Winter 12-15-2019

Towards Interpretable Machine Learning with Applications to Clinical Decision Support

Zhicheng Cui
Washington University in St. Louis

WASHINGTON UNIVERSITY IN ST.LOUIS

School of Engineering & Applied Science
Department of Computer Science and Engineering

Dissertation Examination Committee:
Yixin Chen, Chair
Joanna Abraham
Roger Chamberlain
Brendan Juba
Thomas Kannampallil
William Yeoh

Towards Interpretable Machine Learning with Applications to Clinical Decision Support

by

Zhicheng Cui

A dissertation presented to
The Graduate School
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

December 2019
St. Louis, Missouri

Table of Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I would like to express my deep gratitude to my advisor, Prof. Yixin Chen, for his guidance and support throughout my PhD journey. Without him, none of the work described in this dissertation would have been possible. The countless times of research discussions, manuscripts refining, and career shaping have truly led me to successfully completing this doctoral degree. It was a really enjoyable experience to work with him over the past five years.

My sincere thanks also goes to other committee members: Joanna Abraham, Roger Chamberlain, Brendan Juba, Thomas Kannampallil and William Yeoh for their thoughtful comments and inspiring suggestions that helped my complete this dissertation.

I would like to thank Lu Wang, Bo-Yuan Ye and Weiber Tang of Google LLC who provided me a perfect internship experience and made me determined to continue my career in the industry.

I thank my talented lab mates, Wenlin Chen, Bradley Fritz, Yujie He, Christopher Ryan King, Haishuai Wang and Muhan Zhang for those inspiring discussions and wonderful collaborations. I would also like to thank my friends, Zihao Chen, Nan Cui, Tianben Ding, Zhiyang Huang, Shali Jiang, Chen Liu, Jiayi Song, Yijian Xiang, Hang Yan, Hao Yan, Zhen Zhang and Yao Zhou for their companion, making my PhD life full of joy.

Last but not the least, I would like to thank my parents Huairui Cui and Huanju Zhang, my elder brother Fei Cui for their unconditional love and continuous encouragement, making me brave and fearless.

Zhicheng Cui

Washington University in Saint Louis

December 2019

Dedicated to my parents.

ABSTRACT OF THE DISSERTATION

Towards Interpretable Machine Learning with Applications to Clinical Decision Support

by

Zhicheng Cui

Doctor of Philosophy in Computer Science

Washington University in St. Louis, 2019

Professor Yixin Chen, Chair

Machine learning models have achieved impressive predictive performance in various applications such as image classification and object recognition. However, understanding how machine learning models make decisions is essential when deploying those models in critical areas such as clinical prediction and market analysis, where prediction accuracy is not the only concern. For example, in the clinical prediction of ICU transfers, in addition to accurate predictions, doctors need to know the contributing factors that triggered the alert, which factors can be quickly altered to prevent the ICU transfer. While interpretable machine learning has been extensively studied for years, challenges remain as among all the advanced machine learning classifiers, few of them try to address both of those needs. In this dissertation, we point out the imperative properties of interpretable machine learning, especially for clinical decision support and explore three related directions. First, we propose a post-analysis method to extract actionable knowledge from random forest and additive tree models. Then, we equip the logistic regression model with nonlinear separability while preserving its interpretability. Last but not least, we propose an interpretable factored generalized additive model that allows feature interactions to further increase the prediction accuracy. In the end, we propose a deep learning framework for 30-day mortality prediction, that can handle heterogeneous data types.

Chapter 1

Introduction

Machine learning (ML) is unprecedented popular nowadays as it has achieved significant success in almost every area and widely deployed in real-world applications. In 2015, ML model for the first time beat human benchmark at image recognition task using complex deep learning techniques [63]. Despite their superior performance, we cannot fully trust them as the decision-making process is often obscure to us. We don't know why ML models make such prediction, when the models will fail and how to take advantage of the well-learned models for discovering new knowledge. The lack of interpretability hinders us from deploying them in domains that inaccurate result will cause unacceptable losses. An example is the Waymo's self-driving car that is capable of analyzing the surrounding environment and driving accordingly with the help of ML models. Although self-driving more than 10 million miles, those cars are still in the experimental stage because no one can guarantee the sophisticated ML models won't make any unforeseen mistake.

In this dissertation, we narrow our focus to supervised machine learning instead of unsupervised learning and reinforcement learning. For ease of representation, we still refer to

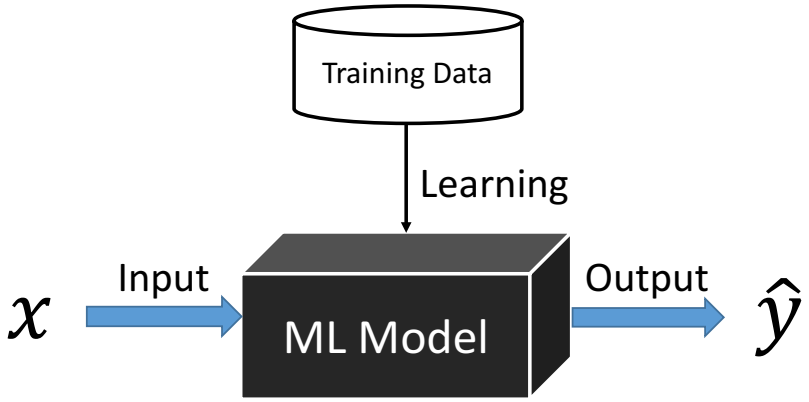

Figure 1.1: A black box ML model is learned from training data. When used for prediction, the ML model only provide the final output.

our scope of models as ML models. ML models are commonly treated as black-box models shown in Fig 1.1. When an ML model is trained based on a large amount of data, what the model has learned is obscure to us. Given an input, the ML model will merely provide the prediction result, without any additional explanation. Interpretable ML techniques try to open this black box and understand its internal mechanism. Before we dive into the details, we need to be clear about what is interpretability.

In the remaining of this chapter, we first introduce the concept of interpretability and summarize the key properties that we're interested in. Then we exhibit the taxonomy of interpretable machine learning methods. We also analyze the advantages and disadvantages of commonly used interpretable methods as well as ours and point out the motivation of this dissertation.

1.1 Definition of Interpretability

Interpretability can be defined as the degree to which an observer can understand the cause of a decision [98]. A human can consistently predict the model's output and explain its behavior in human understandable terms if the model has the highest interpretability. With that being said, the behavior of models with higher interpretability can be apprehended more easily. Nonetheless, there is no mathematical definition of interpretability [100], in other words how to measure the degree of interpretability is subjective. One might say that the rule-based decision tree is interpretable while it becomes infeasible to interpret when the depth is large. The authors [44, 90] also define interpretable in a high level and point out several essential proprieties that an interpretable ML model should have. We still have a long way to go in this direction.

The above-stated definition of interpretability is very vague. To make it concrete, we need to know why do we need interpretability first. Below are some demanding features that interpretable models can bring to us.

- **Model defect identification:** When deploying ML models to critical areas such as self-driving and clinical decision support, model stability is a must. ML model should consistently make reasonable predictions. However, images with only imperceptible changes can mislead state-of-the-art deep neural networks [83]. Due to model opacity, the flaw cannot be identified in advance. If the decision process works in an understandable manner, model defect issue can be addressed accordingly.
- **Model correctness validation:** There was a famous story about detecting the existence of camouflaged tanks using neural networks. The well trained model achieved fairly high accuracy in both train and test set by just picking an irrelevant feature

(cloudy or not). This happens when biased data is used. If we can examine the feature importance, data bias problem can be alleviated, ensuring a correct model.

- **Model fairness evaluation:** Model fairness refers to prediction integrity and avoiding kinds of discrimination. Developing for more than four years, Amazon's ML recruiting engine was reported to dislike resumes that included the word "women's". Amazon ultimately disbanded the team because of incompleteness on fixing this problem [39]. With the help of interpretable ML methods, engineers can discover the contribution of gender-related features and make the model neutral to particular terms.
- **Model knowledge extraction:** The ultimate goal of developing ML models is to gain new knowledge. In the era of technology, a massive amount of data in the scale of gigabyte or even terabyte are generated each second. Compressing this amount of data into knowledge is impractical for human beings while the ML models can. Therefore, extracting knowledge from ML models can be a surrogate method for people to learn things from the massive data.

In order to achieve the first three goals, understanding the decision process such as feature importance, feature interactions are essential. To realize the last goal, actionable knowledge should be obtainable from the ML models. Thus, we focus on the following two aspects in this proposal, 1) *accountability* (revealing the significance of each feature), 2) *actionability* (identifying changes to input features that can turn the model output to the desired label). Accountability can help us understand the decision process based on feature importance and actionability can help us extract new knowledge from the model. In areas such as clinical prediction and market analysis, these two properties are critically needed in addition to high accuracy. For example, in clinical prediction of ICU transfers, in addition to accurate

prediction, doctors need to know the contributing factors that triggered the alert (accountability), which factors can be quickly altered to prevent the ICU transfer (actionability). Unfortunately, most existing algorithms cannot balance these competing needs well.

Accountability Accountability has been studied for a long time. To reveal the feature importance, two different kind of algorithms have been proposed. The first category relies on analyzing the model directly. For example, the feature importance of logistic regression and density-based logistic regression[25] is indicated by its corresponding weight parameters. When growing trees, random forest can memorize the entropy gain for each feature and use this as an indicator of feature importance. Instead of identifying single feature importance, fast flux discriminant [27] first generates subsets of features and then use LR to identify the importance of subsets. The second category post-processes the well trained model when given an input instance. To check whether the prediction of deep neural networks is based on the right region of an image, guided back propagation [128] was proposed. Provided with an image and a well trained convolutional neural network (CNN), the authors run back propagation on the image space and highlight pixels by their corresponding absolute value of gradients as important features.

Actionability Actionable knowledge discovery has been studied mostly in the domain of business and marketing. Research on this subject is still very limited in the data mining and machine learning community. Some earlier works on actionable knowledge have focused on the development of effective interestingness metrics. Hilderman et al. proposed an two-step process for ranking the interestingness of discovered patterns [64]. Chi-square test is used in the first step and objective measures of interestingness is used in the second step. Cao et al. highlighted both technical interestingness and domain-specific expectations and proposed a two-way framework to measure knowledge actionablility [18]. They developed a ranking mechanism which balances the technical and business interests. Another line of work on

actionable knowledge discovery develops post-analysis techniques. Liu et al. tried to discover actionable knowledge by pruning and summarizing the learnt rules, as well as matching rules by similarity [91, 92]. Cao et al. proposed domain-driven data mining, a paradigm shift from a research-centered discipline to a practical tool for actionable knowledge [19, 20]. Ubiquitous intelligence must be involved and meta-synthesized into the mining process. They proposed several types of frameworks capable of handling different problems and applications. Techniques have also been proposed to post-process decision trees to extract actionable knowledge [70, 143]. For example, Yang et al. considers the problem of suggesting actions that maximize the expected profit based a decision tree model [143]. Note that there is no guarantee that the associations learned by machine learning models reflect the causal relationships. Actionable knowledge extracted from the machine learning models can be used to discover potential causal relationships. Domain experts or experiments are needed to verify those findings.

Now we illustrate the accountability, actionability, and non-linearity of general machine algorithms in table 1.1. Non-linearity is an important indicator of model capacity. As we can see, the decision tree has accountability, actionability, and non-linearity. However, rule-based decision tree algorithm is not stable enough to handle small change in the data and has relatively low performance compared with random forest. Thus, our first goal is to extract actionable knowledge from the random forest. Specifically, in Chapter 2, we transform action extraction from the random forest as an optimization problem and use optimization solver to solve it. K-nearest neighbors (KNN) doesn't have accountablity as we cannot measure how important each feature is. But we can understand its decision process by checking its k nearest neighbors. However, the evaluation phase of KNN is time consuming especially when the dataset is huge. Owing to the curse of dimensionality, KNN doesn't work well on high dimensional data. Support vector machine tries to find a hyper-plane

Table 1.1: The accountability, actionability and non-linearity of traditional machine learning methods

Algorithm	Accountability	Actionability	Non-linearity
Decision Tree	✓	✓	✓
Random Forest	✓	✗	✓
Support Vector Machine	✗	✗	✓
K-Nearest Neighbors	✗	✓	✓
Logistic Regression	✓	✓	✗
Deep Neural Network	✓	✗	✓

that maximizes the margin between the two classes. SVM itself can be interpreted easily by analyzing the linear hyper-plane. Yet when kernel methods are applied to SVM, the hyper-plane becomes nonlinear, making SVM less interpretable. Due to time limit, we don't study KNN and SVM in this dissertation. Logistic regression is one of the most widely used models because of its simplicity and interpretability. While at the same time, those two properties limit its classification accuracy. Deep neural networks (DNNs), instead, achieve state-of-the-art performance in many domains. However, the nonlinearity and complexity of DNNs make it less interpretable. To balance interpretability and classification performance, we combine LR model with DNNs, preserving the interpretability of LR while incorporating LR with non-linearity. Details can be found in chapter 3. Furthermore, we propose a factored generalized additive model in chapter 4 that allows feature interaction between static feature and time-varying features that are abundant in clinical dataset.

1.2 Taxonomy of Interpretable ML

Interpretable machine learning can be roughly divided into two categories, *designing interpretable models* (incorporating interpretability directly into the structure of the model) and *post-hoc interpretation* (devising interpretable methods).

Interpretable model means the model itself can be analyzed easily. Examples are logistic regression and decision tree model. Complex models can be interpretable through adding additional constraints. In [146], the authors combine traditional CNN with Deconvolutional Network [147] to visualize the input stimuli that excites individual feature maps at any layer in the model. By penalizing on the difference between pattern template and convolutional filters, [148] is able to visualize the semantically relevant region of the input image in an unsupervised way. Instead of modifying on the ML models, interpretable mimic learning distills knowledge from a complex teacher model to train the student model (i.e. simpler interpretable model such as generalized linear model) [7]. Attention mechanism that allows DNNs to focus on a subset of most discriminative features is also used to explain model prediction for a specific input sample [4, 30, 43, 75, 123]. However, there is no free lunch. Adding interpretable constraints or mimic learning hurts the model performance to some extent as accuracy is no longer the only optimization criteria.

In contrast, post-hoc interpretation methods, also known as model-agnostic methods, detach model interpretation from model training process. By doing so, interpretable methods can deliver interpretability without any performance loss. [114] assumes linear monotonicity of each feature and use generalized linear model to locally simulate the behavior of ML model. In [80], influence function, a technique from robust statistics is used trace the most influential data samples in the training set for a given prediction. In [3], the authors measure the feature importance by calculating the corresponding gradient. Although post-hoc interpretation methods do no harm to the model performance, the post-analysis approaches are not too general and are often attached with strong assumptions, deteriorating their flexibility and quality in achieving specific and accurate interpretation. Note that these two categories are not conflicting with each other. Post-hoc interpretable methods can also be applied to interpretable models to provide a different perspective of explanation.

Interpretable ML can also be divided into model level interpretability and instance level interpretability. Model level interpretability indicates that ML models provide general information about the decision mechanism by examining the model structure and parameters. LR reveals the importance of each input feature by the corresponding weight parameter. Instance level interpretability explains how a specific prediction is made given an input data point. It helps people understand the decision process case by case. Saliency detection method [1, 126, 128, 148] works in this way. Given an input image, the salience detection method can point out the relevant region leading to the final prediction.

Thees two different categorization methods are independent. By apply Cartesian product operation, we will have four different categories in total, instance level interpretable model [115, 138, 141], instance level post-hoc method [126, 128, 148], model level interpretable model [117, 137, 148] and model level post-hoc method [3, 104, 126]. Our optimal action extraction method described in chapter 2 is instance level post-hoc method. Our DELR model in chapter 3 is model level interpretable model and can be analyzed using instance level post-hoc method. FGAM is an instance level interpretable model that can extracts interpretability from generalized additive model in chapter 4.

1.3 Organization

In this dissertation, we propose three interpretable machine learning models, namely optimal action extraction method for random forests and boosted trees in Chapter 2, deep embedding logistic regression in Chapter 3 and factored generalized additive model in Chapter 4. We also propose an interpretable multi-scale convolutional neural network that can deal with time series that commonly appeared in clinical dataset in chapter 5 & 6. All our proposed

methods are applied to the real-world clinical datasets. Finally, Chapter 7 discuss the major findings and implications of our current study in interpretable machine learning area.

Chapter 2

Optimal Action Extraction for Random Forests and Boosted Trees

2.1 Background and Motivation

Additive tree models (ATMs) are ensemble models built on top of multiple decision/regression trees. A large scope of widespread models can be viewed as special cases of ATMs, such as random forest, adaboost with trees, and gradient boosting trees. In addition to their superior classification/regression performance, ATM enjoys many appealing properties that other machine learning models lack [49], including the support for multiclass classification, nonlinear classification capability, and natural handling of missing values and data of mixed type (categorical and numerical features). Often referred to as the best off-the-shelf classifiers [49], ATMs have been widely deployed into many industrial products such as Kinect [125] and face detection in camera [139], and are the must-try methods for lots of data mining competitions such as web search ranking [99].

Despite these advantages, ATMs often face a key hurdle in many practical applications: the lack of actionability. Given an input instance and a desired output, the actionability of a model is the ability to identify a set of changes to the input features that transforms the model prediction of this input to the desired output. Automatic extraction of actionable knowledge rather than resorting to domain experts is badly needed in many real-world applications such as clinical applications. For example, there has been an emerging trend of using machine learning models as the workhorse for the early warning systems to predict sudden deterioration (e.g., septic shock, stroke, respiratory arrest) for hospitalized patients based on their vital signs (such as blood pressure, heart rate, oxygen saturation and etc) [9, 96] with interpretability in mind [25, 27]. A model with actionability can be readily translated into intervening

We propose a novel approach to post-process any ATM classifier to extract an optimal actionable plan that can change a given input to a desired output with a minimum cost [35]. In particular, we model the problem as an integer linear programming (ILP) problem which is efficient to solve with state-of-the-art solvers such as CPLEX [32]. Any approximation method for the ILP problem will get the same approximation ratio for our problem of extracting actionable knowledge. This is akin to the spirit of other reduction methods in machine learning [26, 151] which reduce a new problem to another well-established problem to exploit its extensive research and development.

Contributions To the best of knowledge, our method is the first action extraction work on ATMs. Our main contributions are:

(1) We formally define the problem of optimally action extraction (OAE) based on a general formulation of ATMs that covers a broad spectrum of machine learning models.
(2) We prove the NP-hardness of the OAE problem on ATMs.

(3) We propose a novel method that translates the OAE problem into a closed-form integer linear programming (ILP) problem which can be efficiently solved by offthe-shelf ILP solvers. (4) We empirically demonstrate the effectiveness of our method on various datasets with excellent performance

2.2 Additive Tree Models

In this section, we first introduce a general formulation for *additive tree models* that our approach can be applied. Our formulation is very general and encompasses several most important and popular models for classification and regression, including random forests, adaboost, and gradient boosting trees. As a result, the proposed action extraction method has very wide applicability.

2.2.1 General model

An **additive tree model (ATM)** is an ensemble of T decision trees where T is the number of trees. Let $\mathbf{x} = (x_1, \cdots, x_D) \in \mathcal{D}_1 \times \cdots \mathcal{D}_D$ be a D dimensional feature vector, where each x_i, $i = 1, \cdots, D$, can be either categorical or numerical.

Each decision tree outputs a real value; let the output from tree t be $f_t(\mathbf{x})$. For both classification and regression, the output F of the additive tree model is a weighted sum of all the tree outputs as follows:

$$F(\mathbf{x}) = \sum_{t=1}^{T} w_t f_t(\mathbf{x}), \tag{2.1}$$

where $w_t \in \mathcal{R}$ is the weight of tree t. The formulation in (2.1) is very general and include some popular models as special cases. Next, we describe several additive tree models that are widely used in real-world applications.

2.2.2 Random forests

Random forest is arguably the most popular and powerful off-the-shelf classifier [16]. It can cope with regression and multi-class classification on both categorical and numerical datasets with superior accuracy. In essence, random forest is a bagging model [15] of trees where each tree is trained independently on a group of randomly sampled instances with randomly selected features.

Suppose we are given a dataset $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$ where N is the number of training instances, $\mathbf{x}^{(i)}$ and $y^{(i)}$ are feature vector and label of the i-th sample, respectively . The training of a random forest is as follows:

For $t = 1, \cdots, T$,

1. Sample n_{try} instances from the dataset with replacement.
2. Train an unpruned decision or regression tree f_t on the sampled instances with the following modification: at each node, choose the best split among m_{try} features randomly selected rather than among all features.

Both n_{try} and m_{try} and predefined constants. The final random forest model is the average of the outputs from all the trees, *i.e.* $F(\mathbf{x}) = \sum_{t=1} \frac{1}{T} f_t(\mathbf{x})$. For ease of presentation, we only consider binary classification for now. It is straightforward to extend our work to multi-class classification.

We can see that the random forest is a special case of (2.1) with $w_t = \frac{1}{T}$. To use random forests for classification, $f_t(\mathbf{x}) \in \{0, 1\}$ for all $t = 1, \cdots, T$, and $F(\mathbf{x})$ represents the probability of the label being 1. From the perspective of bias and variance decomposition, each tree in the random forest has low training bias but high variance since the tree is not pruned. The use

of bagging over those independent low-bias trees greatly reduces the variance of the resulting model, leading to superior generalization ability.

2.2.3 Boosted trees

Unlike a random forest that combines low-bias unpruned trees, the boosting method ensembles multiple weak learners to make a strong final model [47]. For boosting, each weak learner is typically a shallow tree with high bias and low variance.

The boosting method trains the additive model sequentially in a forward stage-wise manner. Suppose $F_t(\cdot)$ is the resulting model up to stage t. The model of the next stage is

$$F_{t+1}(\mathbf{x}) \leftarrow F_t(\mathbf{x}) + \alpha_t f_t(\mathbf{x}),$$

where $f_t(\cdot)$ is the weak learner obtained at stage t and α_t is the weight of this weak learner. The final model turns out to be a weighted sum of all trees:

$$F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}),$$

which is a special case of the general additive tree model in (2.1) with $w_t = \alpha_t$.

There are two common ways to train the weak learners, leading to two different models: adaboost and gradient boosting trees.

Adaboost When training a new weak learner, adaboost focuses more on instances that previous weak learners cannot correctly classify [47]. At stage t, the weight α_t and the tree

model $f_t(\cdot)$ are jointly optimized to minimize the loss function L of the resulting model F_{t+1}.

$$\underset{\alpha_t, f_t}{\text{minimize}} \quad \sum_{i=1}^{N} L(y^{(i)}, F_t(\mathbf{x}^{(i)}) + \alpha_t f_t(\mathbf{x}^{(i)})) \tag{2.2}$$

where L is a loss function measuring the difference between the true label $y^{(i)}$ and the model $F_{t+1}(\mathbf{x}^{(i)})$. In particular, when L is the exponential loss ($L(a, b) = e^{-ab}$), there is a nice closed-form solution for α_t, and f_t can be learned by training on the *weighted* instances.

Gradient boosting Gradient boosting is an even more general technique for boosting [48]. It counts each addition to the current model $F_t(\cdot)$ as a gradient update of (2.2) in the function space of $F_t(\cdot)$, where α_t is the learning rate and $f_t(\cdot)$ is the negative gradient of minimizing the loss function L. Specifically, $f_t(\cdot)$ is trained so that

$$f_t(\mathbf{x}^{(i)}) \approx -\frac{\partial L(y^{(i)}, F_t(\mathbf{x}^{(i)}))}{\partial F_t(\mathbf{x}^{(i)})}$$

which is equivalent to training a regression tree on the original instances with new labels defined by the negative gradient. The learning rate α_t can be set as a small constant or determined by a line search.

2.3 The Optimal Action Extraction Problem

In this section, we formally define the problem of optimally extracting actionable knowledge from additive tree models defined in (2.1), which as we show above is very general and encompasses random forests, adaboost, and gradient boosted trees as special cases.

Definition 1. The optimal action extraction (OAE) problem. *Given a feature vector* $\boldsymbol{x}^c$ *and an additive tree model in* (2.1) *with output function* F*, the problem is to find a feature*

vector $\boldsymbol{x}$ *such that* $F(\boldsymbol{x})$ *reaches a target value and the change from* $\boldsymbol{x}^c$ *to* $\boldsymbol{x}$ *incurs a minimum cost as measured by a cost function* $\ell(x^c, x)$*. Formally, it is:*

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} \quad \ell(\boldsymbol{x}^c, \boldsymbol{x}), \\ &\text{subject to} \quad F(\boldsymbol{x}) \geq z, \end{aligned} \tag{2.3}$$

where $z \in \mathcal{R}$ *is the target output.* $\ell(\boldsymbol{x}^c, \boldsymbol{x})$ *is the cost function measuring the cost of changing from* $\boldsymbol{x}^c$ *to* $\boldsymbol{x}$*. For example,* $\ell(\boldsymbol{x}^c, \boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{x}^c\|_0$ *evaluates the number of features changed.*

We now show that the OAE problem is in general a *NP-hard* problem. We prove it by reducing from the DNF-MAXSAT problem defined below.

Definition 2. The DNF-MAXSAT problem *is defined over a boolean formula in disjunctive normal form. Suppose a set of boolean variables* $\{v_1, v_2, \cdots, v_D\}$ *where* $v_i \in \{0, 1\}$*, and a set of* T *clauses where each clause* $\phi_i, i = 1, \cdots, T$ *has the form* $\phi_i = \wedge_j l_{ij}$*, where* l_{ij} *is the* j^{th} *literal in the* i^{th} *clause and is either a positive or a negative expression of a variable. If clause* i *is satisfied, we have* $\phi_i = 1$*; otherwise* $\phi_i = 0$*. The problem is to determine if there exists an assignment of all the boolean variables such that there are at least* m *clauses satisfied,* i.e. $\sum_i^T \phi_i \geq m$*.*

Theorem 1. *The OAE problem in* (2.3) *is NP-hard.*

Proof. It is well known that the DNF-MAXSAT problem is NP-hard [53]. We now reduce the DNF-MAXSAT problems to the OAE problem in (2.3).

Given any DNF-MAXSAT problems with T clauses, we construct an additive tree model (and in particular, a random forest) with D binary features and T binary trees where each tree represents a clause. To construct tree i, we iterate through all literals in clause ϕ_i and execute the following steps:

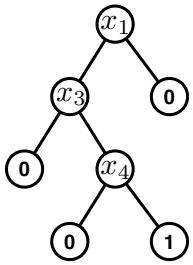

Figure 2.1: An illustration of constructing the tree for clause $v_1 \wedge \neg v_3 \wedge \neg v_4$.

1. Construct the root node and denote it as q. Let $j \leftarrow 1$.
2. Suppose l_{ij} is a literal involving boolean variable v_k, then node q is a decision node for feature k. If $l_{ij} = v_k$, construct two children for node q where the right child is a leaf node labeled as 0, and assign q's left child to q. If $l_{ij} = \neg v_k$, a similar operation will be applied with the left child being a leaf node and right child assigned to q.
3. Let $j \leftarrow j + 1$. If $j = n_i + 1$, make node q a leaf node with label 1; otherwise, go to Step 2.

An example of constructing the tree for clause $v_1 \wedge \neg v_3 \wedge \neg v_4$ is shown in Figure 2.1. Following the above procedure, we construct a random forest for regression. In particular, each tree only has one leaf node labeled as 1. The output of tree i is 1 if and only if all literals in clause i are made true. Let all $w_t = \frac{1}{T}$, the output of this constructed random forest is exactly $\frac{1}{T}\sum_{i=1}^{T} \phi_i$. Finally, the DNF-MAXSAT problem reduces to the OAE problem in (2.3) with $\ell(\mathbf{x}^c, \mathbf{x}) = 0$ and $z = m/T$. □

Given that the OAE problem is NP-hard, we do not expect to have any efficient algorithm for optimally solving it. Note that when there is only one decision tree, the OAE problem is in fact easy. We can enumerate all the paths leading up to leaves with desired outputs and choose one with the minimum cost. However, such enumeration does not work for ATMs since the outputs from different trees are inter-connected due to the features they share. Changing the value of a feature may impact the outputs of multiple trees and the final weighted-sum

output. Next, we present our solution which formulates the OAE problem as an integer linear programming problem and utilizes highly-optimized optimization solvers.

2.4 An Integer Linear Programming Approach

In this section, we describe our integer linear programming approach for solving the OAE problem (2.3). Specifically, we convert the constraint $F(\mathbf{x}) \geq z$ to a more approachable format so that efficient off-the-shelf optimization solvers can be directly applied to it.

The formulation consists of four parts: 1) tree output formulation, 2) feature value formulation, 3) decision logic formulation, and 4) objective function. We will discuss each part separately before putting them together.

2.4.1 Tree output

We note that (2.3) is not in closed form because $f_t(\mathbf{x})$ in (2.1), representing the output from a tree, is not in a mathematical formula. To address this problem, suppose m_t is the number of leaf nodes in the t^{th} tree.

Tree output variables. For each leaf node $k = 1, \cdots, m_t$, we use a binary variable $\phi_{t,k} \in \{0, 1\}$ to denote whether a given instance $\mathbf{x}$ reaches it.

Tree output constraints. Due to the property of the tree structure, each instance reaches exactly one leaf node. Therefore, we have,

$$\sum_{k=1}^{m_t} \phi_{t,k} = 1, \quad \text{for } t = 1, \cdots, T, \tag{2.4}$$

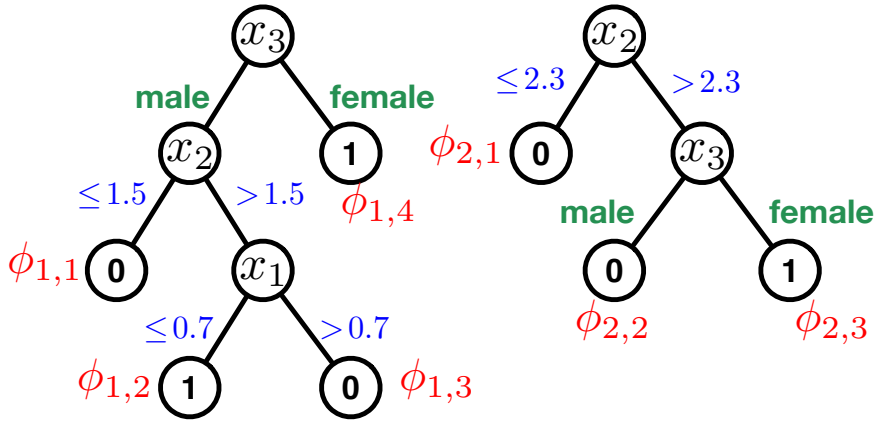

Figure 2.2: An illustration of random forest containing three features where the the third feature is categorical and the other two are numerical. The leaf nodes with label 1 are the target leaf nodes.

where T is the number of trees in the additive tree model. This way, $f_t(\mathbf{x})$ can be expressed as:

$$f_t(\mathbf{x}) = \sum_{k=1}^{m_t} h_{t,k}\phi_{t,k}. \tag{2.5}$$

where $h_{t,k} \in \mathcal{R}$ is the value of leaf node k in tree t.

Note that although (2.4) (2.5) and (2.1) form an optimization problem in (2.3), with $\phi_{t,k}$ and $\mathbf{x}$ being the variables, it is incorrect as it does not consider the correlation between $\phi_{t,k}$ among different trees. For example, Figure 2.2 shows a small random forest with two trees. In this case, it is impossible to have both $\phi_{1,1} = 1$ and $\phi_{2,3} = 1$. If $\phi_{1,1} = 1$, x_3 should be "male". If $\phi_{2,3} = 1$, x_3 should be "female" instead, which is conflicting. Therefore, we need to impose more constraints so that the solution respects the property of the additive tree model.

2.4.2 Feature value

Definition 3. ***(Feature partitions.)*** *Given an additive tree model, each feature x_i, $i = 1, \cdots, D$, is split into a number of partitions.*

- *If x_i is categorical with n categories, then x_i naturally has n partitions.*
- *If x_i is numerical, we assume each tree node branches in the form of $x_i \geq b$ where $b \in \mathcal{R}$ is a **splitting point**. If there are n splitting points for x_i in all the trees in the additive tree model, the feature x_i is naturally split into $n+1$ partitions.*

In the following, let n_i be the number of partitions for feature x_i.

Feature value variables. Given an instance $\mathbf{x}$, we use a binary variable $v_{ij} \in \{0, 1\}$ to denote whether x_i is in the j^{th} partition of dimension i. $v_{i,j} = 1$ if and only if x_i is in the j^{th} partition.

Feature value constraints. Since an instance could only reside in exactly one partition for each feature, we know that v_{ij} satisfies the following property:

$$\sum_{j=1}^{n_i} v_{ij} = 1 \tag{2.6}$$

Example 1. As shown in Figure 2.2, the feature x_2 occurs twice in the non-leaf nodes of the random forest. The split points are 1.5 and 2.3, respectively, leading to three partitions for this feature, *i.e.* $(-\infty, 1.5]$, $(1.5, 2.3]$ and $(2.3, +\infty)$. Given an instance $\mathbf{x} = (x_1, \cdots, x_D)$, if $1.5 < x_2 \leq 2.3$ (say $x_2 = 1.9$), then $v_{2,2} = 1$ and $v_{2,1} = v_{2,3} = 0$. $\square$

2.4.3 Decision logic

Now we need to link $v_{i,j}$ with $\phi_{t,k}$. Given a leaf node k in tree t, suppose $\pi_{t,k}$ is the set of all its ancestor nodes in the tree. For any node $p \in \pi_{t,k}$, suppose p branches on feature i, we define $S_{k,p}$ to be the set containing all predicates $v_{i,j}$ satisfying that $v_{ij} = 1$ leads to the branch towards leaf node k.

Example 2. Continuing with Figure 2.2, the second leaf node of the first tree (marked with $\phi_{1,2}$) has three ancestors, *i.e.* $\pi_{1,2} = \{x_3, x_2, x_1\}$[1]. For node x_2, the corresponding $S_{2,x_2} = \{v_{2,2}, v_{2,3}\}$, since both $v_{2,2} = 1$ (*i.e.* $x_2 \in (1.5, 2.3]$) and $v_{2,3} = 1$ (*i.e.* $x_2 \in (2.3, +\infty)$) are in the branch where the second leaf node lies. Likewise, $S_{2,x_3} = \{v_{3,1}\}$ where $v_{3,1} = 1$ stands for $x_3 =$ "male". $\square$

Making use of the tree structure, we have the following properties:

- If $\phi_{t,k} = 1$, meaning that the instance $\mathbf{x}$ lies in the leaf node k in tree t, then there is always one of the predicates v_{ij} in $S_{k,p}$ being 1 for any node $p \in \pi_{t,k}$.
- If $\phi_{t,k} = 0$, then there exists at least one node p such that all the predicates in $S_{k,p}$ are 0. For example, to have $\phi_{1,2} = 1$, either $v_{2,2}$ or $v_{2,3}$ should be 1. To have $\phi_{1,2} = 0$, there should be at least one node that does not branch towards the leaf node k.

Putting the above properties in a mathematical form, we have that

$$\begin{aligned} \phi_{t,k} = 1 &\Longleftrightarrow \sum_{v \in S_{k,p}} v = 1, \quad \forall p \in \pi_{t,k} \\ \phi_{t,k} = 0 &\Longleftrightarrow \sum_{v \in S_{k,p}} v = 0, \quad \exists p \in \pi_{t,k} \end{aligned} \tag{2.7}$$

(2.7) is still not in a closed form and we need to further reduce it. To simplify (2.7), we first prove the following results.

Lemma 1. *Given that* $0 \leq u_i \leq 1$, $\frac{1}{U}\sum_{i=1}^{U} u_i = 1$ *implies* $u_i = 1$ *for* $i = 1, \cdots, U$.

[1]With a slight abuse of notations, we now use x_i to represent the node for simplicity of presentation, because each feature only occurs once in the first tree of Figure 2.2.

Theorem 2. *If* (2.6) *holds, we have that*

$$\frac{1}{|\pi_{t,k}|}\sum_{p\in\pi_{t,k}}\sum_{v\in S_{k,p}} v \leq 1 \tag{2.8}$$

where $|\pi_{t,k}|$ *is the cardinality of* $\pi_{t,k}$, i.e. *the number of ancestor nodes of the leaf node* k. *In addition, there is only one leaf node satisfying* (2.8) *with equality.*

Proof. First, we prove (2.8) holds. Suppose that node $p \in \pi_{t,k}$ branches at feature $r(p)$. Combined with (2.6), we have that

$$\sum_{v\in S_{k,p}} v \leq \sum_{j=1}^{n_{r(p)}} v_{r(p),j} = 1, \quad \forall p \in \pi_{t,k} \tag{2.9}$$

which leads to (2.8).

Next, we prove that only one leaf node satisfies (2.8) with equality. We prove this by contradiction. Suppose there are two leaf nodes satisfying (2.8) with equality. Let node a and b be these two leaf nodes. Hence, we have that

$$\frac{1}{|\pi_{t,k}|}\sum_{p\in\pi_{t,k}}\sum_{v\in S_{k,p}} v = 1, \quad \text{for } k = a, b.$$

According to (2.9) and Lemma 1, we have that $\sum_{v\in S_{a,p}} v = 1$ and $\sum_{v\in S_{b,p}} v = 1$ for any node p in π_a and π_b.

Let node q be the lowest common ancestor of a and b and it branches at feature i, *i.e.* $r(a) = r(b) = i$. We know that $S_{a,q} \cap S_{b,q} = \emptyset$, since they belong to different branches of q. Hence, we have:

$$\sum_{i=1}^{n_i} v_{i,j} \geq \sum_{v\in S_{a,q}\cup S_{b,q}} v = \sum_{v\in S_{a,q}} v + \sum_{v\in S_{b,q}} v = 2, \tag{2.10}$$

which contradicts (2.6). □

According to (2.8), we can simplify (2.7) into the following equivalent equation:

$$\phi_{t,k} = \left\lfloor \frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v \right\rfloor, \forall t, k \tag{2.11}$$

Eq. (2.11) implies the following three properties:

1. From Theorem 2, we know that the term within the floor operation is equal to or less than 1. Therefore, $\phi_{t,k} \leq 1$, $\forall t, k$.
2. From Lemma 1, we also have that $\phi_{t,k}$ in (2.11) is 1 if and only if $\sum_{v \in S_{k,p}} v = 1$, $\forall p \in \pi_{t,k}$.
3. From Theorem 2, we have that there is only one leaf node k in tree t satisfying that $\phi_{t,k} = 1$.

The presence of the floor operator poses great difficulty to optimization. To address it, we replace the equality constraint by an inequality constraint so that the floor operator can be removed. This gives rise to the following **decision logic constraints**:

$$\phi_{t,k} \leq \frac{1}{|\pi_{t,k}|} \sum_{p \in \pi_{t,k}} \sum_{v \in S_{k,p}} v, \forall t, k \tag{2.12}$$

Next, we show that (2.12) is equivalent to (2.11) when the tree output constraints in (2.4) and feature value constraints in (2.6) are all satisfied.

Theorem 3. *When* (2.4) *and* (2.6) *hold,* (2.12) *is equivalent to* (2.11).

Proof. It is easy to show that (2.11) implies (2.12) due to the property of the floor operator. Now we show that (2.12) implies (2.11). There are two cases:

a) If

$$\frac{1}{|\pi_{t,k}|}\sum_{p\in\pi_{t,k}}\sum_{v\in S_{k,p}} v < 1,$$

from (2.12) we have that $\phi_{t,k} < 1$. Since $\phi_{t,k} \in \{0, 1\}$, we must have $\phi_{t,k} = 0$.

b) If

$$\frac{1}{|\pi_{t,k}|}\sum_{p\in\pi_{t,k}}\sum_{v\in S_{k,p}} v = 1,$$

according to Theorem 2, there is only one leaf node k' in tree t that satisfies this property. Since all $\phi_{t,k} \leq 1$ and only $\phi_{t,k'}$ is possible to be 1, if $\phi_{t,k'} = 0$, we have $\sum_{k=1}^{m_t} \phi_{t,k} = 0$, which contradicts (2.4). Therefore, $\phi_{t,k'} = 1$.

In both cases, (2.12) implies (2.11). □

Based on the above analysis, we have shown that (2.12) correctly models the decision logic of additive trees.

2.4.4 Objective function

Now we consider the objective function $\ell(\mathbf{x}^c, \mathbf{x})$ in (2.3). The objective plays a key role for users to specify their preferences over which kind of changes to make. For business operations, for example, each change may involve certain investment or operational cost and we want to achieve our goal with the minimum expense. For clinical prediction, the doctors may want the algorithm to suggest as few vital signs as possible, in order to understand and identify the key factor for intervention.

Let $\mathbf{v}$ be the vector containing all feature value variables $v_{i,j}$. Since $\mathbf{v}$ represents the partition each dimension of $\mathbf{x}$ should fall in, we see that all $\mathbf{x}$'s that correspond to the same $\mathbf{v}$ have the same outputs from the trees and the ATM. Therefore, we see that we can replace the

cost function $\ell(\mathbf{x}^c, \mathbf{x})$ in (2.3) by $\ell(\mathbf{x}^c, \mathbf{v})$, which models the minimum cost of moving a given sample $\mathbf{x}^c$ to the partitions specified in $\mathbf{v}$.

A typical cost function is defined over a **cost matrix** $\mathbf{C}(\mathbf{x}^c)$, in which $C_{i,j}(\mathbf{x}^c) \in \mathcal{R}$ is the cost of moving the i^{th} feature of $\mathbf{x}^c$ to the partition $v_{i,j}$. The objective function can be expressed as follows.

$$\ell(\mathbf{x}^c, \mathbf{v}) = \sum_{i=1}^{D} \sum_{j=1}^{n_i} v_{i,j} C_{i,j}(\mathbf{x}^c) \tag{2.13}$$

which is a linear function of $v_{i,j}$. In principle, $C_{i,j}(\mathbf{x}^c)$ is user-defined, offering total flexibility in modeling the cost of changes in practice. For example, if a feature i cannot be easily changed (such as gender or age), we can assign $C_{i,j}(\mathbf{x}^c) = \infty$ for all $j \neq i$.

For each numerical feature i, we can define a general form for $C_{i,j}(\mathbf{x}^c)$. Let $b_{i,j}$ and $e_{i,j}$ be the beginning and end values of partition $v_{i,j}$, respectively. For example, if $v_{i,j} = (1, 5, 2.3]$, $b_{i,j} = 1.5$ and $e_{i,j} = 2.3$. A general form of the cost matrix is as the following.

$$C_{i,j}(\mathbf{x}^c) = \begin{cases} 0, & x_i^c \text{ is in } v_{i,j} \\ \min\{(x_i^c - b_{i,j})^p, (x_i^c - e_{i,j})^p\}, & \text{otherwise} \end{cases} \tag{2.14}$$

where x_i^c is the i^{th} feature value of $\mathbf{x}^c$, and $p \in \mathcal{R}$ is a constant.

With the cost matrix in (2.14), our cost function in (2.13) is general enough to accommodate most common needs in practice. In fact, the above cost function represents an ℓ_p **norm**. For example, when $p = 0$, the cost matrix defines the ℓ_0 norm:

$$C_{i,j}(\mathbf{x}^c) = \begin{cases} 0, & x_i^c \text{ is in } v_{i,j} \\ 1, & \text{otherwise} \end{cases} \tag{2.15}$$

which makes our OAE problem minimize the number of changed features, *i.e. Hamming distance*. When $p = 1$, it defines an ℓ_1 norm and the OAE problem minimizes the *Manhattan distance*. When $p = 2$, it defines an ℓ_2 norm and the OAE problem minimizes the *Euclidean distance*.

2.4.5 Overall optimization formulation

Combining all the four parts above, we now can fully express the OAE problem in (2.3) in a closed-form optimization problem by integrating the objective in (2.13) and constraints in (2.1), (2.4), (2.5), (2.6), and (2.12). The overall problem is:

$$\begin{aligned}
\underset{v_{i,j},\phi_{t,k}\in\{0,1\}}{\text{minimize}} \quad & \sum_{i=1}^{D}\sum_{j=1}^{n_i} v_{i,j}C_{i,j}(\mathbf{x}^c) \\
\text{subject to:} \quad & \sum_{t=1}^{T} w_t \sum_{k=1}^{m_t} h_{t,k}\phi_{t,k} \geq z \\
& \sum_{j=1}^{n_i} v_{ij} = 1, \quad i = 1, \cdots, D \\
& \phi_{t,k} \leq \frac{1}{|\pi_{t,k}|} \sum_{p\in\pi_{t,k}} \sum_{v\in S_{k,p}} v, \quad \forall t, k \\
& \sum_{k=1}^{m_t} \phi_{t,k} = 1, \quad t = 1, \cdots, T
\end{aligned} \tag{2.16}$$

Since all the objective and constraints are linear functions over the binary variables $\mathbf{v}$ and ϕ, (2.16) is an **integer linear programming (ILP)** problem. ILPs have been extensively studied and can be efficiently solved by powerful off-the-shelf solvers such as IBM ILOG CPLEX. Note that CPLEX supports a distributed parallel algorithm that can naturally leverage parallel and multi-core computers.

Table 2.1: The number of instances (N), number of features (D), and number of classes (C) of all testing datasets

Dataset	N	D	C
heart	270	13	2
liver disorders	345	6	2
breast cancer	683	10	2
australian	690	14	2
ionosphere	351	34	2
a1a	1000	123	2
mushrooms	8124	112	2
dna	3386	180	3
glass	214	9	6
vowel	990	10	11

2.5 Experimental Results

In this section, we conduct extensive experiments on several benchmark datasets to evaluate the proposed methods on the OAE problem. For ease of presentation, we refer to our method as the ILP method.

2.5.1 Baseline methods

For comparison, we consider the following baseline methods for solving the OAE problem. These baseline methods are reasonable and in some cases highly competitive. None of them can guarantee optimality as our ILP method does. We do not report results on some exhaustive search methods that can guarantee optimality since they are prohibitively expensive and can only solve tiny cases.

1. **Iterative testing.** This method iterates through all training instances available, denoted as $\mathbf{x}^{(i)}$, and compute the prediction $F(\mathbf{x}^{(i)})$ as well as the cost $\ell(\mathbf{x}^c, \mathbf{x}^{(i)}$. The final solution has the minimum cost among all instances with the desired output, as

follows:

$$\min_i \ell(\mathbf{x}^c, \mathbf{x}^{(i)}), \quad s.t.: \ F(\mathbf{x}^{(i)}) \geq z$$

2. **Greedy algorithm.** Starting from $\mathbf{x}^c$, this method greedily changes the feature that maximizes $F(\mathbf{x})$. Mathematically, it starts from $\mathbf{x}^0 = \mathbf{x}^c$, and solves the following problem in the i^{th} iteration:

$$\max_{\mathbf{x}^i} F(\mathbf{x}^i) - F(\mathbf{x}^{i-1}), \quad s.t.: \ \|\mathbf{x}^i - \mathbf{x}^{i-1}\|_0 = 1. \tag{2.17}$$

   To solve (2.17) and find $\mathbf{x}^i$, the greedy algorithm first pick a feature, denoted as d, and fixes all other features of $\mathbf{x}^{i-1}$. Then it iterates through all partitions of feature d and finds one that maximizes the marginal utility $F(\mathbf{x}^i) - F(\mathbf{x}^{i-1})$. Namely, each iteration changes one feature to a value that maximizes the gain. The greedy algorithms keeps iterating until $F(\mathbf{x}) \geq z$ is met or there is no more features to change. The goal of this method is simply finding a feasible solution without considering the cost.

3. **Cost-aware greedy algorithm.** This method is similar to the greedy algorithm except that it takes the cost into account by changing the utility to a cost-aware utility. In particular, (2.17) is changed to the following and the rest of the greedy process remains the same:

$$\max_{\mathbf{x}} \frac{F(\mathbf{x}^i) - F(\mathbf{x}^{i-1})}{\ell(\mathbf{x}^i, \mathbf{x}^{i-1})}, \quad s.t. \ \|\mathbf{x}^i - \mathbf{x}^{i-1}\|_0 = 1 \tag{2.18}$$

   From (2.18), we can see that with this method a feature change with a high cost $\ell(\mathbf{x}^c, \mathbf{x})$ is less likely to be chosen.

2.5.2 Experimental setup

We test all the methods on nine benchmark datasets from the UCI repository[2] and the LibSVM website[3]. The information of datasets are listed in Table 2.1. For compactness, we only show the experimental results on random forest for classification since we found the basic trend is consistent among all ATMs. We split each dataset into training and testing sets. The random forest is built on the training set. Implementation-wise, we adopt the R package implemented by Liaw et al., which is the current state-of-the-art random forest package [87]. For the OAE problem, we choose a weighted Euclidean distance as the loss function,

$$\ell(\mathbf{x}^c, \mathbf{x}) = \sum_{i=1}^{D} \beta_i (x_i^c - x_i)^2 \tag{2.19}$$

where β_i is the cost weight on feature i. For each dataset, we randomly generate ten sets of feature cost β_i in the range between 1 and 100. For each set of cost, we sample ten testing instances at random and solve the OAE problem with desired outputs being the other classes.

In summary, for each dataset, for each method, we solve the OAE problem under $100(C-1)$ number of different settings where C is the number of unique labels in a dataset. The reported results are the average of all tested settings. All experiments are run on a desktop computer with 2.5GHz CPU and 16G memory.

2.5.3 Comprehensive results

Table 2.2 shows a comprehensive comparison in terms of the running time and the solution quality measured by the cost $\ell(\mathbf{x}^c, \mathbf{x})$. The reported results are the average performance of

[2]https://archive.ics.uci.edu/ml/datasets.html
[3]http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

Dataset	Iterative testing		Greedy		Cost-aware Greedy		ILP method	
	Time(sec)	Cost	Time(sec)	Cost	Time(sec)	Cost	Time(sec)	Cost
a1a	0.12	116.46	3.38	39.48	4.75	32.74	7.35	**9.24**
liver disorders	0.00	11.64	2.21	8.21	2.57	4.02	31.22	**1.10**
australian	0.00	41.76	2.99	12.30	3.13	6.51	108.31	**3.42**
breast cancer	0.00	74.19	0.16	41.69	0.22	45.57	31.80	**18.35**
dna	0.08	1112.22	4.23	34.45	5.00	34.47	9.84	**24.10**
glass	0.00	16.40	1.38	21.63	1.36	11.82	57.67	**1.18**
heart	0.00	53.30	1.12	32.52	1.10	22.67	5.71	**2.51**
ionosphere	0.01	195.67	15.01	53.53	21.20	40.87	48.89	**6.98**
mushrooms	0.27	199.68	0.91	55.51	1.26	68.69	3.49	**32.53**
vowel	0.02	34.44	7.65	22.55	7.57	10.86	68.72	**1.24**

Table 2.2: Solution time and action costs of various methods. All results are averaged over 100(C-1) runs. Optimal costs are marked in bold. See text for details of experimental setup.

different settings as stated before. We omit the settings where either the greedy algorithm or cost-aware greedy algorithm fails to find a feasible solution, since it results in an infinite loss.

From Table 2.2, we make the following observations: 1) Our ILP method always attains the minimum cost and largely outperforms all other methods in terms of quality, which is under our expectation since the ILP method is guaranteed to find the optimal solution. As a result, the ILP method also takes more time than all other methods. 2) The cost-aware greedy algorithm outperforms the greedy algorithm in most cases in terms of solution quality. This is because the greedy algorithm does not take costs into consideration when searching for a feasible solution. In terms of running time, these two methods are comparable. 3) Though the iterative testing algorithm is the most efficient among all tested methods, it has the worst performance in terms of the solution quality.

We show a scatter plot of the performance of all methods under all settings in Figure 2.3. The x-axis stands for relative cost, which is the ratio of the cost obtained by each method over the optimal cost (found by the ILP method). We can see that all results by the ILP method lies in the spike whose relative cost is 1. For these baseline methods, although they are in general faster, their solution quality can be extremely (up to 10^3 times) poor, making their solution

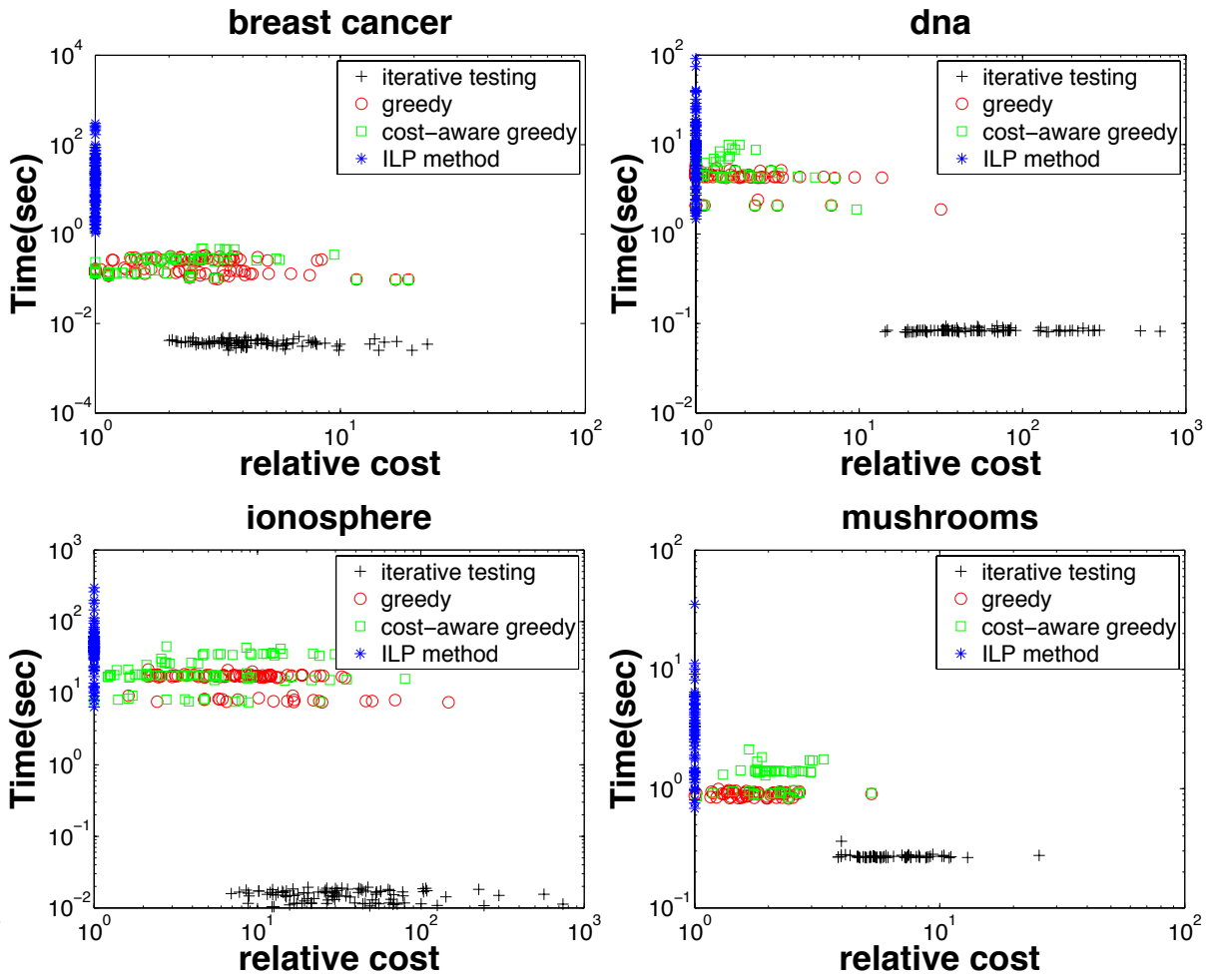

Figure 2.3: Results of running time and relative costs of all the methods under different settings.

not useful. As we have discussed before, in many cases, extracting the optimal actions is a means for individualized feature selection. Non-optimal methods with poor quality are not useful in this case.

2.6 Discussion and Conclusion

We proposed an integer linear programming (ILP) formulation to model the OAE problem and proved its correctness to support actionability for additive tree models (ATMs). Extensive

experimental results showed that the proposed method is optimal, efficient, and reliable. It significantly outperforms other baseline methods in terms of the solution quality.

Solving the OAE problem not only provides an action plan, but also helps identify individualized feature importance for each instance. Most feature selection algorithms are based on a given training dataset and classifier, but are not customized for an individual instance. In contrast, the output from OAE can be viewed as a result of individualized feature selection, as it identifies the few features that can most efficiently change the prediction output of a particular instance. Such individualized feature selection may find many applications, such as personalized health care and targeted marketing.

Chapter 3

Deep Embedding Logistic Regression

3.1 Background and Motivation

LR is widely used, especially on high-dimensional cases, due to its simplicity and efficiency. Moreover, LR offers accountability and actionability. Weights in LR can measure feature importance and tell how we can alter certain features with a minimum cost to achieve a desired output. However, being a linear classifier, LR has limited separation ability and generally low accuracy. In addition, categorical features are not naturally supported by LR. DNNs keep breaking records in applications such as image classification [66], speech recognition [21] and objection detection [113]. However, the prediction results of DNNs are known to be very hard to interpret due to the extreme nonlinearity and complexity of the model [2]. The lack of interpretability greatly restricts their prevalence in fields such as clinical predictions [22] and business intelligence [23] where explanation, insights, and actionability are much needed. In clinical applications, interpretability helps doctors analyze the status of patients and take actions to intervene. Thus, prediction accuracy is often sacrificed by using

more traditional but more interpretable algorithms such as logistic regression and decision trees.

Thus, we proposed an end-to-end logistic regression model, deep embedding logistic regression (DELR), which incorporates LR with deep learning based feature embedding [37]. By taking the advantage of DNNs' superior expressing power, each feature is first transformed into non-linear representation before being fed into a LR layer. DELR has high efficiency by leveraging GPU computing. Nonlinear feature transformation equips DELR with nonlinear separation ability. Using deep embedding, we can also naturally handle and quantify categorical features, which is not supported by LR. Last but not least, accountability and actionability are offered by the LR layer.

Contributions In summary, our contributions are as follows.

1) We propose DELR, a classifier that offers scalability, nonlinearity, support for mixed data types and most importantly, excellent interpretability.

2) We analyze and demonstrate the model accountability and actionability of DELR through case studies.

3) We empirically validate the accuracy performance of DELR as compared with existing interpretable methods including LR, DT, DLR and gradient boosting decision stumps (GBDS). Commonly used non-interpretable models such as SVM-rbf, random forest (RF) are also tested for reference.

4) We visualize the quantification of categorical feature embedding and further verify the interpretability of DELR.

5) We apply DELR on a real-world clinical dataset and show how interpretability can help doctors make decisions.

3.2 Related Work

DELR is designed to preserve the interpretability with minimum sacrifice on the prediction performance. For DELR, we can not only show the feature importance using weight parameter, but also visualize the trending of each feature as it changes.

Among all classifiers, DELR resembles to gradient boosted decision stumps (GBDS) [62] most in terms of individual feature transformation. Different from gradient boosted decision trees (DBDT) [24], GBDS is an ensemble of one level decision trees. For each decision stump, only a single feature is used for classification. After training the GBDS, we can group decision stumps with same features together and draw plots similar to Fig. 3.3. However, the coordinate plot of DELR is much smoother than GBDS, leading to better generalizability. We will compare the model performance in the experimental section.

Compared with accountability, extracting knowledge from machine learning model is an even harder task. Rule based algorithms can be post-analyzed through pruning and summarization [93, 94]. In [144], the authors use a greedy algorithm to provide actions that can maximize the expected profit from the decision tree. Despite their actionability, decision trees cannot achieve high accuracy. Reference [35] further proposed an integer linear programming (ILP) algorithm to extract optimal actionable knowledge from random forests. However, ILP is a NP-Complete problem, restricting its usage in large scale datasets. For deep neural networks, meaningfully actionable knowledge is hard to extracted due to model complexity [55]. DELR inherits all the advantages of LR, especially efficiency, accountability, and actionability, making it capable of extracting knowledge from various kinds of datasets.

3.3 Preliminaries

In this section, we introduce the notations and briefly review the limitations of LR and its extensions.

Suppose we are given a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N$ with N instances, where $\mathbf{x}_i$ and y_i are D dimensional feature vector and label of instance i, respectively. Each feature vector is the concatenation of two types of feature vectors, numerical feature vector $\mathbf{x}_i^R \in \mathcal{R}^{D_1}$ and categorical feature vector $\mathbf{x}_i^{\mathcal{C}} = ([x_i^C]_1, [x_i^C]_2, ..., [x_i^C]_{D_2})$. Each element in $\mathbf{x}_i^R$ is a real number and each element in $\mathbf{x}_i^{\mathcal{C}}$ is an ordinal number. DELR can handle both binary and multi-class classification. For ease of presentation, we consider binary classification where $y_i \in \mathcal{C} = \{0, 1\}$. $\mathcal{D}_k$ contains all the data samples with label k. Multi-class classification can be easily supported by replacing LR with softmax regression classifier.

A common way for many classifiers to handle categorical features is one-hot encoding, which converts a categorical feature to a numerical vector. However, one-hot encoding is known to be prohibitively expensive when the cardinality is high.

LR models the conditional probability of y given an instance $\mathbf{x}$ using a sigmoid function:

$$p(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\mathbf{T}\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\mathbf{T}\mathbf{x})} \tag{3.1}$$

where $\mathbf{w}$ is weight parameters to be learned, making the decision boundary a hyperplane. The confidence score is controlled by the weighted sum of input features. LR assumes that there is a monotonic relationship between $p(y = 1|\mathbf{x})$ and x_d, while in practice often does not exist. DLR was proposed to fix this problem by embedding each feature x_d into a nonlinear

representation:

$$\Phi(\mathbf{x}) = (\phi_1(\mathbf{x}), ..., \phi_D(\mathbf{x})).$$

By assuming all the attributes of $\mathbf{x}$ are conditionally independent given the label y, each attribute was represented using logit transformation:

$$\phi_d(\mathbf{x}) = \ln \frac{p(y=1|x_d)}{p(y=0|x_d)}. \tag{3.2}$$

3.4 Deep Embedding Logistic Regression

In this section, we propose a deep embedding logistic regression (DELR) framework, in which, no prior assumption for data distribution is needed. Instead of using kernel based estimator, we use deep neural networks for dimension-wise feature embedding. The overall architecture of DELR is depicted in Fig. 3.1. DELR contains three different blocks: numerical feature embedding block, categorical feature embedding block and logistic/softmax regression block.

3.4.1 Numerical Feature Embedding Block

For the numerical part, each feature is not necessarily in a monotonic relationship with class probability. To address this problem, we apply a multi-layer perceptron (MLP) for the numerical feature embedding as MLP has the ability to learn complex feature representation automatically. In order to reduce the total number of parameters to be learned, we design a deep and narrow MLP as shown in Fig. 3.1. Although the feature embedding block has 6 layers, each of which contains only 4 hidden neurons. Thus, the total number of parameters is only 80. Compared with classical MLP that has millions of parameters, our model greatly reduced the learning time in updating each parameters, being able to handle very high

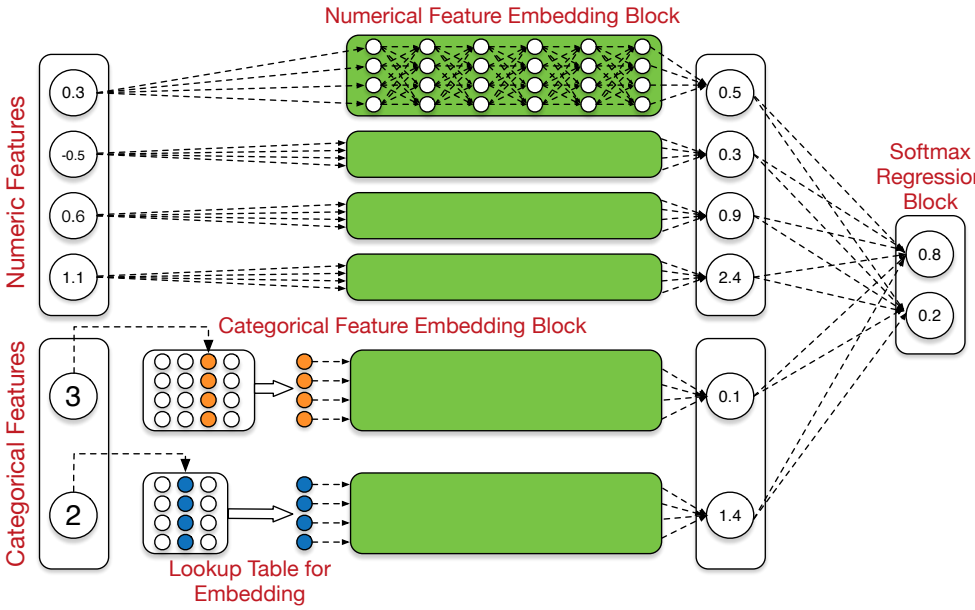

Figure 3.1: Overall architecture of DELR

dimensional dataset. Rectified linear unit $A(x) = \max(x, 0)$ is used as activation function between two adjacent layers. Batch normalization [68] is also applied before every activation function. The input and output dimension of numerical feature embedding block are one. After the conversion, we have

$$f_d(\mathbf{x}) = A(b_d^{nh} + W_d^{nh} A(\cdots A(b_d^1 + W_d^1 x_d))) \tag{3.3}$$

where nh is a user defined number of hidden layers.

3.4.2 Categorical Feature Embedding Block

Suppose the categorical part $\mathbf{x}^C$ has D_2 features, $\mathbf{x}^C = (x_1^C, ..., x_{D_2}^C)$. The d^{th} categorical feature has K_d categories, $x_d^C \in \{1, 2, ..., K_d\}$.

The first component of the categorical feature embedding block is a lookup table that contains a numerical embedding for each category as shown in Fig. 3.1. The number of embeddings of each feature is equal to the number of categories of the corresponding categorical feature. At its core, each lookup table is a matrix $\mathbb{U}_d \in \mathcal{R}^{k \times K_d}$ where each column vector represents a k dimensional embedding for a corresponding category. $k > 0$ is a user defined integer. For example, in Fig. 3.1, we have $k = 4$. Each categorical feature would retrieve its corresponding embedding in the lookup table as its new feature representation.

Mathematically, we let $\mathbf{u}_i^d$ be the i^{th} column vector in lookup table $\mathbb{U}_d$. After embedding retrieve, we have the new feature representation $e_d(\mathbf{x}^C) = \mathbf{u}_i^d$.

This new representation is then fed into a new numerical feature embedding block to get the univariate output,

$$g_d(\mathbf{x}) = f_{D_1+d}(\mathbf{u}_i^d) \tag{3.4}$$

3.4.3 Classification

After obtaining the embedding for both numerical and categorical features, DELR concatenate them together to form a new representation, $\Phi(\mathbf{x})$, for the original input $\mathbf{x}$. Note that each feature in the new representation can be traced to its corresponding raw attribute. Now we have

$$\Phi(\mathbf{x}) = (f_1(\mathbf{x}), ...f_{D_1}(\mathbf{x}), g_1(\mathbf{x}), ..., g_{D_2}(\mathbf{x})) \tag{3.5}$$

We let $\phi_i(\mathbf{x})$ to denote the ith element of $\Phi(\mathbf{x})$.

LR is then used to estimate the probability that a given input is positive. We can extend our model to support multi-class classification by replacing LR with softmax regression.

Following the same training procedure as softmax regression, we minimize the cross entropy between true label distributions and prediction distribution. The embedding of each single dimension is learned jointly through stochastic back-propagation. To reduce the effect of over fitting, ℓ_2 normalization is applied when training the model.

3.5 Discussions

In this section, we discuss the nonlinearity, interpretability, i.e. accountability and actionability of our proposed DELR model through a toy dataset shown in Fig. 3.2. We use red pluses and white circles to denote the positive and negative instances, respectively.

Nonlinearity. This dataset is not linearly separable as we cannot draw a single line that can perfectly split those two categories. LR obviously fails here. We train a DELR model using this toy dataset, and draw the probability output in this 2-D space. As we can see from Fig. 3.2, the decision boundary is a smooth circle, containing all the red pluses. The power of nonlinear separability granted by DELR yields a very reasonable separation curve on this 2-D space.

Accountability. Accountability refers to model's ability to reveal the significance of each feature in making the final prediction for some instance. In DELR, each attribute x_i is first transformed into $\phi_i(\mathbf{x})$ through numerical feature mapping layer. Final prediction is determined by the sign the following equation,

$$w_1\phi_1(\mathbf{x}) + w_2\phi_2(\mathbf{x}) + b \tag{3.6}$$

where b is the bias term. For this toy dataset, we find $b = 0$ after training. To discover the contribution of each input feature to the final prediction, we plot the value of $w_1\phi_1(\mathbf{x})$

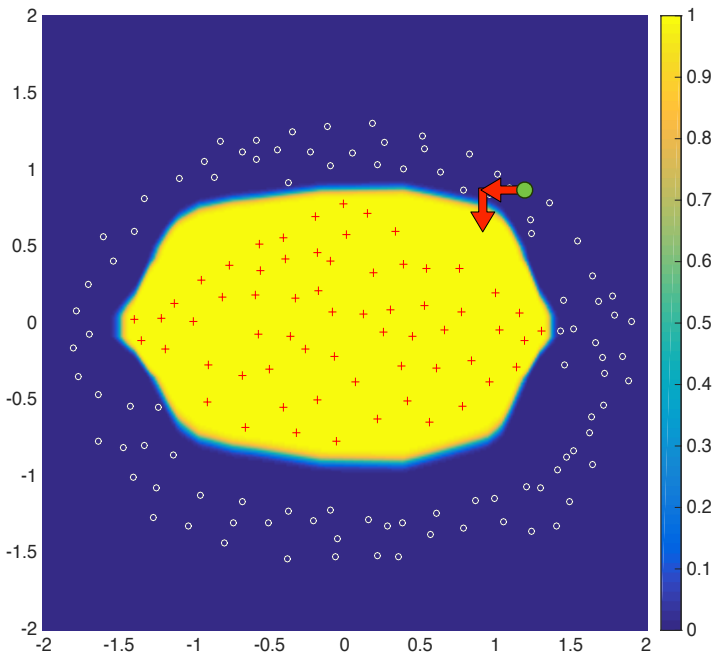

Figure 3.2: Probability output of DELR on a toy example. The green dot represent an instance with the input value of $\{1.2, 0.8\}$. The red arrows are extracted actions to flip the label of greed dot.

and $w_2\phi_2(\mathbf{x})$ with the change of x_1 and x_2 in Fig. 3.3. These two figures clearly illustrate the correlation between the feature value and its corresponding contribution, which is not monotonic as in LR. We call them coordinate plots in the rest of this chapter.

Now suppose we are given a data point $(x_1, x_2) = (1.2, 0.8)$, which is plotted as a green dot in Fig. 3.2. The model predicts it as negative. We can get the contributions of the two dimensions from Fig. 3.3, which are -0.21 and -8.4. At this point, both features are making negative contributions to the final prediction. Note that such accountability is not offered in neural networks. This is because features are intricately interwoven in the hidden layers of neural networks, making it hard to account for each feature's contribution to the final outcome.

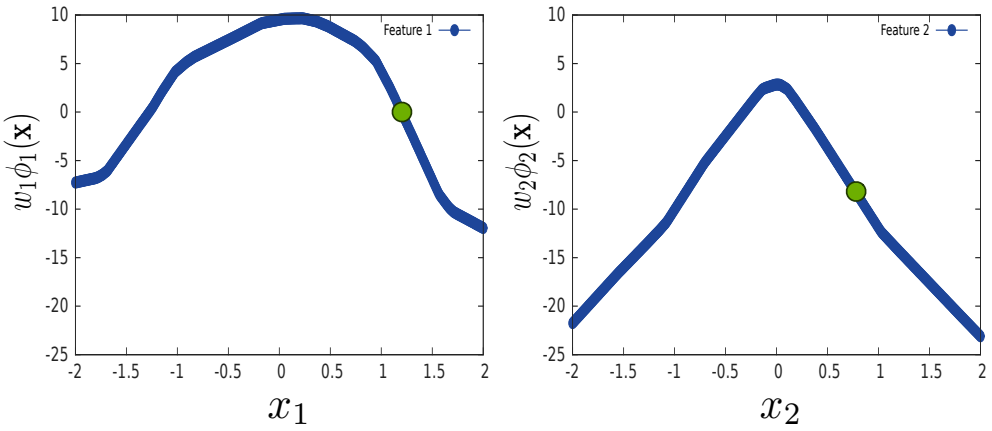

Figure 3.3: The contribution of each input attribute for the final prediction. Left: The correlation between x_1 and the first component of Equation 3.6. Right: The correlation between x_2 and the second component of Equation 3.6.

Actionability. Actionability is a much desired property in areas such as clinical prediction and market analysis. Classifiers need to not only identify critical features that lead to the prediction, but also make recommendations on how to change these features in order to clear the threats or achieve goals.

A practical constraint we should consider is that we can only change a limited number of input features. For example, in clinical predictions, it may be impractical to change all features of a patient simultaneously. Instead, we aim to make small changes to one or a few features, such as temperature, blood pressure or pulse, which are enough to revert the label.

In LR, each feature is independent, and has a constant slope in its contribution curve due to LR's linearity. Thus, we can make small changes on a few features with the largest absolute weights (slopes). In DELR, we can also look at the slope of each feature in the contribution curves as in Fig. 3.3. However, the slope is no longer a constant everywhere. Thus, we propose an iterative algorithm which selects features to change one by one. For an instance $\mathbf{x}$ that we want to revert its label, our iterative action extraction algorithm first selects the feature d with the maximum slope $\left|\frac{d\phi_d(\mathbf{x})}{dx_d}\right|$ and updates it by a small step. Then, the

algorithm again selects the feature with the largest slope, but at the updated point. The process is iterated until reaching a budget. The selected features are returned for making changes.

The slope can be calculated through backpropagation from the second to last layer, or using the finite element method, both of which are computationally efficient. We use the same cost for changing each feature by a unit. The cost can also be non-uniform as we can use step sizes inversely proportional to the costs.

Let's again use the green dot in Fig. 3.2 for demonstration. Now that we know the label of this instance is negative, we want to make some changes on the input features to revert its label. Suppose we have a budget $B = 2$ and step size 0.25. At first, the slope of x_1 is larger, as seen from Fig. 3.3. Thus, the point moves left for 0.25. At the next point, the slope of x_2 becomes larger than that of x_1. Thus, the point moves down for 0.25 and reaches the positive region. In comparison, if we stick to one feature and do not iteratively update the slope calculation, it will cost more to reach the positive region either moving down or moving left.

3.6 Experiments

We conduct extensive experiments to evaluate DELR on several benchmark datasets. We first test the performance of DELR on UCI datasets[4], which contain both numerical feature datasets and mixed-type feature datasets. Then we exhibit the convenience and interpretability on classifying a large-scale dataset with a huge amount of categories. All tested datasets are publicly available. For each dataset, we run 5 fold cross validation and report the average performance and standard derivation. We further hold out 1/3 from the training set as the validation set. All algorithms are trained on the training set, choosing hyper-parameters

[4]https://archive.ics.uci.edu/ml/datasets.html

Table 3.1: Performance of various methods on various UCI datasets. Note that LR, DT, DLR and DELR have accountability. LR, DT, DLR, GBDT and DELR have actionability. "N/A" indicates memory overflow.

Dataset	breast_cancer	splice	musk	mushroom	mnist38	nursery	adult	census-income
N	683	1,000	6,598	8,124	11,982	12,960	30,162	299,285
P	9	60	166	22	784	8	14	41
Data-type	Numerical	Categorical	Numerical	Mixed	Numerical	Categorical	Mixed	Mixed
LR	89.70±5.33	81.50±3.05	95.13±0.61	100±0.00	96.87±0.36	91.93±0.85	84.01±0.20	94.97±0.07
DT	90.70±1.27	87.80±1.94	96.49±0.66	100±0.00	96.10±0.26	98.61±0.14	79.56±0.41	93.06±0.07
DLR	96.42±1.61	92.40±2.68	95.20±0.74	100±0.00	N/A	N/A	N/A	N/A
GBDS	92.99±1.89	92.20±1.36	95.60±0.41	99.95±0.41	96.12±0.24	92.32±0.39	84.20±0.34	95.00±0.03
DELR	**97.57±1.16**	**93.00±0.55**	**97.57±1.16**	**100.00±0.00**	96.05±0.38	92.83±0.38	**84.57±0.28**	**95.43±0.14**
SVM-rbf	93.27±1.95	91.80±1.72	95.08±5.99	100±0.00	97.83±1.23	98.01±0.33	83.12±0.46	94.26±0.14
RF	94.27±1.62	94.50±2.11	97.39±0.25	100±0.00	98.73±0.25	99.06±0.21	83.82±0.21	95.25±0.09

based on the validation set and evaluated on the test set in each fold. Hyper-parameters are selected for all algorithms with Bayesian optimization [127] implemented in the *spearmint*[5] package.

3.6.1 Evaluation on UCI datasets

We evaluate the performance of DELR on several UCI benchmark datasets as shown in Table 3.1. We show the dataset statistics on the left part of the table and the test accuracy on the right part. Here, *N*, *P* and *Data-Type* represents the number of instances, the number of features and data type in each datasets, respectively. We try to include datasets of different scales, covering small sized dataset such as "breast-cancer" to large scale datasets such as "census-income". Among all these datasets, "adult", "mushroom" and "census-income" contain both categorical feature and numerical features. "splice" and "nursery" only contain categorical features while the rest datasets only have numerical features.

From Table 3.1, we can make the following observations. First, LR, the linear classifier performs the worst for the most of the time. Second, DELR performs the best among all classifiers on six out of eight datasets, demonstrating its superior nonlinear separability. For

[5]https://github.com/JasperSnoek/spearmint

Table 3.2: Summary of the triptype dataset

Feature Name	#Categories
VisitNumer	95,674
Weekday	7
UPC	97,715
ScanCount	39
adultDepartment Description	69
Fineline Number	5,196

Table 3.3: Test accuracy on the triptype dataset

Dataset	SFT	DT	DELR
Triptype	71.41%	61.86%	**72.59%**

dataset nursery, DT outperform the rest interpretable models by a large margin, indicating the highly complex distribution of this dataset. Rule based algorithms handle extreme cases better in this case. DLR come across memory overflow on four largest datasets and DELR outperform DLR on all datasets, demonstrating the superior of deep feature embedding compared with kernel density estimator. In addition, DELR is time efficient. The running time on the smallest dataset is less than one minute and it takes about 3 hours to train the largest dataset. As we are using stochastic gradient descent, training time is proportional to the number of instances.

We further perform action extraction on adult datasets, determining whether a person makes over 50K a year. Given a negative instance, the algorithm suggests the person to switch his work class to Self-emp-inc or achieve doctoral education level. Further, if we set the cost of changing categorical features to a large number, then the algorithm suggests increasing working hour, which is quite reasonable.

3.6.2 Evaluation on the Walmart dataset

We evaluate DELR on triptype[6], a large real-world market analysis related dataset with many categories to demonstrate the distinctive advantages. This dataset is a transactional dataset of items purchased at Walmart. The goal of the task is to predict the type of each customer trip, which would help Warlmart's decision making in business and improve customers' shopping experiences. There are 38 trip types in total including a small daily dinner trip, a weekly large grocery trip, and so on. This dataset contains 647,054 instances, each of which contains 6 categorical features[7] .

The difficulty of mining this dataset lies in the huge amount of categories as shown in Table 3.2. There are in total 198,700 distinct categories, leading to a 198,700-dimensional feature vector if one-hot encoding is used. Few classifiers can handle this dataset directly.

We only show the results of softmax regression (SFT), DT and DELR as RF, SVM-rbf and DLR cannot achieve any meaningful results within one day on this dataset even with state-of-the-art packages. For DELR, we adopt the same architecture as the previous experiment. We intentionally use the raw feature without any feature engineering. We show the test performance of each method on Table 3.3. We can see that DELR again outperforms all the other classifiers.

Another advantage of DELR is that it can learn meaningful feature embedding and quantify categorical features. We interpret the category correlation through visualizing the output of the nonlinear feature embedding block. We set the embedding dimensionality to 2 and train DELR from scratch till it converges. Next, we plot the categories using their embeddings as

[6]https://www.kaggle.com/c/walmart-recruiting-trip-type-classification

[7]Note that only the training set is available online, we randomly select 70% as the training set and the rest as the test set.

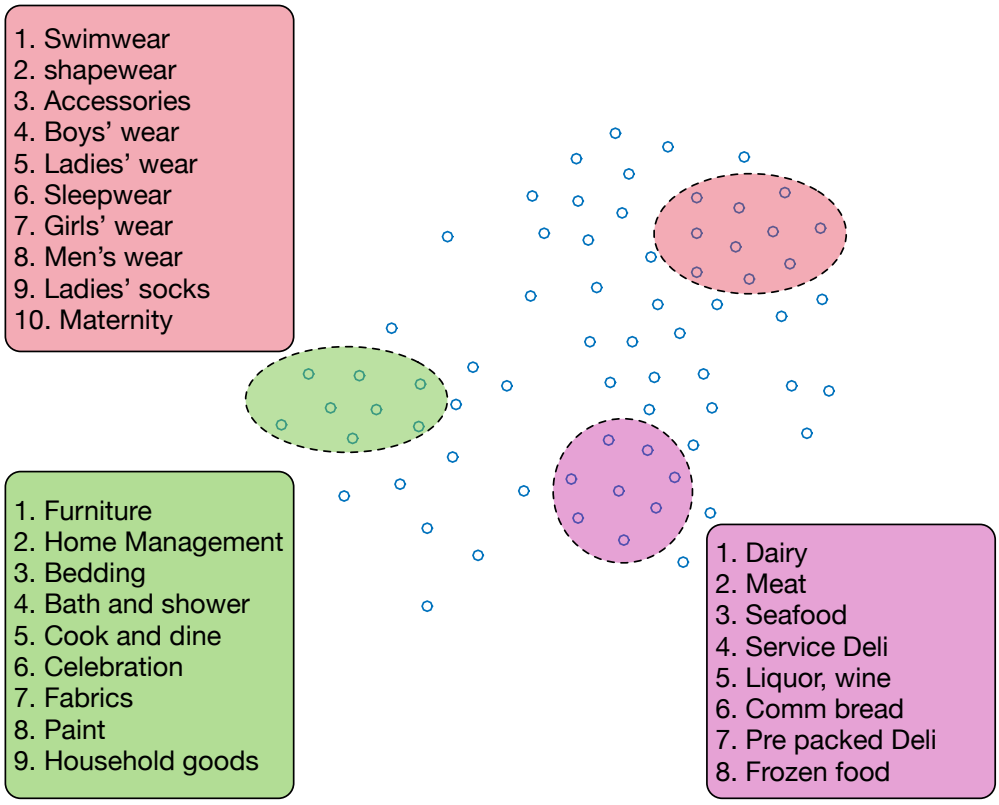

Figure 3.4: The embedding for the "Department Description" feature. Each point in this figure represents a unique category. We show three representative clusters.

coordinates. As "Department Description" is the only feature whose semantic meaning of each category is released, we only present the visualization of this feature, which contains 69 categories as shown by blue circles in Figure 3.4. Due to space limit, we are not allowed to show all the category names in the figure. We can observe that similar categories are located closely while dissimilar categories are far away from each other. We can find many surprisingly meaningful clusters. In Fig. 3.4, the red cycle covers clothes related features, such as "ladies' wear", "mens' wear", etc. There are also some clusters relating to food, home decoration, horticulture, and so on. These results indicate that DELR learns meaningful embeddings, which helps knowledge extraction and further augments interpretability.

Table 3.4: Experimental results on 30-day mortality prediction. DELR outperform all the baseline methods.

Method	AUROC	AUPRC	Specificity	Sensitivity
DT	0.6513	0.0598	0.95	0.3398
LR	0.8455	0.0749	0.95	0.4175
GBDS	0.8658	0.0911	0.95	0.4439
SVM	0.8609	0.0823	0.95	0.4417
RF	0.8536	0.0750	0.95	0.4175
DELR	**0.8725**	**0.0981**	0.95	**0.4515**

3.7 Evaluation on real world clinical dataset

In this section, we apply DELR onto a real-world clinical dataset, performing 30-day postoperative mortality prediction. This work is done in partnership with Barnes-Jewish Hospital (BJH), one of the largest hospitals in the United States. Our data includes all preoperative, intraoperative and postoperative data combined with other inpatient and outpatient EMR data. More than 110,000 surgeries' data is collected between 2012 and 2016, each of which contains 44 preoperative EMR features and 49 vital signs. Thirty-day postoperative mortality is used as the output label. After data screening, we randomly split the dataset into training set (70,000 patients), validation set (10,000 patients), and testing set (19,791 patients), at the ratio of roughly 7:1:2.

Preoperative data are static data collected from patients before the operations. 15 numerical features and 32 categorical features are includes in the Pre-op data. Intraoperative data are in the form of multi-variate time series of patients' vital signs and general signals monitored throughout patients' operations, in which, 10 vital signs are selected. We calculate its mean and standard deviation.

Our target outcome is 30-day mortality, which has a positive-negative ratio of approximately 1:100. We have tried two different methods to deal with this imbalance. The first method is

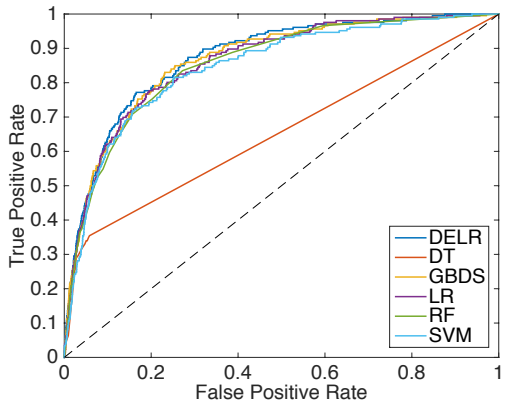

Figure 3.5: Receiver Operating Characteristic (ROC) of the model performance on the test set.

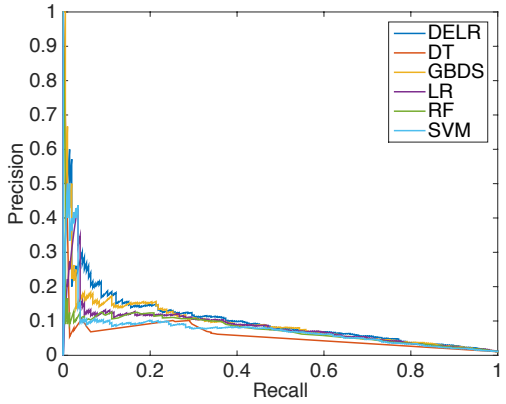

Figure 3.6: Precision-Recall curve of the model performance on test set.

to use class weights inversely proportional to their proportion to multiply the loss of positive training examples 100 times larger than that of negative training examples. The second method is to upsample positive training examples by 100 times each. We tested all baseline methods and DELR and all of them perform better using the second upsampling method. All the following experiments use upsampling method.

We evaluate all the forecasting models using well-accepted criteria including: Area Under the curve of Receiver Operating Characteristic (AUROC), sensitivity and specificity.

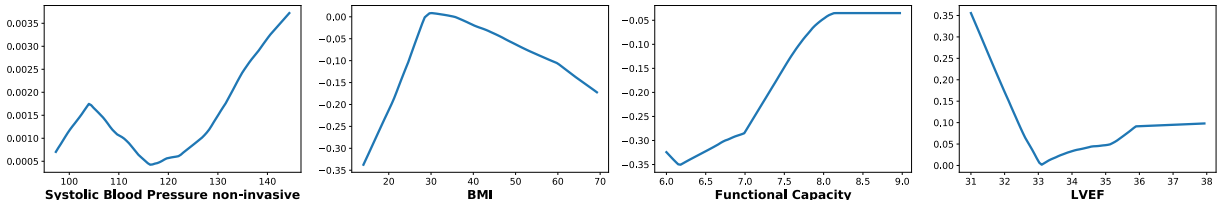

Figure 3.7: We select four features that is not in linear relationship between the input value and 30-day mortality rate.

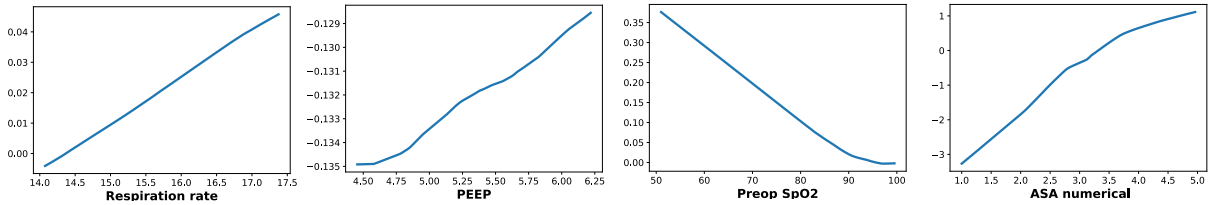

Figure 3.8: We select four features that remains linear after feature embedding. This indicates that DELR regularizes very well, not memorizing the input data through learning complex decision rules.

In addition, as our data is extremely imbalanced, Receiver Operating Characteristic (ROC) can be sometimes deceptive on evaluating the model performance [14, 40]. Therefore, we also show the Precision-Recall curve of each model and adopt Area Under Precision-Recall Curve (AUPRC), which measures the average precision as an additional evaluation criterion. All the models are tuned based on the AUPRC performance on the validation set and then report the model performances evaluated on the testing set.

Another key metric we evaluate and compare is the sensitivity at 95% specificity since it is important to maintain a high specificity (i.e., low false alarm rate) for meaningful clinical decision support.

3.7.1 Baseline Methods

We compare DELR with the most widely used classifiers, decision tree (DT), logistic regression (LR), support vector machine (SVM) and random forest (RF) and gradient boosted stumps (GDBS). Categorical features are represented as one-hot encoding vectors for baseline methods.

3.7.2 Experimental Results

Table 3.4 shows the model performances. We observe that DELR consistently outperforms both interpretable models and non-interpretable models in terms of AUROC and AUPRC. The reason DELR can beat SVM and RF is that this clinical dataset is very easy to overfit. Our model regularizes very well in this scenario. We will discuss more about it in the next paragraph. Sensitivity at 95% specificity level is included in the performance chart as well. Even under this high specificity, DELR achieves 0.4515 sensitivity, which is much higher than rest models. We also plot out all the ROC curves in Figure 3.5. DELR achieves the highest AUROC of 0.8725. As our dataset is extremely imbalance, the Precision-Recall curve shown in Figure 3.6 contains more meaningful information. The positive-negative ratio is approximately 1:100, indicating that random guess can get only 0.01 AUPRC. DELR achieves an AUPRC of 0.0981, which is nearly ten times better than random guess and more than 30% gain compared with the LR model.

Next, we check the interpretability of DELR on this clinical dataset. As we've talked in the discussions section, coordinate plot of each feature can be drawn to visualize its relationship with 30-day mortality. We plot out eight features, shown in Fig. 3.7 and Fig. 3.8. The x-axis is the feature value and y-axis is the feature contribution to the final prediction. Positive value enhance the probability of 30-day mortality. Among these eight plots, the first four plots in Fig. 3.7 show nonlinear relationship between the input feature and final contribution

to the prediction. Classifiers such as logistic regression cannot handle this situation. In addition, the nonlinear transformations are also in line with our expectation. We also discover some features remain linear after transformation as shown in Fig. 3.8. Take SpO2 as an example, the higher the value of SpO2, the less likely the patient will die within 30 days. We've checked SVM-rbf and RF, no such linear relationship is found. Without mapping all features to complex nonlinear representation, DELR can generalize better. We will further analyze DELR on this clinical dataset with domain experts in the future based on coordinate plots and extract reliable suggestions from this model.

3.8 Conclusions

Complex models such as DNNs have strong learning ability and high accuracy. However, in many tasks such as marketing and clinical prediction, LR is still a preferred choice as it provides good interpretability and scales well. In this chapter, we propose a novel DELR model, inheriting those nice properties of LR while overcoming its drawbacks of linearity and inability to handle categorical features. DELR incorporates dimension-wise nonlinear feature embedding using deep neural networks and feeds the embeddings into LR for classification. Extensive analysis and experimental results demonstrate that DELR is a compact yet powerful model, achieving both high accuracy and excellent interpretability.

Chapter 4

Factored Generalized Additive Model

4.1 Motivation and Introduction

Patients undergoing surgery and anesthesia experience external stresses that place them at risk for numerous complications, including acute kidney injury and acute respiratory failure. One of the roles of the anesthesia clinician is to regulate the patient's physiology to minimize these risks. Logistic regression-based models for predicting postoperative acute kidney injury [29, 74, 105] and acute respiratory failure[6, 59, 69] have been developed by multiple groups. Linear models offer a high degree of transparency regarding which features drive the output, but they are inherently limited in their flexibility, which limits their predictive accuracy. Various machine learning (ML) models have been proposed to solve these clinical prediction tasks with greater accuracy. Classifiers for acute kidney injury have been described in postoperative [136] and non-surgical hospitalized patients[72, 81]. Although these ML models outperform logistic regression, they are not frequently used in clinical practice in part due to their lack of interpretability.

An interpretable model must provide predictions that are both accountable and actionable. An accountable model provides information about which features are contributing to the output prediction. This information can include feature importance and feature interactions. An actionable model provides guidance regarding how to modify the input features so that the post-intervention features will lead to the desired output.

Interpreting ML models is an extremely active field [17, 58, 124]. Incorporating interpretability constraints directly into the structure of the model and using post-hoc interpretation methods are two of the main directions[101]. Most existing work focuses on accountability. Che et al [22] transfer the DNN's knowledge to gradient boosting trees (GBT) using knowledge distillation and interpret feature importance through measures of variable importance designed for GBT. Another work [109] visualizes the region of interest through class activation maps. Ge et al.[54] feed features extracted from recurrent neural networks into a logistic regression model for prediction, where importance of the transformed features can be directly read off. Neural networks with attention-like mechanisms are also popular to visualize the features which contribute the most to a classifier for a particular case [124]. A smaller number of techniques address the actionability requirement. An early work [35] proposed an integer linear programming method to extract actionable knowledge from a random forest. Gardner et al. proposed a label changing method by searching semantically meaningful changes to an image under its manifold space[52]. As far as we know, little work has been done on addressing accountability and actionability at the same time in the clinical area.

An extension to generalized linear models, generalized additive models (GAMs), can address accountability and actionability simultaneously. Examples includes LR, density based logistic regression [25] (DLR), generalized additive neural networks [108] (GANN), and deep embedding logistic regression [37] (DELR). LR assumes a fixed (up to a parameter) monotonic relationship between each feature and the outcome probability, limiting its flexibility and

predictive performance. GAMs loosen these assumptions by inferring a transformation of the inputs with the full flexibility of non-parametric or parametric methods. For example, DLR transforms each feature through a kernel estimator, and our DELR performs feature-wise nonlinear transformation using neural networks. GANN, which used a single hidden layer, can be treated as a special case of DELR, which used multi-layer DNNs. Despite the increased flexibility, with suitable constraints we can extract accountability and actionability from GAMs. Given an input example, GAMs allow us to calculate the contribution of each feature to that example's predicted value. Feature contribution curves can be drawn to provide actionable directions on the optimal change and magnitude of improvement for each numeric feature. However, only when all the features are conditionally independent (given the label) can GAMs model the true distribution of data[85]. Feature interactions are not allowed in GAMs, restricting their performance in dealing with complex datasets. In addition, GAMs have the undesirable property of treating static and time-varying features equally. For example, demographic characteristics such as age, gender, and height are not possible to change. On the other hand, it is possible to deliver interventions that modify a patient's vital signs during surgery.

To address these problems, we propose a variation of GAMs that splits features into time-varying (or targeted) features and static features. F-GAM fits a context-based scaling for each time-varying factor based on the static factors, substantially increasing its flexibility compared to models which require the effect of a feature to be the same for all examples, but retains the ability to derive personalized feature-effect curves [36]. F-GAM retains the full flexibility of a DNN for the effect of static features and DNN-based flexibility for the transformation of time-varying factors. We implement F-GAM as an end-to-end trained model with minimal hyper-parameters. In extreme cases where there are no static features available, F-GAM reduces to DELR. If there are no time-varying features, F-GAM becomes a

DNN. We empirically validate the accuracy performance of F-GAM with existing ML models, including other GAMs and demonstrate the interpretability of F-GAM through a case study on predicting acute kidney injury.

4.2 Background and Notation

4.2.1 Notation

Operating room data contains both preoperative data such as demographic information and intraoperative data such as vital signs and medications administered. Given a patient i, pre-op data $\mathbf{x}_i^S \in \mathcal{R}^{D_1}$ collected before the surgery are treated as static feature vectors while intra-op data represented as $\mathbf{x}_i^{TV} \in \mathcal{R}^{D_2}$ can be modified in real time. Together, we use $\mathbf{x}_i = [\mathbf{x}_i^S, \mathbf{x}_i^{TV}] \in \mathcal{R}^D$ to denote input features and $y_i \in \{0, 1\}$ to represent the binary outcomes. Our examples are binary classification, but the extension to multi-class classification is straightforward with a final softmax transformation and appropriate loss function.

4.2.2 Generalized Additive Models

A generalized additive model (GAM) is an ensemble of D univariate functions, where D is the number of features. We use x_j and y to denote the jth dimension of input $\mathbf{x}$ and class label, respectively. The output of each univariate function, denoted as $f_t(x_j)$ is a real number. We can write the GAM structure as

$$g(E(y)) = \beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_D(x_D), \tag{4.1}$$

where the function g is the link function, bounding the range of right hand side value of Eq.4.1, and $E(y)$ is the expected value of the label conditional on $\mathbf{x}$. Constraints on f_k such as smoothness or degrees of freedom regularize the estimation problem to decrease out-of-sample loss. With a little abuse of notations, we use $F(\mathbf{x})$ to denote $E(y|\mathbf{x})$ throughout this chapter for ease of presentation. By inversing the link function, the GAM has the form,

$$F(\mathbf{x}) = g^{-1}[\beta_0 + f_1(x_1) + f_2(x_2) + \cdots + f_D(x_D)], \tag{4.2}$$

where the model output is controlled by the sum of each univariate function. GAM assumes all the features of input $\mathbf{x}_i$ are making contributions independently. Interpreting a GAM is straightforward as the marginal impact of a specific feature does not rely on the rest of features; we are able to know the importance of a feature by plotting its corresponding univariate function or calculating its variance over the sample. Actionable changes can be made based the shape of each $f_k(\mathbf{x}_k)$.

Logistic regression is a special case of GAM by choosing logit function $g(x) = \ln \frac{x}{1-x}$ as the link function and setting $f_k(x_k)$ to be $w_k x_k$ yielding

$$F(\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_D x_D), \tag{4.3}$$

where the sigmoid function $\sigma(x) = \frac{1}{1+\exp(-x)}$ is the inverse form of the logit function. LR assumes a monotonic relationship between the final output $F(\mathbf{x})$ and input features due to the linear function f_k. However, this condition doesn't hold in many cases, such as the relationship between ICU transfer rate and age[25].

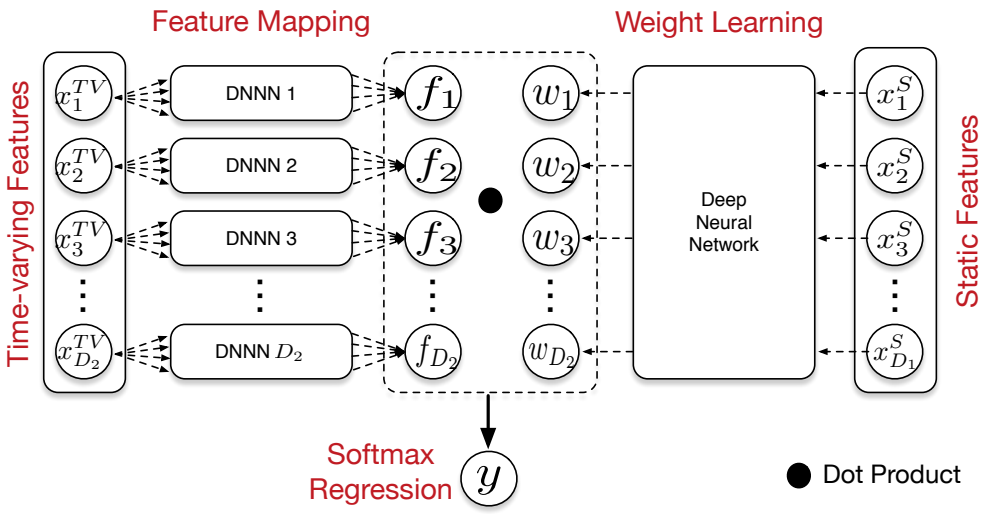

Figure 4.1: Overall architecture of F-GAM. Each circle denotes a scalar. Upper left part is feature mapping module. Every time-varying feature is fed to its own deep and narrow neural network (DNNN) separately. Weight learning module, which is shown in upper right part takes static features as input and calculates feature weights. Note that bias learning module is not plotted in this figure for simplicity.

4.3 Methods - Model Algorithm

In this section, we propose a factored generalized additive model (F-GAM) framework in which interactions between time-varying features and static features are allowed. The overall model has the form

$$F(\mathbf{x}) = \sigma\left[\sum_{t=1}^{D_2} w_t(\mathbf{x}^S) f_t(x_t^{TV}) + w_0(\mathbf{x}^S)\right] \tag{4.4}$$

In F-GAM, w_t is no longer a constant weight parameter, but the output of a DNN that accepts the static feature vector as input and estimates the weight of tth time-varying feature for each case. The feature-wise nonlinear transformation functions f_t, $t = 1, 2, ...D_2$ are jointly estimated. w_0 is a bias / intercept term that also depends only on the static features. In our operative examples, w_0 represents the estimate of risk before any intra-operative data becomes available as long as the input features have been appropriately centered.

F-GAM can be decomposed into four different modules: time-varying feature mapping module, feature weights learning module, bias term learning module and logistic/softmax regression module. We display the F-GAM architecture in Figure 4.1.

4.3.1 Time-varying feature mapping module

In traditional GAMs, the ability of the univariate function f_k to approximate the unknown transformation plays a crucial role in model performance. We choose to use deep and narrow neural networks (DNNN)[37] for the nonlinear feature embedding. Being a universal approximator, a DNNN is able learn complex patterns automatically. The general tools for regularizing neural networks are immediately available to control overfitting without the difficult-to-understand smoothness or degree-of-freedom constraints of other GAM transformations. Each time-varying feature is fed into a DNNN with distinct parameters; however, several hyperparameters (depth, width, dropout, training stopping time) are shared across t to avoid having to search over a large hyperparameter space. The shared architectural parameters also tend to prevent over-fitting of just a few features (data not shown). A learnable look-up table (categorical embedding) is attached before a DNNN for categorical features. In our examples, all time-varying features are quantitative or ordinal rather than categorical.

4.3.2 Feature weights learning module

Rather than applying fixed weights for the input features, we use nonlinear functions w_t to adjust the feature weight dynamically. The nonlinear function should have the following two properties. First, the nonlinear function should not increase the number of parameters dramatically. Second, the nonlinear function should be able to handle both numerical features and categorical ones. Thus, we choose to use deep neural networks as the nonlinear functions.

Rather than assigning each w_t a standalone DNN as we did for f_t, all the weight-learning functions are estimated with a common DNN except the last layer. With this multi-task setup, we are able to exploit the shared structure of the data to reduce the effective number of parameters. Joint predictions of w_t also allow the module to dynamically choose between potentially correlated x_t^M to emphasize, meaning that w_t represents both the relevance and precision of f_t in the given context. For the second property, we use categorical embedding. When there are no static features, w_t is a constant per time-varying feature and F-GAM reduces to DELR. That is to say, DELR is a special case of our model.

4.3.3 Bias term learning module

In order to increase expressiveness of the final model, we add a bias term based on the static features. Again, we use a DNN to model the bias term. This DNN is appended to the penultimate layer of the feature weights module to reduce redundancy. When there are no time-varying features, only the bias term controls the final output. In this case, F-GAM simplifies to a deep neural network.

4.3.4 Logistic/softmax regression module

With all the transformed features and weights ready, we apply the dot product operation to the time-varying feature mapping and the learned weights. After adding the bias term w_0, a sigmoid function σ is used to model the positive rate given input data.

Our F-GAM is trained end-to-end by minimizing the cross entropy loss between true label distribution and prediction distribution. We also apply weight decay and an early stopping strategy to avoid over-fitting. The code is available at https://github.com/nostringattached/FGAM.

4.4 Methods - Experiments

4.4.1 Data Sources

Models were trained and validated using a dataset obtained from a single academic medical center (Barnes Jewish Hospital, St. Louis, Missouri). All adult patients who received surgery with anesthesia between June 2012 and August 2016 were eligible for inclusion. Due to the limited incidence of acute respiratory failure among patients who were not admitted to the intensive care unit (ICU) after surgery, prediction of this complication was limited to patients admitted to the ICU after surgery.

Acute kidney injury and acute respiratory failure were the two complications that were used as targets in the experimental models. Per Kidney Disease: Improving Global Outcomes (KDIGO) criteria, acute kidney injury was defined as an increase in the serum creatinine value by >0.3 mg/dL or >50% within 48 hours, compared to the preoperative value[73]. Acute kidney injury was undefined if the patient was receiving dialysis before surgery. The preoperative creatinine was the most recent value available before surgery, but no more than 30 days before surgery. Acute respiratory failure was defined as mechanical ventilation for >48 hours after surgery or reintubation within 48 hours. Acute respiratory failure was undefined if the patient was receiving mechanical ventilation before surgery, if the patient had a second surgery within 48 hours, or if the patient died within 48 hours.

Baseline demographic characteristics, comorbid health conditions, and preoperative laboratory values were retrieved from the electronic medical record. The total doses of commonly used medications (including intravenous fluids, blood pressure-raising and -lowering agents, sedatives, pain medications, and nephrotoxic antibiotics) were also retrieved. The full list of

features included in the analysis is shown in Table 4.1.

Table 4.1: Features included in the model.

Demographic Characteristics	Age, Height, Weight, Ideal body weight, Body mass index, Sex, Race, CharlsonComorbidity Index, Functional capacity, American Society of Anesthesiologistsphysical status, Surgery type
Comorbid Conditions	Hypertension, Coronary artery disease, Prior myocardial infarction, Congestive heart failure, Diastolic function, Left ventricular ejection fraction, Aorticstenosis, Atrial fibrillation, Pacemaker, Prior stroke, Peripheral artery disease, Deepvenous thrombosis, Pulmonary embolism, Diabetes mellitus, Outpatient insulin use,Chronic kidney disease, Ongoing dialysis, Pulmonary hypertension, Chronic obstructivepulmonary disease, Asthma, Obstructive sleep apnea, Cirrhosis, Cancer, Gastro-esophageal reflux, Anemia, Coombs positive, Dementia, Ever-smoker
Preop Vital Signs	Systolic blood pressure, Diastolic blood pressure, Pulse oximeter, Heart rate
Preop Labs	Albumin, Alanine phosphatase, Creatinine, Glucose, Hematocrit, Partialthromboplastin time, Potassium, Sodium, Urea Nitrogen, White blood cells
Intraoperative Time Series	Mean arterial pressure, Systolic blood pressure, Diastolic blood pressure, Heart rate,Pulse oximeter, Temperature, Respiratory rate, Tidal volume, Peak inspiratory pressure, Positive end-expiratory pressure, Fraction inspired oxygen, End-tidal carbon dioxide,End-tidal anesthetic concentration
Intraoperative Meds and Fluids	Albumin, Amiodarone, Crystalloid (lactated ringers + normal saline), Dobutamine, Ephedrine, Epinephrine, Fentanyl, Furosemide, Gentamicin, Hydromorphone, Midazolam, Nicardipine, Norepinephrine, Packed red blood cells,Phenylephrine, Propofol, Remifentanil, Vancomcyin, Vasopressin, Other blood products

For intraoperative time series features, summary measures were derived. For each feature, the mean, standard deviation, maximum, and minimum over the entire surgery were calculated.

The maximum pulse oximeter reading was omitted due to ceiling effects, while minimum peak inspiratory pressure and minimum tidal volume were omitted due to expected lack of clinical significance. In addition, the fraction of surgery with extreme values of certain parameters were also calculated, using multiple cutoff values. These included duration of low mean arterial pressure (<55, <60, or <65 mmHg), high heart rate (>100, >110, or >120 beats per min), low heart rate (<60, <55, or <50 beats per min), low temperature (<36 or <35.5 °C), low pulse oximeter (<90 or <85%), high exhaled carbon dioxide (>50 mmHg), low exhaled carbon dioxide (<30 mmHg), high peak inspiratory pressure (>30 mmHg), and high tidal volume (>10 mL per kg). Lung compliance was also calculated as final tidal volume divided by final peak inspiratory pressure.

4.4.2 Experimental Technique

For each of the two target outcomes, F-GAM was compared to four baseline models (decision tree [DT], random forest [RF], support vector machine [SVM], and deep neural network [DNN]) and to three GAMs (logistic regression [LR], gradient boosting decision stumps [62][GBDS] and deep embedding logistic regression [DELR]). Note that density based logistic regression (DLR) was not included as it did not finish training in 24 hours. Each model was trained using a 70% random sample of the dataset. 10% of the dataset was selected as a validation set for hyper-parameter tuning and performance was tested on the remaining 20% of the dataset. Model performance was quantified using area under the receiver operating characteristic curve (AUROC) and area under the precision-recall curve (AUPRC). We calculate two-sided 95% confidence intervals for each measure using the statistical analysis method given by Hanley and McNeil [61].

4.5 Results

Table 4.2: AUROC score, AUPRC score and their corresponding 95% confidence interval (CI) of different methods. DT = decision tree. RF = random forest. SVM = support vector machine, DNN = deep neural network. LR = logistic regression, GBDS = gradient boosting decision stumps, DELR = deep embedding logistic regression, F-GAM = factored generalized additive model.

Model		Acute Kidney Injury		Acute Respiratory Failure	
		AUROC 95% CI	AUPRC 95% CI	AUROC 95% CI	AUPRC 95% CI
Baselines	DT	0.580 [0.563, 0.597]	0.137 [0.130, 0.145]	0.535 [0.474, 0.595]	0.043 [0.033, 0.053]
	RF	0.820 [0.806, 0.835]	0.253 [0.243, 0.266]	0.718 [0.658, 0.777]	0.085 [0.068, 0.102]
	SVM	0.794 [0.779, 0.809]	0.215 [0.205, 0.226]	0.698 [0.638, 0.758]	0.094 [0.076, 0.113]
	DNN	0.787 [0.772, 0.802]	0.216 [0.206, 0.227]	0.698 [0.638, 0,758]	0.084 [0.072, 0.109]
GAMs	LR	0.794 [0.783, 0.813]	0.221 [0.212, 0.233]	0.650 [0.052, 0.712]	0.073 [0.058, 0.088]
	GBDS	0.803 [0.788, 0.818]	0.253 [0.242, 0.265]	0.713 [0.654, 0.773]	0.084 [0.070, 0.105]
	DELR	0.800 [0.786, 0.815]	0.235 [0.225, 0.247]	0.708 [0.648, 0.768]	0.083 [0.066, 0.099]
Our Method	F-GAM	**0.824 [0.813, 0.842]**	**0.264 [0.258, 0.282]**	**0.718 [0.659, 0.777]**	**0.106 [0.091, 0.134]**

The dataset included 111,890 patients. Of these patients, 5,018 were excluded from the acute kidney injury model because they were receiving dialysis before surgery or because no postoperative creatinine value was available. Of the remaining 106,872 patients, 6,472 (6.1%) experienced acute kidney injury. Of the original 111,890 patients, 89,688 were excluded from the acute respiratory failure model because they were not admitted to the intensive care unit, while 6,578 were excluded due to preoperative mechanical ventilation or one of the other

exclusion criteria. Of the remaining 15,624 patients, 489 (3.1%) experienced acute respiratory failure.

Performance of the models is shown in Table 4.2, while the receiver-operating characteristic and precision-recall curves are shown in Figure 4.2. For both outcomes, F-GAM provided the highest AUROC and the highest AUPRC. DT and RF are excluded from Figure 4.2 for readability purposes.

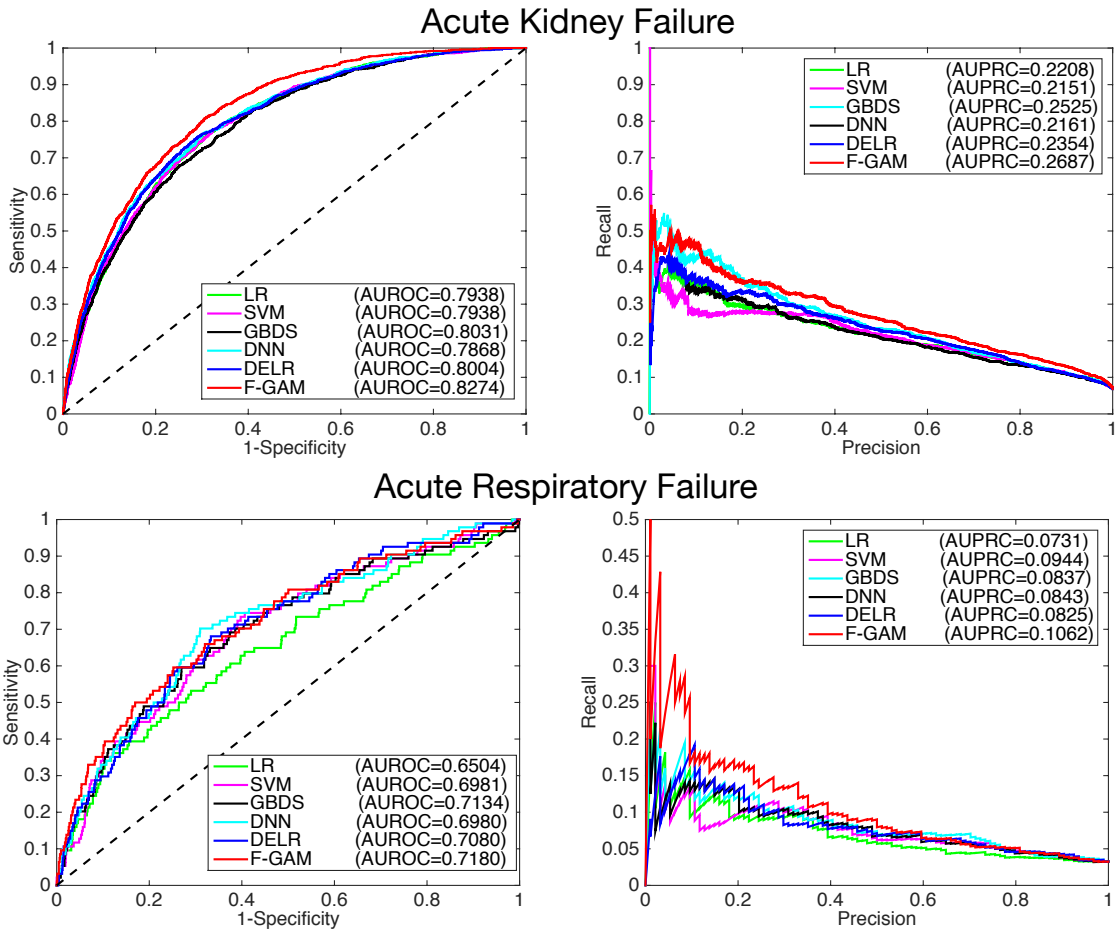

Figure 4.2: ROC curve and precision recall curve (PRC) of different models predicting acute kidney injury and acute respiratory failure. LR = logistic regression, SVM = support vector machine, GBDS = gradient boosting decision stumps, DNN = deep neural network, DELR = deep embedding logistic regression, F-GAM = factored generalized additive model.

Figure 3 demonstrates how the contribution $w_t(\mathbf{x}^S)f_t(x_t^{TV})$ to the predicted risk of acute kidney injury changes at different values x_t^{TV} of four representative time-varying features in two randomly selected patients. Each panel assumes that all other time-varying features remain constant. Points that are higher on the vertical axis represent a larger contribution to the predicted probability.

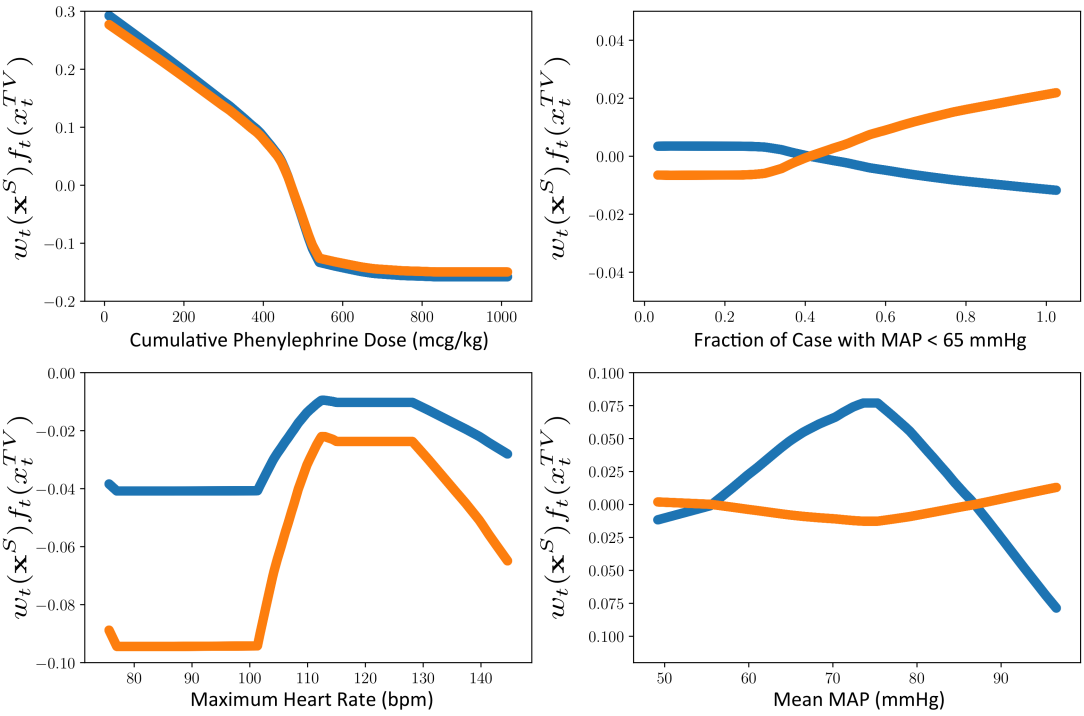

Figure 4.3: Contribution of each feature to the predicted probability of acute kidney injury as a function of feature value. Each panel assumes that all other dynamic features are held constant. The blue curve shows the feature contributions in a 57-year-old healthy female (who ultimately did not have AKI), while the orange curve shows the feature contributions in a 49-year-old female with hypertension, chronic kidney disease, and cirrhosis of the liver (who did have AKI).

4.6 Conclusion

In this chapter, we have described a novel Factored Generalized Additive Model (F-GAM) and demonstrated its use in predicting postoperative acute kidney injury and acute respiratory failure in a historical cohort of patients receiving surgery with anesthesia. F-GAM allows for interactions between static and time-varying input features while retaining the qualities of accountability and actionability. Our model outperformed baseline models and other GAMs in predicting both of the complications tested, and the graphical displays of risk indicated associations that have face validity to an anesthesia clinician. Next steps include application of this technique to other outcomes and prospective deployment of these models for prediction of complications.

Chapter 5

Multi-Scale Convolutional Neural Networks for Time Series Classification

5.1 Introduction

Our daily lives constantly produce time series data, such as stock prices, weather readings, biological observations, health monitoring data, etc. In the era of big data, there are increasing needs to extract knowledge from time series data, among which a main task is time series classification (TSC), the problem of predicting class labels for time series. It has been a long standing problem with a large scope of real-world applications. For example, there has been active research on clinical prediction, the task of predicting whether a patient might be in danger of certain deterioration based on the patient's clinical time series such as ECG signals. A real-time deterioration warning system powered by TSC has achieved unprecedented performance compared with traditional clinical approaches and been applied in major hospitals [96].

Most existing TSC approaches fall into two categories [140]: distance-based methods and feature-based methods.

For distance-based methods, the key part is to measure the similarity between any given two time series. Based on the similarity metrics, the classification can be done using algorithms such as k-nearest neighbors (kNN) or support vector machines (SVM) with similarity-based kernels. The most notable similarity measurement is dynamic time warping (DTW) which aligns two time series with dynamic warping to get the best fit. It could be easily done through dynamic programming.

For feature-based methods, each time series is characterized with a feature vector and any feature-based classifier (e.g. SVM or logistic regression) can be applied to generate the classification results. There have been many hand-crafted feature extraction schemes across different applications. For example, in a clinical prediction application, each time series is divided into several consecutive windows and features are extracted from each window. The final feature vector is a concatenation of feature vectors from all windows [96]. The features include simple statistics such as mean and variance, as well as complex features from detrended fluctuation analysis and spectral analysis. Another approach extracts features based on shapelets which can be regarded as a signature subsequence of the time series. Typically, potential candidate shapelets are generated in advance and they can be used in different ways. For example, they can be considered as a dictionary and each shapelet is regarded as a word. The time series is then described by a bag-of-word model. A more recent study [89] constructs the feature vector such that the value of each feature is the minimum distance between anywhere in the time series and the corresponding shapelet. A drawback of the shapelet method is that it requires extensive search for the discriminative shapelets from a large space. To bypass the need of trying out lots of shapelet candidates, Grabocka *et*

al. [56] propose to jointly learn a number of shapelets of the same size along with the classifier. However, their method only offers linear separation ability.

In recent years, convolutional neural networks (CNNs) have led to impressive results in object recognition [82], face verification [121], and audio classification [84].

A key reason for the success of CNNs is its ability to automatically learn complex feature representations using its convolutional layers. With the great recent success of deep learning and the presence of so many various handcrafted features in TSC, it is natural to ask a question: is it possible to automatically learn the feature representation from time series? However, there have not been many research efforts in the area of time series to embrace deep learning approaches. In this work, we advocate a novel neural network architecture, Multi-Scale Convolutional Neural Network (MCNN), a convolutional neural network specifically designed for classifying time series [34].

A distinctive feature of MCNN is that its first layer contains multiple branches that perform various transformations of the time series, including those in the frequency and time domains, extracting features of different types and time scales. Subsequently, convolutional layers apply dot products between the transformed waves and 1-D learnable filters, which is a general way to automatically recognize various types of features from the input. As a single convolutional layer can detect local patterns similar to shapelets, stacking multiple convolutional layers can construct more complex patterns. As a result, MCNN is a powerful general-purpose framework for TSC. Different than traditional TSC methods, MCNN is an end-to-end model without requiring any handcrafted features. We conduct comprehensive experiments and compare with many existing TSC models. Strong empirical results show that MCNN elevates the state-of-the-art of TSC. It gives superior overall performance, surpassing most existing models by a large margin, especially when enough training data is present.

5.2 Related Work

TSC has been studied for long time. A plethora of time series classification algorithms have been proposed. Most traditional TSC methods fall into two categories: distance based methods that use kNN classifiers on top of distance measures between time series, and feature based classifiers that extract or search for deterministic features in the time or frequency domain and then apply traditional classification algorithms. In recent years, some ensemble methods that collect many TSC classifiers together have also been studied. A full review of these methods is out of the scope here but we will do a comprehensive empirical comparison with leading TSC methods in the next section. Below, we review some works that are most related to MCNN.

In recent years, there have been active research on deep neural networks [5, 11, 65] that can combine hierarchical feature extraction and classification together. Extensive comparison has shown that convolution operations in CNN have better capability on extracting meaningful features than ad-hoc feature selection [97]. However, applications of CNN to TSC have not been studied until recently.

A multi-channel CNN has been proposed to deal with multivariate time series [149]. Features are extracted by putting each time series into different CNNs. After that, they concatenate those features together and put them into a new CNN framework. Large multivariate datasets are needed in order to train this deep architecture. While for our method, we focus on univariate time series and introduce two more branches that can extract multi-scale and multi-frequency information and further increase the prediction accuracy. [38] feeds CNN with variables post-processed using an input variable selection (IVS) algorithm. The key difference compared with MCNN is that they aim at reducing the input size with different IVS algorithms. In contrast, we are exploring more raw information for CNN to discover.

In addition to classification, CNN is also used for time series metric learning. In [150], Zheng *et al.* proposed a model called convolutional nonlinear neighbourhood components analysis that preforms CNN based metric learning and uses 1-NN as the classifier in the embedding space.

Shapelets attract lots of attention because people can detect shapes that are crucial to TSC, providing insights and interpretability. However, searching shapelets from all the time series segmentations is time consuming and some stoping methods are proposed to accelerate this procedure. In [56], Grabocka *et al.* proposed a model that can learn global shapelets automatically instead of searching. As discussed in Section 3.2, MCNN is general enough to be able to learn shapelets.

CNN can achieve scale invariance to some extent by using the pooling operation. Thus, it is beneficial to introduce a multi-scale branch to extract short term as well as long term features. In image recognition, CNNs keep feature maps in each stage and feed those feature maps altogether to the final fully connected layer [86] . By doing this, both short term and higher level features are preserved. For our model, we down sample the raw data into different time scales which provides low level features of different scales and higher level features at the same time.

5.3 Multi-Scale Convolutional Neural Network (MCNN) for TSC

In this section, we formally define the aforementioned time series classification (TSC) problem. Then we describe our MCNN framework for solving TSC problems.

5.3.1 Notations and Problem Definition

A time series is a sequence of real-valued data points with timestamps. In this work, we focus on time series with identical interval length. We denote a time series as $T = \{t_1, t_2, ..., t_n\}$, where t_i is the value at time stamp i and there are n timestamps for each time series.

We denote a labelled time series dataset as $D = \{(T_i, y_i)\}_{i=1}^{N}$ which contains N time series and their associated labels. For each $i = 1, \cdots, N$, T_i represents the i^{th} time series and its label is y_i. For ease of presentation, we consider classification problems where y_i is a categorical value in $\mathcal{C} = \{1, \cdots, C\}$ where $C \in \mathbb{Z}^+$ is the number of labels. However, our framework can be easily extended to real-valued regression tasks. The TSC problem is to build a predictive model to predict a class label $y \in \mathcal{C}$ given an input time series T. Unlike some previous works, we do not require all training and testing time series to have the same number of timestamps in our framework.

5.3.2 MCNN framework

Time series classification is a long standing problem that has been studied for decades. However, it remains a very challenging problem despite great advancement in data mining and machine learning. There are some key factors contributing to its difficulty. First, different time series may require feature representations at different time scales. For example, it is found that certain long-range (over a few hours involving hundreds of time stamps) patterns in body temperature time series have predictive values in forecasting sepsis [45]. Existing TSC features can rarely adapt to the right scales. Second, in real-world time series data, discriminative patterns in the time series is often distorted by high-frequency perturbations and random noises. Automatic smoothing and de-noising procedures are needed to make the overall trend of the time series more clear.

To address these problems for TSC, we propose a multi-scale convolutional neural network (MCNN) framework in which the input is the time series to be predicted and the output is its label. The overall architecture of MCNN is depicted in Figure 5.1.

The MCNN framework has three sequential stages: transformation, local convolution, and full convolution.

1) The **transformation stage** applies various transformations on the input time series. We currently include identity mapping, down-sampling transformations in the time domain, and spectral transformations in the frequency domain. Each part is called a *branch*, as it is a branch input to the convolutional neural network.

2) In the **local convolution stage**, we use several convolutional layers to extract the features for each branch. In this stage, the convolutions for different branches are independent from each other. All the outputs will pass through a max pooling procedure with multiple sizes.

3) In the **full convolution stage**, we concatenate all extracted features and apply several more convolutional layers (each followed by max pooling), fully connected layers, and a softmax layer to generate the final output. This is an entirely end-to-end system and all parameters are trained jointly through back propagation.

5.3.3 Transformation stage

Multi-scale branch. A robust TSC model should be able to capture temporal patterns at different time scales. Long-term features reflect overall trends and short-term features indicate subtle changes in local regions, both of which can be potentially crucial to the prediction quality for certain tasks.

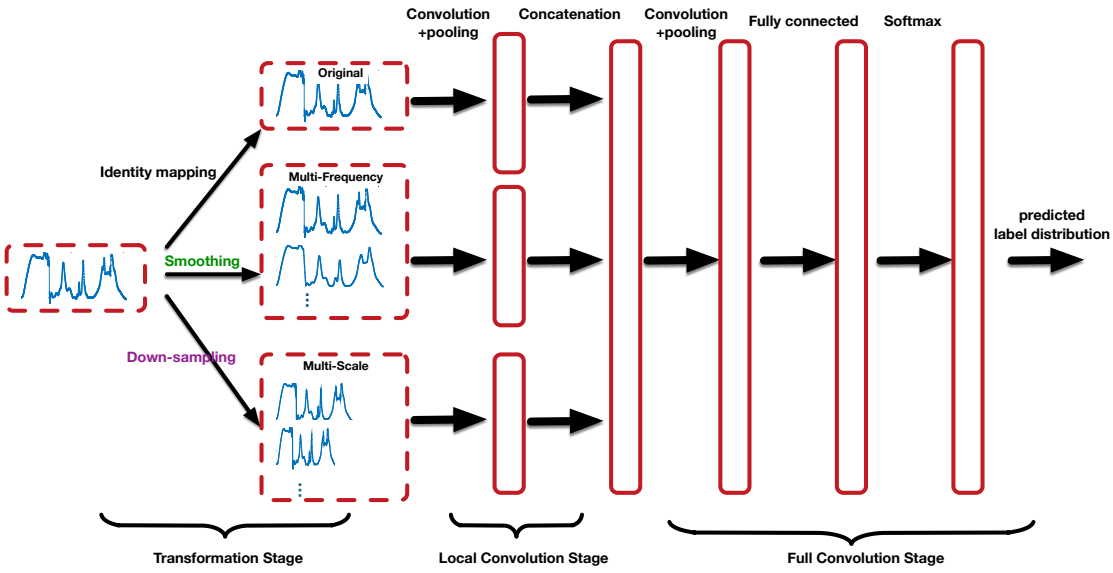

Figure 5.1: Overall architecture of MCNN.

In the multi-scale branch of MCNN, we use down-sampling to generate sketches of a time series at different time scales. Suppose we have a time series $T = \{t_1, t_2, ..., t_n\}$ and the down-sampling rate is k, then we will only keep every k^{th} data points in the new time series:

$$T^k = \{t_{1+k*i}\},\ i = 0, 1, ..., \lfloor \frac{n-1}{k} \rfloor. \tag{5.1}$$

Using this method, we generate multiple new input time series with different down sampling rates, e.g. $k = 2, 3, \cdots$.

Multi-frequency branch. In real-world applications, high-frequency perturbations and random noises widely exist in the time series data due to many reasons, which poses another challenge to achieving high prediction accuracy. It is often hard to extract useful information on raw time series data with the presence of these noises. In MCNN, we adopt low frequency filters with multiple degrees of smoothness to address this problem.

A low frequency filter can reduce the variance of time series. In particular, we employ moving average to achieve this goal. Given an input time series, we generate multiple new time series with varying degrees of smoothness using moving average with different window sizes. This way, newly generated time series represent general low frequency information, which make the trend of time series more clear. Suppose the original time series is $T = \{t_1, t_2, ..., t_n\}$, the moving average works by converting this original time series into a new time series

$$T^{\ell} = \frac{x_i + x_{i+1} + ... + x_{i+\ell-1}}{\ell}, \tag{5.2}$$

where ℓ is the window size and $i = 0, 1, ...n - \ell + 1$. With different ℓ, MCNN generates multiple time series of different frequencies, all of which will be fed into the local convolutional layer for this branch. Different from the multi-scale branch, each time series in the multi-frequency branch has the same length, which allows us to assemble them into multiple channels for the following convolutional layer.

5.3.4 Local convolution stage

Local convolution. After down sampling, we obtain multiple time series with different lengths from a single input time series. We apply independent 1-D local convolutions on each of these newly generated time series. In particular, the filter size of local convolution will be the same across all these time series. Note that, with a same filter size, shorter time series would get larger local receptive field in the original time series. This way, each output from the local convolution stage captures a different scale of the original time series. An advantage of this method is that, by down sampling the time series instead of increasing the filter size, we can greatly reduce the number of parameters in the local convolutional layer.

Max pooling with multiple sizes. Max pooling, a form of non-linear down-sampling, is also performed between successive convolutional layers in MCNN. This can reduce feature maps' size as well as the amount of following layers' parameters to avoid overfitting and improve computation efficiency. More importantly, the max pooling operation introduces invariance to spatial shifting, making MCNN more robust.

Instead of using small pooling sizes like 2 or 5, in MCNN we introduce a variable called the *pooling factor*, p, which is the length after max pooling. Suppose the output time series after convolution has a length of n, then both our pooling size and stride in max pooling are $\frac{n}{p}$. The pooling size is fairly large since p is often chosen from $\{2, 3, 5\}$. By doing this, we can have more filters and enforce each filter to learn only a local feature, since in the backpropogation phase, filters will be updated based on those few activated convolution parts.

5.3.5 Full convolution stage

After extracting feature maps from multiple branches, we concatenate all these features and feed them into other convolutional layers as well as a fully connected layer followed by a softmax transformation. Following [133] , we adopt the technique of *deep concatenation* to concatenate all the feature maps vertically.

The output of MCNN will be the predicted distribution of each possible label for the input time series. To train the neural network, MCNN uses the cross-entropy loss defined as:

$$\max_{\mathbb{W},\mathbf{b}} \sum_{i=1}^{N} \log o_{y_i}^{(i)}, \tag{5.3}$$

where $o_{y_i}^{(i)}$ is the y_i^{th} output of instance i through the neural network, which is the probability of its true label. The parameters $\mathbb{W}$ and bias $\mathbf{b}$ in MCNN are those in local and full convolutional layers, as well as those in the fully connected layers, all of which are learned jointly through back propagation.

5.3.6 Data augmentation

One advantage for our framework is the ability to deal with large scale datasets. When dealing with smaller datasets, convolutional nets tend to overfit. Currently, most publicly available TSC datasets have limited sizes. To overcome this problem, we propose a data augmentation technique on the original datasets in order to avoid overfitting and improve the generalization ability. For massive datasets with abundant training data, data augmentation may not be needed.

We propose *window slicing* for the data augmentation. For a time series $T = \{t_1, \cdots, t_n\}$, a slice is a snippet of the original time series, defined as $S_{i:j} = \{t_i, t_{i+1}, ..., t_j\}$, $1 \leq i \leq j \leq n$. Suppose a given time series T is of length n, and the length of the slice is s, our slicing operation will generate a set of n-s+1 sliced time series:

$$Slicing(T, s) = \{S_{1:s}, S_{2:s+1}, \cdots, S_{n-s+1:n}\}, \tag{5.4}$$

where all the time series in $Slicing(T, s)$ have the same label as their original time series T does.

We apply window slicing on all time series in a given training dataset. When doing training, all the training slices are considered independent training instances. We also do window slicing when predicting the label of a testing time series. We first use the trained MCNN to predict the label of each of its slices, and then use a majority vote among all these slices

to make the final prediction. Another advantage of slicing is that the time series are not required to have equal length since we can always cut all the time series into the same length using window slicing.

5.4 Discussion

In this section, we discuss several properties of the MCNN framework and its relations to some other important works.

5.4.1 Interpretability of convolution filters

Convolution has been a well-established method for handling sequential signals [95]. We advocate that it is also a good fit for capturing characteristics in time series. Suppose f is a filter of length m and T is a time series. Let $T \cdot f$ be the result of 1-dimensional discrete convolution. The i^{th} element of the result is given by

$$(T \cdot f)[i] = \sum_{j=1}^{m} f_{m+1-j} \cdot t_{i+j-1}$$

Depending on the filter, the convolution is capable of extracting many insightful information from the original time series. For example, if $f = [1, -1]$, the result of the convolution would be the gradient between any two neighboring points. However, is MCNN able to learn such kind of filters? The answer is yes. To show this, we train MCNN on a real-world dataset Gun_Point with a filter whose size is 15 ($m = 15$). For illustration, we pick one of the filters learned by MCNN as well as 3 time series from the dataset.

We show the shape of this selected filter and these 3 time series on the left of Figure 5.2, and the shape after convolution with the filter. Here, the two blue curves belong to one class and

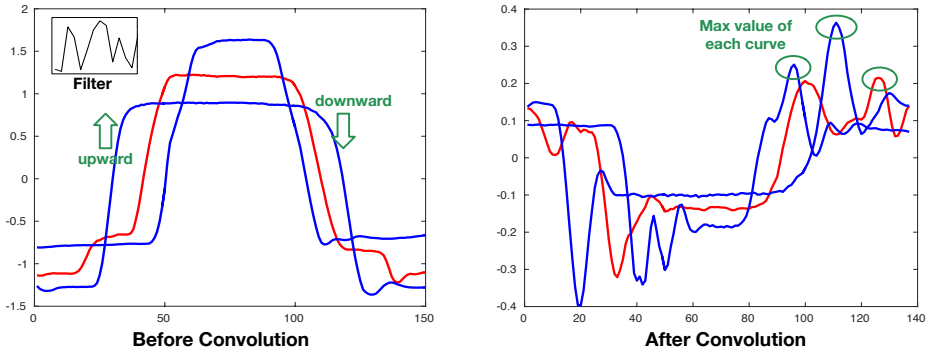

Figure 5.2: Three time Series on Gun_Point dataset before and after performing the convolution operation. The two blue curves belong to one class and the red curve belongs to a different class.

the red curve belongs to a different class. The learned filter (shown in the left figure) may look random at the first glance. However, a closer examination shows that it makes sense.

First, we can observe from the left figure that each time series has a upward part and a downward part no matter which label it has. After convolution with the filter, all three new signals form a valley at the location of the upward part and a peak at the location of the downward part. Second, since in MCNN we use max pooling right after convolution, the learned filter correctly finds that the downward part is more important, as max pooling only picks the maximum value from each convolved signal. As a result, the convolution and max pooling correctly differentiate the blue curves and red curve, since the maximum values of the two blue curves after convolution is greater than that of the red curve. By this visualization, MCNN also offers certain degree of interpretability as it tells us the characteristic found by MCNN. Third, these three time series have similar overall shapes but different time scales, as the “plateau" on the top have different lengths. It is very challenging for other methods such as DTW or shapelet to classify them. However, a single filter learned by MCNN coupled with max pooling can classify them well.

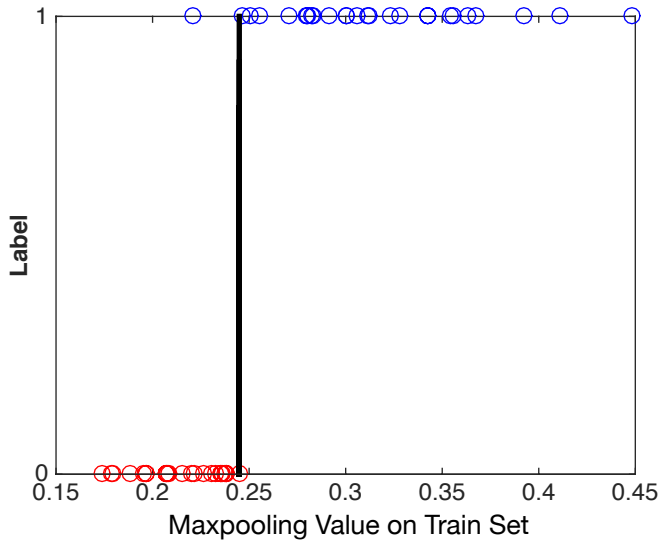

Figure 5.3: After training MCNN, we picked one filter and perform the convolution operation and max pooling on all training data.

To further demonstrate the power of convolution filters for TSC, we compute the max pooling result of all times series in the train set convolving with the filter shown in Figure 5.2, and show all of them in Figure 5.3. Here, each point corresponds to a time series in the dataset. The blue and red points correspond to two different classes, respectively. The x-axis is the max-pooling value of each point, and y-axis is the class label. We can see from Figure 5.3 that, if we set the classification threshold at around 0.25, one single convolution filter can already achieve very high accuracy to classify the dataset.

5.4.2 Relation to learning shapelets

A major class of TSC methods are based on shapelet analysis which assumes that time series are featured by some common subsequences. Shapelet can either be extracted from existing time series, or learned from the data. A recent study [56] proposes a learning time series shapelet (LTS) method which achieves unprecedented performance improvement over simple extraction. In the LTS method, each time series can be represented by a feature vector

in which each feature is the similarity between the time series and a shapelet. A logistic regression is applied on this new representation of time series to get the final prediction. Both the shapelets and parameters in the logistic regression model are jointly learned.

There is a strong relevance between the LTS method and MCNN, as both learn the parameters of the shapelets or filters jointly with a classifier. In fact, LTS can be viewed as a special case of MCNN. To make this more clear, let us first consider a simpler architecture, a special case of MCNN, where there is only one identity branch, and the input time series is processed by a 1-D convolutional layer followed by a softmax layer. The 1-D convolutional filter in the model can be regarded as a shapelet. The second layer (after convolution) is the new representation of the input time series. In this case, each neuron in the second layer is a inner product between the filter (or shapelet) and the corresponding window of the input time series. From this, we can see that MCNN model adopts inner product as the similarity measurement while LTS employs the Euclidean distance.

To further show the relationship between inner product in convolution and Euclidean distance, we can actually express the Euclidean distance in the form of convolution. Let $T \ominus f$ be the Euclidean distances between a time series $T = \{t_1, \cdots, t_n\}$ and a filter $f = \{f_1, \cdots, f_m\}$, its i^{th} element is:

$$
\begin{aligned}
(T \ominus f)[i] &= \sum_{j=1}^{m} \left(t_{i+j-1} - f_{m+1-j} \right)^2 \\
&= \sum_{j=1}^{m} t_{i+j-1}^2 + \sum_{j=1}^{m} f_{m+1-j}^2 \\
&\qquad - 2 \sum_{j=1}^{m} t_{i+j-1} f_{m+1-j} \\
&= \sum_{j=1}^{m} t_{i+j-1}^2 + \sum_{j=1}^{m} f_j^2 - 2(T \cdot f)[i] \qquad (5.5)
\end{aligned}
$$

From Eq. (5.5), the Euclidean distance is nothing but the combination of convolution $T \cdot f$ (after flipping the sign of f) and the ℓ_2 norms of f and a part of T. The first term in Eq. (5.5) is a constant for each time series, and therefore can be regarded as a bias which MCNN has incorporated in the model. We can thus see that learning shapelets is a special case of learning convolution filters when the filters are restricted to have the same ℓ_2 norm. Moreover, if we consider the full MCNN framework, its multi-scale and multi-frequency branches make it even more general to handle different time scales and noises.

Eq. (5.5) also gives us a hint on how to use convolution neural networks to implement Euclidean distances. By doing this, the Euclidean distance of between the time series and the shapelets can be efficiently computed leveraging deep learning packages and the speedups from their GPU implementation.

5.5 Experimental Results

In this section, we conduct extensive experiments on various benchmark datasets to evaluate MCNN and compare it against many leading TSC methods. We have made an effort to include the most recent works.

5.5.1 Experimental setup

We first describe the setup for our experiments.

Baseline methods. For comprehensive evaluation, we evaluate two classical baseline methods: 1-NN with Euclidean distance (ED) [46] and 1-NN DTW [13]. We also select 11 existing methods with state-of-the-art results published within the recent three years, including: DTW with a warping window constraint set through cross validation (DTW CV) [110], Fast Shapelet (FS) [111] , SAX with vector space model (SV) [122], Bag-of-SFA-Symbols (BOSS) [119], Shotgun Classifier (SC) [120], time series based on a bag-of-features (TSBF) [10], Elastic Ensemble (PROP) [88], 1-NN Bag-Of-SFA-Symbols in Vector Space (BOSSVS) [118], Learn Shapelets Model(LTS) [56], and the Shapelet Ensemble (SE) model [8].

We also test standard convolutional neural network with the same number of parameters as in MCNN to show the benefit of using the proposed multi-scale transformations and local convolution. For reference, we also list the results of **flat-COTE** (COTE), an ensemble model proposed by *Bagnall et al.* [8], which uses the weighted votes over 35 different classifiers. MCNN is orthogonal to flat-COTE and can be incorporated as a constituent classifier.

Datasets. We evaluate all methods thoroughly on the UCR time series classification archive [28], which consists of 46 datasets selected from various real-world domains. We omit Car and Plane because a large portion of baseline methods do not provide related results. All the datasets in the archive are publicly available[8]. Following the suggestions in [106], we *z-normalize* the following datasets during preprocessing: Beef, Coffee, Fish, OSULeaf and OliveOil.

[8] `http://www.cs.ucr.edu/~eamonn/time_series_data/`

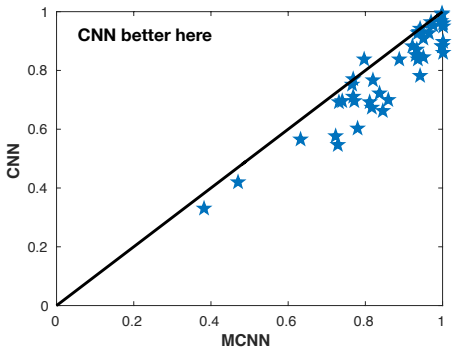

Figure 5.4: Scatter plot of test accuracies of standard CNN against MCNN on all 44 UCR datasets. MCNN is better in all but 3 datasets.

All the experiments use the default training and testing set splits provided by UCR, and the results are rounded to three decimal places. For authoritative comparison, we adopt the experimental results collected by Bagnall *et al.* [8] and Schafer [118] for the baseline methods.

Configuring MCNN. For MCNN, we conduct the experiments on all the datasets with the same network architecture as in Figure 1. Since most of the datasets in the UCR archive are not large enough, we first use window slicing to increasing the size of the training size. For window slicing, we set the length of slices to be $0.9n$ where n is the original length of the time series. We set the number of filters to be 256 for the convolutional layers and include 256 neurons in the fully connected layer.

We use mini-batch stochastic gradient with momentum to update parameters in MCNN. We adopt the *grid search* for hyper-parameter tuning based on cross validation. The hyper parameters MCNN include the filter size, pooling factor, and batch size.

In particular, the search space for the filter size is $\{0.05, 0.1, 0.2\}$, which denotes the ratio of the filter length to the original time series length; the search space for the pooling factor is $\{2, 3, 5\}$, which denotes the number of outputs of max-pooling. Early stopping is applied for preventing overfitting. Specifically, we use the error on the validation set to determine

the best model. When the validation error does not get reduced for a number of epochs, the training terminates.

MCNN is implemented based on theano [12] and run on NVIDIA GTX TITAN graphics cards with 2688 cores and 6 GB global memory. For full replicability of the experiments, we will release our code and make it available in public[9].

CNN vs. MCNN. Before comparing against other TSC classifiers, we first compare MCNN with standard CNN. We test a CNN that has the same architecture and number of parameters as our MCNN but does not have the multi-scale transformation and local convolutions. Figure 5.4 shows the scatter plot of the test accuracies of CNN and MCNN on the 44 datasets. We can see that MCNN achieves better results on 41 out of 44 datasets. A binomial test confirms that MCNN is significantly better than CNN at the 1% level.

5.5.2 Comprehensive evaluation

For each dataset, we rank all the 15 classifiers. MCNN is very competitive, achieving the highest accuracy on 10 datasets. MCNN has a mean rank of 3.95, lower than all the state-of-the-art methods except for COTE, which is an ensemble of 35 classifiers.

To further analyze the performance, we make pairwise comparison for each algorithm against MCNN. Binomial test (BT) and the Wilcoxon signed rank test (WSR) are used to measure the significance of difference. Corresponding p-values are listed in table 5.1, indicating that MCNN is significantly better than all the other methods except for BOSS and COTE at the 1% level ($p < 0.01$). Moreover, it shows that the differences between COTE, BOSS, and MCNN are not significant.

[9]Source codes of the programs developed by our lab are published at `http://www.cse.wustl.edu/~ychen/psd.htm`.

Table 5.1: Pairwise comparison with MCNN. p(BT) and p(WSR) are the p-values of binomial test and Wilcoxon signed rank test, respectively.

Model	#better	#tie	#worse	p(BT)	p(WSR)
DTW	5	2	37	9.43×10^{-7}	4.24×10^{-7}
ED	1	1	42	2.15×10^{-10}	1.54×10^{-8}
DTWCV	7	1	36	8.96×10^{-9}	3.24×10^{-6}
FS	1	0	43	5.12×10^{-12}	1.11×10^{-8}
SV	4	3	35	3.35×10^{-7}	2.48×10^{-6}
SC	5	2	37	4.43×10^{-7}	1.60×10^{-5}
TSBF	13	1	30	1.73×10^{-2}	1.60×10^{-3}
TSF	4	1	39	3.10×10^{-8}	5.08×10^{-7}
BOSSVS	8	5	31	2.94×10^{-4}	6.77×10^{-5}
SE	7	1	36	8.96×10^{-6}	8.90×10^{-5}
PROP	11	5	28	9.50×10^{-3}	1.20×10^{-2}
LTS	10	4	30	2.20×10^{-3}	1.60×10^{-2}
BOSS	16	4	24	2.68×10^{-1}	7.38×10^{-2}
COTE	22	2	20	8.78×10^{-1}	5.95×10^{-1}

Figure 5.5 shows the critical difference diagram, as proposed in [42]. The values shown on the figure are the average rank of each classifier. Bold lines indicate groups of classifiers which are not significantly different. The critical difference (CD) length is shown on the graph. Figure 5.5 is evaluated on MCNN, all baseline methods and COTE. MCNN is among the most accurate classifiers and its performance is very close to COTE. It is quite remarkable that MCNN, a single algorithm, obtains the same state-of-the-art performance as an ensemble model consisting of 35 different classifiers. Note that MCNN is orthogonal to flat-COTE as MCNN can also be included as a predictor in flat-COTE to further improve the performance. There are obvious margins between MCNN and other baseline classifiers.

We now group these classifiers into three categories and provide mode detailed analysis.

Distance based classifiers. These classifiers use nearest neighbor algorithms based on distance measures between time series. The simplest distance measure is the Euclidean distance (ED). Dynamic time warping (DTW) is proposed to extract the global similarity

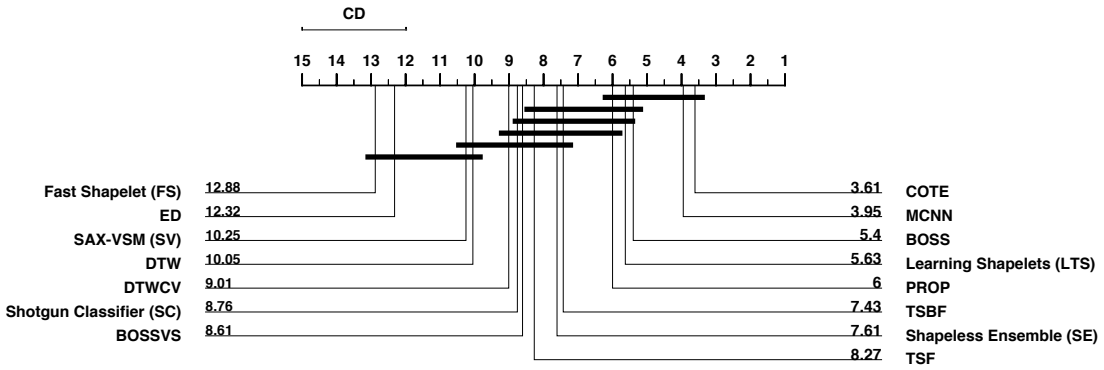

Figure 5.5: Critical Difference Diagram [42] over the mean ranks of MCNN, 13 baseline methods and the COTE ensemble. The critical difference is 3.01. COTE is the best classifier which ensembles 35 classifiers. MCNN performs equally well compared with COTE.

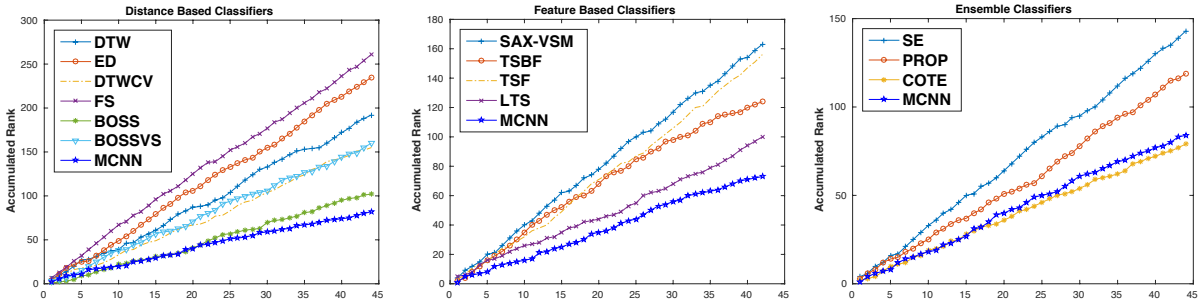

Figure 5.6: Comparison of MCNN against three groups of classifiers in terms of accumulated ranks.

while addressing the phase shift problem. DTW with 1-NN classifier has been hard to beat for a long time and now become a benchmark method. DTW with warping set through cross-validation (DTWCV) is also a traditional bench mark. k-NN classifiers also uses transformed features. Fast shapelet (FS) search shapelet on a lower transformed space. Bag-of-SFA-Symbols (BOSS) [119] proposes a distance based on histograms of symbolic Fourier approximation words. The BOSSVS model combines the BOSS model with the vector space model to reduce the time complexity. They are all combine with 1-NN for final prediction.

By grouping distance based classifiers together, we can compare their average performance with MCNN. In order to illustrate the overall performance of different algorithms, we plot accumulated rank on all the tested datasets in Figure 5.6. We order all the datasets alphabetically by their name and show the accumulated rank. For example, if a method is always ranked #1, its accumulated rank is N for the N^{th} dataset. From Figure 5.6, we see that MCNN has the lowest accumulated rank, outperforming all the distance based classifiers.

Feature based classifiers. For feature based classifiers, we selected SAX-VSM, TSF, TSBF, LTS. Symbolic aggregate approximation (SAX) has become a classical method to discretize time series based on piecewise mean value., SAX-VSM achieved state-of-the-art classification accuracy on UCR dataset by combining vector space Model (VSM) with SAX. Time series forest (TSF) divide time series into different intervals and calculate the mean, standard deviation and slop as interval features. Instead of using traditional entropy gain, TSF proposed a new split criteria by adding an addition term measuring the nearest distance between interval features and split threashold to the entropy and achieved better results than traditional random forests. The bag-of-features framework (TSBF) also extracts interval features with different scales. The features from each interval form an instance, and each time series forms a bag. Random forest is used to build a supervised codebook and classify time series. Finally, learning time series shapelets (LTS) provides not only competitive results, but also the ability to learn shapelets directly. Classification is made based on logistic regression.

The middle plot of figure 5.6 compares the performance of MCNN against some on feature based classifiers, including SV, TSBF, TSF, and LTS. It is clearly that MCNN is substantially

better than these feature based classifiers, as its accumulated rank is consistently the lowest by a large margin.

Ensemble based classifiers There is a growing trend in ensembling different classifiers together to achieve higher accuracy. The Elastic Ensemble (PROP) combined 11 distinct classifiers based on elastic distance measures through a weighted ensemble scheme. This was the first classifier that significantly outperformed DTW at that time. Shapelet ensemble (SE) combines shapelet transformations with a heterogeneous ensemble method. The weight of each classifier is assigned based on the cross validation accuracy. The flat collective of transform-based ensembles (flat-COTE) is an ensemble of 35 different classifiers based on features from time and frequency domains and has achieved state-of-the-art accuracy performance. Despite its high testing accuracy, ensemble methods suffer high complexity during the training process as well as testing. From the third plot in figure 5.6, we can observe that MCNN is very close to COTE and much better than SE and PROP. Critical difference analysis in Figure 5.5 also confirms that there is no significant difference between COTE and MCNN. It is in fact quite remarkable that a single algorithm in MCNN can match the performance of the COTE ensemble. The performance of MCNN is likely to improve further if it is trained with larger datasets, since convolutional neural networks are known to be able to absorb huge training data and make improvements.

5.6 Conclusions

We have presented MCNN, a convolutional neural network tailored for time series classification. MCNN unifies feature extraction and classification, and jointly learns the parameters through back propagation. It leverages the strength of CNN to automatically learn good feature representations in both time and frequency domains. In particular, MCNN contains multiple

branches that perform various transformations of the time series, which extract features of different frequency and time scales, addressing the limitation of many previous works that they only extract features at a single time scale. We have also discussed the insights that learning convolution filters in MCNN generalizes shapelet learning, which in part explains the excellent performance of MCNN.

We have conducted comprehensive experiments and compared with leading time series classification models. We have demonstrated that MCNN achieves state-of-the-art performance and outperforms many existing models by a large margin, especially when enough training data is present.

More importantly, an advantage of CNNs is that they can absorb massive amount of data to learn good feature representations. Currently, all the TSC datasets we have access to are not very large, ranging from a training size of around 50 to a few thousands. In the next chapter, we will combine MCNN and DNN together on a real-world clinical dataset.

Chapter 6

A Deep Learning Model for Predicting 30-Day Postoperative Mortality

6.1 Introduction

Most patients undergo elective surgery to cure a condition or improve quality of life; however, 0.5-3% of elective surgery patients do not survive until hospital discharge based on a survey over ten thousands patients [31]. How to reduce the rate of adverse outcomes therefore becomes a pressing public health problem.

Some of this mortality may be preventable through early identification of risk factors and better communication regarding risk mitigation. However, there are a number of reasons that anesthesiology teams fail to make accurate risk assessment in diagnosis and treatment. One major obstacle to optimal treatment for surgical patients in the operating rooms (OR) is information overload, beyond the capacity of practicing clinicians with the tools currently at disposal. To be more specific, clinicians are unable to accurately assess a patient's risk based

on pre-op data and rapidly evolving responses to general surgery and anesthesia. Furthermore, anesthesiologists are increasingly managing multiple surgical patients concurrently, and are expected to integrate numerous interacting patient characteristics with dynamic changes in a complex array of physiological parameters (e.g., electroencephalography waveforms, blood pressure, heart rate, temperature, fluid balance, metabolic homeostasis) in order to make informed patient-centered management decisions for each patient. Anesthesiologists, like all medical practitioners, have limitations in their cognitive and decision making abilities [41, 130, 131] . There is a limited amount of data that physicians can process per unit of time and the mismatch between human decision making ability and excess information is a source of medical errors [107]. In the realm of anesthesiology, the OR environment is a source of numerous stimuli that can at times overwhelm a single clinician. Less experienced practitioners are likely to have the most difficulty and require the most assistance in these circumstances. Cognitive biases commonly lead to medical mishaps, and cognitive errors specific to anesthesiology practice have been identified [132]. Further information overload through intrusive alarms or overwhelming IT-based decision support is likely to be unappreciated and even counter-productive. Given the known limits of human cognitive abilities and the high likelihood of cognitive biases in anesthesiology, there is therefore a pressing need for empowering and unobtrusive perioperative system that support critical decision-making without inundating the clinician with unfiltered and often irrelevant information.

Traditional data mining algorithms such as statistical models and logistic regression (LR) are used to analyze the relationship between the mortality and patient characteristics and factors, containing both non-modifiable and modifiable features. Non-modifiable factors include medical conditions (e.g., kidney disease, coronary artery disease), age, functional status, history of intraoperative awareness and duration and invasiveness of surgery. Other risk factors such as low intraoperative blood pressures and anesthetic concentrations are modifiable.

However, traditional data mining algorithms are far from satisfactory as they fall into the following shortcomings. 1). The performance of those models is not good enough for practical deployment in real world, limited by their model complexity. Take LR as an example, it is a linear parametric model, restricting its usage to linear separable datasets. However, there is no such guarantee on clinical datasets. 2). They cannot handle heterogeneous datasets well. Clinical datasets contains multi-type features and are usually collected from multi-sources. Statistical features such as age, gender and height are collected by nursers. Numerous time series (TS) are being collected automatically by modern intraoperative monitoring system from ORs every minute. Only first order statistics such as mean value and second order statistics such as standard deviation are used as model features.

With the success of deep learning in computer vision, Convolution Neural Network (CNN) on clinical data was mainly applied on clinical imaging, such as predicting the onset of Alzheimer's disease using the analysis of brain magnetic resonance imaging scans. Compared with clinical imaging, deep neural networks (DNNs) on large scale heterogeneous clinical datasets are not well studied. In this chapter, we proposed an interpretable machine learning framework that combines MCNN with deep neural network for 30-day mortality prediction [50].

6.2 Related Work

A number of forecasting algorithms exist that use medical data for outcomes prediction. These algorithms either rely on medical knowledge or general machine learning techniques.

A number of scoring systems that already exist use medical knowledge for various medical conditions. For example, the effectiveness of Several Community-Acquire Pneumonia (SCAP) and Pneumonia Severity Index (PSI) in predicting outcomes in patients with pneumonia is evaluated in [142]. Similarly, outcomes in patients with renal failures may be predicted

using the Acute Physiology Score (12 physiologic variables), Chronic Health Score (organ dysfunction) [77], APACHE (Acute Physiology, Age, Chronic Health Evaluation) score [76], and APACHE-III Score [78]. However, these algorithms are best for specialized hospital units for specific visits, especially for Intensive Care Unit(ICU). In contrast, the detection of clinical deterioration on general hospital units requires more general algorithms. For example, the Modified Early Warning Score (MEWS) [79] uses systolic blood pressure, pulse rate, temperature, respiratory rate, age and BMI to predict clinical deterioration. These physiological and demographic parameters may be collected at bedside, making MEWS suitable for a general hospital. These scoring systems have been widely used in hospital for even decades, so in practice it is imperative that the choice of the severity score scale, index, or model accurately match the event, setting or application, as mis-application of such systems can result in avoidable wastage of time, increase in cost incorrect extrapolations and may contribute to mismanagement and death [112]. Despite the existence of various kinds of scoring system, no score is specifically designed for usage in the operation rooms.

An alternative to algorithms that rely on medical knowledge is adapting statistical technique or standard machine learning techniques. This approach has two important advantages over traditional rule-based scoring algorithms. First, it allows us to consider a large number of parameters during prediction of patients' outcomes. Second, since they do not use a small set of rules to predict outcomes, it is possible to improve accuracy. Machine learning techniques such as decision trees [135], neural networks [67], and logistic regression [57] [60] have been used to identify clinical deterioration. In [145], integrate heterogeneous data (neuron-images, demographic, and genetic measures) is used for Alzheimer's disease (AD) prediction based on a kernel method. A support vector machine (SVM) classifier with radial basis kernels and an ensemble of templates are used to localize the tumor position in [33]. Also, in [67], a hyper-graph based learning algorithm is proposed to integrate micro array gene expressions

and protein-protein interactions for cancer outcome prediction and bio-marker identification. In recent years, with coming of big data in medical area [102], a lot of new algorithms have been developed taking advantage of the larger available data. [116] proposed an ensembling approach that combines multiple algorithms into a single algorithm and returns a prediction function with the best cross-validated mean squared error. In [134], the proposed local big data–driven, machine learning approach using random forest methods outperformed existing clinical decision rules and traditional analytic techniques for predicting in-hospital mortality of ED patients with sepsis.

There are a few distinguishing features of our approach comparing to previous work. Most previous work uses a snapshot method that takes all the features at a given time as input to a model, discarding the temporal evolving of data. There are some existing time-series classification method [51] [71] [149]. However, these methods assume a regular, constant gap between data records (e.g. one record every second) and are hard to handle large dataset. Our medical data, on the contrary, contains irregular gaps due to factors such as the workload of nurses. Existing work cannot handle such high-dimensional data with irregular, multi-scale gaps across different features. Yet another challenge is class imbalance: the data is severely skewed as there are much more normal patients than those with deterioration. To overcome those difficulty, we propose a deep learning method that allows us to incorporate both temporal data and static data at the same time, while handling the imbalance issue as well.

6.3 Data Descriptoin

This work is done in partnership with Barnes-Jewish Hospital (BJH). Our data is pulled from Washington University School of Medicine Institute of Quality Improvement, Research and Informatics (INQUIRI) perioperative databases including all preoperative, intraoperative

and postoperative data combined with other inpatient and outpatient EMR data. More than 110,000 surgeries' data is collected between 2012 and 2016, each of which contains 44 preoperative EMR features and 49 vital signs, represented as multi-variate TS. Thirty-day postoperative mortality is used as the output label. All data was initially recorded for clinical purposes.

Specifically, preoperative EMR data contains discrete data elements derived from a comprehensive history and physical examination. The information collected includes detailed comorbidity data, composite risk indices (American Society of Anesthesiologists' Physical Status and Charlson Comorbidity Index), home medications, laboratory results, special investigations, and targeted screening for quality of life, functional impairment, cognitive decline, history of falls, chronic pain and obstructive sleep apnea. The intraoperative medical record streams physiological data elements (e.g., anesthetic concentration, blood pressure, heart rate, temperature, oxygen saturation, exhaled carbon dioxide, ventilator parameters) continuously at 1-minute intervals, and collects data on fluids, drugs, blood transfusions and specific adverse events.

6.4 Data Preprocessing

In this section, we introduce the data representation and preprocessing methods that we apply to the dataset.

Our dataset can be represented as $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{t}_i), y_i\}_{i=1}^{N}$ with N instances, where $\mathbf{x}_i$ and y_i are D_f dimensional pre-op feature vector and the label of patient i, respectively. $\mathbf{t}_i$ is in-op multi-variate time series with the dimension of $N_t \times D_t$, where N_t is the number of vital signs we use and D_t is the length of each time series. Next, we perform data preprocessing on pre-op data and in-op data separately.

Table 6.1: Pre-op features

Feature Type	Feature Name
Numerical	height, weight, ideal body weight, body mass index, Charlson comorbidity index, functional capacity, left ventricular ejection fraction (LVEF), valvular disease, platelet, diastolic blood pressure, systolic blood pressure, American Society of Anesthesiologists (ASA) status
Categorical	sex, race, ASA emergency, hypertension, coronary artery, myocardial infarction, congestive heart failure, congestive heart failure diastolic function, atrial fibrillation, pacemaker, stroke, peripheral artery, deep vein thrombosis, pulmonary embolism, diabetes, outpatient Insulin, chronic kidney, dialysis history, pulmonary hypertension, chronic obstructive pulmonary, asthma, sleep apnea, cirrhosis, cancer history, gastroesophageal reflux, anemia, Coombs test, dementia, smoking, ulcer, anesthesia type, surgery type

6.4.1 Pre-op data preprocessing

Preoperative data are static data collected from patients before the operations. 12 numerical features and 32 categorical features are includes in the Pre-op data. Table 6.1 lists all the features. Pre-op data contains both numerical and categorical features. We preprocess the numerical features and categorical features in different ways.

Numerical feature preprocessing

We first preprocess the numerical features by removing the outliers. The raw numerical data typically contain many reading and input errors because information are recorded by nurses and there are inevitably errors introduced by manual operations. We list the acceptable

Table 6.2: Acceptable ranges of numerical features

Feature	Minimum	Maximum
height(cm)	120	225
weight(kg)	30	300
ideal body weight	45	105
body mass index	14	70
Charlson index	0	15
functional capacity	6	9
LVEF	31	38
valvular disease	61	65
platelet	0	750
diastolic	30	120
systolic	50	250
ASA status	1	5

ranges of every numerical feature in Table 6.2 based on the domain knowledge of the medical experts in our team. Then we perform a sanity check of the data and label the abnormal values that are outliers as missing value.

Second, not all patients have values for all signs in a real clinical data, and many types of clinical features involved in lab tests are not routinely performed on all patients. We use the medium value of a feature over the entire historical dataset to fill the missing values for most of the numerical features. For the feature "functional capacity", we use median instead as medical experts recommend.

Finally, we normalize the numerical features to scale the range of every feature in [0,1]. Such normalization is helpful for the forecasting algorithms.

Categorical feature preprocessing

We maintain a dictionary of all valid categories in all features. To eliminate the outliers as the first step, all values that do not exist in the dictionary are labeled as missing values.

Different from numerical data preprocessing, the missing categorical values are regarded as a new category, instead of being replaced. Our method can deal with categorical feature represented by ordinal number. For traditional algorithms, we apply One-hot encoding to transfer the categorical features to a sparse vector with one corresponding element being one and all others being zero, which usually dramatically increasing the number of dimensions. The encoded categorical data then can be combined with numerical features, and fed into the forecasting algorithms.

6.4.2 In-op data preprocessing

Intraoperative data are in the form of multi-variate time series of patients' vital signs and general signals monitored throughout patients' operations. Due to the inconsistent recording frequency of different time series and the large variance of their scales, we developed a sequence of useful techniques for preprocessing in-op data.

Data storing and querying

Because of the large number of patients and the huge length of time series of our data, it is too memory and time consuming to store all data in one file and load the entire file during training. Thus, we need to develop alternative ways for data storing and only query the data according to needs.

We chose to use the MySQL database for storing and querying data. The format of each row is "PatientID, ParameterID, ParameterName, Time, Value". An example row is as follows:

PatientID	ParameterID	ParameterName	Time	Value
338665	23	BJ N Heart Rate	2012-06-04 12:44:00	57

Note that each patient typically has thousands of rows describing the values of different variates at different timestamps. When we want to get a patient's in-op time series of some particular variates within some time range, we can thus query through MySQL by specifying "PatientID", "ParameterID", and "Time". Such a database solution enables querying data on the fly during training instead of loading all the data into memory, which is particularly suitable for storing large-scale time series dataset.

Time series interpolation

We use multi-channel convolutional neural network (CNN) and long short-term memory network (LSTM) to learn from multi-variate time series, where every channel corresponds to one variate. Analogous to image CNNs, we need to ensure that every channel of time series has the same length. However, different variates have different recording frequencies, which makes it necessary to do interpolation within some channels of time series. For example, the Heart Rate data are recorded every minute, while the Diastolic Blood Pressure data are recorded about every three to ten minutes – a 60-minute multi-variate time series containing both Heart Rate and Diastolic Blood Pressure will require interpolating Diastolic Blood Pressure so that it also has a length 60.

Considering the data volume, we use a simple linear interpolation (time-aware) for those time series with insufficient lengths. Because the minimum granularity is 1 minute for all variates, we interpolate those time series with insufficient lengths so that they all have granularity of 1 minute. If some time series have no records during the query time span, we fill zeros to the missing entries.

Multi-resolution standardization

Due to the large variances between the scales of different time series signals, proper standardization or normalization is required. We propose a multi-resolution standardization technique to standardize both the values within each time series of one patient, and the mean/std statistics of each time series across different patients.

Firstly, for each queried single-variate time series of a patient, we calculate its mean and standard deviation (std). Then within this time series, we standardize the values by $x = \frac{x-mean}{std}$ and return the new time series. This step is for making different single-variate time series all rescaled to 0 mean and 1 std, such that our model is not dominated by particular variates with large numerical scales and thus focusing on learning the shapes and variations.

However, we should not delete the mean and std statistics just calculated, since they indicate the general magnitude of the scale and variance of this particular variate of the patient. For example, two patients may have the same standardized Heart Rate time series, although patient A has a mean Heart Rate of 50 and patient B has a mean Heart Rate of 100. Simply deleting these statistics will lose important knowledge. To keep such information, we concatenate the mean and std statistics of a patient's different time series to this patient's pre-op data, in order to describe the patient's inherent bias w.r.t. these variates.

Since all other pre-op features have been normalized to between 0 and 1, we need to roughly keep these time series statistics to have the same scale too. Therefore, we additionally standardize each statistic by subtracting its mean and std (calculated across different patients). For example, assume the Heart Rate mean statistic for patient A is 80, and the mean and std of all patients' Heart Rate mean statistics are 60 and 10, respectively, then the standardized Heart Rate mean for patient A will be $\frac{80-60}{10} = 2$.

We call our method multi-resolution standardization, since we not only standardize the values within time series, but also standardize the high-level statistics of the time series across different patients.

6.4.3 Upsampling for imbalanced outcome

Finally, we discuss how we addressed the class imbalance problem. Our target outcome is 30-day mortality, which has a positive-negative ratio of approximately 1:100. We have tried two different methods to deal with this imbalance. The first method is to use class weights inversely proportional to their proportion to multiply the loss of positive training examples 100 times larger than that of negative training examples. The second method is to upsample positive training examples by 100 times each. We will report results of both two methods and discuss their advantages/disadvantages in the experiment section.

6.5 Method

In this work, we propose a multi-path deep neural network (MPCNN) framework, that can handle heterogeneous dataset directly. The overall architecture of MPCNN is depicted in Figure 6.1. MPCNN has three different paths: numerical feature mapping path, categorical feature embedding path and time series feature mapping path.

6.5.1 Numerical Feature Mapping Path

For the numerical part, each feature is doing an identity mapping to the next layer. No feature transformation is need before the concatenation of feature embeddings as deep neural networks can naturally handle this type of data.

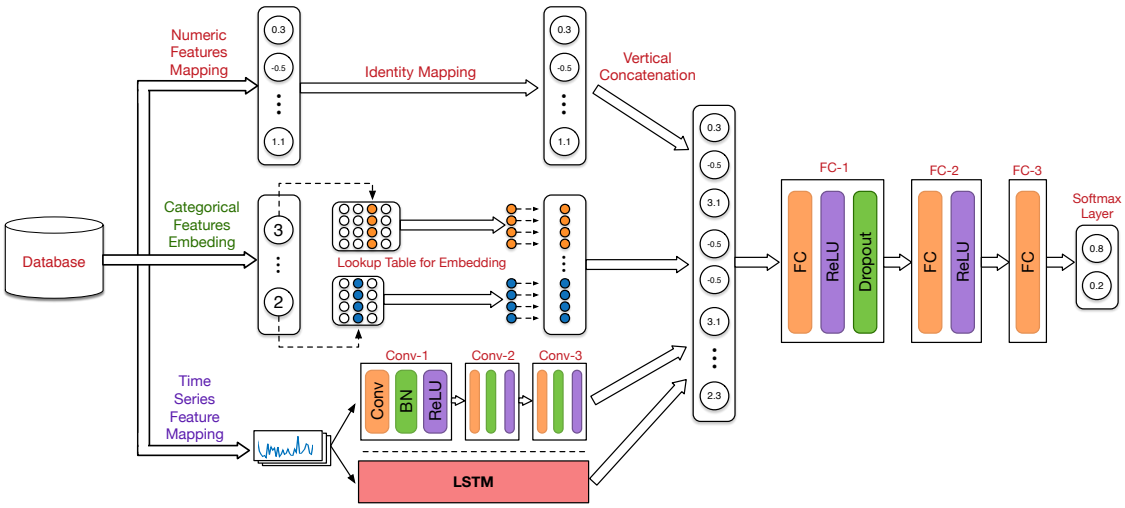

Figure 6.1: Overall architecture of MPCNN

6.5.2 Categorical Feature Embedding Path

The categorical feature embedding path is a lookup table that contains a numerical embedding for each category as shown in Figure 6.1. The number of embeddings of each feature is equal to the number of categories of the corresponding categorical feature. At its core, each lookup table is a matrix where each column vector represents a k dimensional embedding for a corresponding category. $k > 0$ is a user defined integer. For example, in Figure 6.1, we have $k = 4$. Each categorical feature would retrieve its corresponding embedding in the lookup table as its new feature representation.

6.5.3 Time Series Feature Mapping Path

Time Series (TS) has been studied for decades as the Time Series Classification (TSC) problem. TSC is a very challenging task as there are two key factors contributing to its difficulty. First, different time series may require feature representations at different time scales. For example, it is found that certain long-range (over a few hours involving hundreds

of time stamps) patterns in body temperature time series have predictive values in forecasting sepsis. Existing TSC features can rarely adapt to the right scales. Second, in real-world time series data, discriminative patterns in the time series can not be aligned to the same starting point.

Deep learning methods can automatically learn discriminative features from multi-variate time series. For the time series feature mapping path, we use two deep learning models that can exploit time series in different manner, 1) 1-dimension multi-variate convolution neural network (CNN). 2) Long Short-Term Memory (LSTM) network. These two models has the shift invariance and scale invariance to some extant and performs quite well in practice.

CNN 1-dimension multi-variate convolution neural network can extract information based on the shape of TS. Extensive experiments demonstrate the effectiveness of 1-D CNN on extracting discriminative features [34]. In this paper, we use a CNN with three convolutional blocks where each block consists a convolution layer (Conv), a batch normalization layer (BN) [68] and a nonlinear activation layer. Rectified linear unit (ReLU) [103] is used as activation function. The size of filters for each block are 2×128, 2×128, 2×64 where the first number refers kernel size and the second number refers to the number of output channels. The output of CNN are feed into concatenation layer.

LSTM Recurrent neural networks, specifically Long Short-Term Memory (LSTM) networks are designed to handle long-term sequential dependencies. As we use multi-variate time series, real-valued input vectors of vital signs from different sensors are feed into LSTM every minute. LSTM cell can memorize the previous state and make predictions together with current input value. In this paper, we adopt a 3-layer LSTM network to discover temporal sequence dependence. LSTM takes all the vital signs at each time frame as input to the

model and generate the patient's current health status sequentially. We concatenate all these outputs vertically and feed them into next layer.

6.5.4 Feature Concatenation and Classification

After extracting features from multiple data paths, we use deep concatenation technique [133] to concatenate all these features vertically. Then we feed those features into multiple fully connected layers (FC), each of which followed by a ReLU layer. Dropout [129] is also used after the first FC block to prevent overfitting. We feed the output of FC-3 to a softmax layer for prediction.

The output of MPCNN will be the predicted distribution of each possible outcomes for the input data. To train this deep neural network, MPCNN minimize the cross-entropy loss between the true outcome distribution and the prediction distribution.

6.5.5 Experimental Result

The target outcome (30-day mortality) had a positive-to-negative ratio of approximately 1:100. We tried two different methods to deal with this imbalance. The first method was to use class weights inversely proportional to their proportion, such that positive examples carried a weight 100 times greater than negative examples. The second method was to upsample positive training examples by 100 times. The upsampling method led to higher area under the precision-recall curve in the validation cohort as shown in Table 6.3, so this method was used in all analyses.

For intraoperative time series, data from a randomly selected epoch from each surgery were retrieved from the database. Epoch durations of 45 and 60 minutes were tested. The 60-minute epochs led to higher area under the receiver operating characteristic curve and

Table 6.3: Performance of the two methods for handling data imbalance in the multi-path convolutional neural network model (with long short-term memory handling of time series data). AUROC = Area under receiver-operating characteristic curve. AUPRC = Area under precision-recall curve.

Method	AUROC	AUPRC
Upsampling MPCNN	**0.867**	**0.097**
Cost-sensitive MPCNN	0.790	0.033

Table 6.4: Performance of multi-path convolutional neural network model (with long short-term memory handling of time series data) with time series of varying length. AUROC = Area under receiver-operating characteristic curve. AUPRC = Area under precision-recall curve.

TS Length	AUROC	AUPRC
0-min	0.824	0.078
45-min	0.855	0.085
60-min	**0.867**	**0.097**

precision-recall curve in the validation cohort as shown in Table 6.4, so this epoch length was used for all analyses. Simple linear interpolation was used to impute less-frequently measured time series to every minute. If a time series had no records during the query time span, then the patient mean value was used to fill in the missing entries. For intraoperative medications, the cumulative dose until the end of the epoch was used, normalized by the time since the beginning of surgery.

Then we plot out all the ROC curves in the left panel of Figure 6.2. MPCNN-LSTM achieves the highest AUROC of 0.867. Both of MPCNN-CNN and MPCNN-LSTM outperform all the baseline methods. As our dataset is extremely imbalance, the Precision-Recall curve shown in the right panel of Figure 6.2 contains more meaningful information. The positive-negative ratio is approximately 1:100, indicating that random guess can get only 0.01 AUPRC. MPCNN-LSTM achieves an AUPRC of 0.095, which is eight times better than random guess and nearly 12% gain compared with the LR model. Both of MPCNN-CNN and MPCNN-LSTM beat the baselines methods by a significantly large margin. Together with

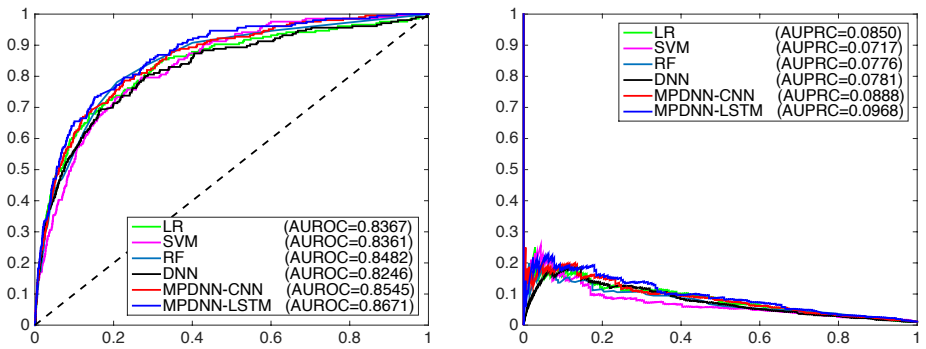

Figure 6.2: Performance characteristic curves for the multi-path convolutional neural network using convolution neural network (MPCNN-CNN), multi-path convolutional neural network using long short-term memory (MPCNN-LSTM), deep neural network (DNN), support vector machine (SVM), and logistic regression (LR). Panel A shows the receiver operating characteristic curves for each model. Panel B shows the precision-recall curves for each model

ROC and Precision-Recall curve, we can describe the property of forecasting models more accurately. One of the key results was that the addition of the time series feature mapping pathway improved the predictive ability of the model compared to models using only the mean and standard deviation of each intraoperative vital sign. This likely occurred because the MPCNN was able to identify patterns in vital signs, such as sudden drops in blood pressure or fluctuations in pulse oximeter readings. These patterns may identify acute intraoperative events, such as major blood loss or pulmonary congestion, that have important effects on postoperative outcomes.

When testing the MPCNN, we further visualize the case-wise feature importance using the backpropagation-based method. Given an input sample, this method calculates the gradient of each neuron with respect to the input space using backpropagation. Because the magnitude of the gradient reflects the degree to which input features affect the class score, it can be regarded as an indicator of feature importance. Figure 6.3 shows the importance of each mean arterial pressure value in the 60-minute epoch of a randomly selected patient.

Table 6.5: 30-day mortality prediction on ACTFAST dataset

Model	AUROC(95% CI)	AUPRC(95% CI)
LR	0.837 (0.803 to 0.871)	0.085 (0.074 to 0.096)
SVM	0.836 (0.802 to 0.870)	0.072 (0.062 to 0.081)
DNN	0.825 (0.790 to 0.856)	0.078 (0.068 to 0.088)
MPCNN-LSTM	0.867 (0.835 to 0.899)	0.095 (0.085 to 0.109)
MPCNN-CNN	0.855 (0.822 to 0.887)	0.089 (0.077 to 0.100)

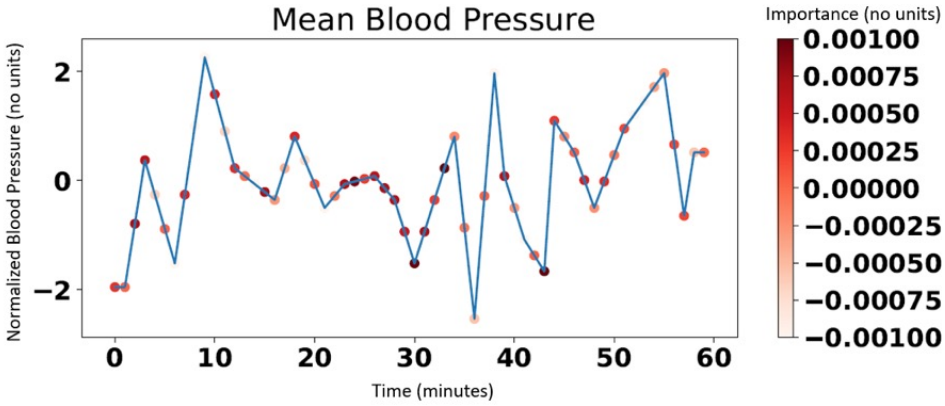

Figure 6.3: Importance of each mean blood pressure value in the 60-minute epoch of a randomly selected patient..

6.6 Conclusion

In this chapter, we have reported the development of a multi-path convolutional neural network model that outperforms support vector machine and logistic regression models in the prediction of postoperative thirty-day mortality. This model can be used intraoperatively to identify patients whose baseline characteristics and recent vital sign trends suggest an elevated risk for death. These patients may benefit from intraoperative interventions or additional resource allocation. Future investigations can explore whether prospective real-time use of this forecasting algorithm leads to accurate predictions, whether this model can generalize to other institutions, and whether incorporation of this model into a larger patient-monitoring programme leads to improved patient outcomes

Chapter 7

Conclusions

In this modern world, our daily lives are surround with machine learning models. We use our face to unlock cell phones. We use our voice assistant to wake up smart speakers. We use our movement to interact with kinect sensors. None of these could be done without machine learning models. However, there is an increasing need for interpretation when applying ML models to critical areas such as clinical decision support and market analysis. In this dissertation, we identify two demanding proprieties for model interpretability, i.e. accountability and actionability. We also present three interpretable machine learning methods.

Optimal action extraction algorithms is an instance level post-hoc methods that can extract actionable knowledge from additive tree models. In particular, we prove the NP-hardness of the optimal action extraction problem for ATMs and transform this problem into an integer linear programming formulation which can be efficiently solved by existing packages. We also empirically demonstrate the effectiveness of the proposed framework by conducting comprehensive experiments on challenging real-world datasets. However, the proposed method

involves solving an ILP problem. Its efficiency highly depends on the dimensionality of the dataset since the number of binary variables increases as the dimensionality gets higher. Therefore, our method is not suitable for datasets with very high dimensions such as text and images. Early stoping can be introduced when optimality is not necessary.

Deep Embedding Logistic Regression is a model level interpretable model that preserves model interpretability while equipping LR with non-linear separability. In DELR, each feature embedding is learned through a deep and narrow neural network and LR is attached to decide feature importance. A compact and yet powerful model, DELR offers great interpretability: it can tell the importance of each input feature, yield meaningful embedding of categorical features, and extract actionable changes, making it attractive for tasks such as market analysis and clinical prediction.

Factored general additive model is a model level interpretable model that allows feature interactions between static features and time-varying features. General additve model including DELR assign the same weight to the same feature no matter what the input data is, restricting their model performance in dealing with complex datasets. This assumption doesn't hold when dealing with clinical data. FGAM allows preoperative patient characteristics to modify their relationships between intraoperative features and their outcomes.

Last but not least, we propose an interpretable deep learning framework for 30-day mortality prediction. This model can handle heterogeneous data containing both static features and time series. We will develop a real-time early warning system using all proposed methods in this dissertation.

References

[1] Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. "Sanity checks for saliency maps." In: *Advances in Neural Information Processing Systems*. 2018, pp. 9505–9515.

[2] Guillaume Alain and Yoshua Bengio. "Understanding intermediate layers using linear classifier probes." In: *arXiv preprint arXiv:1610.01644* (2016).

[3] André Altmann, Laura Toloşi, Oliver Sander, and Thomas Lengauer. "Permutation importance: a corrected feature importance measure." In: *Bioinformatics* 26.10 (2010), pp. 1340–1347.

[4] David Alvarez-Melis and Tommi S Jaakkola. "Towards robust interpretability with self-explaining neural networks." In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Curran Associates Inc. 2018, pp. 7786–7795.

[5] Itamar Arel, Derek C Rose, and Thomas P Karnowski. "Deep machine learning-a new frontier in artificial intelligence research [research frontier]." In: *Computational Intelligence Magazine, IEEE* 5.4 (2010), pp. 13–18.

[6] Ahsan M Arozullah, Jennifer Daley, William G Henderson, Shukri F Khuri, National Veterans Administration Surgical Quality Improvement Program, et al. "Multifactorial risk index for predicting postoperative respiratory failure in men after major noncardiac surgery." In: *Annals of surgery* 232.2 (2000), p. 242.

[7] Jimmy Ba and Rich Caruana. "Do deep nets really need to be deep?" In: *Advances in neural information processing systems*. 2014, pp. 2654–2662.

[8] Anthony Bagnall, Jason Lines, Jon Hills, and Aaron Bostrom. "Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles." In: ().

[9] Thomas C Bailey et al. "A trial of a real-time Alert for clinical deterioration in Patients hospitalized on general medical wards." In: *Journal of hospital medicine* 8.5 (2013), pp. 236–242.

[10] Mustafa Gokce Baydogan, George Runger, and Eugene Tuv. "A bag-of-features framework to classify time series." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 35.11 (2013), pp. 2796–2802.

[11] Yoshua Bengio. "Learning deep architectures for AI." In: *Foundations and trends® in Machine Learning* 2.1 (2009), pp. 1–127.

[12] James Bergstra et al. "Theano: a CPU and GPU math expression compiler." In: *Proceedings of the Python for scientific computing conference (SciPy)*. Vol. 4. Austin, TX. 2010, p. 3.

[13] Donald J Berndt and James Clifford. "Using Dynamic Time Warping to Find Patterns in Time Series." In: *KDD workshop*. Vol. 10. 16. Seattle, WA. 1994, pp. 359–370.

[14] Kendrick Boyd, Vitor Santos Costa, Jesse Davis, and C David Page. "Unachievable region in precision-recall space and its effect on empirical evaluation." In: *Proceedings of the... International Conference on Machine Learning. International Conference on Machine Learning*. Vol. 2012. NIH Public Access. 2012, p. 349.

[15] L. Breiman. "Bagging predictors." In: *Machine Learning* 26 (1996), pp. 123–140.

[16] L. Breiman. "Random forests." In: *Machine Learning* 45 (2001), pp. 5–32.

[17] William Caicedo-Torres and Jairo Gutierrez. "ISeeU: Visually interpretable deep learning for mortality prediction inside the ICU." In: *arXiv preprint arXiv:1901.08201* (2019).

[18] L. Cao, D. Luo, and C. Zhang. "Knowledge Actionability: Satisfying Technical and Business Interestingness." In: *Int. J. Bus. Intell. Data Min.* 2.4 (Dec. 2007), pp. 496–514.

[19] L. Cao, Y. Zhao, H. Zhang, D. Luo, and C. Zhang nad E. K. Park. "Flexible Frameworks for Actionable Knowledge Discovery." In: *IEEE Trans. Knowledge and Data Engineering* 22.9 (2009), pp. 1299–1312.

[20] L. Cao et al. "Data driven actionable knowledge discovery." In: *IEEE Intelligent Systems* 22.4 (2007), pp. 77–88.

[21] William Chan, Navdeep Jaitly, Quoc Le, and Oriol Vinyals. "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition." In: *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016, pp. 4960–4964.

[22] Zhengping Che, Sanjay Purushotham, Robinder Khemani, and Yan Liu. "Interpretable deep models for icu outcome prediction." In: *AMIA Annual Symposium Proceedings*. Vol. 2016. American Medical Informatics Association. 2016, p. 371.

[23] Hsinchun Chen, Roger HL Chiang, and Veda C Storey. "Business intelligence and analytics: from big data to big impact." In: *MIS quarterly* (2012), pp. 1165–1188.

[24] Tianqi Chen. "Introduction to boosted trees." In: *University of Washington Computer Science* (2014).

[25] Wenlin Chen, Yixin Chen, Yi Mao, and Baolong Guo. "Density-based logistic regression." In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 140–148.

[26] Wenlin Chen, Yixin Chen, and Kilian Weinberger. "Filtered search for submodular maximization with controllable approximation bounds." In: *Artificial Intelligence and Statistics*. 2015, pp. 156–164.

[27] Wenlin Chen, Yixin Chen, and Kilian Q Weinberger. "Fast flux discriminant for large-scale sparse nonlinear classification." In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 621–630.

[28] Yanping Chen, Eamonn Keogh, Bing Hu, Nurjahan Begum, Anthony Bagnall, Abdullah Mueen, and Gustavo Batista. *The UCR Time Series Classification Archive*. `www.cs.ucr.edu/~eamonn/time_series_data/`. July 2015.

[29] Glenn M Chertow, J Michael Lazarus, Cindy L Christiansen, E Francis Cook, Karl E Hammermeister, Frederick Grover, and Jennifer Daley. "Preoperative renal risk stratification." In: *Circulation* 95.4 (1997), pp. 878–884.

[30] Edward Choi, Mohammad Taha Bahadori, Jimeng Sun, Joshua Kulas, Andy Schuetz, and Walter Stewart. "Retain: An interpretable predictive model for healthcare using reverse time attention mechanism." In: *Advances in Neural Information Processing Systems*. 2016, pp. 3504–3512.

[31] GlobalSurg Collaborative. "Mortality of emergency abdominal surgery in high-, middle- and low-income countries." In: *Br J Surg* 103.8 (2016), pp. 971–88.

[32] IBM ILOG CPLEX. "V12. 1: User's Manual for CPLEX." In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[33] Ying Cui, Jennifer G Dy, Gregory C Sharp, Brian M Alexander, and Steve B Jiang. "Learning methods for lung tumor markerless gating in image-guided radiotherapy." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 902–910.

[34] Zhicheng Cui, Wenlin Chen, and Yixin Chen. "Multi-scale convolutional neural networks for time series classification." In: *arXiv preprint arXiv:1603.06995* (2016).

[35] Zhicheng Cui, Wenlin Chen, Yujie He, and Yixin Chen. "Optimal action extraction for random forests and boosted trees." In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM. 2015, pp. 179–188.

[36] Zhicheng Cui, Bradley A Fritz, Christopher R King, Michael S Avidan, and Yixin Chen. "A Factored Generalized Additive Model for Clinical Decision Support in the Operating Room." In: *arXiv preprint arXiv:1907.12596* (2019).

[37] Zhicheng Cui, Muhan Zhang, and Yixin Chen. "Deep embedding logistic regression." In: *International Conference on Big Knowledge*. Vol. 00. Nov. 2019, pp. 176–183. DOI: 10.1109/ICBK.2018.00031.

[38] Mladen Dalto. "Deep neural networks for time series prediction with applications in ultra-short-term wind forecasting." In: *Rn (Θ1)* 1 (), p. 2.

[39] Jeffrey Dastin. *Amazon scraps secret AI recruiting tool that showed bias against women*. https://www.reuters.com/article/us-amazon-com-jobs-automation-insight/amazon-scraps-secret-ai-recruiting-tool-that-showed-bias-against-women-idUSKCN1MK08G. 2018.

[40] Jesse Davis and Mark Goadrich. "The relationship between Precision-Recall and ROC curves." In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 233–240.

[41] Neal V Dawson and Hal R Arkes. "Systematic errors in medical decision making." In: *Journal of General Internal Medicine* 2.3 (1987), pp. 183–187.

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

[43] Yanzhuo Ding, Yang Liu, Huanbo Luan, and Maosong Sun. "Visualizing and understanding neural machine translation." In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017, pp. 1150–1159.

[44] Finale Doshi-Velez and Been Kim. "Towards a rigorous science of interpretable machine learning." In: *arXiv preprint arXiv:1702.08608* (2017).

[45] A. Drewry, B. Fuller, T. Bailey, and R. Hotchkiss. "Body temperature patterns as a predictor of hospital-acquired sepsis in afebrile adult intensive care unit patients: a case-control study." In: *Critical Care* 17.5 (2013), doi: 10.1186/cc12894.

[46] Christos Faloutsos, Mudumbai Ranganathan, and Yannis Manolopoulos. *Fast subsequence matching in time-series databases*. Vol. 23. 2. ACM, 1994.

[47] Y. Freund and R. E. Schapire. "A decision-theoretic generalization of online learning and an application to boosting." In: *Journal of Computer and System Sciences* 55 (1997), pp. 119–139.

[48] J. H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29 (), pp. 1189–1232.

[49] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, NY, USA: 2001.

[50] Bradley A Fritz et al. "Deep-learning model for predicting 30-day postoperative mortality." In: *British Journal of Anaesthesia* (2019).

[51] Ben D Fulcher and Nick S Jones. "Highly comparative feature-based time-series classification." In: *Knowledge and Data Engineering, IEEE Transactions on* 26.12 (2014), pp. 3026–3037.

[52] Jacob R Gardner, Paul Upchurch, Matt J Kusner, Yixuan Li, Kilian Q Weinberger, Kavita Bala, and John E Hopcroft. "Deep manifold traversal: Changing labels with convolutional features." In: *arXiv preprint arXiv:1511.06421* (2015).

[53] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. New York, NY, USA: W. H. Freeman & Co., 1990.

[54] Wendong Ge, Jin-Won Huh, Yu Rang Park, Jae-Ho Lee, Young-Hak Kim, and Alexander Turchin. "An Interpretable ICU Mortality Prediction Model Based on Logistic Regression and Recurrent Neural Networks with LSTM units." In: *AMIA Annual Symposium Proceedings*. Vol. 2018. American Medical Informatics Association. 2018, p. 460.

[55] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." In: *arXiv preprint arXiv:1412.6572* (2014).

[56] Josif Grabocka, Nicolas Schilling, Martin Wistuba, and Lars Schmidt-Thieme. "Learning time-series shapelets." In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 392–401.

[57] M Pamela Griffin and J Randall Moorman. "Toward the early diagnosis of neonatal sepsis and sepsis-like illness using novel heart rate analysis." In: *Pediatrics* 107.1 (2001), pp. 97–104.

[58] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. "A survey of methods for explaining black box models." In: *ACM computing surveys (CSUR)* 51.5 (2018), p. 93.

[59] Himani Gupta, Prateek K Gupta, Xiang Fang, Weldon J Miller, Samuel Cemaj, R Armour Forse, and Lee E Morrow. "Development and validation of a risk calculator predicting postoperative respiratory failure." In: *Chest* 140.5 (2011), pp. 1207–1215.

[60] Gregory Hackmann, Minmin Chen, Octav Chipara, Chenyang Lu, Yixin Chen, Marin Kollef, and Thomas C Bailey. "Toward a two-tier clinical warning system for hospitalized patients." In: *AMIA Annual Symposium proceedings*. Vol. 2011. American Medical Informatics Association. 2011, p. 511.

[61] James A Hanley and Barbara J McNeil. "The meaning and use of the area under a receiver operating characteristic (ROC) curve." In: *Radiology* 143.1 (1982), pp. 29–36.

[62] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "The Elements of Statistical Learning." In: Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001, pp. 361–364.

[63] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[64] R. J. Hilderman and H. J. Hamilton. "Applying Objective Interestingness Measures in Data Mining Systems." In: *Proceedings of the 4th European Symposium on Principles of Data Mining and Knowledge Discovery*. 2000, pp. 432–439.

[65] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets." In: *Neural computation* 18.7 (2006), pp. 1527–1554.

[66] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. "Densely connected convolutional networks." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. Vol. 1. 2. 2017, p. 3.

[67] TaeHyun Hwang, Ze Tian, Rui Kuangy, and Jean-Pierre Kocher. "Learning on weighted hypergraphs to integrate protein interactions and gene expressions for cancer outcome prediction." In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE. 2008, pp. 293–302.

[68] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." In: *arXiv preprint arXiv:1502.03167* (2015).

[69] Robert G Johnson, Ahsan M Arozullah, Leigh Neumayer, William G Henderson, Patrick Hosokawa, and Shukri F Khuri. "Multivariable predictors of postoperative respiratory failure after general and vascular surgery: results from the patient safety in surgery study." In: *Journal of the American College of Surgeons* 204.6 (2007), pp. 1188–1198.

[70] M. Karim and R. Rahman. "Decision Tree and Naive Bayes Algorithm for Classification and Generation of Actionable Knowledge for Direct Marketing." In: *Journal of Software Engineering and Applications* 6 (2013), pp. 196–206.

[71] Rohit J Kate. "Using dynamic time warping distances as features for improved time series classification." In: *Data Mining and Knowledge Discovery* 30.2 (2016), pp. 283–312.

[72] Rohit J Kate, Ruth M Perez, Debesh Mazumdar, Kalyan S Pasupathy, and Vani Nilakantan. "Prediction and detection models for acute kidney injury in hospitalized older adults." In: *BMC medical informatics and decision making* 16.1 (2016), p. 39.

[73] John A Kellum and Norbert Lameire. "Diagnosis, evaluation, and management of acute kidney injury: a KDIGO summary (Part 1)." In: *Critical care* 17.1 (2013), p. 204.

[74] Sachin Kheterpal, Kevin K Tremper, Michael Heung, Andrew L Rosenberg, Michael Englesbe, Amy M Shanks, and Darrell A Campbell. "Development and validation of an acute kidney injury risk index for patients undergoing general surgeryresults from a national data set." In: *Anesthesiology: The Journal of the American Society of Anesthesiologists* 110.3 (2009), pp. 505–515.

[75] Jinkyu Kim and John Canny. "Interpretable learning for self-driving cars by visualizing causal attention." In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2942–2950.

[76] William A Knaus, Elizabeth A Draper, Douglas P Wagner, and Jack E Zimmerman. "APACHE II: a severity of disease classification system." In: *Critical care medicine* 13.10 (1985), pp. 818–829.

[77] William A Knaus, Jack E Zimmerman, Douglas P Wagner, Elizabeth A Draper, and Diane E Lawrence. "APACHE-acute physiology and chronic health evaluation: a physiologically based classification system." In: *Critical care medicine* 9.8 (1981), pp. 591–597.

[78] William A Knaus et al. "The APACHE III prognostic system: risk prediction of hospital mortality for critically III hospitalized adults." In: *Chest* 100.6 (1991), pp. 1619–1636.

[79] JeongGil Ko et al. "MEDiSN: Medical emergency detection in sensor networks." In: *ACM Transactions on Embedded Computing Systems (TECS)* 10.1 (2010), p. 11.

[80] Pang Wei Koh and Percy Liang. "Understanding black-box predictions via influence functions." In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org. 2017, pp. 1885–1894.

[81] Jay L Koyner, Kyle A Carey, Dana P Edelson, and Matthew M Churpek. "The development of a machine learning inpatient acute kidney injury prediction model." In: *Critical care medicine* 46.7 (2018), pp. 1070–1077.

[82] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks." In: *NIPS*. 2012.

[83] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world." In: *arXiv preprint arXiv:1607.02533* (2016).

[84] H. Lee, P. Pham, Y. Largman, and A. Y. Ng. "Unsupervised feature learning for audio classification using convolutional deep belief networks." In: *NIPS*. 2009.

[85] Roger Levy. "Probabilistic models in the study of language." In: *NA, NA* (2012).

[86] Jia Li, Changyong Niu, and Ming Fan. "Multi-scale convolutional neural networks for natural scene license plate detection." In: *Advances in Neural Networks–ISNN 2012*. Springer, 2012, pp. 110–119.

[87] Andy Liaw and Matthew Wiener. "Classification and Regression by randomForest." In: *R News* 2.3 (2002), pp. 18–22.

[88] Jason Lines and Anthony Bagnall. "Time series classification with ensembles of elastic distance measures." In: *Data Mining and Knowledge Discovery* 29.3 (2015), pp. 565–592.

[89] Jason Lines, Luke M Davis, Jon Hills, and Anthony Bagnall. "A shapelet transform for time series classification." In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 289–297.

[90] Zachary C Lipton. "The mythos of model interpretability." In: *arXiv preprint arXiv:1606.03490* (2016).

[91] B. Liu and W. Hsu. "Post-Analysis of Learned Rules." In: *Proc. AAAI*. 1996, pp. 828–834.

[92] B. Liu, W. Hsu, and Y. Ma. "Pruning and Summarizing the Discovered Associations." In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '99. 1999, pp. 125–134.

[93] Bing Liu and Wynne Hsu. "Post-analysis of learned rules." In: *AAAI/IAAI, Vol. 1*. 1996, pp. 828–834.

[94] Bing Liu, Wynne Hsu, and Yiming Ma. "Pruning and summarizing the discovered associations." In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 1999, pp. 125–134.

[95] Stéphane Mallat. *A wavelet tour of signal processing*. Academic press, 1999.

[96] Yi Mao, Wenlin Chen, Yixin Chen, Chenyang Lu, Marin Kollef, and Thomas Bailey. "An integrated data mining approach to real-time clinical monitoring and deterioration warning." In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 1140–1148.

[97] Hector P Martinez, Yoshua Bengio, and Georgios N Yannakakis. "Learning deep physiological models of affect." In: *Computational Intelligence Magazine, IEEE* 8.2 (2013), pp. 20–33.

[98] Tim Miller. "Explanation in artificial intelligence: Insights from the social sciences." In: *Artificial Intelligence* (2018).

[99] Ananth Mohan, Zheng Chen, and Kilian Weinberger. "Web-search ranking with initialized gradient boosted regression trees." In: *Proceedings of the learning to rank challenge*. 2011, pp. 77–89.

[100] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. `https://christophm.github.io/interpretable-ml-book/`. 2019.

[101] Grégoire Montavon, Wojciech Samek, and Klaus-Robert Müller. "Methods for interpreting and understanding deep neural networks." In: *Digital Signal Processing* 73 (2018), pp. 1–15.

[102] Travis B Murdoch and Allan S Detsky. "The inevitable application of big data to health care." In: *Jama* 309.13 (2013), pp. 1351–1352.

[103] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines." In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.

[104] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks." In: *arXiv preprint arXiv:1602.03616* (2016).

[105] H Palomba, I De Castro, ALC Neto, S Lage, and L Yu. "Acute kidney injury prediction following elective cardiac surgery: AKICS Score." In: *Kidney international* 72.5 (2007), pp. 624–631.

[106] John Paparrizos and Luis Gravano. "k-shape: Efficient and accurate clustering of time series." In: *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM. 2015, pp. 1855–1870.

[107] Vimla L Patel, David R Kaufman, and Jose F Arocha. "Emerging paradigms of cognition in medical decision-making." In: *Journal of biomedical informatics* 35.1 (2002), pp. 52–75.

[108] William JE Potts. "Generalized additive neural networks." In: *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 1999, pp. 194–200.

[109] Sivaramakrishnan Rajaraman, Sema Candemir, Incheol Kim, George Thoma, and Sameer Antani. "Visualization and interpretation of convolutional neural network predictions in detecting pneumonia in pediatric chest radiographs." In: *Applied Sciences* 8.10 (2018), p. 1715.

[110] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. "Searching and mining trillions of time series subsequences under dynamic time warping." In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2012, pp. 262–270.

[111] Thanawin Rakthanmanon and Eamonn Keogh. "Fast shapelets: A scalable algorithm for discovering time series shapelets." In: *Proceedings of the thirteenth SIAM conference on data mining (SDM)*. SIAM. 2013, pp. 668–676.

[112] Amy Grace Rapsang and Devajit C Shyam. "Scoring systems in the intensive care unit: A compendium." In: *Indian journal of critical care medicine: peer-reviewed, official publication of Indian Society of Critical Care Medicine* 18.4 (2014), p. 220.

[113] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster r-cnn: Towards real-time object detection with region proposal networks." In: *Advances in neural information processing systems*. 2015, pp. 91–99.

[114] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?: Explaining the predictions of any classifier." In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. ACM. 2016, pp. 1135–1144.

[115] Tim Rocktäschel, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočisk, and Phil Blunsom. "Reasoning about entailment with neural attention." In: *arXiv preprint arXiv:1509.06664* (2015).

[116] Sherri Rose. "Mortality risk score prediction in an elderly population using machine learning." In: *American journal of epidemiology* 177.5 (2013), pp. 443–452.

[117] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. "Dynamic routing between capsules." In: *Advances in Neural Information Processing Systems*. 2017, pp. 3856–3866.

[118] Patrick Schäfer. "Scalable time series classification." In: *Data Mining and Knowledge Discovery* (2015), pp. 1–26.

[119] Patrick Schäfer. "The BOSS is concerned with time series classification in the presence of noise." In: *Data Mining and Knowledge Discovery* 29.6 (2015), pp. 1505–1530.

[120] Patrick Schäfer. "Towards time series classification without human preprocessing." In: *Machine Learning and Data Mining in Pattern Recognition*. Springer, 2014, pp. 228–242.

[121] F. Schroff, D. Kalenichenko, and J. Philbin. "Facenet: A unified embedding for face recognition and clustering." In: *arXiv preprint arXiv:1503.03832*. 2015.

[122] Pavel Senin and Sergey Malinchik. "Sax-vsm: Interpretable time series classification using sax and vector space model." In: *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE. 2013, pp. 1175–1180.

[123] Sungyong Seo, Jing Huang, Hao Yang, and Yan Liu. "Interpretable convolutional neural networks with dual local and global attention for review rating prediction." In: *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM. 2017, pp. 297–305.

[124] Ying Sha and May D Wang. "Interpretable predictions of clinical outcomes with an attention-based recurrent neural network." In: *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. ACM. 2017, pp. 233–240.

[125] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. "Real-time human pose recognition in parts from single depth images." In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. Ieee. 2011, pp. 1297–1304.

[126] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." In: *arXiv preprint arXiv:1312.6034* (2013).

[127] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical bayesian optimization of machine learning algorithms." In: *Advances in neural information processing systems*. 2012, pp. 2951–2959.

[128] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. "Striving for simplicity: The all convolutional net." In: *arXiv preprint arXiv:1412.6806* (2014).

[129] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[130] Marjorie P Stiegler and Keith J Ruskin. "Decision-making and safety in anesthesiology." In: *Current Opinion in Anesthesiology* 25.6 (2012), pp. 724–729.

[131] Marjorie Podraza Stiegler and Avery Tung. "Cognitive processes in anesthesiology decision making." In: *Anesthesiology: The Journal of the American Society of Anesthesiologists* 120.1 (2014), pp. 204–217.

[132] MP Stiegler, JP Neelankavil, C Canales, and A Dhillon. "Cognitive errors detected in anaesthesiology: a literature review and pilot study." In: *British journal of anaesthesia* 108.2 (2012), pp. 229–235.

[133] Christian Szegedy et al. "Going deeper with convolutions." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1–9.

[134] R Andrew Taylor, Joseph R Pare, Arjun K Venkatesh, Hani Mowafi, Edward R Melnick, William Fleischman, and M Kennedy Hall. "Prediction of In-hospital Mortality in Emergency Department Patients With Sepsis: A Local Big Data–Driven, Machine Learning Approach." In: *Academic emergency medicine* 23.3 (2016), pp. 269–278.

[135] Steven W Thiel, Jamie M Rosini, William Shannon, Joshua A Doherty, Scott T Micek, and Marin H Kollef. "Early prediction of septic shock in hospitalized patients." In: *Journal of hospital medicine* 5.1 (2010), pp. 19–25.

[136] Paul Thottakkara, Tezcan Ozrazgat-Baslanti, Bradley B Hupf, Parisa Rashidi, Panos Pardalos, Petar Momcilovic, and Azra Bihorac. "Application of machine learning techniques to high-dimensional clinical data to forecast postoperative complications." In: *PloS one* 11.5 (2016), e0155705.

[137] Gilles Vandewiele, Olivier Janssens, Femke Ongenae, Filip De Turck, and Sofie Van Hoecke. "GENESIM: genetic extraction of a single, interpretable model." In: *arXiv preprint arXiv:1611.05722* (2016).

[138] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[139] Paul Viola and Michael J Jones. "Robust real-time face detection." In: *International journal of computer vision* 57.2 (2004), pp. 137–154.

[140] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. "A brief survey on sequence classification." In: *ACM SIGKDD Explorations Newsletter* 12.1 (2010), pp. 40–48.

[141] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. "Show, attend and tell: Neural image caption generation with visual attention." In: *International conference on machine learning*. 2015, pp. 2048–2057.

[142] Pedro Pablo Espana Yandiola et al. "Prospective comparison of severity scores for predicting clinically relevant outcomes for patients hospitalized with community-acquired pneumonia." In: *Chest* 135.6 (2009), pp. 1572–1579.

[143] Q. Yang, J. Yin, C. X. Ling, and T. Chen. "Postprocessing Decision Trees to Extract Actionable Knowledge." In: *Proc. ICDM*. 2003.

[144] Qiang Yang, Jie Yin, Charles Ling, and Rong Pan. "Extracting actionable knowledge from decision trees." In: *IEEE Transactions on Knowledge and data Engineering* 19.1 (2007), pp. 43–56.

[145] Jieping Ye et al. "Heterogeneous data fusion for alzheimer's disease study." In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 1025–1033.

[146] Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks (2013)." In: *arXiv preprint arXiv:1311.2901* (2013).

[147] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. "Adaptive deconvolutional networks for mid and high level feature learning." In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, pp. 2018–2025.

[148] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. "Interpretable convolutional neural networks." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8827–8836.

[149] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J Leon Zhao. "Time series classification using multi-channels deep convolutional neural networks." In: *Web-Age Information Management*. Springer, 2014, pp. 298–310.

[150] Yi Zheng, Qi Liu, Enhong Chen, J Leon Zhao, Liang He, and Guangyi Lv. "Convolutional Nonlinear Neighbourhood Components Analysis for Time Series Classification." In: *Advances in Knowledge Discovery and Data Mining*. Springer, 2015, pp. 534–546.

[151] Quan Zhou, Wenlin Chen, Shiji Song, Jacob R Gardner, Kilian Q Weinberger, and Yixin Chen. "A Reduction of the Elastic Net to Support Vector Machines with an Application to GPU Computing." In: *AAAI*. 2015, pp. 3210–3216.