Extracting Rules from Trained Machine Learning Models with Applications in Bioinformatics

機械学習モデルからの知識抽出と生命情報学への応用

Pengyu Liu

劉鵬宇

Abstract

Machine learning methods have been successfully applied to various areas. However, some machine learning-based models are not explainable (e.g., Artificial Neural Networks), which may prevent massive applications in related biological fields. In this study, we try to make the model interpretable by extracting explicit rules between the inputs and outputs.

We first propose two approaches to extracting rules from a trained neural network consisting of linear threshold functions. The first one leads to an algorithm that extracts rules in the form of Boolean functions. Compared with an existing one, this algorithm outputs much more concise rules if the threshold functions correspond to 1-decision lists, majority functions, or certain combinations of these. The second one extracts probabilistic rules representing relations between some of the input variables and the output using a dynamic programming algorithm. The algorithm runs in pseudo-polynomial time if each hidden layer has a constant number of neurons. We demonstrate the effectiveness of these two approaches by computational experiments.

Then we consider applying an explainable machine learning model to predict human Dicer cleavage sites.

Human Dicer is an enzyme that cleaves pre-miRNAs into miRNAs. Several models have been developed to predict human Dicer cleavage sites, including PHDCleav and LBSizeCleav. Given an input sequence, these models can predict whether the sequence contains a cleavage site. However, these models only consider each sequence independently and lack interpretability. Therefore, it is necessary to develop an accurate and explainable predictor, which employs relations between different sequences, to enhance the understanding of the mechanism by which human Dicer cleaves pre-miRNA.

In this part, we develop an accurate and explainable predictor for human Dicer cleavage site – ReCGBM. ReCGBM is based on lightGBM, which can output the relationship between the features and outputs inherently. We design relational features and class features as inputs to a lightGBM model. Computational experiments show that ReCGBM achieves the best performance compared with the existing methods. Further, we find that features in close proximity to the center of pre-

miRNA are more important and make a significant contribution to the performance improvement of the developed method.

The results of this study show that ReCGBM is an interpretable and accurate predictor. Besides, the analyses of the relationship between the feature importance and the output labels show that it might be of particular interest to consider more informative features close to the center of the pre-miRNA in future predictors.

Acknowledgments

During my career at Kyoto University, I have received support and help from many people.

First of all, I would like to express my gratitude to my supervisor, Professor Tatsuya Akutsu, for your valuable comments, which always helped me building my research ideas. I have also benefited a lot from your advice on writing skills as well as mathematical proofs. I will always keep these things in mind.

I appreciate Professor Akihiro Yamamoto and Professor Hisashi Kashima for being my co-advisor. Your valuable comments greatly improved my thesis and presentations.

I want to thank Professor Avraham Melkman at Ben-Gurion University. Although you only visited our lab for several months in my Ph.D. career, your advice and encouragement are treasures to me.

Thank you to Professor Jiangning Song at Monash University and Professor Chun-yu Lin at National Chiao Tung University for providing me so much interesting insights into bioinformatics.

I want to thank all members, former members, and visiting students in the Akutsu lab for their encouragement and help at Kyoto University. It's my pleasure to work with you.

Finally, I would like to appreciate my father–Manyuan Liu, my mother–Min Guo, and my wife–Xiaohui Huang. Without your support, I cannot get my degree smoothly.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

With the rapid growth of data, it is crucial to analyze data to help us make conclusions in different studies. Therefore, methods that can analyze data, such as machine learning, play an essential role in many fields.

According to the types of data, studies on machine learning can mainly be divided into two types:

- **Unsupervised learning**: Given a dataset with N samples $\{x_1, ..., x_N\}$, unsupervised learning is trying to divide the data into different classes.
- **Supervised learning**: Given a dataset with N paired samples $\{(x_1, y_1), ..., (x_N, y_N)\}$, supervised learning learns a function with input x_i and output y_i, where the available sample x_i in a supervised learning task has a corresponding label y_i for $1 \leq i \leq N$.

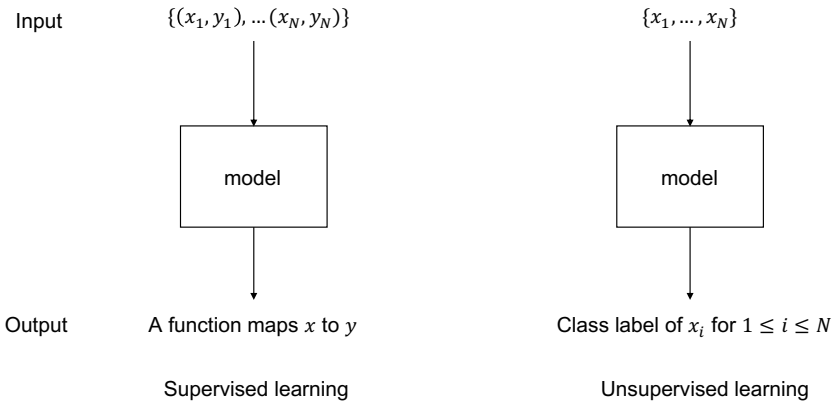


Figure 1.1: Supervised learning and unsupervised learning

Figure 1.1 shows the difference between supervised learning and unsupervised learning. In this study, we will focus on methods and applications for supervised learning.

In supervised learning, there are two kinds of tasks: classifications and regressions.

In classification tasks, each y_i for a given x_i is a class label. One example is the classification task on MNIST dataset [LBBH98]. In this dataset, there are some figures of handwritten digits. Each figure has a label (from 1 to 9). The task aims to train a model so that given an input x_i, the model outputs the correct label y_i. In regression tasks, each y_i for a given x_i is a continuous number. An example is the Boston housing dataset [HJR78].

There are plenty of methods that can be used to finish these tasks. One choice is artificial neural networks. The artificial neural network has recently become a compelling method with the development of deep neural networks [LBH15]. A deep neural network can build an accurate model when enough data are available. However, overfitting is a serious problem when we employ neural network-based methods [CLG01]. To learn more about overfitting in trained neural networks and how a trained neural network makes decisions, we can find some explicit relationships between the inputs and outputs of a trained neural network. Therefore, it is necessary to develop methods to extract explicit rules between the inputs and outputs.

Another good choice is gradient boosting. Gradient boosting shows powerful performance in the Kaggle competition and the KDD Cup. Popular implementations such as XGBoost [CG16], LightGBM [KMF$^+$17] make it convenient to use. One advantage of this method is that rule extraction is inherently possible. That is, feature importance related to the outputs can be extracted directly from a trained model.

1.2 Contribution

In this study, our contributions mainly include two parts.

First, we propose two algorithms for rule extractions in trained neural networks. One algorithm is based on Boolean operations. It can extract a Boolean function representing the relationship between the inputs and outputs of a trained neural network. However, complicated Boolean functions may be obtained when there are no simple Boolean functions representing the target neural network. To partially solve this problem, we develop a rule extraction algorithm based on conditional probability and dynamic programming. This algorithm can extract probabilistic rules for trained neural networks with a proper size.

Second, we apply the gradient boosting algorithm to identify the human Dicer

cleavage sites. In this part, we develop a predictor – ReCGBM based on gradient boosting and affinity propagation. The proposed method shows better performance comparing with some existed methods. Besides, some biological rules have been found through the feature importance of the trained model.

1.3 Organization

In Chapter 2, we give a brief introduction about the models and methods which will be used in later chapters.

In Chapter 3, we introduce methods of rule extractions for artificial neural networks. In this study, we propose two rule extraction methods. The first one extracts Boolean rules. The second one extracts probabilistic rules. We show the effectiveness of these methods by computational experiments.

In Chapter 4, we introduce gradient boosting for the human Dicer cleavage sites prediction. In this study, we train a classifier – ReCGBM, which is based on gradient boosting. ReCGBM makes accurate predictions. Besides, it outputs the relationship between the feature importance and the output labels, which leads to the discovery of biological meanings.

Finally, Chapter 5 summarizes the methods and applications.

Chapter 2

Preliminaries

2.1 Artificial Neural Networks

2.1.1 Neurons

The basic unit in a artificial neural network is artificial neurons (or perceptrons). A neuron is a function $f(x)$:

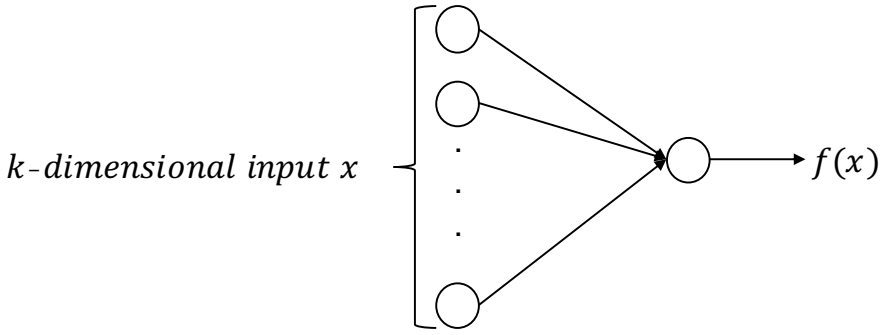


Figure 2.1: A artificial neuron (perceptron)

where x is the k-dimensional input and $f(x)$ is the output.

2.1.2 Feedforward Neural Networks

A feedforward neural network (or multilayer perceptron) is a directed acyclic graph that consists of many neurons. The neurons without indegrees are input neurons . Similarly, the neurons without outdegrees are output neurons. The input neurons are situated in the input layer, while the output neurons are located in the output layer. The other neurons belong to the middle layers. Figure 2.2 gives an example.

Given a 4-dimensional input x, the output of this neural network is $f_3([f_1(x), f_2(x)])$, where $[f_1(x), f_2(x)]$ is the 2-dimensional input of f_3.

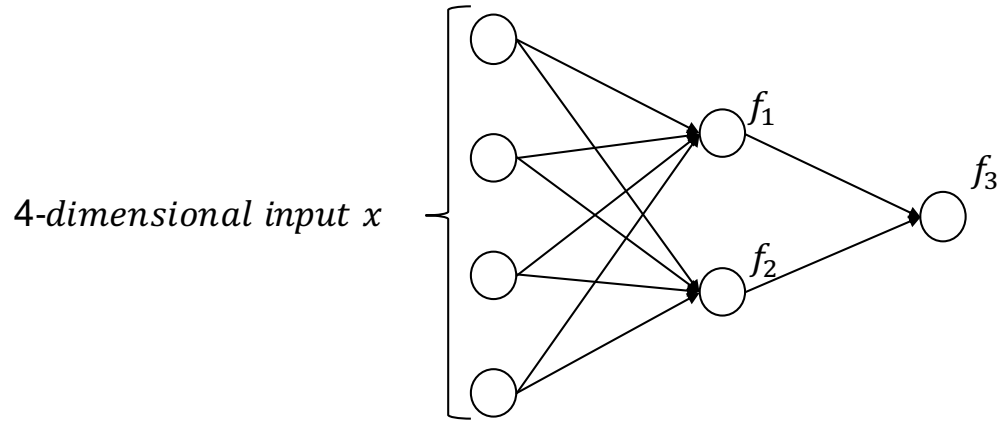


Figure 2.2: A feedforward neural network

The training procedure of a feedforward neural network is mostly conducted by some variants of gradient descent such as Adam [KB14], AdaGrad [DHS11].

The choice of function $f(x)$ in each neuron may affect the performance and training speed of a neural network. A fundamental choice is the linear threshold function. It is a binary function that usually appears in the theoretical analyses of a neural network. The problem with the linear threshold function is that it is challenging to be trained by gradient-based methods. A more advanced choice is the sigmoid function. It is widely used in different neural networks. However, it suffers from gradient vanishing in deep neural networks. Another popular choice is the ReLU function. In this study, we focus on the theoretical analyses of artificial neural networks. Therefore, our algorithms assume that $f(x)$ is the linear threshold function. The formal definition of linear threshold function is given in Chapter 3.

2.2 Boosting

Boosting is a group of machine learning methods that build a strong learner through some weak learners. Figure 2.3 shows how a boosting-based algorithm works. In this part, we will first introduce the AdaBoost [FS97] algorithm. Then we will talk about Gradient Boosting and some famous implementations.

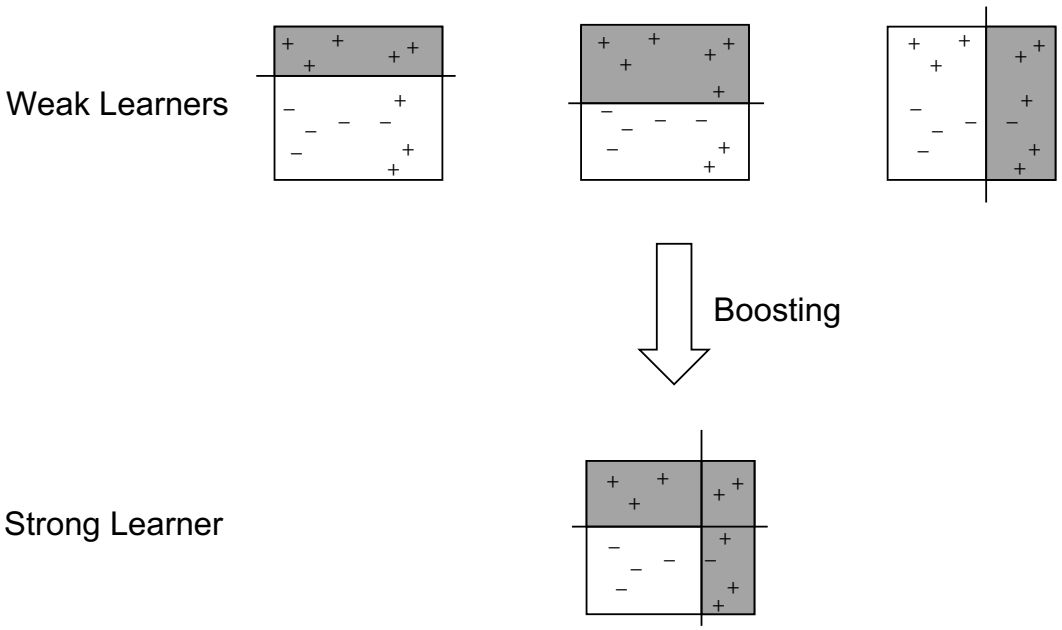


Figure 2.3: An illustration of intuitions behind the Boosting methods

2.2.1 AdaBoost

Given a training set $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$ where $x_i \in \mathcal{X} \subseteq \mathbb{R}^n$ and $y_i \in \mathcal{Y} = \{-1, +1\}$, and the algorithm to learn the weak learners (e.g. decision trees). The AdaBoost algorithm [FS97] learn a classifier $G(x)$ through the following procedures:

1. Initialize distribution D_1 as:

$$D_1 = (w_{11}, ..., w_{1i}, ..., w_{1N}), w_{1i} = \frac{1}{N}, i = 1, 2, ..., N$$

2. For m = 1, ..., M:

   (a) Train weak learner $G_m(x)$ using distribution D_m:

$$G_m(x) : \mathcal{X} \to \{-1, +1\}$$

   (b) Compute the error ϵ_m of $G_m(x)$ on the training set:

$$\epsilon_m = \sum_{i=1}^{N} P(G_m(x_i) \neq y_i) = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)$$

(c) Compute the coefficient of $G_m(x)$:

$$\alpha_m = \frac{1}{2}\log\frac{1-\epsilon_m}{\epsilon_m}$$

where log is natural logarithm.

(d) Update

$$D_{m+1} = (w_{m+1,1}, ..., w_{m+1,i}, ..., w_{m+1,N})$$
$$w_{m+1,i} = \frac{w_{mi}}{Z_m}\exp(-\alpha_m y_i G_m(x_i)) \quad for \quad 1 \le i \le N$$

where Z_m is a normalization factor of D_{m+1}:

$$Z_m = \sum_{i=1}^{N} w_{mi}\exp(-\alpha_m y_i G_m(x_i))$$

3. Output the final classifier:

$$G(x) = sign(\sum_{m=1}^{M}\alpha_m G_m(x))$$

2.2.2 Additive Model

The additive model is in the following form:

$$f(x) = \sum_{m=1}^{M}\beta_m b(x;\gamma_m)$$

where $b(x;\gamma_m)$ is the mth weak learner and β_m is the corresponding coefficient.

The additive model can be solved by Forward Stagewise Additive Modeling:

1. Initialize $f_0(x) = 0$.
2. For $1 \le m \le M$:
   (a) Compute
   $$(\beta_m, \gamma_m) = arg\min_{\beta,\gamma}\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)).$$
   (b) $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma_m)$

$L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma))$ represent the loss function. The AdaBoost algorithm can be regarded as a special case of the additive model with an exponential loss function [FHT+00] in the following form:

$$L(y, f(x)) = \exp(-yf(x))$$

2.2.3 Gradient Boosting

The AdaBoost algorithm is efficient. However, this method is sensitive to noise data since the usage of the exponential loss function. Jerome H. Friedman [Fri01] developed the gradient boosting algorithm, which is more robust since many loss functions can be applied.

Given a training set $T = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$. Assume a differentiable loss function $L(x, f(x))$ is choosen. the gradient boosting algorithm works in the following procedure:

1. Initialize model with a constant value:
$$f_0(x) = arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$
2. For $1 \leq m \leq M$:
   (a) Compute the pseudo-residuals:
   $$r_{im} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=f_{m-1}(x)}$$
   where $1 \leq i \leq N$.
   (b) Train a weak learner $G_m(x)$ on the training set $\{(x_1, r_{1m}), (x_2, r_{2m}), ..., (x_N, r_{Nm})\}$.
   (c) Compute γ_m by solving the following problem:
   $$\gamma_m = arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \gamma G_m(x_i))$$
   (d) Update the model:
   $$f_m(x) = f_{m-1}(x) + \gamma_m G_m(x)$$
3. Output the classifier $f_M(x)$.

For two-class classification task, the loss function is:

$$L(y, f(x)) = \log(1 + \exp(-2yf(x)))$$

where $y \in \{-1, 1\}$.

Compare with the AdaBoost algorithm, the Gradient Boosting algorithm is more flexible. It is suitable for both regression tasks and classification tasks. Besides, the versatile choice of loss functions makes this algorithm more robust with noise data.

Usually, decision trees are chosen as the weak learners in the Gradient Boosting algorithm. Therefore, we call this GBDT (Gradient Boosting Decision Tree).

2.2.4 Implementations

XGBoost

XGBoost [CG16] is one of the most famous implementations of the gradient boosting algorithm. It made many improvements compared with previous methods: First, they proposed a theoretically justified weighted quantile sketch for efficient proposal calculation; Second, to handle sparse structures and missing values in practical datasets, they developed a novel sparsity-aware algorithm for parallel tree learning; Third, to make the algorithm works faster, they proposed an effective cache-aware block structure for out-of-core tree learning.

LightGBM

LightGBM [KMF+17] is a famous implementation of the gradient boosting algorithm. It is even faster than XGBoost. Besides, categorical features can be used directly with one-hot encoding. We will briefly introduce the advantage of LightGBM in Chapter 4.

Chapter 3

Extracting Boolean and Probabilistic Rules from Trained Neural Networks

3.1 Background

Recent developments in deep learning technologies have demonstrated the power of artificial neural networks in making predictions in various areas. It is therefore of great interest to develop a methodology for interpreting how a trained neural network arrives at its predictions: better understanding makes for greater trust. There are two major approaches to the interpretation of trained networks: mechanism explanation and behavior explanation. The former sees the network as a white box and aims to explain its inner workings. The latter settles for explaining what outputs will be obtained from the different inputs, viewing the network as if it were a black box. Many studies adopt the latter approach because it is easier and useful.

Different methodologies can be broadly divided as follows:

- Boolean function extraction: Extract a Boolean function from a trained neural network [Tsu00].
- Activation maximization: Create an input to maximize the network output and show a representative example of a specific category [Le13].
- Sensitivity analysis: Examine the sensitivity of the network against changes of each input [STK+17].
- Deconvolution: Trace the path from the output to the input and analyze attributes learned by a convolutional neural network [SDBR14].
- LIME: Explain the predictions by approximation with an interpretable model [RSG16].

We focus here on Boolean function extraction. Most such extraction algorithms fall into of two categories: decompositional or pedagogical algorithms.

Decompositional algorithms such as [Tsu00] extract a Boolean function for each individual neuron, and merge these rules into a set of rules describing the behavior of the neural network . Such algorithms generate their results in terms of the structure of the network so that we can understand how each neuron works.

Pedagogical algorithms, such as [AK12] and [SWI07], view rule extraction as the task of learning a Boolean function from data samples. Although these methods generate accurate results, the results may not reflect the function of each neuron.

This study is motivated by Tsukimoto's work [Tsu00]. His algorithm extracts Boolean functions in disjunctive normal form from each of the neurons independently and then combines these functions. It works in polynomial time if the size of each term is bounded by a constant. In this study we explore several extensions and modifications of this pioneering work, motivated by the following considerations.

First, Tsukimoto's algorithm requires the specification of the maximum size of a term; if that size is too small the algorithm may fail to extract a Boolean function, and if it is large, the running time may be too long. Second, for a neural network with hidden layers the size of the Boolean function may be large.

To begin addressing these issues we propose two algorithms; admittedly, further refinements will be needed to make them practical for large-scale neural networks. The first one extracts Boolean functions in the form of nested canalyzing functions (equivalent to 1-decision lists), majority functions, or certain combinations of the two. This algorithm is useful if each neuron indeed has such a form. Otherwise, as for Tsukimoto's algorithm, the algorithm may output large Boolean functions. This phenomenon is inevitable if exact prediction rules are to be extracted in the form of Boolean functions. The second algorithm extracts rules in the form of conditional probabilities. Each conditional probability represents a relation between an input attribute, or a combination of input attributes, and an output. Although the prediction rules obtained by this approach are usually not exact they can be concise. The potential usefulness of these two algorithms are demonstrated by means of computational experiments.

3.2 Preliminaries

3.2.1 Neural Networks

In this study a neuron is a linear threshold function [O'D14], with the exception of Section 3.5.2, in which we consider also neurons that are sigmoid functions.

Definition 1. *Given $\theta, w_1, \cdots, w_n \in \mathbb{R}$, a linear threshold function is a Boolean-valued function $f : \{0,1\}^n \to \{0,1\}$ defined by*

$$f(x) = H(w_1x_1 + \cdots + w_nx_n - \theta)$$

where $H(z)$ is the Heaviside function

$$H(z) = \begin{cases} 1 & z \geq 0, \\ 0 & z < 0. \end{cases}$$

We will assume that all weights and threshold values are integers. This is a reasonable assumption in practice inasmuch as a computer handles only rational numbers, and a threshold function with rational weights can be transformed into an equivalent threshold function with integer weights.

A sigmoid function is in general characterised by its "S"-shaped curve, but here we limit it to the logistic function:

Definition 2. *The one-dimensional sigmoid function is*

$$S(x) = \frac{1}{1 + e^{-x}}.$$

More generally, given $\theta, w_1, \cdots, w_n \in \mathbb{R}$, the n-dimensional sigmoid function is $S(w_1x_1 + \cdots + w_nx_n - \theta)$.

Definition 3. *In this study, a neural network is a directed* acyclic *graph with neurons situated at the vertices. The neurons with zero indegree are input neurons, which receive signals only from the outside world, and those with zero outdegree are output neurons.*

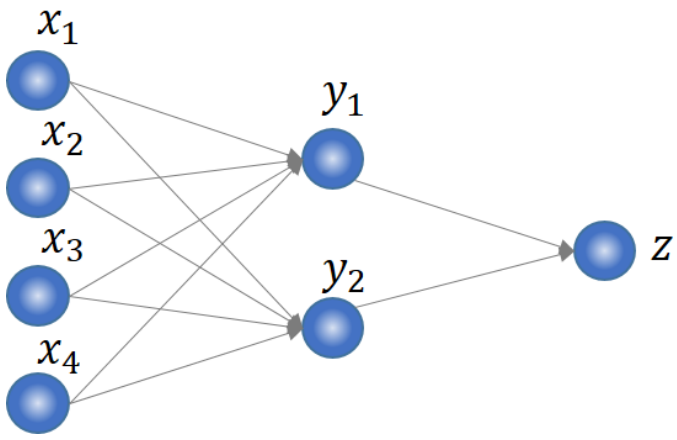


Figure 3.1: Example of a neural network.

Fig. 3.1 shows an example of a neural network in which x_1, x_2, x_3, x_4 are the input neurons and z is the output neuron. Since the input values are 0 or 1 the output value is determined by a Boolean function of the input values. The problem we address is the extraction of this Boolean function in a simple form from a given neural network.

3.2.2 Majority and Nested Canalyzing Functions

For Boolean variables x and y, $\bar{x}$, $x \wedge y$, $x \vee y$ denote the negation of x, conjunction of x and y, and disjunction of x and y, respectively. A literal is either a Boolean variable or its negation. A term is a conjunction of literals, and a conjunction of k literals is called a k-term.

Definition 4. *Given n literals $\ell_1, \ldots, \ell_n$ and $k \in \{1, \ldots, n\}$, the k-out-of-n majority function $Maj_k(\ell_1, \ldots, \ell_n)$ is the Boolean function that is the disjunction of all conjunctions of exactly k out of the n literals. Thus its output is 1 when k or more literals are 1, and 0 otherwise.*

Definition 5. *A Boolean function $f(x)$ is nested canalyzing if it can be represented as*

$$f(x) = \ell_1 \vee \cdots \vee \ell_{k_1-1} \vee (\ell_{k_1} \wedge \cdots \wedge \ell_{k_2-1} \wedge (\ell_{k_2} \vee \cdots \vee \ell_{k_3-1} \vee (\cdots)))$$

where $\ell_i \in \{x_1, \bar{x}_1, \cdots, x_n, \bar{x}_n\}$ and $1 \leq k_1 < k_2 < \cdots$. It can be assumed that each x_i appears at most once in $f(x)$ either positively or negatively.

It has been reported that many biologically relevant functions are nested canalyzing ones [HSWK02, JRL07]. Therefore, extracting rules in the form of nested canalyzing functions is meaningful. Next we define the decision list [Ant10]:

Definition 6. *Let M be a set of Boolean functions on $\{0,1\}^n$. The decision list f on M is a finite sequence:*

$$f = \langle (f_1, c_1), (f_2, c_2), \cdots, (f_r, c_r) \rangle,$$

such that $f_i \in M$ and $c_i \in \{0,1\}$ for $i = 1, \ldots, r$. The value of f is defined by $f(y) = c_j$ where $j = \min\{i | f_i(y) = 1\}$, or 0 if there is no i such that $f_i(y) = 1$.

A 1-decision list is a decision list in which each f_i is a literal.

The decision list f is like a sequence of "if then else" commands:

if $f_1(y) = 1$ **then** $f(y) = c_1$
else if $f_2(y) = 1$ **then** $f(y) = c_2$
.
.
else if $f_r(y) = 1$ **then** $f(y) = c_r$
end if

Proposition 7 (Theorem 2.8 of [JRL07].)**.** *The class of* 1*-decision lists is equivalent to the class of nested canalyzing functions.*

Proposition 8. *The number of nested canalyzing functions on n variables is bounded above by $n! \cdot 5^n$.*

Proof. There are $n!$ possible orderings of variables. Consider a particular order, say $x_1, x_2, x_3, \cdots, x_n$. The number of functions with this order, can be bounded recursively from the following five possibilities for $f(x_1, \cdots, x_n)$:

- $x_1 \wedge g(x_2, \cdots, x_n)$,
- $\bar{x}_1 \wedge g(x_2, \cdots, x_n)$,
- $x_1 \vee g(x_2, \cdots, x_n)$,
- $\bar{x}_1 \vee g(x_2, \cdots, x_n)$,
- x_1 is not used.

Therefore, there are at most $n! \cdot 5^n$ possibilities in total. □

A more precise but more complicated bound is given in [JRL07]. Since every nested canalyzing function can be represented as a linear threshold function [Ant01], and the number of linear threshold functions on the Boolean domain is at least $2^{n(n-1)/2}$ [Ant01], what these bounds show is that the class of nested canalyzing functions is a proper subset of the class of linear threshold functions.

3.3 Extraction of Boolean Functions

3.3.1 Extraction of a Nested Canalyzing Function or a Majority Function

In this subsection, we consider first the extraction of a rule having the form of a nested canalyzing function, and then the extraction of a rule in the form of a nested majority function. Notice that we also use the expression $w_i x_i + \cdots + w_n x_n \geq \theta$ to represents a linear threshold function $H(w_i x_i + \cdots + w_n x_n - \theta)$.

When analyzing the running times of algorithms we will use $O^*(f(N))$ to represent $O(f(N)poly(N))$, where N is the total size of input data. We assume that the size of each parameter is polynomially bounded with respect to n, where n denotes the number of input variables. Recall that addition, subtraction, and multiplication can be done in time polynomial in the total size of the terms involved.

But first let us present a simple but very useful Lemma.

Lemma 9. *For the purpose of rule extraction from a linear threshold function it can be assumed without loss of generality that the threshold function has the form* $H(w_1x_1 + \cdots + w_nx_n - \theta)$ *with* $w_1 \geq \cdots \geq w_n > 0$.

Proof. Repeat the following step until no negative weights remain: if $w_i < 0$ for some i, replace θ by $\theta - w_i$, w_i by $-w_i$, and x_i by $\bar{x}_i$. These two threshold functions are clearly equivalent.

Next, reorder the resulting linear threshold function so that the terms w_ix_i appear in descending order of w_i.

Finally, for convenience, replace each $\overline{x_i}$ by x_i. After extraction of the rule this replacement can be reversed. □

To illustrate the procedure described in the proof, consider the linear threshold function:

$$3x_1 - 5x_2 + 6x_3 \geq 5.$$

Dealing with the negative weight and reordering of the terms yields

$$6x_3 + 5\bar{x}_2 + 3x_1 \geq 10.$$

Summarizing, we assume w.l.o.g. that the input to the rule-extracting algorithms is a *canonical linear threshold function*

Definition 10. *A linear threshold function* $t(\mathbf{x}) = H(w_1x_1 + \cdots + w_nx_n - \theta)$ *is in canonical form if* $w_1 \geq w_2 \geq \cdots \geq w_n > 0$.

Algorithm 1 *ExtractNC* - Extracts a Nested Canalyzing Function, if possible

Input: A canonical threshold function $t \equiv w_i x_i + \cdots + w_n x_n \geq \theta$, $1 \leq i \leq n$.
Output: An NC-function equivalent to t if there is one, else NONE.

ExtractNC$(w_i x_i + \cdots + w_n x_n \geq \theta)$:
if $\theta \leq 0$ **then**
  return 1;
else if $\sum_{j=i}^{n} w_j < \theta$ **then**
  return 0;
end if/* $0 < \theta \leq \sum_{j=i}^{n} w_j$ */
if $i = n$ **then**
  return x_n;
end if; /* $i < n$ and $0 < \theta \leq \sum_{j=i}^{n} w_j$ */
if $w_i \geq \theta$ **then**
  $g(x_{i+1}, \ldots, x_n) \leftarrow$ *ExtractNC*$(w_{i+1} x_{i+1} + \cdots + w_n x_n \geq \theta)$;
  return $x_i \vee g(x_{i+1}, \cdots, x_n)$;
else if $\sum_{j=i+1}^{n} w_j < \theta$ **then**
  $g(x_{i+1}, \ldots, x_n) \leftarrow$ *ExtractNC*$(w_{i+1} x_{i+1} + \cdots + w_n x_n \geq \theta - w_i)$;
  return $x_i \wedge g(x_{i+1}, \cdots, x_n)$;
else
  return NONE;
end if

Note that 1, 0 and NONE satisfy $1 \vee g(\cdots) = 1$, $1 \wedge g(\cdots) = g(\cdots)$, $0 \wedge g(\cdots) = 0$, $0 \vee g(\cdots) = g(\cdots)$, NONE $\wedge\, g(\cdots) =$ NONE, and NONE $\vee\, g(\cdots) =$ NONE.

Theorem 11. *ExtractNC takes polynomial time to extract a nested canalyzing function from a given canonical threshold function if possible, outputting NONE if there is no such function.*

Proof. It is straightforward to see that *ExtractNC* works in polynomial time. We prove the correctness of *ExtractNC* by induction on the number of variables.

Clearly *ExtractNC* returns a correct nested canalyzing function when the threshold function consists of a single variable. For the induction step suppose i is given and that *ExtractNC* returns a correct solution (possibly NONE) given any canonical threshold function of the form $w'_{i+1} x_{i+1} + \cdots + w'_n x_n \geq \theta$. We will show that then a correct solution is returned for any canonical threshold function $w_i x_i + \cdots + w_n x_n \geq \theta$.

When $w_i \geq \theta$, the threshold function has value 1 on an assignment $(a_i, \ldots, a_n)$ to $(x_1, \ldots, x_n)$, i.e. $w_i a_i + \cdots + w_n a_n \geq \theta$, if and only if either $a_i = 1$ or both $a_i = 0$ and $w_{i+1} a_{i+1} + \cdots + w_n a_n \geq \theta$. Therefore, $w_i x_i + \cdots + w_n x_n \geq \theta$ corresponds to a nested canalyzing function iff $w_{i+1} x_{i+1} + \cdots + w_n x_n \geq \theta$ corresponds to a nested canalyzing function $g(x_{i+1}, \ldots, x_n)$, and if so it corresponds to $x_i \vee g(x_{i+1}, \ldots, x_n)$.

In case $\sum_{j=i+1}^{n} w_j < \theta$ the threshold function has value 1 on an assignment $(a_i, \ldots, a_n)$ if and only if $a_i = 1$ and $w_{i+1}a_{i+1} + \cdots + w_n a_n \geq \theta - w_i$. Thus $x_i \wedge g(x_{i+1}, \ldots, x_n)$ is the nested canalyzing function corresponding to the threshold function, where $g(x_{i+1}, \ldots, x_n)$ corresponds to $w_{i+1}x_{i+1} + \cdots + w_n x_n \geq \theta - w_i$.

To prove that the the algorithm is correct also when it returns NONE, we prove that if there is a nested canalyzing function f that is equivalent to the canonical threshold function $t(\mathbf{x}) \equiv w_i x_i + \cdots + w_n x_n \geq \theta)$ then either $w_i \geq \theta$ or $\sum_{j=i+1}^{n} w_j < \theta$. As stated in the algorithm we can assume that $i < n$ and

$$0 < \theta < \sum_{j=i}^{n} w_j. \tag{3.1}$$

Suppose first that f has the form $f(x_i, \ldots, x_n) = \ell_k \vee g(x_i, \ldots, x_n)$, for some $i \leq k \leq n$, and g not a function of x_k. Although ℓ_k could in general be x_k or $\overline{x_k}$, here $\ell_k = \overline{x_k}$ is impossible: otherwise $f(\mathbf{0}) = 1$, whereas $t(\mathbf{0}) = 0$ since $0 < \theta$.

Thus $f = 1$ whenever $x_k = 1$. Consider the assignment $\mathbf{a}$ with values $a_k = 1$, and $a_j = 0$ for $j \neq k$. Then from $t(\mathbf{a}) = 1$ it follows that $w_k \geq \theta$.

If f is of the form $f(\mathbf{x}) = x_k \wedge g$ then $f(\mathbf{a}) = 0$ for any assignment $\mathbf{a}$ such that $a_k = 0$. Arguing analogously to the above it follows that $\sum_{j \neq k} w_j < \theta$.

Finally, we prove that the index k appearing in the argument above can in fact be taken to be i, because t is in canonical form. Consider, for example, the case that f has the form $x_k \vee g(x_i, \ldots, x_n)$, with $i \neq k$ and $g(x_i, \ldots, x_n)$ not dependent on x_k. We saw that then $w_k \geq \theta$. From $w_i \geq w_k$ it follows that $w_i + w_{i+1}a_{i+1} + \cdots + w_n a_n \geq \theta$ for any $a_{i+1}, \ldots, a_n$, i.e. $t(\mathbf{a}) = 1$, whenever $a_i = 1$. Thus in fact $f = x_i \vee x_k \vee g'$.

Similarly, if f has the form $x_k \wedge g(x_i, \ldots, x_n)$ then $\sum_{j>i} w_j \leq \sum_{j \neq k} w_j < \theta$, so that $t(\mathbf{a}) = 0$ whenever $a_i = 0$. Consequently $f = x_i \wedge x_k \wedge g'$. □

Next we present an algorithm to test whether a canonical threshold function is equivalent to the k-out-of-n majority function, $Maj_k(x_1, \ldots, x_n)$. Although several methods have been developed for the simplification of a linear threshold function into a set of majority functions [Bar93], it is unclear whether any of these methods is directly applicable to the extraction of a majority function, and our algorithm has the virtue of being very simple.

Algorithm 2 *ExtractMF* - Extracts a Majority Function if possible

Input: A canonical threshold function t with weights $w_1 \geq \cdots \geq w_n > 0$.
Output: $Maj_k(x_1, \ldots, x_n)$, $k > 1$, if it is equivalent to t, else NONE.

```
ExtractMF(w_1x_1 + ··· + w_nx_n ≥ θ):
for k = 2 to n do
  if w_{n-k+1} + ··· + w_n ≥ θ and w_1 + ··· + w_{k-1} < θ then
    return Maj_k(x_1, ..., x_n);
  end if
end for
return NONE;
```

Proposition 12. *ExtractMF returns the correct answer in polynomial time.*

Proof. The algorithm clearly takes polynomial time, computing no more than n times two sums of no more than n terms. To prove correctness assume first that t does correspond to Maj_k for some $k > 1$. Since $Maj_k(x_1, \ldots, x_n)$ is the disjunction of all conjunctions consisting of k variables it has the value 1 for the assignment $a_1 = \cdots = a_{n-k} = 0$, $a_{n-k+1} = \cdots = a_n = 1$, meaning that $w_{n-k+1} + \cdots + w_n \geq \theta$. But it has the value 0 for the assignment $a_1 = \cdots = a_{k-1} = 1$, $a_k = \cdots = a_n = 0$, as each conjunction contains at least one 0 multiplicand. Since $t(\mathbf{a}) = 0$ means that $w_1 + \cdots + w_{k-1} < \theta$, this Maj_k will be output by *ExtractMF*.

Conversely, assume that *ExtractMF* returns Maj_k, so that $w_{n-k+1} + \cdots + w_n \geq \theta$. Because $w_1 \geq \cdots \geq w_n > 0$ this implies that $\sum_{j=1}^{k} w_{i_j} \geq \theta$ for any $1 \leq i_1 \leq \cdots \leq i_k \leq n$, i.e. that $t(\mathbf{a}) = 1$ for any assignment $\mathbf{a}$ that contains at least k 1's. Furthermore, $w_1 + w_2 + \cdots + w_{k-1} < \theta$ also holds for this k, which similarly implies that $t(\mathbf{a}) = 0$ for any assignment $\mathbf{a}$ that contains at most $k-1$ 1's. Thus, the input canonical threshold function is equivalent to MF_k. □

3.3.2 Extraction of a general Boolean Function

If the given canonical threshold function is equivalent to a nested canalyzing function or a k-out-of-n majority function then the algorithms of the previous section are able to construct a concise rule. What to do if the algorithms are unsuccessful is the question we address in this section. The basic idea is to break the threshold function up into two threshold functions, using the following equivalence, and to recurse :

$$\begin{aligned} w_1x_1 + \cdots + w_nx_n \geq \theta \iff \\ (x_1 \wedge (w_2x_2 + \cdots + w_nx_n \geq \theta - w_1)) \quad \vee \quad (\bar{x}_1 \wedge (w_2x_2 + \cdots + w_nx_n \geq \theta)). \end{aligned}$$

In order to avoid as much as possible the potentially exponential size of the resulting Boolean function, the Algorithm tries at each stage to extract a concise Boolean function using Algorithm 1 and Algorithm 2.

Algorithm 3 ExtractBF - extracts a Boolean Function

Input: A canonical threshold function.
Output: A Boolean function.
$ExtractBF(w_i x_i + \cdots + w_n x_n \geq \theta)$:
$g \leftarrow ExtractNC(w_i x_i + \cdots + w_n x_n \geq \theta)$;
if g is not NONE **then**
return g;
end if
$g \leftarrow ExtractMF(w_i x_i + \cdots + w_n x_n \geq \theta)$;
if g is not NONE **then**
return g;
end if
return $(x_i \wedge ExtractBF(w_{i+1}x_{i+1} + \cdots + w_n x_n \geq \theta - w_i)) \vee ExtractBF(w_{i+1}x_{i+1} + \cdots + w_n x_n \geq \theta)$;

Proposition 13. *ExtractBF transforms any linear threshold function into a Boolean function composed of disjunctions, conjunctions, and majority functions of literals in $O^*(2^n)$ time.*

In the worst case, the algorithm may output an exponential size Boolean function using exponential time. However, the algorithm works in polynomial time with respect to the output size, and the output size is expected to be small if the given threshold function can be represented as a composition of majority functions, nested canalyzing functions, disjunctions and conjunctions.

Let us briefly compare our algorithm with Tsukimoto's algorithm [Tsu00], which outputs the DNF form of a Boolean function, and works in polynomial time if the size of each term is bounded by a constant. Otherwise it is an exponential time algorithm, or an approximate one. Our algorithm is exact and works in polynomial time if a given linear threshold function corresponds to a nested canalyzing function or a k-out-of-n majority function.

In all cases that we checked our algorithm outputs a Boolean function that is more concise than the one output by Tsukimoto's algorithm. For example, given a linear threshold function $0.5x_1 + 0.4x_2 + 0.1x_3 + 0.1x_4 + 0.1x_5 + 0.1x_6 \geq 1$, our algorithm outputs $x_1x_2(x_3 \vee x_4 \vee x_5 \vee x_6)$, whereas Tsukimoto's algorithm outputs $x_1x_2x_3 \vee x_1x_2x_4 \vee x_1x_2x_5 \vee x_1x_2x_6$. For another example, given a linear threshold function $0.5x_1 + 0.5x_2 + 0.4x_3 + 0.4x_4 + 0.4x_5 \geq 0.8$, our algorithm outputs

$Maj_2(x_1, \ldots, x_5)$. whereas Tsukimoto's algorithm outputs $x_1x_2 \vee x_1x_3 \vee x_1x_4 \vee x_1x_5 \vee x_2x_3 \vee x_2x_4 \vee x_2x_5 \vee x_3x_4 \vee x_3x_5 \vee x_4x_5$.

However, the rule extracted by our algorithm can be longer than necessary, even if the rule is a majority function but not on all n variables. Consider the function whose weights are 12,11,11,10,10,10,1 and its threshold is $\theta = 29$. This function is equivalent to $Maj_3(x_1, \ldots, x_6)$, but our algorithm outputs

$$
\begin{aligned}
&(x_1 \wedge ((x_2 \wedge (x_3 \vee x_4 \vee x_5 \vee x_6)) \vee (x_3 \wedge (x_4 \vee x_5 \vee x_6)) \vee (x_4 \wedge (x_5 \vee x_6)) \\
&\vee(x_5 \wedge x_6))) \\
&\quad \vee(x_2 \wedge ((x_3 \wedge (x_4 \vee x_5 \vee x_6)) \vee (x_4 \wedge (x_5 \vee x_6)) \vee (x_5 \wedge x_6))) \\
&\quad \vee(x_3 \wedge ((x_4 \wedge (x_5 \vee x_6)) \vee (x_5 \wedge x_6))) \\
&\quad \vee(x_4 \wedge x_5 \wedge x_6).
\end{aligned}
$$

If it is important to extract a concise (and therefore more informative) rule, even if it may take considerably more time, one can modify the above algorithm by replacing its last line by the following:

$$
\begin{aligned}
&g_1 \leftarrow (x_i \wedge ExtractBF(w_{i+1}x_{i+1} + \cdots + w_nx_n \geq \theta - w_i)) \\
&\quad \vee ExtractBF(w_ix_i + \cdots + w_{n-1}x_{n-1} \geq \theta); \\
&g_2 \leftarrow (x_n \wedge ExtractBF(w_ix_i + \cdots + w_{n-1}x_{n-1} \geq \theta - w_n)) \\
&\quad \vee ExtractBF(w_{i+1}x_{i+1} + \cdots + w_{n-1}x_{n-1} \geq \theta); \\
&\textbf{if } |g_1| \leq |g_2| \text{ return } g_1 \textbf{ else } \textit{return } g_2;
\end{aligned}
$$

In the above, we considered the problem of extracting a Boolean function from a single neuron. We consider next a network with hidden layers. In this case the extraction of the Boolean function corresponding to a given neuron involves combining the Boolean functions extracted for all its input neurons, which may result in a very complicated function.

We propose two methods for alleviating this problem. The first one is to add constraints on the length of the generated Boolean functions. Computational experiments using this approach are presented in Section 3.5.1. The second one is to extract rules that are conditional probabilities, rather than deterministic ones. In the next section we describe this probabilistic rule extraction algorithm.

3.4 Extraction of Probabilistic Rules from a Neural Network

3.4.1 Problem Definition

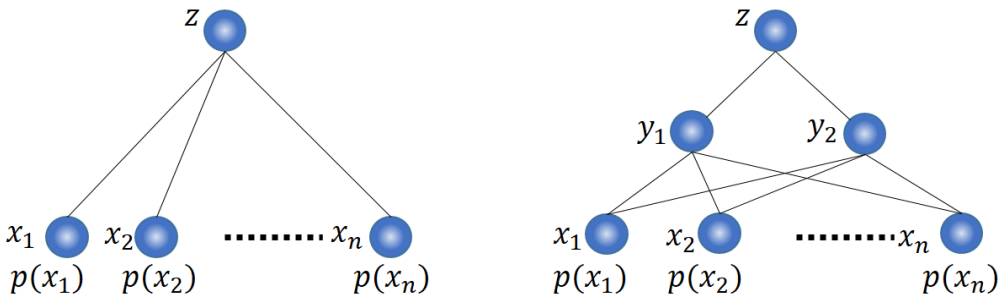


Figure 3.2: A network with two layers (left) and a network with one hidden layer (right).

In this section, we consider the problem of extracting conditional probabilities given a combination of some input nodes values. Suppose that z is the output neuron directly connected to n input neurons $x_1, \ldots, x_n$ (see Fig. 3.2, left). We assume that the input neurons obey independent distributions, meaning that the value $Pr(x_i = 1)$ depends only on i. We also assume that the representation of $Pr(x_i = 1)$ uses only $O(1)$ bits, as is the case for the uniform distribution $Pr(x_i = 1) = 0.5$. The *Probabilistic Rule Extraction* problem is to compute the probabilities $Pr(z = a)$, $Pr(z = a|x_i = b)$, $Pr(z = a|x_{i_1} = b_1, x_{i_2} = b_2)$, $\cdots$ for $a, b, b_i \in \{0, 1\}$, up to a fixed number of input nodes. The problem can be extended to neural networks having hidden layers (Fig. 3.2, right).

3.4.2 NP-hardness

We prove that probabilistic rule extraction is NP-hard, even for a neural network with one hidden layer.

Theorem 14. *Deciding whether $Pr(z = 1) > 0$ is NP-hard for a neural network with one hidden layer.*

Proof. The proof consists of a polynomial time reduction from 3SAT [GJ79]. Let $v_1, \ldots, v_N$ be the variables of the 3-SAT formula, and let $c_1, \ldots, c_L$ be its clauses, each clause being a logical OR of at most three literals, $c_i = \ell_{i_1} \vee \ell_{i_2} \vee \ell_{i_3}$ where ℓ_{i_j} is either v_{i_j} or $\bar{v}_{i_j}$. 3SAT is the problem of deciding whether or not there exists an

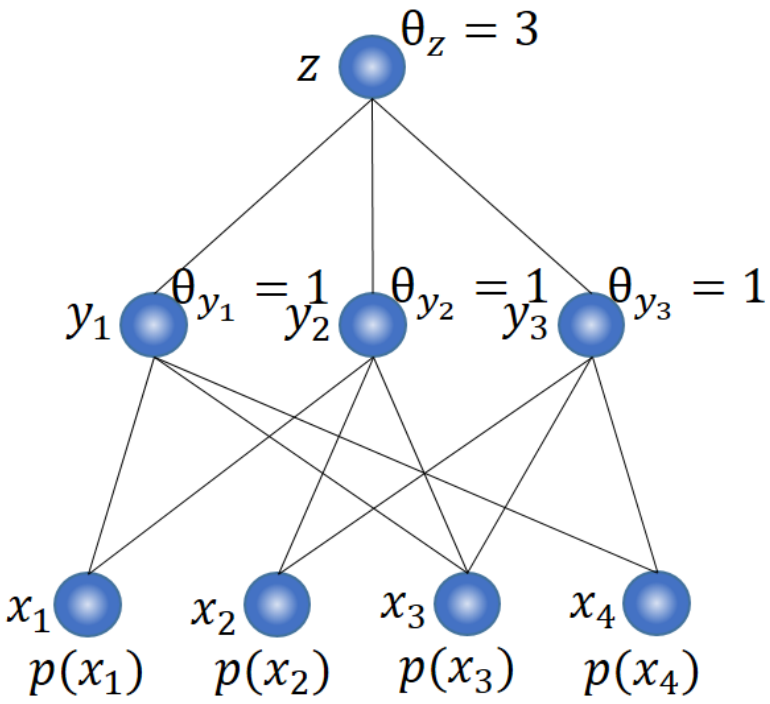


Figure 3.3: Reduction from a 3-SAT instance $\{v_1 \vee v_3 \vee v_4, v_1 \vee v_2 \vee v_3, v_2 \vee v_3 \vee v_4\}$ to probabilistic rule extraction.

assignment of 0-1 values to $v_1, \ldots, v_N$ which satisfies all the clauses (i.e., all clauses evaluate to 1).

From an instance of 3SAT, we construct a neural network as follows (see also Fig. 3.3).

The input nodes are $X = \{x_1, \ldots, x_N\}$, and the hidden layer nodes are $Y = \{y_1, \ldots, y_L\}$, where we assume the uniform 0-1 distribution for each of the x_i. Node y_i in the hidden layer corresponds to clause c_i, and its weights are $w_{i,i_j} = 1$ if $\ell_{i_j} = v_{i_j}$, and $w_{i,i_j} = -1$ if ℓ_{i_j} is $\bar{v}_{i_j}$, with $w_{i,k} = 0$ for $k \neq i_1, i_2, i_3$. The threshold of node y_i is $\theta_{y_i} = 1 - |\{\ell_{i_j} | \ell_{i_j} = \bar{x}_{i_j}\}|$. The weights assigned to node z are $w_1 = w_2 = \cdots = w_L = 1$ and its threshold is $\theta = L$.

It is straightforward to verify that $Pr(z = 1) > 0$ iff there exists an assignment such that all $c_1, \ldots, c_L$ evaluate to 1. The reduction clearly takes polynomial time. □

In this reduction the number of nodes in the hidden layer is not bounded, whereas the weights are restricted to be -1, 0, and 1. The next Theorem points out that the problem remains NP-hard even if the number of nodes in the hidden layer is restricted to be at most 2, provided the weights can be arbitrary integers.

Theorem 15. *Deciding whether $Pr(z = 1) > 0$ is NP-hard for a neural network with one hidden layer with two neurons if $O(n)$ bits integer weights can be assigned.*

Proof. As in the proof of the co-NP hardness of testing the equivalence of two threshold functions [MCCA18], we use a polynomial-time reduction from the subset sum problem. Recall that the subset sum problem asks: given a set of positive integers $A = \{a_1, a_2, \ldots, a_N\}$ and an integer b, is there a 0-1 assignment $(v_1, \ldots, v_N)$ such that $a_1v_1 + a_2v_2 + \cdots + a_Nv_N = b$?

From an instance (A, b) of the subset sum problem, we construct a neural network similar to the one used in the proof of Theorem 14, with only two nodes y_1 and y_2 in the hidden layer. The weights associated with these two nodes are $w_{i,j} = a_j,\ i = 1, 2,\ j = 1, \ldots, N$, and their thresholds are $\theta_1 = b$, and $\theta_2 = b + 1$. The weights of the output node z are $w_1 = 1$ and $w_2 = -1$, and $\theta_z = 1$.

Clearly $Pr(z = 1) > 0$ iff there exists a 0-1 assignment $(v_1, \ldots, v_N)$ such that $a_1v_1 + a_2v_2 + \cdots + a_Nv_N \geq b$ and $a_1v_1 + a_2v_2 + \cdots + a_Nv_N < b + 1$. Furthermore, the time taken by the reduction is polynomial in the total number of bits used to represent the instance of the Subset Sum problem. □

3.4.3 Probabilistic Rule Extraction from a Linear Threshold Function

Let us turn now to developing an algorithm for calculating $Pr\ (z = 1)$ given a linear threshold function. As a first step consider the case that the threshold function is a nested canalyzing function.

Let $ncf(\mathbf{x})$ be a nested canalyzing function on the variables $\mathbf{x}$ and let p_i denote the probability

$$p_i = Pr(x_i = 1). \tag{3.2}$$

We assume that a constant number of bits suffices to specify p_i. According to Lemma 9 we can assume that the threshold function is in canonical form, and therefore its nested canalyzing form contains no negated variables, and the variables appear ordered by non-increasing weights (cf. Algorithm 1). Here is the pseudo-code for calculating $Pr(ncf(\mathbf{x}) = 1)$.

Algorithm 4 Probability Extraction Algorithm for a nested canalyzing function

Input: A canonical threshold function in the form of a nested canalyzing function $ncf(x_i, \cdots, x_n)$; probabilities $p_j, j = i, \ldots, n$;
Output: $Pr(ncf(x_i, \ldots, x_n) = 1)$;
ExtractP$(ncf(x_i, \cdots, x_n))$:
if $i = n$ **then** return $p_n = 1$;
end if
if $ncf(x_i, \cdots, x_n) = x_i \wedge g(x_{i+1}, \cdots, x_n)$ **then**
return $p_i * ExtractP(g(x_{i+1}, \cdots, x_n))$;
else if $ncf(x_i, \cdots, x_n) = x_i \vee g(x_{i+1}, \cdots, x_n)$ **then**
return $p_i + (1 - p_i) * ExtractP(g(x_{i+1}, \cdots, x_n))$;
end if

Proposition 16. *A probabilistic rule can be extracted from a nested canalyzing function in polynomial time.*

Proof. The correctness of Algorithm 4 is self-evident. Since by assumption the probabilities p_i are represented with a constant number of bits, each call makes at most one multiplication, and the number of calls is linear in n, the algorithm works in time polynomial in n. □

Note that $Pr(z = 1 | x_{i_1} = a_{i_1}, \ldots, x_{i_k} = a_{i_k})$ can be computed by applying Algorithm 4 after setting $(x_{i_1}, \ldots, x_{i_k})$ to $(a_{i_1}, \ldots, a_{i_k})$.

Next, we present an algorithm for a general canonical threshold function. It is based on the following straightforward recursion.

$$Pr(f(x_1, \ldots, x_n) = 1) = p_1 Pr(f(x_1, \ldots, x_n) = 1 | x_1 = 1) + (1 - p_1)) Pr(f(x_1, \ldots, x_n) = 1 | x_1 = 0). \quad (3.3)$$

To implement the recursion efficiently we use dynamic programming, as in [MCCA18]. Given a canonical threshold function $w_1 x_1 + \cdots + w_n x_n \geq \theta$ with positive integer weights w_i and threshold θ, define $T(i, s)$ by

$$T(i, s) = Pr(\sum_{j=i}^{n} w_j x_j \geq s).$$

From recursion 3.3 we get a recursion for $T(i, s)$:

$$T(i, s) = p_i T(i + 1, s - w_i) + (1 - p_i) T(i + 1, s). \quad (3.4)$$

Note that the value we are after is $Pr(z = 1) = T(1, \theta)$; it is obtained by computing recursively, with memorization, the values $T(i, s)$, $i = 1, \ldots, n$, for appropriate

values of s. Since all weights are positive, the smallest and largest values of s of possible interest are 0 and θ. The table is initialized by $T(i,s) = 1$ if $s \leq 0$ and

$$T(n,s) = 1 \text{ if } s \leq 0,\ T(n,s) = p_n \text{ if } 0 < s \leq w_n,\ T(n,s) = 0 \text{ if } \ w_n < s. \quad (3.5)$$

Theorem 17. *Given a canonical threshold function the value $Pr(z = 1)$ can be computed in $O^*(n\theta)$ time.*

Proof. The correctness of the dynamic programming procedure is self evident.

The number of table entries $T(i,s)$ needed for the computation is $n\theta$. The size of each entry is polynomially bounded, so that the computations of Equations (3.4) and (3.5) take polynomial time. □

Since θ may be exponentially large in terms of the input size, this algorithm runs in pseudo-polynomial time, although if θ is polynomially large then so is the running time.

Here is an example illustrating the working of the algorithm. Suppose that we are requested to compute $Pr(2x_1 + 3x_2 + x_3 - 2x_4 \geq 2)$ where $Pr(x_i = 1) = 0.5$ for $i = 1, \ldots, 4$. The canonical form of the threshold function is $3x_1+2x_2+2x_3+x_4 \geq 4$. According to Equation (3.5) the initial values are

$$T(4,s) = 1 \text{ if } s \leq 0,\ \frac{1}{2} \text{ if } 0 < s \leq 1,\ 0 \text{ if } 1 < s.$$

A straightforward computation using Equation (3.4) yields

$$Pr(2x_1 + 3x_2 + x_3 - 2x_4 \geq 2) = T(1,4) = p_1 T(2,1) + (1 - p_1)T(2,4) = \cdots = \frac{9}{16}.$$

Only 8 values were computed, half of the table.

3.4.4 Probabilistic Rule Extraction: Network with One Hidden Layer

Here we extend the dynamic programming algorithm presented above to a neural network with one hidden layer; the next subsection describes how to handle multiple hidden layers. In these subsections n_i denotes the number of input nodes in layer i, with $n = n_1$ being the number of input nodes. The right side of Fig. 3.2 shows the structure of a network with one hidden layer.

Caution: In these two subsections we consider two or more functions simultaneously. Therefore we do not assume that all threshold functions are given in canonical form. Using the method of Lemma 9 we can still assume that the weights of the threshold functions are all non-negative; the price we pay is that the threshold functions are combinations of *literals* rather than of variables. Given an assignment **a**

to the variables $\mathbf{x}$ of a threshold function $w_1\ell_1 + \cdots + w_n\ell_n \geq \Gamma$, we will denote by $\ell_j(a_j)$ the value given to ℓ_j by the assignment $x_j = a_j$; for example, if $\ell_j = \overline{x_j}$ and $a_j = 1$ then $\ell_j(a_j) = 0$.

Consider to begin with the simplest case of a neural network having one hidden layer with two nodes y_1 and y_2. The values of y_1 and y_2 are determined by the threshold functions

$$w_{k,1}\ell_{k,1} + \cdots + w_{k,n}\ell_{k,n} \geq \Gamma_k, k = 1, 2, \tag{3.6}$$

with $w_{k,j} \geq 0, k = 1, 2,\ 1 \leq j \leq n$. The value of the output node z is 1 iff $w_{y_1}\ell_{y_1} + w_{y_2}\ell_{y_2} \geq T$ with $w_{y_1}, w_{y_2} \geq 0$, and $0 < T < w_{2,1} + w_{2,2}$ (otherwise the value is constant). For simplicity we will assume in the following that $\ell_{y_k} = y_k, k = 1, 2$; the other possibilities are handled similarly.

We describe next two ways to compute the probability $Pr(z = 1)$.

1. The first method starts by initializing $P_3(1)$ to 0. Then it considers all assignments $\mathbf{a}$ to the variables of the input layer, and for each it computes

   (a) the resulting assignment (b_1, b_2) to (y_1, y_2):

   $$b_k = \left(w_{k,1}\ell_{k,1}(\mathbf{a}) + \cdots + w_{k,n}\ell_{k,n}(\mathbf{a}) \geq \Gamma_k\right), k = 1, 2;$$

   (b) the resulting value of z: $c = (w_{y_1}\ell_{y_1}(b_1) + w_{y_2}\ell_{y_2}(b_2) \geq T)$;

   (c) adds $Pr(\mathbf{x} = \mathbf{a})$ to $P_3(1)$ if $c = 1$.

   Since the input node variables are independent $Pr(\mathbf{x} = \mathbf{a}) = \prod_{j=1}^{n} Pr(x_j = a_j)$. $Pr(z = 1)$ equals the final value of $P_3(1)$. The total computation time is $O(n2^n)$.

   This computation is easily generalized to a network with n_1 input nodes and a hidden layer containing n_2 nodes $\mathbf{y}$: (1a) computes the assignment $\mathbf{b}$ to $\mathbf{y}$ in time $O(n_1n_2)$, and (1b) computes $c = \left(\sum_{j=1}^{n_2} w_{y_j}\ell_{y_j}(b_j) \geq T\right)$. Thus the computation time is

   $$O(n_1n_2 2^{n_1}). \tag{3.7}$$

2. The basic idea of the second method is to compute the joint probabilities of the two nodes in the hidden second layer, $P_2(b_1, b_2) = Pr(y_1 = b_1, y_2 = b_2)$, $b_1, b_2 \in \{0, 1\}$. With these values in hand the probability that $z = 1$ is

   $$Pr(z = 1) = H(w_{y_1} - T)P_2(1, 0) + H(w_{y_2} - T)P_2(0, 1) + P_2(1, 1). \tag{3.8}$$

The method first computes the matrix $F(i, s_1, s_2)$, $1 \leq i \leq n$, $0 \leq s_k \leq \Gamma_k$, defined by

$$F(i, s_1, s_2) = Pr(\sum_{j=i}^{n} w_{1,j}\ell_{1,j} \geq s_1 \text{ and } \sum_{j=i}^{n} w_{2,j}\ell_{2,j} \geq s_2).$$

Once all of F has been computed, the value of $P_2(1,1)$ is given by $P_2(1,1) = F(1, \Gamma_1, \Gamma_2)$. Although the other values needed for Equation 3.8, $P_2(1,0)$ and $P_2(0,1)$, do not themselves appear in F, they can be derived from values that do appear: $P_2(1,*) = Pr(y_1 = 1) = P_2(1,1) + P_2(1,0) = F(1, \Gamma_1, 0)$, and $P_2(*,1) = F(1, 0, \Gamma_2)$. Note that the computation accessed all values $P_2(c_1, c_2)$, $c_1, c_2 \in \{0, 1, *\}$.

F can be computed using a recursion similar to the one for T.

$$\begin{aligned} F(i, s_1, s_2) = {} & p_i F(i+1, s_1 - \ell_{1,i}(1)w_{1,i}, s_2 - \ell_{2,i}(1)w_{2,i}) \\ & + (1 - p_i)F(i+1, s_1 - \ell_{1,i}(0)w_{1,i}, s_2 - \ell_{2,i}(0)w_{2,i}), \end{aligned} \tag{3.9}$$

with the initialization

$$\begin{aligned} F(n, s_1, s_2) = {} & p_n H((\ell_{1,n}(1)w_{1,n} - s_1)H((\ell_{2,n}(1)w_{2,n} - s_2) \\ & + (1 - p_n)H((\ell_{1,n}(0) - s_1)H((\ell_{2,n}(0) - s_2). \end{aligned} \tag{3.10}$$

In particular, $F(n, s_1, s_2) = p_n H(\ell_{1,n}(1)w_{1,n} - s_1) + (1 - p_n)H(\ell_{1,n}(0) - s_1)$ for $s_2 \leq 0$, and $F(n, s_1, s_2) = 1$ when both s_1 and s_2 are non-positive.

The time complexity of the resulting algorithm is pseudo-polynomial, $O^*(n\Gamma_1\Gamma_2)$.

To extend these ideas to a neural network with a single hidden layer containing n_2 nodes, define the matrix $F(i, s_1, \ldots, s_{n_2})$; it can be computed recursively using the analogs of Equations (3.9) and (3.10). From this matrix we derive the values $P_2(\mathbf{c})$ with $\mathbf{c} \in \{0, 1, *\}^{n_2}$ in n_2 stages, where at stage r we consider those $\mathbf{c}$ containing exactly r 0's. For such a $\mathbf{c}$ let j_r be the first index such that $c_{j_r} = 0$, and define $\mathbf{t}$ and $\mathbf{u}$ by $t_j = u_j = c_j$ if $j \neq j_r, 1 \leq j \leq n_2$ and $t_{j_r} = *, u_{j_r} = 1$. Then $P_2(\mathbf{c}) = P_2(\mathbf{t}) - P_2(\mathbf{u})$. Each such computation subtracts one already computed value from another and takes constant time. The initialization stage of the computation is to copy the values of $P_2(\mathbf{c})$ for $\mathbf{c} \in \{1, *\}^{n_2}$ from F: $P_2(\mathbf{c}) = F(1, \mathbf{t})$, where $t_j = \Gamma_j$ if $c_j = 1$, $t_j = 0$ if $c_j = *$, $1 \leq j \leq n_2$.

On completing the computation of P_2 from F we know all 3^{n_2} values $P(\mathbf{c})$ with $\mathbf{c} \in \{*, 0, 1\}^{n_2}$. Consequently the time to compute all of P_2 is $O(3^{n_2})$, and the overall computation time for this method is $O(n_1 \prod_{k=1}^{n_2} \Gamma_k + 3^{n_2})$. Thus this method is much faster than the first naive method, Equation (3.7), if $n_1 \gg \max\{\log_2 3 \cdot n_2, \sum_{k=1}^{n_2} \log_2 \Gamma_k\}$.

3.4.5 Probabilistic Rule Extraction: Network with Multiple Hidden Layers

To extend the approach of the previous subsection further to multilayer neural networks, we assume that our computation model is the real RAM (random access machine) [Sha78]. The real RAM is an idealized model in which it is assumed that any real number can be stored in a word and that each arithmetic operation takes constant time. This model has been widely used for analyzing algorithms in computational geometry [PS85]. In actual computers, real numbers are of course represented as finite precision floating point numbers. Therefore, when implemented on actual computers, our algorithms may output approximate probabilities.

Consider a neural network with L ($L \geq 4$) layers, the first one being the input layer, the last one consisting of the lone output node z, and the ith layer having n_i nodes for $i = 1, \ldots, L$. We briefly indicate how to generalize the algorithms of the previous sections to the current setting.

The first generalized method computes, for each assignment $\mathbf{a}$ to the variables of the input layer, the assignment generated to the variables of the $i+1$-st layer by the assignment that was generated to the variables of the i-th layer, $i = 1, \ldots, L-1$. If the resulting assignment to z is 1, then $Pr(\mathbf{a})$ is added to the current value of $P_L(1)$. At the completion of the computation $Pr(z = 1) = P_L(1)$. The total time taken by the computation is $O(2^{n_1} \sum_{i=1}^{L-1} n_i n_{i+1})$.

The second generalized method is a combination of the previous two methods: it begins by applying the second method to compute all joint probabilities for the first hidden layer, Next it essentially uses the first generalized method just described, but starts it from the second layer. Thus, for each assignment $\mathbf{b}$, whose probability $Pr(\mathbf{b})$ was just calculated, the method traces the assignment it generates to the variables of the i-th layer, $i = 3, \ldots, L-1$. If the resulting assignment to z is 1, then $Pr(\mathbf{b})$ is added to the current value of $P_L(1)$. Upon termination this method returns $P_L(1)$ as the value of $Pr(z = 1)$. Thus the running time is

$$O(n_1 \prod_{k=1}^{n_2} \Gamma_k + 3^{n_2} + 2^{n_2} \sum_{i=2}^{L-1} n_i n_{i+1}). \tag{3.11}$$

Then the following Theorem summarizes the foregoing description.

Theorem 18. *Given a neural network with L ($L \geq 4$) layers, suppose $n_1 \geq \max\{\log_2 3 \cdot n_2, \sum_{k=1}^{n_2} \log_2 \Gamma_k\}$. Then $Pr(z = 1)$ can be computed in time $O(n_1 \prod_{k=1}^{n_2} \Gamma_k + 3^{n_2} + 2^{n_2} \sum_{i=2}^{L-1} n_i n_{i+1})$.*

Theorem 11 means that the second method is at least as fast as the first naive enumeration based method if $n_1 \geq \max\{\log_2 3 \cdot n_2, \sum_{k=1}^{n_2} \log_2 \Gamma_k\}$ and is much faster

if $n_1 \gg \max\{\log_2 3 \cdot n_2, \sum_{k=1}^{n_2} \log_2 \Gamma_k\}$. Although many neurons are used in hidden layers in recent applications, there are several practical cases in which only a few neurons are used in hidden layers [ZT00] [MHZ+08]. In such cases, the second method might work much faster than the naive method. At least this result gives a theoretical advance over the native method.

The analyses above are all for binary classifications. It might be possible to extend this probabilistic rule extraction to multiclass classification. A possible way is to compute $P(z_1 = 1), P(z_2 = 1), \ldots, P(z_p = 1)$ separately for neural networks with p outputs.

3.5 Computational Experiments

3.5.1 Computational Experiments with Boolean Rule Extraction

In addition to the theoretical analyses of the Boolean rules extraction, we performed computational experiments to demonstrate its potential usefulness.

Although the algorithms presented in Section 3.3 are applicable only to single neurons, they can be extended to neural networks with hidden layers, using a strategy similar to the one given in [Tsu00]. Namely, starting with the first layer, and proceeding layer by layer, the Boolean functions for all neurons in a layer are computed. The Boolean function so obtained for the single neuron in the final layer is the function for the output .

To study the performance of our proposed method, different neural networks were trained on one synthetic dataset and two real datasets, all with sigmoid functions using the Adam optimizer in PyTorch version 1.0.0 [PGCC17]. Binary cross-entropy is selected as our loss function. Since we want to see how well our Boolean rules approximate the neural networks we used in each case all of the dataset after samples with missing attributes were removed. The training was terminated once the training loss was less than 0.02.

In order to obtain concise Boolean rules, we ceased the generation of terms at the third order, where a term is of order i if it has the form $\prod_{k=1}^{i} x_k$. Therefore, the rules contain only first, second and third order terms. Technically this meant adding to Algorithms 1 and 3 an order parameter $r \geq 0$ which indicates the highest order to be generated by the algorithm. For example, the heading of Algorithm 1 was changed to $ExtractNC(w_i x_i + \cdots + w_n x_n \geq \theta;\ r)$, to its body we added another base case "**if** $r = 0$ return 0" as a first statement, and the order parameter was decreased as necessary in other statements. For simplicity in this part the procedure $ExtractMF$ was deleted from Algorithm 3. Here is an example of this order-capped Boolean function generation: the threshold function $5x_1 + 3x_2 + 3x_3 + 2x_4 + 2x_5 \geq 10$ with order

parameter $r = 3$ yields the function $x_1 \wedge ((x_2 \wedge (x_3 \vee x_4 \vee x_5)) \vee (x_3 \wedge (x_4 \vee x_5)))$.

1. *Regulation of the Mammalian Cell Cycle*

   As mentioned in Section 3.2.2, many biologically important functions are reported to be nested canalyzing. For example, the following rule is used in a logical model of the mammalian cell cycle network [FNCT06]:

   $$UbcH10 : (\overline{Cdh1}) \vee (Cdh1 \wedge Ubc \wedge (Cdc20 \vee CycA \vee CycB))$$

   In order to examine whether our algorithm can reconstruct this rule from samples, we generated at random a dataset of 16384 samples with the 15 binary attributes $Cdh1$, Ubc, $Cdc20$, $CycA$, $CycB$ and 10 noise attributes. Classes were $UbcH10$ and $\overline{UbcH10}$.

   These data were then used to train each one of the three neural networks shown in Fig. 3.4.

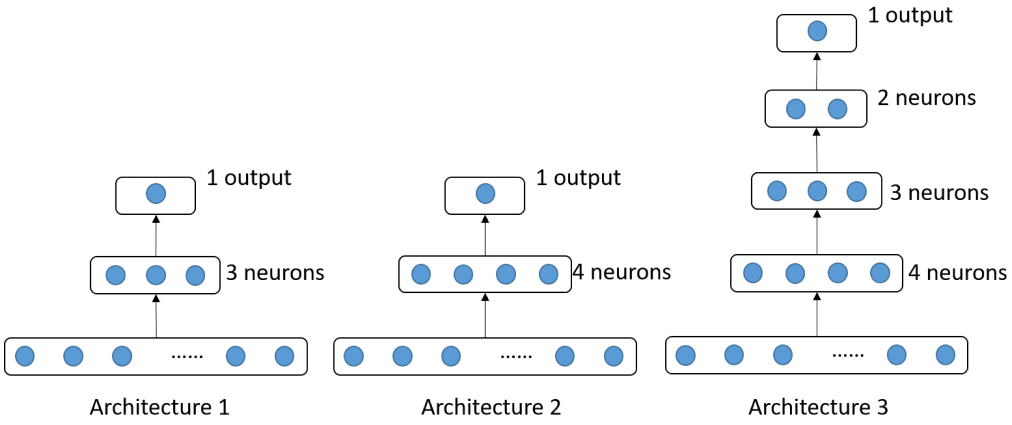


Figure 3.4: Architectures of the Neural Networks.

   Applying our Boolean rule extraction algorithm to the resulting trained neural networks yielded the correct nested canalyzing function for $UbcH10$ in each case.

2. *The Votes Data*

   The votes data set consists of 232 out of the 435 voting records of congressmen in 1984, as they appear in the UCI Machine Learning Repository [DG17], after entries with missing data were removed. Each sample has 16 binary attributes (the votes of the congressman on 16 issues). Classes are *Democrat*

and *Republican*. Each extracted Boolean variable is of the form 'a:v', meaning that a has value v.

We trained each of the three neural networks depicted in Fig. 3.4 on these data. Then we tested our Boolean rule extraction algorithm on each of the resulting neural networks. In all three cases the following Boolean rule was obtained:

$$\begin{aligned} \textit{Democrat} : (\text{physician-fee-freeze:}n) \vee ((\text{adoption-of-the-budget-resolution:}y) \\ \wedge (\text{anti-satellite-test-ban:}n) \\ \wedge (\text{synfuels-corporation-cutback:}y)) \end{aligned}$$

The accuracy of the rule is 98.3%. This rule is the same as the one obtained for the *votes* data by Tsukimoto [Tsu00] (with initial weight parameters 1).

3. *The Mushroom Data*

   This dataset, also taken from the UCI Machine Learning Repository [DG17], consists of 5644 mushrooms (after removal of samples with missing values), each with 22 discrete physical characteristics. Classes are *edible* and *poisonous*. The following Boolean rule was obtained for all three networks depicted in Fig. 3.4:

$$\begin{aligned} \textit{edible} : \neg(\text{spore-print-color:green}) \wedge ((\text{odor:none}) \\ \vee (\text{odor:almond}) \\ \vee (\text{odor:anise})) \end{aligned}$$

   The accuracy of this Boolean rule is 99.7%.

   The rule that Tsukimoto [Tsu00] extracted for the mushroom data is:

$$\begin{aligned} \textit{edible} : (\text{gill-size:board}) \wedge ((\text{odor:none}) \\ \vee (\text{odor:almond}) \\ \vee (\text{odor:anise})) \end{aligned}$$

   and its 95.6% accuracy for the 4062 mushrooms that Tsukimoto used is not as good as ours. Its accuracy on the full dataset of 5644 mushrooms is even slightly less, 94.5%.

Applying the C4.5 algorithm of [Qui93] to our dataset yields the following rule:

$$(\text{odor:none}) \vee (\text{odor:almond}) \vee (\text{odor:anise}) \rightarrow \textit{edible}$$

with an accuracy of 98.7%, slightly less than ours.

The best rule that extracted from the mushroom dataset is [Ish00]:

$$\begin{aligned} \textit{edible} : ((\text{gill-size:board}) \wedge \neg(\text{stalk-surface-below-ring:scaly}) \wedge \neg(\text{population:clustered})) \wedge \\ \neg(\text{spore-print-color:green}) \wedge ((\text{odor:none}) \\ \vee (\text{odor:almond}) \\ \vee (\text{odor:anise})) \end{aligned}$$

The accuracy of this rule is 100%. Therefore, the method by Ishikawa [Ish00] might be useful in practice. However, there is no theoretical guarantee on what kinds of rules are extracted by his method. Therefore, we continued our experiments on our proposed methods.

We tested architecture 1 on this dataset using mean square error with different values. When the loss is 0.0228, we obtained the same rule as the experiments using binary cross-entropy loss even if we ceased the generation of terms at the sixth order. We continued the training until the loss achieved 0.00258. We got the same rule as previous. However, when the loss achieved 0.00086, we obtained a rule with more than 20 literals even we ceased the generation of terms at the third order. These experiments indicate that use of very small loss values might lead to overfitting and generation of complex rules. Therefore, the stopping condition should be carefully chosen in practice.

The above experiments are all focus on binary classifications. It might be possible to extend this Boolean rule extraction to multiclass classifications. One way is to consider neurons connected to each output as individual neural networks. Then Boolean rules can be obtained for each output.

3.5.2 Computational Experiments with Probabilistic Rule Extraction

This subsection has three parts. First, we examine the differences between the rules extracted by our algorithm as compared with the rules extracted directly from the trained sigmoid network. Second, we demonstrate that our algorithm

can extract probabilistic rules with very small probabilities through experiment. Finally, we study the possibility of extracting important attributes from trained neural networks.

The probabilistic rule extraction algorithm outputs accurate probabilities for neural networks consisting of linear threshold functions. However, in practice sigmoid functions are often used in neural networks in order to enable training with sample data. For such networks, our algorithm outputs approximate probabilities. We report here on computational experiments aimed at examining the differences between the rules extracted by our algorithm as compared with the rules extracted directly from the trained sigmoid network.

In the experiments we trained a neural network consisting of ten input neurons, one output neuron, and one hidden layer with two neurons, all with sigmoid activation functions using the Adam optimizer in PyTorch version 1.0.0 [PGCC17]. Three sets of training samples with 500, 1024, 1024 samples respectively, ten attributes and one output label (all binary valued), were randomly generated from the following three Boolean functions:

1. 5-out-of-10 majority function ($Maj_5(x_1, \ldots, x_{10})$),

2. conjunction of two majority functions ($Maj_2(x_1, \ldots, x_5) \vee Maj_2(x_6, \ldots, x_{10})$),

3. $Maj_2(x_1, \ldots, x_5) \vee (x_6 \vee (x_7 \wedge (x_8 \vee (x_9 \wedge x_{10}))))$,

where we used $Pr(x_i = 1) = 0.5$ for all $i = 1, \ldots, n$. The weights and thresholds of the resulting trained network were converted to integers by the quantization method shown in Fig. 3.5.

Here w_{max} and w_{min} represent the maximum and the minimum values among $w_1 \cdots w_n$ and b, respectively. The constant d was set to 200.

The quantized weights were used to construct a linear threshold network, the *derived linear threshold network*. From this network five probabilistic rule values ($P_1, \ldots, P_5$, to be shown in the following paragraphs) were extracted by the Probabilistic Rule Extraction Algorithm, PE, as follows. Using the threshold functions (3.6) for the hidden nodes, the matrix F was computed by means of the recursion (3.9) and the initialization (3.10); the value of the target is deduced from F by equation (3.8). We compare these target values with the exact probability computed from the original Boolean functions and the ones obtained by the following simple sampling-based algorithm for extracting probabilistic rules directly from the trained sigmoid network.

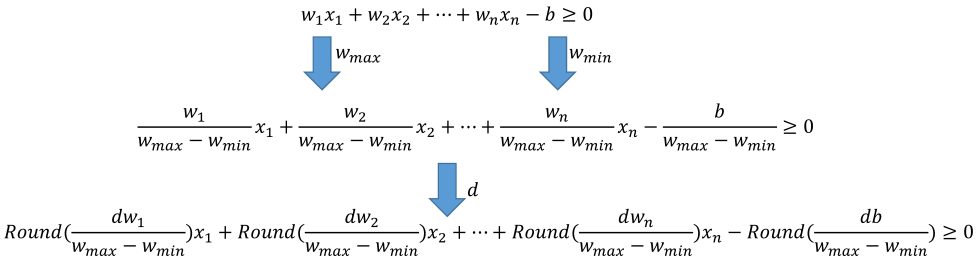


Figure 3.5: Quantization of weights and threshold.

Algorithm 5 Probabilistic Rule Extraction by Sampling

$count = 0$
for $i = 1$ to m **do**
  generate a random 0-1 assignment to the variables in the input layer;
  compute the 0-1 value of node z output by the trained neural network;
  if $z = 1$ **then** $count++$;
end for
output $count/m$ as an approximation of $Pr(z = 1)$

For the 5-out-of-10 majority function the targets were $P_1 = Pr(z = 1|x_1 = 1)$, $P_2 = Pr(z = 1|x_1 = 1, x_2 = 1)$, $P_3 = Pr(z = 1|x_1 = 1, x_2 = 1, x_3 = 0)$, $P_4 = Pr(z = 1|x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0)$, and $P_5 = Pr(z = 1|x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 1)$, and for each target the comparison value was obtained by running Algorithm 5 100 times with $m = 100$ samples.

The results, displayed in Table 3.1, show that the values output by our algorithm are essentially exact, whereas the results of Algorithm 5 depend on the samples that were generated. The small differences between the exact probabilities and those by PE are due to that the latter ones were computed using the linear threshold network obtained from the trained neural network.

Algorithms	Parameters	P_1	P_2	P_3	P_4	P_5
exact probability		0.74609	0.85547	0.77344	0.65625	0.8125
PE		0.7461	0.8555	0.7734	0.6563	0.8125
PA	Mean	0.7488	0.8601	0.7727	0.6588	0.8084
	STD	0.0451	0.0347	0.0423	0.0507	0.0314

Table 3.1: Results for the 5-out-of-10 majority function. PE is the probabilistic rule extraction algorithm. PA refers to Algorithm 5. Consult the text for the definitions of $P_1, \ldots, P_5$.

For the second Boolean function the targets were $P_1 = Pr(z = 1|x_1 = 1)$, $P_2 = Pr(z = 1|x_1 = 1, x_2 = 0)$, $P_3 = Pr(z = 1|x_1 = 1, x_2 = 0, x_3 = 1)$, $P_4 = Pr(z = 1|x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0)$, and $P_5 = Pr(z = 1|x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 0, x_5 = 1)$. The results, displayed in Table 3.2, show that again the values output by our algorithm are essentially exact, and those of Algorithm 5 depend on the set of samples.

Algorithms	Parameters	P_1	P_2	P_3	P_4	P_5
exact probability		0.76172	0.71094	0.8125	0.8125	0.8125
PE		0.7617	0.7109	0.8125	0.8125	0.8125
PA	Mean	0.7631	0.7088	0.8129	0.8155	0.8121
	STD	0.0437	0.0401	0.0380	0.0373	0.0383

Table 3.2: Results for the conjunction of two 2-out-of-5 majority functions. PE is the probabilistic rule extraction algorithm. PA refers to Algorithm 5. Consult the text for the definitions of $P_1, \ldots, P_5$.

For the third Boolean function the targets were $P_1 = Pr(z = 1|x_1 = 0)$, $P_2 = Pr(z = 1|x_1 = 0, x_2 = 1)$, $P_3 = Pr(z = 1|x_1 = 0, x_2 = 1, x_3 = 0)$, $P_4 = Pr(z = 1|x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0)$, and $P_5 = Pr(z = 1|x_1 = 0, x_2 = 1, x_3 = 0, x_4 = 0, x_5 = 1)$. The results, displayed in Table 3.3, show that our algorithm gives more accurate values than PA gives.

Algorithms	Parameters	P_1	P_2	P_3	P_4	P_5
exact probability		0.89258	0.95703	0.91406	0.82813	1
PE		0.8828	0.9570	0.9141	0.8281	1
PA	Mean	0.8802	0.9602	0.9157	0.8249	1.0
	STD	0.0347	0.0196	0.0305	0.0385	0.0

Table 3.3: Results for $Maj_2(x_1, \ldots, x_5) \vee (x_6 \vee (x_7 \wedge (x_8 \vee (x_9 \wedge x_{10}))))$. PE is the probabilistic rule extraction algorithm. PA refers to Algorithm 5. Consult the text for the definitions of $P_1, \ldots, P_5$.

One advantage of our algorithm is that it is deterministic, so that its results do not depend on random samples. Another advantage is that it can extract probabilistic rules with very small probabilities.

In order to verify this property, we performed the following experiment. We designed a neural network with one hidden layer of 2 neurons and sigmoid activation functions for the Boolean function $x_1 \wedge \cdots \wedge x_{20}$ since it is hard to train a neural network for this function. From the designed network we constructed the derived threshold network with integer weights and threshold, and extracted from it $Pr(z = 1)$ as 9.53674×10^{-7}, which is in fact the exact value of that probability. In contrast, 100 trials of Algorithm 5 on the sigmoid network, each with 100 samples, were unable to find this non-zero probability, because the number of samples is too small. Possibly using a large number of samples will yield a non-zero probability, but it is difficult to estimate in advance the required number of samples.

Finally, we examine the possibility of extracting important attributes from trained neural networks using our probabilistic rule extraction algorithm. In this experiment, we train a neural network with n binary inputs x_i for $1 \leq i \leq n$ and one binary output z. We would like to compute $p(z = 1|x_i = 0)$ and $p(z = 1|x_i = 1)$ for all x_i using our probabilistic rule extraction algorithm. We assume the trained neural network has one hidden layer (3 hidden neurons).

We tested our proposed algorithm on three different datasets.

1. The Votes Data

   This is the same dataset we considered in Sect. 3.5.1. The training was terminated once the loss was less than 0.02. The results in Fig. 3.6 show that physician-fee-freeze:n is significantly more important than others. In fact, the accuracy of using only physician-fee-freeze:n is 96.9%.

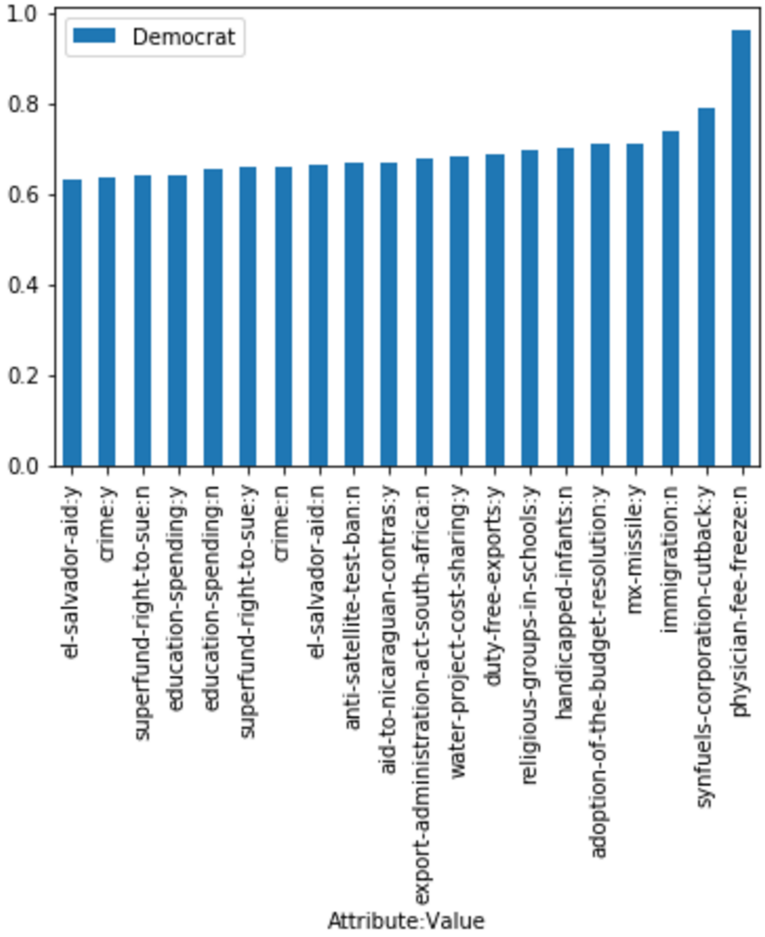


Figure 3.6: Rules with the Top 20 Highest Probabilities (Vote)

2. The Moral Reasoner Data

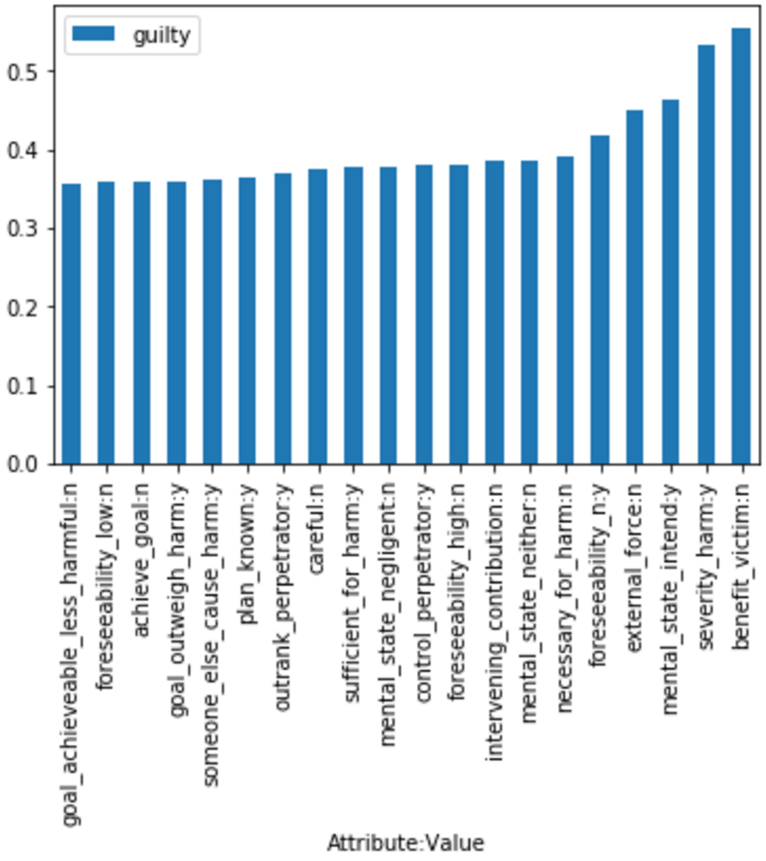


Figure 3.7: Rules with the Top 20 Highest Probabilities (Moral Reasoner)

This dataset, also part of the UCI Machine Learning Repository [DG17], contains 202 samples each having 23 attributes, of which 21 are binary. The other two discrete attributes are mental_state : $neither/negligent/reckless/intend$ and foreseeability : $n/low/high$. The classes are *guilty* and *not guilty*.

We consider mental_state : $neither/negligent/reckless/intend$ as 4 binary attributes mental_state_neither : n/y, mental_state_negligent : n/y,

mental_state_reckless : n/y and mental_state_intend : n/y. Discrete attribute foreseeability : $n/low/high$ can be converted to 3 binary attributes in a similar way. Therefore there are 28 binary attributes. Training was halted once the training loss fell below 0.02.

It is fairly easy to construct from a few of the attributes that predict "guilty" with highest probabilities, as shown in Fig. 3.7, a rule with high accuracy.

The following rule, for example, has an accuracy of 96%.

$$guilty : (benefit\_victim : n) \wedge (external\_force : n) \wedge (severity\_harm : y)$$

Interestingly, although the attribute $mental\_state\_intend : y$ has higher probability than $external\_force : n$, it does not participate in this rule.

The above results show that if the training loss is small, that is, the neural network approximates the corresponding dataset well, it is possible to obtain important rules through the probabilistic rule extraction algorithm. Next, we test whether it is possible to obtain some important rules when the training loss is not so small.

3. The Chess (King-Rook vs. King-Pawn) Data

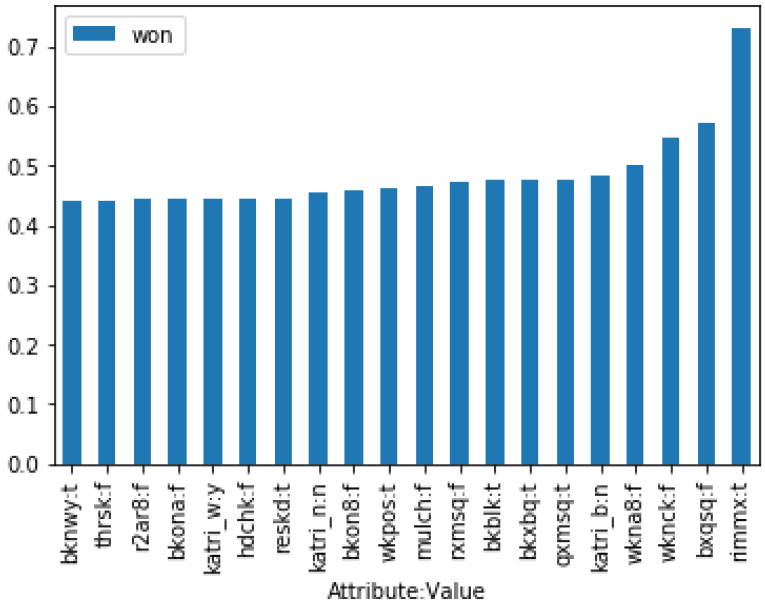


Figure 3.8: Rules with the Top 20 Highest Probabilities (Chess)

This dataset, also from the UCI Machine Learning Repository [DG17], includes 3196 samples. Each sample consists of the description of a legal initial board for this chess endgame and whether it is won by white, assuming optimal play by both sides. There are 36 discrete attributes of which 35 are binary. The discrete attribute $katri : b/n/w$ was converted into 3 binary attributes $katri\_b : n/y$, $katri\_n : n/y$, $katri\_w : n/y$. So there are 38 binary attributes. Classes are *won* and *nowin*. The training process is terminated when the training loss is 0.02. Fig. 3.8 shows

rules with the top 20 highest probabilities. Among these rules, we found that the following Boolean rule achieved an accuracy of 90.4%.

$$won : (rimmx : t) \vee (wknck : f) \wedge (bxqsq : f)$$

3.6 Discussions

In this study we presented several theoretical results for extracting knowledge from trained neural networks.

We described first an algorithm for extracting a Boolean function from a trained neural network. The algorithm is efficient if the linear threshold functions in the neural network can be represented as nested canalyzing functions, k-out-of-n majority functions, or some combinations of the two. If that is not the case, or if the neural network has hidden layers, the extracted Boolean function may turn out to be very large.

In order to cope with hidden layers, we limited the generation of rule terms to the third order. Computational experiments show that even if there are three hidden layers, the extracted Boolean functions approximate the corresponding neural networks well. Since banning high-order terms may not be appropriate in some applications, one important future direction of work is to obtain small size Boolean functions by combining our algorithm with simplification algorithms for Boolean functions such as [EBM09]. Although this modification may lead to generation of simple and understandable rules, the computational complexity in the first phase remains the same. Consequently, reduction of the computational complexity of the first phase is another important future work.

We also proposed an algorithm that extracts probabilistic relations between the input values and the output value in the form of conditional probabilities. Although this problem is NP-hard in general, our proposed algorithm works in pseudo-polynomial time if the number of neurons in hidden layers is bounded by a constant. Compared with a naive sampling-based algorithm, our algorithm has the advantages that it is deterministic, it outputs the exact probabilities if linear threshold functions are assigned to all nodes, and it does not miss rare relations. The potential usefulness of the algorithm was demonstrated by means of computational experiments. One disadvantage of this algorithm is that it needs to perform an exhaustive search for the second layer (i.e., the first hidden layer). Therefore, this algorithm is not practical if there are many neurons in the second layer and thus some improvements should be done. Use of sparse modeling [RG14] might be useful to find important input attributes without exhaustive search. Therefore, incorporation of sparse modeling into our proposed methods is important future

work. Another problem is that we assume that the input neurons obey independent distributions. Although computational experiments show the effectiveness of this algorithm under the independence assumption, dependencies between input attributes is almost inevitable in real world data. It might be useful to study the quantitative relation between the difference and the degree of dependencies. Thus we leave it as an open problem.

Chapter 4

ReCGBM: a Gradient Boosting-based Method for Predicting Human Dicer Cleavage Sites

4.1 Background

Human Dicer is an RNase III enzyme that cleaves double-stranded RNA (dsRNA) and pre-miRNA into short small interfering RNA and microRNA (miRNA), respectively. It consists of six domains: Helicase, DUF283, PAZ, RNase IIIa, RNase IIIb, and dsRBD. Among these domains, the RNase IIIa domain and RNase IIIb domain cleave the 3p-arm and 5p-arm of a pre-miRNA, leading to two miRNAs.

MicroRNA (miRNA) is a class of 20-22nt long, noncoding RNA molecules. They play an important role in the posttranscriptional regulation of gene expression. Usually, one miRNA can regulate the expressions of several proteins. They are necessary for a myriad of cellular processes, such as cell differentiation, cell cycle progression, and apoptosis [TOB12]. Several studies [IFL$^+$05, TKY$^+$04, HJL$^+$05] show that miRNAs are related to different types of cancers such as breast, lung, and thyroid cancers. Understanding how Dicer specifically selects cleavage sites may help us interpret the effects of mutations in miRNA coding genes [GMUTN$^+$19]. Therefore, it is of great interest to investigate how Dicer selects cleavage sites from the 3p-arm and the 5p-arm of a pre-miRNA.

Recently, machine learning-based approaches such as support vector machine [WTR06, WTR07, OSM$^+$11, PLN$^+$10], support vector regression [STS$^+$10, STP$^+$12, WZT$^+$14], deep neural networks [SS16, LYD$^+$19] and conditional random fields [FZS13] have been widely used for cleavage site predictions. However, these methods

mainly aim at predicting protein cleavage sites. To predict human Dicer cleavage sites, different feature encoding schemes and feature extraction methods are needed.

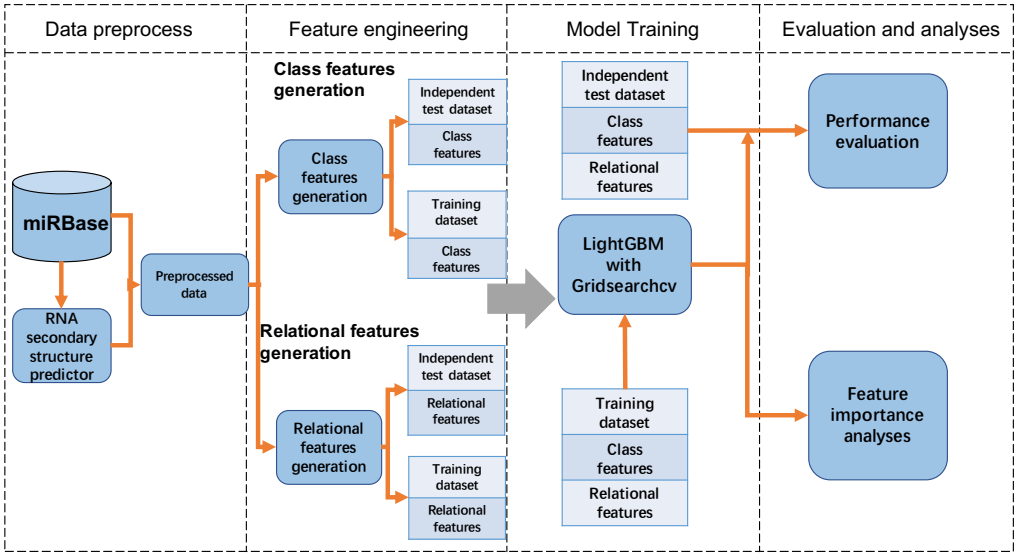
There are some existing studies on human Dicer cleavage sites. Ahmed et al. [AKR13] developed an SVM-based model (PHDCleav) of Dicer cleavage site prediction. The inputs to this model are extracted from pre-miRNA nucleotide sequences with loop/bulge structures, and the output is whether an input pattern is a correct Dicer cleavage site. They demonstrated this method outperformed other approaches such as Random Forest, CART, and Naïve Bayes by computational experiments. Bao et al. [BHA16] combined the loop/bulge size with pre-miRNA nucleotide sequences as inputs and proposed another SVM-based prediction model (LBSizeCleav). However, there are some shortcomings with these methods. First, they only considered each sequence with its loop/bulge independently. Second, their models are not explainable.

To address the above issues, we propose an explainable predictor based on the gradient boosting machine [Fri01]—ReCGBM. Our main contributions include: (i) extract relational features to combine each sequence and its complementary strand; (ii) design class features through affinity propagation, and (iii) identify some rules from the feature importance of ReCGBM. We summarize the design and evaluation process of ReCGBM in Figure 4.1a.

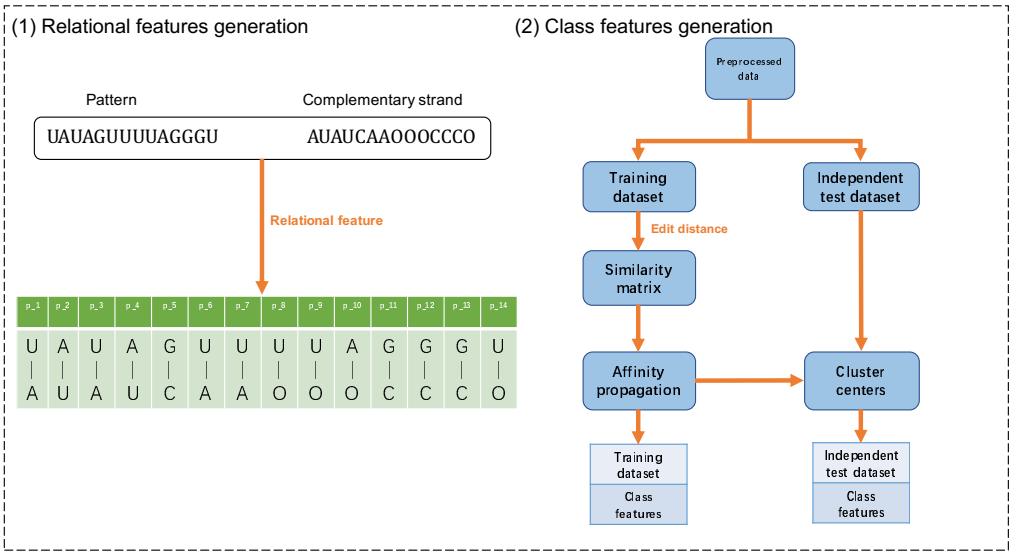
4.2 Methods

4.2.1 Data Preparation

In this study, we extracted the cleavage pattern and non-cleavage pattern from each pre-miRNA sequence. First, we collected 956 validated pre-miRNA sequences from miRBase (Version 22.1) [GJSvDE07]. To obtain the structural information for each pre-miRNA sequence, we employed quickfold [MZK] and RNAFold from ViennaRNA [Hof03] to generate RNA secondary structure. We chose these two RNA secondary structure prediction methods because both are powerful tools for RNA secondary structure prediction. Besides, previous methods like PHDCleav and LBSizeCleav all employed these tools to predict RNA secondary structures. Then, we generated a cleavage pattern for each pre-miRNA sequence. Each cleavage pattern consisted of a 14 nt long sequence with the cleavage site located at the center. Finally, we extracted each non-cleavage pattern that was a 14 nt long sequence with the center 6 nt away from the corresponding cleavage site. Figure 4.2 illustrates how to obtain cleavage pattern and non-cleavage pattern. In this figure, the cleavage pattern of 5p-arm is the sequence between the two red bars in the structure of the pre-miRNA sequence, which is 'UAUAGUUUUAGGGU'. The non-cleavage pattern of 5p-arm is 'AGGUUGUAUAGUUU' according to our selection rules.



(a)



(b)

Figure 4.1: Flowchart of the data preprocessing, feature extraction, model training and evaluation of the developed ReCGBM approach. (a) shows the design and evaluation process of ReCGBM. (b) describes how to generate relational features and class features.

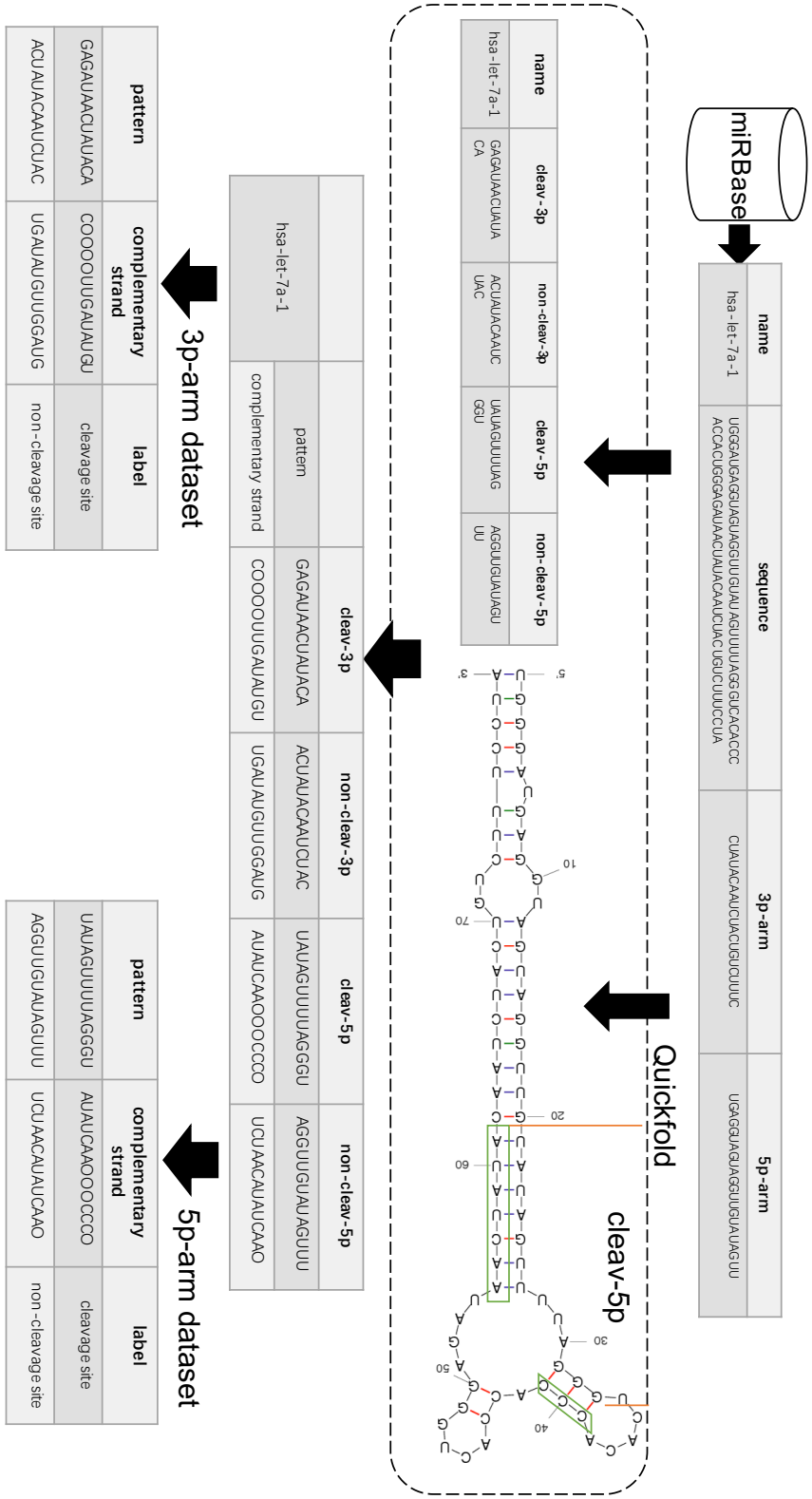


Figure 4.2: An example of data preprocessing. The sequence 'UAUAGUUU-UAGGGU' between the two red bars in the RNA structure represents the cleavage pattern of the 5p-arm. The complementary strand of this cleavage pattern is 'AUAUCAAOOOCCCO', which can be constructed by the sequences in the green boxes and loops/bulges.

To incorporate structural information of each pattern, we first obtained the structure of each pre-miRNA through quickfold or RNAFold. Then, we extracted the complementary strands for each cleavage pattern and non-cleavage pattern from the structural information of each pre-miRNA. Notice that if a nucleotide in a pattern did not have a complementary nucleotide according to the structural information, we would define it as a 'loop/bulge'. To combine 'loop/bulge' in our data, we represented a 'loop/bulge' as 'O' and considered it as a complementary nucleotide of the corresponding nucleotide in a pattern. In Figure 4.2, the complementary strand of the cleavage pattern of 5p-arm is given by the complementary nucleotides in the green boxes and loops/bulges ('O'). The complementary strand of the non-cleavage pattern of 5p-arm is given in the same way.

An example of data preprocessing is shown in Figure 4.2. Since structural information was generated through quickfold or RNAFold, four datasets were obtained after the preprocessing: 3p-arm with quickfold structure, 5p-arm with quickfold structure, 3p-arm with RNAFold structure and 5p-arm with RNAFold structure.

4.2.2 Relational Features

In previous studies [AKR13, BHA16], each pattern and its complementary strand were encoded by one-hot encoding separately. However, nucleotides in a pattern may also form a base pair with nucleotides in the corresponding complementary strand.

To better encode the relation between each pattern and its complementary strand, we considered relational features. For example, given a pattern 'UAUAGUUUUAGGGU' and its complementary strand 'AUAUCAAOOOCCCO', the relational features can be obtained according to (1) shown in Figure 4.1b. An advantage of relational features is that it offers the important base-pairing information between each pattern and its complementary strand.

4.2.3 Class Features

Previous methods [AKR13] and [BHA16] considered each input independently to make predictions. However, similar inputs may lead to the same prediction outputs. In this study, we made an assumption that similar inputs will lead to the same prediction results and accordingly designed the class feature, which assigned similar inputs to the same class.

To obtain class features, we first defined the pairwise similarities between different inputs based on the edit distance (Figure 4.3). Then, an unsupervised learning method — affinity propagation [FD07] was used to cluster the inputs to obtain the class features.

		U	A	U	G	G	U	U	U	A	G	A	G	U	U
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
U	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13
A	2	1	0	1	2	3	4	5	6	7	8	9	10	11	12
U	3	2	1	0	1	2	3	4	5	6	7	8	9	10	11
A	4	3	2	1	1	2	3	4	5	5	6	7	8	9	10
G	5	4	3	2	1	1	2	3	4	5	5	6	7	8	9
U	6	5	4	3	2	2	1	2	3	4	5	6	7	7	8
U	7	6	5	4	3	3	2	1	2	3	4	5	6	7	7
U	8	7	6	5	4	4	3	2	1	2	3	4	5	6	7
U	9	8	7	6	5	5	4	3	2	2	3	4	5	5	6
A	10	9	8	7	6	6	5	4	3	2	3	3	4	5	6
G	11	10	9	8	7	6	6	5	4	3	2	3	3	4	5
G	12	11	10	9	8	7	7	6	5	4	3	3	3	4	5
G	13	12	11	10	9	8	8	7	6	5	4	4	3	4	5
U	14	13	12	11	10	9	8	8	7	6	5	5	4	3	4

Figure 4.3: An example of calculating the edit distance by matrix.

Edit Distance

In order to measure the pairwise similarities between different sequences, we used the edit distance [Lev66].

Given two strings A and B, suppose that the lengths of A and B are $|A|$ and $|B|$, respectively. The edit distance between A and B is given by $D_{edit}(|A|, |B|)$ as follows

$$D_{edit}(i,j) = \begin{cases} max(i,j), & \text{if } min(i,j) = 0; \\ min \begin{cases} D_{edit}(i-1,j) + 1 \\ D_{edit}(i,j-1) + 1 \\ D_{edit}(i-1,j-1) + 1_{a_i \neq b_j} \end{cases} & \text{otherwise.} \end{cases}$$

where i represents the first i characters of A and j represents the first j characters of B, respectively, where $i, j \geq 1$.

$1_{a_i \neq b_j}$ is an indicator function where

$$1_{a_i \neq b_j} = \begin{cases} 0 & \text{if } a_i = b_j; \\ 1 & \text{otherwise.} \end{cases}$$

Since our inputs are 14 nt RNA sequences, the similarity between two sequences can be calculated by the edit distance. For example, given two sequences 'UAUAGUUUUAGGGU' and 'UAUGGUUUAGAGUU', the edit distance between them can be calculated through a distance matrix (Figure 4.3).

As Figure 4.3 shows, the edit distance between these two sequences is 4.

In this study, each input consisted of a pattern and its complementary strand. Therefore, the similarity between the two inputs E and F can be defined as follows:

$$D_{similar}(E, F) = D_{edit}(|E_1|, |F_1|) + D_{edit}(|E_2|, |F_2|)$$

where E_2 and F_2 are the complementary strands of E_1 and F_1 respectively.

Given a dataset that includes n samples, we define a $n \times n$ similarity matrix S where the entry in the ith row and jth column $s(i, j) = -d_{i,j}$. Notice that $d_{i,j}$ denotes the similarity $D_{similar}(i, j)$ between the ith training sample and the jth training sample.

Affinity Propagation

Affinity Propagation [FD07] is a clustering algorithm based on message passing. It identifies classes of similar inputs. Given n data points $x_1, \ldots, x_n$, the algorithm works as follows:

- Define an $n \times n$ similarity matrix S with $s(i, j) = -d_{i,j}$ for $1 \leq i \leq n, 1 \leq j \leq n$. $d_{i,j}$ is the distance between x_i and x_j;
- Define an $n \times n$ responsibility matrix R with $r(i, j) = 0$ for $1 \leq i \leq n, 1 \leq j \leq n$;
- Define an $n \times n$ availability matrix A with $a(i, j) = 0$ for $1 \leq i \leq n, 1 \leq j \leq n$;
- Iteratively execute the follow steps:

  1. Responsibility updates:

  $$r(i, k) \leftarrow s(i, k) - \max_{k' \neq k}\{a(i, k') + s(i, k')\}$$

  2. Availability updates:

  $$a(i, k) \leftarrow \min(0, r(k, k) + \sum_{i' \in \{i,k\}} \max(0, r(i', k))) \text{ for } i \neq k$$

  $$a(k, k) \leftarrow \sum_{i' \neq k} \max(0, r(i', k))$$

  until $A+R$ remain unchanged over a number of steps, or after some predefined numbers of steps. For each point x_i, the data point x_k that maximizes $a(i, r)+r(i, k)$ gives us the class information of x_i.

We chose affinity propagation to generate class features as it only requires a few hyperparameters. More importantly, affinity propagation does not need to choose the number of classes. To employ affinity propagation, we used the edit distance to generate similarity matrices.

Given a training set and a test set, we first calculated the similarity matrix of the training set. Then we applied affinity propagation to the similarity matrix. The affinity propagation will assign each sample in the training set a cluster label, which is our class features. Besides, the number of clusters and the center of each cluster were also obtained from the results of affinity propagation (Figure 4.6). We then measured the edit distance between each sample in the test set and these cluster centers. Finally, we assigned each test sample the same cluster label as the cluster center with the minimum edit distance. The whole procedure is given by (2) in Figure 4.1b.

4.2.4 LightGBM

Gradient boosting machine [Fri01] is a machine learning algorithm that uses a group of weak prediction models (often decision trees) to make predictions. In this study, we utilized a gradient boosting machine-based framework — lightGBM.

LightGBM [KMF+17] is an efficient implementation of gradient boosting machine. It has been widely used in the field of bioinformatics and compuational biology since it has the following advantages:

- High speed and low memory cost: LightGBM uses a histogram-based algorithm [RS98, JA03, LWB07]. Such algorithm can assign continuous feature values into discrete bins, thereby leading to high training speed and low memory cost.

- High accuracy: Traditional decision tree-based learning algorithms generate trees level-wise. However, lightGBM generates trees leaf-wise. This strategy usually causes lower loss than level-wise algorithms.

- Support categorical features: One-hot encoding is an efficient encoding scheme for categorical features. However, for tree-based learning algorithms, one-hot features tend to generate very unbalanced trees, which may prevent the prediction model from achieving good accuracy. Instead of one-hot encoding, lightGBM allows users to input categorical features directly to train the model, which may lead to more balanced trees and more accurate results.

We built a lightGBM-based model — termed ReCGBM with relational features and class features as inputs. The outputs were cleavage sites or non-cleavage sites.

4.2.5 Evaluation Metrics

To assess the performance of our prediction model, we used several different metrics including sensitivity (Sn), specificity (Sp), accuracy (Acc) and Matthews correlation coefficient (MCC):

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

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

$$Acc = \frac{TP + TN}{TP + TN + FP + FN}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}}$$

where TP, TN, FP and FN denote the numbers of true positives, true negatives, false positives, and false negatives, respectively.

4.3 Results

4.3.1 Predictive Performance

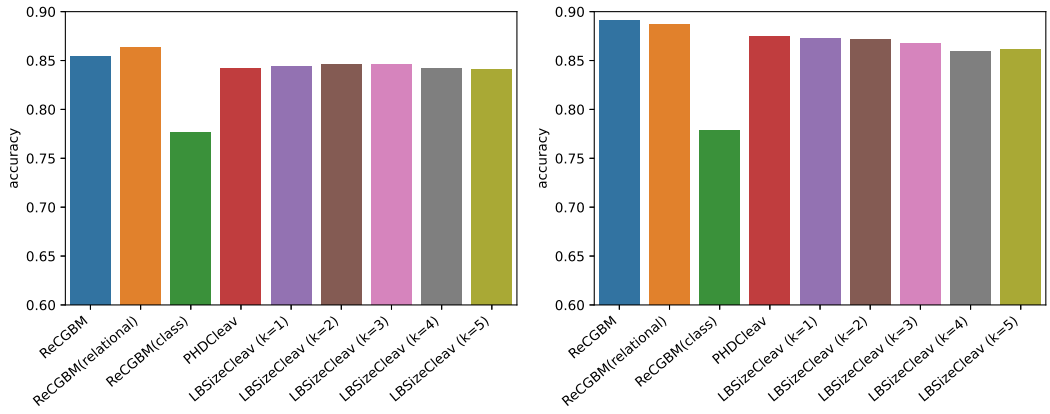
The main goal of this study is to develop an accurate prediction model for the Dicer cleavage sites. In this section, we show the predictive performance of ReCGBM and compare our results with other existing models.

We built prediction models for the 5p-arm dataset and 3p-arm dataset with secondary structures predicted by quickfold and RNAFold respectively.

To ensure the effectiveness of our model, we trained 10 models for each dataset, where we only considered the cases in which the affinity propagation converged. For each model, we randomly divided our preprocessed dataset into two subsets. The first subset that included 800 cleavage patterns and 800 non-cleavage patterns was used as the training set. The other subset was used as the independent test set, which included 156 cleavage patterns and 156 non-cleavage patterns. We computed the average sensitivity (Sn), specificity (Sp), accuracy (Acc), and MCC of the 10 models for each dataset.

We compared the predictive performance of ReCGBM with the existing methods, PHDCleav and LBSizeCleav. Since the performance of LBSizeCleav highly depends on the variable k, which represents the effect of length of loops/bulges on the kernel computation, we trained LBSizeCleav with $k = 1, 2, 3, 4, 5$ as previously described [BHA16]. All models were trained on the same 10 training sets and evaluated on the same 10 test sets for each dataset.

In order to tune the hyperparameters, we performed grid search on the training set for each models with GridSearchCV in scikit-learn [PVG+11]. For ReCGBM, we performed a grid search with $max\_depth \in [10, 20, 30, 40, 50, 60]$, $learning\_rate \in [0.05, 0.1, 0.15]$, and $num\_leaves \in [200, 300, 400]$, respectively. For PHDCleav and LBSizeCleav-based models, we performed grid search with $C \in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, and $gamma \in [0.1, 0.01, 0.001]$. After the best hyperparameters were chosen, the models with the best hyperparameters were trained on each training set.



(a) 5p-arm

(b) 3p-arm

Figure 4.4: Performance comparison between different models based on the datasets with secondary structures predicted by quickfold.

We use ReCGBM(relational) and ReCGBM(class) represent ReCGBM with only relational features and ReCGBM with only class features. The predictive performance of different models is illustrated in Figure 4.4 and Table 4.1 for both the 5p-arm dataset and the 3p-arm dataset with secondary structures predicted by quickfold. For the 5p-arm dataset with secondary structures predicted by quickfold, ReCGBM(relational) achieved a sensitivity of 0.881, a specificity of 0.846, an accuracy of 0.863 and an MCC of 0.728, respectively, which outperformed all other predictors in terms of three out of the four evaluation metrics. For the 3p-arm dataset with secondary structures predicted by quickfold, ReCGBM achieved the best sensitivity, specificity, accuracy, and MCC of 0.883, 0.899, 0.891, and 0.783, respectively. Therefore, ReCGBM with only relational features achieved the best performance for the 5p-arm dataset with secondary structures predicted by quickfold. ReCGBM with only relational features did not show better performance compared with other methods. However, ReCGBM, which includes class features and

Table 4.1: Performance comparison between different models based on the datasets with secondary structures predicted by quickfold. Best results are highlighted in bold. Sn, Sp, Acc and MCC represent sensitivity, specificity, accuracy and Matthews correlation coefficient, respectively

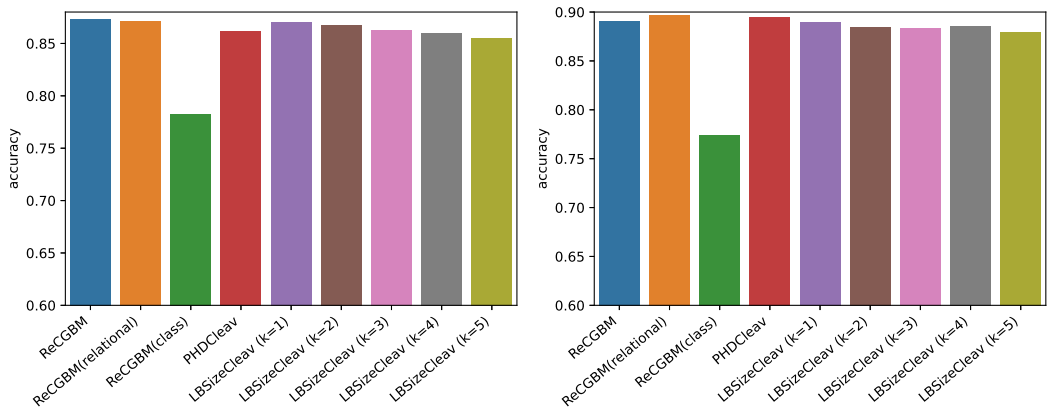
Method	5p-arm				3p-arm			
	Sn	Sp	Acc	MCC	Sn	Sp	Acc	MCC
ReCGBM	0.863	**0.846**	0.854	0.709	**0.883**	**0.899**	**0.891**	**0.783**
ReCGBM(relational)	**0.881**	**0.846**	**0.863**	**0.728**	**0.883**	0.890	0.887	0.774
ReCGBM(class)	0.729	0.824	0.777	0.556	0.746	0.810	0.778	0.558
PHDCleav	0.853	0.831	0.842	0.685	0.875	0.874	0.874	0.749
LBSizeCleav (k=1)	0.865	0.823	0.844	0.689	0.869	0.877	0.873	0.747
LBSizeCleav (k=2)	0.864	0.828	0.846	0.693	0.869	0.874	0.871	0.744
LBSizeCleav (k=3)	0.870	0.822	0.846	0.693	0.863	0.871	0.867	0.735
LBSizeCleav (k=4)	0.863	0.822	0.842	0.686	0.851	0.868	0.860	0.720
LBSizeCleav (k=5)	0.863	0.819	0.841	0.684	0.854	0.868	0.861	0.723

relational features, achieved the best performance for the 3p-arm dataset with secondary structures predicted by quickfold. Thus, it might be better to apply both relational features and class features for the 3p-arm dataset with secondary structures predicted by quickfold.

Figure 4.5 and Table 4.2 show the average specificity, sensitivity, accuracy, and MCC of models on both the 5p-arm dataset and the 3p-arm dataset with secondary structures predicted by RNAFold. For the 5p-arm dataset with secondary structures predicted by RNAFold, ReCGBM achieved the best specificity (0.862), accuracy (0.873) and MCC (0.747). In contrast, LBSizeCleav (k=3) achieved the best sensitivity (0.888). For the 3p-arm dataset with secondary structures predicted by RNAFold, ReCGBM(relational) achieved the best specificity (0.892), accuracy (0.897) and MCC (0.795), while PHDCleav achieved the best sensitivity (0.904).

Overall, ReCGBM and ReCGBM(relational) outperformed the other models, highlighting the effectiveness of this model for predicting human Dicer cleavage sites.

To further investigate whether different RNA secondary structures affect our prediction accuracy, we also generated relational features and class features using the RNAstructure package [BRSM13, LHK+13]. For $\sim$ 70nt RNA structure, RNAstructure predicts secondary structures of the pre-miRNA-size RNAs with the accuracy near 100% .We trained 5 models for each dataset, where we only considered the cases in which the affinity propagation converged. For each model, we randomly divided our preprocessed dataset into two subsets. The first subset that included 800 cleavage patterns and 800 non-cleavage patterns was used as the training set. The other subset was used as the independent test set, which included 156 cleav-



(a) 5p-arm (b) 3p-arm

Figure 4.5: Performance comparison between different models based on the datasets with secondary structures predicted by RNAFold.

Table 4.2: Performance comparison between different models based on the datasets with secondary structures predicted by RNAFold. Best results are highlighted in bold. Sn, Sp, Acc and MCC represent sensitivity, specificity, accuracy and Matthews correlation coefficient, respectively

Method	5p-arm				3p-arm			
	Sn	Sp	Acc	MCC	Sn	Sp	Acc	MCC
ReCGBM	0.884	**0.862**	**0.873**	**0.747**	0.888	**0.892**	0.890	0.781
ReCGBM(relational)	0.887	0.856	0.871	0.744	0.903	**0.892**	**0.897**	**0.795**
ReCGBM(class)	0.737	0.827	0.782	0.568	0.739	0.809	0.774	0.550
PHDCleav	0.878	0.845	0.862	0.724	**0.904**	0.884	0.894	0.789
LBSizeCleav (k=1)	0.886	0.855	0.871	0.743	0.895	0.885	0.890	0.780
LBSizeCleav (k=2)	0.887	0.847	0.867	0.736	0.890	0.879	0.885	0.770
LBSizeCleav (k=3)	**0.888**	0.837	0.863	0.727	0.890	0.877	0.883	0.767
LBSizeCleav (k=4)	0.876	0.844	0.860	0.721	0.892	0.878	0.885	0.770
LBSizeCleav (k=5)	0.879	0.831	0.855	0.712	0.883	0.874	0.879	0.758

age patterns and 156 non-cleavage patterns. The results are shown in Table 4.3. The performance of ReCGBM with RNAstructure is close to the performance of ReCGBM with RNAFold.

Table 4.3: Performance comparison between ReCGBM with different secondary structures. Sn, Sp, Acc and MCC represent sensitivity, specificity, accuracy and Matthews correlation coefficient, respectively

Structures	5p-arm				3p-arm			
	Sn	Sp	Acc	MCC	Sn	Sp	Acc	MCC
quickfold	0.855	0.862	0.858	0.717	0.881	0.890	0.885	0.771
RNAFold	0.879	0.869	0.874	0.749	0.901	0.895	0.898	0.797
RNAstructure	0.874	0.871	0.872	0.745	0.896	0.897	0.897	0.794

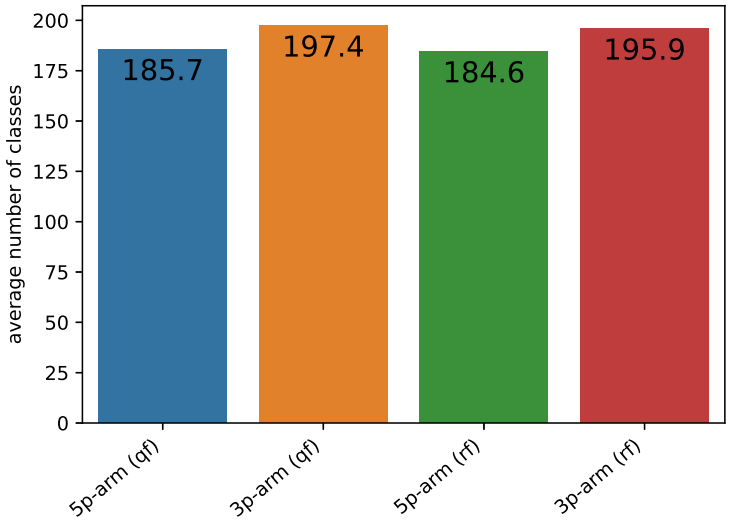


Figure 4.6: Average number of classes for different data types. The terms qf and rf represent quickfold and RNAFold, respectively.

Affinity Propagation

The goal of this experiment is to explore the relationship between class features and cleavage/non-cleavage sites. To address this, we applied affinity propagation to each training set of 3p-arm and 5p-arm with secondary structures predicted by quickfold and RNAFold.

Figure 4.6 shows the average cluster results of 10 training sets of 3p-arm and 5p-arm with secondary structures predicted by quickfold and RNAFold.

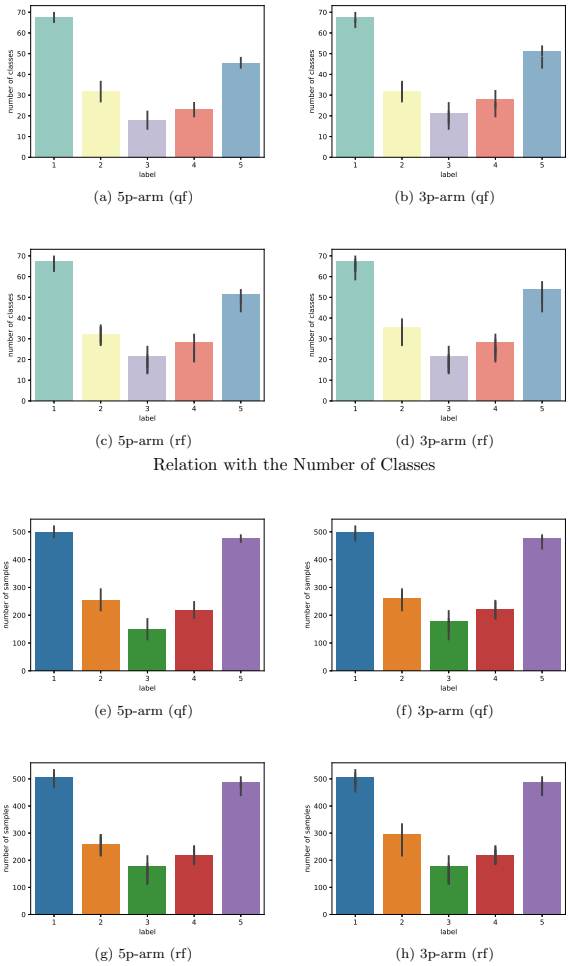
To further investigate the relationship between the class features and the cleavage/non-cleavage sites, we defined $ratio_i$(cleavage) for each class i in Figure 4.6

as follows:

$$ratio_i(\text{cleavage}) = \frac{N_i(\text{cleavage})}{N_i(\text{non-cleavage}) + N_i(\text{cleavage})}$$

where N_i(cleavage) and N_i(non-cleavage) represent the number of cleavage patterns and the number of non-cleavage patterns in class i, respectively. If $ratio_i$(cleavage) is close to 0, the samples in class i are almost non-cleavage patterns. On the other hand, if $ratio_i$(cleavage) is close to 1, the samples in class i are almost cleavage patterns. Classes with very high $ratio$(cleavage) and classes with very low $ratio$(cleavage) are desirable as these classes reflect that the class features have the potential to distinguish cleavage sites from non-cleavage sites.

We assume class i belongs to label 1, 2, 3, 4, 5 if $ratio_i(\text{cleavage}) \in [0, 0.2]$, $(0.2, 0.4]$, $(0.4, 0.6]$, $(0.6, 0.8]$, $(0.8, 1.0]$, respectively.



(a) 5p-arm (qf) (b) 3p-arm (qf)

(c) 5p-arm (rf) (d) 3p-arm (rf)

Relation with the Number of Classes

(e) 5p-arm (qf) (f) 3p-arm (qf)

(g) 5p-arm (rf) (h) 3p-arm (rf)

Relation with the Number of Samples

Figure 4.7: Results of affinity propagation. (a), (b), (c) and (d) plot the relationships between the average number of classes and different labels for the 5p-arm dataset (qf), 3p-arm dataset (qf), 5p-arm dataset (rf) and 3p-arm dataset (rf), respectively where qf represents secondary structures predicted by quickfold server and rf denotes secondary structures predicted by RNAFold. (e), (f), (g) and (h) show the relationships between the average number of samples and different labels for the 5p-arm dataset (qf), 3p-arm dataset (qf), 5p-arm dataset (rf) and 3p-arm dataset (rf), respectively.

In Figure 4.7, (a), (b), (c) and (d) show the relationships between the average number of classes and each label for the 5p-arm dataset(qf), 3p-arm dataset(qf), 5p-arm dataset(rf) and 3p-arm dataset(rf) in Figure 4.6, respectively. The commonality in these four figures is that the numbers of classes in label 1 and label 5 were much higher than others. (e), (f), (g), and (h) in Figure 4.7 describe the relationships between the average number of samples and each label for four datasets. It is obvious that the numbers of samples in label 1 and label 5 were much higher than others.

Thus, the results of affinity propagation show that the majority of classes are overwhelmed by either only cleavage sites or non-cleavage sites, which indicates that the clusters based on edit distance may improve the prediction of the cleavage site.

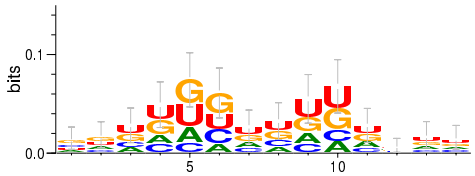
4.3.2 Sequence Logo Representations

To explore the difference between cleavage sites and non-cleavage sites at the RNA sequence level, we draw the sequence logo representations (Figure 4.8) of the 5p-arm cleavage sites, 5p-arm non-cleavage sites, 3p-arm cleavage sites and 3p-arm non-cleavage sites by WebLogo 3 [CHCB04].

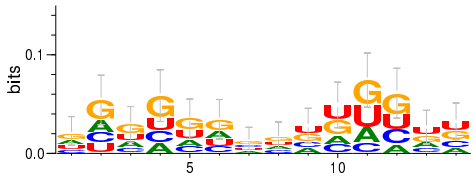
As can be seen from Figure 4.8, the 5p-arm cleavage sites and 5p-arm non-cleavage sites show different preferences for the neighboring nucleotides. For cleavage sites, sequence motifs associated with over-represented nucleotides at the positions 3-11 (Figure 4.8 (a)) can be easily observed, while for non-cleavage sites, no specific nucleotides were found to be over-represented at the positions 7 and 8 (Figure 4.8 (b)). For the 3p-arm cleavage sites (Figure 4.8 (c)), nucleotides at two specific positions 7 and 8 showed most distinctive preferences, compared with the 3p-arm non-cleavage sites (Figure 4.8 (d)). There also exist subtle differences in other positions such as the positions 5, 9, 10, and 11 between the 3p-arm cleavage sites and non-cleavage sites. Altogether, the nucleotide preferences shown in Figure 4.8 represent patterns important for distinguishing the Dicer cleavage sites from non-cleavage sites.

4.3.3 Feature Importance

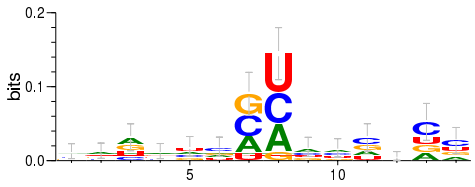
Gradient boosting machine typically uses decision trees as the base learners. An advantage of the decision tree is that it is an explainable model. Therefore, it is also possible to interpret a gradient boosting machine as an ensemble of decision trees. Fortunately, lightGBM has a built-in module that provides a score to describe the usefulness of each feature for a trained model. Here we will discuss the feature importance of ReCGBM.



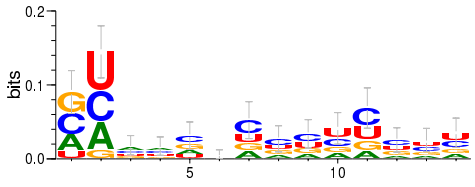
(a) 5p-arm cleavage sites



(b) 5p-arm non-cleavage sites



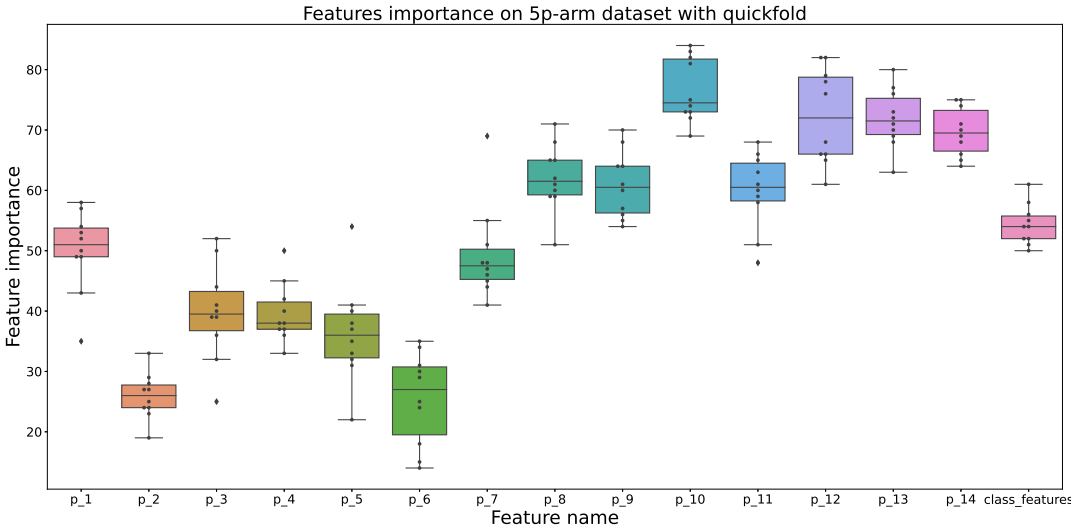
(c) 3p-arm cleavage sites



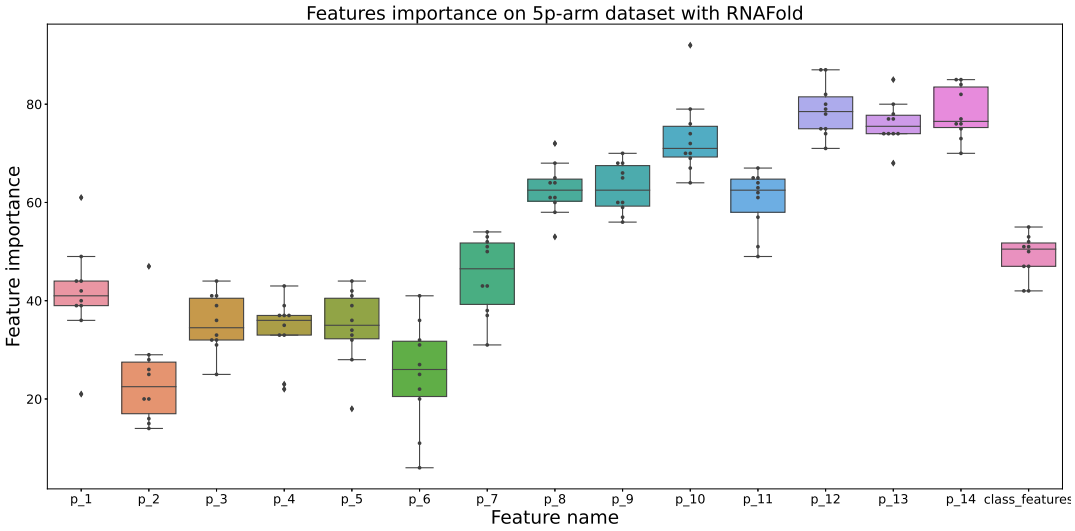
(d) 3p-arm non-cleavage sites

1

Figure 4.8: Sequence logo representations of cleavage sites and non-cleavage sites. (a) and (b) are sequence logo representations of the 5p-arm cleavage sites and the 5p-arm non-cleavage sites. (c) and (d) are sequence logo representations of the 3p-arm cleavage sites and the 3p-arm non-cleavage sites.

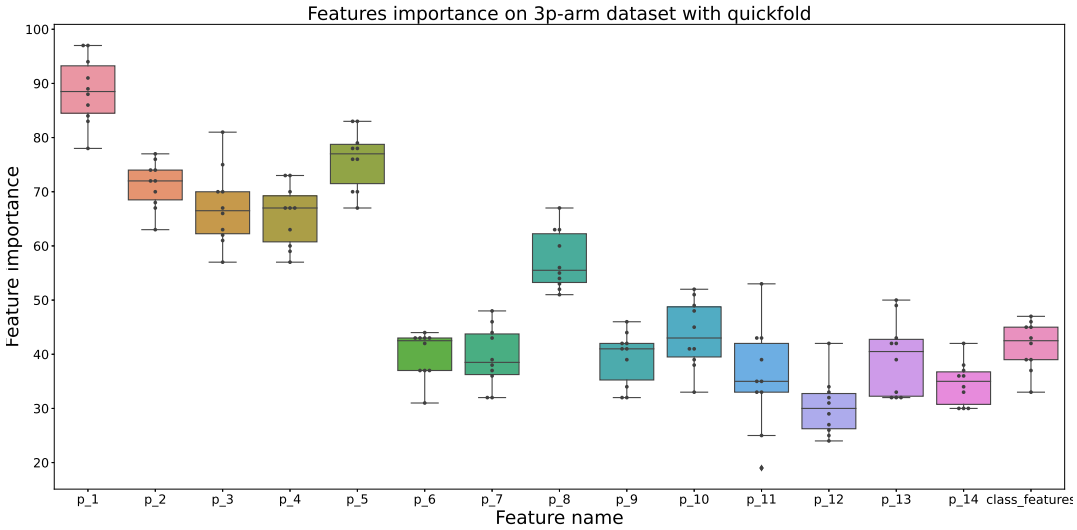


(a)

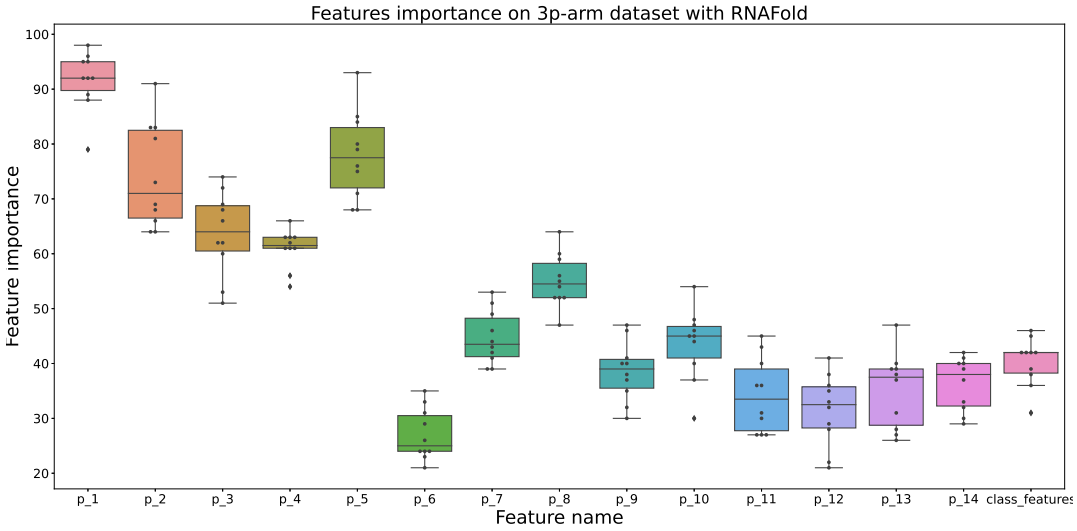


(b)

Figure 4.9: Results of feature importance on the 5p-arm datasets. The secondary structures of (a) and (b) are predicted by quickfold and RNAFold, respectively.



(a)



(b)

Figure 4.10: Results of feature importance on the 3p-arm datasets. The secondary structures of (a) and (b) are predicted by quickfold and RNAFold, respectively.

Since we trained 10 models for each dataset, we list the average feature importance of the 10 trained models for each dataset.

Let $p_1, \ldots, p_{14}$ denote the 14 pairs of the relational features (Figure 4.1b (1)). Figure 4.9 and Figure 4.10 show the results of feature importance for different

datasets.

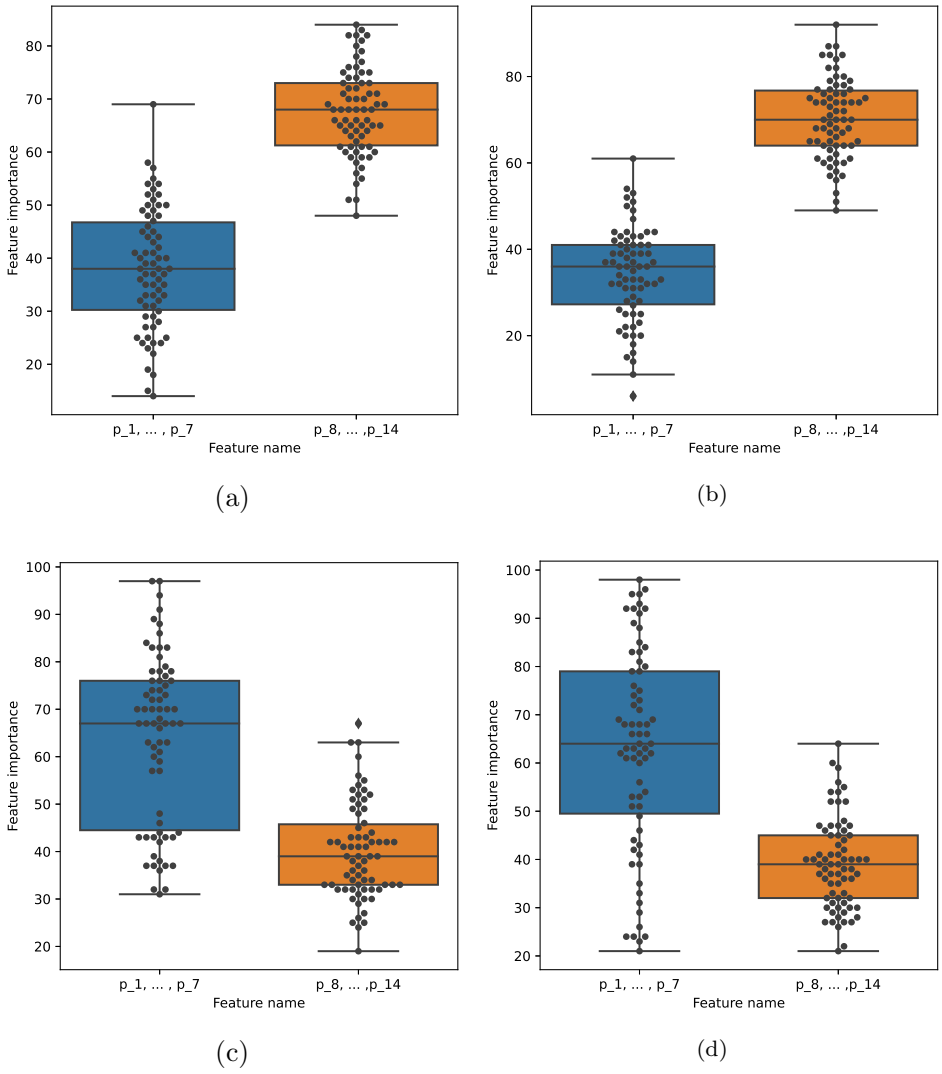


Figure 4.11: Comparisons of feature importance between $p_1, \ldots, p_7$ and $p_8, \ldots, p_14$. (a) is the result on 5p-arm data with secondary structures predicted by quickfold; (b) is the result on 5p-arm data with secondary structures predicted by RNAFold; (c) is the result on 3p-arm data with secondary structures predicted by quickfold; (d) is the result on 3p-arm data with secondary structures predicted by RNAFold.

Figure 4.9 gives the feature importance of the 5p-arm datasets with the secondary structures predicted by quickfold and RNAFold, respectively. It can be seen

that $p_8, \ldots, p_{14}$ were more important than $p_1, \ldots, p_7$. Figure 4.11a and 4.11b shows the relationships between $p_8, \ldots, p_{14}$ and $p_1, \ldots, p_7$ on 5p-arm dataset directly.

Figure 4.10 shows the feature importance of the 3p-arm datasets with the secondary structures predicted by quickfold and RNAFold, respectively. For the 3p-arm datasets, $p_1, \ldots, p_7$ were more important than $p_8, \ldots, p_{14}$. Figure 4.11c and 4.11d shows the relationships between $p_8, \ldots, p_{14}$ and $p_1, \ldots, p_7$ on the 3p-arm dataset directly.

4.4 Discussion

Although ReCGBM showed a better performance on three of the four datasets, PHDCleav achieved the best performance on the 3p-arm dataset with the secondary structures predicted by RNAFold. There are several potential reasons. First, the performance of ReCGBM highly depends on the predicted secondary structures. To obtain relational features and class features, the secondary structure information is necessary. However, the secondary structure generated by quickfold or RNAFold may include several structures. In ReCGBM, only one structure can be included in our input features, which might affect the prediction performance.

Second, some class features generated by the affinity propagation may not be accurate. As shown in Figure 4.6, the number of classes given by the affinity propagation for each dataset was over 180. However, the number of samples in each dataset was 1912. Thus such classes may exist: the distance between each sample in this class may not be so close. The distance between samples in this class and samples in other classes is relatively farther. The affinity propagation may fail to find data that are 'close enough' to each sample in such classes. By 'close enough' we mean different samples are close enough such that they share some common properties.

The feature importance of ReCGBM also shows some connections between the RNA secondary structures and human Dicer cleavage site prediction. Considering the position of relational features in the secondary structure of pre-miRNA (Figure 4.12), $p_8, \ldots, p_{14}$ of 5p-arm and $p_1, \ldots, p_7$ of 3p-arm are closer to the center of the pre-miRNA. Therefore, relational features close to the center of pre-miRNA may contribute more to human Dicer cleavage sites prediction.

Another observation is that class features are more important in the 5p-arm datasets, which can be concluded based on the results shown in Figure 4.9 and Figure 4.10.

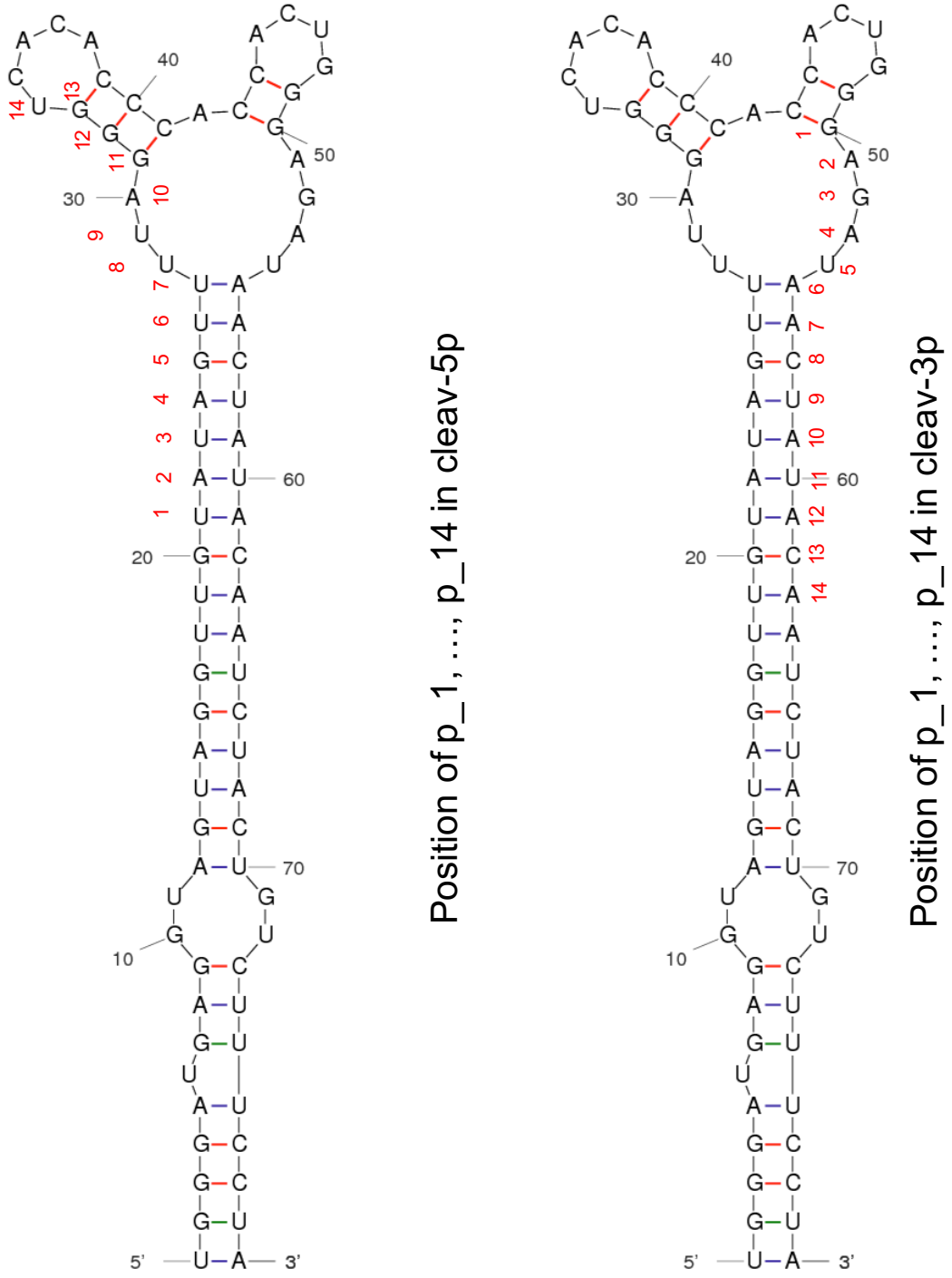


Figure 4.12: Positions of relational features in the secondary structure of pre-miRNA

In summary, we have introduced a lightGBM-based model—termed as ReCGBM

for accurate prediction of human Dicer cleavage sites and analyzed the feature importance of ReCGBM. Computational experiments demonstrated the effectiveness of this model. However, possible improvements can be achieved in the future. First, affinity propagation finds the number of clusters automatically. However, in some cases it is hard to converge on small datasets. Thus, an easy-to-converge cluster algorithm is desirable. Second, the predictive performance of this model highly depends on the predicted secondary structures of pre-miRNA. Therefore, a feature encoding strategy that can combine several secondary structures given by quickfold or RNAFold for a pre-miRNA is desirable. Finally, the analyses of feature importance showed that the relational features that are localized in close proximity to the center of the pre-miRNA are more important than the other features. Accordingly, it might be potentially useful to consider more informative features close to the center of the pre-miRNA in future predictors.

Chapter 5

Conclusion

In Chapter 3, we aim to design algorithms that can extract simple rules from trained neural networks. Based on this motivation, we develop two algorithms to extract rules from trained neural networks. The first algorithm extracts Boolean rules from a trained neural network. If the trained neural network can be represented by a k-out-of-n majority function or a nested canalyzing function, this algorithm can rapidly output the corresponding Boolean rules. For other cases, this algorithm may output the related Boolean rules with exponential size. Therefore, to conduct this algorithm in practice, we ban the generation of high-order rules. Computational experiments show that this approximating algorithm outputs simple and concise Boolean rules even we only allow the generation of rule terms to the third order. However, if there is no simple Boolean rule that can represent the trained neural network, or if the trained neural network has multiple hidden layers, our algorithm may output nothing if we ban the high-order rules.

To solve the above issue, we propose a probabilistic rule extraction algorithm. This algorithm can output the relationship between input values and output values in the form of conditional probabilities given a trained neural network. Our theoretical analyses and computational experiments show that this algorithm can extract simple probabilistic rules from neural networks with multiple hidden layers. Another merit is that this algorithm does not assume any simple Boolean representations for a trained neural network. It extracts probabilistic rules from any trained neural networks with a proper size. However, the computational cost is high for neural networks with many computational neurons since this algorithm performs an exhaustive search in the second layer of a neural network.

Although our algorithms have partially solved some issues in previous methods, there are still many problems. Thus, future work can consider the following aspects.

First, the time cost of the probabilistic rule extraction algorithm is very high due to the exhaustive search for the second layer of a trained neural network. To accelerate this procedure, we may consider combining the algorithm with sparse

modeling.

Second, in our algorithms, we assume that the input neurons are mutually independent. However, in some real-world datasets, different input features may have some correlations. Therefore, it is necessary to develop rule extraction algorithms that consider the correlations between different input features.

Third, our Boolean rule extraction algorithm and probabilistic rule extraction algorithm assume each neuron's activation function is a linear threshold function. Therefore, it is reasonable to apply these algorithms in trained neural networks with sigmoid functions since linear threshold functions are similar to sigmoid functions. In many recent studies, rectified linear unit (ReLU) is considered as the activation function because of the good performance in very deep neural networks. However, there is a big difference between rectified linear units and linear threshold functions. Thus, it is interesting to consider algorithms to extract simple rules from trained neural networks with rectified linear units.

Finally, our algorithms in this study focus on feedforward neural networks for supervised learning. Recent studies in neural networks also focus on unsupervised learning. Therefore, it is also interesting to explore the rule extraction algorithms of neural networks for unsupervised learning tasks.

Chapter 4 introduces an explainable gradient boosting model to predict human Dicer cleavage sites – ReCGBM. In this model, we first construct relational features by combining the sequences and its complementary strands into pairs. Then we create class features using edit distance and affinity propagation. Finally, we train a gradient boosting model by LightGBM. ReCGBM has two advantages: First, it makes more accurate predictions than previous methods. Second, ReCGBM can output the feature importance, which can partially explain how the model makes decisions. For RNA sequence data, this may also lead to some biological explanations. In our experiments, the feature importance in the ReCGBM shows that relational features close to the center of pre-miRNA may contribute more to human Dicer cleavage site prediction.

Despite the fact that ReCGBM performs better than previous methods, some improvements can be made in the following directions.

First, we consider the affinity propagation to cluster our samples and assign class labels to the corresponding samples in the construction of class features. One advantage of affinity propagation is that it does not need to set the number of classes. However, sometimes the algorithm can not converge. Therefore, it might be better to try some easy-to-converge cluster algorithms.

Second, although our features include the secondary structure information, only one secondary structure can be included in a sample. However, the number of possible secondary structures of a sample may be more than one. Therefore, to improve the prediction performance, a feature engineering method that can include multiple potential secondary structures in one sample is desirable.

Third, we generate the negative samples through a fixed way. However, the real negative samples may be diverse. Instead of considering all kinds of negative samples, it might be interesting to consider PU learning, which is the abbreviation for positive-unlabeled learning. In the miRBase dataset, pre-miRNA with a known cleavage site can be used as positive samples. Other pre-miRNA with unknown cleavage sites can be considered as unlabeled samples. This is more reasonable since positive samples, and unlabeled samples are known. Instead of throwing all unlabeled samples away, it might be a better choice to consider these samples as the training data in PU learning.

Bibliography

[AK12] M Gethsiyal Augusta and Thangairulappan Kathirvalavakumar. Reverse engineering the neural networks for rule extraction in classification problems. *Neural Processing Letters*, 35(2):131–150, 2012.

[AKR13] Firoz Ahmed, Rakesh Kaundal, and Gajendra PS Raghava. Phdcleav: a svm based method for predicting human dicer cleavage sites using sequence and secondary structure of mirna precursors. *BMC Bioinformatics*, 14(14):1–11, 2013.

[Ant01] Martin Anthony. *Discrete Mathematics of Neural Networks: Selected Topics*. SIAM, Philadelphia, PA, USA, 2001.

[Ant10] Martin Anthony. Decision lists and related classes of boolean functions. In *Crama, Yves and Hammer, Peter L., Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, pages 577–598, New York, NY, USA, 2010. Cambridge University Press.

[Bar93] Peter Barth. Linear 0–1 inequalities and extended clauses. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning*, pages 40–51. Springer, 1993.

[BHA16] Yu Bao, Morihiro Hayashida, and Tatsuya Akutsu. Lbsizecleav: improved support vector machine (svm)-based prediction of dicer cleavage sites using loop/bulge length. *BMC Bioinformatics*, 17(1):487, 2016.

[BRSM13] Stanislav Bellaousov, Jessica S Reuter, Matthew G Seetin, and David H Mathews. Rnastructure: web servers for rna secondary structure prediction and analysis. *Nucleic Acids Research*, 41(W1):W471–W474, 2013.

[CG16] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining, pages 785–794. ACM, 2016.

[CHCB04] Gavin E Crooks, Gary Hon, John-Marc Chandonia, and Steven E Brenner. Weblogo: a sequence logo generator. *Genome Research*, 14(6):1188–1190, 2004.

[CLG01] Rich Caruana, Steve Lawrence, and Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. *Advances in Neural Information Processing Systems*, 13:402–408, 2001.

[DG17] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

[EBM09] Hazem M El-Bakry and Nikos Mastorakis. A fast computerized method for automatic simplification of boolean functions. In *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering*. WSEAS, 2009.

[FD07] Brendan J Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[FHT+00] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Annals of Statistics*, 28(2):337–407, 2000.

[FNCT06] Adrien Fauré, Aurélien Naldi, Claudine Chaouiya, and Denis Thieffry. Dynamical analysis of a generic boolean model for the control of the mammalian cell cycle. *Bioinformatics*, 22(14):e124–e131, 2006.

[Fri01] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[FZS13] Yong-Xian Fan, Yang Zhang, and Hong-Bin Shen. Labcas: labeling calpain substrate cleavage sites from amino acid sequence using conditional random fields. *Proteins: Structure, Function, and Bioinformatics*, 81(4):622–634, 2013.

[GJ79] Michael R Garey and David S Johnson. *Computers and Intractability: a Guide to the Theory of NP-completeness (Series of Books in the Mathematical Sciences)*. WH Freeman, 1979.

[GJSvDE07] Sam Griffiths-Jones, Harpreet Kaur Saini, Stijn van Dongen, and Anton J Enright. mirbase: tools for microrna genomics. *Nucleic Acids Research*, 36(suppl_1):D154–D158, 2007.

[GMUTN+19] Paulina Galka-Marciniak, Martyna Olga Urbanek-Trzeciak, Paulina Maria Nawrocka, Agata Dutkiewicz, Maciej Giefing, Marzena Anna Lewandowska, and Piotr Kozlowski. Somatic mutations in mirna genes in lung cancer–potential functional consequences of non-coding sequence variants. *Cancers*, 11(6):793, 2019.

[HJL+05] Huiling He, Krystian Jazdzewski, Wei Li, Sandya Liyanarachchi, Rebecca Nagy, Stefano Volinia, George A Calin, Chang-gong Liu, Kaarle Franssila, Saul Suster, et al. The role of microrna genes in papillary thyroid carcinoma. *Proceedings of the National Academy of Sciences*, 102(52):19075–19080, 2005.

[HJR78] David Harrison Jr and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5(1):81–102, 1978.

[Hof03] Ivo L Hofacker. Vienna rna secondary structure server. *Nucleic Acids Research*, 31(13):3429–3431, 2003.

[HSWK02] Stephen E. Harris, Bruce K. Sawhill, Andrew Wuensche, and Stuart Kauffman. A model of transcriptional regulatory networks based on biases in the observed regulation rules. *Complexity*, 7(4):23–40, 2002.

[IFL+05] Marilena V Iorio, Manuela Ferracin, Chang-Gong Liu, Angelo Veronese, Riccardo Spizzo, Silvia Sabbioni, Eros Magri, Massimo Pedriali, Muller Fabbri, Manuela Campiglio, et al. Microrna gene expression deregulation in human breast cancer. *Cancer Research*, 65(16):7065–7070, 2005.

[Ish00] Masumi Ishikawa. Rule extraction by successive regularization. *Neural Networks*, 13(10):1171–1183, 2000.

[JA03] Ruoming Jin and Gagan Agrawal. Communication and memory efficient parallel decision tree construction. In *Proceedings of the*

2003 SIAM International Conference on Data Mining, pages 119–129. SIAM, 2003.

[JRL07] Abdul Salam Jarrah, Blessilda Raposa, and Reinhard Laubenbacher. Nested canalyzing, unate cascade, and polynomial functions. *Physica D: Nonlinear Phenomena*, 233(2):167–174, 2007.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*, 2014.

[KMF+17] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[Le13] Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8595–8598. IEEE, 2013.

[Lev66] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[LHK+13] Christopher W Leonard, Christine E Hajdin, Fethullah Karabiber, David H Mathews, Oleg V Favorov, Nikolay V Dokholyan, and Kevin M Weeks. Principles for understanding the accuracy of shape-directed rna structure modeling. *Biochemistry*, 52(4):588–595, 2013.

[LWB07] Ping Li, Qiang Wu, and Christopher J Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems*, volume 20, pages 897–904, 2007.

[LYD+19] Zexian Liu, Kai Yu, Jingsi Dong, Linhong Zhao, Zekun Liu, Qingfeng Zhang, Yimeng Du, Shihua Li, and Han Cheng. Precise prediction of calpain cleavage sites and their aberrance caused by mutations in cancer. *Frontiers in Genetics*, 10:715, 2019.

[MCCA18] Avraham A Melkman, Xiaoqing Cheng, Wai-Ki Ching, and Tatsuya Akutsu. Identifying a probabilistic boolean threshold network from samples. *IEEE Transactions on Neural Networks and Learning Systems*, 29(4):869–881, 2018.

[MHZ+08] Maciej A Mazurowski, Piotr A Habas, Jacek M Zurada, Joseph Y Lo, Jay A Baker, and Georgia D Tourassi. Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance. *Neural networks*, 21(2-3):427–436, 2008.

[MZK] NR Markham, M Zuker, and JM Keith. Unafold: software for nucleic acid folding and hybridization. *Bioinformatics*, 453:3–31.

[O'D14] Ryan O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, New York, NY, USA, 2014.

[OSM+11] Yasuko Ono, Hiroyuki Sorimachi, Hiroshi Mamitsuka, et al. Calpain cleavage prediction using multiple kernel learning. *PloS ONE*, 6(5):e19035, 2011.

[PGCC17] Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. PyTorch. `https://pytorch.org/get-started/locally/`, 2017. Accessed: 2018-12-17.

[PLN+10] Mirva Piippo, Niina Lietzén, Olli S Nevalainen, Jussi Salmi, and Tuula A Nyman. Pripper: prediction of caspase cleavage sites from whole proteomes. *BMC Bioinformatics*, 11(1):320, 2010.

[PS85] Franco P Preparata and Michael I Shamos. *Computational Geometry: an Introduction*. Springer, New York, NY, USA, 1985.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Qui93] J Ross Quinlan. *C4. 5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[RG14] Irina Rish and Genady Grabarnik. *Sparse Modeling: Theory, Algorithms, and Applications*. CRC press, 2014.

[RS98] Sanjay Ranka and V Singh. Clouds: A decision tree classifier for large datasets. In *Proceedings of the 4th Knowledge Discovery and Data Mining Conference*, volume 2, pages 2–8, 1998.

[RSG16] 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*, pages 1135–1144. ACM, 2016.

[SDBR14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *ArXiv Preprint ArXiv:1412.6806*, 2014.

[Sha78] Michael Ian Shamos. *Computational geometry.* Ph. D. thesis, Yale University, 1978.

[SS16] Onkar Singh and Emily Chia-Yu Su. Prediction of hiv-1 protease cleavage site using a combination of sequence, structural, and physicochemical features. *BMC Bioinformatics*, 17:279–289, 2016.

[STK+17] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *ArXiv Preprint ArXiv:1706.03825*, 2017.

[STP+12] Jiangning Song, Hao Tan, Andrew J Perry, Tatsuya Akutsu, Geoffrey I Webb, James C Whisstock, and Robert N Pike. Prosper: an integrated feature-based tool for predicting protease substrate cleavage sites. *PloS ONE*, 7(11):e50300, 2012.

[STS+10] Jiangning Song, Hao Tan, Hongbin Shen, Khalid Mahmood, Sarah E Boyd, Geoffrey I Webb, Tatsuya Akutsu, and James C Whisstock. Cascleave: towards more accurate prediction of caspase substrate cleavage sites. *Bioinformatics*, 26(6):752–760, 2010.

[SWI07] Emad W Saad and Donald C Wunsch II. Neural network explanation using inversion. *Neural Networks*, 20(1):78–93, 2007.

[TKY+04] Junichi Takamizawa, Hiroyuki Konishi, Kiyoshi Yanagisawa, Shuta Tomida, Hirotaka Osada, Hideki Endoh, Tomoko Harano, Yasushi Yatabe, Masato Nagino, Yuji Nimura, et al. Reduced expression of the let-7 micrornas in human lung cancers in association with shortened postoperative survival. *Cancer Research*, 64(11):3753–3756, 2004.

[TOB12] Cristiana Tanase, Irina Ogrezeanu, and Corin Badiu. *Molecular Pathology of Pituitary Adenomas*. Elsevier, 2012.

[Tsu00] Hiroshi Tsukimoto. Extracting rules from trained neural networks. *IEEE Transactions on Neural Networks*, 11(2):377–389, 2000.

[WTR06] Lawrence JK Wee, Tin Wee Tan, and Shoba Ranganathan. Svm-based prediction of caspase substrate cleavage sites. *BMC Bioinformatics*, 7(S5):S14, 2006.

[WTR07] Lawrence JK Wee, Tin Wee Tan, and Shoba Ranganathan. Casvm: web server for svm-based prediction of caspase substrates cleavage sites. *Bioinformatics*, 23(23):3241–3243, 2007.

[WZT+14] Mingjun Wang, Xing-Ming Zhao, Hao Tan, Tatsuya Akutsu, James C Whisstock, and Jiangning Song. Cascleave 2.0, a new approach for predicting caspase and granzyme cleavage targets. *Bioinformatics*, 30(1):71–80, 2014.

[ZT00] Liang Zhao and Charles E Thorpe. Stereo-and neural network-based pedestrian detection. *IEEE Transactions on Intelligent Transportation Systems*, 1(3):148–154, 2000.

Publication Notes

Pengyu Liu, Avraham A Melkman, Tatsuya Akutsu. (2020) Extracting boolean and probabilistic rules from trained neural networks, *Neural Networks*, 126, 300 – 311.

Pengyu Liu, Jiangning Song, Chun-Yu Lin, Tatsuya Akutsu. (2021) ReCGBM: a gradient boosting-based method forpredicting human Dicer cleavage sites, *BMC Bioinformatics*, in press.

The thesis is based on the above two papers. You can find the permissions to reuse the contents in the following links:

1. Neural Networks: www.elsevier.com/about/policies/copyright/permissions
2. BMC Bioinformatics: www.biomedcentral.com/about/policies/reprints-and-permissions

Publications not Relevant to the Thesis

Yu Bao, Morihiro Hayashida, Pengyu Liu, Masayuki Ishitsuka, Jose C. Nacher and Tatsuya Akutsu. (2018) Analysis of Critical and Redundant Vertices in Controlling Directed Complex Networks Using Feedback Vertex Sets, *Journal of Computational Biology*, 25, 1071-1090.

Sini Guo, Pengyu Liu, Wai-Ki Ching and Tatsuya Akutsu. (2021) On the Distribution of Successor States in Boolean Threshold Networks, *IEEE Transactions on Neural Networks and Learning Systems*, in press.

Avraham A Melkman, Sini Guo, Wai-Ki Ching, Pengyu Liu, Tatsuya Akutsu. (2020) On the Compressive Power of Boolean Threshold Autoencoders, *https://arxiv.org/abs/2004.09735*, in revision.