperparameters which we are required to tune.

The number of steps, S , determines the number of different feature subsets that are modelled to produce a prediction. With $S = 1$ only a single subset of the features is used, with more steps resulting in more feature subsets. Intuitively, more steps increases the overall number of features used, but also increases the depth of the network and destabilizes training.

The so-called ‘relaxation parameter’, $\gamma \geq 1$, is designed to encourage different feature subsets to be selected at each step. When $\gamma = 1$ TabNet has the special property of being able to prevent reuse of features between steps. As γ increases, TabNet is more able to reuse features between steps.

The sparsity regularization coefficient, $\lambda \geq 0$, is used to encourage more input features to be zeroed out in each step. As λ increases the network can zero out more features, even if it decreases training accuracy.

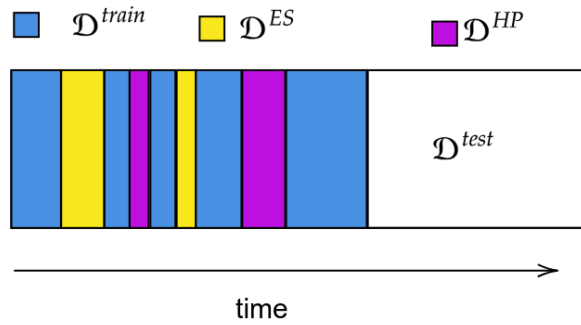


Figure 3.2: Dataset partitioning strategy for both hyperparameter tuning and final evaluation, where $\mathcal{D}^{ES}$ and $\mathcal{D}^{train}$ are shuffled per replication of a given experiment. $\mathcal{D}^{ES}$ is a split of data used for early stopping. $\mathcal{D}^{HP}$ is a split of data used for evaluation of hyperparameters. $\mathcal{D}^{HP}$ is sampled randomly in time, $\mathcal{D}^{test}$ is exclusively future data.

3.4 Experimental method

To investigate the impact of preprocessing schemes for handling missing values, we first tuned the hyperparameters of each model. Preliminary analysis, described in Appendix A.1.4, suggested the best hyperparameters did not vary with preprocessing scheme. Therefore, the hyperparameter tuning process was done independently of later missing value investigation.

To prevent data leakage, the chronologically last 15% of the data was set aside into a test set, $\mathcal{D}^{test}$, shown in Figure 3.2. This $\mathcal{D}^{test}$ was always withheld from training or validation procedures and only used to report the metrics presented in Section 3.5.

Early stopping, described in Chapter 2, was applied to improve training speed and prevent overfitting in both hyperparameter tuning and final model training. As discussed in Chapter 2, early stopping is a form of regularization where out-of-sample model performance is evaluated at regular intervals on a dataset withheld from training. When the performance on the withheld dataset decreases, the training algorithm is terminated. Preliminary analysis confirmed there was no decrease in accuracy from using early stopping.

Section 3.4.1 describes the tuning of hyperparameters discussed in Section 3.3. Section 3.4.2 describes the experimental setup used to investigate the impact of missing value handling and categorical encoding. Practical commentary on the training speeds of the algorithms can be found in Appendix A.1.2.

3.4.1 Hyperparameter tuning

Hyperparameter tuning on our ResNet and TabNet consisted of a grid search optimizing for accuracy. For each modelling strategy a hyperparameter (HP) grid was subjectively chosen after initial trial and error. Although grid search may be a common practice in industry, it is also well known to theoretically underperform more principled methods of HPO such as Optuna (Akiba et al., 2019). As such, we also reran analyses for our ResNet using Optuna with the same upper and lower bounds on numeric hyperparameter values. Those results can be found in Appendix A.1.3 but we note that in this instance Optuna HPO did not significantly change performance. For consistency with McElfresh et al., 2023, Optuna was used for HPO on the RTDL ResNet model.

A single data subset, $\mathcal{D}^{\text{HP}}$, was sampled once for accuracy evaluation of *all* models under any given HP configuration. For clarity, $\mathcal{D}^{\text{HP}}$ was *not* a future partition of the data, but rather randomly sampled in time.

For each node in the grid, we randomly split the remaining data (after removing $\mathcal{D}^{\text{test}}$ and $\mathcal{D}^{\text{HP}}$) into $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{ES}}$ subsets, illustrated in Figure 3.2. Training was performed solely on $\mathcal{D}^{\text{train}}$ whilst $\mathcal{D}^{\text{ES}}$ was used to trigger early stopping (ES). Once training was completed, the root mean squared error (RMSE) for a given node was evaluated on $\mathcal{D}^{\text{HP}}$. This random splitting of $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{ES}}$, and subsequent evaluation of RMSE on $\mathcal{D}^{\text{HP}}$, was independently repeated 10 times for each HP node in the grid. We note the $\mathcal{D}^{\text{HP}}$ did not change between these samples, as it was only sampled once. This repeated split sampling is called Monte Carlo cross-validation (Kuhn, Johnson, et al., 2013, p. 71-72).

The best HPs for a given model were chosen on the basis of the lowest RMSE averaged across the 10 Monte Carlo cross-validation samples. This best HP configuration for a given model was then used for the model-to-model comparison as described in Section 3.4.2.

The use of separate data subsets for early stopping, $\mathcal{D}^{\text{ES}}$, and accuracy evaluation, $\mathcal{D}^{\text{HP}}$, allowed us to remove bias associated with evaluation on data that was indirectly used for training. This is an especially rigorous process in contrast to what is often done in practice, where one

validation set would be used for both early stopping and evaluation. However, as we had sufficient data, we opted for separate subsets to minimize potential bias.

Further details of the training algorithm and hyperparameter selection for each modelling strategy follow.

Catboost hyperparameter tuning

Initial manual exploration of the hyperparameters gave no significant improvements in $\mathcal{D}^{\text{HP}}$ RMSE. The step size, η , was varied between 0.005 and 0.018. The ensemble size, T , was varied between 1000 and 10,000. As no noticeable improvement came from initial manual tuning, a complete grid search was not performed and all hyperparameters were left as defaults for evaluation of Catboost in model-to-model comparison. The only non-default choice was inclusion of early stopping which was used to speed up training, and had no noticeable effect on accuracy in the hyperparameter tuning stage.

TabNet tuning

For TabNet we follow the original paper (Arik and Pfister, 2021) in using the Adam optimizer (Kingma and Ba, 2014) and an exponential decay learning rate schedule with a fixed initial learning rate of 0.001. The sparsity regularization coefficient, λ ; number of steps, S ; and relaxation parameter γ were tuned. Following the recommendations and ablations of Arik and Pfister, 2021: λ was varied between 0.0001 and 0.01, γ was varied between 1.3 and 2 and S was varied between 3 and 10.

ResNet MLP tuning

Currently, there is no principled way to construct a deep learning architecture, for example choosing depth and width, beyond intuition; trial and error; and neural architecture search (Ren et al., 2021). As neural architecture search was prohibitively expensive from a computational

perspective we instead opted for a heuristic trial and error approach for choosing our ResNet architecture.

We began by arbitrarily choosing some expressive high level block, denoted B in Figure 3.1. We built this block from sub-blocks, denoted A in Figure 3.1. Sub-blocks A utilize a ResNet (He et al., 2016b) layout of layers as described in Section 3.3.

We then chose the depth: i.e. the number of B blocks composed together prior to a dense layer with a single output unit, with no activation, for prediction. A depth of 3 was chosen; subjectively balancing simplicity of the model with expressive power.

We then parameterized the width of a block with d , the number of units in each dense layer of the ResNet block, denoted A in Figure 3.1. We varied d between heuristically identified limits wherein the model exhibited underfitting and the capacity to overfit training data. Underfitting was identified by both the training and validation error being similar and approximately constant per training epoch. Capacity to overfit was identified by the ability for the model to keep reducing training error whilst validation error is constant or getting worse. These criteria were considered fulfilled when the validation loss was not improving whilst training loss was still decreasing after 1000 epochs.

Varying d within the bounds of under and overfitting did not noticeably impact accuracy. As such, d was chosen for a total model size of four hundred thousand parameters. This was between the number of parameters which under- or overfitted. After the architecture was heuristically selected, grid search hyperparameter optimization followed. The optimizers compared were Adam (Kingma and Ba, 2014) and RMSProp (Tieleman and G. Hinton, 2012). Both L1 and L2 regularization of kernel weights were used; coefficients for each were varied between 0.01 and 1. Exponential Decay (Murphy, 2022, p. 288) and Cosine Decay (Loshchilov and Hutter, 2017) learning rate schedules were both compared; where initial learning rate was varied between 0.001 and 0.01 for each.

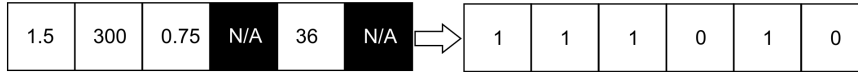


Figure 3.3: Example row undergoing **Binarize** transformation; where ‘N/A’ represents missing values. Present covariates map to 1; missing to 0.

3.4.2 Investigating different preprocessing schemes

We aimed to investigate the impact of missing value handling through the comparison of each model under different preprocessing schemes.

To enable a pairwise comparison of trained models, the data subsets for these experiments were different from those of the hyperparameter tuning in Section 3.4.1. Instead of sampling different subsets per node in a grid search *all* experiments used the same set of 20 Monte Carlo cross-validation samples of $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{ES}}$. These were sampled once prior to running any experiments, enabling pairwise model comparison. As evaluation was performed on $\mathcal{D}^{\text{test}}$, there was no need for another withheld evaluation partition, like $\mathcal{D}^{\text{HP}}$.

A given $\mathcal{D}^{\text{train}}$ subset was used for training all models with all missing value handling schemes, described below, and the corresponding $\mathcal{D}^{\text{ES}}$ was used to trigger early stopping. After the models were trained, RMSE was evaluated on $\mathcal{D}^{\text{test}}$ for each of the 20 $\mathcal{D}^{\text{train}}$ and $\mathcal{D}^{\text{ES}}$ samples. The mean and standard error of the RMSE across the 20 Monte Carlo cross-validation samples for each model and preprocessing scheme are reported in Table 3.1. These results are discussed in Section 3.5.

Missing value handling

Four missing value handling strategies were investigated. Two of the methods, **Drop** and **Binarize**, were used in an attempt to disentangle the capability of a model to successfully extract signal from observed covariates and from missing value structure. The other two, **LT Min Impute** and **Mean Impute**, were used to investigate model capability to jointly use covariates and missing value structure through imputation. The missing value handling strategies are summarized:

- **Drop:** Rows with any missing values were dropped after first removing features with more than 30% missing values – as described in Section 3.2.1. From this scheme we aimed to isolate the ability of each modelling strategy to extract signal from observed covariates. This was the approach used in Grinsztajn, Oyallon, and Varoquaux, 2022.
- **Binarize:** The whole dataset was converted into a binary representation of whether the covariate was observed or not, see Figure 3.3. This retained only the signal inherent in the missing value structure. With this scheme we isolate the ability of model strategies to regress against missing value structure.
- **LT Min Impute:** Missing values were imputed using less than the minimum of the numeric feature. This is the default missing imputation strategy of Catboost. The use of this strategy allows the weak learner trees, G_t in Section 3.3, to separate the missing values completely from observed values. However, this also means that should the tree split on an observed value it will include all missing values on the lesser side of the split.
- **Mean Impute:** Missing values were imputed using the mean of the corresponding present numeric feature. This was the approach used in McElfresh et al., 2023.

We also present results using *both* the **LT Min Impute** and **Binarize** strategies simultaneously in Appendix A.1.6 – finding there is no noticeable benefit to combining the approaches over **LT Min Impute** for our dataset.

Categorical encoding

Categorical encoding was performed using target mean encoding for the majority of the experiments, this compared the modelling strategies fairly by holding potential confounders in categorical encoding constant.

In the interests of identifying the best performance, we also evaluated the original Catboost categorical encoding with the Catboost GBDT algorithm, described in Chapter 2. We report the model names in Table 3.1 with the suffix(CE) indicating that Catboost encoding was used; otherwise mean encoding was used.

3.4.3 Effects of imputation scheme on other datasets

To study the broader relevance of results obtained from our insurance dataset, further analyses were performed on two other datasets under **Drop**, **Mean Impute** and **LT Min Impute**. The leading two GBDTs and leading two DL algorithms from McElfresh et al., 2023 were compared on two datasets using the TabZilla framework. The datasets did not contain missing values; so MCAR missing values were simulated by removal. As MCAR data are not the focus of this case study we present the results in Appendix A.1.5. Overall, these preliminary analyses showed less impact from missing value handling than our not-MCAR application but still demonstrate the relevance of our work as the ordering of accuracy varied between missing value handling schemes. The smaller difference between schemes is potentially due to the artificial MCAR nature of the injected missing values, meaning no signal needs to, or can, be extracted from the missing value structure – nor is there any relation between missing values and the response that is important to extract.

3.5 Results and discussion

The results for various preprocessing strategies are presented in Table 3.1. As a benchmark, we include a **Mean prediction** row which shows the performance that is obtained by using the mean SV of a given training dataset as a constant prediction, using no feature information. Note the performance evaluated with the **Drop** preprocessing scheme is not directly comparable to the performance for other preprocessing schemes, i.e. **Drop** cannot be compared with other strategies across rows in Table 3.1. This is because the missingness mechanism is not-MCAR and causes a distributional shift within both the training and evaluation data when dropping rows. However, due to consistency of the training datasets, these results are comparable within modelling strategies, i.e. one can compare down columns of Table 3.1.

The TabNet row of Table 3.1 shows that TabNet performed substantially worse than both Catboost and the ResNet MLP. Notably, TabNet performs on par with the **Mean prediction** benchmark. This indicates that TabNet is either not suitable for micro-level reserving with

Table 3.1: $\mathcal{D}^{\text{test}}$ RMSE mean ($\pm$) standard error, se , over cross validation partitions. **Lower RMSE is better**. RMSE is to the nearest integer, se is to 2 significant figures. (CE) indicates Catboost encoding was used for categorical encoding. $\dagger$ replicates the HPO, architecture and imputation scheme of McElfresh et al., 2023. $\ddagger$ corresponds to an Optuna HPO scheme.

Model	Drop		LT Min Impute		Mean Impute		Binarize	
Catboost (CE)	1969	(± 10)	1452	(± 6.3)	1519	(± 17)	2302	(± 0.18)
Catboost	2060	(± 1.7)	1574	(± 2.1)	1524	(± 5.2)	2268	(± 0.28)
Our ResNet	2046	(± 1.2)	1872	(± 4.5)	7855	(± 1600)	2288	(± 1.8)
RTDL ResNet $\ddagger$	2439	(± 24)	2007	(± 20)	2387	(± 270) $\dagger$	2282	(± 2.4)
TabNet	2702	(± 8.1)	2748	(± 5.3)	3081	(± 180)	2754	(± 5.1)
Mean prediction	2813	(± 0.34)	2764	(± 0.020)	2764	(± 0.020)	2764	(± 0.020)

this dataset or, at best, that TabNet is difficult to train. This runs counter to published work on the use of TabNet for claim severity modelling (McDonnell et al., 2023), but in line with surveys of DL for structured data (Grinsztajn, Oyallon, and Varoquaux, 2022; McElfresh et al., 2023) where TabNet performed poorly. As mentioned in Section 3.1, this could be explained by the fact that the results of McDonnell et al., 2023 may be biased because the modelled data is itself generated from a neural network.

The relative accuracy of the RTDL ResNet MLP row of Table 3.1 agrees with McElfresh et al., 2023 in outperforming TabNet and underperforming CatBoost. Although for **Drop**, **LT Min Impute** and **Binarize** the RTDL ResNet has lower accuracy than our ResNet it is interesting to note that under **Mean Impute** the RTDL ResNet performs substantially better than any other DL algorithm that also used **Mean Impute**. This highlights the sensitivity of previously studied algorithms to imputation schemes.

However, we note that the performance of **Mean Impute** is still generally worse than that of **LT Min Impute** across most models. **Mean Impute** demonstrates moderately high standard error for the RTDL ResNet and high mean RMSE and standard error for our ResNet and TabNet. The exceptionally high RMSE for our ResNet appeared to be due to the rare prediction of low value claims many orders of magnitude larger than they were. It is unclear why this would be the case specifically for **Mean Impute** and our ResNet. We hypothesize this could be due to **Mean Impute** making it difficult to distinguish meaningfully missing data, where absence of data is not due to random lack of records but from the reality of the data generating process, e.g. no additional drivers on the policy. Again this highlights the sensitivity of algorithms to

missing value (MV) handling.

The **Drop** and **Binarize** columns of Table 3.1 show that both the covariates and the missing value structure are important in the performance of both Catboost and ResNet, respectively offering approximately a 27% and 18% improvement over a **Mean prediction** benchmark. Interestingly, neither our ResNet nor Catboost is practically more accurate than the other when trained on either covariates (**Drop**) or missing value structure (**Binarize**) alone. This similarity of performance on **Drop** suggests studies comparing GBDTs and DL, mentioned in Section 3.1, do not have conclusions that are necessarily transferable to the context of claim reserve modelling. Those studies found a performance edge for GBDTs, notably either ignoring or in the absence of missing data. Whereas we find, for this micro-level reserving dataset, Catboost and ResNet are practically the same in terms of accuracy with missing data ignored, due to **Drop**.

However, the **LT Min Impute** and **Mean Impute** columns of Table 3.1 shows that Catboost with **LT Min Impute** both achieves the best overall performance and has a better accuracy than both our ResNet and the RTDL ResNet under either imputation scheme. We note that this better accuracy is between well-tuned ResNets and a default Catboost, underscoring the robustness of Catboost. The improved accuracy may be because Catboost better leverages both covariate signal and missing value structure, at least under a **LT Min Impute** or **Mean Impute** strategy. The strongest performance being achieved under **LT Min Impute** could also be due to the fact that minimum imputation intuitively lends itself to the partitioning strategies of a GBDT algorithm. It is unclear how minimum imputation would interact with ResNets, nor DL more broadly. This discrepancy in performance suggests an indicator of when Catboost may outperform ResNet for micro-level reserving: when there is a high proportion of missing values. This indicator is particularly relevant as missing values are pervasive in real world datasets, and are currently not studied in their role in micro-level reserving nor their contribution to GBDT and DL performance.

These results do not rule out the potential for ResNets to perform comparably or even better than Catboost given a suitable imputation strategy. However, there does not appear to be a

scientific consensus on simple and robust neural network imputation strategies. Some generative imputation approaches exist (Yoon, Jordon, and Schaar, 2018) but these involve considerable extra complication and have not shown much industry adoption.

3.6 Conclusion

In this paper we investigated four different imputation schemes, described in Section 3.4.2, for handling missing values using gradient boosted decision tree and deep learning models. We investigated the impact of these imputation schemes using a large, real world insurance dataset with *not* MCAR missing values. Under two of the missing value handling schemes there is no noticeable difference between Catboost and the ResNets. However, using imputation, Catboost substantially outperforms the other models. This result highlights the importance of missing value handling in claim reserving.

More broadly, this result adds a case study to the body of evidence that gradient boosted decision trees can outperform deep learning for tabular data, but emphasizes the importance of missing values in drawing this conclusion. The significant impact of missing value handling on accuracy also suggests that when analyzing datasets with missing values, extra care should be taken choosing the missing value handling method and not to just focus on model selection. Furthermore, our results suggest research comparing gradient boosted decision trees to deep learning for tabular data could benefit from including more datasets with missing values, especially missing values that are *not* MCAR. The importance of including missing value handling in comparisons was corroborated on other datasets using the Tabzilla repository (McElfresh et al., 2023) by injecting MCAR missing values, described in Appendix A.1.5. However, with MCAR missing values the effect of missing value handling was lessened.

Based on the analysis in this work we recommend exercising caution when using deep learning models for claim reserving as they require thorough tuning and their interaction with imputation schemes is not understood. Furthermore, using Catboost for claim reserving has some practical advantages, beyond potentially better accuracy with the correct missing value han-

dling. Catboost is fast, robust and easy to use off-the-shelf. In comparison, deep learning methods require more expertise to deploy successfully: requiring both architecture selection and hyperparameter tuning. Even with the expertise required to select, implement and optimize deep learning models, there is no compelling empirical or theoretical evidence that they are likely to produce better results for claim reserving.

Chapter 4

Investigating the importance of natural language pipeline steps in applications of regression

4.1 Introduction

Regression, the task of predicting numeric values from data, is an extensively studied machine learning problem. Regression algorithms typically also require the input to be numeric. However, many applications are emerging that would benefit from regression with natural language (NL). For example: in commerce to predict product prices from descriptions; in finance for stock price regression from reports; or in insurance for claim damage prediction from accident reports. This gives rise to the question of how to use natural language for regression.

This chapter investigates construction of NLP regression *pipelines*. We use ‘pipelines’ to refer to *every* step of NL regression. This includes preprocessing that modifies raw text; tokenization that maps text to tokens; featurization that maps tokens to numeric representation; and modelling that predicts numeric outputs given the numeric representation.

Our contributions include a systematic study of NLP for regression that, to the authors’ knowl-

edge, for the first time quantifies the relative importance of certain pipeline choices. Specifically, we empirically demonstrate that contextual vector embeddings outperform non-contextual vector embeddings for NLP regression on the datasets studied. We also provide quantitative evidence that: i) finetuning is required for BERT featurizers to achieve better accuracy than BoW featurizers, highlighting the implications for resource constrained applications, ii) that the choice of preprocessing, tokenizer and featurizer is associated with variance in accuracy comparable to that of choosing the model, and iii) that GBDT models perform worse using vector embeddings than with bag of words featurization.

Section 4.2 presents background on the NLP pipeline steps that we study. Section 4.3 presents relevant related work. Section 4.4 outlines the experimental procedure used in this study. Section 4.5 presents results and discussion.

4.2 Background

As mentioned in Chapter 2, a typical NLP pipeline comprises four steps:

- Preprocessing: maps text to text. The aim is to remove information which is irrelevant and make relevant information easier to extract. Multiple different preprocessors may be used in the same pipeline.
- Tokenization: maps text to a sequence of ‘tokens’, which are chunks of the text that will be used in the featurization algorithm.
- Featurization: maps a sequence of tokens to a numeric representation.
- Modeling: maps a numeric representation to a prediction.

We summarize the choices we study for each of these steps in Figure 4.1. In Section 4.2.1 we give background on the preprocessors used in this study. In Section 4.2.2 we briefly introduce the tokenizers used in this study. A more detailed introduction into the concept of tokenization

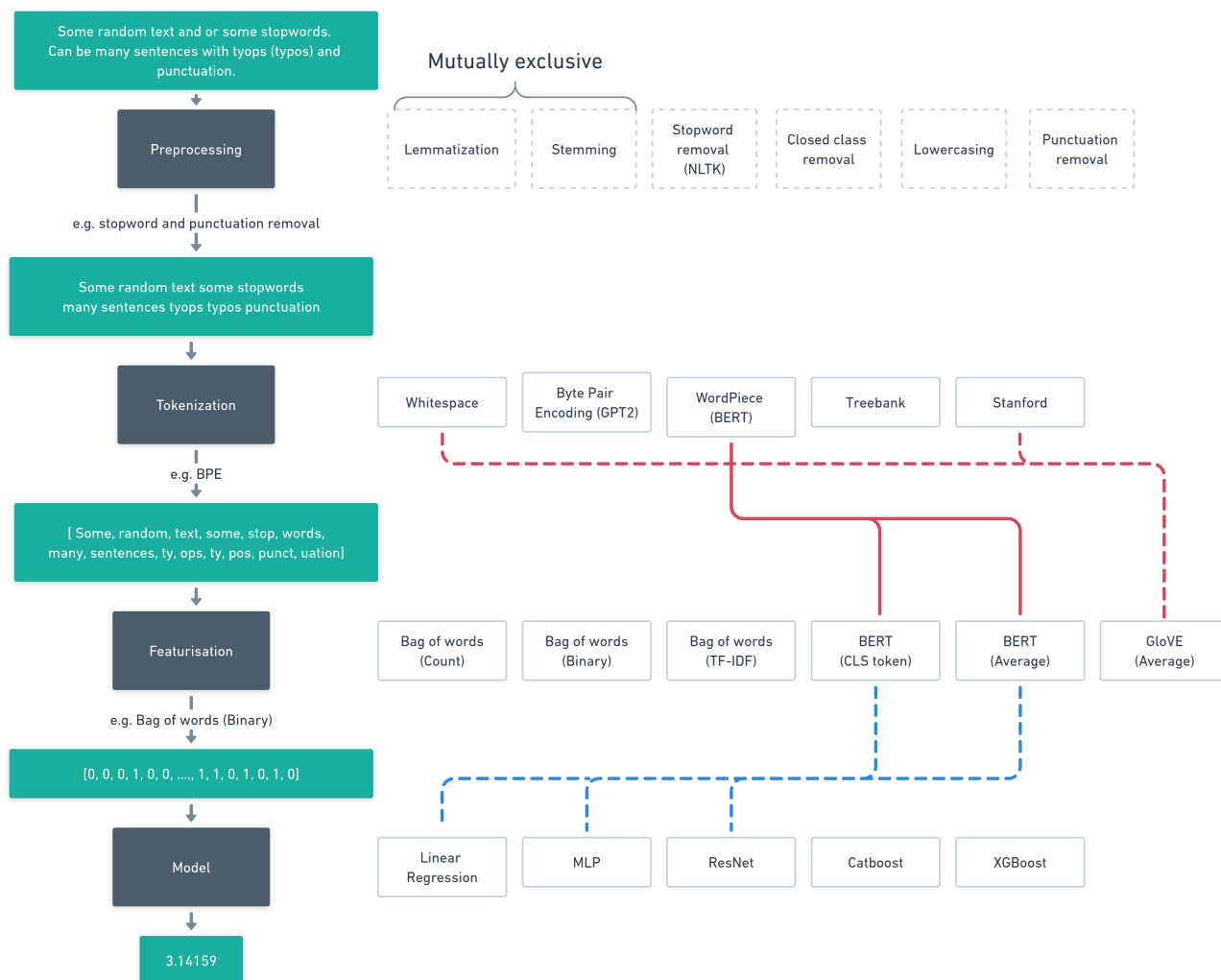


Figure 4.1: An NL regression pipeline with choices for each step. We show the flow of information for an example sentence beginning ‘*Some random text...*’ in green boxes. Dashed boxes represent choices that can be combined, solid boxes are mutually exclusive. Red lines represent that the featurizer must be used with connected tokenizers. Note, a tokenizer with red line connections may be still used with other featurizers. Blue lines represent that the given modelling choice can be finetuned with the given featurizer.

was given in Chapter 2, Section 2.1. In Section 4.2.3 we give background on the featurizers used in this study. In Section 4.2.4 we outline the modeling approaches included in this study.

4.2.1 Preprocessing

In this section we summarize the preprocessors that we study.

Stemming and Lemmatization are both methods to map different forms of the same concept to one form. For example, a document may contain the words ‘ran’ and ‘running’, or ‘room’

and ‘rooms’ and instead of treating these words as distinct concepts we may wish to map them to the same concept to enable fitting to the concept instead of fitting to (potentially sparse) forms. The difference between stemming and lemmatization is in how this is done.

Stemming is a heuristic process in which sections of words are deleted or altered according to a rule set, for example a rule might be the removal of ‘s’ at the end of any word. This would map most plurals to their singular form but would map singular words ending in ‘s’ to a non-word, e.g. ‘miss’ to ‘mis’. In this work we include the Porter stemmer (Porter, 2001).

Lemmatization is a more complex process that attempts to map words to their *base form* (e.g. ‘am’, ‘are’ and ‘is’ are all mapped to ‘be’). We include the spaCy lemmatizer (Honnibal et al., 2020) in our study. The lemmatizer first predicts part of speech (e.g. verb, adjective etc.) for each word, then transforms suffixes and finally uses a lookup table in order to predict a word’s lemma.

As both of these preprocessors aim to achieve the same goal, we made the choice of them mutually exclusive.

Stopword removal is the process of removing words that are common among all text analyzed and thus are hypothesized to have little predictive relevance. We include two approaches for stopword removal in our study that are not mutually exclusive. The first removes words that are in a predefined list as provided by the NLTK package (Bird, 2006). The second removes all words that are in a ‘closed class’ of connecting English words such as prepositions, see Universal Dependencies, 2024 for a complete list of closed class parts of speech. For clarity, this only leaves words that are predicted to be verbs, adverbs, nouns, adjectives or numerals.

Punctuation removal has similar logic to stopword removal. The presence of some common punctuation in a sentence is hypothesized to have little predictive relevance to the regression and so is removed.

Lowercasing maps all characters to their respective lowercase forms. For English this essentially ensures words that appear at the beginning of a sentence map to the same representation as in the middle of a sentence; as well as working around capitalization inconsistency among

input texts. However, this could conflate proper names and nouns, e.g. ‘President Bush’ would become ‘president bush’ which when tokenized could map the former head of state to the same token as vegetation.

4.2.2 Tokenization

In this section we summarize the tokenizers we select for this chapter.

Whitespace tokenization defines tokens as anything separated by whitespace characters. For space-delimited languages, such as English, this is one of the simplest forms of tokenization.

Treebank tokenization splits on whitespace but also uses a regex to handle some edge cases for contractions (e.g. ‘don’t’ is tokenized to ‘do’ and ‘n’t’) and punctuation.

The **Stanford** tokenizer uses a set of custom rules crafted by NLP researchers (Manning et al., 2014).

Byte-pair encoding (BPE) tokenization (Sennrich, Haddow, and Birch, 2016) is an algorithm that learns a tokenizer from text. Two key distinctions with the Whitespace, Treebank and Stanford tokenizers are i) BPE can produce sub-word tokens, e.g. ‘bedroom’ could be tokenized as [‘bed’, ‘room’] and ii) BPE is typically pretrained from a separate body of text. We specifically include the GPT2 BPE tokenizer, that was trained as part of Radford, Wu, et al., 2019.

WordPiece tokenization (Devlin et al., 2019) is another form of trained sub-word tokenizer, with a very similar construction logic to BPE but with a different objective function during pretraining.

4.2.3 Featurization

Given a sequence of tokens representing the input, featurizers map the sequence to a single numeric representation.

As mentioned in Chapter 2, a **BoW** featurizer is defined by a set of tokens, $v_i \in \{v_1, v_2, \dots, v_V\}$, called a vocabulary. The BoW representation of some tokenized text is a vector, $f \in \mathbb{R}^V$, where f_i is the count of token v_i in the tokenized text. For example, given a vocabulary $[\mathbf{a}, \mathbf{b}, \mathbf{c}]$ and a tokenized input $[\mathbf{b}, \mathbf{b}, \mathbf{a}, \mathbf{a}, \mathbf{b}, \mathbf{d}]$ the BoW representation could be $f = [2, 3, 0]^T$. Tokens that are not in the vocabulary are discarded.

Due to its construction, BoW is order invariant, as can be seen in the prior example: swapping the positions of any **a** and **b** would not change the featurization. One way of partially mitigating this drawback is to include some *N-grams* in the vocabulary. *N-grams* are token sequences of length N and instead of matching a token to increase the count of the vocabulary index; the whole *N-gram* must be matched. Extending the previous example, if the vocabulary was to include a 2-gram, $[\mathbf{a}, \mathbf{b}, \mathbf{c}, [\mathbf{b}, \mathbf{a}]]$, the representation would be $f = [2, 3, 0, 1]^T$. In this work we choose the vocabulary to include the 50,000 most frequent tokens or 2-grams in the training set.

We also include a *modified* BoW implementation where f_i is a binary indicator on the presence of v_i in the tokenized text; the first example would thus be featurized as $f = [1, 1, 0]^T$. We distinguish the count and the indicator implementations through the featurizer names **bow-count** and **bow-binary** respectively. Furthermore, we include an extension of **bow-count** called **tf-idf** (Salton, 1991). This extension normalizes counts in a manner that down-weights tokens that are common among many entries in the training examples.

Alternatively, as also discussed in Chapter 2, **vector embeddings** are mappings between *single* tokens and *fixed size* vectors. This is compared to mapping *sequences* of tokens to *vocabulary sized* vectors as in BoW. These vectors are learned from very large datasets to perform well for a generic objective (such as predicting words that are intentionally removed) and then can be re-purposed for other NLP tasks (Devlin et al., 2019; Mikolov et al., 2013; Pennington, Socher, and Manning, 2014). As vector embeddings are learned mappings for *specific* tokens, trained vector embedding featurizers must be used with *specific* tokenizers. Otherwise, the tokens may be not present in, or semantically different to, the featurizer’s vocabulary. This is why not all paths through the Figure 4.1 are possible.

Vector token embeddings can be divided into two classes: contextual and non-contextual. Non-contextual word embeddings are purely a lookup of a token in a mapping between tokens and vector. We include the GloVe featurizer (Pennington, Socher, and Manning, 2014) as a non-contextual word embedding. We also include the BERT (Devlin et al., 2019) featurizer as a popular Transformer based contextual word embedding in our study.

To represent a variable length sequence of vectors in a fixed size for downstream modelling we use an elementwise mean on the embeddings. For BERT contextual embeddings we also have the option to use the CLS token for sequence embedding, instead of mean pooling. To differentiate the two methods of featurization using BERT, we call them `bert-mean` and `bert-cls` respectively.

4.2.4 Modeling

Once the rest of the pipeline is chosen, we apply a machine learning model to the obtained numeric representation, solving a standard regression problem. We include Catboost (Prokhorenkova et al., 2018), XGBoost (Chen and Guestrin, 2016), an MLP, a residual network (ResNet) and a linear model trained through gradient descent as choices for the modelling step of the pipeline. Background on all of these models, bar XGBoost, are given in Section 2.2, 2.3 and 2.4. As discussed in Section 2.3, generally GBDTs, including Catboost, approximate a first order functional gradient descent. XGBoost is a GBDT algorithm that instead approximates a second order functional gradient descent, along with computational optimizations, and is described well in the original work by Chen and Guestrin, 2016.

We note a nuance of the modelling: if the model can be trained through gradient descent we call this model ‘differentiable’ (i.e. MLP, ResNet and the linear model). If the model is differentiable **and** the featurizer is also pretrained through gradient descent (i.e. a BERT model) then we can jointly train both the model and the featurizer. This is done in a process we refer to as **supervised finetuning**. Firstly only the model is trained, with the featurizer frozen, until early stopping terminates the training. Then the model and featurizer are jointly trained with a very small learning rate until early stopping is triggered.

4.2.5 ‘Traditional’ and ‘Modern’ approaches

We note that historically, two relatively independent approaches have been developed for NLP, centered around the choice of featurizer. First, a more ‘traditional’ framework uses featurizers that originate from a BoW, or variations thereof, with other components being variable. Alternatively, ‘modern’ deep learning approaches use featurizers that generate vector embeddings (which induce fixed tokenizer choice), minimal preprocessing, and models that allow for finetuning. Few works have studied mixing between these two approaches for NL regression.

4.3 Related work

Although there is much literature on the task of NL *classification* (Minaee et al., 2021), there is much less attention on NL *regression*. Prior work has mostly compared a small number of preselected pipelines or even a single pipeline with one step varied.

Early work by Joshi et al., 2010 investigated linear regression for movie box office gross from reviews. They used metadata and BoW featurization, but did not investigate the impact of altering pipeline steps. This work was followed up by Bitvai and Cohn, 2015 which compared the results of Joshi et al., 2010 to a new pipeline consisting of convolutional neural networks on non-contextual vector embeddings. The work of Bitvai and Cohn, 2015, in principle, validated a deep learning approach to NL regression by achieving substantially better accuracy metrics than Joshi et al., 2010 with a noticeable improvement due to the inclusion of text.

More recently, studies of NL regression focused mainly on topics related to finance, where several groups attempted to predict stock volatility using company reports. Dereci and Saraclar, 2019 investigated a convolutional neural network model with a non-contextual word embedding featurizer and found an improvement over support vector regression with BoW (Tsai and Wang, 2014). H. Y. Yeh, Y. C. Yeh, and Shen, 2020 varied both the featurizer and modelling step. They compared i) `tf-idf`, ii) pretrained non-contextual vector embeddings and iii) custom non-contextual vector embeddings using both linear regression and support vector regression

models. It was found that using a vector embedding representation achieved better accuracy than a `tf-idf` approach; with a minor edge to custom vector embeddings. However, the preprocessing and tokenization of H. Y. Yeh, Y. C. Yeh, and Shen, 2020 was standardized across the experiments. Caron and Müller, 2020 studied contextual vector embedding featurization for stock volatility regression. The study compared BoW to BERT as a text featurization strategy for support vector regression modelling. The analysis concluded that using BERT for text featurization was more accurate than BoW. Interestingly, they found, for the first time, a text-only model that could achieve better accuracy than a model using historic numeric only features. However, it could be noted that this superiority was partly due to a large improvement in one specific year where all models generally achieved very poor accuracy.

Overall, no prior work has systematically varied all NLP pipeline components simultaneously, leaving many potential configurations, including hybrid approaches, underexplored. For instance, whether there is merit in using vector embeddings as features for gradient boosted decision trees. Furthermore, it leaves questions around which steps in the pipeline are important decisions, i.e. which steps have large variance in performance between good and bad choices. In this study we aim to explore the impact of pipeline construction as a whole: allowing all parts of the pipeline to vary.

4.4 Method

The core methodology centers on the generation, training and evaluation of many pipelines with varied step choices. Each pipeline was generated by randomly selecting choices for each step. After random selection the choices were checked for compatibility, i.e. the BERT featurizer must use the BERT tokenizer. Valid pipelines are summarized in Figure 4.1. Initially, a random subset of preprocessors was chosen. This includes the choice of no preprocessing. If both lemmatization and stemming were chosen, we randomly selected only one of them to keep in the subset. Then a tokenization, featurization and modelling strategy were also chosen at random. If the featurization strategy was only compatible with a specific tokenizer

we reselected the tokenizer with a random choice from the valid tokenizers, i.e. Whitespace or Stanford tokenization for GloVe embeddings and only BERT WordPiece tokenization for `bert-cls` or `bert-mean`. Finally, if the generated model and featurizer were capable of being finetuned we randomly selected whether to finetune or not.

This was done for two real world datasets: i) a dataset with 13,000 instances for predicting consumer good prices from textual descriptions, which we call `jcp` and ii) a dataset with 2,000 instances for predicting boat prices from online auction textual descriptions, which we call `obl`. Samples of the datasets are presented in Appendix A.2.2.

We generated hundreds of random valid pipelines for each dataset and trained them on a 60/20/20% train/validation/test split; recording test set RMSE for each. We reshuffled the train/validation splits 4 times in total for each random pipeline configuration, using these reshuffled datasets for Monte Carlo cross validation. Therefore, we have 4 samples of RMSE for each pipeline configuration.

For reproducibility, all experiment code, datasets and analysis code are made available on [GitHub](#). We also make the results available at the same GitHub link, enabling investigation of the results without needing to run simulations. The count of choices at each step across the generated pipelines is summarized in Appendix A.2.1.

4.5 Results and Discussion

This section discusses interesting aspects of pipeline construction revealed through our experiments. Section 4.5.1 compares contextual to non-contextual vector embeddings. Section 4.5.2 discusses the importance of finetuning for BERT featurized methods. Section 4.5.3 discusses the relative performance of BoW to BERT featurization and their implications for low resource environments. Section 4.5.4 discusses the importance of choices upstream from modelling. Section 4.5.5 discusses the performance of ‘traditional’ GBDT models with ‘modern’ vector embeddings. For clarity, when we refer to ‘upstream’ choices the most upstream choice is preprocessing and the most downstream choice is model selection.

Table 4.1: Paired Monte Carlo permutation tests for comparison of BERT and GloVe featurization strategies. ‘mean-difference’ reports the difference in average RMSE between BERT and GloVe respectively, i.e. negative mean-difference indicates BERT was on average more accurate.

dataset	mean-difference (BERT-GloVe)	p-value
obl	-32,909	<0.001
jcp	-17	0.004

4.5.1 Contextual word embeddings outperform non-contextual word embeddings

In this section we study the relative performance of contextual and non-contextual word embeddings. We selected pipelines that use either BERT (-cls and -mean) or GloVe featurization. Then these pipelines were further refined to *pairs* of pipelines, where each pair is the same except one uses BERT featurization and the other uses GloVe (e.g. use the same train/validation split, preprocessor and model). This enables a pairwise comparison of featurization approaches. For clarity, tokenizers were not the same between these pairs, as tokenizers that are compatible with BERT and GloVe are mutually exclusive.

We then performed a paired Monte Carlo permutation test on mean RMSE between these pairs of pipelines. Table 4.1 shows the mean difference and p-values. The paired permutation tests shows that BERT featurization has significantly lower RMSE than GloVe featurization for both datasets. This suggests that contextual vector embeddings are better for textual regression than non-contextual embeddings, for these datasets.

4.5.2 Finetuning is required for BERT methods to outperform bag of words

In this section, we study the importance of finetuning in comparing contextual word embeddings to BoW. We selected pipelines that either used BoW featurization with any model, or used BERT featurization *with a differentiable model*. A differentiable model being one that is capable of backpropagating loss gradients for training, i.e. linear, MLP or ResNet. The BERT pipelines

were then partitioned into those that did or did not use finetuning. Overall, this results in three types of pipeline: BERT with finetuning, BERT with differentiable models but without finetuning and BoW pipelines. Figure 4.2 and Table 4.2 show boxplots and a Monte Carlo permutation test comparing these three approaches respectively.

The ‘ft vs no-ft’ rows of Table 4.2 show that for both datasets finetuning is significantly better than not finetuning compatible pipelines. The ‘no-ft vs bow’ rows of Table 4.2 show that models without finetuning are either comparable (obl) or significantly worse than BoW (jcp). The ‘ft vs bow’ row of Table 4.2a shows that finetuning significantly outperforms BoW for the obl dataset. However, the ‘ft vs bow’ row of Table 4.2b shows that BoW and finetuning are not significantly different in performance for jcp, with relatively small mean difference and with comparable RMSE range (shown in Figure 4.2b).

Overall, BERT featurization without finetuning had comparable or worse accuracy than BoW. However, with finetuning, BERT accuracy substantially improved, being comparable to or significantly surpassing the performance of BoW. This suggests that although better than non-contextual GloVe embeddings, finetuning is necessary (but not sufficient) for contextual BERT embeddings to outperform BoW, for these datasets. It is reasonable to expect contextual embeddings to outperform non-contextual, as the importance and even semantic meaning of words can be highly contextual. However, it is perhaps surprising that non-contextual embeddings had significantly lower accuracy than BoW based methods: as BoW can be interpreted as vocabulary sized non-contextual embeddings. The relatively lower performance of non-contextual vector embeddings, and the need for contextual vector embeddings to be finetuned to match or exceed BoW, suggests that there could be a loss of information on which words are present in vector embedding approaches. This loss of information could be due to the pooling operations used to map variable length sequences to a fixed size representation.

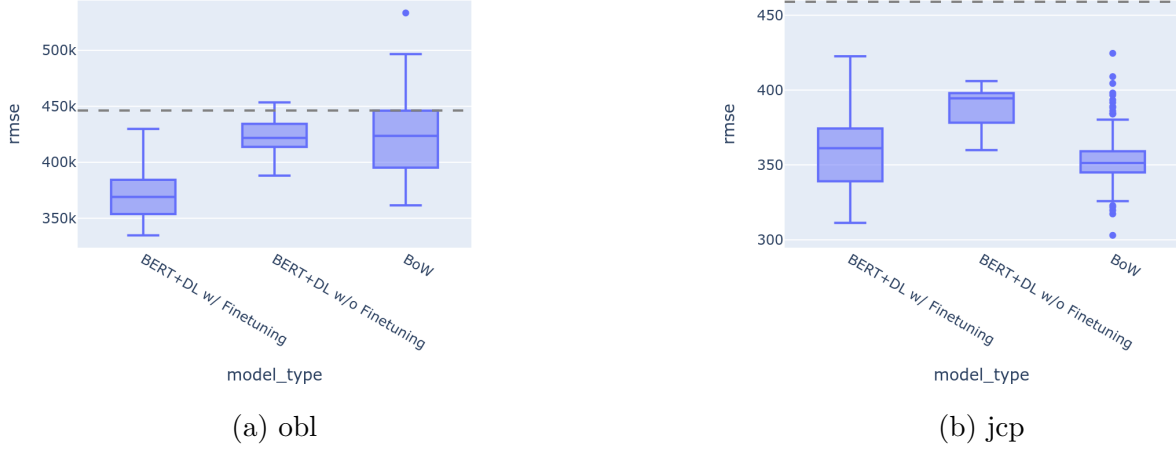


Figure 4.2: RMSE distributions across featurization types for different datasets. The dashed line is RMSE of a baseline predicting the training set mean.

Table 4.2: Monte Carlo permutation test for comparison of featurizations. P-values from 100,000 permutations. Negative mean difference indicates the first choice is better. The mean-difference is calculated from taking mean RMSE of all pipelines of the first class and subtracting the mean RMSE of all pipelines in the second. ‘ft’ is BERT with finetuning. ‘no-ft’ is BERT with differentiable model but *no finetuning*. ‘bow’ is BoW featurizer with any model.

comparison	mean-difference	p-value	comparison	mean-difference	p-value
ft vs no-ft	-52,352	<0.001	ft vs no-ft	-30	<0.001
ft vs bow	-49,932	<0.001	ft vs bow	5	0.092
no-ft vs bow	2,420	0.732	no-ft vs bow	35	<0.001

(a) obl

(b) jcp

4.5.3 Bag of words approaches are still relevant for resource constrained applications

In this section we discuss the relevance of BoW featurizers in the context of resource constraints in memory or time. Table 4.3 and 4.4 show the ten pipelines with lowest average RMSE for `obl` and `jcp` respectively, where the average is taken over Monte Carlo cross validation folds. Although we find that a finetuned BERT-featurized pipeline had the lowest average RMSE for both datasets studied, the performance of BoW models is still competitive. For `jcp` the third best pipeline uses `tf-df` (a BoW featurizer) and had mean RMSE within one standard error from the best pipeline. We also note that several of the top ten pipelines for both datasets used BoW approaches. This suggests that competitive performance may be achieved using BoW, for these datasets.

The requirement for model finetuning to reach comparable, or better, accuracy than BoW, discussed in Section 4.5.2, introduces substantial computational cost for longer text. Transformer-based featurizers, like BERT, scale quadratically with input length in memory and time due to their attention mechanisms. Furthermore, this quadratic cost is incurred for every *step* of the finetuning process - which can take multiple *epochs*. In contrast, BoW featurization scales linearly in both memory and time (assuming a fixed vocabulary size) and is trained in one go. Therefore, the competitive performance of BoW suggests that BoW is a valid choice for long text inputs or when computational resources are limited. We show the difference in mean runtime between finetuned and BoW models, along with a Monte Carlo permutation test, in Table 4.5. This table shows analyzed BoW pipelines are indeed significantly faster than finetuned pipelines for regression on the datasets studied.

4.5.4 Pipeline steps upstream from modelling have performance impact comparable to model choice

In this section we discuss the relative importance of model selection compared to choices upstream from modelling. Table 4.6 presents the effect of choices upstream from modelling on

Table 4.3: Top ten performing pipelines by RMSE on `obl` test set (sorted by mean, smaller is better). Standard error of mean denoted by ‘sem’. Models with finetuning denoted by ‘ft’. Mean and sem over 4 cross validation replicates.

preprocessor(s)	tokenizer	featurizer	model	rmse	
				mean	sem
empty	bert	bert-mean	deep-linear-ft	350,867	1,645
Le,Lo	bert	bert-mean	deep-linear-ft	351,302	6,202
empty	bert	bert-cls	deep-linear-ft	354,329	8,511
NP	bert	bert-cls	deep-linear-ft	359,783	9,632
Le,Lo,NP,NSN	bert	bert-cls	deep-linear-ft	366,544	7,608
empty	bert	bert-cls	mlp-ft	367,336	6,514
Le	bpe	tf-idf	mlp	367,446	2,355
empty	bert	bert-mean	mlp-ft	371,003	8,945
Le	bpe	tf-idf	deep-linear	371,669	2,117
Le,NSN	bert	bert-mean	resnet-ft	374,181	8,615

Le = Lemmatize, Lo = Lowercase, NP = No punctuation, NSN = No stopwords (NLTK), NSCC = No stopwords (closed class), S = Stem

Table 4.4: Top ten performing pipelines by RMSE on `jcp` test set (sorted by mean, smaller is better). Standard error of mean denoted by ‘sem’. Models with finetuning denoted by ‘ft’. Mean and sem over 4 cross validation replicates.

preprocessor(s)	tokenizer	featurizer	model	rmse	
				mean	sem
NSN	bert	bert-mean	mlp-ft	328	8
empty	bert	bert-mean	mlp-ft	330	5
	whitespace	tf-idf	mlp	334	5
NSN	whitespace	bow-count	mlp	335	5
empty	treebank	tf-idf	mlp	335	4
	bert	bert-mean	deep-linear-ft	337	6
	stanford	tf-idf	mlp	337	4
NSN,S	whitespace	glove-mean	resnet	338	2
Lo,NSN	treebank	bow-binary	mlp	338	4
Le	bpe	tf-idf	mlp	339	1

Le = Lemmatize, Lo = Lowercase, NP = No punctuation, NSN = No stopwords (NLTK), NSCC = No stopwords (closed class), S = Stem

Table 4.5: Monte Carlo permutation test for mean difference in runtime, in seconds. P-values from 100,000 permutations. Negative mean difference indicates that BoW is faster.

dataset	mean-difference (BOW - FT)	p-value
obl	-1,299	<0.001
jcp	-2,767	<0.001

Table 4.6: Effect of upstream modelling choices on RMSE. Maximal % reduction, from worst to best pipeline, in mean RMSE for different modelling strategies. Mean is taken across cross validation folds. Finetuned models are indicated by ‘ft’.

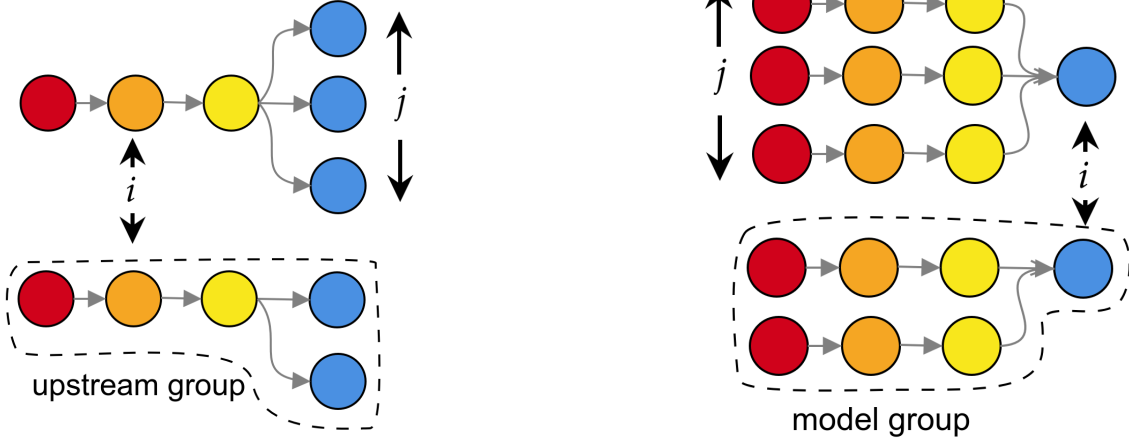
Dataset	catboost	deep-linear	deep-linear-ft	mlp	mlp-ft	resnet	resnet-ft	xgboost
obl	5	17	4	22	4	9	7	27
jcp	13	19	6	24	7	10	6	20

RMSE. This effect is measured as the % reduction in mean RMSE (where the mean is taken over cross validation folds for a given pipeline) from best to worst pipeline, conditional on model choice. For example, in the `obl` row with the Catboost column we read 5. This indicates that across all of the pipelines which used a Catboost as the model choice, the difference between worst and best upstream pipeline corresponds to a 5% reduction in mean RMSE. Table 4.6 shows that optimal selection of upstream steps can reduce RMSE by up to 27%. We note that finetuned models have smaller reductions from best to worst pipeline than their non-finetuned counterparts. However, this could be due to the constrained nature of the finetuned pipelines, as the tokenizer choice is fixed and the featurizer choice can only be `bert-mean` or `bert-cls`. Therefore, these finetuned mean RMSE reductions effectively only show changes associated with preprocessing, not the whole upstream pipeline.

Overall, although these reductions in RMSE are approximate upper limits on the improvement for a given model (as it is measuring from worst to best), they nonetheless suggest any given model can be substantially improved by tuning of the upstream steps, for these datasets.

Further investigating the importance of choices upstream of modelling, the variance associated with choosing the model was compared to the variance associated with choosing the upstream steps. We use this variance to quantify how ‘important’ it is to optimize certain parts of the pipeline, assuming that larger variance implies a greater potential for improvement from optimization. The variances were quantified as follows.

We defined an ‘upstream group’, as a group of pipelines that only differ by model, i.e. has the same preprocessing, tokenization and featurization. We then selected every ‘upstream group’ with more than one modelling choice, as variance calculation requires more than one sample. Variance in mean RMSE, across modelling choices, was determined for each group and then



(a) Upstream group and i, j indexing for $\bar{V}_{\text{model}}$.

(b) Model group and i, j indexing for $\bar{V}_{\text{upstream}}$.

Figure 4.3: Illustrations of upstream and model groups. Red indicates preprocessor choice, orange indicates tokenizer choice, yellow indicates featurizer choice and blue indicates model choice. Also illustrates the i and j indexing from Equation 4.1 for $\bar{V}_{\text{model}}$ and $\bar{V}_{\text{upstream}}$.

averaged across all groups, effectively computing a variance associated with choosing the model.

This average variance is denoted as $\bar{V}_{\text{model}}$:

$$\bar{V}_{\text{model}} = \frac{1}{N} \sum_i \text{Var}[\overline{\text{RMSE}}_i] = \frac{1}{N} \sum_i \frac{1}{J_i - 1} \left(\sum_j^{J_i} [\overline{\text{RMSE}}_{i,j} - \frac{1}{J_i} \sum_j \overline{\text{RMSE}}_{i,j}]^2 \right), \quad (4.1)$$

where i indexes *between* upstream groups; N is the count of upstream groups; j indexes *within* upstream groups which have J_i different models; and $\overline{\text{RMSE}}_{i,j}$ is average RMSE across cross validation folds for the j th pipeline in the i th upstream group. That is, we estimate variance over *model* choices as each upstream group has fixed upstream choices. Figure 4.3a illustrates an upstream group and $\bar{V}_{\text{model}}$ equation indexing.

We defined a ‘model group’, as a group of pipelines where each pipeline has the same model choice. For each model group, the variance in mean RMSE was determined. This variance was then averaged across all model groups, effectively computing a variance associated with choosing the upstream steps. This average variance is denoted as $\bar{V}_{\text{upstream}}$ and is calculated similarly to Equation 4.1 but with i indexing a *model group* instead, and j indexing between different upstream pipelines. That is, we estimate variance over *upstream* choices as each model group has a fixed model choice. Figure 4.3b illustrates a model group and $\bar{V}_{\text{upstream}}$ equation

indexing.

Table 4.7 presents these variances. For both datasets, the variance associated with upstream choices is of a similar scale to that associated with the model choice. This suggests that tuning the pipeline upstream from modelling is of similar importance to selecting the model, for these datasets.

Table 4.7: Comparison of average variance across either upstream pipeline or model choice.

Dataset	$\bar{V}_{\text{model}}$	$\bar{V}_{\text{upstream}}$
obl	16e+08	4.3e+08
jcp	3.4e+02	3.7e+02

4.5.5 Vector embeddings are not suited to regression with GBDTs

In this section we compare vector embeddings to BoW featurizers for GBDT models. Figure 4.4 shows the performance distribution with and without vector embedding featurization for GBDT models across both datasets. It can be seen that the performance of GBDTs was better for pipelines with traditional BoW featurizers (i.e. `bow-count`, `bow-binary`, `tf-df`) instead of vector embeddings (i.e. `bert-cls`, `bert-mean` and GloVE). Table 4.8 presents Monte Carlo permutation test results comparing GBDT pipelines using BoW and vector embedding featurization. We find that there is a significant difference in mean RMSE between BoW featurization and vector featurization for GBDTs. This suggests that for NL regression, the pipelines mixing ‘traditional’ GBDTs and ‘modern’ vector embeddings are less accurate than using BoW with GBDTs, for these datasets. A hypothesis for why GBDTs exhibit lower accuracy with vector embeddings is that these embeddings are trained with neural network

Table 4.8: Monte Carlo permutation test results for comparison of BoW and vector featurization for GBDT pipelines. P-values from 100,000 permutations. ‘mean-difference’ is difference of RMSE between BoW and vector embeddings, i.e. negative mean difference suggests that GBDT without vector embeddings is more accurate.

dataset	mean-difference (BoW - vector)	p-value
obl	-27347	<0.001
jcp	-26	<0.001

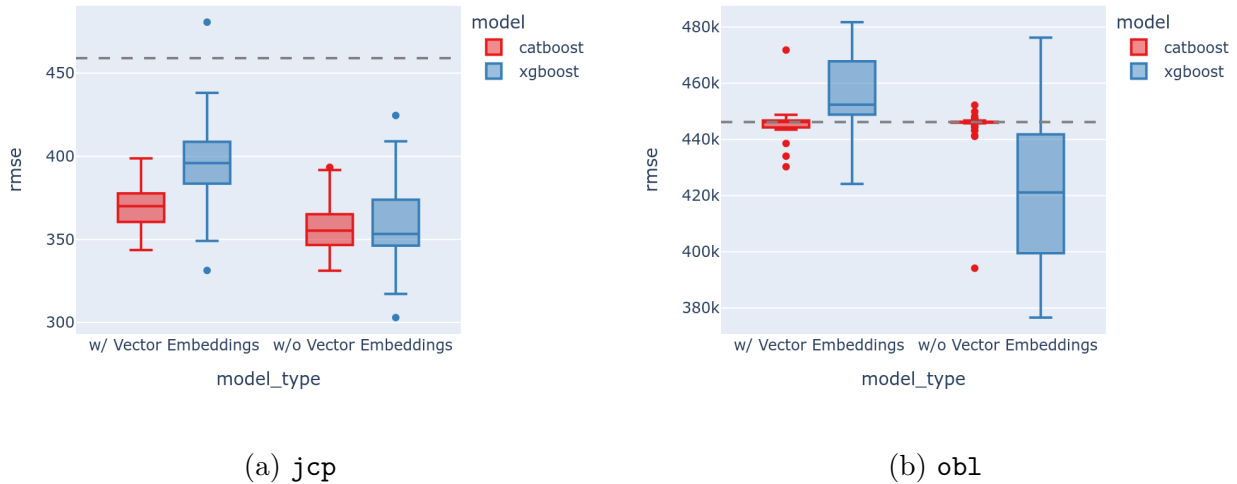


Figure 4.4: Comparison of GBDT models with and without vector embeddings. The dashed line is RMSE of predicting the training set mean. For `obl` some outliers have been removed to focus on the region of interest. We also note Catboost performed poorly across all pipelines for `obl`.

models, for various natural language tasks described in Chapter 2, Section 2.1. Therefore, it could be that to use the relevant information within the vector embeddings requires non-linear decision boundaries that do not align with axes; and are thus difficult for tree based methods to leverage.

4.6 Conclusion

This study systematically evaluated the impact of pipeline design choices on textual regression accuracy by analyzing randomly generated pipelines on two NL regression datasets, revealing several insights.

We found that contextual vector embeddings from BERT significantly outperformed non-contextual GloVe vector embeddings, highlighting the value of capturing context-aware semantic representations. However, BERT featurization only achieved comparable (`jcp`), or substantially better (`obl`), RMSE over traditional BoW when finetuned. This emphasizes the need to adapt pretrained language models to specific applications. We also found that BoW approaches are capable of providing good performance for natural language regression, which

could be particularly relevant for long textual inputs or with limited computational resources.

Furthermore, we found that upstream pipeline components – such as preprocessing, tokenization, and featurization – contributed variability in performance comparable to model selection itself. This comparable variability highlights the importance of optimizing upstream pipeline choices, not just the choice of model.

We also found that GBDTs paired with BoW featurization consistently outperformed GBDTs using vector embedding featurization. This suggests incompatibility between tree-based models and vector embeddings for natural language regression.

These findings collectively demonstrate the importance of holistic pipeline optimization for textual regression.

While this study explored a variety of pipeline configurations, it did not analyze all possible steps. Future work could, for instance, extend the analysis to other modeling approaches, like convolutional neural networks or support vector regression, which have been used in some prior publications. Furthermore, a deeper exploration could involve *separately pretraining* vector embedding featurizers (e.g., BERT or GloVe) with different tokenizers and preprocessors from scratch, removing the practical constraints imposed by using pretrained featurizers. Although this would be computationally intensive it is an interesting avenue for future work.

Finally, it is worth noting that there is substantial difference in the performance of different pipelines for each dataset. For example, contextual vectors with finetuning beat BoW for `obl`, but only achieved comparable performance for `jcp`. In addition, the top ten pipelines (by average RMSE) are distinctly different in qualitative nature between the datasets. The top ten for the `obl` dataset is primarily composed of BERT featurizers, whereas there is a mix of BoW and vector embedding approaches in the top ten for `jcp`. This suggests that there could be underlying properties of these datasets that makes them amenable to a certain form of featurization, and discovering these properties would be an interesting direction for future research.

Chapter 5

Evaluating ensembling for text and tabular modelling

5.1 Introduction

Historically, much machine learning research considered either modelling with *structured* (a.k.a. *tabular*) data **or** *unstructured* data, e.g. text, images, audio etc. In Chapter 3 we discussed some issues that can occur when overlooking missing value handling in tabular modelling. In Chapter 4 we investigated the role of different pipeline components in unstructured text modelling. In this Chapter we will look at joint text and tabular, or text-tabular, modelling. As discussed in Chapter 3, there is research that proposes specific DL architectures that claim to have comparable accuracy, or even outperform, GBDTs for tabular data (Arik and Pfister, 2021; Gorishniy et al., 2021; Kadra et al., 2021; Popov, Morozov, and Babenko, 2020). However, large studies (Grinsztajn, Oyallon, and Varoquaux, 2022; McElfresh et al., 2023; Tschalzev et al., 2024) as well as the results in Chapter 3 suggest GBDTs are on average the more accurate model for tabular data. On the other hand deep learning has come to dominate unstructured data problems involving text, audio, images, video etc. Importantly for this chapter, deep learning has also come to dominate *multimodal* learning tasks for unstructured data. We use multimodal learning to refer to the combination of data from different ‘modalities’. A modality

is typically explained by example: text, image, audio are all different modalities. Unstructured multimodal tasks, which DL approaches dominate, include captioning images (text + image, Radford, Kim, Hallacy, et al., 2021), captioning audio (text + audio, Radford, Kim, Xu, et al., 2023), large language models which can input and output text and images (Yin et al., 2023), video generation models (text + video, Liu et al., 2024) and others.

However, relatively little work has studied how to combine *structured* and *unstructured* data. In this chapter we focus on text-tabular modelling, where text-tabular modelling refers specifically to models or ensembles of models that use *both* text and tabular data, as apposed to text-only or tabular-only models.

There are many practical tasks where the underlying data has both a structured tabular component and some free-form text associated with each observation. For example, in claim reserving (explained in Chapter 3), there is tabular information associated with the policy and accident - as well as a text associated with both the damage to the vehicle and the accident. In financial stock prediction problems there is tabular information (e.g. historic prices) and textual company reports. In medical applications we may wish to predict prognostic events such as likelihood of readmission given text (e.g. doctor's notes) and tabular (e.g. gender, blood pressure) data from electronic health records. Similarly, in commercial applications where we wish to regress the price of a product (e.g. to list on an auction, or know how much to bid in one) we typically have both tabular features of the product (e.g. mileage of a vehicle) and textual descriptions of the condition or quality of the product.

As GBDTs are on average more accurate for structured data regression (discussed in Chapter 3), and transformer-based DL dominates the unstructured modelling literature (also achieving competitive regression accuracy for text-only regression, with finetuning, in Chapter 4) it is unclear how to jointly model text and tabular data. However, there are three clear initial directions. Firstly, modelling everything with a GBDT, a competitive state of the art tabular model. The text can be represented using BoW, concatenated with tabular features, and input into the GBDT for prediction. We specifically do not consider using representations from vector featurization, as their performance with GBDTs was observed to be poor in Chapter 4.

Secondly, everything could be modelled end to end with one DL model, the state of the art approach for unstructured modelling. In such a purely DL approach we can have separate DL layers acting on the tabular and textual data, which are combined with some ‘fusion’ layers, and trained end to end to produce a prediction. Thirdly we could train separate models for text and tabular data, using the leading model for each; then ensemble the predictions from each modality. In this work we will compare these three approaches on three datasets for the task of price prediction.

Section 5.2 presents prior work on text-tabular modelling. Section 5.3 describes the experimental method used to compare different modelling approaches. Section 5.4 presents and discusses the results.

5.2 Related work

Most publications in the field of joint text-tabular modelling have focused on specific ad hoc applications, most notably in the domains of medicine (Arnaud et al., 2020; Seinen et al., 2022; Suvon et al., 2022; Zhang et al., 2020) and finance (Holtz and Gomez, 2023; Sardelich and Manandhar, 2018). Despite the rise in ad hoc text-tabular applications; there has been relatively little work approaching the problem methodologically (Ebrahimi et al., 2024; Gu and Budhkar, 2021; Shi et al., 2021). In this section we will clarify terminology, then give an example of an ad hoc study to illustrate the limitations of such research and the need for more systematic study. We will then discuss work which studies the problem of text-tabular modelling more systematically. Finally, we will conclude by highlighting that an ensemble of text-only and tabular-only models is a notable gap in the literature.

5.2.1 Terminology

In principle there are at least two ways to use textual and tabular data jointly in a prediction task: *ensembling* and *fused-representation* modelling. As different modelling strategies appear

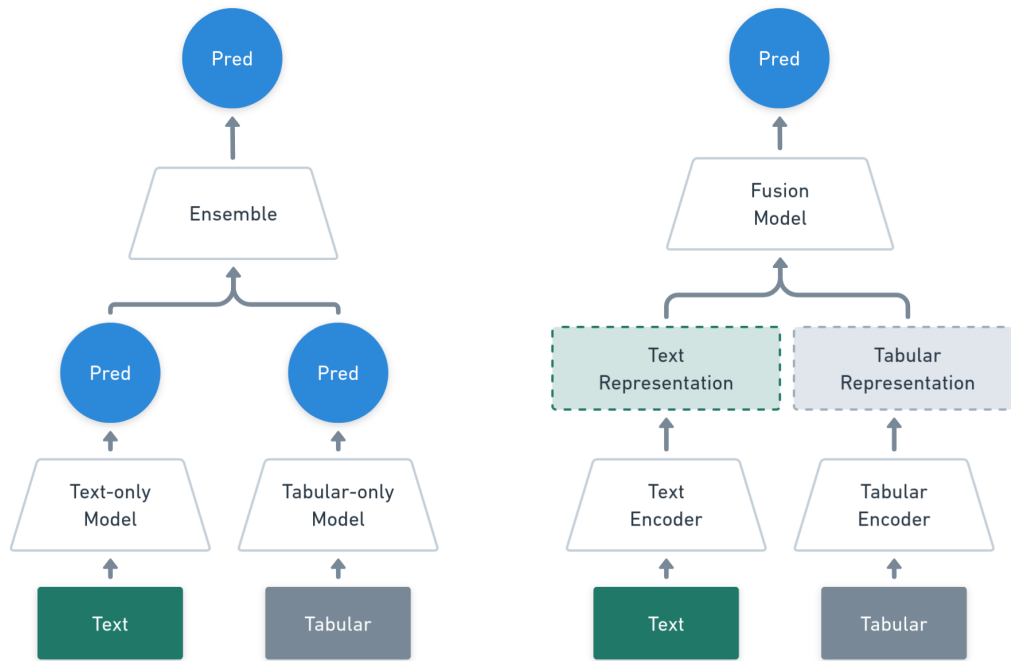


Figure 5.1: Architectural differences between ensembling (left) and fused-representation modelling (right). Green indicates relation to textual data, grey to tabular. Trapezoids are models or sub-components of models. Blue circles correspond to predictions. Dashed boxes correspond to numerical representations. Solid boxes correspond to data.

best for text and tabular data (DL and GBDT respectively), it is reasonable to approach text-tabular modelling by training a separate model on each data modality. To combine the predictions from each modality, a separate model can be trained to *ensemble* predictions into one, depicted in the left panel of Figure 5.1. Alternatively, a single model could be trained on both the text and tabular data at the same time to predict the label. This involves getting numeric representations of the text and tabular data and then in some way combining them (usually with concatenation) into a single *fused-representation*. This fused-representation is then put through a ‘fusion’ model, to obtain a final prediction, depicted in the right panel of Figure 5.1. Figure 5.1 illustrates the difference and establishes a diagrammatic convention we will use for all models discussed.

5.2.2 An example of ‘ad hoc’ text-tabular modelling in medicine

Joint text-tabular modelling in the medical field is mainly focused on prognostic classification models, most of which use fused-representations (Seinen et al., 2022). For example, Zhang

et al., 2020 studied the application of convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to multiple hospital triage classification tasks. The work focused on constructing an architecture that can handle time series, including text time series (e.g. doctor's notes) and tabular time-series (e.g. blood pressure readings) associated with each hospital visit; as well as some time invariant tabular data (e.g. gender and age). Zhang et al., 2020 found that the classification accuracy using both text and tabular data is substantially higher than using either modality alone. Although there is some comparison of sub-component choice in the same overall architecture, there is no comparison to alternative generic text-tabular models or ensembles. Also, as can be seen in Figure 5.2, the architecture was specific to the time series nature of the data. This example illustrates that, although valuable for providing initial insights, there a key issue with application focused ad hoc studies: each architecture is often custom to the application, mainly aiming to achieve acceptable performance. Furthermore, in ad hoc work there is often no tabular-only or text-only baseline to verify the fusion models are improving over the raw baselines. Finally, although this study investigates modifications to it's proposed architecture, often other ad hoc studies to not compare between different approaches making the results difficult to contextualize.

5.2.3 Methodological text-tabular studies

As mentioned before, there has been a small number of recent papers that considered text-tabular modelling as a generic problem instead of through an application. We discuss these papers in more detail below.

Gu and Budhkar, 2021 proposed a completely DL approach, illustrated in Figure 5.3. They used a large pretrained transformer (BERT, Devlin et al., 2019) to map the text to a representation and studied different ways to represent tabular data prior to fusion; as well as different ways of fusing the modality representations. The whole architecture was jointly trained through backpropagation - including the pretrained model. Gu and Budhkar, 2021 found that there is no most accurate way to fuse tabular data with textual data, as the fusion mechanism with the best accuracy varied between the three datasets studied. However all fusion models outperformed

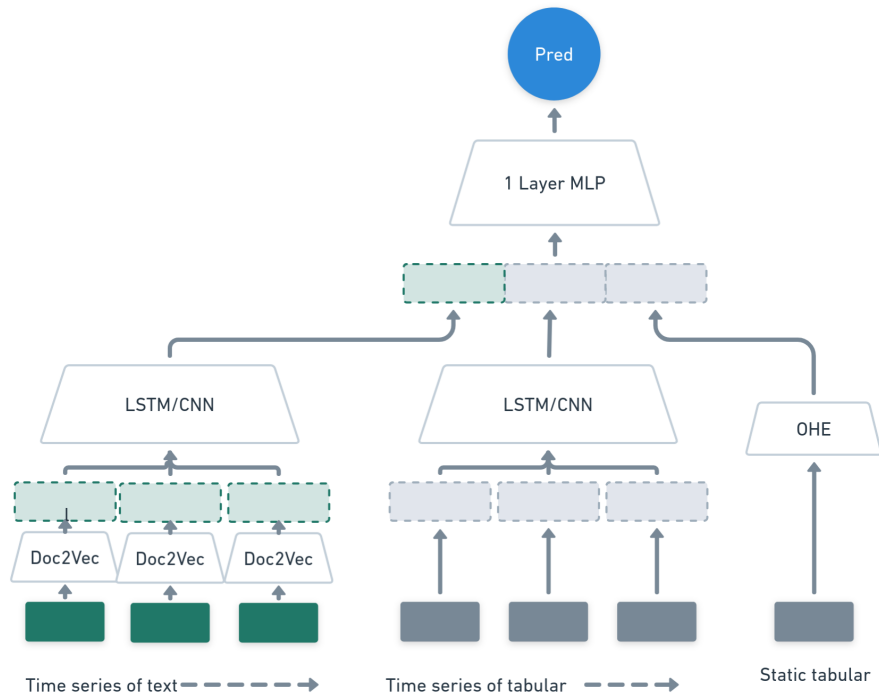


Figure 5.2: Example architecture from applied medical literature (Zhang et al., 2020). No model block between tabular data and tabular representation indicates that the data was used directly without processing, as it was already fully numeric. OHE means one hot encoding. LSTM/CNN indicates the paper studied both approaches. ‘Static tabular’ refers to tabular data with no time-variation, e.g. gender.

text-only models. Also, simple fusion methods such as using an MLP on the tabular data, concatenating with textual representation and then fusing with a classification/regression MLP worked well. No tabular-only models were included as a baseline. Notably, the study did not use common transfer learning best practices designed to adapt large pretrained models such as warmup scheduling, layerwise learning rate decay, unsupervised finetuning or gradual unfreezing (Howard and Ruder, 2018).

Shi et al., 2021 also approached text and tabular modelling as a generic problem, however they performed more comprehensive baselines across many datasets. Similar to Gu and Budhkar, 2021, Shi et al., 2021 proposed a completely DL approach with the text represented using a pretrained transformer and the tabular data represented using MLPs, see Figure 5.4. Like Gu and Budhkar, 2021 the whole architecture was jointly trained through backpropagation - including the pretrained model. Also like Gu and Budhkar, 2021 they investigated different approaches to fusion - with some overlap in the approaches studied. Shi et al., 2021 also noted that a simple fusion mechanism using an MLP achieved better accuracy on the majority

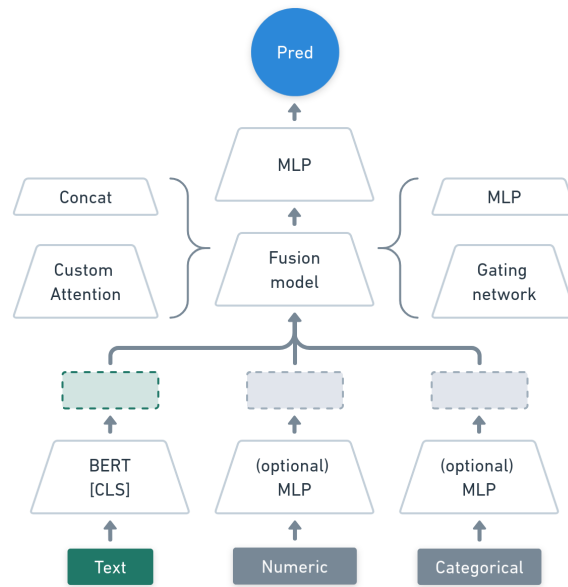


Figure 5.3: Architecture of Gu and Budhkar, 2021. BERT [CLS] indicates the CLS token was used to pool BERT outputs. The study compared fusion mechanisms - indicated by braces. Numeric and categorical tabular features could optionally have their own separate MLP.

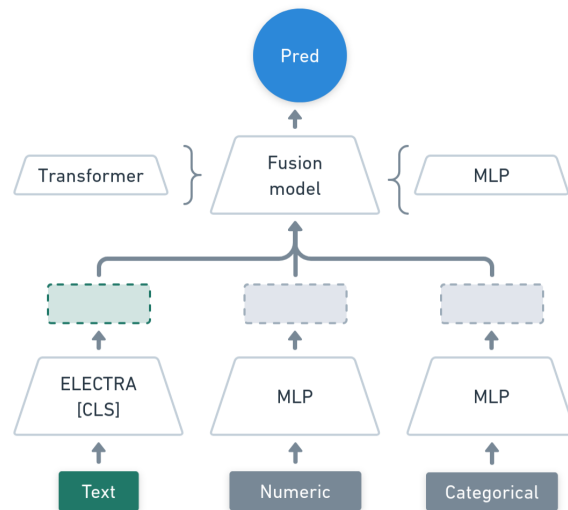


Figure 5.4: Architecture of fused-representation model from Shi et al., 2021. The fusion model was varied between many fusion mechanisms - indicated by braces. ELECTRA (K. Clark et al., 2020) is the same architecture as BERT but is pretrained in a different manner.

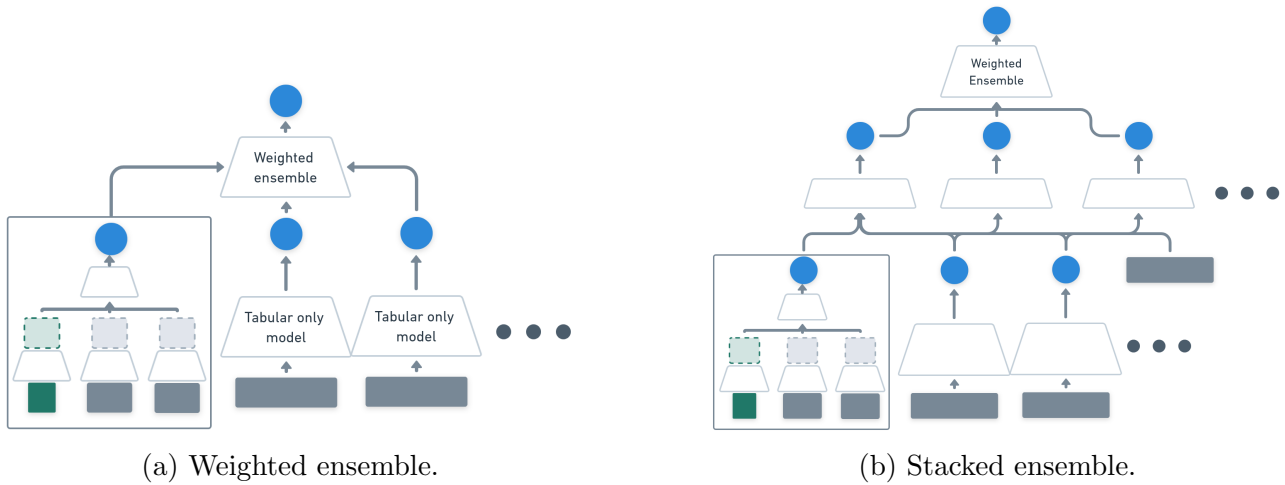


Figure 5.5: Different ensembling approaches from Shi et al., 2021. The fused-representation model is indicated in the bottom left with a square box. Dots indicate other tabular-only models were included in the ensemble.

of datasets, as compared to fusing with a transformer. However, unlike Gu and Budhkar, 2021, Shi et al., 2021 presented both text-only and tabular-only baselines; finding their fused-representation model to outperform both approaches across almost all of the datasets studied. Importantly, Shi et al., 2021 considered ensembles. However, instead of a simple ensemble of a text-only and tabular-only model, they studied an ensemble of their *fused-representation* model with tabular-only models. They studied both weighted ensembles (see Figure 5.5a) and somewhat more complex ‘stacked ensembles’ (see Figure 5.5b). These stacked ensembles train both tabular-only models as well as the *fused-representation* model to predict the target, then concatenate these predictions *again* with the original tabular data and train *another* ensemble of models on that concatenated data. They found the most performant model to be this stacked ensemble – adding this model to their AutoML library AutoGluon (Amazon, 2024). Notably, despite the variety of complex ensembling approaches studied, this work *did not include* a simple ensemble of a text-only and a tabular-only model. We note Shi et al., 2021 did use some best practices for transfer learning pretrained transformers: layerwise learning rate decay and warmup schedules. Although they did not use gradual unfreezing or unsupervised finetuning of the model.

Ebrahimi et al., 2024 also approached text-tabular modelling as a generic problem with a fused-representation approach. They used a pretrained transformer for text representation and

TabNet (Arik and Pfister, 2021) for tabular representation with a transformer as the fusion model. Their model also supported images with a ViT-B/16 encoder (Dosovitskiy et al., 2021) and time series with a custom transformer. The overall model is called LANISTR, illustrated in Figure 5.6. The main difference between the work of Ebrahimi et al., 2024 and Gu and Budhkar, 2021 or Shi et al., 2021, beyond being compatible with extra modalities, is that they proposed a novel unsupervised pretraining objective. The pretraining objective is an amalgamation of reconstruction losses for each modality (e.g. masked language modelling for text, masked feature modelling for tabular, masked image modelling for images etc.), as well as a fused-representation cosine reconstruction loss illustrated in Figure 5.7. Ebrahimi et al., 2024 evaluated their model with and without the pretraining objective on two classification problems: a hospital mortality task (referred to as MIMIC-IV) and a product rating prediction task (referred to as Amazon Reviews). They included AutoGluon’s fused-representation model (Shi et al., 2021) as baseline for the Amazon reviews task, evaluating AutoGluon with both an MLP and transformer fusion layers. They found their architecture with or without pretraining to be the more accurate architecture out of the multimodal models studied. They also find that using the pretraining objective substantially improves performance over not using it for both tasks. However, we note that the datasets studied both included images and one dataset did not contain tabular data - rather including time series. This complicates generalizing the results to purely text-tabular contexts. Specifically, it is unclear whether the importance of the proposed pretraining objective extends to text-tabular price regression. Furthermore, the lack of text-only, tabular-only, BoW or ensemble baselines makes it difficult to contextualize their results with the results of Shi et al., 2021. Finally, Ebrahimi et al., 2024 also used some of the transfer learning best practices: unsupervised finetuning, scheduling and partial gradual unfreezing. However, they do not use layerwise learning rate decay.

We note that discussion of the runtime of these methods is difficult to compare, as different methods use different implementations, with different libraries, different versions of those libraries that make better or worse usage of modern hardware etc. Overall, this makes a fair comparative analysis of the speed of these methods difficult. Although CPU times can be determined; we note that even such simple comparisons of available implementations require

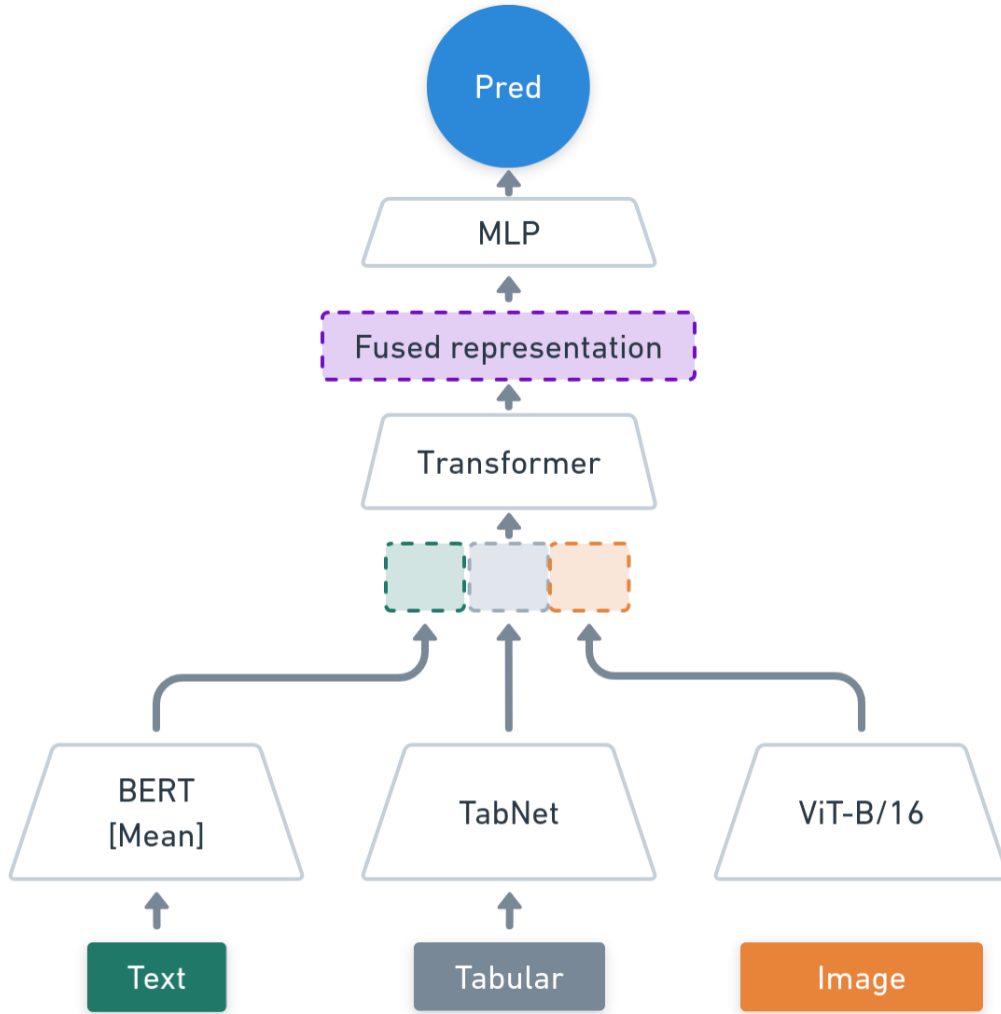


Figure 5.6: Architecture of LANISTR. Orange indicates image inputs. Purple dashed box is the fused-representation. BERT [Mean] indicates BERT outputs are pooled using the mean across tokens.

standardization of server load, memory usage (for fair comparison of memory cache hits etc.) and other technical details that are outside the scope of this thesis.

5.2.4 Gap in the literature: simple ensembles

The reviewed literature shows that text-tabular modelling is useful in various fields outside of price regression. Notably, in medical (Arnaud et al., 2020; Seinen et al., 2022; Suvon et al., 2022; Zhang et al., 2020) and commercial contexts (Ebrahimi et al., 2024; Gu and Budhkar, 2021; Shi et al., 2021). Relatively few studies approach text-tabular modelling methodologically. By methodologically, we mean proposing generic models or comparing performance across different

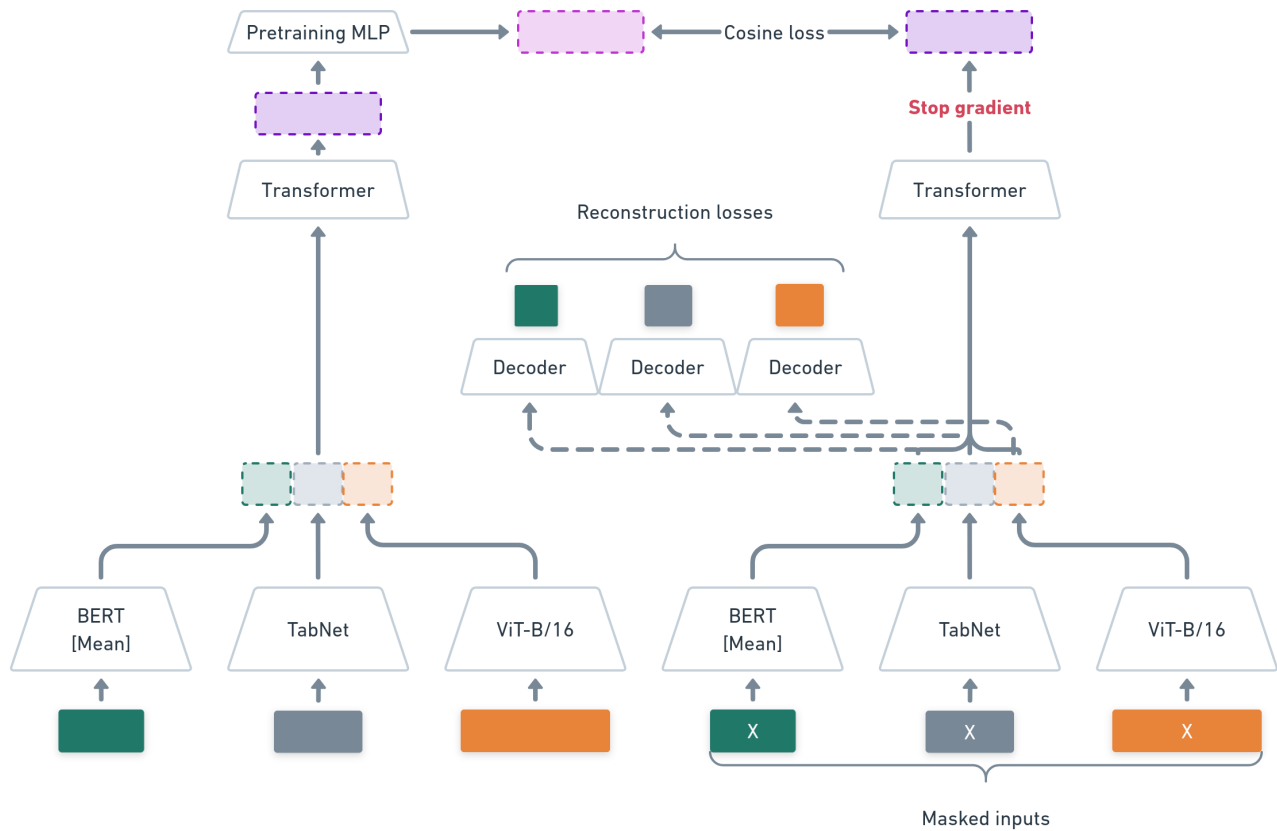


Figure 5.7: Outline of pretraining algorithm. Masked inputs marked with X, masking for text is done by replacement with [MASK] token, masking for tabular is done by zeroing a feature, masking for images is done by [MASK] patch tokens. The overall loss consists of individual reconstruction loss for each, added to a cosine reconstruction loss for the fused-representation.

models. Of those studies, only one (Shi et al., 2021) include a study of ensembles, where they give competitive performance. However, there is a notable gap in investigating a simple ensemble of separately trained text-only/tabular-only models. Therefore, in this chapter we will investigate such a model applied to the task of price regression. We will study a finetuned BERT as the textual model, using transfer learning best practices, and Catboost, a GBDT, as the tabular model. The rationale for choosing these models is that DL transformers are the ‘defacto’ standard for most modern NLP tasks and that GBDTs are considered to be the better model for tabular data (Grinsztajn, Oyallon, and Varoquaux, 2022; McElfresh et al., 2023). This ensemble will be compared to the models proposed in Shi et al., 2021 and Ebrahimi et al., 2024 as well as a tabular-only, text-only and a concatenated BoW baseline.

5.3 Method

In this Section we begin with a detailed description of the ensemble we evaluate. We then describe the datasets; the experimental procedure used to evaluate the ensemble; and the models included in our evaluation.

5.3.1 Ensemble model

Our ensemble model consisted of a DL text-only model and Catboost (GBDT) tabular-only model, see Figure 5.8. We use BERT (Devlin et al., 2019) to featurize the text, with ‘concat’ pooling to create a fixed size representation of the text and finally a one layer MLP to map the representation to a prediction. Concat pooling was proposed in Howard and Ruder, 2018 and concatenates the [CLS] token output, the mean (elementwise) token output and the maximum (elementwise) token output. We also used each of the best practices for transfer learning from Howard and Ruder, 2018. Namely, we applied layerwise learning rate decay, a warmup learning rate schedule, unsupervised finetuning and gradual unfreezing (Howard and Ruder, 2018). We used a layerwise learning rate decay of $1/2.6$ and a triangular learning rate schedule for all learning stages (unsupervised finetuning, supervised frozen finetuning and supervised unfrozen

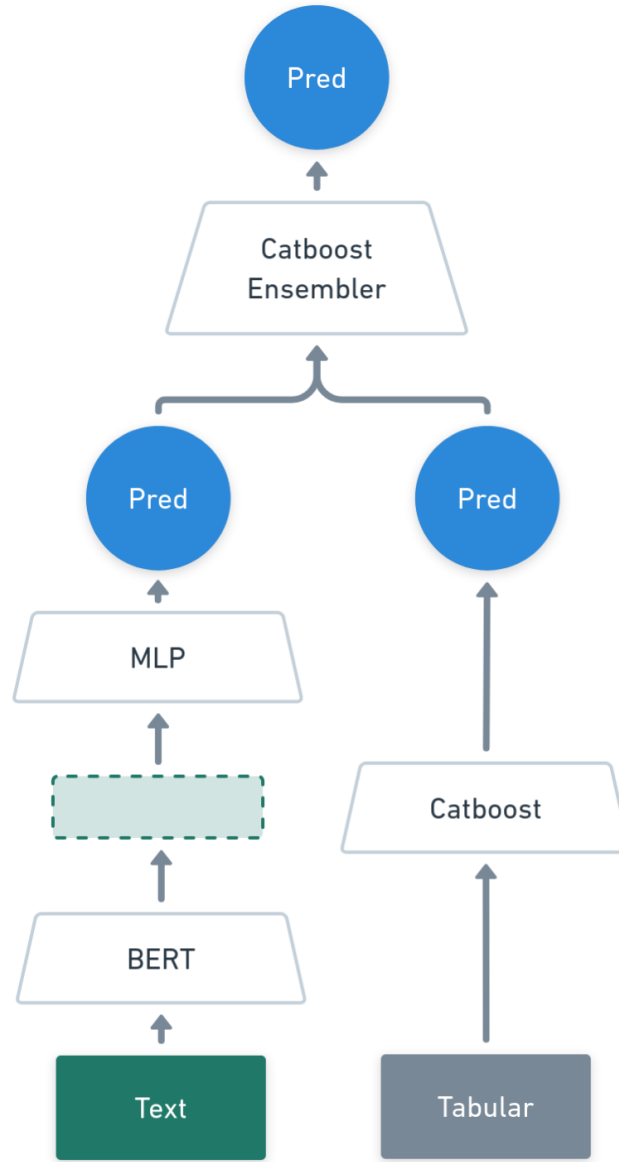


Figure 5.8: Our BERT based text-only and tabular-only ensemble architecture.

finetuning) as recommended by Howard and Ruder, 2018. For the unsupervised finetuning, we perform the BERT masked language modelling task with a masking probability of 15% for each dataset. For gradual unfreezing, we start training the regression MLP with the BERT encoder frozen and train the regression head until supervised validation loss plateaus. We then unfreeze the *whole* BERT model and train until supervised validation loss plateaus.



Figure 5.9: Response distribution across datasets. Histogram has y axis on log scale.

5.3.2 Experimental procedure

We use three different price regression datasets to compare different approaches for text and tabular regression:

1. `jcp` - A dataset for predicting consumer good prices from textual descriptions, as well as tabular data such as total number of reviews and the brand.
2. `obl` - A dataset for predicting boat prices from online auction textual descriptions, as well as tabular data such as the year of manufacture and fuel type.
3. `car` - A dataset for predicting car auction advert prices from listing descriptions, as well as tabular data such as mileage and number of owners.

Datasets `jcp` and `obl` were also used in Chapter 4 for investigating text-only regression, however for this chapter the tabular data was included in the modelling. Categorizing textual, numeric and categorical columns is not always clear cut, which was also recognized in Shi et al., 2021. For example the names of a product could be considered text or a high-cardinality categorical -

Table 5.1: Dataset properties. N, C, T correspond to counts of number of numeric, categorical, and text columns respectively.

Dataset	Total Instances	N	C	T
car	58609	4	1	1
ob	1850	2	8	3
jcp	13575	2	1	2

depending on how generic the names are. For each dataset the numeric, textual and categorical column allocations were all manually chosen, including an allocation for ‘irrelevant’ columns such as row id. A summary of the datasets is presented in Table 5.1 and Figure 5.9. Given that Chapter 3 highlighted tabular modelling can be sensitive to missing value handling schemes for not-MCAR data, we checked for informative missing values to assess whether HPO of missing value handling was required. As a crude check for informative missing values, we fit a Catboost model using the `binarize` strategy from Chapter 3. However, the performance was the same as a mean baseline and so HPO of missing value handling was not included as part of this investigation.

Each dataset was initially split into `test` and `not-test` partitions - with a 20/80 split using the label as a stratifying variable. Each `not-test` partition was then randomly subdivided into `train` and `val` partitions, with an 80/20 split. This random subdivision into `train` and `val` partitions was repeated five times in total; to be used for Monte Carlo cross-validation. The `val` partition was used to trigger early stopping for all models.

For each dataset, and each Monte Carlo cross-validation partition, the following models were trained - also summarized in Table 5.2:

- **catboost-tabular**: A Catboost tabular-only model. This model was included to assess the relative performance of tabular-only models.
- **bert-text**: *only* a BERT regressor model, as depicted in the left hand side of Figure 5.8, similar to that of the finetuned MLP model of Chapter 4. This model was included to assess the relative performance of text-only models.

Table 5.2: Summary and comparison of modality handling methods for each model compared in this chapter. ‘Interactions?’ column indicates whether text and tabular features can interact, i.e. impact of one modality can depend on the other. * indicates the Catboost model was used on concatenated tabular and textual features. We note ELECTRA is a flavour of BERT.

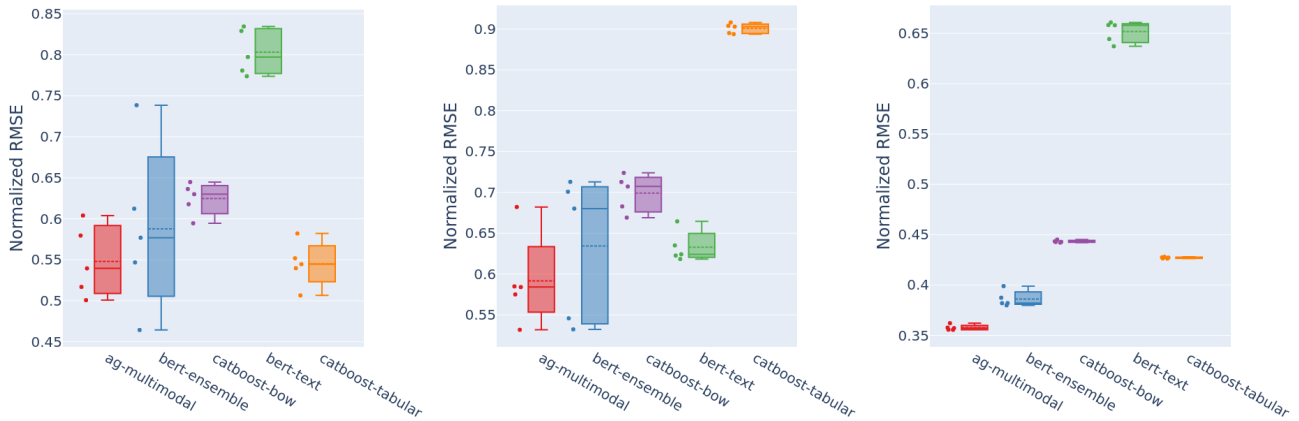
Orange indicates DL methods. Olive indicates GBDT methods.

Method	Tabular	Textual	Fusion mechanism	Interactions?
catboost-tabular	Catboost	N/A	N/A	x
bert-text	N/A	BERT	N/A	x
bert-ensemble	Catboost	BERT	Catboost	x
ag-multimodal	MLP	ELECTRA	MLP	✓
lanistr	TabNet	BERT	Transformer + MLP	✓
catboost-bow	Catboost*	BoW	Catboost*	✓

- **bert-ensemble**: The ensemble depicted in Figure 5.8, using a BERT regressor for the textual data, Catboost for the tabular data and a Catboost model for ensembling of predictions. This approach has not been previously studied.
- **ag-multimodal**: The fused-representation model proposed in Shi et al., 2021. This model was *without* further stacked ensembling as the direct comparison between fused-representation and ensembling text-only/tabular-only models was of interest.
- **lanistr**: The model proposed by Ebrahimi et al., 2024, depicted in Figure 5.6. We used the proposed unsupervised pretraining objective (depicted in Figure 5.7) followed by supervised finetuning with the textual encoder frozen.
- **catboost-bow**: A Catboost model applied to the concatenation of tabular data with a BoW featurization of the text. This acts as a baseline representing a ‘traditional’ method for text-tabular modelling.

Each trained model was evaluated on the same `test` partition using RMSE, producing five samples of each metric across the Monte Carlo cross-validation.

We note that the code implementation of LANISTR provided by Ebrahimi et al., 2024 had both syntactic and semantic bugs throughout, especially in the pretraining code for TabNet. These bugs were addressed to run the analysis in this chapter, however this means this analysis did not use an exact replica of the code provided by Ebrahimi et al., 2024.



(a) `obl`. Normalizing mean base-line RMSE = 406,018 (b) `jcp`. Normalizing mean base-line RMSE = 459. (c) `car`. Normalizing mean base-line RMSE = 2945

Figure 5.10: Boxplot of *normalized* RMSE for each Monte Carlo cross validation sample, across datasets, excluding `lanistr`. The normalization was done as described in the text of Section 5.4. Dashed lines in boxplots represent mean of the samples, solid line the median. Individual normalized RMSE samples are scattered next to boxplots with jitter for visibility. The two rightmost boxplots in each figure are for methods that *use only one modality*.

5.4 Results and Discussion

Figure 5.10 shows the normalized RMSE of each model across each dataset, excluding `lanistr`. For clarity, a normalized RMSE of 1 indicates performance equivalent to predicting a constant `not-test` label mean for all inputs, and a normalized RMSE of 0.8 indicates a 20% reduction in RMSE relative to said mean prediction. We note that `lanistr` performed poorly across all datasets, resulting in performance similar to a mean prediction for all cases. We provide Figure 5.10 with `lanistr` included in Appendix A.3.1. Investigating the poor training results further, we found that the pretraining objective associated with the tabular portion of the model (TabNet Arik and Pfister, 2021) exhibited rapid gradient explosions due to instability in the reconstruction loss associated with tabular features. We also note that the implementation of `lanistr` provided by Ebrahimi et al., 2024 practically differed from that reported in their work, as the implementation contained semantic errors that ignored all TabNet pretraining. This suggests that care needs to be taken if including a tabular reconstruction loss in the pretraining of the `lanistr` model and that further work is required to make this model stable and easy to train.

Figure 5.10a shows normalized RMSE values for the `obl` dataset. The figure shows that

`catboost-tabular`, `bert-ensemble` and `ag-multimodal` models perform comparably. The `bert-text` showed a large normalized RMSE, although it still outperformed a mean baseline by around 20%. Out of the text-tabular models, `catboost-bow` had higher RMSE than other approaches (`bert-ensemble` and `ag-multimodal`). The comparable performance of the best text-tabular models with the tabular-only model suggests that the signal in the textual data is already in the tabular data, for `obl`. This highlights the importance of running tabular-only (and by extension text-only) model baselines for text-tabular problems, as it may be that only one of the data modalities is sufficient for the regression. The higher RMSE of `catboost-bow` compared to `catboost-tabular` indicates that concatenation of BoW features reduces the prediction accuracy for Catboost for this dataset. This suggests Catboost could not ignore irrelevant textual features in the BoW.

Figure 5.10b shows normalized RMSE values for the `jcp` dataset. The figure shows that `ag-multimodal` had the lowest mean and median RMSE of all of the models. Although `bert-ensemble` achieved comparable performance, we note the higher variability associated with this ensembling approach. It could be noted that the single modality `bert-text` model achieved better performance than the text-tabular `catboost-bow` model. Also, the tabular-only `catboost-tabular` RMSE was relatively high, only reaching around a 10% reduction in RMSE over the mean prediction. However, we note that `ag-multimodal`, a text-tabular model, consistently performed better than a text-only `bert-text`. This suggests the presence of complementary signal the text and tabular data of this dataset, and the ability of a fused-representation architecture to extract this signal better than BoW concatenation and more consistently than a simple ensemble.

Figure 5.10c shows normalized RMSE values for the `car` dataset. In this dataset, both the text and tabular modalities substantially reduced the RMSE on their own, relative to the mean baseline. The `bert-text` model achieved around a 35% reduction and `catboost-tabular` around a 57% reduction in RMSE. As in the other datasets, the text-tabular `catboost-bow` showed worse RMSE than the best single modality model (`catboost-tabular`). However, the `ag-multimodal` and `bert-ensemble` both outperform either modality in isolation; highlighting the importance of joint text-tabular modelling. Notably, the `ag-multimodal` model performs

distinctly better than `bert-ensemble` for this dataset.

Overall, both `bert-ensemble` and `ag-multimodal` were consistently amongst the most accurate models. Both models showed the same, best, accuracy on the effectively tabular-only `obl` dataset, indicating resilience to irrelevant signals from the text modality. Although the `bert-ensemble` and `ag-multimodal` performed comparably on `jcp`, the fused-representation `ag-multimodal` achieved higher accuracy on the `car` dataset. Furthermore, `ag-multimodal` exhibited lower variability in RMSE across all datasets. These findings suggest that the fused-representation model, `ag-multimodal`, is a strong approach for text-tabular regression. In particular, the simple ensemble (`bert-ensemble`, which was previously unevaluated) did not surpass the fused-representation model on the studied datasets. More generally, an ensemble approach, by construction, cannot extract interactive effects between features and text. The better performance of fused-representation, for at least the `car` dataset, suggests that there is important ‘early’ (i.e. word to feature) multimodal *interactions* that the fused-representation approach is able to extract.

Additionally, across all datasets, the `catboost-bow` model consistently showed higher RMSE than the best single-modality model. This suggests that concatenating BoW features with tabular data for use with Catboost is not an effective strategy for text-tabular regression in these datasets.

5.5 Conclusion

In this chapter, we studied a previously unevaluated ensembling strategy for textual-tabular data in the context of price regression. We compared a simple ensemble to a fused-representation approach, as well as a concatenated BoW text-tabular approach. Text-only and tabular-only baselines (using a finetuned BERT regressor and CatBoost respectively) were also included for comparison. Our analysis provided the following insights.

First, our experiments showed that while the simple ensemble (`bert-ensemble`) model performed comparably for `jcp` and `obl`, the fused-representation (`ag-multimodal`) outperformed

the simple ensemble on the `car` dataset. Furthermore, the fused-representation model also demonstrated lower variability in RMSE across all datasets. This suggests that fused-representations are more effective at capturing complementary signals from both text and tabular modalities.

Second, our results showed that the `catboost-bow` model (which concatenates bag-of-words features with tabular data) consistently underperformed relative to the best single-modality models. This suggests that Catboost with concatenation of BoW was not a strong text-tabular approach for our data. This model was either unable to sufficiently filter irrelevant signal or leverage complementary signal where it existed.

Third, we found that LANISTR consistently only reached mean baseline performance. Further investigation showed that this was due to exploding gradients from the pretraining objective associated with the masked tabular modelling. This suggests that more work is required to make this model stable to train.

Finally, in several cases, the tabular-only and text-only models provided competitive performance, highlighting the importance of including single modality baselines when evaluating multimodal approaches.

Overall, we conclude that the previously published (Shi et al., 2021) fused-representation model (`ag-multimodal`) is a strong approach for text-tabular modelling. Importantly, even though our results highlight the performance of `ag-multimodal`, we have for the first time benchmarked it against a simple ensemble approach. We found that the simple ensemble showed comparable results in some datasets. Therefore, future studies in text-tabular regression would benefit from inclusion of a simple ensemble baseline.

Future work could investigate application of the text-tabular methods studied here on the insurance dataset of our industry partner, with missing value handling included in HPO in light of the results of Chapter 3. Furthermore, in light of the results of Chapter 4, the `catboost-bow` model may achieve better performance through HPO of the NLP pipelines.

Chapter 6

Conclusion

Throughout this thesis we have investigated the inclusion of text in price regression. This is an important problem as regression is used throughout society and is relevant to a wide variety of industries, e.g. in car insurance (discussed in Chapter 3).

Before assessing how to include textual data we began with a study of how to baseline current methods for tabular regression. Chapter 3 especially focused on the importance of missing value handling for tabular modelling, with application to real data from our industry partner (a leading UK car insurer). Specifically, we assessed the relative performance of two competing tabular data modelling approaches, deep learning (DL) and gradient boosted decision trees (GBDTs), under different missing value handling schemes for the task of claim reserve regression. Notably, the real insurance data was distinctly not-missing completely at random (MCAR). In the context of this informative missingness, we identified that the comparison of DL and GBDTs can be heavily influenced by the choice of missing value handling scheme. We found that under two of the schemes there was no substantial difference between these modelling approaches – however under mean imputation or less-than-minimum imputation the GBDT model was more accurate. This work established the importance of missing value handling for claim reserving applications and also contributed to the wider discussion on how to benchmark DL against GBDTs (Larionov, Adams, and Webster, 2024). Namely, it provided rigorous empirical demonstration of how using appropriate missing value handling caused GBDTs to achieve

better relative accuracy to DL than it would appear under schemes used in prior work, with not-MCAR data. This highlights the need to include not-MCAR data in future comparisons between GBDTs and DL and the need for careful consideration of the interaction between missing value handling schemes and modelling approaches.

Subsequently, Chapter 4 highlighted the growing interest in natural language regression, along with a gap in the literature in the systematic study of natural language (NL) regression pipelines. In this chapter, we randomly sampled hundreds of NL regression pipelines with varying choices in the preprocessing, tokenization, featurization and modelling. These pipelines were then evaluated on two price regression datasets. From evaluating these pipelines we confirmed that contextual vector embeddings outperform non-contextual. We also provided quantitative evidence that vector embeddings performed poorly with GBDT models. Furthermore, we established several other interesting novel observations on the construction of pipelines. We found that finetuning was required for modern contextual vector embeddings (as implemented by BERT) to have regression accuracy comparable to, or better than, traditional bag of words (BoW). Additionally, for the first time, we quantified the relative importance associated with different pipeline steps. We found that choices upstream of the modelling step contributed variance comparable to choice of the model itself, highlighting the importance of tuning the entire pipeline and not just the model choice.

In Chapter 5 we concluded by studying the emerging field of joint text-tabular modelling. We began by outlining the main approaches studied so far and identified that study of a simple ensemble was a notable gap in the literature. Specifically, we studied an ensemble of a text-only finetuned BERT regressor and a tabular-only Catboost model. This was after confirming the strong performance of Catboost in tabular-only regression in Chapter 3, and the competitive accuracy of finetuned BERT approaches for text-only regression in Chapter 4. We used three datasets to compare this simple ensemble to the main approaches proposed in prior literature. It was found that the simple ensemble performed comparably to a previously published DL fused representation model for two datasets, but for one dataset the fused representation model had distinctly lower root mean squared error (RMSE). This suggests that a fused representation model can more consistently extract joint text-tabular signal than a simple ensemble, making

it a promising approach for text-tabular regression. However, the comparable performance of the simple ensemble suggests that future studies would benefit from inclusion of such an ensemble. Furthermore, we found that concatenation of BoW with tabular features, and Catboost modelling, consistently achieved worse accuracy than the best text-only or tabular-only model; suggesting that this ‘traditional’ approach was weaker on our datasets. Finally, we found that the pretraining objective of the previously proposed LANISTR model (Ebrahimi et al., 2024) required more work to improve the training stability.

6.1 Future Work

The contributions of this thesis suggest some interesting directions for future research. In particular, the interaction of missing value handling and text-tabular modelling could be further investigated for not-MCAR data, e.g. using real insurance datasets. Furthermore, the importance of missing value handling, along with the weaker performance of DL tabular methods on not-MCAR data, suggests that development of missing value handling for DL is an interesting avenue of research.

The work in Chapter 4 could also be expanded, given sufficient computational resources. This could include more replicates to cover more of the pipeline space and removing step choice dependencies (e.g. pretrain BERT using other tokenizers) to further investigate the importance of upstream steps and their interactions.

Outside of direct extension of specific chapters, interesting future research may include the interpretability of text-tabular models, e.g. answering questions such as ‘*given a description of expensive rooms, does the predicted property price correspondingly increase per room? Can we attribute words or phrases to this increase?*’. Also, given the freedom of natural language data to express extreme events and uncertainty (in contrast to tabular data) it would be interesting to study the potential of text-tabular modelling for uncertainty quantification and improving accuracy for rare extreme events. Finally, just as natural language data is widely used, image data is also very prevalent. Hence, a future direction could be systematic study of joint text-

tabular-image regression.

Bibliography

- Akiba, Takuya et al. (2019). “Optuna: A next-generation hyperparameter optimization framework”. In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.
- Amazon (2024). *Autogluon documentation*. URL: <https://auto.gluon.ai/stable/index.html> (visited on 01/2025).
- Ao, Junyi et al. (2022). “SpeechT5: Unified-modal encoder-decoder pre-training for spoken language processing”. In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*. Vol. 1, pp. 5723–5738.
- Arik, Serkan and Tomas Pfister (2021). “TabNet: Attentive interpretable tabular learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.8, pp. 6679–6687.
- Arnaud, Émilien et al. (2020). “Deep learning to predict hospitalization at triage: integration of structured data and unstructured text”. In: *Proceedings of the 2020 IEEE International Conference on Big Data*, pp. 4836–4841.
- Baktha, Kiran and BK Tripathy (2017). “Investigation of recurrent neural networks in the field of sentiment analysis”. In: *Proceedings of the 2017 International Conference on Communication and Signal Processing*, pp. 2047–2050.
- Bird, Steven (2006). “NLTK: The natural language toolkit”. In: *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*, pp. 69–72.
- Bishop, Christopher M. and Hugh Bishop (2024). *Deep Learning: Foundations and Concepts*. Springer International Publishing. ISBN: 9783031454684.
- Bitvai, Zsolt and Trevor Cohn (2015). “Non-linear text regression with a deep convolutional neural network”. In: *Proceedings of the 53rd Annual Meeting of the Association for Compu-*

- tational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pp. 180–185.
- Blier-Wong, Christopher et al. (2021). “Machine learning in P&C insurance: A review for pricing and reserving”. In: *Risks* 9.1, p. 4.
- Borisov, Vadim et al. (2022). “Deep neural networks and tabular data: A survey”. In: *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21.
- Bornhuetter, Ronald L and Ronald E Ferguson (1972). “The actuary and IBNR”. In: *Casualty Actuarial Society* 59.112, pp. 181–195.
- Breiman, Leo et al. (1984). *Classification and Regression Trees*. Wadsworth and Brooks/Cole. ISBN: 9780412048418.
- Caron, Matthew and Oliver Müller (2020). “Hardening soft information: A Transformer-based approach to forecasting stock return volatility”. In: *Proceedings of the 2020 IEEE International Conference on Big Data*.
- Catboost (2024a). *Codebase tutorial on Catboost encoding*. URL: https://github.com/catboost/catboost/blob/master/catboost/tutorials/categorical_features/categorical_features_parameters.ipynb (visited on 04/2024).
- (2024b). *Transforming categorical features to numerical features*. URL: https://catboost.ai/en/docs/concepts/algorithm-main-stages_cat-to-numeric (visited on 04/2024).
- Chen, Tianqi and Carlos Guestrin (2016). “XGBoost: A scalable tree boosting system”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.
- Clark, Kevin et al. (2020). “ELECTRA: Pre-training text encoders as discriminators rather than generators”. In: *International Conference on Learning Representations*.
- Dauphin, Yann N. et al. (2017). “Language modeling with gated convolutional networks”. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. PMLR, pp. 933–941.
- De Felice, Massimo and Franco Moriconi (2019). “Claim watching and individual claims reserving using classification and regression trees”. In: *Risks* 7.4, p. 102.

- Delong, Lukasz, Mathias Lindholm, and Mario V. Wüthrich (2022). “Collective reserving using individual claims data”. In: *Scandinavian Actuarial Journal* 2022.1, pp. 1–28.
- Delong, Lukasz and Mario V. Wüthrich (2020). “Neural networks for the joint development of individual payments and claim incurred”. In: *Risks* 8.2, p. 33.
- Dereli, Neşat and Murat Saraclar (2019). “Convolutional neural networks for financial text regression”. In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pp. 331–337.
- Devlin, Jacob et al. (2019). “BERT: Pre-training of deep bidirectional transformers for language understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*. Vol. 1, pp. 4171–4186.
- Dosovitskiy, Alexey et al. (2021). “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *Proceedings of the 2021 International Conference on Learning Representations*.
- Duval, Francis and Mathieu Pigeon (2019). “Individual loss reserving using a gradient boosting-based approach”. In: *Risks* 7.3, p. 79.
- Ebrahimi, Sayna et al. (2024). “LANISTR: Multimodal learning from structured and unstructured data”. In: *arXiv preprint arXiv:2305.16556*.
- Fauzan, Muhammad Arief and Hendri Murfi (2018). “The accuracy of XGBoost for insurance claim prediction”. In: *International Journal of Advances in Soft Computing and its Applications* 10.2, pp. 159–171.
- Friedman, Jerome (2001). “Greedy function approximation: A gradient boosting machine”. In: *The Annals of Statistics* 29.5.
- Gorishniy, Yury et al. (2021). “Revisiting deep learning models for tabular data”. In: pp. 18932–18943.
- Grinsztajn, Léo, Edouard Oyallon, and Gaël Varoquaux (2022). “Why do tree-based models still outperform deep learning on typical tabular data?” In: *Proceedings of the 2022 Advances in Neural Information Processing Systems*, pp. 507–520.

- Gu, Ken and Akshay Budhkar (2021). “A package for learning on tabular and text data with transformers”. In: *Proceedings of the Third Workshop on Multimodal Artificial Intelligence*, pp. 69–73.
- Guo, Cheng and Felix Berkhahn (2016). “Entity embeddings of categorical variables”. In: *arXiv preprint arXiv:1604.06737*.
- Hanafy, Mohamed and Ruixing Ming (2021). “Machine learning approaches for auto insurance big data”. In: *Risks* 9.2.
- He, Kaiming et al. (2016a). “Deep residual learning for image recognition”. In: *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778.
- (2016b). “Identity mappings in deep residual networks”. In: *Proceedings of the 2016 European Conference on Computer Vision*, pp. 630–645.
- Hearst, Marti A. et al. (1998). “Support vector machines”. In: *IEEE Intelligent Systems and their applications* 13.4, pp. 18–28.
- Herring, Amy H., Joseph G. Ibrahim, and Stuart R. Lipsitz (2004). “Non-ignorable missing covariate data in survival analysis: A case-study of an international breast cancer study group trial”. In: *Journal of the Royal Statistical Society Series C: Applied Statistics* 53.2, pp. 293–310.
- Hinton, Geoffrey E et al. (2012). “Improving neural networks by preventing co-adaptation of feature detectors”. In: *arXiv preprint arXiv:1207.0580*.
- Holtz, Niklas and Jorge Marx Gomez (2023). “Multimodal transformer for risk dcassification: analyzing the impact of different data modalities”. In: *Natural Language Processing and Machine Learning*, pp. 29–39.
- Honnibal, Matthew et al. (2020). *spaCy: Industrial-strength natural language processing in Python*. URL: <https://spacy.io/>.
- Howard, Jeremy and Sebastian Ruder (2018). “Universal language model fine-tuning for text classification”. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. Vol. 1, pp. 328–339.
- Huang, Xin et al. (2020). “TabTransformer: Tabular data modeling using contextual embeddings”. In: *arXiv preprint arXiv:2012.06678*.

- Ioffe, Sergey and Christian Szegedy (2015). “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *Proceedings of the 2015 International Conference on Machine Learning*, pp. 448–456.
- JAX (2025). URL: <https://jax.readthedocs.io/en/latest/> (visited on 02/03/2025).
- Jin, Liang et al. (2021). “A comparative study of evaluating missing value imputation methods in label-free proteomics”. In: *Scientific Reports* 11.1, p. 1760.
- Joshi, Mahesh et al. (2010). “Movie reviews and revenues: An experiment in text regression”. In: *Proceedings of the 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 293–296.
- Kadra, Arlind et al. (2021). “Well-tuned simple nets excel on tabular datasets”. In: *Proceedings of the 2021 Advances in Neural Information Processing Systems*, pp. 23928–23941.
- Keras Team (2024). *Keras Documentation: Transfer Learning & Fine-Tuning*. URL: https://keras.io/guides/transfer_learning/ (visited on 09/2024).
- Kingma, Diederik P and Jimmy Ba (2014). “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980*.
- Kudo, Taku and John Richardson (2018). “SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing”. In: *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 66–71.
- Kuhn, Max, Kjell Johnson, et al. (2013). *Applied predictive modeling*. Springer. ISBN: 9781461468493.
- Kuo, Kevin (2020). “Individual claims forecasting with Bayesian mixture density networks”. In: *Casualty Actuarial Society*.
- Larionov, Alexander, Niall M. Adams, and Kevin N. Webster (Sept. 3, 2024). “Investigating the impact of missing value handling on boosted trees and deep learning for tabular data: A claim reserving case study”. In: *Transactions on Machine Learning Research*.
- Lee, Gee Y, Scott Manski, and Tapabrata Maiti (2020). “Actuarial applications of word embedding models”. In: *ASTIN Bulletin* 50.1, pp. 1–24.
- Little, Roderick JA and Donald B Rubin (2019). *Statistical analysis with missing data, 3rd Edition*. John Wiley & Sons. ISBN: 9781119482260.

- Liu, Yaofang et al. (2024). “EvalCrafter: Benchmarking and evaluating large video generation models”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 22139–22149.
- Lodhi, Huma et al. (2002). “Text classification using string kernels”. In: *Journal of Machine Learning Research* 2.Feb, pp. 419–444.
- Lopez, Olivier, Xavier Milhau, and Pierre-E Thérond (2019). “A tree-based algorithm adapted to microlevel reserving and long development claims”. In: *ASTIN Bulletin: The Journal of the International Actuarial Association* 49.3, pp. 741–762.
- Loshchilov, Ilya and Frank Hutter (2017). “SGDR: Stochastic gradient descent with warm restarts”. In: *Proceedings of the 2017 International Conference on Learning Representations*.
- Manning, Christopher D et al. (2014). “The Stanford CoreNLP natural language processing toolkit”. In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics*.
- McCullagh, Peter (2019). *Generalized linear models*. Routledge.
- McDonnell, Kevin et al. (2023). “Deep learning in insurance: Accuracy and model interpretability using TabNet”. In: *Expert Systems with Applications* 217, p. 119543.
- McElfresh, Duncan et al. (2023). “When do neural nets outperform boosted trees on tabular data?” In: *Proceedings of the 2023 Advances in Neural Information Processing Systems*, pp. 76336–76369.
- Micci-Barreca, Daniele (2001). “A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems”. In: *ACM SIGKDD Explorations Newsletter* 3.1, pp. 27–32.
- Mikolov, Tomas et al. (2013). “Distributed representations of words and phrases and their compositionality”. In: *Proceedings of the 2013 Advances in Neural Information Processing Systems*.
- Minaee, Shervin et al. (2021). “Deep learning-based text classification: A comprehensive review”. In: *ACM Computing Surveys* 54.3.
- Murphy, Kevin P. (2022). *Probabilistic machine learning: An introduction*. MIT Press. ISBN: 9780262046824.

- Nisbet, Robert, John Elder, and Gary D Miner (2009). *Handbook of statistical analysis and data mining applications*. Academic press. ISBN: 9780123747655.
- Olivieri, Annamaria and Ermanno Pitacco (2015). *Introduction to insurance mathematics: technical and financial features of risk transfers*. Springer. ISBN: 9783642160295.
- ONS (2024). *Postal geographies - Office for national statistics*. URL: <https://www.ons.gov.uk/methodology/geography/ukgeographies/postalgeography> (visited on 02/03/2024).
- Pennington, Jeffrey, Richard Socher, and Christopher D Manning (2014). “GloVe: Global vectors for word representation”. In: *Proceedings of the 2014 conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543.
- Popov, Sergei, Stanislav Morozov, and Artem Babenko (2020). “Neural oblivious decision ensembles for deep learning on tabular data”. In: *Proceedings of the 2020 International Conference on Learning Representations*.
- Porter, Martin F (2001). *Snowball: A language for stemming algorithms*. URL: <https://snowballstem.org/>.
- Prokhorenkova, Liudmila et al. (2018). “CatBoost: unbiased boosting with categorical features”. In: *Proceedings of the 2018 Advances in Neural Information Processing Systems*.
- PyTorch (2025). URL: <https://pytorch.org/> (visited on 02/03/2025).
- Radford, Alec, Jong Wook Kim, Chris Hallacy, et al. (2021). “Learning transferable visual models from natural language supervision”. In: *Proceedings of the 2021 International Conference on Machine Learning*, pp. 8748–8763.
- Radford, Alec, Jong Wook Kim, Tao Xu, et al. (2023). “Robust speech recognition via large-scale weak supervision”. In: *Proceedings of the 40th International Conference on Machine Learning*.
- Radford, Alec, Jeffrey Wu, et al. (2019). *Language models are unsupervised multitask learners*. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.
- Ramesh, Aditya et al. (2021). “Zero-shot text-to-image generation”. In: *Proceedings of the 2021 International Conference on Machine Learning* 139, pp. 8821–8831.

- Ren, Pengzhen et al. (2021). “A comprehensive survey of neural architecture search: challenges and solutions”. In: *ACM Computing Surveys* 54.4.
- Rombach, Robin et al. (2022). “High-resolution image synthesis with latent diffusion models”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.
- Salton, Gerard (1991). “Developments in automatic text retrieval”. In: *Science*, pp. 974–980.
- Sardelich, Marcelo and Suresh Manandhar (2018). “Multimodal deep learning for short-term stock volatility prediction”. In: *arXiv preprint arXiv:1812.10479*.
- Seinen, Tom M et al. (2022). “Use of unstructured text in prognostic clinical prediction models: A systematic review”. In: *Journal of the American Medical Informatics Association* 29.7, pp. 1292–1302.
- Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). “Neural machine translation of rare words with subword units”. In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715–1725.
- Shavitt, Ira and Eran Segal (2018). “Regularization learning networks: Deep learning for tabular Datasets”. In: *Proceedings of the 2018 Advances in Neural Information Processing Systems*.
- Shi, Xingjian et al. (2021). “Benchmarking multimodal AutoML for tabular data with text fields”. In: *Proceedings of the 2021 Advances in Neural Information Processing System: Datasets and Benchmarks Track*.
- Simonyan, Karen and Andrew Zisserman (2015). “Very deep convolutional networks for large-scale image recognition”. In: *Proceedings of the 2015 International Conference on Learning Representations*.
- So, Banghee, Jean-Philippe Boucher, and Emiliano A. Valdez (2021). “Synthetic dataset generation of driver telematics”. In: *Risks* 9.4.
- Soguero-Ruíz, Cristina et al. (2016). “Support vector feature selection for early detection of anastomosis leakage from bag-of-Words in electronic health records”. In: *IEEE Journal of Biomedical and Health Informatics* 20, pp. 1404–1415.
- Somepalli, Gowthami et al. (2021). “SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training”. In: *arXiv preprint arXiv:2106.01342*.

- Statista (2024). *Motor Vehicle Insurance - Global: Statista market forecast*. URL: <https://www.statista.com/outlook/fmo/insurances/non-life-insurances/motor-vehicle-insurance/worldwide> (visited on 02/2024).
- Suvon, Mohammad N. I. et al. (2022). “Multimodal learning for predicting mortality in patients with pulmonary arterial hypertension”. In: *2022 IEEE International Conference on Bioinformatics and Biomedicine*, pp. 2704–2710. ISBN: 9781665468190.
- Tao, Andrew, Karan Sapra, and Bryan Catanzaro (2020). “Hierarchical multi-scale attention for semantic segmentation”. In: *arXiv preprint arXiv:2005.10821*.
- Taylor, Greg, Gráinne McGuire, and James Sullivan (2008). “Individual claim loss reserving conditioned by case estimates”. In: *Annals of Actuarial Science* 3.1-2, pp. 215–256.
- TensorFlow (2025). URL: <https://www.tensorflow.org/> (visited on 02/2025).
- Tieleman, Tijmen and Geoffrey Hinton (2012). “Lecture 6.5-RMSprop: Divide the gradient by a running average of its recent magnitude”. In: *Coursera: Neural networks for machine learning* 4.2, pp. 26–31.
- Touvron, Hugo et al. (2023). “LLaMA: Open and efficient foundation language models”. In: *arXiv preprint arXiv:2302.13971*.
- Tsai, Ming-Feng and Chuan-Ju Wang (2014). “Financial keyword expansion via continuous word vector representations”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1453–1458.
- Tschalzev, Andrej et al. (2024). “A data-centric perspective on evaluating machine learning models for tabular data”. In: *Proceedings of the 2024 Advances in Neural Information Processing Systems: Datasets and Benchmarks Track*.
- Universal Dependencies (2024). *Universal POS Tags*. URL: <https://universaldependencies.org/u/pos/> (visited on 02/2025).
- Van Ness, Mike et al. (2023). “The missing indicator method from low to high dimensions”. In: *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 5004–5015.
- Vaswani, Ashish et al. (2017). “Attention is all you need”. In: *Proceedings of the 2017 Advances in Neural Information Processing Systems*.

- Werner, Geoff and Claudine Modlin (2010). “Basic ratemaking”. In: *Casualty Actuarial Society* 4, pp. 1–320.
- Wuthrich, Mario V. (2018). “Machine learning in individual claims reserving”. In: *Scandinavian Actuarial Journal* 2018.6, pp. 465–480.
- Xue, Linting et al. (2022). “ByT5: Towards a token-free future with pre-trained byte-to-byte models”. In: *Transactions of the Association for Computational Linguistics* 10, pp. 291–306.
- Yeh, Hsiang Yuan, Yu Ching Yeh, and Da Bai Shen (2020). “Word vector models approach to text regression of financial risk prediction”. In: *Symmetry* 12.1.
- Yin, Shukang et al. (2023). “A survey on multimodal large language models”. In: *arXiv preprint arXiv:2306.13549*.
- Yoon, Jinsung, James Jordon, and Mihaela van der Schaar (2018). “GAIN: Missing data imputation using generative adversarial nets”. In: vol. 80. PMLR, pp. 5689–5698.
- Zhang, Dongdong et al. (2020). “Combining structured and unstructured data for predictive models: A deep learning approach”. In: *BMC Medical Informatics and Decision Making* 20.1, p. 280.
- Ziegler, Daniel M et al. (2019). “Fine-tuning language models from human preferences”. In: *arXiv preprint arXiv:1909.08593*.

Appendix A

Appendices

A.1 Missing values in tabular modelling

A.1.1 Data properties

This appendix presents further properties of the data analyzed in Chapter 3.

Missing value concentration

Tables A.1 and A.2 characterize the concentration of missing values in our insurance dataset.

Fraction f missing	Percentage of columns with given fraction missing
$f = 0$	45
$0 < f \leq 0.2$	17
$0.2 < f \leq 0.4$	13
$0.4 < f \leq 0.6$	11
$0.6 < f \leq 0.8$	7
$0.8 < f \leq 1.0$	6

Table A.1: Missingness distribution by column, to nearest %

Fraction f missing	Percentage of rows with given fraction missing
$0 < f \leq 0.2$	46
$0.2 < f \leq 0.4$	19
$0.4 < f \leq 0.6$	12
$0.6 < f \leq 0.8$	11
$0.8 < f \leq 1.0$	11

Table A.2: Missingness distribution by row, to nearest %

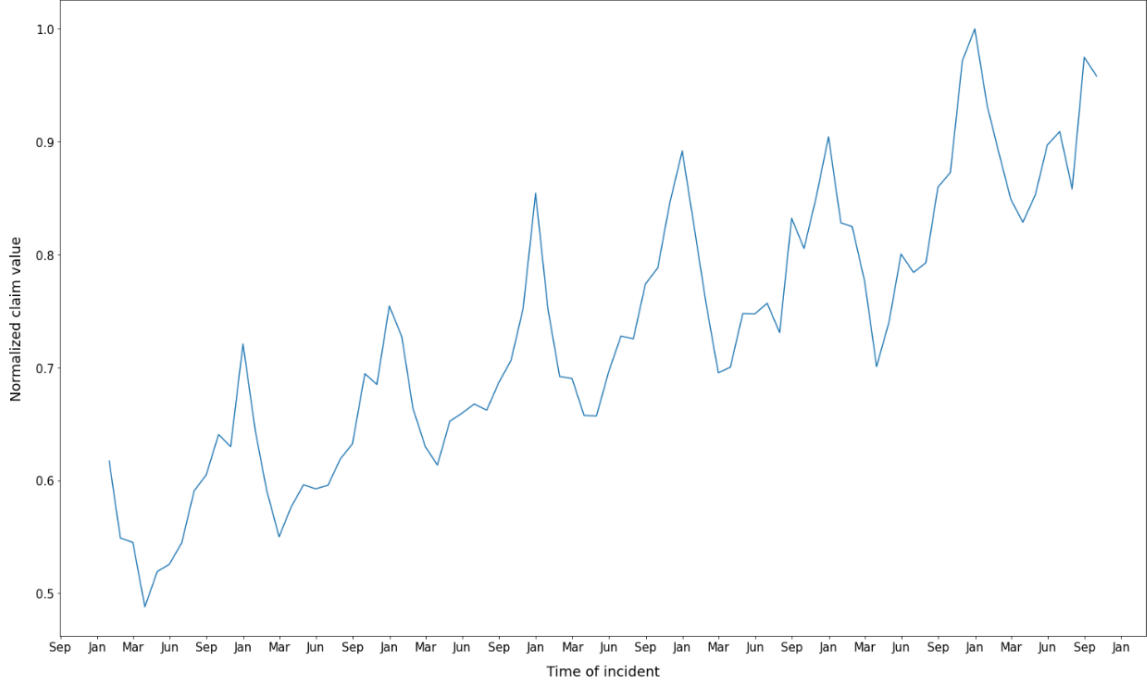


Figure A.1: Time series plot of mean standardized claim values against anonymized time. Claim values were aggregated on a month. Claim values and years of data are anonymized for confidentiality.

Time-varying properties

Figure A.1 depicts the mean claim value per month, with claim value and year of incident anonymized for commercial confidentiality. The figure shows that mean claim value is clearly dependent on the time of the claim – exhibiting both seasonality and trend. This has implications for both the evaluation of model accuracy and encoding of time data.

When evaluating the accuracy of the machine learning (ML) models, described in Section 3.3, the time series nature of the data requires the definition of the test set data to be in the future relative to all training and validation data. This is to prevent bias in the estimation of

performance, a form of data leakage (Nisbet, Elder, and Miner, 2009, p. 742).

When considering how to encode the time variables, we aim to encode in such a way as to make it easier to fit seasonality and trend. To explicitly encode cyclic timestamp properties, we encode month and day-of-month variables separately. Having cyclic values in the input data, like month and day-of-month, intuitively makes it easier to fit conditional on cyclic seasons. In order to fit overall trend, we chose to also encode timestamps as a monotonic value, enabling the model to order claims in time from a single numeric value. We did this at two resolutions: i) year and ii) seconds since the epoch, also known as *Unix time*. The goal of encoding in years is to allow a fit to large scale trends such as inflation. The goal of encoding in seconds is to enable more granular ordering in time. Therefore, overall we encode a single timestamp feature into 4 variables: year, month, date and seconds since the epoch.

High-cardinality categorical variables

The dataset contains some high-cardinality categorical variables: insuree car make with hundreds of categories, insuree car model with thousands of categories and *first half* of insuree postcode with thousands of categories. The typical one-hot encoding (Murphy, 2022, p. 23) of these features would dramatically blow up the dataset size in number of features. Therefore, we applied numeric encoding strategies such as impact encoding (Micci-Barreca, 2001), also known as *target mean encoding*, and Catboost’s novel categorical encoding, described in Section 2.4.

As the UK has 1.7 million postcodes (ONS, 2024), attempting to treat full postcodes as a categorical variable would give rise to at least hundreds of thousands of categories. Even target mean encoding such a high cardinality categorical could exhibit high bias and high variance due to the low number of samples per category; following the intuition outlined in Prokhorenkova et al., 2018. Only modelling the first half of postcodes was used to decrease the cardinality of the variable to thousands of categories to potentially reduces the bias and variance of the mean encoding.

A.1.2 Training speed

Using 12 Intel Xeon 6136 CPUs and approximately 60GB of RAM: Catboost trains on our dataset in the order of 3 minutes per training run and the ResNet multi layer perceptron (MLP) architecture trains in the order of 45-120 minutes per training run. TabNet trains in the order of 300 minutes per training run. Although both Catboost and ResNet exhibited training times low enough to be retrained many times a day to keep up with changes in underlying distribution; the significantly faster training time of Catboost enables greater experimentation. This is potentially relevant for other applications as it has been shown that the difference in modelling strategies without hyperparameter tuning is just as great as the difference within a model class with hyperparameter tuning when training on tabular data (Kadra et al., 2021).

A.1.3 Optuna HPO Results

Table A.3 shows a table analogous to Table 3.1 comparing results obtained using Optuna for the HPO instead of grid search. It can be seen that there is no substantive difference to the conclusions in Table 3.1 of the work from swapping to Optuna HPO; as core points are made in comparison of our ResNet MLP and Catboost (which did not undergo HPO).

Table A.3: Results when using Optuna HPO

Model	Drop	LT Min Impute	Mean Impute	Binarize
Our ResNet MLP + Grid Search (ME)	2046 (± 1.2)	1872 (± 4.5)	7855 (± 1600)	2288 (± 1.8)
Our ResNet MLP + Optuna HPO (ME)	2039 (± 5.1)	1887 (± 6.5)	7673 (± 1600)	2283 (± 1.3)

A.1.4 Impact of imputation scheme on HPO

This appendix presents preliminary experimental data used to justify the process of performing HPO independently of imputation scheme; as described in the beginning of Section 3.4. The closeness in parameter magnitude value and the robustness of HPO performance to varying missing value scheme suggested it was acceptable to reduce the computational complexity of

experiments by performing HPO under one imputation scheme and evaluating on all. The **LT Min Impute** scheme was chosen as initial results suggested it would be the scheme with the best performance; and as such chosen in an attempt to give a level playing field for best performance across models.

Grid search based

The optimum hyperparameters obtained using grid search with **LT Min Impute** and **Drop** for our ResNet from preliminary analysis are presented in Table A.4. Due to similarity in the optimal hyperparameters obtained it was concluded from this preliminary analysis that hyperparameter optimization (HPO) could be performed independent of imputation scheme; keeping overall computational cost down.

Table A.4: Best performing hyperparameters under different imputation schemes; following a grid search HPO procedure.

Hyperparameter	LT Min Impute	Drop
Regularization coefficient	0.01	0.01
Initial learning rate	0.01	0.01
Learning rate schedule	CosineDecay	CosineDecay
Optimizer	RMSPProp	RMSPProp
Weight Decay	0.01	0.0001

Optuna based

The optimum hyperparameters obtained using Optuna with **LT Min Impute** and **Drop** for our ResNet are presented in Table A.5. With corresponding final outcomes presented in Table A.6. It can be seen from A.6 that although tuning with **Drop** gives different hyperparameters; and gives our ResNet more stable results for **Mean Impute**; the relative rankings in the final Table 3.1 would be unaffected.

Table A.5: Best performing hyperparameters under different imputation schemes; following an Optuna HPO procedure.

Hyperparameter	LT Min Impute	Drop
Regularization coefficient	0.017	0.68
Initial learning rate	0.009	0.006
Learning rate schedule	CosineDecay	ExponentialDecay
Optimizer	RMSProp	RMSProp
Weight Decay	0.0002	0.0002

Table A.6: Effect of imputation scheme on HPO results when using Optuna HPO

Model	Drop	LT Min Impute	Mean Impute	Binarize
Our ResNet MLP + LT Min Impute Optuna HPO	2039 (± 5.1)	1887 (± 6.5)	7673 (± 1600)	2283 (± 1.3)
Our ResNet MLP + Drop Optuna HPO	2047 (± 4.3)	1885 (± 4.6)	4548 (± 680)	2292 (± 2.0)

A.1.5 TabZilla replication

This appendix details the results obtained from comparing **LT Min Impute**, **Mean Impute** and **Drop** on some extra datasets and algorithms from the TabZilla Benchmark (McElfresh et al., 2023). Source code can be found at <https://github.com/paper3193/tabzilla>.

30% of numeric values were randomly replaced with missing values from each dataset and then imputed using the imputation schemes studied. Then the corresponding OpenML task was performed using Catboost, XGBoost, FTTransformer (Gorishniy et al., 2021) and the RTDL ResNet. We present the results by dataset in Table A.7.

From Table A.7 we can see that the overall impact of imputation schemes is not large and in some cases causes no performance difference between certain imputation schemes. However, the changes in accuracy rankings of the datasets under different imputation schemes demonstrates that in principle it is possible for the rankings, and therefore results such as those in McElfresh et al., 2023, to be influenced by imputation scheme. We note that, by design, Table A.7 shows results from injected MCAR missing values and as such we do not attempt to interpret the absolute performance of any imputation scheme as MCAR data is not the focus of our work.

In summary, the TabZilla replication with different imputation schemes demonstrates in principle that handling missing values could be important; but the MCAR nature of the injected

missing values and low signal value of the numerics in the chosen datasets produces a less pronounced effect than our work with non MCAR claim reserving. This highlights the importance of analyses using real world large datasets with not-MCAR missing value structure in comparing GBDTs and DL models.

Table A.7: Relative accuracy and ranking per dataset per algorithm using the TabZilla repository under different imputation schemes. We report TabZilla mean 10-fold cross-validation test accuracy; as extracted from the tuned aggregated results output.

Dataset	Model	Mean Impute Accuracy Rank		Drop Accuracy Rank		LT Min Impute Accuracy Rank	
ada_agnostic	CatBoost	0.857	1	0.854	2	0.857	1
ada_agnostic	FTTransformer	0.844	4	0.845	3	0.846	3
ada_agnostic	RTDL ResNet	0.846	3	0.842	4	0.841	4
ada_agnostic	XGBoost	0.855	2	0.855	1	0.855	2
LED-display	CatBoost	0.716	2	0.714	3	0.716	3
LED-display	FTTransformer	0.708	4	0.728	1	0.728	1
LED-display	RTDL ResNet	0.724	1	0.724	2	0.722	2
LED-display	XGBoost	0.710	3	0.706	4	0.710	4

A.1.6 Combining missing value handling strategies

Table A.8 shows the performance obtained when combining both the **LT Min Impute** and **Binarize** approaches, turning each D dimensional feature vector into a $2D$ dimensional feature vector. For each original feature we generate two features: one following **LT Min Impute** and another that is a binary indicator of whether the data is (**Binarize**). This could be interpreted as providing both features and ‘missingness masks’ of present features to the model. Table A.8 also includes results for **LT Min Impute** and **Binarize** from Table 3.1 for reference.

Although it is a natural question to investigate combinations of compatible methods, we find that for our dataset there is no noticeable improvement for combining **LT Min Impute** and **Binarize** in a missing masks approach. This could be due to the relatively high dimensionality of the data (D in order of hundreds) and as such it is not trivial for the learning algorithms to correctly learn which mask corresponds to which feature, a combinatorial problem. We note, if a mask is forced to correspond to a feature x_i through a function, f , of the form $f(x_i) = \theta \mathbb{1}_{\text{missing}} + x_i(1 - \mathbb{1}_{\text{missing}})$ we recover a typical imputation scheme.

Table A.8: $\mathcal{D}^{\text{test}}$ RMSE mean and standard error, se , over cross validation partitions. **Lower RMSE is better**. RMSE is to the nearest integer, se is to 2 significant figures. (CE) indicates Catboost encoding was used for categorical encoding. ‡ corresponds to an Optuna HPO scheme.

Model	LT Min	Impute	Binarize	LT Min	Impute + Binarize
Catboost (CE)	1452	(± 6.3)	2302 (± 0.18)	1449	(± 1.6)
Catboost	1574	(± 2.1)	2268 (± 0.28)	1575	(± 2.8)
Our ResNet	1872	(± 4.5)	2288 (± 1.8)	1883	(± 8.9)
RTDL ResNet‡	2007	(± 20)	2282 (± 2.4)	1994	(± 18.7)
Mean prediction	2764	(± 0.020)	2764 (± 0.020)	2764	(± 0.020)

A.2 NLP pipelines for textual regression

A.2.1 Pipeline component coverage

Table A.10 and Table A.9 show the coverage of randomly selected choices across preprocessor, tokenizer, featurizer and model for the `obl` and `jcp` datasets respectively. That is, the tables show how much each pipeline component choice was used for a given dataset across all simulations in this work, showing a good number of replicates for each approach.

no-punctuation	196
stem	164
no-stopwords-nltk	120
empty	168
lowercase	220
lemmatize	156
no-stopwords-spacy	140

(a) preprocessor coverage.

tf-idf	116
bow-binary	112
bert-cls	104
bow-count	100
bert-mean	96
glove-mean	92

(c) Featurizer coverage.

bert	260
stanford	120
whitespace	116
bpe	72
treebank	52

(b) Tokenizer coverage.

catboost	140
deep-linear	124
xgboost	112
mlp	108
resnet	80
resnet-ft	24
mlp-ft	16
deep-linear-ft	16

(d) Model coverage. ‘-ft’ corresponds to model with featurizer finetuning.

Table A.9: Component coverage across random pipelines for the `obl` dataset.

no-stopwords-nltk	100		
empty	168		
lemmatize	136	bert	248
lowercase	196	whitespace	116
no-punctuation	168	stanford	92
stem	136	bpe	56
no-stopwords-spacy	104	treebank	48
(a) preprocessor coverage.		(b) Tokenizer coverage.	
bow-binary	116	mlp	124
tf-idf	104	deep-linear	112
bert-cls	92	catboost	104
bert-mean	84	xgboost	96
bow-count	84	resnet	72
glove-mean	80	deep-linear-ft	20
		resnet-ft	20
(c) Featurizer coverage.		mlp-ft	12
		(d) Model coverage. ‘-ft’ corresponds to model with featurizer finetuning.	

Table A.10: Component coverage across random pipelines for the jcp dataset.

A.2.2 Dataset properties

Tables A.11 and A.12 show samples of the data stratified by the label value for obl and jcp respectively. Text samples are truncated to 200 characters where they exceed 200 characters.

Table A.11: Stratified samples from `jcp`. Price column includes the stratified percentiles that the sample was drawn from, i.e. (0-20) indicates between the 0 and 20th percentile.

Text	Price
This fast-absorbing blend of oils moisturizes and nourishes the hair and scalp with essential nutrients for silky, smooth, healthy-looking hair. includes tea tree oil, peppermint oil, silk and chamomi...	\$15 (0 - 20)
This basketweave-style tie from Van Heusen® catches the light in all your favorite colors. 3½" wide silk dry clean imported	\$30 (20 - 40)
With a classic peasant shape, our fit-and-flare dress embraces an iconic silhouette for a bold bohemian look. 34" length from shoulder knit polyester/spandex washable made in America	\$33 (40 - 60)
Keep him comfortable all day in our modern boys running shoes. technology flexible and lightweight Phylon promotes a natural stride construction mesh/synthetic upper synthetic/rubber sole de...	\$47 (60 - 80)
Round and round, these earrings sparkle brilliantly with 1/3 ct. t.w. diamonds in a timeless 10K white gold setting. Metal: Rhodium-plated 10K white gold Stones: 1/3 ct. t.w. round diamonds Color: I-J Cla...	\$906 (80 - 100)

Table A.12: Stratified samples from `obl`. Price column includes the stratified percentiles that the sample was drawn from, i.e. (0-20) indicates between the 0 and 20th percentile.

Text	Price
Designed by the famous S&S studio in 1969, built by Winfield & Partners, it was produced in England and Australia in 400 units. Exceptional construction and level of finish, the S&S 34 is a small eleg...	\$25,000 (0 - 20)
Well built ship. A timeless classic with many innovations and kept up to date. In 2012, the engine was overhauled, new batteries, excellent navigation equipment and a beautiful sailing wardrobe. A...	\$36,000 (20 - 40)
Bow thruster, shore power, boiler, mast lowering installation, etc. Surprisingly spacious Blom Lemsteraak. General Hull shape: Round bottom Controls: Helmet wood Windows: In brass and in teak rebate...	\$79,900 (40 - 60)
Make: Bénétteau Model: Beneteau 58 2010 Beneteau 58 Year: 2010 Show more	\$297,000 (60 - 80)
THE CNB 76 FIFTY FIFTY is a sturdy, high-performance blue cruiser, featuring an aluminium hull with teak deck and Sparcraft aluminium mast and boom. The vessel has been carefully maintained and upgrad...	\$690,000 (80 - 100)

A.3 Evaluating ensembling for text and tabular modelling

A.3.1 Results including LANISTR

Figure A.2 shows normalized RMSE values across all datasets, including LANISTR (Ebrahimi et al., 2024). Note, on `jcp`, LANISTR noticeably underperforms even the mean prediction baseline. Upon further investigation we found that this dataset specifically had some numeric features that cause rapid gradient explosion in the pretraining phase. The last ‘stable’ parameter values were already sufficiently large that finetuning also rapidly exhibited exploding gradients. Overall, this caused the model to learn a constant prediction lower than the mean, thus underperforming it.

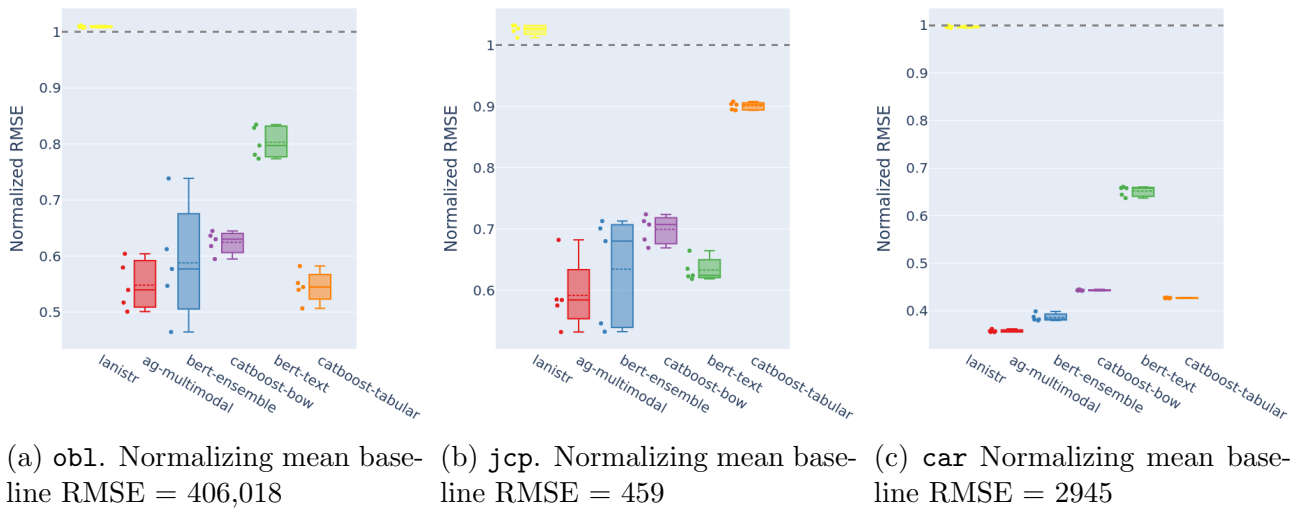


Figure A.2: Boxplot of normalized RMSE for each Monte Carlo cross validation sample, across datasets. RMSE is normalized by division with the RMSE obtained from a constant training data mean prediction. Dashed line in boxplot represents mean. Raw RMSE samples are scattered next to boxplots with jitter for visibility.