NEW LONGITUDINAL CLASSIFICATION METHODS WITH AUTOMATED DETECTION OF MONOTONICITY CONSTRAINTS

A THESIS SUBMITTED TO

THE UNIVERSITY OF KENT

IN THE SUBJECT OF COMPUTER SCIENCE

FOR THE DEGREE

OF PHD.

By

Sergey Ovchinnik

September 2023

Abstract

Longitudinal datasets contain repeated measurements of the same attributes measured at different points in time. They can be used to discover interesting knowledge based on time-based trends and create prediction models that can predict future values of target attributes, like class attributes in the case of the classification task of data mining (or machine learning).

Monotonicity is a special type of relation between attributes in which their values have either a strongly positive or a strongly negative relationship. These relations can occur in real-world data and can be used to enhance the predictive performance and the acceptability of prediction models by users.

In recent years, there has been an increasing interest in using longitudinal data and monotonicity relations in the field of classification, resulting in the emergence of the fields of Longitudinal Classification and Monotonic classification.

While there has been a lot of development in each of these fields separately, there have not been any studies that united these two fields. The aim of this study is to develop new approaches to longitudinal and monotonic classification and develop a unified approach for using monotonicity relations found in longitudinal datasets in order to construct highly accurate and useful classification models.

As a result of this study, several contributions were made to both fields, as follows. First, a novel approach to automated monotonicity detection has been created. Second, a novel longitudinal classification algorithm was developed, based on the idea of nested decision trees. The basic idea is that each internal node of

the outer tree consists of a decision tree built using as input all the temporal variations (in different time points) of the same longitudinal attribute. Third, two new methods for deriving monotonicity-based longitudinal attributes were created, and a unified algorithm was created that united all of these features into one monotonic longitudinal classification algorithm, the first algorithm to combine the two approaches.

The proposed methods were evaluated on longitudinal datasets derived from the English Longitudinal Study of Ageing (ELSA), and the results confirmed that the Nested Tree algorithm outperformed a conventional decision tree algorithm regarding predictive performance.

Acknowledgements

Firstly, I would like to express my gratitude to the School of Computing at University of Kent for giving me the opportunity to study for a PhD degree in Computer Science and providing me with funding for the first three years.

I would like to thank my academic supervisors: Prof. Alex A. Freitas and Dr. Fernando E. B. Otero for their support and patience during the past five years.

I would like to thank Research Associate Caio Ribeiro for being very helpful at the initial stages of my research and providing pre-processed ELSA datasets that I have extensively used in my studies.

I would like to thank Prof. Ian Witten, who has sadly passed away earlier this year, for his work on teaching and popularising the field of Data Mining and for creating his online course on Data Mining with Weka that became my first venture into this field.

Finally, I would like to thank my family and friends who have provided support during the hardest times of my life.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Overview of the Research Topic

This research is focused mainly on the field of Classification, although it also involves the field of Regression. Classification and regression are two fields of Machine Learning (or Data Mining) that focus on predicting the values of a target attribute in an input dataset (Chapter 3 in [10]). Classification algorithms aim to build machine learning models capable of analysing new data instances and assigning class labels - discrete values that categorise each new data instance into one of the previously observed categories. Regression algorithms aim to construct machine learning models that can analyse new data instances and make regression predictions about them, in the form of continuous numerical values (Section 1.4 in [10]).

For example, in a medical data mining scenario, these models may be used to observe the patterns in the data of current patients and learn to make predictions about the health conditions of new patients. In this scenario, a classification model may be used to analyse the medical test results of a new patient and make predictions on whether they have a certain disease or not - assigning that patient a categorical label containing either a positive or negative value. Regression models

can be used to provide more nuanced estimations in the form of numerical predictions based on previously observed data. For example, medical data of patients previously treated for kidney failure can be used to predict the best dosage for a prescription drug, the optimal length of dialysis treatment sessions, or even the expected survival time based on currently received treatment.

In practice, regression models can sometimes be used for classification as well, as long as their numerical predictions can be sensibly converted into class labels. For example, a model can be constructed that predicts the numerical probability that a patient has a certain disease. Then, by applying a numerical threshold (e.g. a positive class probability above 75%), this probability can be turned into one of two labels: high-risk or low-risk, and these labels can be used as guidance for future diagnosis. Alternatively, the class variable may be treated as a continuous numerical variable, e.g. with negative class represented as 0 and positive class represented as 1. The regression model can then be trained to predict the value of this continuous target attribute, and a threshold can be applied to convert its predictions into discreet categorical class predictions, e.g. negative class for values below 0.5, and positive class for values above or equal to 0.5. This is sometimes called classification-by-regression.

This study focuses on developing novel classification and classification-by-regression approaches in order to expand two further subfields of classification: Monotonic Classification and Longitudinal Classification.

Monotonic relations are relations in data where some attributes have a specific type of relationship, as follows [6]. In a positive monotonic relation, an increase in the value of one attribute tends to correlate with an increase in the value of another attribute, and a decrease in the former likewise correlates to a decrease in the latter. In a negative monotonic relation, the opposite is true: as the value of one attribute increases, the value of another attribute tends to decrease; or vice-versa. In real world datasets, a monotonic relation can be a result of a known

common causality, it can be evidence of a causal connection between the two, or it can just be an interesting correlation that has not yet been explored.

Monotonic classification is a subfield of classification that aims to utilise monotonic relations in data in order to produce better classification models [6, 8, 4]. Some studies use monotonic relations to provide additional information for classification algorithms in order to produce more accurate classification models [22, 11, 21]. Some studies use real-world data and have to produce models that follow certain real-world restrictions [5, 62, 67]. In such studies, monotonic relations are presented to the classification algorithm in the form of constraints that limit and prevent the algorithm from constructing models that would violate these real-world rules. Additionally, monotonicity relations can be used to remove noise from the datasets and ensure that the data follows the expected patterns [50, 41, 18].

Longitudinal datasets contain repeated measurements of the same data attributes, measured at different points in time [37, 56]. Conventional datasets can be represented as flat 2-dimensional tables, with one dimension being the horizontal dimension of attributes and another being the vertical dimension of data instances. Longitudinal datasets effectively add a third dimension of time to the datasets. By repeatedly measuring the same attributes at different time points, they are effectively creating a dataset that consists of multiple versions of itself, recorded at different times. Such datasets can be used to study time-based trends that occur in the datasets, observe how attribute values changed over time, and even make predictions about future values of these attributes [29].

For example, in the medical context, longitudinal datasets can be used to represent the results of regular medical screenings, where the same patients complete the same set of medical tests every few years. After a number of years, when multiple time points have been recorded for each patient, such dataset can be used to study the dynamics of patients' health and new data mining models can be built

to make useful predictions about the future health of new and current patients.

Longitudinal classification is a subfield of classification that aims to build classification models that utilise longitudinal data to make class predictions. With the rise in availability of longitudinal datasets, and an increasing demand for time-based predictions, especially in the field of bioinformatics, longitudinal classification is becoming an increasingly popular field of study [56].

The English Longitudinal Study of Ageing (ELSA) dataset [36], which was used heavily in this research, is an example of longitudinal medical data, containing regular medical measurements taken from elderly patients at two-year intervals. In this study it is used to build classification models that make predictions on whether a patient will have a certain age-related disease at a specific time point.

1.2 Contributions of this Research

In this research, I propose and develop novel approaches for monotonic and longitudinal classification and use datasets derived from the ELSA database to build and evaluate classification models constructed with the proposed approaches. This research has three main contributions, as follows.

The first contribution is a methodology for the automated detection of monotonic relations in datasets. A popular tree-boosting ensemble algorithm XGBoost was used to construct monotonic and non-monotonic classification models and evaluate the effect of monotonicity constraints on predictive accuracy. The results on ELSA datasets have shown that automated monotonicity detection is capable of producing useful monotonicity constraints and the enforcement of these constraints has resulted in a significant improvement in the predictive accuracy of XGBoost.

The second contribution of this research is the Nested Trees algorithm - a novel

classification algorithm that can use longitudinal datasets to produce longitudinally-aware and interpretable models. The algorithm was evaluated using longitudinal classification tasks of disease prediction in ELSA datasets and was able to outperform the conventional, longitudinally-unaware decision trees.

The final contribution of this research is a unified approach that used the Nested Trees algorithm with a range of monotonicity detection tools, thus creating the first monotonic longitudinal classification algorithm. This monotonic version of Nested Trees includes tools for detecting monotonic relations in longitudinal datasets, enforcing monotonic constraints in longitudinal classification models, constructing derived attributes representing monotonic trends in the values of longitudinal attributes, and provides additional metrics that provide better longitudinal interpretability for its models. The monotonic version of the Nested Trees algorithm was tested using ELSA datasets and different combinations of monotonicity-based features were evaluated. Overall, significant predictive accuracy differences between two versions of the Nested Trees algorithm were discovered and the impact of the longitudinal monotonicity-based derived attributes was demonstrated.

1.3 Thesis' Structure

The remainder of the thesis is structured as follows.

Chapter 2 provides a background for the thesis, introducing the terminology, describing the core concepts used in the study and providing examples for some of the more advanced concepts.

Chapter 3 provides a detailed literature review of the fields of Monotonic and Longitudinal classification. It provides an overview of each field, detailed descriptions of the relevant studies, as well as criticism of existing approaches and proposals for improvement.

Chapter 4 describes the first contribution of this research: a novel approach to automated detection of monotonic relations and construction of monotonic constraints for monotonic classification algorithms.

Chapter 5 describes the second contribution: a novel longitudinal classification algorithm that can produce longitudinally-aware and interpretable classification models.

Chapter 6 describes the final contribution of this research: a unified approach to longitudinal classification that makes use of automated monotonicity detection to define monotonic constraints and create useful longitudinally-derived attributes in order to build monotonic longitudinally-aware classification models.

Chapter 7 concludes the thesis; it summarises the main contributions of the research and it proposes future avenues of research.

1.4 Publications Derived From This Thesis

Two publications were already made as a result of this research. The first publication [48] was a paper that presented the automated monotonicity detection approach described in Chapter 4. It was published in 2019 in the conference proceedings of International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-AI 2019).

The second publication [47] was a paper presenting an early version of the novel longitudinal classification algorithm described in Chapter 5, and it was published in the proceedings of SAC '22: the 37th ACM/SIGAPP Symposium on Applied Computing.

Overall, these publications cover the contributions described in Chapter 4 and the majority of Chapter 5. The basic version of the Nested Trees algorithm implementation was also published on GitHub (github.com/NestedTrees/NestedTrees), making it the first open-source longitudinal classification algorithm. Additionally,

the Nested Trees algorithm has been integrated into a custom python library built on scikit-learn, called Scikit-longitudinal, currently developed by a University of Kent PhD student Simon Provost. Public version currently available on GitHub (https://github.com/simonprovost/scikit-longitudinal).

Chapter 2

Background

The research presented in this thesis involves developing and testing classification algorithms that performed the tasks of monotonic and longitudinal classification. This chapter will introduce the core concepts used in this research: the data mining tasks of classification and regression, decision tree classification models, structures of flat and longitudinal datasets, the concept of monotonic relations and their use in classification, as well as some of the methodologies used for evaluating the predictive performance of classification and regression models.

2.1 Classification and Regression

One of the main tasks in Data Mining is the task of predicting a value of a target attribute based on other attribute values. This is a supervised learning task that requires the model to be trained using data instances in which the target attribute value is already known. After the models have been trained, they can be used to analyse new data instances and make predictions for the value of the target attribute (Section 1.4 in [10]).

In some cases, the target attribute is categorical, having a small number of discrete values in its domain. In those cases, the task of predicting the value of that

attribute is called Classification (Chapter 3 in [10]) and the attribute is usually referred to as the ”Class“ attribute. It is very common for the class attribute to only have two possible values, and in those cases the task is referred to as Binary Classification.

In some cases the target attribute instead has a domain consisting of a range of continuous numerical values. In those cases, the task of predicting the value of that attribute is called Regression (Section 1.4 in [10]).

Some data mining models focus solely on the task of classification. They make predictions by analysing the attribute values of a specific data instance and choosing the most suitable value from the class attribute domain. Other models focus solely on the task of regression. These models analyse attribute values of a specific data instance and make a numerical prediction for the value of the target attribute. It is important to note that regression models aim to make a prediction that best fits the pattern learned during training and they can produce target attribute values that are not present in the training data.

Depending on the nature of the domain of the class attribute in classification tasks, the categorical values of the class attribute can sometimes be represented as numerical values, thus allowing regression models to be applied to classification datasets. This is usually possible in datasets where the domain of the class attribute is ordinal, meaning that a sensible ordering can be defined between the values of the class attribute. Additionally, this approach can always be used for binary classification tasks, since it is always safe to assume an order between just two categorical values. Some classification algorithms use this approach to first construct a regression model using the training data and then apply numerical thresholds to turn its continuous numeric predictions into discrete class predictions. We refer to this approach as classification-by-regression.

2.2 Decision Tree Models and Their Components

This study focuses on classification models that learn to classify data instances by recursively subdividing the training set of instances into smaller and purer subsets – where a subset is deemed pure if all data instances in that subset have the same value for the class attribute. These models are commonly refereed to as Decision Trees, due to their tree-like structure and their classification process consisting of a series of decisions based on attribute values. Two more classification models are relevant in the context of decision trees: ZeroR and Decision Rules.

ZeroR classifier is considered to be the simplest classification algorithm. It does not use any attributes in its training and simply learns to predict the class value that was most common in the training set. A classification algorithm that simply predicts the most frequent class value has little use on its own, but it can be used to construct the leaf nodes of a decision tree model.

Decision rules are classification algorithms that use a predictor attribute to split the training set into subsets. A decision rule is constructed by iteratively trying every available data split based on every available predictor attribute and selecting the split that maximises the value of a certain split metric. In classification, a split serves as a ”test“ for a value of a specific data attribute which results in some instances passing and going into one subset and other instances failing and going into another subset. Decision rules can be built using a variety of different metrics, with the most common being Information Gain and Gini Impurity. The purpose of these metrics is to numerically measure the purity of the subsets produced by a split.

Decision Tree (Chapter 4 in [10]) is a classification algorithm that recursively splits the training set into smaller and purer subsets using decision rules. It starts by using the training set to construct the first decision rule that becomes the first, root node of the tree. It then splits the training set into subsets using the decision rule and recursively applies the same process to the subsets. The process

Table 2.1: Flat dataset example

Id	Attr1	Attr2	Attr3	Attr4	Class
1	0.3	0.7	0.7	0.8	1
2	0.5	0.6	0.3	0.5	0
3	0.2	0.5	0.6	0.7	1
4	0.3	0.5	0.9	0.2	1
5	0.4	0.6	0.2	0.5	0

continues recursively until either the subsets only contain instances belonging to the same class or until some other termination condition is reached. When these conditions are reached, the algorithm constructs a leaf node that makes a class prediction. The class prediction is determined by the class distribution in the subset, essentially applying the ZeroR algorithm to construct the leaf node. The resulting tree structure of decision rules and leaf nodes can then be used to classify new data instances by starting from the root node and following the decision rules until a leaf node is reached and a prediction is made.

2.3 Flat and Longitudinal Datasets

Most conventional classification datasets can be represented as tables. Table 2.1 shows a simple flat dataset that contains five data instances and six attributes. The last attribute is designated as the class attribute for the classification problem. Note that the attribute Id is included in this table only for the reader's convenience when identifying individual data instances. In practice an attribute such as Id should not be used as a predictor attribute by a classification algorithm, because this kind of attribute takes a unique value for each data instance, and so it does not have any generalisation power.

The majority of classification algorithms are designed to work with this type of datasets. In these datasets, each data instance has one value per attribute and one class value, so classification algorithms can safely rely on new data instances

all having the same tabular structure and build their models accordingly.

Longitudinal datasets contain repeated measurements of the same attributes at different points in time. Such datasets can be used to represent data acquired from repeating events, such as medical data measurements collected during regular patient screenings [38]. These datasets contain multiple sets of values for all or some of the data attributes.

Tables 2.2, 2.3, 2.4, and 2.5 show an example of a longitudinal dataset. In this example, five patients participate in the measurements across four different time points (waves of measurements). At each point, their blood glucose level is measured and either a positive or a negative diabetes diagnosis is given to each patient. We can use this data to see how patients' blood glucose levels have changed across time and how three additional patients have developed diabetes towards the end of the study. In this example, the longitudinal dataset is represented as four separate waves, each in itself being represented by a flat dataset.

Table 2.2: Wave 1

Id	Glucose	Diabetes
1	5.7	No
2	7.2	Yes
3	6.7	No
3	6.8	No
5	6.5	No

Table 2.3: Wave 2

Id	Glucose	Diabetes
1	5.9	No
2	7.8	Yes
3	6.7	No
3	6.9	No
5	7.6	Yes

Table 2.4: Wave 3

Id	Glucose	Diabetes
1	6.1	No
2	7.5	Yes
3	6.8	No
3	7.0	Yes
5	7.6	Yes

Table 2.5: Wave 4

Id	Glucose	Diabetes
1	6.3	No
2	7.3	Yes
3	7.1	Yes
3	7.2	Yes
5	7.7	Yes

Processing a dataset that is comprised of a few smaller datasets can be cumbersome, and many applications combine all of the longitudinal data into one

Table 2.6: Flattened longitudinal dataset example

Id	**Glu_w1**	**Glu_w2**	**Glu_w3**	**Glu_w4**	**Diabetes_w4**
1	5.7	5.9	6.1	6.3	No
2	7.2	7.8	7.5	7.3	Yes
3	6.7	6.7	6.8	7.1	Yes
4	6.8	6.9	7.0	7.2	Yes
5	6.5	7.6	7.6	7.7	Yes

dataset [56]. This process is called data flattening. Table 2.6 shows an example of a flattened longitudinal dataset. Here, data about the same five patients has been merged into a single dataset, with four measurements of blood glucose levels represented as four different data attributes. The data attributes have been renamed to reflect both the attribute meaning and which wave of measurements it originated from. Diabetes data from the final wave was added to represent the final outcome of the diagnosis. The resulting dataset therefore has six data attributes, with four of them all being part of the same longitudinal attribute. In this representation, longitudinal attributes can be expressed as groups of data attributes.

This data representation is very common in longitudinal classification studies where a longitudinal dataset gets flattened and standard classification algorithms are used to create classification models with it [66, 42, 45]. It is common to designate a specific time point of one of the longitudinal attributes as the target attribute for classification. In the previous example, a classification problem was defined where four measurements of blood glucose levels taken at different time points were used as predictor attributes and the diabetes diagnosis from the final time point was designated as the class attribute.

2.3.1 Longitudinal Datasets and Time Series Analysis

Longitudinal datasets contain repeated measurements of the same attributes taken at different points in time. The same can be said about the datasets used in the

field of Time Series Analysis [59] - a field of machine learning that is in some ways similar to the field of Longitudinal Classification - one of the main focuses of this research. The two fields can be considered similar in that they both aim to extract useful knowledge from data attributes measured repeatedly at different time points, and learn to make predictions about the values of some of these attributes.

Some conventional time series studies use datasets measuring the value of one attribute repeatedly over a series of time points - hence the name Time Series. A subfield of Time Series called Multivariate Time Series Analysis uses multiple different time series at the same time, effectively applying time series analysis to multiple variables measured at a series of different time points. Multivariate Time Series datasets are noticeably more complex than the conventional, univariate Time Series datasets. While the univariate datasets can be represented as one-dimensional vectors, effectively only having variations across the time dimension, the multivariate datasets introduce a second dimension - the dimension of different variables that are being measured.

Longitudinal Classification datasets are conventionally three-dimensional, including the dimension of data instances, the dimension of data attributes, and the dimension of time. Tables 2.2-2.5 demonstrate the three-dimensional nature of longitudinal datasets, with multiple two-dimensional datasets being measured across a third dimension of multiple time points.

Another difference between the two fields is the time points used in the datasets. The field of Time Series analysis conventionally focuses on datasets which contain measurements taken repeatedly at specific intervals - so the time between the measurements is consistent. In longitudinal classification studies, while it is common to see datasets with measurements occurring at regular intervals, it is not technically a requirement for the time points to be separated by equal intervals. Additionally, conventional Time Series analysis methods rely on a larger number

of time points being present in the dataset, while the longitudinal classification studies, as will be shown in the literature review, are sometimes conducted using as few as just two time points.

While it may be possible, with some dataset pre-processing, to apply Multivariate Time Series analysis methods to longitudinal classification datasets, e.g. by analysing the temporal values of the different longitudinal attributes of a specific data instance, it would necessarily require the data instance to be analysed separately from the rest of the population, and would therefore not be applicable to the task of learning classification models for data instances. Classification tasks in the field of Time Series Analysis focus on the classification of time points rather than data instances, so those methods are not directly applicable to Longitudinal Classification.

2.4 Monotonicity

Monotonicity is a mathematical property of a relationship between two variables. A function is called monotonic when changing the value of one variable results in a non-negative change in either always the same or always the opposite direction in another variable. Figure 2.1 shows examples of two monotonic functions and Figure 2.2 shows examples of two non-monotonic functions between variables X and Y.

In the context of data mining, the monotonic property between two pairs of ordinal attribute values (x_i, y_i) and (x_j, y_j) is defined as follows:

The attributes x and y are considered monotonic if any two attribute value pairs (x_i, y_i) and (x_j, y_j) in the dataset are monotonic, meaning that for a higher value of one attribute (x), the value of another attribute (y) can not be lower. This can also be used for negative monotonic relations, where for a higher value of one attribute, the value of another attribute can not be higher.

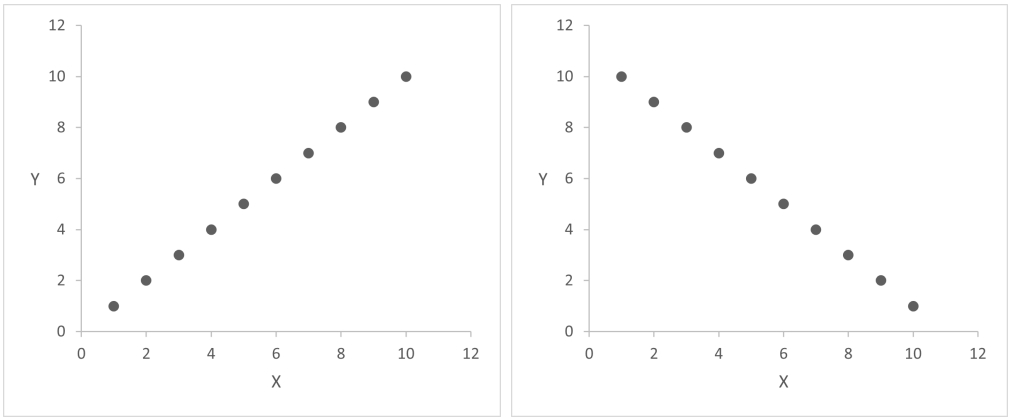

Figure 2.1: Monotonic functions between two variables

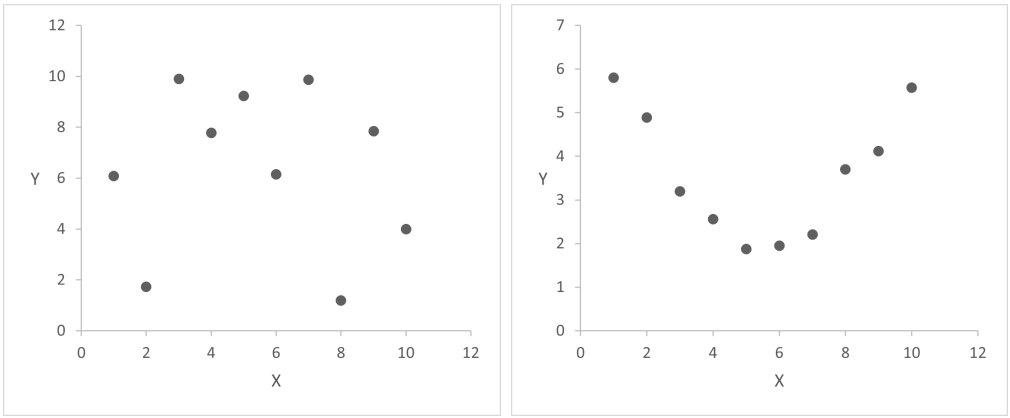

Figure 2.2: Non-Monotonic functions between two variables

$$x_i > x_j \implies y_i \geq y_j$$

In practice, fully monotonic relations are very rare in real world datasets. It is, however, not unusual for two attributes to have a relatively strong monotonic relationship that may be useful for classification tasks. Table 2.7 shows an example of a classification dataset. This dataset contains ten data instances, four predictor attributes and a class attribute that has three discrete ordinal values. In this dataset, one of the predictor attributes has a strong monotonic relation with the class attribute.

We can easily see which predictor attribute has a monotonic relation with the

Table 2.7: A classification dataset with one attribute having a strong monotonic relation with the class

Id	**Attr1**	**Attr2**	**Attr3**	**Attr4**	**Class**
1	0.8	0.4	0.2	0.6	1
2	0.1	0.9	0.1	0.8	1
3	0.7	0.6	0.3	0.1	1
4	0.6	0.5	0.4	0.1	2
5	0.1	0.4	0.6	0.2	2
6	0.3	0.7	0.5	0.8	2
7	0.8	0.3	0.7	0.4	2
8	0.1	0.9	0.8	0.6	3
9	0.6	0.4	1.0	0.5	3
10	0.2	0.1	0.9	0.3	3

class if we plot the graphs of each attribute versus the class. Figure 2.3 shows these four attributes plotted against the class. It is obvious from the graphs that the third predictor attribute has a strong monotonic relationship with the class attribute.

The strength of the monotonic relationship between two variables can be measured using the Spearman rank correlation coefficient [23]. This correlation coefficient is a mathematical measure that estimates how well the relationship between two variables can be described using a monotonic function. It assesses the strength of a monotonic relationship between two variables by first converting each variable to a ranked variable. It then applies the Pearson's correlation coefficient to the two ranked variables to estimate the degree of linear correlation between them.

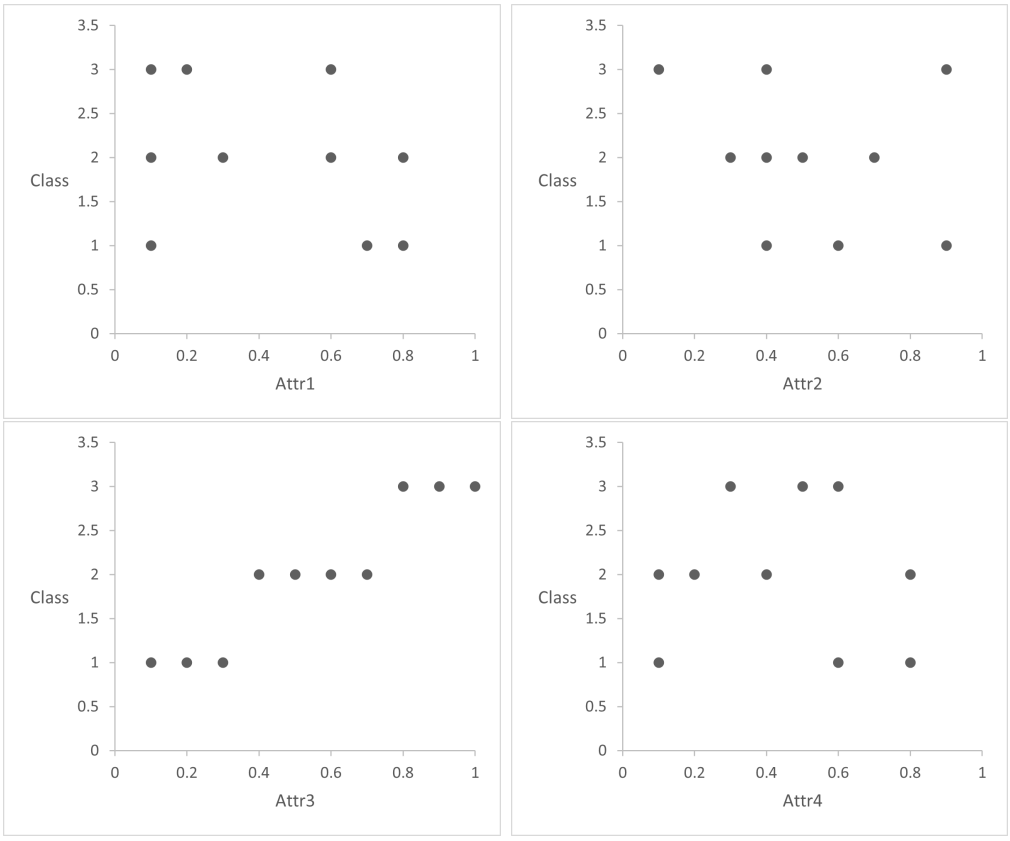

Figure 2.3: Predictor attributes plotted against the class

The Pearson's correlation coefficient [9] is estimated using the covariance of two attributes divided by the product of their standard deviations.

$$Pearson_{X,Y} = \frac{cov(X,Y)}{\sigma_X \times \sigma_Y}$$

2.5 Monotonic Classification

Monotonic classification is a subfield of classification that aims to utilise the monotonic relations found in data to construct more accurate and more acceptable classification models [4]. Monotonic relations are usually represented in the form of monotonic constraints – rules that the classification algorithm must follow when constructing the model. These rules represent monotonic relations that must

be preserved by the model, e.g. an increase in the value of a specific predictor attribute must always result in a non-decrease in the predicted class value (for ordinal class attributes) because the two have a positive monotonic relation.

The necessity for the use of monotonicity constraints can also arise from the restrictions of the application domain [6]. In some applications domains, it is required that the outcome of a decision always scales positively with some predictor attributes. In such cases, preserving monotonic relations may be necessary for the model to be accepted by domain experts.

Monotonicity enforcement is the term used to describe the process of incorporating the monotonic relations into the classification models [6]. Monotonicity enforcement approaches can be broadly categorised [11] based on how monotonicity constraints are enforced:

- Hard monotonicity constraints are used to produce models which have a fully monotonic output. This approach restricts the model to only classify instances in full accordance with monotonicity constraints provided even if all other elements of the model (e.g., the leaf node value of the decision tree or a raw regression value of a boosted model) would suggest otherwise;

- Soft monotonicity constraints are used to produce models which are more monotonic than unconstrained ones, but may not necessarily produce fully monotonic output. This approach essentially guides the model to classify instances in a more monotonic fashion, but some instances can still be classified based solely on raw attribute values regardless of the constraints.

2.6 Evaluating Classification Algorithms

This study focuses on creating and evaluating new classification approaches. There is a large variety of tools and metrics that can be used to evaluate and compare the performance of different classification algorithms.

2.6.1 Predictive accuracy metrics

Accuracy is the simplest metric that can be used to evaluate the quality of a classification model's predictions. It is calculated as the proportion of class predictions that were correct.

When quantifying the class predictions made by a classification model, it can be useful to estimate metrics for individual class values to gain a deeper understanding into the model's capability for predicting specific classes. It is common to use four metrics to represent different classification outcomes:

- **True Positives (TP)** are all predictions where the positive class value was assigned to an instance that actually had that class value, so the class prediction was correct.
- **False Positives (FP)** are all predictions where the positive class value was assigned to an instance that did not have that class value, so the class prediction was incorrect.
- **False Negatives (FN)** are all predictions where the positive class value was not assigned to an instance that had that class value, so the class prediction was incorrect.
- **True Negatives (TN)** are all predictions where the positive class value was not assigned to an instance that did not have that class value, so the class prediction was correct.

These four metrics are used in calculating more specialised predictive accuracy metrics (Chapter 12 in [10]).

F-Measure is metric that is used to evaluate the ability of the classification model to recognise and predict specific class values. F-Measure is estimated separately for each class value and is estimated as the harmonic mean of two other metrics: precision and recall. Precision is estimated as the number of correctly

classified positive instances (TP) divided by the total number of instances that were predicted to have the positive class label (TP + FP). Recall is estimated as the number of correctly classified positive instances (TP) divided by the total number of instances that belonged to the positive class (TP + FN).

$$FMeasure = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

The F-Measure (sometimes also called F1 score (Section 12.1 in [10]) can be calculated separately for each class and the values can be combined to create a meaningful measure of the classification model's ability to distinguish between instances belonging to different classes. The best way to combine F-Measures of different classes is to calculate the Average F-Measure by calculating their arithmetic mean. This ensures that the resulting metric assesses the classification performance of the classification model without being biased towards any individual class labels.

Another useful metric is the Area Under ROC curve (AUROC) (Section 12.4 in [10]). This metric relies on the model having the ability to predict the regression value of the class attribute instead of just assigning categorical class labels. The ROC curve is constructed by varying the classification threshold by which the regression values are being converted to class predictions and recording the True Positive Rate (TPR) and the False Positive Rate (FPR) at each threshold.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

The two are plotted against each other on a graph, forming the ROC curve. The area under the ROC curve is then calculated to provide a measure of how well the classification model is able to differentiate data instances belonging to different classes.

2.6.2 Evaluating the performance of classification algorithms

When evaluating classification algorithms, it is useful to collect information from different runs of each algorithm, using different classification problems. There are a few approaches that can be useful for getting a more reliable set of results from classification experiments.

The important approach for reliable classification model evaluation is the separation of the training set from the test set (Section 7.2 in [10]). This means that the data instances used in model construction will not be used for model evaluation. This approach is practically the minimal requirement for a valid classification model evaluation.

Cross-validation is a powerful approach that can be used to evaluate multiple classification models constructed by the same classification algorithm using the same dataset (Section 7.3 in [10]). This approach divides the original dataset into several parts, called folds, with approximately the same number of instances in each fold. A particular type of cross-validation, which ensures that the distribution of class labels within each fold stays the same as it is in the whole dataset, is called stratified cross-validation. Once the folds have been created, the cross-validation procedure iteratively constructs classification models using one fold in turn as the test set and merging the remaining folds into the training set. The number of

models constructed is equal to the number of folds, so each fold is only used exactly once for testing. It is common to split the dataset into 10 folds, this is called 10-fold cross-validation. The results are averaged across all test folds in order to produce a reliable estimation of the predictive performance. This approach is very useful as it allows a dataset to be used to produce a large number of classification models, trained and tested using different combinations of data instances, and provides a reliable estimate for the performance of the classification algorithm.

To increase the reliability of results even further, the cross-validation experiment can be repeated multiple times and the results can be averaged across all experiments. In order to ensure that different cross-validation experiments don't end up using the same folds, the cross-validation process can be implemented in a way where the splitting of the dataset into folds relies on a random number generator and the random seed can be set to a different value on every repetition. This allows us to acquire a highly-reliable set of predictive performance measures averaged over a large number of experiments. For example, 30 repetitions of 10-fold cross-validation with different random seeds means that the final results will be averaged over 300 classification models constructed using 300 different training sets.

In practical evaluation of classification algorithms, it is also useful to apply the algorithms to multiple different classification problems (datasets) in order to get a substantial amount of results for comparison. When evaluating a novel and complex classification algorithm, it is useful to also include a more standard algorithm into the experiments to serve as the baseline for all predictive performance comparisons.

If a classification algorithm has optional procedures, it is useful to run different experiments using the algorithm with different sets of procedures enabled. Having results originating from experiments using different versions of the same algorithm can be useful for evaluating the effects that each individual procedure has on the

algorithm's predictive performance.

Finally, after a number of algorithm versions have been evaluated on a number of datasets, their predictive performance metrics can be compared using statistical significance tests. Statistical significance testing usually includes evaluating the hypothesis that there are no significant predictive performance differences between the different classification algorithms. The predictive performance data collected from the experiments is used to calculate test statistics and make an estimation of the probability that two sets of measurements could be produced by the same processes. This probability value is referred to as p-value and a low value (usually a value below 0.05) is considered to represent a significant difference between measurements. Therefore, if a novel algorithm delivers predictive performance metrics that are deemed by statistical tests to be significantly better than those produced by the baseline algorithm, it can be strongly argued that the novel algorithm is superior. Same principle applies to different versions of the same algorithm - their predictive performance metrics can be compared and significant differences can indicate the superiority of certain feature sets.

A variety of statistical significance tests have been used in the literature for the purpose of evaluating the differences in performance of different classification algorithms. While there is no consensus in the literature on the most appropriate statistical tests to be used, some useful guidelines have been suggested by different authors. In a recent 2024 paper [46], Rainio et al. have provided a very useful set of guidelines for selecting the significance test when comparing classification models. According to their guidelines, the Friedman test is the most appropriate testing method when comparing multiple classification models using metrics such as the F-Measure and AUROC. In an earlier 2006 paper [19], Demsar et al. similarly assessed the applicability of different statistical tests to the task of evaluating the difference in the performance of classification algorithms. In this paper, the authors similarly concluded that non-parametric tests such as the Wilcoxon and

Friedman tests are most suitable for this task. Additionally, the paper included some evaluation of the different post-hoc tests that could be applied after the Friedman test to identify the significant differences between groups of algorithms. The paper does not provide any conclusive results on these, however, citing the similarity in their results. The main post-hoc tests highlighted in the analysis are the Holm test, highlighted for its simplicity, and the Hommel test, highlighted for its marginally higher power. Similar comments have been made by some other researchers in the field [44], highlighting the usefulness of the step down methods such as the Holm post-hoc test.

In summary, cross-validation allows for a reliable predictive performance estimation, repeated experiments enhance that reliability further, classic algorithms can be used to acquire baseline predictive performance measures and statistical tests can be used to find significant differences between predictive performances of different classification algorithms.

Chapter 3

Literature Review

This chapter will provide a literature review on the main topics related to my research. Firstly, the literature on monotonicity detection and monotonic classification will be reviewed and major monotonic classification studies will be described in detail. Secondly, studies from the field of longitudinal classification will be reviewed, along with some interesting algorithms. Finally, a critical assessment of two fields will be provided and potential areas for development identified.

3.1 Monotonic Classification

Monotonic classification is a task of classification that uses monotonic relations in data to produce more monotonic predictions, build classification models more acceptable to users, improve the predictive accuracy of the classification models, or adhere to monotonic constraints imposed by experts on the target application domain. The field of monotonic classification has existed for over 30 years now, and a large variety of monotonicity classification approaches have been proposed by various authors.

The first known use of monotonicity in classification tasks was in a 1989 paper by Ben-David et al.[5]. In that study, monotonic constraints were first used to

restrict the model construction process to avoid violating these constraints. Following the initial work by Ben-David et al., the field of monotonic classification started to develop as a sub-field of the broader field of classification in machine learning (or data mining).

The expansion of the field of monotonic classification is best illustrated by a recent review of monotonic classification approaches that was published in 2019 by Cano et al. [4]. A total of 64 different approaches were reviewed in their publication and a new taxonomy for monotonic classification approaches was proposed. The taxonomy only focuses on classification approaches and, while some regression algorithms are mentioned, they are only reviewed in context of classification. This new taxonomy divided monotonic classification approaches into two categories:

- **Monotonic Data Pre-Processing**. This category represents all approaches that use data pre-processing techniques to improve the predictive performance of classification algorithms and ensure partial or full monotonicity of the training data. This category is divided into subgroups based on the core approaches used in the studies:
  - **Relabeling** techniques are methods for altering the class labels of the instances that produce monotonicity violations, thus producing fully monotonic training sets.
  - **Feature selection** techniques aim to improve the performance by selecting the most relevant features for a given classification task. The features are selected based not only on their predictive power but also on the monotonic assumptions being used.
  - **Instance and Training set selection** techniques aim to select an optimal set of instances to create a partially or fully monotonic classification model.

- **Monotonic Classifiers**. This category represents all approaches that generate fully or partially monotonic predictive models. These approaches include algorithm-based adjustments made to ensure full or partial monotonicity of learned models and/or post-processing procedures that modify the learned model to achieve higher monotonicity. This category is further divided into several subgroups, where approaches are grouped based on the strategy used to learn the classification model. These include instance-based classifiers, decision tree and rule induction classifiers, ensemble models, neural networks, support vector machines, among others.

3.1.1 Monotonic Data Pre-Processing

There is a number of studies that focused solely on pre-processing the training dataset in order to make it more/fully monotonic before applying classification algorithms to it.

Monotonic Relabeling

A set of three data relabeling algorithms were proposed by Pijls and Potharst [50]. The Naive Relabel is a greedy algorithm for relabeling datasets that produces fully monotonic outputs, but does not guarantee an optimal solution to the relabeling problem. It uses a monotonicity violation graph data structure to store all of the data instances and monotonicity violations between them. It then relabels all of the data instances that violate monotonic constraints by iteratively assigning them new class labels until there are no more violations left. Since it is a greedy algorithm, it only aims to minimise the number of monotonicity violations and does not attempt to make the smallest possible number of relabelings. The Borders algorithm is an extension of Naive Relabel that minimises the differences between the new and original labels and uses a simpler approach for selecting the new label values. The Antichain algorithm is a further extension to the Borders algorithm

that minimises the total number of relabelings by constructing a monotonicity violation graph and finding the maximum antichain in it. The Antichain approach produces a fully monotonic dataset with a minimal number of relabelings. The paper provides some analysis of the optimality of the resulting relabelings and the time complexity of the three proposed algorithms. However, no further insight is provided into the usefulness of such relabelings for monotonic classification.

Another relabeling algorithm was proposed by Rademaker et al. [41]. The Optimal Ordinal Monotone Relabeling algorithm constructs a minimum flow network of input data instances and reassigns the class labels of all instances that are deemed too high or too low with relation to the core set of monotone instances. The paper claims that this approach is optimal in that it guarantees to relabel a minimal number of data instances. Authors do not provide any evidence to support the applicability of this algorithm to monotonic classification. Relabeling experiments are conducted using only one dataset without applying any classification algorithms at all.

In his 2010 paper, Ad Feelders proposed a new algorithm for monotonic relabeling [22]. The problem of computing an optimal monotone relabeling is reduced to the convex cost closure problem on the set of data instances. In the original 2010 paper, a kNN algorithm and a decision tree algorithm were used to measure the effects that the relableing had on classification performance using 10 classification problems. Follow-up papers on Feelders relabeling studies [1, 2] have expanded the analysis of the same approaches and evaluated it on synthetic data. Overall, the approach proposed by Feelders have had a few issues. Firstly, the first two Feelders studies used datasets that were already highly monotonic, with 80.79% to 99.81% of instances being monotonic. Secondly, the number of relabeled instances was unreasonably high in some experiments: relabeling 424 instances out of 1000 in the Lev dataset and 140 out of 235 in Ohsumed dataset. Thirdly, the predictive accuracy results were only marginally better on relabeled datasets compared to

the original datasets.

The relabeling algorithm proposed by Daniels-Velikova is perhaps the most intuitive of them all [18]. It is a greedy relabeling algorithm that iteratively applies label changes, making at least one change at each iteration, until the dataset is fully monotonic. The changes are selected based on their overall improvement to monotonicity of the dataset, which is measured as the total number of non-monotonic data instance pairs that would be eliminated from the dataset if the change was applied. It is a remarkably simple algorithm, however it does not guarantee an optimal solution to the monotonic relabeling problem.

It is very important to note here that the improvement in predictive accuracy of classification models may be a result of the introduction of additional knowledge about the nature of the dataset through a conceptually wrong evaluation methodology, as follows. In the 5-fold cross-validation experiments conducted in this study, the monotonicity constraints were pre-determined based on the trends that occur in the whole dataset, not just in the training set, so the introduction of such monotonicity constraints has used some properties of the test set in the model construction, which goes against the nature of cross-validation.

Although there is some variety in the different algorithms proposed for monotonic relabeling of dataset instances, all relabeling algorithms use largely the same set of approaches to resolving the issue of non-monotonicity in the dataset. They all rely on some pre-determined assumptions about monotonicity relations in datasets, they all aim to achieve perfect compliance with those assumptions, and they all alter the dataset class labels to achieve this.

The relabeling approach relies largely on the correctness of those monotonicity relations assumptions made prior to the re-labeling process, however none of the papers have shown any evidence of those assumptions being evaluated against the real data. In some cases, the quality of these monotonicity assumptions has been shown to be highly questionable. For example, in the studies published by

Ad Fielders [22, 1, 2] the number of monotonic relabelings was often very high, sometimes having over 50% of the data instances relabeled in order to achieve compliance with the presumed monotonicity relations.

It is, in theory, possible to achieve adherence to any arbitrary monotonicity assumptions in any arbitrary dataset by relabeling a large enough portion of the dataset. If a relabeling algorithm was able to achieve full monotonicity by applying only a few such relabelings, then it would be evident that the monotonicity assumptions were generally sound and matched the real monotonic relations that were present in the data. However, if a very large portion of data instances needed to be relabeled in order to achieve full monotonicity, then it can be argued that the initial monotonicity assumptions were questionable and did not accurately represent the real monotonic relations in the dataset. None of the monotonic relabeling papers mentioned this issue and none have provided any evaluation of the correctness of the assumed monotonic relations.

Overall, the practice of monotonic relabeling is highly questionable. It relies on the correctness of the initial monotonicity assumptions, never evaluating their correctness against the real data, and makes unjustified changes to the dataset in order to "force" the dataset to match those assumptions, potentially disregarding some interesting knowledge that could have been discovered from the original dataset.

Monotonic attribute selection

Hu et al. propose the use of the margin-based attribute selection algorithms O-ReliefF and O-Simba [31]. They incorporate pre-determined monotonicity constraints into the margin-based attribute selection algorithms ReliefF [57] and Simba [17] in order to create algorithms that select the optimal set of attributes for the task of monotonic classification. They test the proposed algorithms on a combination of synthetic datasets and 11 datasets from the UCI repository and

run experiments using the C4.5 classification algorithm. The synthetic dataset experiments demonstrated in the paper show that the proposed algorithm is indeed capable of consistently assigning higher weights to the attribute that has a strong monotonic relation with the class. It is, however, important to note that the synthetic datasets used in these experiments are highly tailored to the specific task of attribute selection, only containing predictor attributes that are strongly monotonic with the class and predictor attributes consisting of uniform random values. No weak monotonic relations are used in synthetic data tests. The C4.5 algorithm used in classification experiments is not inherently monotonic, so there is no guarantee that the models and predictions produced by it will be fully monotonic. The monotonicity of classification predictions is not reported in the paper and the only indication of real improvement brought about by introducing monotonicity constraints into attribute selection process is a mostly consistent improvement in predictive accuracy.

The min-Redundancy Max-Relevance (mRMR) algorithm for monotonic attribute selection was proposed, also by Hu et al [30]. In this algorithm, the authors have used an attribute selection algorithm based on the principle of minimising redundancy among the selected attributes and maximising the relevance of the selected attributes for predicting the class, and introduced the Rank Mutual Information (RMI) metric into its search strategy. The resulting algorithm was evaluated on 14 classification problems using two monotonic classification algorithms. The experimental methodology in this study seems problematic. Firstly, the RMI metric is used as a monotonicity measurement metric, despite more monotonicity-oriented metrics being available such as the Spearman correlation coefficient [23] or the Non-Monotonicity Index (NMI) [6]. Secondly, some of the datasets used in the experiments are way too small for a conclusive evaluation of the attribute selection algorithm, with half of the datasets having only around 10-15 attributes and one dataset having 8 attributes. Thirdly, despite providing a

large amount of data about the classification experiment results and monotonicity metrics of individual attributes, the study does not report on how monotonic each dataset was as a whole and how much its monotonicity changed after the monotonic attribute selection was applied, thus no conclusion about the monotonicity of the resulting models can be made. Finally, the study proposes and evaluates a novel algorithm for attribute selection but does not provide any metrics regarding the amount of attributes that were selected in the experiments.

The attribute selection studies ultimately suffer the same drawbacks as the monotonic relabeling studies. They use monotonicity assumptions without evaluating their correctness using real data, they manipulate the dataset to adhere to these questionable assumptions and they fail to report any numerical measurements of monotonicity of the original datasets and attribute-selected subsets. In addition, the studies listed above do not specify that such dataset pre-processing has been applied to the training sets only, so it is possible that such data manipulation has unduly affected the predictive accuracy measurements by simply accessing information from the test set when selecting the features – an invalid methodology in machine learning.

Monotonic instance selection

One of the reasons why relabeling studies were conducted was the non-compliance of real-world data with the assumed monotonic relations. In order to construct classification models that adhear to the monotonicity assumptions, they have proposed ways to alter the datasets in order to achieve a higher degree of monotonicity in the training data. One interesting study has approached this problem differently. Instead of altering class labels of the dataset instances, this study used an algorithm that selects a subset of training instances based on their adherence to the monotonicity constraints, thus ensuring that the model is trained on a highly-monotonic dataset.

Cano et al. have proposed a Monotonic Training Set Selection (MonTSS) algorithm that can be used to reduce the size of the training sets used for monotonic classification tasks [12]. MonTSS starts by estimating the correlation between each predictor attribute and the class attribute using Rank Mutual Information (RMI) metric and then applies a probabilistic method for iteratively removing some of the training instances that cause monotonicity violations. The selection process for these instances is stochastic, with each instance having a chance to be removed proportional to the number of monotonic violations it causes. The removal probabilities are re-calculated after each removal in an attempt to prioritise the most conflicting instances and attempt to minimise the total number of removals. After some of the monotonicity-violating instances have been removed, the algorithm starts searching for useful instances to return back to the training set. It achieves this by searching through the removed instances and bringing back the ones located closest to the class boundaries. The algorithm therefore arrives at a training set that preserves both the monotonic relations and the class-defining features present in the original dataset. The authors provide an extensive evaluation of the algorithm, using 4 monotonic classification algorithms to produce models for 30 classification problems and reporting the predictive accuracy measures achieved as well as measures of monotonicity and size of the training sets used for model construction. The resulting classification models consistently improved the performance of classification models while significantly reducing the size of the training sets.

To the best of my knowledge, MonTSS is the first and only instance selection algorithm for monotonic classification in the literature. It manages to successfully integrate the monotonicity constraints into the process of instance selection, resulting in an algorithm that can both improve the monotonicity of the constructed models and improve their predictive accuracy. The authors have provided a very good level of analysis of the algorithm performance, much better than the majority

of studies in the field of monotonic pre-processing.

3.1.2 Monotonic classification algorithms

In addition to monotonic dataset pre-processing techniques, many studies have proposed new classification algorithms designed specifically for the task of monotonic classification. The majority of such algorithms use already existing classification algorithms and add extensions to allow models to preserve monotonic relations present in the data.

The first known monotonic classification algorithm was published in 1989 by Ben-David et al.[5]. In this work, an instance-based learning approach was used to create a classification model and monotonic constraints were used as part of the model construction and class prediction process. The approach used a Consistent and Irredundant Set of Examples (CISE) to hold all learned instances. The algorithm starts by creating an empty set of instances, empty CISE. During the learning stage, new instances were added to CISE provided they were consistent and not redundant. The algorithm considers "consistent" all instances that do not violate the monotonicity constraints. It considers a new instance redundant if it does not provide any additional information for classification, e.g. an instance that is identical to one already present in CISE. Predictions were made by comparing the instance whose class is being predicted to the instances already learned by the model and deciding on the class based on the monotonic constraints. The approach had multiple drawbacks, such as only being able to use ordinal attributes and having to discard all examples that were inconsistent with the monotonicity constraints.

Duivesteijn and Fielders proposed an algorithm for monotonic classification based on k-nearest neighbours (kNN) classification algorithm [21]. The proposed Monotone kNN (MkNN) approach uses a procedure for re-labeling the instances in the training set to ensure full monotonicity before the kNN model is created.

The algorithm used a monotonicity violation graph (MVG) as a representation of all instances within the training set and the ordinal relations between them. The graph is then used to determine the optimal set of instances to be re-labeled and the relabeling step is done. Authors argue that since the kNN algorithm uses the training data as classification model, monotonising the training data at the pre-processing step is sufficient to produce a fully monotonic classification algorithm. Overall, monotonicity enforcement has improved the predictive accuracy of the kNN classifiers and a positive effect on model acceptability was suggested.

Potharst and Bioch proposed a decision tree algorithm that creates a fully monotonic decision tree for multiclass datasets [52]. The proposed algorithm uses a standard tree generation algorithm (C4.5 or CART) and resolves non-monotonic splits in a post-processing stage. The algorithm uses a process called cornering, which expands the potentially non-monotonic nodes of the tree by adding new splits that ensure the monotonicity of the split in relation to other attributes. The proposed method can also be used as a "tree repairing" tool for ensuring monotonicity in decision trees produced by other algorithms.

Verbeke at al. proposed a post-processing algorithm to create monotonic decision rule and tree-based models [62]. The proposed RUle LEarning of ordinal classification with Monotonicity constraints (RULEM) algorithm can guarantee full monotonicity of the constructed model by applying post-processing changes to any model constructed by a decision tree or rule induction classification algorithms. It evaluates the monotonicity of a rule set by generating a decision grid based on the rules and evaluating the Conflict score (C-score) of each cell on the grid to detect monotonicity violations. It then adds complementary rules to the rule set to resolve the monotonicity violations. The results of the experiments indicate that the proposed approach preserves the predictive power of the original rule induction techniques while guaranteeing monotone classification, at the cost of a small increase in the size of the rule set.

An Ant Colony Optimisation (ACO)-based approach for monotonic classification was proposed by Brookhouse and Otero [11]. The proposed algorithm uses an ACO-based approach to generate lists of monotonic classification rules using a modification of Ant-Miner [49] classification algorithm. The proposed classifier, named cAnt-MinerPB+MC, allows for soft monotonicity constraints to be enforced using a tree pruning approach based on a weighted quality function that allows monotonicity-violating pairs of rules to remain in the model as long as the increase in predictive accuracy outweighs the decrease in monotonicity of the model. Weights can be adjusted to allow for more monotonic or accurate models to be constructed. Hard monotonicity can also be enforced using post-processing pruners, which guarantee that the final rule model is fully monotonic.

A neural network-based monotonic classification algorithm, called Monotonic Classification Extreme Learning Machine (MCELM), was proposed by Zhu et al. [67]. The proposed algorithm is an extension of Extreme Learning Machine (ELM) [32] — a single hidden layer feed-forward neural network learning algorithm. The proposed algorithm approaches the classification model construction task as a quadratic programming problem, where the training error rate is used as the objective for optimisation. The monotonicity constraints are used to restrict the signs of partial derivatives of the decision attributes with respect to the features in order to restrict the impact that the data attribute values can have on the class prediction. The algorithm can therefore ensure that the model produced by MCELM is fully monotonic and does not require training data to be fully monotonic.

A similar approach was taken by Chen and Li [13] using an SVM-based classification algorithm. In this study, a Monotonicity Constrained Support Vector Machine (MC-SVM) classification algorithm was proposed for monotonic classification of financial credit rating data. It uses quadratic programming approach in a similar fashion to MCELM and approaches the classifier construction as an

optimisation problem with pre-defined monotonicity constraints being added as conditions to the problem. This approach produces fully monotonic models but has a drawback of requiring a fully monotonic set of training data.

XGBoost is a tree boosting algorithm that has a built-in feature for monotonicity enforcement [15]. It constructs an ensemble of decision tree models with each subsequent model aiming to improve the overall predictive accuracy by fixing classification errors of the previous models. It uses strict monotonicity enforcement within its decision tree models, always ensuring that whenever a split is made using a monotonically-constrained attribute, the top and bottom prediction boundaries are set for the branches in such a way as to ensure that the branch corresponding to the lower attribute values will always make predictions lower than the branch corresponding to higher attribute values. Since XGBoost is a regression algorithm, tree leaves make numerical predictions based on class frequencies, so it can add thresholds to restrict how low and how high those prediction values can get. When the ensemble model needs to make a classification prediction, it performs "classification-by-regression", essentially making a prediction like a regression model would and then converting it into the closest available class value. The algorithm itself does not include any features for detecting monotonic relations in data but allows for monotonic constraints to be specified for each predictor attribute.

As shown above, there is a large variety of monotonic classification algorithms that have been proposed over the years. Many different types of classification models have been adapted for use with monotonic data. Some algorithms, such as Monotonic Trees use simpler and more interpretable models to construct initial classification models and then make monotonic modifications in a post-processing stage. Other algorithms, such as MkNN focus instead on pre-processing the data in order to construct a fully monotonic classification model. There are also algorithms, such as cAnt-MinerPB+MC that use a more adaptive approach —

constructing the model with soft monotonicity enforcement and allowing for the model to be made fully monotonic in post-processing. This later approach appears to be more pragmatic and more practical.

It is important to note that none of the studies listed above have implemented monotonicity detection as part of their algorithms, instead relying on domain knowledge as a source of monotonicity constraints. This approach seems a bit questionable in context of classification as while it does not technically use any information that would invalidate the experiment results, like many monotonic pre-processing studies do by using information from the full dataset instead of just from the training set, it introduces additional information based on experts' domain knowledge. In an ideal scenario, provided that this additional information is correct, it would already be present in the dataset itself, so a thorough monotonic classification algorithm would be able to learn it on its own. Instead, all monotonic classification studies rely solely on domain knowledge for these constraints, making these algorithms scarcely applicable for experiments with new, previously unexplored datasets.

3.2 Longitudinal Classification

Longitudinal datasets contain repeated measurements of the same attributes taken at different points in time. Over the past few years there have been several studies focused on classifying such datasets, thus the field of Longitudinal classification was formed.

Despite the growing importance of the longitudinal data and an increasing number of longitudinal approaches, there is no singular complete taxonomy for longitudinal approaches. Two notable attempts at categorising longitudinal classification approaches have been made.

Jie et al. [37] proposed a classification of longitudinal learning problems based

on the number of time points in the input (the data) and in the output (the learned models): (1) Single-time-point Input and Single-time-point Output (SISO); (2) Single-time-point Input and Multi-time-points Output (SIMO); (3) Multi-time-points Input and Single-time-point Output (MISO); and (4) Multi-time-points Input and Multi-time-points Output (MIMO). These 4 types broadly cover all types of longitudinal learning problems and they can effectively be used as short descriptors for problem types. However, to the best of my knowledge, there are no examples of SISO and SIMO approaches in the longitudinal learning literature and the SISO approach can effectively refer to any non-longitudinal learning task.

More recently, a survey of longitudinal approaches applied to machine learning (or data mining) by Ribeiro and Freitas [56] proposed a new taxonomy that categorises all longitudinal approaches in two dimensions. The first dimension broadly categorises approaches based on how they handle the longitudinal task:

- **Data transformation** approaches aim to transform the original longitudinal dataset to a simpler non-longitudinal dataset to allow conventional machine learning (classification or regression) algorithms to be applied to it;
- **Algorithm adaptation** approaches transform the conventional machine learning algorithms to handle the original longitudinal dataset and build models that take advantage of the longitudinal nature of the data.

  The second dimension categorises approaches based on their strategy to transform the longitudinal data:

  - **SepWav** approach uses each wave as a separate entry for the learning algorithm;
  - **AggrFunc** approach uses aggregation functions to summarise the data values of attributes across all waves;
  - **MerWav-Time(+)** approach merges data from all waves into a single combined dataset while still keeping record of attribute time indexes;

– **MerWav-Time(-)** approach merges data from all waves into a single combined dataset omitting time indexes.

While the first dimension seems to be the most logical aspect to differentiating longitudinal machine learning approaches, the second one is a lot less intuitive. Despite the fact that only the data transformation category suggests any changes being made to the raw input dataset, the taxonomy contains examples of different transformations being used with algorithm adaptation approach. The MerWav-Time(+) approach seems to dominate the algorithm adaptation category, being listed as data transformation method in all algorithm adaptation examples. Also, in the second dimension, the taxonomy only considers the different transformation techniques for longitudinal features but does not explicitly discuss whether any transformations need to be made to the non-longitudinal features and class features over different waves.

3.2.1 Longitudinal data transformation approaches

A study by Mo et al. [42] proposed one of the simplest approaches to longitudinal data classification, by using a set of non-longitudinal classification algorithms on the data constructed using MerWav-Time(-) feature representation. By removing the longitudinal aspect of the data and using non-longitudinal data mining algorithms, the study trivialised the problem of longitudinal prediction by turning it into a standard classification problem, thus no algorithm adaptation was needed. Five standard classification algorithms were used to construct and evaluate predictive models using a dataset provided by the Alzheimer's Disease Neuroimaging Initiative (ADNI). While other studies commonly aim to build a prediction model to predict the value of the class attribute at the final wave (time point), this study used the second wave to predict the class values on the first wave, thus further simplifying the task. Overall, one can argue that an approach like this may not be considered longitudinal at all, since it uses non-longitudinal algorithms on a

dataset that has no longitudinal information in it, except for the fact that it is a result of a transformation of an originally longitudinal dataset with two waves.

More recently, Zhang et al. [66] have conducted a similar study focused on longitudinal classification of the status of daily living activities among elderly population. The proposed approach used a standard decision tree classification algorithm using a dataset with two waves. In this application, the two waves were combined using MerWav-Time(-) approach and the class value of the first wave was predicted.

A study on longitudinal approaches to liver disease prediction by Niemann et al. [45] used a similar approach, using MerWav-Time(-) data transformation and applying simple classification algorithms such as naive bayes, decision trees and random forests. An additional step is taken during the pre-processing stage, where all data instances are clustered based on similarities in their attribute values and additional features are added. The additional features include cluster-based information such as the id of the cluster and the distance to the cluster centroid. The clustering took place separately on each wave before the MerWav-Time(-) was applied, thus recording separate clustering features for each wave. While being similar to the the approach by Mo et al. [42], this study utilised the longitudinal nature of the data in a very interesting way by adding time point-specific features before combining the waves using MerWav-Time(-).

3.2.2 Longitudinal classification algorithms

A study by Chen and Bowman [14] explored the use of Support Vector Machines (SVMs) classifiers in longitudinal data mining with application to neuroimaging data. The study proposed a Longitudinal Support Vector Classifier (LSVC) for longitudinal classification. It used an algorithm adaptation approach to create a novel SVM-based algorithm and the MerWav-Time(+) approach for data representation. The study also briefly described the feature selection and parameter

tuning aspects of the proposed algorithm. The novel algorithm has been evaluated using neuroimaging data and significantly outperformed the conventional SVM-based classifier in terms of predictive accuracy.

A further study was published by Du et al. [20], which proposed an extension to the LSVC algorithm proposed by Chen and Bowman [14]. The proposed Longitudinal Support Vector Regression (LSVR) algorithm is a generalisation on the original LSVC algorithm that allows it to be used in longitudinal regression tasks. In this study, similarly to the previous study, the data from different waves is combined using a MerWav-Time(+) approach where all data points are united in a single dataset with time indexes being recorded for each attribute value. In one set of experiments, the mean values of the longitudinal attributes are used instead of original values, thus utilising a variation of the AggrFunc approach as well. Overall, the experiments conducted on the DREAM-Phil Bowen ALS Prediction Prize4Life dataset [26] have shown an improvement in predictive accuracy using the LSVR approach as compared to non-longitudinal approaches. However, the experiments with the use of longitudinal attributes generated using the AggrFunc approach have shown a consistent increase in error rate across the majority of experiments.

A study by Minhas et al. [60] also used an SVM-based approach to longitudinal classification. Unlike the two previous approaches, this study used a standard version of an SVM classification algorithm and focused on data preparation and data transformation, in particular on the issues related to missing values in the original dataset and the task of feature selection. Two main data transformation methods were used in this study: SepWav was used in a set of experiments where only the feature values from the first wave were used for training; AggrFunc was used in a set of experiments where the mean and median values of the features across the waves were used. A further study [61] was later published by the same authors focusing on the practical application of the proposed methodology using

a longitudinal dataset on the progression from Mild Cognitive Impairment (MCI) to Alzheimer's Disease (AD), provided by Alzheimer's Disease Neuroimaging Initiative (ADNI).

A study by Huang et al. [33] used an interesting approach to longitudinal regression based on consecutive prediction of class values in subsequent waves using a random forest regression algorithm applied to the task of Alzheimer's Disease clinical scores prediction. The proposed approach was used to predict the score in a wave using data from the current wave and all previous waves. The waves were combined using the MerWav-Time(+) approach. A separate set of experiments was used for each wave and the results of predictions for the previous wave were incorporated into the predictor attributes for the next wave. The proposed algorithm outperformed a standard random forest algorithm and other popular regression algorithms such as Lasso and SVM.

3.2.3 Longitudinal Feature Selection and Construction

Pomsuwan and Freitas [51] presented a study centered around the task of feature selection for longitudinal classification, exploring the effects of adding new predictive attributes based on time differences and general trends of longitudinal attributes. The waves are combined using the MerWav-Time(+) approach and an AggrFunc approach is used to create additional attributes representing differences between the values of longitudinal attributes at different waves, binary attributes representing the direction of the change between waves (increasing or decreasing) and a categorical attribute representing whether the values have consistently increased or consistently decreased or neither. The experiments using data from the English Longitudinal Study of Ageing (ELSA) database have shown an improvement in predictive accuracy when the proposed feature selection method was used with a simple Naive-Bayes classification algorithm. This study is an important

case for the AggrFunc approach in general since it shows that the use of AggrFunc attributes in conjunction with a suitable feature selection algorithm can be used to improve the predictive accuracy of the models. Previous examples of AggrFunc approaches generally have shown negative effects on predictive accuracy when using AggrFunc attributes [11].

3.2.4 Longitudinal classification criticism

As shown above, there are two main approaches to longitudinal classification. The first approach uses data flattening to remove the longitudinal aspect from the dataset and uses non-longitudinal classification algorithms to construct classification models. There is an obvious issue with this approach — the removal of longitudinal aspect from the dataset prevents the classification model from discovering any interesting longitudinal trends that may have been present before flattening. Even though the studies and taxonomies generally refer to such approaches as instances of longitudinal classification, the models that are constructed in this way are not longitudinal. More importantly, the models constructed using flattened datasets rely on values of specific attributes measured at specific time indexes to make their class predictions. This means that if an existing longitudinal dataset was augmented with new data measured at a new time point, previously constructed models would not be able to take this new data into account when making predictions. Therefore, even if a highly accurate model was constructed and tested using a flattened longitudinal dataset, it would not be useful for making future class predictions when new temporal values get recorded.

The second approach works by applying modifications to existing classification algorithms, practically creating new classification algorithms that are inherently aware of the longitudinal nature of the data and can produce models that can be used to make class predictions on longitudinal datasets. Additionally, some of these models do not rely on the dataset having a specific number of time points

and can easily scale up to incorporate new time indexes without having to be re-trained. This distinction makes the second approach much more applicable to real longitudinal studies where new data is added regularly and it may be useful to periodically make new class predictions based on this new data.

An issue that is present in practically all longitudinal classification studies is the lack of longitudinal interpretability of the classification models. While some models might provide some degree of interpretability by providing measures of attribute importance or allowing for the models to be inspected manually, there is currently no example in the literature of any longitudinal classification algorithms providing a measure of attribute importance that would apply to a longitudinal attribute as a whole, not just to its individual temporal measurements.

3.3 Accessible Algorithms and Reproducible Results

In this section I would like to briefly go over the issue of reproducibility of results and accessibility of algorithms in the fields of monotonic and longitudinal classification. In order for the field to continue growing and for new classification models to be developed, it is extremely important that previously developed algorithms are made available to the research community, preferably in the form of source code, datasets, and additional instructions for reproducing the results reported in the studies. However, both fields suffer from a severe lack of both reproducibility of results and availability of the algorithm implementations.

The 2019 monotonic classification review paper [4] provides useful information on algorithm availability in that field. Out of the 64 approaches listed in that review only 9 are highlighted as being available in the form of source code. Out of those 9 that are available, some have very limited availability. For example, two algorithms OLM[7] and OSDL[40] are highlighted as being available as part of the

Weka software package [34, 63]. However, these two algorithms are not present in the main GUI version of Weka and can only be accessed programmatically by using the Weka library for Java. Perhaps the most accessible algorithm for monotonic classification is XGBoost[15] — a decision tree boosting algorithm popularised by online machine learning resources such as Kaggle [28]. Algorithm accessibility situation is very similar in the Longitudinal classification field, with almost no source code being available for any of the published algorithms.

3.4 Literature Review Summary

There is a large variety of classification approaches that can be applied to longitudinal and monotonic datasets. Some use simplistic data pre-processing algorithms to make standard classification algorithms applicable to these types of data, while others use sophisticated algorithms to utilise the unique properties found in these datasets and construct powerful and accurate classification models. Some problems, however, remain unsolved in the longitudinal and monotonic classification literature.

Firstly, there has not been a single classification study that combined the two fields. Longitudinal datasets have an additional temporal dimension in their data structure and can most likely benefit from the introduction of monotonicity-based classification features. A classification algorithm that can utilise the longitudinal nature of the datasets while also introducing monotonicity estimations would most likely be able to produce much more accurate and acceptable classification models than the ones produced by non-monotonic longitudinal classification algorithms.

Secondly, all monotonic classification relies on domain knowledge as a source of information about the monotonic relations in data [43]. All monotonic relabeling, feature selection and classification algorithms require the user to supply the monotonicity constraints as parameters to the algorithm. In theory, there

is nothing wrong with domain experts providing domain knowledge to the algorithm. Classification is a supervised machine learning task, and such inputs can be used to guide the algorithm towards a better solution. However, as mentioned before, this severely limits the applicability of the algorithms to new and previously unexplored datasets, where domain knowledge has not yet been acquired. Additionally, it allows domain experts to introduce biases that may or may not be consistent with the real world data. It would be useful to also have algorithms that can detect monotonic relations in the datasets. If the monotonic relations claimed by the experts are accurate, then a sufficiently powerful algorithm would be able to detect them from the data, without needing any guidance from domain experts. However, if the algorithm detects monotonic relations that were previously unknown or even contradicted the assumptions of domain experts — it may give the users an insight into the nature of the data and provide real knowledge discovery value. Monotonic relations detected automatically are not biased by experts' knowledge, as they are solely based on the trends observed in the real data.

Thirdly, the models learned by longitudinal classification algorithms are scarcely interpretable. All of the algorithms that have been adapted for longitudinal classification produce complex models that can not be easily inspected by humans. For a model to be interpretable, it should be easy to inspect individual elements of the model, trace how each classification decision is made, see which attributes are being used, ideally, without having to use complicated visualisation tools or rely solely on derived metrics. Additionally, since these algorithms use longitudinal datasets, it would be useful to be able to see the impact of each longitudinal attribute as a whole, trace where exactly its temporal values are being used in classification and which other longitudinal attributes tend to be used alongside it. At the moment, none of the longitudinal classification algorithms offer any of these features.

Finally, there is a severe lack of accessible algorithms in both longitudinal and monotonic classification fields. While a lot of authors write about their highly advanced algorithm that achieves great classification performance, very few actually publish the algorithms themselves. If a researcher wanted to make modifications to an existing longitudinal classification algorithm, they would have to either attempt to get the codebase from the original author or re-create the algorithm based on the descriptions provided in the paper, which are often not sufficient for a full re-creation of the results. Some papers provide very high-level descriptions of the algorithms, never even mentioning any aspects of the real algorithm implementation, such as the programming language used, libraries, data structures, dataset file formats, optimisation techniques, etc. It would be very useful to have these algorithms available in the form of source code along with at least a minimal set of instructions for reproducing results reported in the papers.

Another significant limiting factor for the field of longitudinal classification is a distinct lack of accessible pre-processed longitudinal datasets that could easily be accessed and used to evaluate longitudinal classification algorithms. Most longitudinal classification publications do not publish the datasets used in evaluation, severely limiting both the reproducibility of their results, and the ability of researchers to comprehensively compare their new approaches to the ones already present in the literature. This has had a major impact on the work described in this thesis, with a limited evaluation being conducted due to the lack of publicly-available pre-processed longitudinal classification datasets.

In conclusion, a lot can be done to improve the fields of longitudinal and monotonic classification. An automated monotonicity detection approach needs to be developed to address the issue of all monotonic classification algorithms requiring domain knowledge. An interpretable longitudinal classification algorithm needs to be developed to allow researchers to gain insight into the inner workings of a longitudinal classification model. An algorithm needs to be developed that would

be applicable to the longitudinal data and utilise monotonic relations that can be found there. A publicly available, well documented, customisable codebase needs to be published to allow researchers to experiment with both longitudinal and monotonic classification techniques without having to re-implement entire algorithms themselves.

Chapter 4

Monotonicity Detection

This chapter will outline the first contribution of this thesis: an automated approach to monotonicity detection and enforcement applied to longitudinal classification problems. Firstly, an automated monotonicity detection approach is proposed, allowing monotonic classification models to generate monotonic constraints based on the data, without requiring additional domain knowledge from domain experts. Secondly, this will be the first instance of monotonic and longitudinal classification being used in the same study.

4.1 Automated Monotonicity Detection

The main purpose of automated monotonicity detection is to define monotonicity constraints that can be used by monotonic classification algorithms. This means that, given an input dataset, a monotonicity detection algorithm will evaluate the strength of monotonic relations between the class attribute and each of the predictor attributes and generate monotonic constraints accordingly. If a predictor attribute has a very strong positive monotonic relation with the class, the model should enforce a positive monotonic constraint on that attribute; if the monotonic relation is instead strongly negative, the constraint should be negative; and if

there is no strong monotonic relation with the class, the attribute should remain unconstrained.

The strength of the monotonic relations would need to be measured numerically, so a monotonicity metric would need to be selected. Since there is a requirement for the monotonic relation to be ”strong“ enough to be turned into a monotonic constraint, some threshold value needs to be defined that would distinguish between ”strong“ and ”weak“ monotonic relations.

4.2 Monotonicity Metrics

Two metrics were considered for monotonicity detection in datasets: Spearman Correlation Coefficient and Non-Monotonicity Index.

Spearman Correlation Coefficient [23] is a mathematical monotonicity metric that calculates the strength of monotonic relation between two data attributes. It is calculated by first converting the attribute values to rank values and then applying the Pearson correlation coefficient [9]. Pearson correlation coefficient between two attributes is defined as the ratio between their covariance and the product of their standard deviations.

$$Pearson_{X,Y} = \frac{cov(X,Y)}{\sigma_X \times \sigma_Y}$$

Non-Monotonicity Index (NMI) is a metric first defined in monotonic classification studies of Ben-David et al. [6]. For a given monotonic relation, NMI can be calculated as the proportion of data instance pairs that violate such relation, given by:

$$NMI_{X,Y} = \frac{\sum_{i=1}^{k}\sum_{j=1}^{k} non\_monotonic((x_i,y_i),(x_j,y_j))}{k \times (k-1)} \quad (1)$$

where the value of $non\_monotonic((x_i,y_i),(x_j,y_j))$ is 1 if the two attribute

pairs are violating the monotonic relation, and 0 if they do not.

There are a few minor differences in how the two metrics estimate the strength of monotonic relations between data attributes. For example, Spearman correlation coefficient is a signed metric, meaning that it can represent both positive and negative monotonic relations. Spearman value of 1 represents perfect positive monotonicity; -1 represents perfect negative monotonicity; and 0 represents no monotonicity at all. In its original definition, NMI is an unsigned metric that can only estimate the proportion of instance pairs that violate the given monotonic constraint. NMI value of 0 represents perfect monotonicity, as no instance pairs violate the proposed monotonic relation; 1 represents a complete lack of the proposed monotonic relation as all instance pairs violate it.

For a task of pure monotonicity detection, Spearman correlation coefficient has a minor disadvantage in being biased towards a more linear relationship between the ranked attribute values while NMI focuses solely on the monotonic relation between the attributes, regardless of their value distributions. For example, Fig 4.1 illustrates the values of two attributes being plotted on a graph. It is obvious here that the two attributes are perfectly monotonic: higher values of attribute X correspond to higher (or equal) values of attribute Y. NMI metric has no problem detecting this relation as fully monotonic. However, Spearman correlation coefficient considers this relation to only be 86.7% monotonic, due to its bias towards the linear relation between the ranks of the two variables.

This distinction can be very important when choosing the best monotonicity metric. In fact, variables X and Y can represent a very simple classification problem where X is the predictor attribute and Y is a binary class attribute. Since the two are perfectly monotonic, all that is needed for accurate classification is a single binary rule that classifies all data instances with $X \leq 10$ as belonging to class 1 and all instances with $X > 10$ as belonging to class 2.

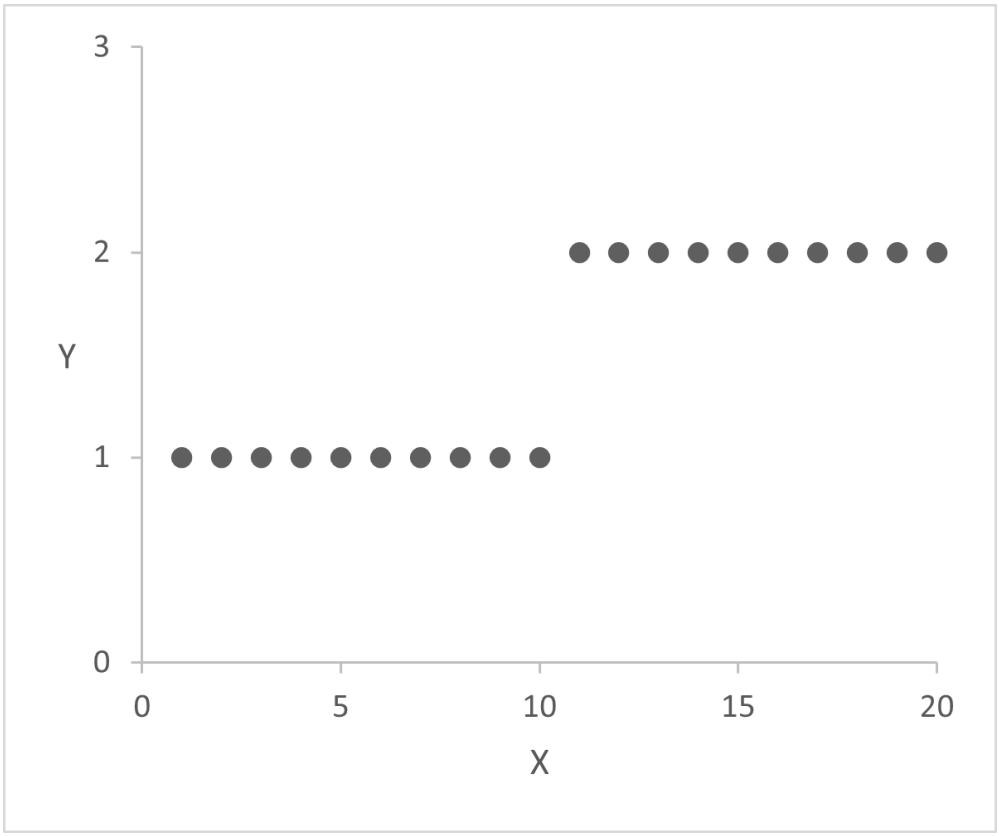

Figure 4.1: Fully monotonic relation between two attributes

Since NMI produces more reliable monotonicity estimations in context of classification, it was chosen as the metric for detecting monotonic relations in classification datasets and constructing monotonicity constraints.

4.3 Monotonicity Detection

A few minor adjustments need to be made in order to make NMI more applicable to detecting monotonic relations in real data.

Firstly, real data can contain missing values, so not all data instance pairs can be used to evaluate the strength of monotonic relations. The NMI calculation was adapted to only count instance pairs that were comparable, and data instances that contained missing values were excluded from the calculation. This treatment

of missing values does not make any assumptions about the sources of missing values and simply calculates the monotonicites using the values that are available.

Secondly, NMI is an unsigned metric that cannot represent negative monotonic relations. For monotonicity detection, NMI values were allowed to be negative in order to be able to represent both positive and negative monotonic relations without the need for additional variables. This was done purely for the sake of simplifying the programmatic implementation and did not change the nature of the NMI calculations in any way, practically using the sign bit of the floating point variable to represent whether the relation it represents was positive or negative, without any other changes.

Thirdly, NMI always assumes that a pair of examples is monotonic if the values of at least one attribute in a pair are equal. This causes no trouble on numeric attributes with barely any repeating values, but can have a huge impact on attributes with small domain space (e.g. binary attributes) and attributes that have many repeated values. Since NMI metric has to make an assumption that two examples with repeated attribute value are monotonic, it can not accurately estimate monotonicity when one or both attributes are largely dominated by a frequent repeated value. For example, a binary attribute that has 95% positive values and 5% negative values will always have very low NMI even when it is estimated against random noise.

To combat this final issue, I have proposed a new metric called Entropy-Adjusted Non-Monotonicity Index (EANMI). It uses an entropy-based coefficient to adjust the monotonicity measure and increase the degree of non-monotonicity for the attributes that are largely dominated by a single value (and thus have low entropy). The adjustment uses the entropy [16] of an attribute and the maximum entropy that the attribute can have, provided it has the same domain of values. In context of classification, the adjustment is only being made for the predictor attribute (X) and no adjustment was made for the class attribute (Y). The EANMI

is calculated as:

$$Entropy(X) = -\sum_{i=1}^{n} P(x_i) \times log_2 P(x_i) \tag{2}$$

$$Max\_Entropy(X) = log_2(n) \tag{3}$$

$$EANMI_{X,Y} = NMI_{X,Y} \times \frac{Entropy(X)}{Max\_Entropy(X)} \tag{4}$$

where n is the number of unique values of x, x_i is the ith unique value of x, $P(x_i)$ is the frequency of value x_i.

The proposed EANMI metric largely solves the third issue in cases where the non-class attribute is heavily dominated by a single value, reducing the degree of monotonicity measured by the NMI. It still suffers the same drawbacks as the non-adjusted NMI metric when the class attribute is also heavily dominated by a single value. It is, however, possible to resolve the issue of low entropy in the values in the class attribute by performing simple class balancing, as demonstrated in the next chapters.

4.4 Derived Monotonic Attributes

As mentioned before, this study applies a monotonic classification algorithm to a longitudinal dataset. Since there are no monotonic classification algorithms that can utilise the longitudinal aspect of the data, the longitudinal dataset would have to be flattened before the monotonic classification algorithm can be applied to it. As highlighted in the literature review, longitudinal classification studies that use flattening tend to produce classification models that have no longitudinal awareness and can scarcely be considered longitudinal. In order to preserve at least some of the longitudinal aspect in this study, additional predictor attributes

Table 4.1: Flattened longitudinal dataset

Id	**Glu_w1**	**Glu_w2**	**Glu_w3**	**Glu_w4**	**Diabetes_w4**
1	5.7	5.9	6.1	6.3	No
2	7.2	7.8	7.5	7.3	Yes
3	6.7	6.7	6.8	7.1	Yes
4	6.8	6.9	7.0	7.2	Yes
5	6.5	7.6	7.6	7.7	Yes

were constructed and added to the flattened dataset. These attributes would represent the longitudinal trends that are present in longitudinal attribute values of individual data instances.

The new attributes would be measured separately for each data instance and each longitudinal attribute. Table 6.1 shows a small flattened longitudinal dataset with five data instances representing five different medical patients, one longitudinal attribute with four time points representing four blood glucose measurements taken at different points in time, and a class attribute representing the diabetes diagnosis at the final time point. This is the same dataset that was used to illustrate data flattening in Chapter 2 of this thesis. To represent the longitudinal trends of the "Glu" attribute, a new predictor attribute is added to the dataset. The values of this new attribute represent the monotonic trend of the longitudinal attribute values across time. This means that, for example, for the data instance with Id 1, the value of the new attribute will be equal to the strength of the monotonic relation between attribute values $[5.7, 5.9, 6.1, 6.3]$ and time indexes $[1, 2, 3, 4]$. Table 4.2 shows the same dataset with the new attribute added. For datasets with multiple longitudinal attributes, the same process would be repeated separately for each longitudinal attribute. This process would only affect longitudinal attributes and no additional values would be constructed using any of the non-longitudinal attributes.

The values for this new attribute are estimated using the Spearman correlation coefficient. The reason for this choice in the sensitivity of Spearman coefficient to

Table 4.2: Flattened longitudinal dataset with a derived TIBI attribute

Id	**Glu_w1**	**Glu_w2**	**Glu_w3**	**Glu_w4**	**Glu_TIBI**	**Diabetes_w4**
1	5.7	5.9	6.1	6.3	1.0	No
2	7.2	7.8	7.5	7.3	0.2	Yes
3	6.7	6.7	6.8	7.1	0.948	Yes
4	6.8	6.9	7.0	7.2	1.0	Yes
5	6.5	7.6	7.6	7.7	0.948	Yes

repeated values and less linear relationships. This allows for the derived attribute to have a larger variety of values and may provide more predictive value than a more discreet NMI estimation.

The derived attribute is called a Time Index-based Individual (TIBI) attribute, since it estimates monotonicity of a longitudinal attribute values against time indexes and only uses values from an individual data instance. Deriving attributes based on data from only one data instance means that these attributes are safe to use in cross-validation experiments since they do not store any population-based trends and won't result in any data leakage from the test sets into the training sets.

Since real datasets may contain missing values, the derived attribute is implemented to only use time indexed values that are not missing. In cases where generating a TIBI attribute value is impossible, e.g. when a data instance is missing all of its longitudinal attribute values, the TIBI attribute is also allowed to have a missing value.

Since these attributes represent the strength of the monotonicity in the time-based trends of longitudinal attributes, it can be argued that classification models constructed using these attributes have some degree of longitudinal awareness and can therefore be considered longitudinal. The addition of TIBI attributes helps to partially solve the issue created by dataset flattening. Instead of completely losing all longitudinal trends as a result of flattening, one such trend is preserved per longitudinal attribute in the form of a derived TIBI attribute.

4.5 Summary of the Proposed Monotonicity Detection Approaches

The two proposed monotonicity detection approaches are the first contribution of this study.

The detection of monotonic relations between predictor attributes and the class attribute and construction of monotonicity constraints is referred to as **Timeless** monotonicity detection, since it does not take into account any longitudinal (time-based) information. The resulting monotonicity constraints can be used by monotonic classification algorithms to produce monotonic classification models. This allows for construction of monotonic classification models using only the dataset itself and without requiring additional domain knowledge from domain experts.

TIBI attributes are derived from each longitudinal attribute and used as additional predictor attributes for classification problems. This approach allows for classification models to be more longitudinally aware and uses a unique monotonicity-based method for creating derived predictor attributes.

It is important to note that when using the two approaches together, Timeless monotonicity detection will always take place after TIBI attributes are added, to allow those to also be monotonically constrained.

4.6 Experimental Setup

In order to evaluate the proposed monotonicity detection methods, a set of experiments was conducted using a longitudinal dataset with 10 classification problems and a popular classification algorithm that has a monotonicity enforcement feature.

4.6.1 ELSA datasets

The dataset used in these experiments was constructed using biomedical data from the English Longitudinal Study of Ageing (ELSA) [36]. It is a popular dataset for longitudinal data mining, used in a variety of studies [3]. Predictive attributes were created from "Nurse Visit" data portion of the ELSA database. Most attributes use raw values of ELSA attributes representing various patient health measurements such as blood test results and physical performance tests. There are 10 class attributes, each containing binary values representing either the presence or absence of a certain age-related disease in the final wave (Wave 8 in these experiments). A separate dataset was constructed for each class attribute. Since most of these attributes in ELSA are only recorded on even numbered waves, only the data from waves 2, 4, 6 and 8 was used. A full description of attributes used and their meaning can be found in a study [51] that previously used the same data preparation techniques using ELSA waves 2, 4, 6 and 7. The 10 datasets used for experiments each contained records for 7097 individuals participating in ELSA study. Each record contained 143 predictive attributes (44 longitudinal attributes, each having up to 4 time indexes, plus two non-longitudinal attributes), and a single class attribute.

Table 4.3 shows the frequency of positive class values for each of the 10 datasets created from ELSA data.

4.6.2 The XGBoost algorithm

As mentioned in the literature review, XGBoost [15] is a popular decision tree-boosting algorithm that has a monotonicity enforcement feature. It was chosen for this study for a number of reasons.

Firstly, it allows the user to provide monotonicity constraints as a construction parameter [27]. The constraints are represented as an array of integer values, same length as the number of predictor attributes, where a value of 1 represents

Table 4.3: Frequencies of positive class values

Class Attribute	Number of positive instances	Frequency
Angina	258	3.64%
Arthritis	3021	42.57%
Cataract	2322	32.72%
Dementia	148	2.09%
Diabetes	946	13.33%
HBP	2854	40.21%
Heart attack	401	5.65%
Osteoporosis	654	9.22%
Parkinsons	66	0.93%
Stroke	421	5.93%

positive monotonicity constraint, -1 represents negative monotonicity constraint, and 0 represents no constraint. This means that the EANMI values detected during the Timeless monotonicity detection can easily be turned into monotonicity constraints by applying a significance threshold to their values.

Secondly, XGBoost produces fully monotonic classification models. XGBoost is a classification-by-regression algorithm, meaning that it first builds a regression model and then converts its continuous regression predictions into discreet, categorical class predictions. While constructing decision tree nodes, it uses a system of upper and lower prediction boundaries to restrict the values of regression predictions in subsequent nodes. Whenever a decision tree node creates a split based on a monotonically-restricted attribute, the regression values of subsequent nodes get restricted accordingly [27]. This means that if an attribute has a positive monotonicity constraint, than the branch corresponding to the higher value of the attribute will be restricted to always have a higher regression prediction value than the branch corresponding to the lower attribute value. This ensures that the models produced by XGBoost are always fully monotonic and their predictions can never violate the monotonicity constraints.

Thirdly, XGBoost produces interpretable models to some extent and it has a built-in feature for measuring attribute importance in constructed models [25]. This means that after the models are constructed, they can be manually inspected by the user. This allows the user to see exactly how each classification decision is made and which attributes are used. The attribute importances feature provides a more concise summary of the most important data attributes and allows for even easier model interpretability.

Finally, XGBoost is one of the very few monotonic classification algorithms that has been made available to the research community [4]. Being the most up-to-date algorithm, that has many useful features for this kind of study, it was an obvious choice.

4.6.3 ELSA experiments with XGBoost

For each of the classification problems described in the previous section, a separate set of experiments was conducted, thus producing separate results for each class attribute used. Each set of experiments consisted of four separate experiments:

- One was performed on the raw dataset without any changes (base run);
- One experiment was performed with an addition of TIBI attributes representing monotonic trends in values of longitudinal attributes between waves. The TIBI attributes were detected using the Spearman's correlation coefficient and the raw values of Spearman were used;
- One experiment which included detection of Timeless monotonic relations using Entropy-Adjusted NMI (EANMI). Only the relations between the class attribute and all other attributes were used. The entropy adjustment was only made for the non-class attribute. The strong monotonic relations ($EANMI < 0.15$) were used as monotonicity constraints for the XGBoost algorithm;

- The final experiment introduced both the TIBI attributes and Timeless monotonic constraints. It is important to note that the Timeless constraints were introduced after the TIBI attributes were added, thus allowing those attributes to be considered for constraints as well.

During each experiment, a cross validation approach was used to estimate the average predictive performance of the XGBoost model built using the provided dataset and monotonicity constraints, if any were present. Ten-fold cross validation was used here to construct 10 models and estimate their average accuracy. In order to get a reliable estimation of average predictive accuracy of the models, the cross validation process was repeated 30 times (using different random seeds for cross validation) and the average predictive accuracy was measured. The predictive accuracy measure used in model training and evaluation was area under ROC curve ($\overline{\text{AUROC}}$).

4.7 Experimental Results

4.7.1 Predictive Accuracies of Constructed Models

The results for the four sets of experiments over the ten classification problems were collected. Table 4.4 shows the average area under ROC curve ($\overline{\text{AUROC}}$) values for each set of experiments. In addition, the average rank value for each monotonicity type combination is included.

From Table 4.4 we can see that the introduction of monotonicity constraints led to an improvement of the AUROC or had no substantial negative effect across all classification problems. Both approaches enforcing monotonicity constraints (Timeless and TIBI+Timeless) achieved the best rankings. The use of monotonicity-based TIBI attributes alone had no significant effects.

Further statistical analysis was performed on the $\overline{\text{AUROC}}$ rankings, where Table 4.5 shows the results of the non-parametric Friedman test [58] with the

Table 4.4: $\overline{\text{AUROC}}$ values for each set of experiments. The highest $\overline{\text{AUROC}}$ value for each dataset is shown in bold.

Class Attribute	Baseline	Timeless	TIBI	TIBI+Timeless
Angina	0.545	**0.682**	0.660	0.668
Arthritis	0.610	0.610	0.610	**0.611**
Cataract	**0.655**	**0.655**	**0.655**	**0.655**
Dementia	0.755	**0.767**	0.750	0.758
Diabetes	0.825	**0.829**	0.824	**0.829**
High Blood Pressure	**0.704**	**0.704**	**0.704**	**0.704**
Heart attack	0.679	0.696	0.688	**0.698**
Osteoporosis	0.666	**0.677**	0.664	0.670
Parkinson's	0.575	**0.621**	0.578	0.606
Stroke	0.666	**0.681**	0.673	0.680
Average Rank	3.3	1.65	3.2	1.85

Table 4.5: Results of the Friedman test with post-hoc Holm test. Statistically significant results at the 5% significance level are shown in bold, indicating that the result of a particular approach is statistically significantly worse than the control approach.

Approach	Average Rank	p-value	Holm
Timeless (Control)	1.65	-	-
Timeless+TIBI	1.85	0.729	0.05
TIBI	**3.2**	**0.007**	**0.025**
Baseline	**3.3**	**0.004**	**0.017**

Holm post-hoc test [24] — statistically significant differences at the 5% significance level are shown in bold. Both TIBI and Baseline approaches are statistically significantly worse than Timeless, the combination with the best ranking (control). It is important to note that while the table 4.4 reports the $\overline{\text{AUROC}}$ values rounded to 3 d.p., the statistical testing was performed using the raw numerical $\overline{\text{AUROC}}$ values produced by XGBoost, preserving the original ranking of the results.

Table 4.6: Average number [standard deviation] of Timeless monotonicity constraints used by XGBoost per classification problem. In all classification problems there were a total of 143 predictive attributes that could be monotonically constrained.

Class Attribute	Number of constraints
Angina	65.0 [0.50]
Arthritis	0.003 [0.005]
Cataract	0.0 [0.0]
Dementia	73.06 [0.50]
Diabetes	20.12 [0.39]
High Blood Pressure	0.0 [0.0]
Heart attack	61.89 [0.50]
Osteoporosis	39.42 [0.46]
Parkinson's	80.87 [0.49]
Stroke	61.61 [0.50]

4.7.2 Monotonicity Constraints and their Effects on Model Sizes

Monotonic relations were detected separately in each training set prior to the experiments and then used as monotonicity constraints during model construction. The results have shown that no monotonic relations were detected between TIBI attributes and the class in any of the TIBI experiments. Table 4.6 shows the average number of monotonicity constraints used by XGBoost for each of the classification problems as well as the corresponding standard deviation (over all training sets produced by cross-validation iterations with different random seeds) of each measurement.

It can be seen from the Table 4.6 that some classification problems used a large number of monotonicity constraints while others used none at all. Standard deviations remain very low for each measurement, meaning that there was only a small variation in number of detected constraints between different training sets.

The number of constraints can be correlated to the percentage of positive class

labels in each dataset as shown in Table 4.3. Generally, the classification problems that used class attributes with lower number of positive class labels had a larger number of monotonic constraints. For example, classes like Arthritis, Cataract and HBP were reasonably well balanced and almost no monotonic constraints were created for them by the Timeless monotonicity detection. Classes such as Angina, Dementia and Parkinsons were the most imbalanced and therefore had the highest number of monotonicity constraints. The reason for this trend may be related to the adjustment used by EANMI, which took into account the entropies of predictor attributes but not the entropy of the class. Therefore, class attributes with lower entropy generally had a high chance of having a strong monotonic relation ($EANMI < 0.15$) and thus had more monotonic constraints.

The average model sizes were measured for each of the experiments to determine the effect of introduction of monotonic approaches to the classification task. Overall, the introduction of both Timeless monotonicity constraints and TIBI attributes resulted in a small decrease in model size across all experiments. The decrease was consistent across all classification problems, yet not very significant: Introduction of Timeless constraints decreased the model size on average by 13% and introduction of TIBI attributes resulted in a 2% decrease in average model sizes. In the experiments where both approaches were used, the model size effects were similar to that of just using Timeless constraints.

4.7.3 Attribute Importance

The average attribute importance was estimated for each attribute in each experiment using the built-in attribute importance measure of XGBoost (based on average predictive accuracy increase) and the most important attributes were analysed. The results were sensible in most experiments: in Diabetes experiments the most important attributes were patient blood glucose levels; in HBP experiments the blood pressure attributes were most important; blood cholesterol levels had

high importance in the Angina, Heart Attack and Stroke experiments. In most experiments, the age attribute was among the top five most important attributes and the sex attribute generally had a high attribute importance in all models.

Interestingly, some of the TIBI attributes achieved very high attribute importance in some of the TIBI experiments. In the Cataract experiments, the TIBI attribute for diastolic blood pressure has achieved 4th highest attribute importance; the TIBI attribute for blood total cholesterol level had high importance in Heart Attack and Stroke TIBI experiments, and in the Heart Attack experiment with TIBI+Timeless approach it was the most important attribute. This means that in each of these classification problems, the attribute representing the monotonicity of the longitudinal trend of the corresponding longitudinal attribute had significant predictive power, overshadowing that of many of the other predictive attributes. Discoveries like these can provide valuable insight into the relationships between the changes in the temporal values of longitudinal attributes and the class, and can themselves provide significant knowledge discovery value.

Some categorical ordinal attributes were also highly important in classification models constructed using training sets with TIBI attributes. In the Angina experiments, the TIBI attribute for the outcome of side-by-side stand test was the attribute with the highest average importance; a TIBI attribute of a binary attribute representing whether the patient had any respiratory infection in the last 3 weeks was the 4th most important attribute in Heart Attack TIBI experiments.

Table 4.7 shows the attributes that achieved high attribute importance in the constructed models across all experiments for a given class. The attributes in bold were often detected as being monotonic with the class and were used as monotonic constraints in Timeless and TIBI+Timeless experiments. Since there was very little variation in the sets of monotonically restricted attributes, it is safe to say that these attributes were constrained in nearly all Timeless and TIBI+Timeless experiments for those classification problems.

Table 4.7: High-importance attributes for each classification problem. The attributes that were often constrained in the Timeless experiments are displayed in bold.

Class Attribute	High-importance attributes
Angina	**Age**, **Arterial pressure**, **Cholesterol level**, Chair rises∗, Side-by-side stands∗
Arthritis	Age, Chair raises∗, Leg raises∗, Main hand grip∗, Hip measurement
Cataract	Age, Leg raises with eyes open∗, Diastolic blood pressure
Dementia	**Age**, Clotting disorder, **Blood ferritin level**, **Grip strength***
Diabetes	Fasting blood glucose level, **LDL cholesterol level**, **Waist measurement**
High Blood Pressure	Age, Systolic blood pressure, Diastolic blood pressure
Heart attack	**Cholesterol level**, **LDL cholesterol level**, **Arterial pressure**
Osteoporosis	**Age**, **Main hand grip∗**, **Weight**
Parkinsons	Blood fibrinogen level, **White blood cell count**, **Non-dominant hand grip∗**
Stroke	**Age**, **Cholesterol level**, **LDL cholesterol level**,

∗ Outcomes of physical exercise tests

4.8 Conclusion

A set of experiments was conducted using the proposed monotonicity detection approaches and their effect on the predictive accuracies of the constructed classification models was evaluated. Enforcement of monotonicity constraints detected using the proposed Entropy Adjusted Non-Monotonicity Index was shown to generally result in an improvement of the predictive accuracy of the model without any significant negative effects. The addition of Time-Index Based Individual (TIBI) monotonic attributes was shown to have a very minor effect on the constructed models. TIBI attributes, despite being frequently used in some classification models, were not detected as being monotonic with the class attributes and thus have not been used as monotonicity constrained attributes in any of the

models.

This was the first study, to the best of my knowledge, where monotonic classification was applied to longitudinal data, creating the first intersection between the subfields of monotonic and longitudinal classification. Also, this was the first monotonic classification study where monotonicity constraints were detected automatically and the classification algorithm did not rely on information provided by domain experts.

Chapter 5

The Proposed Nested Trees Algorithm

This chapter will present the main contribution made to the field of longitudinal classification – a decision tree-based longitudinal classification algorithm. The proposed classification algorithm utilises a novel model structure, building a decision tree model where each node is used to analyse a separate longitudinal attribute. Inside of each node, the values of the longitudinal attribute are analysed using a smaller decision tree. As a result, the classification model looks like a decision tree that has small decision trees nested into it's nodes, hence the name Nested Trees. Since the proposed algorithm uses a highly-interpretable decision tree structure, it inherently has high model interpretability and its longitudinally-aware structure allows for a high degree of longitudinal interpretability. Nested Trees became the first longitudinal classification algorithm to produce interpretable models and the first longitudinal classification algorithm to have its software implementation released publicly.

5.1 Problems of Longitudinal Interpretability and Sparse Use of Longitudinal Values

As mentioned in the longitudinal classification part of the Background chapter, the majority of longitudinal classification studies utilise the flattening approach - separating temporal values of longitudinal attributes and treating them as independent non-longitudinal attributes. While this approach does allow conventional classification algorithms to be applied to longitudinal datasets, it severely impairs the potential value of the knowledge or patterns that could be extracted from the dataset. A major part of that impairment is the lack of longitudinal interpretability in the models constructed using the flattening approach.

To make classification models longitudinally interpretable, the classification algorithms have to provide some way of evaluating the attribute importance of longitudinal attributes. There are some conventional classification models that offer a high degree of attribute interpretability by measuring and reporting the importance of individual attributes and their impact on classification, but this offers very limited value for longitudinal interpretability since the flattening approach allows models to use temporal values of longitudinal attributes in an unrestricted way.

This issue is particularly noticeable in decision tree models. Since conventional decision trees consist of nodes that use values of specific attributes, it is easy to see how a model constructed using a flattened longitudinal dataset can have poor longitudinal interpretability. Figure 5.1 shows a small decision tree constructed using a flattened longitudinal dataset with two longitudinal attributes: alpha and beta, both having time indexes 1 and 2 available. The tree consists of a number of splits that use different temporal values of the two longitudinal attributes. Longitudinal attribute values are not grouped in any meaningful way - splits that utilise values of attribute alpha occur in many different parts of the tree, making

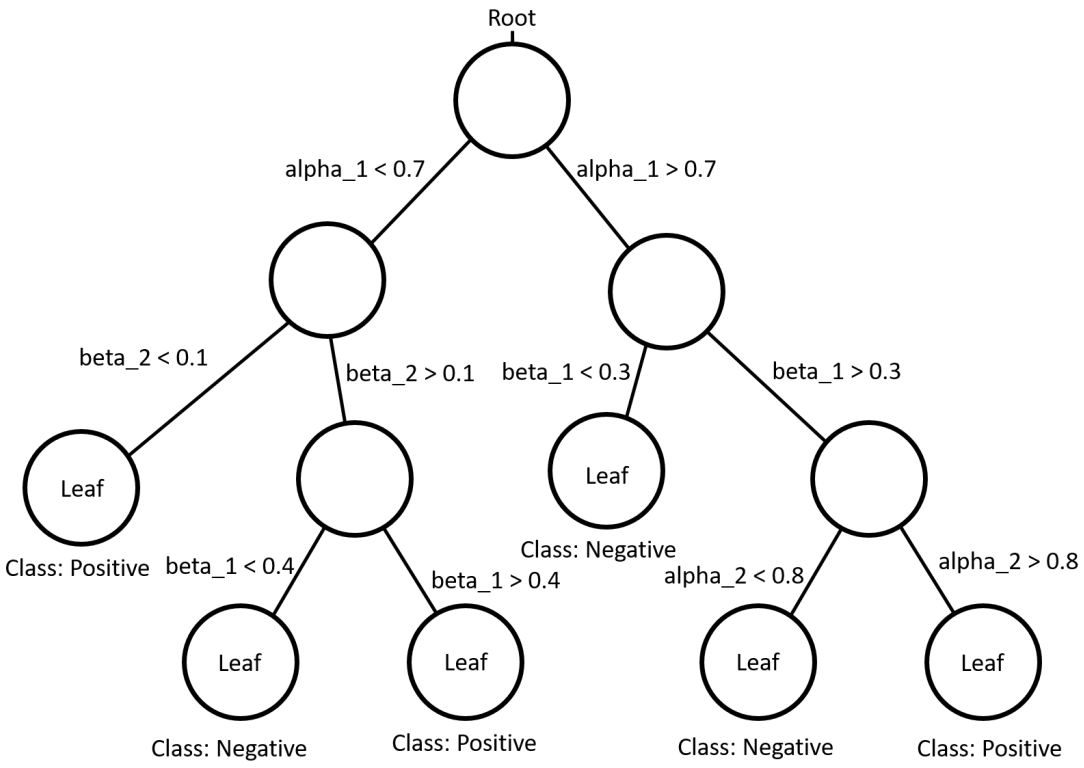

Figure 5.1: A decision tree constructed using a flattened longitudinal dataset.

it hard to evaluate the exact effect this particular longitudinal attribute has on the classification as a whole. This sparse use of longitudinal attributes in classification models is the main limiting factor to achieving longitudinal interpretability.

5.2 The Task of Creating a Novel Longitudinal Classification Algorithm

Before proposing a novel longitudinal classification algorithm, let us highlight a few important requirements for this type of algorithm.

- The new classification algorithm has to be applicable to longitudinal data. It needs to have inherent functionality for processing longitudinal datasets and

handling the temporal values of longitudinal attributes without separating them into independent attributes.

- The new classification algorithm has to construct classification models in a longitudinally-aware way, always analysing each longitudinal attribute as a whole and never considering an attribute's temporal values independently.
- Classification models constructed by the algorithm should be easily interpretable. When a model is constructed, it should be easy to manually inspect it and trace how each classification decision is made.
- The classification models should be longitudinally interpretable, allowing an easy way to gain insight into the effect each longitudinal attribute has on the classification.
- Parts of the classification model where the values of a longitudinal attribute are used should be concentrated to avoid the above mentioned issue of sparse use of temporal values of longitudinal attributes.

5.3 Overview of the Nested Trees Algorithm

The Nested Trees algorithm is a novel algorithm for longitudinal classification. It learns a classification model with two levels of decision trees. More precisely, it learns a model with an outer decision tree where each of its internal nodes consists of an inner decision tree learned by using different temporal values of the same longitudinal attribute. It always analyses each longitudinal attribute as a whole (as described below), and produces a classification model with a high degree of longitudinal interpretability. Additionally, the models constructed by the Nested Trees algorithm can be represented as conventional decision trees, providing a similar level of interpretability as decision tree models have.

The Nested Trees algorithm avoids the issue of sparse use of temporal longitudinal attribute values by constructing tree nodes that analyse a longitudinal attribute as a whole and make decisions based on a set of temporal values of the same longitudinal attribute instead of just one temporal value. More precisely, inside each internal (non-leaf) node of the outer tree, a separate decision tree is constructed using the temporal values of a longitudinal attribute, thus allowing each node to make decisions based on the full set of temporal values of that attribute.

In summary, the classification model is constructed as a decision tree where each internal node contains an embedded decision tree, hence the name Nested Trees.

5.4 Nested Trees Model Structure

As mentioned earlier, the Nested Trees model structure is a decision tree where each internal (non-leaf) node contains an embedded decision tree based on the temporal values of a single longitudinal attribute. Figure 5.2 shows an example of a Nested Trees model that uses three longitudinal attributes.

Figure 5.3 shows the root node of the same tree. The tree inside the node has 3 leaf nodes, thus splitting the local training set in that node into 3 subsets. The node has 3 branches corresponding to those 3 leaf nodes.

Classification using the Nested Trees model is performed in a similar way as with conventional decision trees. A data instance starts at the root node, has one of its attributes evaluated and gets sent down one of the branches, repeating recursively until a leaf node is reached and the classification value is acquired. When entering a new node of the outer tree, the instance is directed to the root node of its inner tree. When the instance reaches a leaf node of the inner tree, it is sent down the branch of the outer node corresponding to that leaf node.

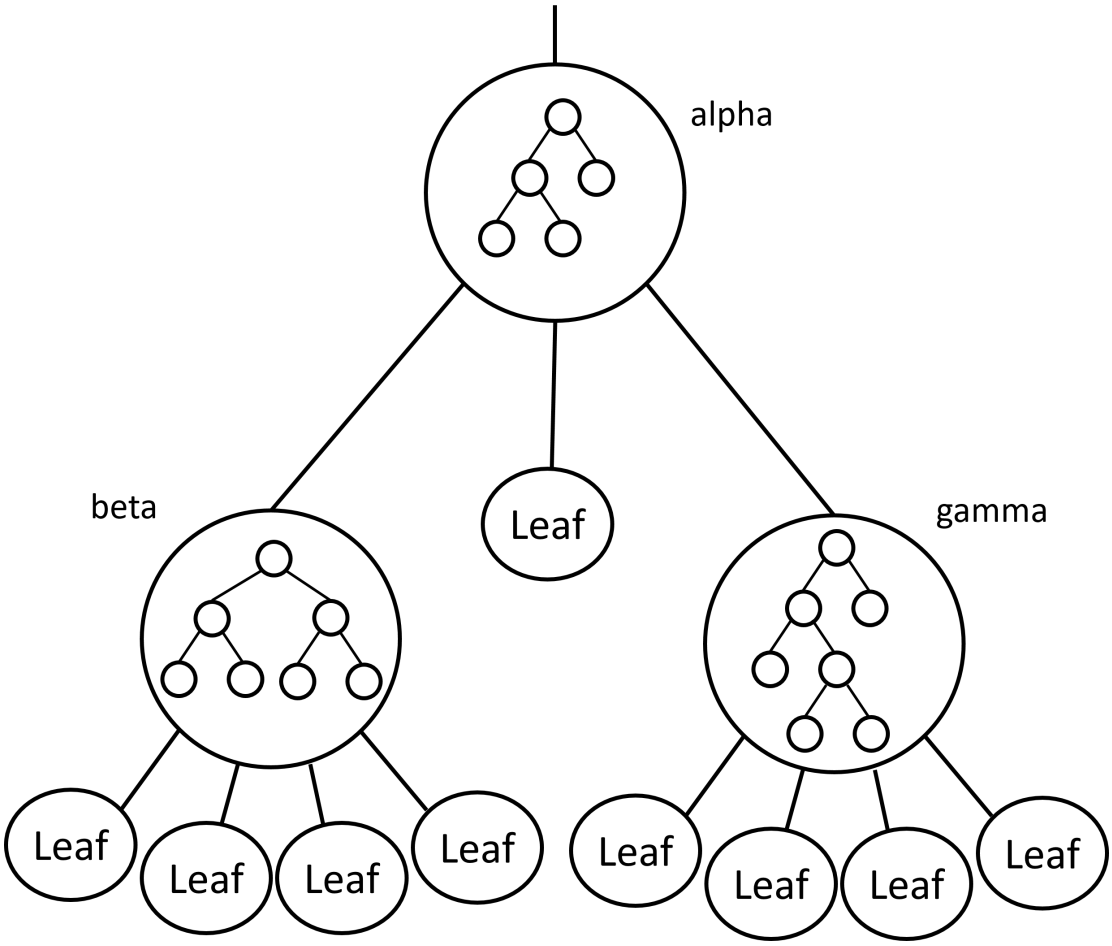

Figure 5.2: A Nested Tree model using three longitudinal attributes.

Figure 5.4 shows an example of an instance being classified by a Nested Trees model; and Figure 5.5 shows the mapping of the inner tree of a node to the branches of the outer node it is embedded into.

5.4.1 Nested Trees model construction

Nested Trees model construction is done on two layers. The outer layer constructs the outer decision tree, where each node uses the values of one longitudinal attribute at a time. The inner layer constructs the trees embedded into the nodes of the outer tree, using a full set of temporal values of the longitudinal attribute

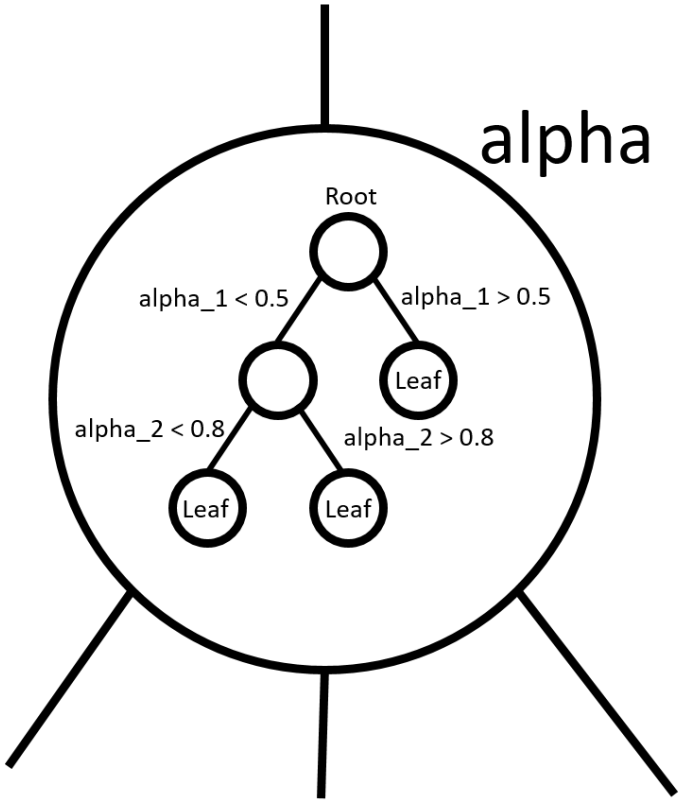

Figure 5.3: The root node of the model in Figure 5.2, with an embeded decision tree making decisions using the temporal values of attribute alpha.

the outer node has chosen.

5.4.2 Inner Layer

The inner layer construction process constructs small decision trees that use the longitudinal values of a single longitudinal attribute. These inner trees are constructed by a custom decision tree algorithm implementation which can be described as a hybrid of the well-known CART [39] and C4.5 [53] and J48 [35] algorithms. It constructs a tree made of binary splits, using a greedy approach based on the Gini impurity metric to select the attribute to split the data in the current node; a minimum node size requirement and a maximum depth limit are used as both the stopping criteria and the pre-pruning methods to mitigate

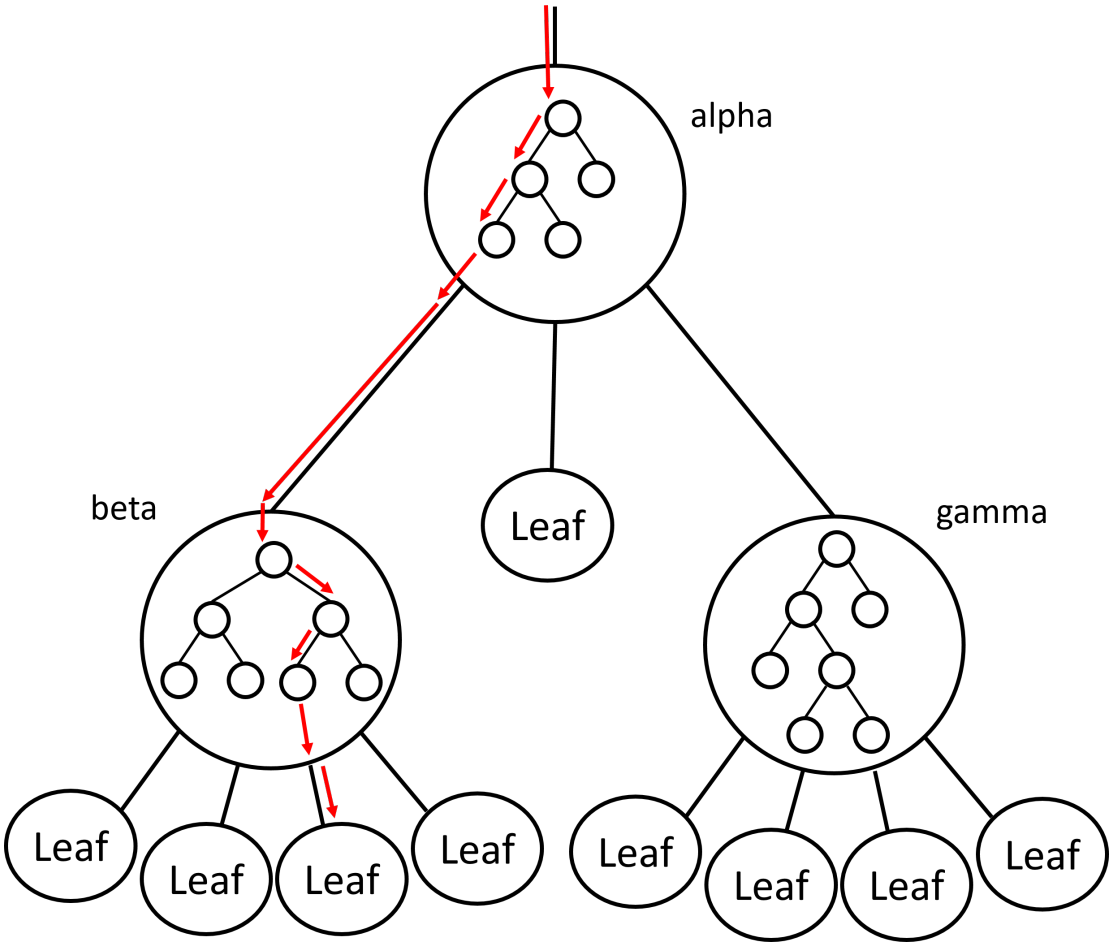

Figure 5.4: A path taken by an instance being classified using the model from figure 5.2

overfitting.

5.4.3 Outer Layer

The outer layer constructs the outer decision tree model in a way similar to how the inner layer does it. It employs the same decision tree construction algorithm as the inner layer does but uses a different method for constructing its individual nodes.

When a new outer tree node is being constructed, instead of iterating over

a full set of attribute values in search for a good split, the construction process iterates over a set of longitudinal attributes and tries constructing nodes using the whole set of their temporal values. Such nodes are constructed by building a small binary decision tree model that has a limited size and only uses the values of a single longitudinal attribute. The nodes are evaluated using a decision tree split metric and the best one becomes the new node of the tree. After the node is constructed, the number of leaf nodes in its inner tree is counted and the same number of branches are created from the outer node, mapping one leaf node of the inner tree to one branch of the node of the outer tree. As a result, the number of branches coming out of each node of the outer tree is always the same as the number of the leaf nodes in the internal tree of that node. Programmatically, the longitudinal attributes are represented as subsets of the full set of attributes, so the outer node construction process simply iterates over those, constructs binary decision trees using only those attributes and the class, and selects the best one to be used as the inner three of that node.

A non-longitudinal attribute in this algorithm is treated as if it was a longitudinal attribute with only one time point. As a result, non-longitudinal attributes are also considered as candidates for the inner tree construction and their trees get considered as outer node candidates on par with the ones constructed using longitudinal attributes.

The pseudocode for the outer layer construction process is presented in Algorithm 5.1. The construction begins by checking if the prepruning conditions were met: either the training set became too small or the maximum tree depth has been reached (line 2). If it has, the tree simply constructs a leaf node (line 3). Otherwise, it initialises a search for the optimal longitudinally-restricted decision tree to use as the new node (lines 5-6). It iterates through all of the longitudinal attributes present in the dataset (line 7). For each longitudinal attribute, it constructs a restricted subset that will only include the attributes that represent

Algorithm 5.1 Construction of an outer node of a Nested Trees model

```
1: function constructOuterNode(TrainingSet, LongitudinalAttributes)
2: if size(TrainingSubset) < minNodeSize OR maxDepthReached then
3:    ConstructLeafNode(TrainingSet)
4: else
5:    BestGini = 1.0
6:    BestTree = NULL
7:    for all LongitudinalAttribute in LongitudinalAttributes do
8:       RestrictedSubset = TrainingSet
9:       for all Attribute in TrainingSet do
10:         if Attribute ∉ LongitudinalAttribute then
11:            remove(RestrictedSubset, Attribute)
12:         end if
13:      end for
14:      InnerTree = constructTree(RestrictedSubset)
15:      Gini = measureGini(InnerTree, RestrictedSubset)
16:      if Gini < BestGini then
17:         BestGini = Gini
18:         BestTree = InnerTree
19:      end if
20:   end for
21:   OuterNode = createNode(BestTree)
22:   NumberOfLeaves = countLeaves(BestTree)
23:   for Leaf = 1 : NumberOfLeaves do
24:      LeafSubset = generateLeafSubset(BestTree, RestrictedSubset, Leaf)
25:      constructOuterNode(LeafSubset, LongitudinalAttributes)
26:   end for
27: end if
28: end function
```

values of the current longitudinal attribute (lines 8-13). After constructing the restricted subset, it uses it to build a new inner decision tree (line 14) and evaluates its split metric (line 15). It keeps track of the inner tree that achieved the best split metric (lines 16-19). After the search is completed, it uses the best inner tree as the new node of the outer tree (line 21). It then starts the recursive process of constructing branches corresponding to each of the leaves in the inner tree. For each leaf, it determines the subset of training instances corresponding to that leaf and recursively constructs nodes using those subsets (lines 22 - 26).

5.4.4 The issue of a large branching factor

In almost all types of classification models one of the key elements of model interpretability is the size of the model. Smaller models are typically easier to analyse and interpret while larger models are usually harder to interpret. This applies even to models such as decision trees, which are generally considered easily interpretable due to their simple structure.

Nested Trees models can suffer from this issue when constructed using large longitudinal datasets. Their size becomes larger, making them harder to interpret and their width allows for more repeated use of the same longitudinal attributes, further hurting the longitudinal interpretability of the model.

While the height of a Nested Trees model can be limited using a simple stopping criteria, the width of the model can not be restricted that easily. Since the number of branches an outer tree node has is always equal to the number of leaf nodes (as shown above in Figure 5.5), a bigger inner tree always leads to a much larger number of branches being generated for the node. Figure 5.3 shows one such outer node and Figure 5.2 shows a scarcely interpretable Nested Trees model with a large branching factor.

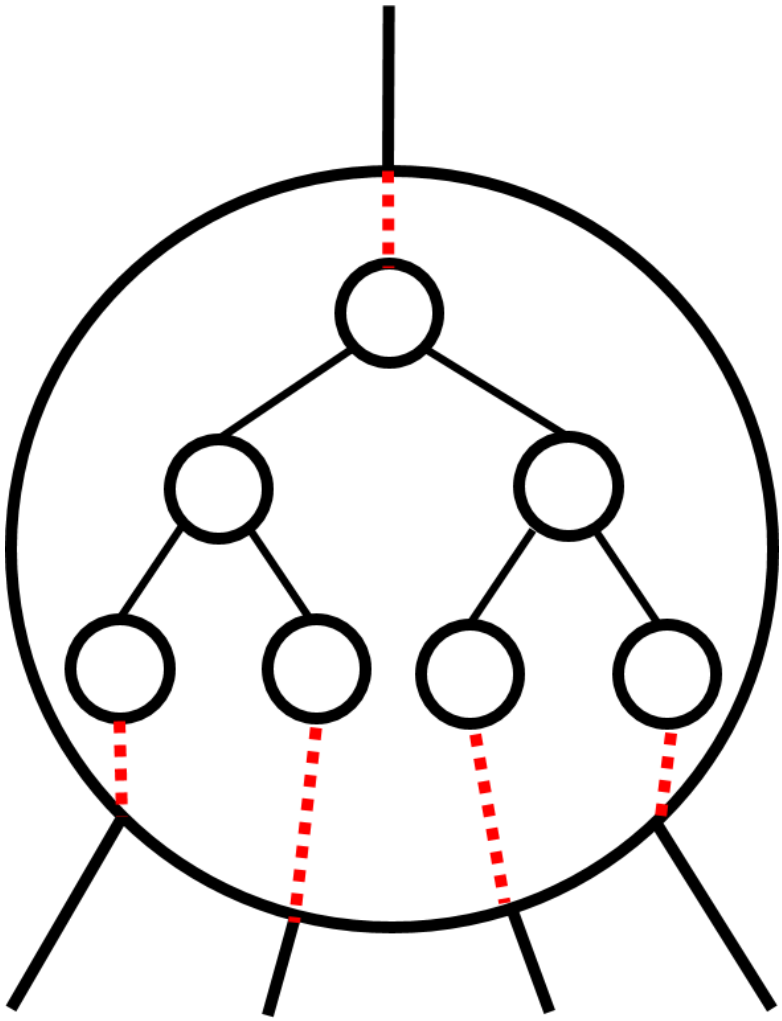

Figure 5.5: Mapping of an inner decision tree to the branches of the outer tree node

It is possible to reduce the branching factor of the outer tree by reducing the height limit for inner trees. Limiting the height of the inner trees would also limit the number of leaf nodes the inner trees can have, thus limiting the branching factor of the outer tree. That, however, would reduce the maximum number of temporal values each inner tree can analyse, limiting the amount of longitudinal analysis the model can conduct in a single outer node, encouraging the model to use the same longitudinal attributes more times, resulting in a more sparse distribution of the longitudinal values in the tree and defeating the purpose of the Nested Trees structure altogether.

5.4.5 Binary Nested Trees

Binary Nested Trees is a variation of the Nested Trees algorithm that solves the issue of large branching factor by forcing all outer tree nodes to be binary, ensuring that the branching factor of the outer tree is always 2. It adds an additional step to the model construction process that turns each outer node into a binary node after it has been constructed.

After an outer tree node is constructed, a binary conversion step is applied, altering the inner tree of that node and turning it into a a tree with only two leaf nodes. This is done by selecting one leaf node as the primary leaf node and marking all other leaf nodes as secondary. The number of branches of the outer node is then changed to two, with one branch being the primary branch, corresponding to the leaf node that was selected as primary, and the second branch being the secondary one. The inner tree is then pruned to remove all non-leaf nodes that do not lead to the primary leaf node. Finally, all non-primary leaf nodes are combined into a single secondary leaf node and mapped to the secondary branch. Figure 5.6 shows a step-by-step example of a binary node conversion process.

For a simple example, let's consider a small inner tree that only contains three

leaf nodes, nested in an outer node trained using 3000 training data instances with a 1200/1800 positive/negative class ratio. Let's say that the distributions of positive/negative class values in the training instances that reached the three leaf nodes were 100/900 for the first leaf node, 300/700 for the second and 800/200 for the third. We can iteratively consider each of the leaf nodes and calculate what the Gini impurity would be if we estimated it over a binary split that separates the instances of that leaf node from the rest. If the first leaf node was to be isolated, it would have 100/900 positive/negative instances, and the remaining leaf nodes would be merged into a 1100/900 distribution of positive/negative instances. Calculating the Gini impurity of this split, we get around 0.660. We can then repeat the same process for the remaining leaf nodes, getting around 0.498 for the second leaf node, and around 0.676 for the third. Since selecting the second leaf node produces the lowest Gini impurity, it would be selected as the designated leaf node (one that would get coloured in red in Figure 5.6), and thus the outer node would be transformed into a binary split node. This binary outer node would still contain the original tree inside, and would direct all nodes that reach the second inner leaf node to the red branch, and all nodes that reach other inner leaf nodes to the black branch. It is also useful to mention here that all three leaf nodes combined would produce a dataset with a 1200/1800 positive/negative ratio, and its Gini impurity would be around 0.520, so unless one of the binary conversions can produce a lower Gini impurity value, the outer node would have to be discarded altogether and replaced with a simple leaf node.

The reader is also invited to review a real binary nested tree model as outputed by the current implementation of the Binary Nested Trees algorithm. Since it was impractical to include a model of this size along with appropriate explanations into the thesis itself and including it as an appendix proved difficult, it is instead published in text format as a permanent pastebin link: https://pastebin.com/f17db4B4.

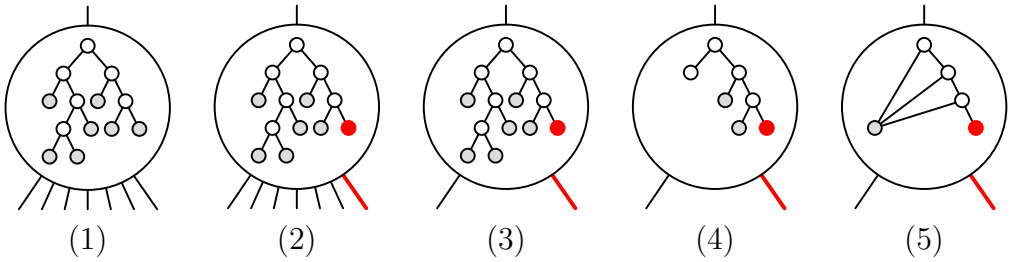

Figure 5.6: Illustration of a Binary Nested Tree node construction with a large starting inner tree in 5 stages: (1) a non-binary outer tree node is constructed; (2) a single leaf node of the inner tree (highlighted in red) is selected as the primary node; (3) the outer node is converted to binary, with one primary (red) and one secondary (black) branch; (4) the tree is pruned, to remove all unnecessary inner non-leaf nodes that do not lead to the primary leaf; finally, in (5), all non-primary inner leaf nodes are merged into one leaf node.

5.4.6 Other features of the Nested Trees algorithm

Some additional useful features are implemented and used in the Nested Trees codebase.

The Nested Trees algorithm has an inherent capability to work with missing values in the data. It uses instance weights to allow for missing values to be handled in a way similar to the well-known C4.5 algorithm [53]. When an instance has a missing value for an attribute used to split the data in a tree node, each branch coming out from that node gets a fraction of the instance equal to the proportion of instances in the current node having the attribute value associated with that branch.

A class balancing feature is implemented to deal with classification problems where the dataset is heavily dominated by a single class. It tackles class imbalance by applying a pre-processing step that alters the instance weights of the training set before model construction, ensuring that the sum of instance weights of all classes is exactly the same, achieving perfect class balance before training the model.

The core Nested Trees algorithm is implemented as a regression algorithm,

storing numerical regression values in its leaf nodes instead of categorical class values. After a model has been constructed, an optimal threshold is selected to be used to turn the numerical regression values outputted by the model into categorical class predictions. The optimal threshold is selected by iterating through the unique regression values produced by the model and finding the one that would result in the highest prediction accuracy of the model (still evaluated using the training set instances only). This classification-by-regression approach allows for the model to be evaluated using the AUROC accuracy measure and helps to avoid having to use a pre-set classification threshold that would work very poorly in combination with the class-balancing feature.

The core Nested Trees codebase also includes a number of tools for pre-processing longitudinal datasets, adding derived attributes (TIBI and TIBIW attributes as mentioned in the next chapter), and running large algorithm evaluation experiments.

It is important to mention here that there is an algorithm in the literature called Tree of Predictors (ToP) which also produces prediction models in the form of a decision tree with nodes containing smaller classification models [64]. This algorithm, despite following a similar design idea, is not a longitudinal classification algorithm and only uses simple regression models as its nodes [65].

5.5 Experimental Methodology

A number of experiments has been conducted in order to evaluate the performance of the Nested Trees algorithm and study its properties. Some parts of these results have been published in the 2022 conference paper [47].

The main aim of these experiments is to evaluate the Nested Trees algorithm and run a comparison with other classification algorithms. The algorithms will

be evaluated by their predictive accuracy, model size, and interpretability, including longitudinal interpretability. The experiments used 10 classification problems (datasets) originating from the ELSA (English Longitudinal Study of Ageing) database. These 10 datasets have exactly the same set of predictive attributes, but they have different class variables, representing different age-related diseases – i.e., each class variable indicates whether or not an individual is diagnosed with an age-related disease. For more details about these datasets, the reader is referred to Section 4.6.1.

The first comparison is between a conventional decision tree algorithm, the original Nested Trees algorithm (without enforcing binary partitions of the data at each node), and the Binary Nested Trees algorithm variation. The aims of this comparison are: to evaluate the effectiveness of the Nested Trees algorithm in achieving higher predictive accuracy than a longitudinally-unaware decision tree algorithm; to compare the predictive performance and model sizes of the binary and non-binary versions of the Nested Trees algorithm; to observe the algorithms' abilities to deal with class balancing; to observe the effects of the novel model structure in the sizes of the constructed models; and to evaluate the longitudinal interpretability of the models constructed by the three algorithms.

The conventional decision tree algorithm used as the baseline was identical to the algorithm that is used to construct the inner decision trees of the Nested Trees models. This was done to ensure that there were no differences between the baseline decision tree algorithm and the Nested Tree variants, except for the longitudinally-aware model construction process.

The second comparison is done between the most accurate Nested Trees algorithm variation and the models constructed by the XGBoost algorithm using the same datasets. The aim of this comparison is to evaluate how a longitudinally-aware Nested Trees algorithm compares to a stronger but longitudinally-unaware gradient boosting decision tree algorithm.

Note that XGBoost can be considered a "stronger" algorithm than Nested Trees in principle because XGBoost learns an ensemble of decision trees, and ensembles are known to increase predictive performance in general [54]. However, ensemble algorithms like XGBoost have the disadvantage of learning a kind of "black box" model, since it is not practical to interpret the large number of decision trees in the learned model. Hence, the comparison between the Nested Trees and XGBoost algorithms is useful to detect how much predictive performance (if any) is sacrificed when using the Nested Trees algorithm (to achieve longitudinal interpretability), by comparison with using XGBoost as a strong baseline method.

A well-known 10-fold cross validation was used to evaluate the performance of the constructed models. Additionally, in each experiment, the cross-validation was repeated 30 times (varying the cross-validation random seed) and the results were averaged over all runs. The cross-validation was applied to data instances only and not to the time points of longitudinal attributes — i.e., both training and test instances had the full set of longitudinal time points. The cross-validation was stratified, meaning that both the training and the test sets have retained the same class balance as the initial full set of instances (except for experiments when class balancing was applied to the training set).

Two measures were used to evaluate the predictive accuracies of the constructed models: (a) the average (unweighted) F-Measure, i.e. the arithmetic mean of the two F-measures computed by treating each of the two classes in turn as the "positive" class for the purposes of computing precision, recall and F-measure; and (b) the Area under ROC curve (AUROC).

Based on preliminary experiments, the following hyperparameter settings were used for the Nested Trees algorithm in all experiments:

- Maximum Outer Tree Depth: 10
- Maximum Inner Tree Depth: 5

Table 5.1: Average F-Measure values – the highest value per classification problem (dataset) is highlighted in boldface.

	Decision Tree		Nested Trees		Binary Nested Trees	
Dataset	Raw	CB	Raw	CB	Raw	CB
Angina	0.491	0.455	0.502	**0.515**	0.491	0.465
Arthritis	0.531	0.560	0.546	0.548	0.550	**0.597**
Cataract	0.565	**0.620**	0.575	0.575	0.592	0.617
Dementia	0.495	0.420	0.504	**0.532**	0.495	0.494
Diabetes	0.479	0.365	0.564	**0.584**	0.469	0.566
HBP	0.600	0.555	0.552	0.602	0.619	**0.636**
Heart attack	0.485	0.382	**0.524**	0.513	0.485	0.492
Osteoporosis	0.476	0.403	0.521	**0.541**	0.476	0.531
Parkinson's	0.498	0.345	0.497	**0.500**	0.498	0.451
Stroke	0.485	0.280	0.508	**0.527**	0.485	0.488
# wins	0	1	1	6	0	2

- Minimum Tree Node Size: 2 (applies to both inner and outer tree nodes)

5.6 Computational Results and Discussion

5.6.1 Predictive accuracy results

Tables 5.1 and 5.2 report the average F-Measure and AUROC values, respectively, for the three algorithms. As mentioned earlier, the average F-measure was computed by considering each class in turn as the positive class and macro-averaging the results, i.e., considering both classes as equally important. In these tables, the column headings "Raw" denote the raw (standard) version of each algorithm – a conventional Decision Tree, Nested Trees or Binary Nested Trees algorithm – whilst the column headings "CB" denote the versions using Class Balancing (applied to the training set only, as mentioned earlier).

Overall, a few interesting trends can be observed in the results. Almost all of

Table 5.2: Average AUROC values – the highest value per classification problem (dataset) is highlighted in boldface.

	Decision Tree		Nested Trees		Binary Nested Trees	
Dataset	Raw	CB	Raw	CB	Raw	CB
Angina	0.500	0.537	0.504	0.517	0.500	**0.587**
Arthritis	0.557	0.572	0.547	0.548	0.569	**0.598**
Cataract	0.582	**0.657**	0.574	0.577	0.591	**0.657**
Dementia	0.500	**0.687**	0.504	0.536	0.500	0.659
Diabetes	0.507	0.584	0.557	0.595	0.502	**0.669**
HBP	0.604	0.577	0.584	0.604	0.619	**0.650**
Heart attack	0.500	0.565	0.521	0.537	0.500	**0.639**
Osteoporosis	0.500	0.573	0.520	0.570	0.500	**0.656**
Parkinson's	0.500	0.587	0.498	0.508	0.500	**0.596**
Stroke	0.500	0.576	0.508	0.550	0.500	**0.622**
# wins	0	2	0	0	0	9

the best results in terms of predictive accuracy are obtained by the two variations of the proposed Nested Trees algorithm. The Nested Trees models seem to achieve higher F-Measure values while the Binary Nested Trees achieve higher AUROC values. More precisely, in Table 5.1, a variation of Nested Trees achieved the highest F-measure value in 7 of the 10 classification problems. In Table 5.2, the Binary Nested Trees with Class Balancing have achieved the highest AUROC value in 9 of the 10 classification problems.

The class balancing pre-processing step has had a major impact on the predictive accuracy measures, often leading to a noticeable improvement. Among the 20 best (boldface) results in Tables 5.1 and 5.2, 19 were results achieved with by a CB variation, with only a single exception. Class balancing has also proven to have a positive impact on classification problems with a particularly severe class imbalance, where the algorithm sometimes struggled to construct meaningful prediction models without it and often defaulted to a trivial decision tree model with

Table 5.3: Results of the post-hoc Holm test at 5% significance for the three algorithms with class balancing, based on the average F-Measure values.

Algorithm	Average Rank	p	Holm
Nested Trees+CB (control)	1.5	–	–
Binary Nested Trees+CB	1.8	0.502	0.05
Decision Tree+CB	2.7	0.007	0.025

Friedman test's p-value = 0.02024

Table 5.4: Results of the post-hoc Holm test at 5% significance for the three algorithms with class balancing, based on the average AUROC values.

Algorithm	Average Rank	p	Holm
Binary Nested Trees+CB (control)	1.15	–	–
Decision Tree+CB	2.05	0.044	0.05
Nested Trees+CB	2.8	0.00022	0.025

Friedman test's p-value = 0.00109

only a single leaf node.

For the statistical analysis of the results, we focus on the more interesting algorithm variants – the ones using Class Balancing. We performed two statistical analyses, involving two predictive accuracy measures and three algorithms. More precisely, we perform two separate statistical significance tests: one using the F-Measure values and one using the AUROC values. In each of these analyses, we compare the results obtained by the three algorithms evaluated in the experiments: a conventional Decision Tree, the original Nested Trees and the Binary Nested Trees algorithms. As in the previous chapter, the statistical tests were conducted using the original F-Measure and AUROC values reported by the algorithm, without any rounding of the results prior to statistical testing.

In each analysis, we use the non-parametric Friedman test [19] to check if there is a statistically significant difference between the 3 algorithms overall, at the usual significance level of 5%. If there is, we use the post-hoc Holm test to

Table 5.5: Average model size calculated as the total number of nodes in the trees. For Nested Trees variations, the number of nodes include both outer and inner nodes.

	Decision Tree		Nested Trees		Binary Nested Trees	
Dataset	Raw	CB	Raw	CB	Raw	CB
Angina	1	681	1286	1884	1	182
Arthritis	658	678	4912	5423	272	268
Cataract	493	390	4351	5380	287	151
Dementia	1	549	758	1214	1	113
Diabetes	221	541	2549	3485	11	251
HBP	567	592	4751	2882	289	305
Heart attack	1	683	1625	1589	5	210
Osteoporosis	27	595	2104	1635	1	203
Parkinson's	1	642	564	539	21	106
Stroke	1	524	1675	1609	1	197

check if the best algorithm is statistically significantly better than the other two algorithms. A Holm test result is significant if the obtained p-value is smaller than a certain threshold that is calculated to correct the significance level for multiple hypothesis testing [19]. In the following statistical result tables, this adjusted threshold is shown in the column "Holm", and a Holm test's result is significant if the p-value in the "p" column is smaller than the value in the "Holm" column.

Table 5.3 reports the results of the Friedman and Holm post-hoc tests for the F-measure, comparing the three algorithms using Class Balancing. The original Nested Trees algorithm has achieved the best (smallest) rank. The Friedman test indicated a significant difference among the three algorithms and the Holm test indicated that the original Nested Trees algorithm significantly outperformed the conventional Decision Tree algorithm, with no significant difference between the original Nested Trees and Binary Nested Trees algorithms.

Table 5.4 reports the results of the of the Friedman and Holm post-hoc tests for the AUROC measure, for the three algorithms using Class Balancing. The

Table 5.6: Average number of internal nodes in the *outer* (*inner*) trees in the Nested Tree models. For the average number of inner nodes, this is the average number per individual node in the outer tree. The '1 (0)' values represent cases where the outer tree is a single leaf node without any inner tree.

	Nested Trees		Binary Nested Trees	
Dataset	Raw	CB	Raw	CB
Angina	666 (20.93)	1024 (15.81)	1 (0)	79 (4.29)
Arthritis	2580 (12.93)	2797 (13.01)	132 (4.19)	144 (4.18)
Cataract	2355 (13.04)	2924 (14.29)	124 (4.24)	87 (4.13)
Dementia	447 (22.61)	626 (17.02)	1 (0)	60 (4.31)
Diabetes	1425 (15.62)	1909 (15.24)	6 (4.50)	114 (4.22)
HBP	2542 (13.16)	1743 (10.08)	135 (4.18)	136 (4.18)
Heart attack	842 (18.92)	770 (11.22)	1 (0)	87 (4.32)
Osteoporosis	1223 (16.88)	952 (11.16)	1 (0)	100 (4.20)
Parkinson's	274 (25.71)	274 (11.41)	1 (0)	39 (4.29)
Stroke	893 (18.55)	855 (11.09)	1 (0)	79 (4.26)

Binary Nested Trees algorithm has achieved the best (smallest) rank, and the Holm test indicated that Binary Nested Trees significantly outperformed both the conventional Decision Tree and the Nested Trees algorithms.

These differences in F-Measure and AUROC are, most likely, the result of the different sizes of the models generated by the two algorithms. The original Nested Trees algorithm tends to create a large number of splits, resulting in a very large tree, where each of the leaf nodes uses a very small number of training instances. The Binary Nested Trees, however, builds a much smaller model with larger sets of instances in leaf nodes. This leads to the model built by the Binary Nested Trees algorithm having much more variety in the class probability values that its leaf nodes can have, resulting in a smoother ROC curve with a larger area under it. The original Nested Trees algorithm tends to have very small leaf nodes, usually containing 2-4 instances. It can therefore only have a very small number of different class probability values, making the ROC curve less defined, having

only a few points between [0,0] and [1,1] and thus having a smaller area under the curve.

Recall that, overall, the model construction process for the Binary Nested Trees is very similar to the original (non-binary) Nested Trees. The original Nested Trees algorithm builds its nodes using inner decision trees, producing a Nested Trees model with a large branching factor. The Binary Nested Trees algorithm uses the same model construction method but adds a restriction on the branching factor, turning the outputs of the leaf nodes of each inner tree (however many there may be) into just two branches for the outer tree. This severely limits the search space, compared to the Nested Trees algorithm, and might provide little advantage in terms of predictive accuracy (safe for a chance to avoid overfitting). Therefore, the Binary Nested Trees may sacrifice some predictive accuracy for the sake of smaller and more interpretable models. It is therefore not unexpected that Binary Nested Trees would underperform in terms of average F-Measure, though it is interesting to see that it still often produces models much better than the conventional Decision Tree algorithm. This F-Measure difference between binary and non-binary versions was found to be not statistically significant.

5.6.2 Model size results

Table 5.5 shows the average number of tree nodes for each model, calculated as the sum of all nodes in the resulting tree model, including leaf nodes. For the Nested Trees variants, the total number of nodes is estimated as the total number of outer tree nodes plus the total number of meaningful inner tree nodes: in the inner tree nodes, the root node and the leaf nodes were not included in this calculation, since they only serve a structural purpose in the Nested Trees models and have no meaning in terms of class prediction. In the experiments where an algorithm learned a trivial decision tree with a single leaf node, we report a value of 1.

Table 5.6 shows the average number of nodes in the Nested Tree models, both

Table 5.7: Most frequently used longitudinal attributes in Nested Trees models.

Dataset (class)	Longitudinal attributes
Angina	Mean grip strength with dominant hand, Blood cholesterol levels (both total and LDL), Blood mean corpuscular haemoglobin level, Age, Mean waist measurement, Mean diastolic blood pressure
Arthritis	Weight, Sex, Mean grip strength with non-dominant hand, Blood hemoglobin level, Mean arterial pressure, Age
Cataract	Age, Blood ferritin level, Vitamin D level, Blood fibrinogen level, Lung Forced Vital Capacity, Mean waist measurement
Dementia	Age, Weight, Blood mean corpuscular hemoglobin level, Lung Peak Flow, Lung Forced Expiratory Volume, Pulse pressure, Mean waist measurement
Diabetes	Fasting glucose level, Blood HDL level, Blood total cholesterol level, BMI, Mean waist measurement
HBP	Mean diastolic blood pressure, Mean systolic blood pressure, Mean waist measurement, Pulse pressure, Mean hip measurement, Lung Forced Vital Capacity
Heart attack	Mean grip strength with dominant hand, Lung Forced Expiratory Volume, Mean hip measurement, Mean diastolic blood pressure, Pulse pressure, Blood LDL cholesterol level
Osteoporosis	Sex, Age, Outcome of side-by-side stand exercise, Presence of a Respiratory infection in last 3 weeks, Blood triglyceride level, Vitamin D level, Mean grip strength with dominant hand
Parkinson's	Age, Blood mean corpuscular hemoglobin level, Blood DHEAS level, Blood hemoglobin level, Lung Forced Vital Capacity, Mean hip measurement, Mean systolic blood pressure, White blood cell count
Stroke	Blood LDL cholesterol level, Lung Forced Vital Capacity, Blood triglyceride level, Mean grip strength with dominant hand, Age, Mean systolic blood pressure, Lung Forced Expiratory Volume

outer and inner nodes, respectively. The 1 values represent the experiments where a meaningful model could not be constructed or the tree got pruned to a single leaf node; 0 values represent experiments where the outer tree was pruned to a single leaf node without any inner tree.

An interesting observation can be made here about the sizes of the constructed models. While the sizes of the inner trees vary a lot in the non-binary Nested Trees models, in general they have almost identical number of nodes in all of the Binary Nested Trees experiments. Recall that the conversion method for the Binary Nested Trees limits the number of the leaf nodes in the inner tree to 2, thus also limiting the number of possible paths a prediction process can take from the root to the leaf of the inner tree. This, in turn, removes the necessity for many of the nodes of the inner tree, so they get removed during the conversion – illustrated in Figure 5.6. As a result, the size of the inner trees tends to be limited by the maximum depth parameter of the inner trees (which in all of these experiments was set to 5). The lower branching factor also tends to reduce the number of outer tree nodes very significantly, which is also a very predictable outcome.

Nested Trees were generally somewhat easier to interpret; having a more longitudinally-aware structure made them much easier to evaluate in terms of general model structure. It was easier to identify the most important longitudinal attributes in Nested Trees models, as all of the high-level comparisons based on them were grouped into the Nested Tree nodes. It was, however, not easy to navigate such Nested Trees models due to their very large number of nodes and a large branching factor.

The Binary Nested Trees has proven to be the most interpretable among the three types of models. They have had the same advantage that the non-binary Nested Trees had in terms of grouping together (in an inner node) different values of a longitudinal attribute, but without the disadvantage of the very large model size. They were relatively easy to navigate due to their smaller size, and the impact

of each longitudinal attribute on the classification outcome was much clearer. The Binary Nested Trees were in general even smaller than the conventional decision trees, as shown in Table 5.5. Overall, the size of the models played a major role in model interpretability with smaller models generally being more interpretable than the larger models of the same type. This was especially true for the inner models constructed by Nested Trees algorithms and resulted in a major difference in interpretability between the binary and non-binary versions.

5.6.3 Attribute importance results

Table 5.7 reports the longitudinal attributes commonly used by the best models constructed by the Nested Trees variants. These attributes were acquired by manually interpreting the top layers of the trees constructed by the top performing algorithms (the ones with results in bold in Tables 5.1 and 5.2). From the non-binary Nested Trees, the top 2 layers (i.e., from the root until nodes of depth 2) were interpreted and from the Binary Nested Trees the top 3 layers. The table represents a combined collection of most used longitudinal attributes. Overall, the attributes selected by the models were sensible, matching well with conventional medical disease characteristics and markers. The list of useful predictive attributes comprised here is arguably more revealing than a similar list comprised by an earlier XGBoost study using the same classification problems [48].

5.6.4 Comparison with XGBoost

According to the results from Table 5.2, the highest AUROC values were achieved by the Binary Nested Trees algorithm with class balancing pre-processing step. Table 5.8 compares the AUROC values achieved by this variation with the AUROC values achieved by XGBoost algorithm using the same classification problems.

XGBoost is obviously a much stronger classification algorithm, utilising a

Table 5.8: Average AUROC values of the two best Nested Trees variations compared to XGBoost—the highest value per classification problem is highlighted in bold.

Dataset	BNT CB	XGBoost
Angina	**0.587**	0.545
Arthritis	0.598	**0.610**
Cataract	**0.657**	0.655
Dementia	0.659	**0.755**
Diabetes	0.669	**0.825**
HBP	0.650	**0.704**
Heart attack	0.639	**0.679**
Osteoporosis	0.656	**0.666**
Parkinson's	**0.596**	0.575
Stroke	0.622	**0.666**
# wins	3	7

unique combination of regression tree construction with an added layer of gradient boosting in an ensemble; so it tends to construct significantly more accurate models than simpler algorithms like a conventional decision tree algorithm.

It is worth noting, however, that while Binary Nested Trees was outperformed by XGBoost in most classification problems, it has achieved significant improvement in two problems that both algorithms found particularly hard - Angina and Parkinsons, with both datasets corresponding to two of the worst AUROC results in both sets of measurements. This improvement is likely the result of the longitudinal awareness of Nested Trees models being able to bring more predictive value than the boosting approach utilised by XGBoost.

The highest advantages achieved by XGBoost in this comparison have originated from some of the easiest classification problems, such as Diabetes and HBP.

Two major advantages that Nested Trees has over XGBoost are model sizes and longitudinal interpretability. XGBoost constructs bulky models consisting of multiple separate decision trees, generally consisting of much larger numbers

of nodes. Binary Nested Trees constructs much smaller models that group the temporal values of longitudinal attributes together, thus allowing for much easier interpretation of the model and much more apparent longitudinal attribute significance.

5.7 Conclusions

Overall, the Nested Trees algorithm has demonstrated a lot of value.

It has a major predictive accuracy advantage when compared against a conventional longitudinally-unaware decision tree algorithm, and has reacted much better to the introduction of the class balancing pre-processing step.

The binary version turned out to be the most useful, constructing compact yet powerful classification models that offer very good longitudinal interpretability compared to other existing longitudinally-aware alternatives. Binary Nested Trees were relatively inexpensive in terms of computational time, allowing for quick model construction and evaluation.

There is a lot of potential for future development in the Nested Trees algorithm too. Its inherent ability to function as a regression algorithm and its built-in capacity for effectively utilising instance weights makes it highly likely that it would perform well in an ensemble or a boosting setting. Additionally, its high degree of customisability allows its codebase to be used as a tool for testing other possible approaches related to decision tree construction and longitudinal classification.

It is important to note here that the evaluation of the Nested Trees algorithm has been severely limited by the lack of publicly available longitudinal classification algorithms that could be used as more reliable baselines than conventional decision trees with the flattening approach or the much more powerful but longitudinally unaware XGBoost.

It is also important to note that the Nested Trees algorithm is the first longitudinal classification algorithm to be released publicly in the form of a GitHub repository (github.com/NestedTrees/NestedTrees). Additionally, the Nested Trees algorithm has been integrated into a custom python library built on scikit-learn, called Scikit-longitudinal, currently developed by a University of Kent PhD student Simon Provost. The public version of this project is currently available on GitHub (https://github.com/simonprovost/scikit-longitudinal).

Chapter 6

Monotonic Nested Trees

This chapter will present the third contribution of this study: an expanded version of the Nested Trees algorithm that includes monotonicity detection features, enforces monotonicity constraints in constructed models, creates derived longitudinal monotonic attributes and is capable of producing fully-monotonic, longitudinally-aware and longitudinally-interpretable classification models.

6.1 A Unified Approach

The experiments reported in Chapter 4 have shown that automated monotonicity detection and enforcement can be beneficial in classification problems. Enforcing monotonic constraints can lead to an improvement in predictive accuracy, a reduction in model size, and a general improvement in model interpretability. On its own, monotonicity detection can provide an insight into the relations between predictor attributes and the class attribute and thus serve as an additional knowledge discovery approach.

The experiments reported in Chapter 5 have shown that the Nested Trees algorithm can serve as an effective algorithm for longitudinal classification and has unique features such as longitudinal awareness and longitudinal interpretability

of its learned models.

Longitudinal datasets contain repeated measurements of the same attributes at different points in time, effectively introducing an additional dimension to conventional flat datasets. This means that there is a lot more room for knowledge discovery, since there can be interesting changes occurring across different time points. Monotonicity detection can be applied to longitudinal datasets in order to explore the monotonic relations that occur in longitudinal data and the results can be used to aid in longitudinal classification tasks.

The Nested Trees algorithm is a longitudinal classification algorithm that produces longitudinally aware and interpretable classification models. It has a high degree of customisability and can be easily modified to introduce additional features.

The objective of this final contribution is to design, implement and evaluate a new version of the Nested Trees algorithm. This new version will include monotonicity detection, as well as creation and enforcement of monotonic constraints. It will introduce features for creating two types of derived attributes that will use longitudinal monotonicity detection to acquire the new attribute values. Finally, it will introduce a higher degree of monotonic interpretability by producing metrics and statistics about the use of monotonicity features in constructed models.

6.2 The Nested Trees Algorithm with Timeless Monotonicity Constraints

In order to implement the detection and enforcement of Timeless monotonicity constraints (see Section 4.3), major changes needed to be made to the Nested Trees codebase.

Firstly, monotonicity detection features were implemented to detect the strengths

of monotonic relations between the class attribute and each of the predictor attributes and convert those into monotonicity constraints. This implementation was identical to the one described in Chapter 4.

Secondly, the Nested Trees algorithm was modified to support monotonicity constraints. The monotonicity constraints enforcement feature was modeled after the same feature from the XGBoost algorithm [27]. The monotonicity constraints can be supplied to the Nested Trees algorithm as an additional construction parameter. Monotonicity constraints are supplied as an array of integer values, where each value corresponds to a predictor attribute. Each attribute constraint can have the value of 1, -1 or 0, corresponding to positive monotonicity constraint, negative monotonicity constraint, or no monotonicity constraint respectively. The Nested Trees algorithm then propagates the same monotonicity constraints to all of the nodes in the decision tree being constructed.

It is important to note here that since the Nested Trees algorithm supports the use of instance weights and uses them to perform class balancing of training sets, the NMI (Non-Monotonicity Index) calculation was also adjusted to be able to deal with weighted data instances. As a result, Timeless monotonic constraints are now being measured using the class-balanced training sets, using adjusted instance weights. This additionally leads to the resolution of the issue of low entropy in the values of the class attributes that was previously impacting the number of monotonicity constraints produced by the Timeless monotonicity detection using EANMI, since after the class balancing is performed, the class attribute has a perfect 50/50 distribution of the positive and negative class values and its entropy is identical to the maximum possible entropy of an attribute with a binary domain.

The actual monotonicity enforcement happens during the process of constructing individual decision rules, inside the nodes of the inner trees. There are two measures put in place to ensure full monotonicity of the constructed models. When a decision rule is being constructed using a monotonically-restricted attribute,

each split is being evaluated against the monotonic constraints. A check is implemented in the split construction to prevent the algorithm from constructing rules that would result in non-monotonic splits. So if the attribute used has a positive monotonicity constraint and a split is being considered that would result in the smaller value of the attribute corresponding to the subset with a higher average class value and the higher value of attribute corresponding to the subset with a lower average class value, then that split would be skipped. This check prevents the algorithm from constructing splits that would likely lead to non-monotonic predictions, but it does not guarantee full monotonicity of model's predictions.

In order to guarantee full monotonicity of the model's predictions, the algorithm also puts regression boundaries on each of its nodes. When the Nested Trees model construction begins at the root node, its boundaries do not restrict the regression value at all – where a regression value is the value of the target attribute estimated at the current node, which is the arithmetic mean of the values of the target attribute over all instances in the current node. The boundaries only change after a decision rule is constructed using a monotonically-restricted attribute. During rule construction, a regression value is calculated based on the instance subset used. That regression value then becomes a new boundary for all nodes that are constructed below this rule. For example, if a split was created using a monotonically restricted attribute, then the average class value of the subset that was used to construct that split would become the upper prediction boundary for the branch corresponding to the lower attribute value and the lower prediction boundary for the branch corresponding to the higher attribute value. This means, that all predictions made by the lower branch will always be lower or equal than the predictions made by the higher branch, thus ensuring full monotonicity of the Nested Trees model predictions. This implementation of hard monotonicity constraints is similar to the implementation used by the XGBoost algorithm [27].

For a simple example, let's consider a node that is being constructed using a predictive attribute that has a positive monotonicity constraint. Let's say the attribute was `chol_w8` - the blood cholesterol measurement taken during wave 8 of ELSA measurements, and the node has selected the threshold of 4.5 for the binary split. Let's also say that the node was trained using 1000 instances, and the positive/negative split was 670/330. If a leaf node was constructed using a dataset with that class distribution, it would simply become a leaf node predicting the regression value of 0.67. However, since this is a binary split on a constrained attribute, it would need to put constraints on the regression values of the branches coming out of the split, so the branch corresponding to `chol_w8` values above 4.5 would have their lower prediction constraint set to 0.67, and the branch corresponding to `chol_w8` values below 4.5 would have its upper prediction constraint set to 0.67. These constraints propagate down the tree, so all of the leaf nodes under the branch corresponding to the lower value of `chol_w8` will make predictions smaller or equal to 0.67, and all leaf nodes under the branch corresponding to the higher value of `chol_w8` will make predictions higher or equal to 0.67.

6.3 Derived Monotonic Longitudinal Attributes

In order to provide additional longitudinal information for the Nested Trees algorithm, two types of derived attributes were introduced to the Nested Trees codebase.

The first type was the Time Index-Based Individual (TIBI) attributes, the same as the one described in Chapter 4 (Section 4.4). As mentioned before, these derived attribute values are constructed separately for each data instance. TIBI attributes represent longitudinal trends in the values of longitudinal attributes and are therefore considered by the Nested Trees algorithm to be part of the longitudinal attribute they were constructed from. This means that if a longitudinal

attribute originally had four attributes corresponding to four time-indexed measurements, then after the TIBI attribute construction is finished it will have five attributes: the four time-indexed attributes plus one derived attribute that does not have a time index. In the context of the Nested Trees algorithm this means when an inner tree is being constructed using that longitudinal attribute, it will now have five attributes that it can use, instead of four.

The second type of derived attributes is called Time Index-Based Individual-Wise (TIBIW) attributes. This type of derived attributes uses a new longitudinal monotonicity detection approach that allows for creation of much more interesting attributes. The TIBIW attributes are, similarly to TIBI attributes, constructed separately for each data instance, and can therefore be used safely in the context of cross-validation experiments. TIBIW attributes are designed to represent the strengths of monotonic relationships between the values of different longitudinal attributes.

Table 6.1 shows an example of a flattened longitudinal dataset with two longitudinal attributes called A and B. The dataset has five data instances and each of the two longitudinal attributes have four time-indexed attributes. A TIBIW attribute can be constructed to represent how the two longitudinal attributes A and B correlate within each data instance. Similarly to TIBI, a Spearman correlation coefficient can be used to estimate these values. This time, the values used for Spearman correlation coefficient will be the time-indexed attribute values of the two longitudinal attributes. For example, for data instance 1, the value of the TIBIW attribute between longitudinal attributes A and B will be equal to the Spearman correlation coefficient between arrays [5.7, 5.9, 6.1, 6.3] and [78, 79, 77, 78]. Table 6.2 shows the same dataset with the TIBIW attribute added.

Since the TIBIW attributes are used to represent the correlations between the values of longitudinal attributes in individual data instances, the total number of TIBIW attributes will be equal to the number of pairs of different longitudinal

Table 6.1: Flattened longitudinal dataset with two longitudinal attributes

Id	A_w1	A_w2	A_w3	A_w4	B_w1	B_w2	B_w3	B_w4
1	5.7	5.9	6.1	6.3	78	79	77	78
2	7.2	7.8	7.5	7.3	91	93	94	98
3	6.7	6.7	6.8	7.1	77	79	83	89
4	6.8	6.9	7.0	7.2	76	73	71	70
5	6.5	7.6	7.6	7.7	85	83	86	87

Table 6.2: Flattened longitudinal dataset with two longitudinal attributes

Id	A_w1	A_w2	A_w3	A_w4	B_w1	B_w2	B_w3	B_w4	A_B_TIBIW
1	5.7	5.9	6.1	6.3	78	79	77	78	-0.316
2	7.2	7.8	7.5	7.3	91	93	94	98	0.2
3	6.7	6.7	6.8	7.1	77	79	83	89	0.949
4	6.8	6.9	7.0	7.2	76	73	71	70	-1
5	6.5	7.6	7.6	7.7	85	83	86	87	0.632

attributes. For N longitudinal attributes there will be $\frac{N\times(N-1)}{2}$ TIBIW attributes. The addition of TIBIW attributes results in a massive increase in the size of the dataset that scales quadratically with the number of longitudinal attributes. When constructing inner nodes, the Nested Trees algorithm allows inner trees that use a particular longitudinal attribute to use all of the TIBIW attributes that were derived using that longitudinal attribute. This means that the total number of data attributes available to the inner tree will be equal to the number of time indexed values of the longitudinal attribute plus $N-1$ TIBIW attributes.

6.4 Evaluation of the Monotonic Features of the Nested Trees Algorithm

The extension to the Nested Trees algorithm described in this chapter introduced three monotonicity-based features: Timeless monotonicity constraints, TIBI attributes, and TIBIW attributes. A set of experiments has been conducted in order to evaluate the effect of these features on the predictive accuracies of the models

learned by the Nested Trees algorithm.

Additional metrics were recorded during each experiment to gain a deeper insight into the effects of adding these features to the Nested Trees algorithm. In addition to the attribute importance metrics that the Nested Trees algorithm produces, aggregate importance metrics were constructed to measure the total attribute importance of all TIBI and all TIBIW attributes. This provided an insight into the usefulness of adding the TIBI and TIBIW attributes to specific classification problems. For monotonic constraints, the output of the Nested Trees algorithm was expanded and it now reports the average number of monotonic constraints used in the experiment, as well as a list of all predictor attributes along with their frequency of being monotonically constrained.

6.5 Experimental Setup

Experiments were conducted using three model types: Decision Trees, Nested Trees without binary conversion and Binary Nested Trees. Each model was evaluated in five different configurations:

- **Raw** configuration used the raw version of the classification algorithm without any monotonicity-based features to learn classification models. These were used as baseline models for evaluation of the effects of the new monotonic features.

- **Timeless** configuration used the version of the classification algorithm with the Timeless monotonicity constraints detection and enforcement. This one was used to evaluate the effects of introducing monotonicity constraints into the models.

- **TIBI** configuration constructed classification models using the datasets with the TIBI attributes constructed. This configuration was used to evaluate

the effects of introducing the TIBI attributes.

- **TIBIW** configuration constructed classification models using the datasets with the TIBIW attributes constructed. This configuration was used to evaluate the effects of introducing a large number of TIBIW attributes.

- **All** configuration used all three features at the same time: it generated TIBI and TIBIW attributes, used Timeless monotonicity detection to create monotonicity constraints, and enforced monotonicity constraints during model construction. This heavy configuration was used to evaluate the effects of using all currently defined monotonicity detection methods in the same longitudinal classification approach.

The experiments were conducted by applying the classification algorithms to 20 classification problems (datasets) derived from the ELSA database: 10 disease prediction problems using predictor attributes from ELSA Nurse dataset (as described in section 4.6.1) and 10 disease prediction problems using predictor attributes from the ELSA Core dataset. The 10 datasets with ELSA Nurse data use exactly the same set of predictor attributes, they vary with respect to class variable, with a different disease representing the class variable in each dataset. Likewise, the 10 datasets with ELSA Core data use exactly the same set of predictor attributes, they have different diseases representing different class variables across the datasets. The 10 ELSA Core-based datasets were also used in another recent study on longitudinal derived attributes [55] and only became available after the experiments from the previous chapters were concluded and published.

The experiments were conducted using 3 classification model types, each with 5 different configurations (as enumerated earlier), each evaluated on 20 classification problems (datasets). Each such experiment included a 10-fold cross-validation repeated 30 times in order to achieve highly-reliable experimental results.

The hyper-parameters of the classification algorithms used in the experiments were set as follows, based on the results of previous experiments:

- All decision rule models (used only as nodes in inner trees) used the Gini impurity metric for choosing the optimal splits. Minimum subset size was set to 2, preventing the decision rules from constructing a split where one of the subsets would only have one instance. This was doubly-enforced using both the sum of instance weights in a subset and a count of instances present in it, to prevent instances that gained high instance weights as a result of class balancing from violating this rule.

- The outer decision tree models used the Gini impurity metric to evaluate subsets produced by inner models. Decision trees used pre-pruning techniques to avoid constructing unnecessary splits. They would stop constructing further splits when the subset only contained instances belonging to one class, when the maximum depth of 10 was reached, and when the size of the subset fell below the minimum allowed size of 4.

- Inner trees of the Nested Trees models used the same parameters as the outer trees, except their maximum depth was set to 5 and minimum node size to 2.

- Timeless monotonicity detection was performed using the EANMI (Entropy-Adjusted Non-Monotonic Index) metric, proposed in Chapter 4 (Section 4.3). Entropy adjustment was only applied based on the entropy ratio of the predictor attribute and did not use the entropy of the class. As mentioned above, the issue of low entropy in the values of the class attribute could potentially be resolved using class balancing. Since class balancing was performed in all experiments described below, the entropy of the class values was the same as the maximum possible entropy for an attribute with a binary domain, thus performing entropy adjustment on the class attribute

would have no effect on the number of monotonic constraints created by the Timeless monotonicity detection method.

The threshold for Timeless monotonicity constraints was 0.15.

- TIBI and TIBIW attributes were generated using the default version of the Spearman correlation coefficient. They were allowed to have missing values in cases where it was not possible to estimate their values, e.g. when the target longitudinal attribute did not contain enough time-indexed non-missing values.

6.5.1 Additional metrics

The Nested Tree algorithm also estimated aggregate attribute importances for TIBI and TIBIW attributes in the experiments that used them. These attribute importances were estimated by adding up the attribute importances of all attributes of the respective type of attributes, e.g. aggregate attribute importance of TIBI attributes is equal to the sum of attribute importances of all TIBI attributes.

Attribute importances of individual data attributes were estimated by iterating through all decision rules in the model (as these are the only models that perform training set splitting) and adding up attribute importances of those. For each decision rule node, the attribute importance was estimated for the attribute that the decision rule used, and was equal to the reduction in Gini impurity caused by the split, multiplied by the proportion of the training set instances that reached that node. So if a node used attribute Attr1, reduced the Gini impurity from 0.41 to 0.36 and was trained on the subset containing 10% of the training instances, it would contribute $10\% \times (0.41 - 0.36) = 0.005$ to the total attribute importance of the Attr1.

In repeated cross-validation experiments, the attribute importances were added

up across all of the experiments and converted to proportions of their sum, in order to ensure that their attribute importances added up to 1. So if after a large series of experiments all of the attributes have added up to 243.5 and the total attribute importances of Attr1 have added up to 29.22, then the attribute importance of Attr1 would be estimated as $29.22 \div 243.5 = 0.12$. This conversion is very useful as it provides an easily interpretable measure of attribute importance: splits using Attr1 contributed 12% of the reduction of Gini impurity across the whole series of experiments. If Attr1 was a TIBIW attribute, that would also add 0.12 to the aggregate importance of TIBIW attributes.

Model sizes were estimated for all models constructed. For Decision Tree models, the model size was equal to the number of nodes in the decision tree. For Nested Tree and Binary Nested Tree models, the model size was calculated as the total number of nodes in both inner and outer trees added together.

6.6 Experimental Results

This section presents the results of the classification experiments using the 20 classification problems (datasets) described above. The 10 classification problems that used predictor attributes derived from the ELSA Nurse dataset are prefixed with "N-" and the 10 classification problems that used ELSA Core attributes are prefixed with "C-".

6.6.1 Decision Tree results

The conventional (longitudinally unaware) Decision Tree algorithm was the simplest classification algorithm used in these experiments. It generally constructed weak classification models and struggled to make good use of the newly introduced monotonic features. Tables 6.3 and 6.4 show the comparisons of predictive accuracy measures of the 5 configurations of monotonicity-based features applied

to decision trees.

Regarding Table 6.3, with the F-measure results, overall, the best results were obtained when using Timeless configuration, which led to the best result in 12 of the 20 datasets – although this includes 4 datasets where Timeless and Raw were joint winners (with a tied average F-measure). The Raw and TIBI configurations also did well, leading to 9 and 10 wins respectively, including some joint wins with Timeless. The worst results were obtained when using TIBIW and All configurations, which led to the best results in only 5 and 3 datasets, respectively, including some joint wins.

Regarding Table 6.4, with the AUROC results, overall, the best results were clearly obtained when using Timeless configuration, which led to the best result in 17 of the 20 datasets – although this includes 6 datasets with joint winners (with a tied average F-measure). The second best configuration overall was All, which led to only 6 wins, all of them joint wins with Timeless.

Table 6.5 shows the average sizes of the Decision Tree models constructed with different types of attributes. The main pattern in these results is that, in general, the decision tree models constructed using attributes from the Nurse data in ELSA were much larger than the models constructed using attributes from the Core ELSA data. The difference of model size across the different attribute sets for each dataset was in general smaller than the difference of model size across the datasets.

Table 6.6 shows aggregate metrics of TIBI and TIBIW attribute importances. TIBI attribute importances are reported for both TIBI and All configurations and TIBIW attribute importances are reported for both TIBIW and All configurations.

Table 6.3: $\overline{\text{F-Measure}}$ values obtained by the conventional Decision Tree algorithm with different configurations. The highest $\overline{\text{F-Measure}}$ value for each classification problem (dataset) is shown in boldface.

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	0.497	**0.526**	0.497	0.497	**0.526**
N-Arthritis	**0.399**	**0.399**	**0.399**	0.398	**0.399**
N-Cataract	**0.459**	**0.459**	**0.459**	**0.459**	0.458
N-Dementia	0.504	**0.531**	0.504	0.498	0.526
N-Diabetes	0.477	**0.490**	0.477	0.476	0.489
N-High Blood Pressure	**0.409**	**0.409**	0.405	0.405	0.408
N-Heart Attack	**0.493**	0.490	**0.493**	**0.493**	0.489
N-Osteoporosis	0.490	**0.512**	0.490	0.490	**0.512**
N-Parkinson's	0.500	**0.510**	0.500	0.500	0.509
N-Stroke	**0.493**	**0.493**	**0.493**	0.492	0.492
C-Angina	**0.510**	0.507	**0.510**	**0.510**	0.506
C-Arthritis	0.438	0.424	**0.439**	0.438	0.418
C-Cataract	**0.464**	0.461	**0.464**	**0.464**	0.460
C-Dementia	0.512	0.494	**0.514**	0.510	0.485
C-Diabetes	0.495	**0.510**	0.495	0.495	0.509
C-High Blood Pressure	0.420	**0.428**	0.420	0.420	0.427
C-Heart Attack	0.500	**0.502**	0.500	0.499	0.501
C-Osteoporosis	**0.498**	0.488	**0.498**	**0.498**	0.483
C-Parkinson's	0.505	**0.520**	0.507	0.505	0.518
C-Stroke	**0.500**	0.487	**0.500**	0.495	0.485

6.6.2 Nested Trees results

Nested Trees is a complex, longitudinally-aware classification algorithm and can generally be expected to produce much more accurate models than the longitudinally unaware Decision Tree. In these experiments, it produced much better F-Measure results than Decision Tree did, however its AUROC values were generally low. Tables 6.7 and 6.8 show the comparisons of the predictive accuracy measures obtained by the Nested Trees algorithms (its non-binary version) when using the 5 configurations of features.

Regarding Table 6.7, with F-measure results, overall the best results were

Table 6.4: $\overline{\text{AUROC}}$ values obtained by the conventional Decision Tree algorithm with different configurations. The highest $\overline{\text{AUROC}}$ value for each classification problem (dataset) is shown in boldface.

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	0.533	**0.566**	0.532	0.532	**0.566**
N-Arthritis	0.577	0.577	**0.578**	0.574	0.577
N-Cataract	0.681	0.681	0.681	**0.682**	0.681
N-Dementia	0.577	**0.695**	0.578	0.537	0.689
N-Diabetes	0.542	**0.626**	0.542	0.529	0.625
N-High Blood Pressure	**0.593**	**0.593**	**0.593**	**0.593**	**0.593**
N-Heart Attack	0.583	**0.620**	0.582	0.581	0.619
N-Osteoporosis	0.583	**0.629**	0.583	0.583	**0.629**
N-Parkinson's	**0.647**	0.615	**0.647**	0.646	0.614
N-Stroke	0.577	**0.637**	0.577	0.563	**0.637**
C-Angina	0.680	**0.688**	0.680	0.674	**0.688**
C-Arthritis	0.747	**0.785**	0.746	0.744	0.773
C-Cataract	0.738	**0.742**	0.738	0.737	0.740
C-Dementia	0.806	**0.812**	0.786	0.702	0.798
C-Diabetes	0.701	**0.706**	0.701	0.700	0.704
C-High Blood Pressure	0.652	**0.684**	0.653	0.653	0.682
C-Heart Attack	0.682	**0.688**	0.683	0.681	**0.688**
C-Osteoporosis	0.717	**0.730**	0.716	0.708	0.723
C-Parkinson's	0.715	**0.751**	0.730	0.720	0.748
C-Stroke	0.674	**0.695**	0.671	0.672	0.692

obtained when using Timeless and All configurations, which led to the best result in 8 of the 20 datasets. The All configuration was the overall winner, because it was the single winner in 8 datasets, whilst Timeless was the single winner in 6 datasets (it was a joint winner in two datasets).

Regarding Table 6.8, with AUROC results, the Timeless configuration clearly led to the best results overall, since this configuration was the winner in 11 of the 20 datasets – although this includes a joint win with Raw in 3 datasets. The TIBIW configuration led to the worst results, this configuration was not a winner in any dataset.

Table 6.5: Average sizes of the Decision Tree models constructed using different configurations (measured as described in section 6.5.1)

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	208	203	207	195	176
N-Arthritis	380	380	382	362	328
N-Cataract	374	374	372	346	323
N-Dementia	260	271	255	86	90
N-Diabetes	520	516	522	474	432
N-High Blood Pressure	325	325	325	310	304
N-Heart Attack	302	304	303	276	254
N-Osteoporosis	344	338	342	331	303
N-Parkinson's	204	203	202	198	190
N-Stroke	315	304	314	225	214
C-Angina	23	22	23	25	23
C-Arthritis	71	72	80	72	70
C-Cataract	61	59	62	70	65
C-Dementia	68	65	83	93	86
C-Diabetes	46	44	46	57	57
C-High Blood Pressure	64	63	63	70	64
C-Heart Attack	49	47	49	54	53
C-Osteoporosis	74	72	85	93	88
C-Parkinson's	67	66	75	78	82
C-Stroke	32	32	36	44	40

Table 6.9 shows the average sizes of the Nested Trees models constructed during the experiments. In general, the model sizes constructed by the Nested Trees algorithms are much larger than the model sizes constructed by the conventional (longitudinal unaware) decision tree algorithms, as can be observed by comparing the results in Tables 6.9 and 6.5, respectively. However, the qualitative results are similar in both tables. That is, the main pattern in the results of Table 6.9 is that, in general, the Nested Trees models constructed using attributes from the Nurse data in ELSA were much larger than the models constructed using attributes from the Core ELSA data. In addition, the difference of model size across the different configurations for each dataset was in general smaller than the difference of model

Table 6.6: Average aggregate attribute importances of TIBI and TIBIW attributes in their respective Decision Tree experiment configurations (measured as described in section 6.5.1). Notably high values (above 0.1) are highlighted in bold.

Classification problem	TIBI	TIBI All	TIBIW	TIBIW All
N-Angina	0.0010	0.0010	0.0019	0.0018
N-Arthritis	0.0015	0.0014	0.0047	0.0046
N-Cataract	0.0013	0.0012	0.0061	0.0057
N-Dementia	0.0330	0.0321	**0.1741**	**0.1622**
N-Diabetes	0.0007	0.0007	0.0203	0.0198
N-High Blood Pressure	0.0011	0.0010	0.0139	0.0140
N-Heart Attack	0.0009	0.0009	0.0432	0.0397
N-Osteoporosis	0.0005	0.0005	0.0012	0.0013
N-Parkinson's	0.0003	0.0003	0.0205	0.0207
N-Stroke	0.0010	0.0009	0.0188	0.0183
C-Angina	0.0001	0.0001	0.0188	0.0174
C-Arthritis	**0.1392**	**0.1331**	**0.2221**	**0.2277**
C-Cataract	0.0143	0.0143	0.0505	0.0480
C-Dementia	**0.1792**	**0.1651**	**0.2698**	**0.2704**
C-Diabetes	0.0016	0.0015	0.0507	0.0488
C-High Blood Pressure	0.0170	0.0161	0.0387	0.0371
C-Heart Attack	0.0003	0.0003	0.0260	0.0249
C-Osteoporosis	0.0710	0.0666	**0.1535**	**0.1485**
C-Parkinson's	0.0486	0.0467	0.0230	0.0221
C-Stroke	0.0153	0.0140	0.0857	0.0818

size across the datasets.

Table 6.10 shows aggregate metrics of TIBI and TIBIW attribute importances. TIBI attribute importances are reported for both TIBI and All configurations and TIBIW attribute importances are reported for both TIBIW and All configurations.

6.6.3 Binary Nested Trees results

Binary Nested Trees is a more advanced version of the Nested Trees algorithm, yet it tends to produce much simpler classification models. Its branching factor is limited to 2, so it does not produce nearly as many nodes. As a result, leaf

Table 6.7: $\overline{\text{F-Measure}}$ values obtained by the non-binary version of the Nested Trees algorithm with different configurations. The highest $\overline{\text{F-Measure}}$ value for each classification problem (dataset) is shown in boldface.

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	0.575	0.580	0.575	0.578	**0.616**
N-Arthritis	0.394	0.394	0.396	0.413	**0.414**
N-Cataract	**0.415**	**0.415**	**0.415**	0.410	0.410
N-Dementia	0.555	**0.563**	0.556	0.543	0.552
N-Diabetes	0.555	**0.561**	0.556	0.556	0.531
N-High Blood Pressure	**0.440**	**0.440**	0.436	0.432	0.428
N-Heart Attack	0.520	0.524	0.526	**0.550**	0.532
N-Osteoporosis	0.534	0.540	0.535	**0.552**	0.532
N-Parkinson's	0.546	0.551	**0.556**	0.555	0.551
N-Stroke	0.528	**0.533**	0.526	0.497	0.499
C-Angina	0.569	0.568	0.607	0.595	**0.632**
C-Arthritis	0.430	0.433	0.444	0.438	**0.456**
C-Cataract	0.477	**0.482**	0.479	0.471	0.471
C-Dementia	0.512	0.521	0.593	0.506	**0.596**
C-Diabetes	0.542	0.547	**0.576**	0.549	0.567
C-High Blood Pressure	0.440	0.442	0.471	0.474	**0.506**
C-Heart Attack	0.502	0.508	0.500	0.506	**0.516**
C-Osteoporosis	0.527	0.532	0.547	0.535	**0.560**
C-Parkinson's	0.502	**0.506**	0.503	0.492	0.496
C-Stroke	0.569	**0.576**	0.569	0.561	0.568

subsets tend to contain more instances, resulting in a larger variety of regression values, smoother ROC curve, and therefore larger AUROC values. Tables 6.11 and 6.12 show the comparisons of predictive accuracies obtained by the Binary Nested Trees algorithm when using 5 configurations of features.

In Table 6.11, regarding F-measure results, overall the best results were obtained by the All and Timeless configurations, which led to the best results in 9 and 8 (out of 20) datasets, respectively. In addition, All was the single winner in 8 datasets (and joint winner in just one dataset), whilst Timeless was a joint winner in only 5 datasets (and joint winner in 3 datasets).

In Table 6.12, regarding AUROC results, overall the best results were obtained

Table 6.8: $\overline{\text{AUROC}}$ values obtained by the non-binary version of the Nested Trees algorithm with different configurations. The highest $\overline{\text{AUROC}}$ value for each classification problem (dataset) is shown in boldface.

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	0.575	0.592	**0.620**	0.531	0.568
N-Arthritis	**0.564**	**0.564**	0.558	0.540	0.533
N-Cataract	**0.649**	**0.649**	0.645	0.615	0.610
N-Dementia	0.561	0.615	0.579	0.551	**0.624**
N-Diabetes	0.555	**0.596**	0.545	0.536	0.558
N-High Blood Pressure	**0.627**	**0.627**	0.620	0.586	0.580
N-Heart Attack	0.548	0.565	0.563	0.539	**0.566**
N-Osteoporosis	0.577	0.600	0.602	0.558	**0.604**
N-Parkinson's	0.661	0.645	**0.673**	0.621	0.616
N-Stroke	0.614	**0.645**	0.614	0.580	0.635
C-Angina	0.656	0.660	**0.746**	0.582	0.692
C-Arthritis	0.731	**0.750**	0.662	0.698	0.691
C-Cataract	0.732	0.734	**0.771**	0.714	0.755
C-Dementia	0.763	**0.766**	0.746	0.659	0.645
C-Diabetes	0.732	0.735	0.600	0.685	**0.737**
C-High Blood Pressure	0.668	**0.684**	0.615	0.662	0.625
C-Heart Attack	0.720	**0.723**	0.717	0.634	0.705
C-Osteoporosis	0.700	0.706	**0.753**	0.634	0.720
C-Parkinson's	0.697	**0.714**	0.629	0.618	0.608
C-Stroke	0.670	**0.681**	0.545	0.594	0.616

by the All and Timeless configurations, each of which led to the best results in 10 (out of 20) datasets. All obtained slightly better results, since it was the single winner in 9 datasets (and joint winner in just one dataset); whilst Timeless was the single winner in 8 datasets (and joint winner in 2 datasets).

Table 6.13 shows the average sizes of the Binary Nested Trees models constructed during the experiments. Similarly to the results reported in Tables 6.5 and 6.9, the main pattern in the results of Table 6.13 is that, in general, the Binary Nested Trees models constructed using attributes from the Nurse data in ELSA were larger than the models constructed using attributes from the Core ELSA data. In addition, again, the difference of model size across the different

Table 6.9: Average model sizes (number of tree nodes) of the models constructed by the non-binary version of the Nested Trees algorithm with different configurations (measured as described in section 6.5.1).

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	1185	1081	1180	1146	1234
N-Arthritis	2343	2343	2343	2215	2378
N-Cataract	2229	2229	2225	2078	2192
N-Dementia	1409	1237	1403	470	489
N-Diabetes	2749	2682	2744	2749	2699
N-High Blood Pressure	1880	1880	1875	1880	1883
N-Heart Attack	1894	1633	1883	1716	1875
N-Osteoporosis	1896	1695	1890	1897	1913
N-Parkinson's	1293	1116	1288	1293	1309
N-Stroke	2096	1935	2090	2096	2116
C-Angina	140	129	140	152	136
C-Arthritis	432	415	416	442	437
C-Cataract	321	279	320	372	320
C-Dementia	443	430	420	594	443
C-Diabetes	298	263	294	371	303
C-High Blood Pressure	389	370	377	421	381
C-Heart Attack	278	243	279	313	279
C-Osteoporosis	404	357	397	520	397
C-Parkinson's	454	467	429	533	473
C-Stroke	216	201	218	216	224

configurations for each dataset was in general smaller than the difference of model size across the datasets.

Table 6.14 shows aggregate metrics of TIBI and TIBIW attribute importances. TIBI attribute importances are reported for both TIBI and All configurations and TIBIW attribute importances are reported for both TIBIW and All configurations.

6.6.4 Number of detected monotonicity constraints

The number of detected monotonicity constraints varied significantly between different classification problems. The variation that was already shown in chapter 4

Table 6.10: Average aggregate attribute importances of TIBI and TIBIW attributes in their respective Nested Trees experiment configurations (measured as described in section 6.5.1). Notably high values (above 0.1) are highlighted in bold.

Classification problem	TIBI	TIBI All	TIBIW	TIBIW All
N-Angina	0.0511	0.0485	0.0026	0.0001
N-Arthritis	0.0650	0.0594	0.0085	0.0009
N-Cataract	0.0215	0.0187	0.0010	0.0000
N-Dementia	**0.1665**	**0.1479**	0.0190	0.0020
N-Diabetes	**0.1089**	**0.1099**	0.0032	0.0001
N-High Blood Pressure	**0.1170**	**0.1230**	0.0017	0.0000
N-Heart Attack	**0.1294**	**0.1160**	0.0182	0.0022
N-Osteoporosis	0.0317	0.0289	0.0040	0.0005
N-Parkinson's	**0.1161**	0.0919	0.0337	0.0078
N-Stroke	0.0572	0.0474	0.0120	0.0020
C-Angina	**0.3618**	**0.3082**	0.0955	0.0199
C-Arthritis	**0.3581**	**0.2646**	**0.1286**	0.0342
C-Cataract	**0.2734**	**0.2470**	0.0395	0.0048
C-Dementia	**0.2708**	**0.1589**	**0.1876**	0.0738
C-Diabetes	**0.3110**	**0.2238**	**0.1163**	0.0325
C-High Blood Pressure	**0.2711**	**0.1871**	**0.1228**	0.0412
C-Heart Attack	**0.3286**	**0.2554**	0.0600	0.0093
C-Osteoporosis	**0.2156**	**0.1819**	0.0446	0.0069
C-Parkinson's	**0.4199**	**0.2908**	**0.1872**	0.0576
C-Stroke	**0.5457**	**0.4770**	0.0552	0.0050

has now been amplified by the class balancing being applied before the monotonicity detection occurred. Three datasets did not contain any monotonic relations strong enough to be used as monotonicity constraints. After the addition of TIBI and TIBIW derived attributes, the total number of predictor attributes increased from 134 up to 1528 in the ELSA Nurse datasets and from 180 to 909 in the ELSA Core datasets, allowing for many more monotonic relations to be discovered.

Table 6.15 shows the total number of monotonicity constraints detected in each dataset. The table includes the number of monotonic constraints detected when

Table 6.11: $\overline{\text{F-Measure}}$ values obtained by the Binary Nested Trees algorithm with different configurations. The highest $\overline{\text{F-Measure}}$ value for each classification problem (dataset) is shown in boldface.

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	0.510	0.509	**0.511**	0.508	0.509
N-Arthritis	**0.406**	**0.406**	**0.406**	0.405	0.405
N-Cataract	0.460	0.460	0.461	0.460	**0.462**
N-Dementia	0.519	**0.521**	0.519	0.517	0.519
N-Diabetes	0.509	0.509	0.508	**0.510**	0.505
N-High Blood Pressure	**0.432**	**0.432**	**0.432**	0.426	**0.432**
N-Heart Attack	0.516	0.515	0.515	0.514	**0.530**
N-Osteoporosis	0.517	0.518	0.517	0.517	**0.520**
N-Parkinson's	**0.501**	**0.501**	0.500	**0.501**	0.490
N-Stroke	0.514	0.514	0.514	0.514	**0.523**
C-Angina	0.518	0.518	0.518	0.516	**0.528**
C-Arthritis	0.436	0.435	0.436	**0.438**	0.417
C-Cataract	0.472	0.472	0.472	0.472	**0.481**
C-Dementia	0.523	**0.527**	0.524	0.519	0.523
C-Diabetes	**0.512**	**0.512**	0.511	0.511	0.509
C-High Blood Pressure	0.431	**0.432**	0.429	0.428	0.431
C-Heart Attack	0.516	0.519	0.515	0.513	**0.522**
C-Osteoporosis	0.520	0.520	0.519	0.517	**0.524**
C-Parkinson's	0.504	0.503	0.503	**0.506**	0.493
C-Stroke	0.518	**0.520**	0.519	0.514	0.494

using the Timeless and All configurations. Recall that the detection of monotonicity constraints happens before model construction, so it does not depend on the model type. Hence, the same monotonic constraints were detected in all Timeless configuration experiments and the same in all All configuration experiments, for all types of classification algorithms used in the experiments.

6.7 Statistical significance tests

Statistical tests have been performed to measure the significance of the differences between the predictive accuracy measurements of the constructed models. For a

Table 6.12: $\overline{\text{AUROC}}$ values obtained by the Binary Nested Trees algorithm with different configurations. The highest $\overline{\text{AUROC}}$ value for each classification problem (dataset) is shown in boldface.

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	0.634	0.630	**0.659**	0.607	0.640
N-Arthritis	**0.620**	**0.620**	0.619	0.607	0.606
N-Cataract	0.703	0.703	0.704	0.702	**0.719**
N-Dementia	0.667	**0.760**	0.661	0.685	**0.760**
N-Diabetes	0.707	0.704	0.705	0.695	**0.728**
N-High Blood Pressure	0.693	0.693	0.695	0.682	**0.707**
N-Heart Attack	0.664	0.663	0.663	0.665	**0.666**
N-Osteoporosis	0.716	0.721	0.716	0.708	**0.734**
N-Parkinson's	0.661	**0.761**	0.659	0.675	0.653
N-Stroke	0.652	**0.670**	0.645	0.636	0.657
C-Angina	0.716	**0.751**	0.699	0.657	0.703
C-Arthritis	0.784	0.781	0.779	0.775	**0.791**
C-Cataract	0.739	0.746	0.741	0.724	**0.763**
C-Dementia	0.746	**0.800**	0.736	0.663	0.770
C-Diabetes	0.736	**0.738**	0.728	0.713	0.722
C-High Blood Pressure	0.672	0.672	0.672	0.667	**0.697**
C-Heart Attack	0.695	**0.717**	0.684	0.636	0.692
C-Osteoporosis	0.722	**0.725**	0.709	0.675	0.758
C-Parkinson's	0.732	**0.768**	0.689	0.661	0.743
C-Stroke	0.707	0.736	0.691	0.642	**0.742**

fair comparison, the three classification algorithms were compared in five groups of measurements, grouped based on their usage the monotonicity features. All statistical tests have been performed using the Friedman test with the Holm post-hoc test.

Tables 6.16, 6.17, 6.18, 6.19 and 6.20 show statistical significance tests of the F-Measure results, grouped by the configuration of the monotonicity-based features. The results are very similar across all configurations - Nested Trees beats both the Binary Nested Trees and the conventional Decision Tree algorithm. The only exception is the TIBIW experiments, where binary conversion did not result in a significant difference in Nested Trees algorithms.

Table 6.13: Average sizes of the models constructed by the Binary Nested Trees algorithm with different configurations (measured as described in section 6.5.1).

Classification problem	Raw	Timeless	TIBI	TIBIW	All
N-Angina	163	154	162	157	136
N-Arthritis	240	245	238	237	207
N-Cataract	134	134	134	123	115
N-Dementia	113	122	111	38	39
N-Diabetes	280	280	277	257	221
N-High Blood Pressure	314	314	307	314	314
N-Heart Attack	222	219	224	222	222
N-Osteoporosis	197	189	197	190	180
N-Parkinson's	103	101	99	103	98
N-Stroke	194	194	191	138	145
C-Angina	28	26	29	29	28
C-Arthritis	85	95	86	87	84
C-Cataract	51	49	51	60	54
C-Dementia	64	59	53	89	64
C-Diabetes	53	50	47	67	65
C-High Blood Pressure	76	81	67	83	77
C-Heart Attack	49	49	48	53	51
C-Osteoporosis	76	73	75	98	93
C-Parkinson's	77	79	64	90	97
C-Stroke	36	35	34	49	46

Tables 6.21, 6.22, 6.23, 6.24 and 6.25 show the same statistical tests using AUROC values. The results are again very similar across all configurations, this time the Binary Nested Trees algorithm beats both of the other algorithms. Once again the pattern is broken in the TIBIW experiments, where the difference between the Binary Nested Trees algorithm and the conventional Decision Tree algorithm is not statistically significant.

Table 6.14: Average aggregate attribute importances of TIBI and TIBIW attributes in their respective Binary Nested Trees experiment configurations. (measured as described in section 6.5.1) Notably high values (above 0.1) are highlighted in bold.

Classification problem	TIBI	TIBI All	TIBIW	TIBIW All
N-Angina	0.0027	0.0037	0.0485	0.0511
N-Arthritis	0.0091	0.0000	0.0631	0.0630
N-Cataract	0.0010	0.0008	0.0227	0.0228
N-Dementia	0.0181	0.0044	**0.1772**	**0.1747**
N-Diabetes	0.0032	0.0042	0.0982	**0.1002**
N-High Blood Pressure	0.0017	0.0028	**0.1114**	**0.1133**
N-Heart Attack	0.0197	0.0229	**0.1190**	**0.1074**
N-Osteoporosis	0.0038	0.0138	0.0323	0.0226
N-Parkinson's	0.0401	0.0263	**0.1251**	**0.1033**
N-Stroke	0.0131	0.0355	0.0550	0.0345
C-Angina	0.0886	0.0277	**0.3387**	**0.3070**
C-Arthritis	**0.1249**	0.0316	**0.3520**	**0.3379**
C-Cataract	0.0455	0.0192	**0.2234**	**0.2087**
C-Dementia	**0.1659**	0.0463	**0.2497**	**0.2208**
C-Diabetes	**0.1094**	0.0267	**0.3102**	**0.2963**
C-High Blood Pressure	**0.1143**	0.0348	**0.2446**	**0.2132**
C-Heart Attack	0.0617	0.0217	**0.3107**	**0.3064**
C-Osteoporosis	0.0438	0.0320	**0.2088**	**0.1969**
C-Parkinson's	**0.1709**	0.0321	**0.3784**	**0.3793**
C-Stroke	0.0518	0.0137	**0.4616**	**0.4706**

6.8 Experiment runtime

A large number of algorithm runs has been conducted as part of Nested Trees evaluation, with different combinations of features and different classification problems. The experiments took many days to run, with some algorithm variations being much more time-costly than others.

Overall, the fastest experiments were the ones using the Binary Nested Trees algorithm without any additional features. After the pre-processing steps have been conducted, the Binary Nested Trees models would take only a few seconds

Table 6.15: Average number of monotonicity constraints detected with and without the derived attributes.

Classification problem	Timeless	All
N-Angina	15	191
N-Arthritis	0	184
N-Cataract	0	191
N-Dementia	17	197
N-Diabetes	10	198
N-High Blood Pressure	0	164
N-Heart Attack	20	176
N-Osteoporosis	13	177
N-Parkinson's	11	221
N-Stroke	18	181
C-Angina	50	248
C-Arthritis	35	223
C-Cataract	41	239
C-Dementia	53	213
C-Diabetes	35	225
C-High Blood Pressure	29	236
C-Heart Attack	35	203
C-Osteoporosis	42	205
C-Parkinson's	42	200
C-Stroke	47	208

to be constructed, depending on the size of the final model. The slowest were, naturally, the non-binary Nested Trees experiments with all of the features enabled.

Binary Nested Trees had a tremendous construction time advantage over the non-binary Nested Trees variant due, as mentioned in the previous evaluation, to their significantly smaller branching factor. The time spent on performing the binary conversions was negligible compared to the time spent constructing the large numbers of branches in the non-binary Nested Trees.

The addition of derived attributes would always lead to an increase in the

Table 6.16: Results of the post-hoc Holm test at 5% significance for the three algorithms without monotonicity features based on F-Measures.

Algorithm	Average Rank	p-value	Holm
NT Raw (control)	1.525	–	–
BNT Raw	1.8	0.385	0.05
Tree Raw	2.675	0.0003	0.025

Table 6.17: Results of the post-hoc Holm test at 5% significance for the three algorithms with Timeless monotonicity constraints based on F-Measures.

Algorithm	Average Rank	p-value	Holm
NT Timeless (control)	1.4	–	–
BNT Timeless	1.95	0.082	0.05
Tree Timeless	2.65	0.00008	0.025

Table 6.18: Results of the post-hoc Holm test at 5% significance for the three algorithms with TIBI derived attributes based on F-Measures.

Algorithm	Average Rank	p-value	Holm
NT TIBI (control)	1.35	–	–
BNT TIBI	1.95	0.058	0.05
Tree TIBI	2.7	0.00001	0.025

Table 6.19: Results of the post-hoc Holm test at 5% significance for the three algorithms with TIBIW derived attributes based on F-Measures.

Algorithm	Average Rank	p-value	Holm
NT TIBIW (control)	1.5	–	–
BNT TIBIW	1.7	0.527	0.05
Tree TIBIW	2.8	0.00003	0.025

Table 6.20: Results of the post-hoc Holm test at 5% significance for the three algorithms with all three monotonicity features based on F-Measures.

Algorithm	Average Rank	p-value	Holm
NT All (control)	1.35	–	–
BNT All	2.025	0.033	0.05
Tree All	2.625	0.00005	0.025

Table 6.21: Results of the post-hoc Holm test at 5% significance for the three algorithms without monotonicity features based on $\overline{\text{AUROC}}$.

Algorithm	Average Rank	p-value	Holm
BNT Raw (control)	1.175	–	–
Tree Raw	2.35	0.00002	0.05
NT Raw	2.475	0.00004	0.025

Table 6.22: Results of the post-hoc Holm test at 5% significance for the three algorithms with Timeless monotonicity constraints based on $\overline{\text{AUROC}}$.

Algorithm	Average Rank	p-value	Holm
BNT Timeless (control)	1.3	–	–
Tree Timeless	2.125	0.009	0.05
NT Timeless	2.575	0.000006	0.025

Table 6.23: Results of the post-hoc Holm test at 5% significance for the three algorithms with TIBI derived attributes based on $\overline{\text{AUROC}}$.

Algorithm	Average Rank	p-value	Holm
BNT TIBI (control)	1.45	–	–
NT TIBI	2.15	0.027	0.05
Tree TIBI	2.4	0.003	0.025

Table 6.24: Results of the post-hoc Holm test at 5% significance for the three algorithms with TIBIW derived attributes based on $\overline{\text{AUROC}}$.

Algorithm	Average Rank	p-value	Holm
BNT TIBIW (control)	1.35	–	–
Tree TIBIW	1.85	0.11	0.05
NT TIBIW	2.8	0.0000004	0.025

Table 6.25: Results of the post-hoc Holm test at 5% significance for the three algorithms with all three monotonicity features based on $\overline{\text{AUROC}}$.

Algorithm	Average Rank	p-value	Holm
BNT All (control)	1.2	–	–
Tree All	2.2	0.001	0.05
NT All	2.6	0.00001	0.025

model construction times, even when the final models were smaller than ones constructed without the derived attributes. TIBI attributes have had a somewhat noticeable effecting the model construction time as they would add one additional constructed attribute per each longitudinal attribute present in the original dataset. The introduction of TIBI attributes would generally increase the construction time by about 50%, depending on the classification problem. The addition of TIBIW was easily the most time-costly operation. As mentioned above, the number of TIBIW attributes scaled quadratically with the number of longitudinal attributes in the dataset. These would need to be calculated for each of the data instances, so the time complexity of adding them would naturally scale linearly with the number of data instances. Additionally, the process of constructing a specific TIBIW value includes iteration over all of the time points available in the corresponding pair of longitudinal attributes, so the time complexity would also scale linearly with the number of time points in the dataset. TIBIW attributes have also had a major effect on the model construction times since the number of attributes that each of the inner nodes could consider was significantly higher. In

the experiments conducted for this evaluation, the experiments with the TIBIW attributes easily took over 80% of the total computational time. Overall, while the addition of these derived attributes has produced some interesting effects, the time cost of the experiments has increased very significantly.

Timeless monotonicity detection has had a somewhat noticeable impact on the initial experiments with the ELSA datasets, adding a few seconds to the pre-processing time before the models would be constructed. The slight reduction in the average model sizes that resulted from the employment of the Timeless monotonicity constraints did not noticeably offset this cost. The addition of derived attributes has always significantly impacted the pre-processing time spent on detecting monotonic relations and constructing constraints. This was especially noticeable in all of the TIBI and TIBIW experiments where the number of constructed predictive attributes completely overshadowed the original number of attributes. Since Timeless monotonicity detection only evaluates the monotonicity between each attribute and the class, the cost of the Timeless monotonicity detection scales linearly with the number of predictive attributes.

A variety of optimisations were put in place in an attempt to accelerate the process of algorithm evaluation. Since the TIBI and TIBIW attributes only depend on the values from individual data instances and do not contain any population-specific data, these attributes were constructed only once prior to the main experimentation loop, in order to avoid having to re-construct them individually for each experiment. Such optimisation was not possible to implement for the Timeless monotonicity detection though, since the monotonic relations detected by it were always specific to the current training set, and its contents change on every repetition of the experiment and on every fold of the cross-validation process. Basic parallelisation was implemented in order to run many experiments in parallel, but the model construction process would always run linearly.

Overall, the algorithm runtime would always noticeably increase when new

major features were added, especially ones that significantly increased the size of the training set. The time-expensive evaluation was largely a result of a very large number of experiments being conducted, with each individual dataset-algorithm-feature combination being evaluated through 30 repetitions of 10-cross validations with different random seeds in order to produce highly-reliable estimations of average predictive accuracy metrics, model sizes and attribute importances.

6.9 Discussion of the Results

The results acquired from these experiments provide valuable insights into the properties of the Nested Tree algorithm, the proposed monotonicity detection methods, and interesting details of some of the classification problems.

Broadly speaking, the predictive accuracy results show the same principal differences between the used classification algorithms as the ones demonstrated in Chapter 5.

The conventional (longitudinally unaware) Decision Tree algorithm struggled to build accurate classification models using the longitudinal datasets and generally did not make good use of the additional monotonicity-based features. It also continued to struggle with the datasets that have been class-balanced, over-prioritising accurate classification of the instances belonging to the minority class just because their weights have been dramatically increased. As a result, it commonly achieved very low F-Measure values, despite making attempts to learn to distinguish the two classes. It was, however, capable of learning this distinction at the regression level - by constructing leaf nodes that predict the average value of the class instead of selecting one of its categorical values. For this reason, this model has a potential to be used as part of the ensemble where multiple such models are constructed and their regression predictions are combined and thresholded to make accurate class predictions. This is precisely the reason why algorithms

like XGBoost are able to achieve great classification performance despite using relatively weak Decision Tree models.

The (non-binary version) Nested Trees algorithm builds large, longitudinally aware classification models, makes use of the derived attributes, and provides much more accurate classification models, as indicated by higher F-Measure values. It is naturally inclined to make more use of the derived TIBI and TIBIW attributes, since their values get included into the structures of longitudinal attributes, thus occupying a significant amount of the search space during inner tree construction. Large model size inevitably leads to very small instance subsets in leaf nodes being used to produce regression predictions, thus severely reducing the variety of regression values that the model can produce. As a result, the ROC curve of Nested Trees models tends to have fewer points on it, creating a rough curve shape, resulting in a lower AUROC value. Interestingly, the Nested Tree algorithm seems to consistently benefit from the introduction of monotonicity constraints, resulting in slightly higher prediction accuracy metrics in almost all experiments. As shown before, Nested Tree models tend to be large, and without additional post-processing of the models, they are scarcely interpretable.

The Binary Nested Tree variant provides a compromise. It constructs relatively small classification models, comparable in size to the models constructed using a conventional Decision Tree algorithm. It has a small branching factor of 2, fewer nodes in the inner trees (after binary conversion deleting the unnecessary tree branches), and produces larger instance subsets in leaf nodes. Larger leaf nodes allow for more variety of regression values, which in turn results in a smoother ROC curve and higher AUROC values. Their predictive accuracy tends to be low, only marginally higher than that of conventional Decision Tree models. The addition of monotonicity-based features mainly results in Binary Nested Trees achieving an improvement in AUROC values and a reduction of model sizes. As mentioned above, conventional Decision Tree models have a potential for use in

ensemble models because of their small size and regression performance. The Binary Nested Trees variant performs even better than conventional Decision Trees, while also being longitudinally-aware, providing longitudinal interpretability, being able to effectively create and utilise monotonicity constraints, and benefiting from derived monotonicity-based attributes.

The Nested Trees algorithm with monotonicity features can also provide additional knowledge discovery value. It is able to detect longitudinal trends in the form of monotonicity-based TIBI and TIBIW attributes, use those attributes in classification, and evaluate their usefulness in predicting the class value, thus potentially discovering new, previously unexplored predictors.

For example, an interesting observation can be made about the experiments using the ELSA Nurse dataset with Dementia as the class attribute. That particular classification problem was greatly affected by the introduction of the TIBIW attributes, resulting in a massive drop in average model sizes. This can largely be attributed to the introduction of one specific derived TIBIW attribute called mch_mmlsre_TIBIW, which alone had an attribute importance value of 0.1602 in the Binary Nested Trees models. This attribute is a TIBIW attribute that represents the strength of the monotonic relation between the attribute mch, a numerical attribute that represents a patient's Blood Mean Corpuscular Haemoglobin level, and the attribute mmlsre, an ordinal categorical attribute that represents the outcome of a physical exercise test that involves the patient performing leg raise exercises with their eyes shut. The most common threshold used for this attribute was 0, so the classification models were making a lot of decisions based on whether the monotonic relationship between those two longitudinal attributes was positive or negative.

6.10 Conclusions

A new version of the Nested Trees algorithm has been developed that combined the longitudinally-aware model construction capabilities of Nested Trees with the automated monotonicity detection approach introduced in Chapter 4. The result of this combination is the first longitudinal classification algorithm that can detect and enforce monotonicity constraints. An extensive testing of the updated algorithm has been conducted, evaluating each of the its new features.

Timeless monotonicity constraints have once again been used to construct fully monotonic classification models and achieve significant improvements in the predictive performance of classification models.

TIBI attributes have been evaluated in a more complex longitudinal classification algorithm. Their effects varied across different classification problems (datasets), sometimes improving the classification performance and sometimes worsening it. Its most interesting effect was its ability to replace other attributes in the classification model construction process, sometimes resulting in noticeable reductions in model size. Overall, TIBI remains a highly dataset-specific type of feature that does not guarantee a consistent improvement of results. Perhaps it would make more sense to apply this attribute derivation in conjunction with an attribute selection algorithm to try and alleviate its negative effects.

TIBIW was a more successful attribute derivation method, providing a much more significant reduction in model sizes at the price of a very small reduction in predictive accuracy metrics. TIBIW attributes were heavily used in many classification experiments, especially the ones using the ELSA Core data. Despite being significantly more useful than TIBI, the TIBIW attribute derivation did not consistently improve prediction accuracy metrics and only provided some longitudinal correlation data that some models used instead of raw longitudinal attribute values.

Chapter 7

Conclusions and Future Research

7.1 Summary of Contributions

As a result of this research, several contributions were made to the fields of longitudinal and monotonic classification. The main contributions are as follows.

A novel way of detecting monotonic relations in classification datasets was derived and was used to automatically detect monotonic relationships in data and construct monotonicity constraints. These constraints were shown to be beneficial to the predictive accuracy of classification models. This is the first work where monotonicity constraints were created based on monotonicity relationships automatically detected from the training data, rather than provided by domain experts based on their assumptions. Automated monotonicity detection was also used to create two types of derived attributes based on monotonic relationships detected in longitudinal attributes.

A novel algorithm for longitudinal classification was developed. Nested Trees is the first longitudinally-aware classification algorithm that produces longitudinally-interpretable classification models. By utilising simple decision tree-based structures nested within a larger decision tree structure, it is able to construct classification models that analyse each longitudinal attribute as a whole, never analysing

its temporal values separately. It produces a classification model that can either be used as it is, or be reduced to a conventional decision tree model by deconstructing the nested structure and rebuilding a new decision tree out of the same nodes. This made these models "in principle" as interpretable as conventional decision trees are (but see the below caveat), whilst also providing some degree of longitudinal interpretability. This is also the first accessible longitudinal classification algorithm, with its source code made available on GitHub. The caveat about the interpretability of Nested Tree models is that they may have a larger number of tree nodes than conventional (non-nested) decision tree models, which could hinder the former's interpretability. However, it is interesting to note that, in the results reported in this thesis, the sizes of the models built by the binary version of the Nested Trees algorithm is often smaller than the sizes of the models built by a conventional decision tree algorithm.

The final main contribution of this study is the integration of the automatic approach for detecting monotonicity constraints in the data into the novel Nested Trees classification algorithm. Nested Trees algorithm with monotonicity features is the first longitudinally-aware classification algorithm capable of enforcing monotonicity constraints. It detects the monotonic relations in training sets automatically, and constructs attributes representing the strengths of the detected monotonic relationships without requiring any additional input from domain experts. It constructs interpretable longitudinal classification models, provides additional metrics for evaluating the importances of individual predictor attributes, whole longitudinal attributes, as well as aggregate metrics estimating the importances of entire groups of derived attributes (TIBI and TIBIW). It is also built as a highly-customisable algorithm, allowing for easy addition and evaluation of new algorithmic features, enabling it to be used in future research into longitudinal and monotonic classification.

7.2 Future Research Directions

Broadly speaking, this research opened up several venues of future research and provided the basis for quicker development and evaluation of new longitudinal classification algorithms. More specifically, the following research directions could be explored.

The most obvious next step would be to develop a more powerful longitudinal classification algorithm based on Nested Trees. As mentioned in Chapter 6, the binary variant of the Nested Trees algorithm has a tendency to produce smaller and more regression-capable models, and a promising research direction would be to develop an ensemble algorithm for longitudinal classification based on it. It could either be implemented as a boosting algorithm that boosts Nested Trees models, or as a longitudinally-aware random forest of Nested Trees.

Derived attributes have been shown to have some merit in longitudinal classification, sometimes leading to significant improvements in predictive accuracies and model sizes. Depending on the size and nature of the dataset, different derived attributes can be constructed in order to enhance the predictive performance of the models. The usage of derived attributes can become increasingly important when working with longitudinal datasets containing a very large number of time points (waves). In such cases, it might be useful to construct derived attributes and remove the original longitudinal values all together in order to reduce the amount of noise in the data and speed up model construction. However, this runs the risk of losing useful training data and instead introducing more useless and misleading predictor attributes. During this study, it has been observed in monotonicity experiments that the attributes that had a strong monotonic relation with the class were generally useful as predictor attributes. While this is not an unexpected observation, it suggests that these monotonicity detection metrics could be used in the task of attribute selection. Perhaps a combined approach can be created that would use monotonicity-based construction of derived attributes along with

a monotonicity-based attribute selection algorithm. This could potentially result in a highly-effective pre-processing algorithm for longitudinal classification tasks.

Some useful work has been done on automated monotonicity detection in datasets, but all research so far came down to estimating values of monotonicity metrics between two arrays of numerical values and using those values to either derive attributes or construct conventional monotonicity constraints. During the early stages of this study, some effort has been put into researching alternative methods for monotonicity detection. I have explored the possibility of detecting monotonic relations between more than two attributes at a time, in order to construct complex monotonic constraints or highly-valuable derived attributes. I have also explored the possibility of piece-wise monotonicity detection, a way to search for monotonic relations that only occur within specific attribute boundaries of the attribute values. Additionally, I have explored the possibility of analysing non-ordinal categorical attributes in classification datasets and using a greedy monotonicity-based search algorithm to turn their values into ordinal values that would be most useful for predicting the class. Neither of these approaches have found a real application yet, but I hope to eventually incorporate these as potentially useful features into my longitudinal-monotonic codebase.

It has been highlighted in the literature review section of this thesis that existing longitudinal approaches are not able to assimilate new data into their models, e.g. when new time points become available in the original datasets. While it is possible to solve this problem by simply re-training the models using the newly available data, one interesting approach remains unexplored: shifting the time point indexes in the classified instances before using them as input for the classification model. This would effectively allow the previously constructed classification models to use the data from new time points, by treating them as if they were the same time points that the model was originally trained on. Additionally, derived attributes representing longitudinal monotonic trends, such as

the proposed TIBI and TIBIW attributes can be derived for each of the classified instances and used directly by the models previously constructed using the TIBI and TIBIW attributes derived from the previous set time points. This venue of research is particularly interesting as it practically offers the possibility of creating longitudinal classification models inherently capable of predicting future class values. It would be interesting to conduct further studies in this direction, though it would perhaps be better to first focus on creating more accessible longitudinal classification algorithms and more accessible longitudinal classification datasets in order to provide a stronger base for algorithm evaluation.

Finally, further experiments could be performed with other types of longitudinal datasets, beyond the 20 classification problems based on the ELSA (English Longitudinal Study of Ageing) datasets used in this research, in order to further evaluate the longitudinal classification algorithms proposed in this research.

Bibliography

[1] Luite Stegeman Ad Feelders. "On Generating all Optimal Monotone Classifications". In: *2011 11th IEEE International Conference on Data Mining*. 2011.

[2] Tijmen Kolkman Ad Feelders. "Exploiting Monotonicity Constraints to Reduce Label Noise: an Experimental Evaluation". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. 2016.

[3] M Ahmadzadeh et al. "Predictors of the rate of cognitive decline in older adults using machine learning". In: *PLoS One* (2023).

[4] Jose-Ramon Cano et al. "Monotonic classification: An overview on algorithms, performance measures and data sets". In: *Neurocomputing 341 (2019) 168–182*. 2019.

[5] Yoh-Han Pao Arie Ben-David Leon Sterling. "Learning and classification of monotonic ordinal concepts". In: *Computational Intelligence 5(1989), pp. 45-49*. 1989.

[6] A Ben-David. "Monotonicity maintenance in information-theoretic machine learning algorithms". In: *Machine Learning, 19, 29-43 (1995)*. 1995.

[7] Arie Ben-David. "Automatic Generation of Symbolic Multiattribute Ordinal Knowledge-Based DSSs: Methodology and Applications*". In: *Decision Sciences* 23.6 (1992), pp. 1357–1372.

[8] Arie Ben-David, Leon Sterling, and TriDat Tran. "Adding monotonicity to learning algorithms may impair their accuracy". In: *Expert Systems with Applications* 36 (Apr. 2009), pp. 6627–6634. DOI: 10.1016/j.eswa.2008.08.021.

[9] Jacob Benesty et al. "Pearson Correlation Coefficient". In: *Noise Reduction in Speech Processing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 1–4. ISBN: 978-3-642-00296-0. DOI: 10.1007/978-3-642-00296-0_5. URL: https://doi.org/10.1007/978-3-642-00296-0_5.

[10] Max Bramer. "Principles of Data Mining (fourth edition)". In: Springer, 2020. Chap. 7.3.

[11] James Brookhouse and Fernando E. B. Otero. "Monotonicity in Ant Colony Classification Algorithms". In: *Swarm Intelligence*. Springer International Publishing, 2016, pp. 137–148.

[12] J.R. Cano and S. García. "Training set selection for monotonic ordinal classification". In: *Data Knowledge Engineering* 112 (2017), pp. 94–105.

[13] Chih-Chuan Chen and Sheng-Tun Li. "Credit rating with a monotonicity-constrained support vector machine model". In: *Expert Systems with Applications* 41.16 (2014), pp. 7235–7247.

[14] Shuo Chen and F. DuBois Bowman. "A novel support vector classifier for longitudinal high-dimensional data and its application to neuroimaging data". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 4(6) (2011), pp. 604–611.

[15] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD*. 2016, pp. 785–794.

[16] Thomas M. Cover and Joy A. Thomas. "Elements of Information Theory". In: Wiley-Interscience, 2006. Chap. 2.1.

[17] A Dalaka et al. "Modelling the effects of environmental conditions on apparent photosynthesis of Stipa bromoides by machine learning tools". In: *Ecological Modelling* 129 (May 2000), pp. 245–257. DOI: 10.1016/S0304-3800(00)00237-4.

[18] H.A.M. Daniels and M.V. Velikova. *Derivation of Monotone Decision Models from Non-Monotone Data*. Tech. rep. 2003.

[19] Janez Demšar. "Statistical comparisons of classifiers over multiple data sets." In: *The Journal of Machine Learning Research* 7 (2006), pp. 1–30. ISSN: 1533-7928.

[20] Wei Du et al. "A longitudinal support vector regression for prediction of ALS score". In: *2015 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 2015, pp. 1586–1590.

[21] Wouter Duivesteijn and Ad Feelders. "Nearest Neighbour Classification with Monotonicity Constraints". In: *Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 2008, pp. 301–316.

[22] Ad Feelders. "Monotone Relabeling in Ordinal Classification". In: *2010 IEEE International Conference on Data Mining*. 2010.

[23] Jan Hauke and Tomasz Kossowski. "Comparison of values of Pearson's and Spearman's correlation coefficients on the same sets of data". In: *Quaestiones geographicae* 30.2 (2011), pp. 87–93.

[24] Karyn Holm and Norma J. Christman. "Post hoc tests following analysis of variance". In: *Research in Nursing & Health* 8.2 (1985), pp. 207–210. DOI: https://doi.org/10.1002/nur.4770080215. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nur.4770080215. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nur.4770080215.

[25] https://machinelearningmastery.com/feature-importance-and-feature-selection-with-xgboost-in-python/.

[26] https://ncri1.partners.org/ProACT/Document/DisplayLatest/3.

[27] https://towardsdatascience.com/how-does-the-popular-xgboost-and-lightgbm-algorithms-enforce-monotonic-constraint-cf8fce797acb.

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

[29] Jianchang Hu and Silke Szymczak. "A review on longitudinal data analysis with random forest". In: *Briefings in Bioinformatics* 24.2 (Jan. 2023), bbad002. ISSN: 1477-4054. DOI: 10.1093/bib/bbad002. eprint: https://academic.oup.com/bib/article-pdf/24/2/bbad002/49559948/bbad002.pdf. URL: https://doi.org/10.1093/bib/bbad002.

[30] Qinghua Hu et al. "Feature selection for monotonic classification". In: *IEEE Transactions on Fuzzy Systems* 20.1 (2012), pp. 69–81.

[31] Qinghua Hu et al. "Large-margin feature selection for monotonic classification". In: *Knowledge-Based Systems* 31 (2012), pp. 8–18.

[32] Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew. "Extreme learning machine: Theory and applications". In: *Neurocomputing* 70.1 (2006). Neural Networks, pp. 489–501.

[33] Lei Huang et al. "Longitudinal clinical score prediction in Alzheimer's disease with soft-split sparse regression based random forest". In: *Neurobiology of Aging* 46 (2016), pp. 180–191.

[34] Witten Ian et al. "The WEKA data mining software: An update". In: *SIGKDD Explorations* 11 (Nov. 2009), pp. 10–18. DOI: 10.1145/1656274.1656278.

[35] Rachida Ihya et al. "J48 algorithms of machine learning for predicting user's the acceptance of an E-orientation systems". In: Oct. 2019, pp. 1–8. ISBN: 978-1-4503-6289-4. DOI: 10.1145/3368756.3368995.

[36] Banks J. et al. "English Longitudinal Study of Ageing: Waves 0-9, 1998-2019". In: *37th Edition. UK Data Service. SN: 5050 DOI: 10.5255/UKDA-SN-5050-24*. 2019.

[37] Biao Jie et al. "Temporally Constrained Group Sparse Learning for Longitudinal Data Analysis in Alzheimer's Disease". In: *IEEE transactions on bio-medical engineering*. 2017.

[38] Angelika Kaiser. "A Review of Longitudinal Datasets on Ageing". In: *Journal of Population Ageing* 6 (June 2013). DOI: 10.1007/s12062-013-9082-3.

[39] Breiman L et al. *Classification and Regression Trees*. CRC Press, 1984.

[40] S. Lievens, B. De Baets, and K. Cao-Van. "A probabilistic framework for the design of instance-based supervised ranking algorithms in an ordinal setting". In: *Annals of Operations Research* (2008).

[41] B. De Baets M. Rademaker and H. De Meyer. "Optimal monotone relabelling of partially non-monotoneordinal data". In: *Optimization Methods Software Vol. 27, No. 1, February 2012, 17–31*. 2012.

[42] Jue Mo et al. "Classification of Alzheimer Diagnosis from ADNI Plasma Biomarker Data". In: *Proc. of the International Conf. on Bioinformatics, Computational Biology and Biomedical Informatics*. ACM, 2013, pp. 569–574.

[43] Géraldin Nanfack, Paul Temple, and Benoıt Frénay. "Constraint Enforcement on Decision Trees: A Survey". In: *ACM Comput. Surv.* 54.10s (Sept. 2022). ISSN: 0360-0300. DOI: 10.1145/3506734. URL: https://doi.org/10.1145/3506734.

[44] Mohammed Amine Chikh Nesma Settouti Mohammed El Amine Bechar. "Statistical Comparisons of the Top 10 Algorithms in Data Mining for Classification Task". In: *International Journal of Interactive Multimedia and Artificial Intelligence, Volume 4, pp. 46-51 DOI: 10.9781/ijimai.2016.419*. 2016.

[45] Uli Niemann et al. "Can We Classify the Participants of a Longitudinal Epidemiological Study from Their Previous Evolution?" In: *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*. 2015, pp. 121–126.

[46] Jarmo Teuho Riku Klén Oona Rainio. "Evaluation metrics and statistical tests for machine learning". In: *Scientific Reports volume 14, Article number: 6086 (2024) https://doi.org/10.1038/s41598-024-56706-x*. 2019.

[47] Sergey Ovchinnik, Fernando Otero, and Alex A. Freitas. "Nested Trees for Longitudinal Classification". In: *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*. SAC '22. Virtual Event: Association for Computing Machinery, 2022, pp. 441–444. ISBN: 9781450387132. DOI: 10.1145/3477314.3507240. URL: https://doi.org/10.1145/3477314.3507240.

[48] Sergey Ovchinnik, Fernando E. B. Otero, and Alex A. Freitas. "Monotonicity Detection and Enforcement in Longitudinal Classification". In: *LNCS, Artificial Intelligence XXXVI*. Springer International Publishing, 2019, pp. 63–77.

[49] R.S. Parpinelli, H.S. Lopes, and A.A. Freitas. "Data mining with an ant colony optimization algorithm". In: *IEEE Transactions on Evolutionary Computation* 6.4 (2002), pp. 321–332. DOI: 10.1109/TEVC.2002.802452.

[50] Wim Pijls and Rob Potharst. "Repairing non-monotone ordinal data sets by changing class labels". In: *Econometric Institute Report EI 2014-29*. 2014.

[51] Tossapol Pomsuwan and Alex A. Freitas. "Feature Selection for the Classification of Longitudinal Human Ageing Data". In: *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*. 2017, pp. 739–746.

[52] Rob Potharst and Jan C. Bioch. "Decision Trees for Ordinal Classification". In: *Intell. Data Anal.* 4.2 (2000), pp. 97–111.

[53] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

[54] Ye Ren, Le Zhang, and P.N. Suganthan. "Ensemble Classification and Regression-Recent Developments, Applications and Future Directions [Review Article]". In: *IEEE Computational Intelligence Magazine* 11.1 (2016), pp. 41–53. DOI: 10.1109/MCI.2015.2471235.

[55] Caio Ribeiro and Alex Freitas. "Constructed Temporal Features for Longitudinal Classification of Human Ageing Data". In: *2021 IEEE 9th International Conference on Healthcare Informatics (ICHI)*. 2021, pp. 106–112. DOI: 10.1109/ICHI52183.2021.00027.

[56] Caio Ribeiro and Alex A. Freitas. "A Mini-Survey of Supervised Machine Learning Approaches for Coping with Ageing-Related Longitudinal Datasets". In: *3rd Workshop on AI for Aging, Rehabilitation and Independent Assisted Living (ARIAL), IJCAI-2019*. 2019.

[57] M. Robnik-Sikonja and Igor Kononenko. "Theoretical and Empirical Analysis of ReliefF and RReliefF". In: *Machine Learning* 53 (2003), pp. 23–69.

[58] Michael R. Sheldon, Michael J. Fillyaw, and W. Douglas Thompson. "The use and interpretation of the Friedman test in the analysis of ordinal-scale data in repeated measures designs". In: *Physiotherapy Research International* 1.4 (1996), pp. 221–228. DOI: https://doi.org/10.1002/pri.66. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pri.66. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pri.66.

[59] Robert H. Shumway and David S. Stoffer. *Time Series Analysis and Its Applications: With R Examples, Fourth Edition*. Springer, 2017.

[60] Minhas Sidra et al. "Early Alzheimer's Disease Prediction in Machine Learning Setup: Empirical Analysis with Missing Value Computation". In: *Intelligent Data Engineering and Automated Learning – IDEAL 2015*. Springer International Publishing, 2015, pp. 424–432.

[61] Minhas Sidra et al. "Predicting Progression From Mild Cognitive Impairment to Alzheimer's Disease Using Autoregressive Modelling of Longitudinal and Multimodal Biomarkers". In: *IEEE Journal of Biomedical and Health Informatics* PP (May 2017), pp. 1–1. DOI: 10.1109/JBHI.2017.2703918.

[62] Wouter Verbeke, David Martens, and Bart Baesens. "RULEM: A novel heuristic rule learning approach for ordinal classification with monotonicity constraints". In: *Applied Soft Computing* 60 (2017), pp. 858–873.

[63] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2011.

[64] Jinsung Yoon, William R. Zame, and Mihaela van der Schaar. "ToPs: Ensemble Learning With Trees of Predictors". In: *IEEE Transactions on Signal Processing* 66.8 (2018), pp. 2141–2152. DOI: 10.1109/TSP.2018.2807402.

[65] Jinsung Yoon et al. "Personalized survival predictions via Trees of Predictors: An application to cardiac transplantation". In: *PLOS ONE* 13.3 (Mar. 2018), pp. 1–19. DOI: 10.1371/journal.pone.0194985. URL: https://doi.org/10.1371/journal.pone.0194985.

[66] Yuejin Zhang et al. "Study on Prediction of Activities of Daily Living of the Aged People Based on Longitudinal Data". In: *Procedia Computer Science* 9 (2016), pp. 470–477.

[67] Hong Zhu et al. "Monotonic classification extreme learning machine". In: *Neurocomputing* 225 (2017), pp. 205–213.