Intelligible Models for Learning Categorical Data via Generalized Fourier Spectrum

by

Xuhong Zhang

B.A. and M.Eng, University of Cambridge (2012)
S.M., Massachusetts Institute of Technology (2015)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

Author . . . .
Department of Electrical Engineering and Computer Science
January 31, 2019

Certified by. . . . .
Gregory W. Wornell
Sumitomo Professor of Engineering
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Intelligible Models for Learning Categorical Data via Generalized Fourier Spectrum

by

Xuhong Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on January 31, 2019, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Machine learning techniques have found ubiquitous applications in recent years and sophisticated models such as neural networks and ensemble methods have achieved impressive predictive performances. However, these models are hard to interpret and usually used as a blackbox. In applications where an explanation is required in addition to a prediction, linear models (e.g. Linear Regression or Logistic Regression) remain to be mainstream tools due to their simplicity and good interpretability.

This thesis considers learning problems on categorical data and proposes methods that retain the good interpretability of linear models but significantly improve the predictive performance. In particular, we provide ways to automatically generate and efficiently select new features based on the raw data, and then train a linear model in the new feature space.

The proposed methods are inspired by the Boolean function analysis literature, which studies the Fourier spectrum of Boolean functions and in turn provides spectrum-based learning algorithms. Such algorithms are important tools in computational learning theory, but not considered practically useful due to the unrealistic assumption of uniform input distribution. This work generalizes the idea of Fourier spectrum of Boolean functions to allow arbitrary input distribution.

The generalized Fourier spectrum is also of theoretical interest. It carries over and meaningfully generalizes many important results of Fourier spectrum. Moreover, it offers a framework to explore how the input distribution and target function jointly affect the difficulty of a learning problem, and provides the right language for discussing data-dependent, algorithm-independent complexity of Boolean functions.

Thesis Supervisor: Gregory W. Wornell
Title: Sumitomo Professor of Engineering

Acknowledgements

It is hard to believe it's time to finish up this thesis and say goodbye to MIT. The past six years have not always been nice and smooth, but it is definitely an incredibly fulfilling and unforgettable journey.

First and foremost, it is hard to put in words how lucky I feel to have Prof. Gregory Wornell as my PhD advisor. Greg taught me how to ask the right set of questions, how to approach a problem from multiple perspectives, how to tell a coherent story, and most importantly, how to take ownership of my work. I am extremely grateful for his support and patience along the way, allowing me to figure things out at my own pace, to make mistakes and learn from them, and to mature as a researcher and as a person.

This thesis will not be the same without my committee members, Prof. Devavrat Shah and Prof. Guy Bresler. Their insightful questions and comments during committee meetings have provided valuable alternative perspectives, and directed me to explore interesting new directions.

Being a member of the Signal, Information and Algorithms lab, I'm privileged to be surrounded by lab mates that are intelligent yet easygoing. I'm thankful to Hongchao Zhou, Ligong Wang, Arya Mazumdar, Ankit Singh Rawat, Christos Thrampoulidis, Amichai Painsky, James Krieger, Da Wang, Ying-zong Huang, Qing He, Gauri Joshi, Atulya Yellepeddi, David Romero, Vadim Smolyakov, Gal Shulkind, Mo Deng, Joshua Lee, Ganesh Ajjanagadde, Gary Lee, Adam Yedidia and Toros Arikan, you guys have made SiA such an awesome place. A special thank you goes to the best administrative assistant ever, Tricia O'Donnell, who is always super nice and ready to help.

Outside the SiA group, I also had the pleasure of making many dear friends: Jialiang Chen, Daniel Li, Duanhui Li, Ling Ren, Guolong Su, Peng Wang, Shujing Wang, Mengfei Wu, Tianfan Xue, Xi Yang, Wenzhen Yuan, Chiyuan Zhang, Yu Zhang, Zhendong Zhang. I will always remember the fun times we had: the hikings, the hot pots, the board games and the interesting discussions, technical or non-technical.

I would not be able to come to MIT and had this fantastic journey without the training I received before graduate school. I would like to take this opportunity to thank my wonderfully inspiring teachers and mentors: Prof. Nick Kingsbury, Dr. Joan Lasenby and Dr. Sumeetpal Singh at University of Cambridge; Mr. Duanxi Fan and Mr. Lihua Tang at No.2 Secondary School attached to East Normal University; and most importantly, Mr. Kefen Wu at No.3 Middle School of Jiaxing, who revealed to me how fun and elegant math is.

Last but not least, I would like to thank my parents and grandparents, who raised me with so much love and made every effort to provide me with the best education possible. I would like to thank my husband, Yutian Lei, for his love and support, and his understanding and encouragement when I was in doubt of myself. I am also in debt to my long-time friends, Xiaojun Bao, who I grew up with like sisters and is always there for me, and Junjia Yang, who has been a constant source of guidance and positive influence for the past 17 years.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The last decade has definitely witnessed a golden era of machine learning. In particular, the success of deep learning has brought about dramatic improvements in areas such as computer vision, speech recognition, and natural language processing, just to name a few. The whole world has been amazed by applications such as the Alpha Go or self-driving cars.

However, despite these impressive achievements, in many disciplines such as social sciences and economics, the mainstream modelling technique remains to be *linear models*. The main reason is that models can be used for very different purposes in different disciplines (Shmueli [2010]): while the more 'traditional' machine learning areas such as computer vision and speech recognition focus on empirical predictive performance, in disciplines such as psychology, education and political sciences, models are usually used to explain and understand the observations.

The most popular and well-performed techniques in the machine learning communities today, such as deep-learning based methods or ensemble methods, have great predictive power but often don't add much to the understanding of the problem. In fact, these methods are mostly used as a blackbox and very little interpretation can be provided, thus they are not suitable for fields such as social sciences. Instead, less powerful but more interpretable linear models are still widely used.

Accuracy and *interpretability* are often treated as a trade-off. But as Breiman [2001] pointed out, "*if a model is a poor emulation of nature, the conclusions may be wrong*". This thesis aims to provide methods that **retains the good interpretability of linear models but significantly improve the predictive performance**. In particular, we provide ways to *automatically generate and efficiently select new features from raw data, and train a linear model in the new feature space.*

The proposed methods are inspired by the *Boolean function analysis* literature and assume that the raw features are *categorical*, with a bounded alphabet size for each feature[1]. This assumption certainly limits the applicability of the proposed methods, yet we argue that there are many practically interesting applications that deal with categorical features, e.g. topics in a document or webpage, data collected via multiple-choice based surveys, etc. Moreover, many real-valued features are traditionally binned into categories to improve human readability, e.g. age brackets or household income brackets.

To be more concrete, let us consider the following example use case where we want to learn from categorical data[2]: assume that an advertisement network wants to serve ads to users. Ideally, only ads on products that the user is potentially interested in should be shown, both for better user experience and higher ads click rate. One way to achieve this is to divide all users into 'interest groups' such as *sports fan* or *movie lovers*, and the goal is to train classifiers predicting which interest groups a certain user belongs to. One can hope to learn such information based on 'topics' extracted from users' browser history. Assume we have access to a pool of pre-trained topics, where an example topic might look like [red sox, fenway, fenway park, baseball]. We can then assign one binary feature to each topic, indicating whether or not it has appeared in

[1]How to deal with categorical features with many categories is itself an interesting research problem, but is beyond the scope of this thesis.

[2]Our description is not realistic since we over-simplify many details. But it should suffice to make the point.

the browser history. These features will be referred to as *raw features*. As mentioned above, this thesis proposes methods to detect useful higher order interaction terms. In this example use case, such an interaction term might take the form of 'exactly one of topic A and topic B appeared'.

Apart from the good interpretability and competitive predictive performance, the proposed methods have some additional advantages. Firstly, the generation and selection of features can be fully parallelized, thus computationally will only incur a low overhead cost. Secondly, the methods turn out to be quite flexible for incorporating various forms of prior knowledge about the dataset to learn.

In addition to the practically useful learning algorithms, the *generalized Fourier spectrum* for Boolean functions we develop is also of theoretical interest. It naturally generalizes the classic definition of Fourier spectrum of Boolean functions by relaxing the assumption of uniform input distribution, offers a framework for discussing *data-dependent, algorithm-independent* complexity for Boolean functions, and also provides the right languages for concepts such as noise stability and learnability under sequential data generation scenarios.

1.1 Connection to and Deviation from Boolean Function Analysis

As mentioned above, this work is inspired by Boolean function analysis. Basically, just as Fourier transform provides a useful representation for real functions, one can represent a Boolean function by its Fourier spectrum, i.e. its projection onto a particular set of orthonormal basis (Fourier basis). The real function Fourier transform and Boolean function Fourier spectrum are not exactly the same. For example, while Fourier transform is usually only applied to one-dimensional or two-dimensional signals, we can talk about the Fourier spectrum of a d-dimensional

Boolean function[3]. But they share the conceptual similarity that functions of practical interest usually have most of their energy concentrated on 'low frequency terms'.

Linial and Mansour [1993] pioneered the spectrum-based approach to learning Boolean functions[4]. Since then, spectrum-based methods have been a widely used tool in computational learning theory to prove learnability results for various classes of Boolean functions. But to the best of our knowledge, such methods are mainly used for theoretical analysis and generally not regarded as pragmatic learning algorithms by practitioners, probably due to the assumption of data being generated from uniform distribution, which is rarely true for a real application.

The main contribution of this work is to generalize the spectrum concepts for Boolean function to any input distribution and develop practically useful learning methods based on these ideas. Due to the different goals in mind, the methods and analysis we develop is quite different in style compared to classic Boolean function analysis.

For example, learning theory usually assumes random access to data generation oracle. In general there is no need to be economical about the training data as long as the order of magnitude of the sample complexity is unchanged. Rather, the main concern is to develop performance guarantees under different learning models (more details in Section 1.3). In contrast, we develop practical algorithms and care about their performance on specific datasets. It is usually not easy to obtain more samples, thus we work with a relatively simple learning model (PAC model) and much of the effort goes into improving practical learning performance by, for instance, providing appropriate regularization.

Another major difference is that learning theory usually work with well defined classes of functions, e.g. half-spaces, intersection of constant number of half-spaces, etc. The

[3]Of course, this is only possible because each dimension is much simpler: only two possible values.

[4]The Fourier calculations on the hypercube go back much further. For more details, please refer to the bibliography nodes of Chapter 3 in O'Donnell [2014].

hard work lies in proving spectral concentration results for such function classes. Instead, we work with given datasets and simply assume that the target function has its spectrum concentrated at low degrees. The focus is on proving how well we can estimate the spectrum from limited amount of available training samples.

1.2 Research Goals

This thesis seeks to define, understand and exploit the Fourier spectrum of Boolean functions under arbitrary data distributions.

On the *practical* side, we aim to bridge the gap between Boolean function analysis theory and real life machine learning problems. Efficient spectrum-based algorithms are proposed to generate and select non-linear features from raw data. Linear models trained in the new feature space remain to be conceptual simple and easy to interpret, but provide substantial gain in predictive performance compared to those trained on raw features. We analyse the proposed algorithms to provide performance guarantees, test them on extensive synthetic and real datasets, and offer hands-on tips on when and how to use the proposed algorithms.

On the *theoretical* side, the generalized Fourier spectrum offers a suitable framework for discussing data-dependent complexity of Boolean functions without assuming a particular learning algorithm[5], which in turn facilitates understanding of how the target function and the data distribution jointly impact the difficulty of a learning problem. Moreover, graphical models (directed graphs in particular) can be naturally combined with the spectral view of Boolean functions, and the generalized Fourier spectrum can be connected to fundamental quantities such as noise stability under sequential data generation setup.

[5]Note that the common complexity measure are either only w.r.t. the function (usually assuming a uniform distribution) e.g. circuit size, or tied to a particular algorithm, e.g. sample complexity.

1.3 Problem Setup

This thesis focuses on standard supervised learning tasks, namely *classification* and *regression*. More attention is given to classification tasks because arguably with categorical features, they are more common than regression ones in practice. However, the proposed algorithms apply to both cases and much of the analysis readily adapts.

Throughout the thesis, we encode categorical features and deal with *binary features* only (e.g. via one-hot encoding). The learning algorithm is given random access to training samples $\mathcal{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$, where $\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^N \in \{-1, 1\}^d$ are independent, identically distributed and drawn from distribution $p(\mathbf{x})$. Depending on how the labels y^n are generated, there are three models commonly studied in learning theory:

- **PAC model**: $\exists f(\cdot) : \{-1,1\}^d \mapsto \mathbb{R}$ s.t. $\forall n,\ y^n = f(\mathbf{x}^n)$. The goal is to approximate $f(\mathbf{x})$ from training samples accurately and with high probability. This is the 'noiseless' setup.

- **Random noise**: $\exists f(\mathbf{x}) : \{-1,1\}^d \mapsto \mathbb{R}$ s.t. $\forall n,\ y^n$ is a noisy version of $f(\mathbf{x}^n)$ and the noise is *independent of* $\mathbf{x}^n$, e.g. $y^n = f(\mathbf{x}^n) + \epsilon^n$, where $\epsilon^n \sim \mathcal{N}(0,1)$.

- **Agnostic model**: $\exists$ joint distribution $p(\mathbf{x}, y)$ from which the samples $(\mathbf{x}^n, y^n)$ are generated. In this case, $y \sim p(\cdot|\mathbf{x})$ and can be viewed as y taking a deterministic value plus some noise. But note that the noise here can depend on the input $\mathbf{x}$. This is the most difficult setup to provide learning theoretic guarantees for. In fact, many strong negative results are known (e.g. Kalai et al. [2005], Kearns et al. [1994], Lee et al. [1995]).

The assumption on learning model does not affect how the proposed algorithms are applied in practice, but theoretical guarantees differ significantly depending on the model. We mostly focus on the PAC model due to its amenability to analysis, and in due course it should become clear that the proposed algorithms are quite robust to random noise.

Learning methods often fall into one of two categories: *generative*, where the full joint distribution $p(\mathbf{x}, y)$ is modelled, and *discriminative*, where only the conditional distribution $p(y|\mathbf{x})$ is modelled. It is well-known that if assumptions on $p(\mathbf{x})$ are correct, generative models are better; if the assumptions are wrong, generative models can still be better with limited amount of training data, but asymptotically will be out-performed by the discriminative counterpart because the incorrect assumptions introduce extra bias (see, e.g. Ng and Jordan [2002]). The methods proposed in this thesis attempt to model the input distribution $p(\mathbf{x})$. However, information of $p(\mathbf{x})$ is only used for defining the feature mapping and the new features can be feed into any downstream learning method, generative or discriminative.

1.4 Organization of the Thesis

Chapter 2 introduces some necessary background from Boolean function analysis.

Boolean function analysis traditionally assumes uniform distribution over input space, which restricts its use to mainly theoretical proofs. We remove this constraint and generalize the concept of Fourier spectrum to arbitrary distribution in Chapter 3, and propose a few practically useful spectrum-based learning algorithms.

The generalized Fourier spectrum provides a framework for discussing data-dependent, algorithm-independent function complexity. Chapter 4 explores how the spectrum of a function varies under different input distributions, how input distribution and target function together impact the difficulty of a learning problem, and makes rigorous the intuition that *uniform distribution is the most difficult to learn from.*

Chapter 5 provides theoretical guarantees of the proposed algorithms under different assumptions of what we know about the input distribution, while Chapter 6 reports the methods' practical performance on various synthetic and real datasets. We also provide some practical tips on when and how to use the proposed algorithms.

Finally, Chapter 7 briefly summarizes the work and elaborates on a few ideas for further exploration.

Chapter 2

Fourier Spectrum of Boolean Functions

This chapter provides a brief introduction to the field of *Boolean function analysis*, which inspired the work in this thesis. Boolean function analysis is itself a rich and active field of research and books such as O'Donnell [2014] serve as great references. This chapter, however, only focus on the basic ideas that are directly related to our own investigations. Section 2.1 defines what we mean by Fourier spectrum in the context of Boolean functions, and Section 2.2 provides an overview of existing results on spectrum-based learning of Boolean functions.

2.1 Fourier Spectrum of Boolean Functions

A Boolean function is a mapping from $\{-1, 1\}^d$ to $\mathbb{R}$. Here $\{-1, 1\}$ can equivalently be $\{0, 1\}$ or $\{\text{True, False}\}$. An important special case is Boolean-valued Boolean functions $f : \{-1, 1\}^d \mapsto \{-1, 1\}$. Boolean functions have multiple representations, e.g. as a truth table, as a circuit, or as a decision tree. The Fourier decomposition of a Boolean function is simply its *multilinear polynomial representation.* In this representation, we stick to the $\{-1, 1\}$ alphabet following convention.

A Boolean function $f : \{-1, 1\}^d \mapsto \mathbb{R}$ can be viewed as a vector in $\mathbb{R}^{2^d}$. Given any set of orthonormal basis of $\mathbb{R}^{2^d}$, the function can be uniquely represented using its projections along these basis vectors. Such a set of orthonormal basis is not unique, but one of them has particularly nice interpretations, namely the set of all *parity functions*:

$$\mathcal{X}_S(\mathbf{x}) \triangleq \prod_{i \in S} x_i, \quad \forall S \subseteq \{1, 2, \cdots, d\} \triangleq [d] \tag{2.1}$$

A definition of inner product is necessary for discussion of orthonormality:

Definition 1. *Given functions f, g: $\{-1, 1\}^d \mapsto \mathbb{R}$, their inner product is defined as*

$$\langle f, g \rangle \triangleq \frac{1}{2^d} \sum_{\mathbf{x} \in \{-1,1\}^d} f(\mathbf{x}) g(\mathbf{x}) = \mathbb{E}[f(\mathbf{x}) g(\mathbf{x})] \tag{2.2}$$

Note that the expectation is with respect to the uniform distribution over $\{-1, 1\}^d$.

Proposition 1. *The 2^d parity functions $\mathcal{X}_S$ form an orthonormal basis, i.e.*

$$\langle \mathcal{X}_S, \mathcal{X}_T \rangle = \mathbb{1}(S = T), \quad \forall S, T \subseteq [d] \tag{2.3}$$

The following proposition provides explicit expressions for projecting a Boolean function onto each basis vector (i.e. each parity function):

Proposition 2. *Any $f : \{-1, 1\}^d \mapsto \mathbb{R}$ can be uniquely Fourier-decomposed as*

$$f(\mathbf{x}) = \sum_{S \subseteq [d]} \hat{f}(S) \mathcal{X}_S(\mathbf{x}) \ , \quad \textit{where } \hat{f}(S) = \langle f, \mathcal{X}_S \rangle, \forall S \subseteq [d] \tag{2.4}$$

Collectively, the coefficients $\{\hat{f}(S)\}_{S \subseteq [d]}$ is called the **Fourier spectrum** of function f and the coefficient $\hat{f}(S)$ is referred to as the **Fourier coefficient** of set S. The following proposition shows how the spectrum of f is related to its l_2-norm:

Proposition 3 (Parseval's Theorem). $\forall f : \{-1, 1\}^d \mapsto \mathbb{R}$,

$$\langle f, f \rangle = \mathbb{E}[f^2(\mathbf{x})] = \sum_{S \subseteq [d]} [\hat{f}(S)]^2 \tag{2.5}$$

Proof of Proposition 3.

$$\begin{aligned}\langle f,f\rangle &= \langle \sum_{S\subseteq[d]} \hat{f}(S)\mathcal{X}_S(\mathbf{x}), \sum_{T\subseteq[d]} \hat{f}(T)\mathcal{X}_T(\mathbf{x})\rangle = \sum_{S,T\subseteq[d]} \hat{f}(S)\hat{f}(T)\langle \mathcal{X}_S(\mathbf{x}), \mathcal{X}_T(\mathbf{x})\rangle \\ &= \sum_{S,T\subseteq[d]} \hat{f}(S)\hat{f}(T)\mathbb{1}(S=T) = \sum_{S\subseteq[d]} [\hat{f}(S)]^2\end{aligned}$$

□

Note that in the special case of Boolean-valued Boolean functions $f : \{-1,1\}^d \mapsto \{-1,1\}$, $\sum_{S\subseteq[d]}[\hat{f}(S)]^2 = 1$ and $[\hat{f}(S)]^2 \geq 0$, thus $[\hat{f}(S)]^2$ can be viewed as the 'probability' or 'weight' of set S.

Proposition 4 (Plancherel's Theorem). $\forall f, g : \{-1,1\}^d \mapsto \mathbb{R}$,

$$\langle f,g\rangle = \mathbb{E}[f(\mathbf{x})g(\mathbf{x})] = \sum_{S\subseteq[d]} \hat{f}(S)\hat{g}(S) \tag{2.6}$$

Proof of Proposition 4.

$$\begin{aligned}\langle f,g\rangle &= \langle \sum_{S\subseteq[d]} \hat{f}(S)\mathcal{X}_S(\mathbf{x}), \sum_{T\subseteq[d]} \hat{g}(T)\mathcal{X}_T(\mathbf{x})\rangle = \sum_{S,T\subseteq[d]} \hat{f}(S)\hat{g}(T)\langle \mathcal{X}_S(\mathbf{x}), \mathcal{X}_T(\mathbf{x})\rangle \\ &= \sum_{S,T\subseteq[d]} \hat{f}(S)\hat{g}(T)\mathbb{1}(S=T) = \sum_{S\subseteq[d]} \hat{f}(S)\hat{g}(S)\end{aligned}$$

□

As a concrete example, Figure 2-1 shows the truth table of the 3-Majority function and its Fourier decomposition.

x1	**1**	**1**	**1**	**1**	**-1**	**-1**	**-1**	**-1**
x2	**1**	**1**	**-1**	**-1**	**1**	**1**	**-1**	**-1**
x3	**1**	**-1**	**1**	**-1**	**1**	**-1**	**1**	**-1**
f(x)	**1**	**1**	**1**	**-1**	**1**	**-1**	**-1**	**-1**

$$f(\mathbf{x}) = \frac{1}{2}x_1 + \frac{1}{2}x_2 + \frac{1}{2}x_3 - \frac{1}{2}x_1x_2x_3$$

Figure 2-1: Concrete example: truth table and Fourier decomposition of 3-Majority function: $f(\mathbf{x}) = \mathrm{sgn}(x_1 + x_2 + x_3)$.

2.2 Spectrum-Based Learning

Assume the PAC learning model described in Section 1.3: given i.i.d. training samples $\mathcal{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$, where $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N$ are generated from distribution $p(\mathbf{x})$ and $\forall n$, $y^n = f(\mathbf{x}^n)$ where $f : \{-1,1\}^d \mapsto \mathbb{R}$ is the target function. The goal is to learn some $g(\mathbf{x})$ that approximates the target function with high accuracy. In classification tasks we want the classification error $\mathbb{E}_{\mathbf{z}\sim p}[\mathbb{1}(f(\mathbf{z}) \neq g(\mathbf{z}))]$ to be small and in regression tasks we aim to minimize the mean square error $\mathbb{E}_{\mathbf{z}\sim p}[(f(\mathbf{z}) - g(\mathbf{z}))^2]$.

This is of course a very general setup. We are particularly interested in **spectrum-based learning**, i.e. approximating the target function via its Fourier spectrum. Section 2.2.1 rigorously justifies why good approximation of the spectrum lead to good prediction performance. We then move on to present some well-known spectrum-based learning algorithms: Section 2.2.2 describes the celebrated *Low Degree Algorithm* (Linial and Mansour [1993]), and Section 2.2.3 presents the *Polynomial Regression Algorithms* (Kalai et al. [2005]).

2.2.1 Approximating A Boolean Function via Its Spectrum

The Fourier spectrum, i.e. the set of 2^d Fourier coefficients $\{\hat{f}(S)\}_{S\subseteq[d]}$, uniquely determines any Boolean function $f : \{-1,1\}^d \mapsto \mathbb{R}$. Given independent samples $\mathbf{x}^1, \mathbf{x}^2, \cdots \mathbf{x}^N$ drawn from uniform distribution over $\{-1,1\}^d$ and their corresponding outputs $f(\mathbf{x}^1), f(\mathbf{x}^2), \cdots, f(\mathbf{x}^N)$, Law of Large Numbers states that each coefficient $\hat{f}(S)$ can be approximated to arbitrary accuracy with high probability, given large enough N. But *does accurate estimation of the Fourier coefficients necessarily lead to accurate reconstruction of the target function*?

For regression, it is a simple corollary of the Parseval's Theorem (Proposition 3) to directly equate the spectral estimation error $\sum_{S\subseteq[d]}(\hat{f}(S) - \hat{g}(S))^2$ to the mean square error in function value $\mathbb{E}[(f(\mathbf{x}) - g(\mathbf{x}))^2]$. For classification, some additional complications arise due to the quantization step needed to output a class label. Yet the following proposition provides an affirmative answer.

Proposition 5 (Proposition 3.31 of O'Donnell [2014])**.** *Suppose* $f : \{-1,1\}^d \mapsto \{-1,1\}$ *and* $h : \{-1,1\}^d \mapsto \mathbb{R}$ *satisfy* $\sum_{S\subseteq[d]} |\hat{f}(S) - \hat{h}(S)|^2 \leq \epsilon$*. Let* $g : \{-1,1\}^d \mapsto \{-1,1\}$ *be defined by* $g(\mathbf{x}) = \mathrm{sgn}(\mathrm{h}(\mathbf{x}))$*, with* $\mathrm{sgn}(0)$ *chosen arbitrarily. Then*

$$\mathbb{P}(f(\mathbf{x}) \neq g(\mathbf{x})) \leq \epsilon$$

2.2.2 Spectral Concentration and Low Degree Algorithm

Proposition 5 naturally leads to a meta learning algorithm: given the training samples, estimate each Fourier coefficient using the corresponding sample mean and plug into Equation (2.4). For classification problems, output the sign of the result.

However, this method is computationally infeasible as there are 2^d coefficients to estimate. Note that we have not yet made any assumptions on the target function $f(\mathbf{x})$. According to *No Free Lunch Theorem*, one cannot learn without making assumptions. The real question is, what assumptions?

On a philosophical level, we implicitly assume that the functions we ever care to learn and understand should be simple and highly structured. Of course, these are vague and non-technical terms, and we defer a more formal discussion on function complexity measures to Chapter 4. In the context of spectrum-based learning, Boolean functions with good *spectral concentration* properties are considered simple.

Definition 2. *Fourier spectrum of* $f : \{-1,1\}^d \mapsto \mathbb{R}$ *is said to be* ϵ*-concentrated on* $\mathcal{C}$*, where* $\mathcal{C}$ *is some collection of subsets of* $[d]$*, if* $\sum_{S\notin\mathcal{C}}[\hat{f}(S)]^2 \leq \epsilon$*. In particular,* f *is* ϵ*-concentrated on degree up to* d_0 *if* $\sum_{S\subseteq[d],|S|>d_0}[\hat{f}(S)]^2 \leq \epsilon$

If the target function $f(\mathbf{x})$ is ϵ-concentrated on some $\mathcal{C}$ where $|\mathcal{C}| \ll 2^d$, the meta learning algorithm described above can be made feasible. More rigorously:

Proposition 6 (Theorem 3.29 in O'Donnell [2014])**.** *Assume the Fourier spectrum of target function* $f(\mathbf{x})$ *is* $\epsilon/2$*-concentrated on* $\mathcal{C}$*. Then the meta algorithm, when given* *random access to i.i.d training samples, can output a hypothesis $g(\mathbf{x})$ that is ϵ-close to $f(\mathbf{x})$ (i.e. $\mathbb{P}(f(\mathbf{x}) \neq g(\mathbf{x})) \leq \epsilon$) with probability at least $1-\delta$ in time polynomial in $|\mathcal{C}|$, d, $1/\epsilon$ and $\log(1/\delta)$.*

The most common choice of the collection of sets is probably low-cardinality sets: $\mathcal{C} = \{S \subseteq [d] : |S| \leq d_0\}$, where d_0 is a constant much smaller than d. In this case, the meta algorithm becomes the *Low Degree Algorithm*. Apply Proposition 6 to this special case, the algorithm has $\text{Poly}(d^{d_0}, 1/\epsilon)$ running time.

The spectral concentration at low-cardinality sets can essentially be viewed as a *smoothness* assumption: note how $\mathcal{X}_{\varnothing}(\mathbf{x}) \equiv 1$ is a constant function and $\mathcal{X}_{[d]}(\mathbf{x})$ has the highest 'switching frequency'. Roughly speaking, smooth functions have their spectrum concentrated at low degree terms. Proposition 7 provides a rigorous statement through a quantity called *total influence* and assuming a classification task. Similar arguments can be made for other quantities such as noise stability and other setup such as regression tasks.

Definition 3. *The influence of coordinate i on $f : \{-1,1\}^d \mapsto \{-1,1\}$ is defined to be the probability that i is pivotal for a uniformly randomly chosen input $\mathbf{x}$:*

$$\text{Inf}_i[f] = \mathbb{P}[f(\mathbf{x}) \neq f(\mathbf{x}^{\oplus i})] \tag{2.7}$$

*where $\mathbf{x}^{\oplus i}$ denotes $(x_1, \cdots, x_{i-1}, -x_i, x_{i+1}, \cdots, x_d)$, i.e. the result of flipping the i^{th} coordinate of $\mathbf{x}$. And the **total influence** of $f(\mathbf{x})$ is defined as*

$$\text{I}[f] \triangleq \sum_{i=1}^{d} \text{Inf}_i[f] \tag{2.8}$$

Proposition 7. *For any $f : \{-1,1\}^d \mapsto \{-1,1\}$, and $\forall \epsilon > 0$, the Fourier spectrum of f is ϵ-concentrated on degree up to $\text{I}[f]/\epsilon$.*

As discussed in Chapter 1, existing works on spectrum-based learning of Boolean functions are mostly found in computational learning theory and the focus is on the theoretical side, e.g. proving learnability or hardness results.

To the best of our knowledge, Low Degree algorithm is not treated as a pragmatic method in the machine learning community. In fact, without modifications, it provides significantly worse performance on both real and synthetic datasets compared to popular learning methods such as Support Vector Machines, Random Forests, etc. Among others, some major obstacles are identified as follows.

Firstly, the classic Boolean function analysis literature assumes **uniform input distribution** $p(\mathbf{x})$. Although it is a reasonable assumption in learnability or hardness results[1], it is almost never true in a real dataset. Thus the Fourier coefficients cannot be directly approximated by sample mean.

Many results in Boolean function analysis can be generalized to the case where $p(\mathbf{x})$ is a product measure (see, e.g. Chapter 8 of O'Donnell [2014], Blais et al. [2008]). Yet even product distribution is not a good enough approximation for many real datasets.

Secondly, the Low Degree algorithm is fundamentally a regression method rather than a classification one since it (approximately) finds the function that minimizes mean square error. In theory a good regressor implies a good classifier. However in practice, given limited amount of data, dedicated classification methods generally perform significantly better than a regression method used for classification purposes by, say, thresholding the output.

Even with a regression task, the Low Degree algorithm estimates each coefficient separately. Although asymptotically it will output the optimal solution, with finite training data, it is often out-performed by methods that estimate the coefficients jointly, e.g. Linear Regression or Support Vector Machines trained on the same

[1]Intuitively one expects uniform distribution is the most difficult to learn from, more discussion on this in Chapter 4)

features $\{\mathcal{X}_S(\mathbf{x})\}_{S\in\mathcal{C}}$.

A third problem with the Low Degree algorithm is its lack of appropriate regularization. While in a learnability result, one usually only care about proving that the order of magnitude of the sample or time complexity, for a real dataset, *constants matter a lot*. Most pragmatic learning methods utilize some form of regularization to enforce simpler models, prevent overfitting and in turn improve learning performance.

The spectrum-based learning methods we propose in Chapter 3 tackle all three obstacles listed above, but first let us present the *Polynomial Regression* algorithms in Section 2.2.3 which, though still not generally considered practical, does not require uniform input distribution.

2.2.3 Polynomial Regression Algorithm

Kalai et al. [2005] points out that Low Degree algorithm can be viewed as a special case of *Least-Squares Polynomial Regression* where input distribution $p(\mathbf{x})$ is known to be uniform. In general, Least-Squares Polynomial Regression attempts to solve

$$\min_{\mathrm{deg(A)\leq d_0}} \mathbb{E}[(A(\mathbf{x})-y)^2] \tag{2.9}$$

where $A(\mathbf{x})$ is a polynomial and the expectation is over joint distribution $p(\mathbf{x},y)$. Given training data, the objective function in Equation (2.9) is usually approximated by an empirical version $\sum_{n=1}^{N}(A(\mathbf{x}^n)-y^n)^2/N$. Of course a polynomial is simply a sum of monomials and if we consider each monomial as a feature, the polynomial regression can in turn be viewed as a standard linear regression problem and can be solved by a single matrix inversion.

In the special case of uniform $p(\mathbf{x})$, Equation (2.9) can be solved in closed form:

$$\arg\min_{\mathrm{deg(A)\leq d_0}} \mathbb{E}_{\mathrm{uniform}}\left[(A(\mathbf{x})-y)^2\right] = \sum_{S\subseteq[d]:|S|\leq d_0} \hat{f}(S)\mathcal{X}_S(\mathbf{x}) \tag{2.10}$$

where $\hat{f}(S)$ is exactly the Fourier coefficient defined in Equation (2.4). Thus the optimal 'weight' of each monomial can be individually estimated from data, giving rise to the Low Degree algorithm.

With non-uniform $p(\mathbf{x})$, Kalai et al. [2005] proposes to run the general Least-Squares Polynomial Regression algorithm. This requires matrix inversion or iterative methods, both of which will incur significant computational cost. We propose algorithms in Chapter 3 that work for arbitrary $p(\mathbf{x})$ but still estimate each coefficient independently.

Analogous to the least-square polynomial regression algorithm, Kalai et al. [2005] also discussed a L1-norm version, where the objective becomes

$$\min_{\mathrm{deg(A)} \leq \mathrm{d_0}} \mathbb{E}[|A(\mathbf{x}) - y|] \approx \min_{\mathrm{deg(A)} \leq \mathrm{d_0}} \frac{1}{N} \sum_{n=1}^{N} |A(\mathbf{x}^n) - y^n| \tag{2.11}$$

The optimization problem in Equation (2.11) can be viewed and solved as a standard *L1 linear regression problem*. L1 linear regression, also known under the name *Least Absolute Deviation*, is a well-studied problem. It is generally known to be more robust than least-squares regression, but unlike least-squares, the solutions are neither stable nor unique. L1 linear regression does not have an analytical solution and an iterative one is necessary.

As Kalai et al. [2005] pointed out, L1 Polynomial Regression Algorithm has its theoretical importance because it provides sharp guarantees in the agnostic learning setup, which is much harder to deal with than the PAC model. But for practical purposes, the optimization problem in Equation (2.11) is closely related to SVM with polynomial kernel and can be solved by standard SVM package. In experiments in Chapter 6, SVM with polynomial kernel is used as a baseline for comparison and we show that the methods we propose in Chapter 3 outperforms this baseline.

Chapter 3

Fourier Spectrum under Arbitrary Input Distribution

Most Boolean function analysis literature makes the assumption that the training data is generated by a uniform distribution over $\{-1, 1\}^d$. This choice makes a lot of sense in the context of theoretical analysis, where usually no extra information about the distribution is available. Moreover, uniform distribution is intuitively the most difficult to learn from as non-uniform distribution can be viewed as providing some 'focus'[1], thus the theory is dealing with the worst case scenario.

In practice, however, the goal is to learn from a particular dataset. It is very rare that a real dataset is generated from uniform distribution. The low degree algorithm in Section 2.2.2 cannot be directly applied because the Fourier coefficients can no longer be approximated from samples, and while Polynomial Regression algorithms (Section 2.2.3) do not explicitly require uniform input distribution and thus are applicable, their performances are not competitive.

This chapter presents the main contribution of this thesis, i.e. **generalizing the definition of Fourier spectrum of Boolean functions to any distribution and proposing practically useful spectrum-based learning algorithms**.

[1]We make this intuition rigorous in the context of spectrum-based learning in Chapter 4.

The rest of this chapter is organized as follows:

- Section 3.1 extends the definition of Fourier spectrum to arbitrary distribution and Section 3.2 compares the generalized definition to the classic one and comments on some of their similarities and differences.

- Section 3.3 describes a direct generalization of the Low Degree algorithm and Section 3.4 discusses its connections to the family of nonparametric regression models called *linear smoothers*.

- It turns out that the direct generalization presented in Section 3.3 is still not practically useful, and Section 3.5 proposes two more variants that generate and select features based on the spectral perspective. Combined with appropriate choice of downstream learning algorithms, these variants produce models that are both simple and powerful.

3.1 Fourier Spectrum beyond Uniform Distribution

There are some existing works that generalize the theory from uniform distribution to product measures. In fact, as commented in O'Donnell [2014], much of the treatment in Boolean function analysis depends only on viewing the domain as a product probability distribution. We briefly present some existing results on Fourier spectrum for product spaces in Section 3.1.1.

It should become clear that the generalization techniques in Section 3.1.1 do not easily carry over to non-product distributions. Instead, we provide our definition of *orthonormal basis* and *Fourier spectrum* for arbitrary input distribution in Section 3.1.2. To discuss orthonormality under a non-uniform distribution, we first need to modify the definition of inner product from Equation (2.2):

Definition 4. *Under $p(\mathbf{x})$, the inner product of functions $f, g : \{-1,1\}^d \mapsto \mathbb{R}$ is*

$$\langle f, g\rangle_p \triangleq \mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})g(\mathbf{x})] = \sum_{\mathbf{x}} p(\mathbf{x})f(\mathbf{x})g(\mathbf{x}) \tag{3.1}$$

3.1.1 Existing Works on Product Measures

The most commonly considered generalized domain in Boolean function analysis is the case of the hypercube with 'biased' bits: a random input from $\{-1,1\}^d$ is generated by independently setting each bit to -1 with probability p and to 1 with probability $1-p$, where $p \in (0,1)$. The distribution is denoted as $\pi_p^{\otimes d}$. This setup is often referred to as *p-biased Fourier analysis*[2].

Definition 5 (Definition 8.39 in O'Donnell [2014]). *In the context of p-biased Fourier analysis, we define $\phi : \{-1,1\} \mapsto \mathbb{R}$, the basis function for coordinate i, by*

$$\phi(x_i) \triangleq \frac{x_i - \mu}{\sigma}$$

where $\mu = \mathbb{E}[x_i] = 1-2p$ and $\sigma = \sqrt{Var(x_i)} = 2\sqrt{p(1-p)}$.

Proposition 8. *Use Definition 5, the functions below form an orthonormal basis under distribution $\pi_p^{\otimes d}$:*

$$\phi_S(\mathbf{x}) \triangleq \prod_{i\in S} \phi(x_i), \quad \forall S \subseteq [d] \tag{3.2}$$

Given a function $f : \{-1,1\}^d \mapsto \mathbb{R}$, we can decompose it under $\pi_p^{\otimes d}$ as

$$f(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}(S)\phi_S(\mathbf{x}), \qquad \textit{where } \hat{f}(S) = \langle f, \phi_S\rangle_{\pi_p^{\otimes d}}$$

The definition of Fourier spectrum over product distribution can also be generalized beyond the hypercube and allow an alphabet size larger than 2.

Definition 6 (Definition 8.12 in O'Donnell [2014]). *Let (Ω, π) be a finite probability space where $|\Omega| \triangleq m \geq 2$ and π has full support. A **Fourier basis** for the inner*

[2]It is actually easy to generalize to the case where each bit has a different p_i but we stick to the same p to simplify presentation.

*product space $L^2(\Omega, \pi)$ is an orthonormal basis $\phi_0, \phi_1 \cdots \phi_{m-1}$ with $\phi_0 \equiv 1$. An **n-dimensional multi-index** is a tuple $\alpha \in \mathbb{N}^d$ where $\alpha_i \in \{0, 1, \cdots, m-1\}$. For each multi-index α, we define $\phi_\alpha \in L^2(\Omega^d, \pi^{\otimes d})$ by*

$$\phi_\alpha(\mathbf{x}) = \prod_{i=1}^{d} \phi_{\alpha_i}(x_i) \tag{3.3}$$

Proposition 9 (Proposition 8.13 in O'Donnell [2014])**.** *The functions $\{\phi_\alpha\}_\alpha$ defined in Equation (3.3) form a Fourier basis for inner product space $L^2(\Omega^d, \pi^{\otimes d})$. Any function f can be uniquely expressed as*

$$f(\mathbf{x}) = \sum_{\alpha \in \mathbb{N}^d : 0 \le \alpha_i < m} \hat{f}(\alpha)\phi_\alpha, \qquad \textit{where } \hat{f}(\alpha) \triangleq \langle f, \phi_\alpha \rangle_{\pi^{\otimes d}}$$

It is not hard to see that the construction of the orthonormal basis in Equations (3.2) and (3.3) both depend vitally on the distribution being a product one and this decomposability assumption seems hard to relax. In the section below, we present an orthonormal basis that makes no such assumption on the input distribution.

3.1.2 Generalized Fourier Spectrum

Given any input distribution $p(\mathbf{x})$, we can apply the chain rule

$$p(\mathbf{x}) = p(x_1)p(x_2|x_1) \cdots p(x_d|x_1^{d-1}) = \prod_{i=1}^{d} p(x_i|\mathbf{x}_{\pi_i}) \tag{3.4}$$

Here $\mathbf{x}_{\pi_i}$ denotes the set of parents of variable x_i. Without any assumption on $p(\mathbf{x})$, $\mathbf{x}_{\pi_i} = \mathbf{x}_1^{i-1}$, but for practical applications, we expect each node to have only a small number of parent nodes. Based on Equation (3.4), the following theorem provides a set of orthonormal basis under $p(\mathbf{x})$:

Theorem 1. *The following 2^d functions form a set of orthonormal basis $\{\phi_S\}_{S \subseteq [d]}$:*

$$\phi_S^p(\mathbf{x}) = \prod_{i \in S} \phi_{\{i\}}^p(\mathbf{x}) \qquad \textit{where } \phi_{\{i\}}^p(\mathbf{x}) = x_i \sqrt{\frac{p(-x_i|\mathbf{x}_{\pi_i})}{p(x_i|\mathbf{x}_{\pi_i})}}, \forall i \in [d] \tag{3.5}$$

In particular, $\phi^p_\varnothing(\mathbf{x}) \equiv 1$

It is not hard to check the correctness of the theorem:

Proof of Theorem 1. Let $S, T \subseteq [d]$ and $S \neq T$. Let $S \triangle T \triangleq (S \setminus T) \cup (T \setminus S)$.

$$\langle \phi^p_S(\mathbf{x}), \phi^p_S(\mathbf{x})\rangle_p = \sum_{\mathbf{x}} p(\mathbf{x}) \prod_{i\in S} \frac{p(-x_i|\mathbf{x}_{\pi_i})}{p(x_i|\mathbf{x}_{\pi_i})} = \sum_{\mathbf{x}} \Big[\prod_{i\in S} p(-x_i|\mathbf{x}_{\pi_i})\Big]\Big[\prod_{j\notin S} p(x_j|\mathbf{x}_{\pi_j})\Big] = 1 \tag{3.6}$$

$$\begin{aligned}\langle \phi^p_S(\mathbf{x}), \phi^p_T(\mathbf{x})\rangle_p &= \sum_{\mathbf{x}} p(\mathbf{x}) \left[\prod_{i\in S} x_i \sqrt{\frac{p(-x_i|\mathbf{x}_{\pi_i})}{p(x_i|\mathbf{x}_{\pi_i})}}\right]\left[\prod_{j\in T} x_j \sqrt{\frac{p(-x_j|\mathbf{x}_{\pi_j})}{p(x_j|\mathbf{x}_{\pi_j})}}\right] \\ &= \sum_{\mathbf{x}} \Big[\prod_{i\in S\bigcap T} p(-x_i|\mathbf{x}_{\pi_i})\Big]\Big[\prod_{j\notin S\bigcup T} p(x_j|\mathbf{x}_{\pi_j})\Big] \times \Big[\prod_{k\in S\triangle T} x_k \sqrt{p(x_k|\mathbf{x}_{\pi_k})p(-x_k|\mathbf{x}_{\pi_k})}\Big] = 0\end{aligned} \tag{3.7}$$

The last equality uses the fact that $S \triangle T \neq \varnothing$ if and only if $S \neq T$. $\square$

We can now decompose f according to the new basis:

$$f(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}^p(S)\phi^{(p)}_S(\mathbf{x}) \quad \text{where } \hat{f}^p(S) = \langle f, \phi^p_S\rangle_p \tag{3.8}$$

Similar to the uniform case, the collection of all Fourier coefficients $\{\hat{f}^p(S)\}_{S\subseteq[d]}$ is referred to as the **(Generalized) Fourier spectrum** of function f under distribution $p(\mathbf{x})$. Given training samples $\{\mathbf{x}^n\}_{n=1}^N$ generated i.i.d. from $p(\mathbf{x})$, $\hat{f}^p(S)$ can be approximated by the corresponding sample mean $\sum_{n=1}^N f(\mathbf{x}^n)\phi^p_S(\mathbf{x}^n)/N$. A quick note: we use the notations $\phi^p_S(\mathbf{x})$ and $\hat{f}^p(S)$ if $p(\mathbf{x})$ is non-uniform, and the notations $\mathcal{X}_S(\mathbf{x})$ and $\hat{f}(S)$ when $p(\mathbf{x})$ is uniform.

Let us make a few comments on the generalized Fourier basis in Equation (3.5):

1. We still have the interpretation that each basis vector $\phi^p_S(\mathbf{x})$ is associated with a subset $S \subseteq [d]$. In fact, if $p(\mathbf{x})$ is indeed uniform, the definition in Theorem 1 falls back to the classic one.

2. Decomposability of $\phi^p_S(\mathbf{x})$ still holds, i.e. $\phi^p_S(\mathbf{x}) = \prod_{i \in S} \phi^p_{\{i\}}(\mathbf{x})$. This property turns out to be very useful in proofs.

3. The definition of this orthonormal basis depends on the topological ordering, i.e. how chain rule is applied. Different ordering will result in different basis. For spectrum-based learning to be feasible, we implicitly assume availability of a 'good' ordering, i.e. each x_i has only a small number of parent nodes.

4 While $\mathcal{X}_S(\mathbf{x})$ only depends on variables $\{x_i\}_{i \in S}$, $\phi^p_S(\mathbf{x})$ in general depends on more coordinates. In fact, Proposition 10 shows that if we want *decomposability* to hold, requiring $\phi^p_S(\mathbf{x})$ to depend only on $\mathbf{x}_S$ for all $S \subseteq [d]$ is possible if and only if $p(\mathbf{x})$ is a product distribution. Moreover, Proposition 11 states that the basis defined in Theorem 1 is unique (up to choice of signs) if we require both *decomposability* and *causal dependence*, i.e. $\phi^p_S(\mathbf{x})$ does not depend on any node that comes after all variables in S in the topological ordering.

Proposition 10. *Assume $p(\mathbf{x})$ is a non-product distribution, i.e. $\nexists \quad p_1, p_2, \cdots, p_d : \{-1,1\} \mapsto [0,1]$ s.t. $p(\mathbf{x}) = \prod_{i=1}^d p_i(x_i)$. Then one cannot find a set of functions $\{h_S : \{-1,1\}^d \mapsto \mathbb{R}\}_{S \subseteq [d]}$ that satisfy the following conditions simultaneously:*

1. *$h_S(\mathbf{x})$ only depends on the coordinates in set S (i.e. $\mathbf{x}_S$)*

2. *$\forall S$ s.t. $|S| > 1$, $h_S(\mathbf{x}) = \prod_{i \in S} h_{\{i\}}(\mathbf{x})$*

3. *$\langle h_S, h_T \rangle_p = \mathbb{1}(S = T)$*

Proposition 11. *Let $1, 2, \cdots, d$ be a topological ordering for $p(\mathbf{x})$. For $\forall S \subseteq [d]$, let $\Psi^p_S(\mathbf{x}) : \{-1,1\}^d \mapsto \mathbb{R}$. Denote the largest element in S (i.e. the last one in the topological ordering) as $\max(S)$. If $\Psi^p_S(\mathbf{x}) = \Psi^p_S(\mathbf{x}_{\text{Ancestor}(S)})$ for any S, where $\mathbf{x}_{\text{Ancestor}}(S)$ denotes all coordinates that are ancestors of any coordinate in S (the coordinates in S themselves are also included) and $\Psi^p_S(\mathbf{x}) = \prod_{i \in S} \Psi^p_{\{i\}}(\mathbf{x})$, $\forall S$, $\forall \mathbf{x}$. Then for any S,*

$$\Psi^p_S(\mathbf{x}) = \phi^p_S(\mathbf{x}) \text{ or } \Psi^p_S(\mathbf{x}) = -\phi^p_S(\mathbf{x}) \tag{3.9}$$

where $\phi^p_S(\mathbf{x})$ is defined as in Theorem 1.

The proof of the above propositions can be found in Appendix A. We wrap up this section by two concrete examples.

Example 1. *Consider the 3-Majority function shown in Figure 2-1, and a product distribution $p(\mathbf{x}) = p(x_1)p(x_2)p(x_3)$ with $\mathbb{P}(x_i = -1) = p$, $i = 1, 2, 3$. The values of the basis vectors are summarized in Table 3.1, where*

$$a \triangleq \sqrt{\frac{p}{1-p}}, \quad b \triangleq -\sqrt{\frac{1-p}{p}}$$

And the Fourier decomposition is

$$y = \left(1 - 6p^2 + 4p^3\right) \cdot 1 + \sum_{i=1}^{3} \left(4p^{3/2}(1-p)^{3/2}\right) \cdot \phi^p_{\{i\}}(\mathbf{x}) \\ + \sum_{1 \le i < j \le 3} \left(2p(1-p)(2p-1)\right) \cdot \phi^p_{\{i,j\}}(\mathbf{x}) + \left(-4p^{3/2}(1-p)^{3/2}\right)\phi^p_{\{1,2,3\}}(\mathbf{x})$$

x_1	1	1	1	1	-1	-1	-1	-1
x_2	1	1	-1	-1	1	1	-1	-1
x_3	1	-1	1	-1	1	-1	1	-1
$\phi^p_{\{1\}}(\mathbf{x})$	a	a	a	a	b	b	b	b
$\phi^p_{\{2\}}(\mathbf{x})$	a	a	b	b	a	a	b	b
$\phi^p_{\{3\}}(\mathbf{x})$	a	b	a	b	a	b	a	b
y	1	1	1	-1	1	-1	-1	-1

Table 3.1: Value table for Example 1.

Example 2. *Again consider the 3-Majority function, but this time with a Markov Chain structured $p(\mathbf{x}) = p(x_1)p(x_2|x_1)p(x_3|x_2)$. For simplicity of presentation, assume $\mathbb{P}(x_1 = -1) = 0.5$, and $\mathbb{P}(x_{i+1} = -x_i | x_i = x_i) = p$, $i = 1, 2$, $x_i \in \{-1, 1\}$. Here we have*

$$\phi^p_{\{1\}}(\mathbf{x}) = x_1\sqrt{\frac{p(-x_1)}{p(x_1)}} \qquad \phi^p_{\{2\}}(\mathbf{x}) = x_2\sqrt{\frac{p(-x_2|x_1)}{p(x_2|x_1)}} \qquad \phi^p_{\{3\}}(\mathbf{x}) = x_3\sqrt{\frac{p(-x_3|x_2)}{p(x_3|x_2)}}$$

The values of the basis vectors are summarized in Table 3.2, where a and b are defined the same way as in Example 1. The corresponding Fourier decomposition is

$$\begin{aligned} y = {} & 0 \cdot 1 + \left(1 - 2p + 2p^2\right) \cdot \phi^p_{\{1\}}(\mathbf{x}) + \left(2p^{1/2}(1-p)^{3/2}\right) \cdot \phi^p_{\{2\}}(\mathbf{x}) \\ & + \left(2p^{3/2}(1-p)^{1/2}\right) \cdot \phi^p_{\{3\}}(\mathbf{x}) + 0 \cdot \phi^p_{\{1,2\}}(\mathbf{x}) + 0 \cdot \phi^p_{\{1,3\}}(\mathbf{x}) + 0 \cdot \phi^p_{\{2,3\}}(\mathbf{x}) \\ & + \left(-2p(1-p)\right)\phi^p_{\{1,2,3\}}(\mathbf{x}) \end{aligned}$$

x_1	1	1	1	1	-1	-1	-1	-1
x_2	1	1	-1	-1	1	1	-1	-1
x_3	1	-1	1	-1	1	-1	1	-1
$\phi^p_{\{1\}}(\mathbf{x})$	1	1	1	1	-1	-1	-1	-1
$\phi^p_{\{2\}}(\mathbf{x})$	a	a	b	b	-b	-b	-a	-a
$\phi^p_{\{3\}}(\mathbf{x})$	a	b	-b	-a	a	b	-b	-a
y	1	1	1	-1	1	-1	-1	-1

Table 3.2: Value table for Example 2.

3.2 Understanding the Generalized Fourier Spectrum

Recall that the Fourier spectrum under uniform $p(\mathbf{x})$ is not just any orthonormal decomposition, but one with nice interpretations and provides useful tools for studying properties of the target functions. It turns out that the generalized Fourier spectrum defined in Theorem 1 naturally carries over many of such properties and intuitions.

For example, in classic Boolean function analysis, we have several basic identities relating the target function values to the Fourier spectrum, including the *Parseval's Theorem* and the *Plancherel's Theorem*. In fact, there identities only depend on the orthonormality of the basis and thus still hold for the generalized Fourier spectrum, as stated in Proposition 12.

Proposition 12. *Given function $f, g : \{-1,1\}^d \mapsto \mathbb{R}$ and distribution $p(\mathbf{x})$, f can be decomposed as $f(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}^p(S)\phi_S^p(\mathbf{x})$ and $g(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{g}^p(S)\phi_S^p(\mathbf{x})$ where $\{\phi_S^p(\mathbf{x})\}_{S\subseteq[d]}$ are defined as in Theorem 1. Then:*

$$\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] = \hat{f}^p(\varnothing) \tag{3.10}$$

$$\mathbb{E}_{\mathbf{x}\sim p}[f^2(\mathbf{x})] = \sum_{S\subseteq[d]} [\hat{f}^p(S)]^2 \qquad \textbf{\textit{(Parseval's Theorem)}} \tag{3.11}$$

$$\mathrm{Var}_{\mathbf{x}\sim p}(f(\mathbf{x})) = \sum_{S\neq\varnothing} [\hat{f}^p(S)]^2 \tag{3.12}$$

$$\langle f, g\rangle_p = \sum_{S\subseteq[d]} \hat{f}^p(S)\hat{g}^p(S) \qquad \textbf{\textit{(Plancherel's Theorem)}} \tag{3.13}$$

Again the proof can be found in Appendix A.

As a more interesting example of how ideas and analysis can be extended to generalized Fourier spectrum, we consider a quantity called *Noise Stability*. Noise stability in a way measures the 'smoothness' of a function and it is closely related to the Fourier spectrum. In fact it is one of the major tools for proving spectral concentration in Boolean function analysis (see, e.g. Klivans et al. [2004], Blais et al. [2008]). Let us first present the usual definition of noise stability (i.e. under uniform $p(\mathbf{x})$) and its connection to the Fourier spectrum. We then generalize the definition to arbitrary $p(\mathbf{x})$ and show a similar connection to the generalized Fourier spectrum.

Definition 7. *Let $\rho \in [0,1)$. For fixed $\mathbf{x} \in \{-1,1\}^d$ we write $\mathbf{y} \sim N_\rho(\mathbf{x})$ to denote that the random vector $\mathbf{y}$ is drawn as follows: for each $i \in [d]$ independently*

$$y_i = \begin{cases} x_i & \text{with probability } \rho \\ \text{uniformly random} & \text{with probability } 1-\rho \end{cases}$$

*If $\mathbf{x} \in \{-1,1\}^d$ is drawn uniformly at random and then $\mathbf{y} \sim N_\rho(\mathbf{x})$, we say that $(\mathbf{x}, \mathbf{y})$ is a ρ-**correlated pair**.*

Given a function $f : \{-1,1\}^d \mapsto \mathbb{R}$, the ***noise stability of*** f ***at*** ρ *is*

$$\textbf{\textit{Stab}}_\rho[f] \triangleq \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\ \rho\text{-}correlated}[f(\mathbf{x})f(\mathbf{y})] \tag{3.14}$$

Proposition 13 (Theorem 2.49 in O'Donnell [2014])**.**

$$\textbf{\textit{Stab}}_\rho[f] = \sum_{S\subseteq[d]} \rho^{|S|}[\hat{f}(S)]^2 \tag{3.15}$$

Intuitively, the noise stability looks at pairs of 'similar' inputs $(\mathbf{x}, \mathbf{y})$ and measure the similarity of $f(\mathbf{x})$ and $f(\mathbf{y})$. The intuition is probably most clear with Boolean-valued functions, thus for the sake of argument let us assume $f : \{-1,1\}^d \mapsto \{-1,1\}$ here. Equation (3.15) expresses the noise stability in terms of the Fourier coefficients, and clearly shows why high noise stability implies spectral concentration at low degrees: recall that $\sum_{S\subseteq[d]}[\hat{f}(S)]^2 = \mathbb{E}[f^2(\mathbf{x})] = 1$ (Parseval's Theorem) and $|\rho| < 1$, the Fourier coefficients are less affected by the 'shrinkage' term $\rho^{|S|}$ if $|S|$ is smaller.

Given an arbitrary input distribution $p(\mathbf{x})$, how we apply the chain rule defines a *sequential* data generation process: a node is generated after all its parents have been generated[3]. In contrast, under the special case of a product $p(\mathbf{x})$, all the variables can be viewed as generated simultaneously. It turns out that the generalized Fourier spectrum provides suitable machinery for discussing concepts such as noise stability under the sequential data generation.

We extend the definition of *noise stability* to arbitrary $p(\mathbf{x})$ by imitating Definition 7, but with a new notion of ρ-correlated pairs (i.e. 'similar' inputs) to accommodate the sequential data generation:

Definition 8. *Let $\rho \in [0,1)$. $p(\mathbf{x})$ is a given distribution. For fixed $\mathbf{x} \in \{-1,1\}^d$, we write $\mathbf{y} \sim N_\rho^p(\mathbf{x})$ to denote the random vector $\mathbf{y}$ drawn as follows:*

$k = d$

[3]Without loss of generality, we will assume $1, 2, \cdots, d$ is the topological ordering.

```
for $i \in \{1, 2, \cdots, d\}$ do
    With probability $1 - \rho$, $k = i - 1$, break;
end for
```
Set $\mathbf{y}_1^k = \mathbf{x}_1^k$, and randomly generate $\mathbf{y}_{k+1}^d$ from $p(\mathbf{y}_{k+1}^d | \mathbf{y}_1^k = \mathbf{x}_1^k)$

*If $\mathbf{x} \in \{-1, 1\}^d$ is drawn from $p(\mathbf{x})$ and $\mathbf{y} \sim N_\rho^p(\mathbf{x})$, we say that $(\mathbf{x}, \mathbf{y})$ is a **ρ-correlated pair under** $p(\mathbf{x})$.*

*Given a function $f : \{-1, 1\}^d \mapsto \mathbb{R}$, the **noise stability of f at ρ under** $p(\mathbf{x})$ is*

$$\textbf{\textit{Stab}}_\rho[f] \triangleq \mathop{\mathbb{E}}_{(\mathbf{x},\mathbf{y})\ \rho\text{-correlated under } p}[f(\mathbf{x})f(\mathbf{y})] \tag{3.16}$$

The following proposition relates the noise stability under arbitrary distribution $p(\mathbf{x})$ to the generalized Fourier spectrum defined in Theorem 1.

Proposition 14. *Let $\max(S)$ denote the largest element in set S (i.e. the element that comes last in the topological ordering). Then*

$$\textbf{\textit{Stab}}_\rho[f] = \sum_{S \subseteq [d]} [\hat{f}^p(S)]^2 \rho^{\max(S)} \tag{3.17}$$

Note that the data generation process described in Definition 8 has not considered the structure of $p(\mathbf{x})$ beyond a topological ordering. In practice we often work with highly structured distributions. If x_i has only a small number of parent variables in $p(\mathbf{x})$, one does not need to wait for all of x_1, x_2, $\cdots x_{i-1}$ to be generated before generating values for x_i. Only the parents' values are required. Thus we can refine the definition of noise stability to explicitly take into account the structure of $p(\mathbf{x})$, and as shown in Proposition 15, the connection to generalized spectrum still holds.

Definition 9. *Let $\rho \in [0, 1)$ and $p(\mathbf{x})$ is the given distribution. For fixed $\mathbf{x} \in \{-1, 1\}^d$, we write $\mathbf{y} \sim \tilde{N}_\rho^p(\mathbf{x})$ to denote the random vector $\mathbf{y}$ drawn as follows:*

```
Let $b_i$ be a flag for $y_i$ s.t. $b_i = 1$ indicates $y_i$ is randomly generated and $b_i = 0$ if $y_i$ is set to equal to $x_i$. Initialize $b_i = 0$ for $\forall i$.
for $i \in \{1, 2, \cdots, d\}$ do
    if $\exists j \in \pi_i$ s.t. $b_j = 1$ then
    Set $b_i = 1$
  else
    Set $b_i = 1$ with probability $1-\rho$ and $b_i = 0$ with probability $\rho$
  end if
  if $b_i = 1$ then
    Generate $y_i$ from $p(y_i|\mathbf{y}_{\pi_i} = \mathbf{y}_{\pi_i})$
  else
    Set $y_i = x_i$
  end if
end for
```

With some abuse of terminology, we say that $(\mathbf{x}, \mathbf{y})$ *is a* ***ρ-correlated pair under*** $p(\mathbf{x})$ *if* $\mathbf{x} \in \{-1,1\}^d$ *is drawn from* $p(\mathbf{x})$ *and* $\mathbf{y} \sim \tilde{N}^p_\rho(\mathbf{x})$*. Given a function* $f : \{-1,1\}^d \mapsto \mathbb{R}$*, the* ***noise stability of** f **at** ρ **under*** $p(\mathbf{x})$ *is*

$$\textbf{\textit{Stab}}_\rho[f] \triangleq \underset{(\mathbf{x},\mathbf{y})\ \rho\text{-correlated under } p}{\mathbb{E}}[f(\mathbf{x})f(\mathbf{y})] \tag{3.18}$$

Proposition 15. *Under Definition 9,*

$$\textbf{\textit{Stab}}_\rho[f] = \sum_{S\subseteq[d]} [\hat{f}^p(S)]^2 \rho^{\sigma(S)} \tag{3.19}$$

where $\sigma_S = |\cup_{i\in S} A_i|$ *and* A_i *denotes the set of all ancestors of* x_i *in* $p(\mathbf{x})$*.*

We defer the proof of Propositions 14 and 15 to Appendix A, but let us make a few comments before moving on:

1. $\sigma(S)$ is upper-bounded by $\max(S)$ (e.g. if $p(\mathbf{x}) = p(x_1)\prod_{i=2}^d p(x_i|x_{i-1})$) and lower-bounded by $|S|$ (e.g. if $p(\mathbf{x})$ is a product distribution).

2. In the case where $p(\mathbf{x})$ is indeed the uniform distribution, Definition 9 coincides with Definition 7 and Proposition 15 coincides with Proposition 13.

3. Recall that through the Low Degree algorithm, learnability of target functions (PAC model, polynomial time) is directly related to the spectral concentration

on the low cardinality subsets. In a similar manner, learnability of functions under the sequential data generation scenario can be directly related to the spectral concentration on the subsets with small values of σ_S.

3.3 Direct Generalization of Low Degree Algorithm

The definition of generalized Fourier spectrum naturally lead to an algorithm that is a direct generalization of the Low Degree algorithm described in Section 2.2.2. The detailed description is presented as Algorithm 1 and we also refer to it as *direct spectrum estimation* algorithm.

A few comments are in order:

1. We deliberately left the details of the subroutine **LearnInputDistribution** unspecified. In fact this is where assumptions need to be made and any available domain knowledge can be incorporated, for example, in the form of structural information or constraints on the distribution $p(\mathbf{x})$. When the structure of $p(\mathbf{x})$ is known, estimating the maximum-likelihood parameters for conditional distributions $q(x_i|\mathbf{x}_{\pi_i})$ reduces to counting.

2. When structural information of $p(\mathbf{x})$ is not available, we are faced with a graphical model structure learning problem, except that our goal is to produce accurate estimation of Fourier coefficients instead of the actual recovery of the structure. More rigorous analysis is carried out in Chapter 5 and simulation results shown in Chapter 6. Here we just comment on a high level that there is a trade-off between *using simple structures* which results in high bias in coefficient estimation, and *using complex structures* which results in high variance. The 'optimal' structure is not necessarily the true structure and in general is a function of the amount of available training data.

3. The computational cost of the algorithm grows as $O(N \cdot d^{d_0})$, and the treatment of the $O(d^{d_0})$ features can be fully parallelized. The exponential dependence on

d_0 of the cost may seem unpleasant, but for practical purposes, the applications where this thesis is targeting at usually only require a d_0 of 2 or 3. Chapter 6 contains more discussion on this point.

4. Instead of a pre-fixed threshold ϵ on the Fourier coefficient magnitude, we can implement a variant where the top K features (in terms of largest absolute value of the corresponding coefficient) are chosen. This gives a better control over the model size, but there will be a sorting step that cannot be easily parallelized.

Algorithm 1 Direct Spectrum Estimation

Input: training data $\mathcal{D} = \{\mathbf{x}^n, y^n\}_{n=1}^N$, test data $\mathcal{D}' = \{\mathbf{z}^m\}_{m=1}^M$, threshold ϵ, a collection of candidate subsets $\mathcal{C} \triangleq \{S \subseteq [d] : |S| \leq d_0\}$

Output: predicted output for each $\mathbf{z}^m$ in the test set

Steps:

1. Estimate input distribution from data:

$$\{\pi_i\}_{i=1}^d, \{q(x_i|\mathbf{x}_{\pi_i})\}_{i=1}^d = \mathbf{LearnInputDistribution}(\mathcal{D})$$

2. Let $\mathcal{C}' = \varnothing$. For each $S \in \mathcal{C}$:

$$\tilde{f}(S) = \frac{1}{N}\sum_{n=1}^{N} y^n \prod_{i\in S} x_i^n \sqrt{\frac{q(-x_i^n|\mathbf{x}_{\pi_i}^n)}{q(x_i^n|\mathbf{x}_{\pi_i}^n)}}$$

$$\text{If } |\tilde{f}(S)| > \epsilon, \mathcal{C}' = \mathcal{C}' \cup \{S\}$$

3. For each $\mathbf{z}^m \in \mathcal{D}'$, output

$$\sum_{S\in\mathcal{C}'} \tilde{f}(S) \prod_{i\in S} z_i^m \sqrt{\frac{q(-z_i^m|\mathbf{z}_{\pi_i}^m)}{q(z_i^m|\mathbf{z}_{\pi_i}^m)}} \tag{3.20}$$

as the prediction for regression task, and output the sign of Equation (3.20) for classification task.

Algorithm 1 is clearly a straightforward extension of the Low Degree algorithm to the generalized Fourier spectrum. As mentioned before, Low Degree algorithm is essentially a regression algorithm without proper regularization, with the assumption of uniform input distribution. Algorithm 1 relaxes the uniform $p(\mathbf{x})$ assumption but still doesn't tackle the problems of improper regularization and designed for regression tasks, therefore it should not be surprising that its performance is not competitive to popular machine learning algorithms[4]. However, Algorithm 1 is still interesting due to its amenability to analysis, which can be viewed as providing lower bounds for performance guarantees. Section 3.5 proposes two more variants of spectrum-based learning algorithms that do offer competitive performance. But before that, we provide an alternative interpretation of Algorithm 1 in Section 3.4.

3.4 Connection to Linear Smoothers

It turns out that the direct spectrum estimation algorithm described in Section 3.3 can be viewed as a member of the broad family of *linear smoothers.*

Given a test point $\mathbf{z}$, the algorithm estimates a label for $\mathbf{z}$ by

$$\sum_{S\in\mathcal{C}} \Big[\frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^n)\phi_S^p(\mathbf{x}^n)\Big]\phi_S^p(\mathbf{z}) = \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^n)\{\sum_{S\in\mathcal{C}} \phi_S^p(\mathbf{x}^n)\phi_S^p(\mathbf{z})\} \tag{3.21}$$

Let $w_{\mathcal{C}}(\mathbf{z},\mathbf{x}^n) \triangleq \sum_{S\in\mathcal{C}} \phi_S^p(\mathbf{x}^n)\phi_S^p(\mathbf{z})$, we have

$$\hat{y}(\mathbf{z}) = \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^n) w_{\mathcal{C}}(\mathbf{z},\mathbf{x}^n) \tag{3.22}$$

In other words, each predicted label is a weighted-average over the labels of training points, thus the direct spectrum estimation algorithm is by definition a linear smoother. The weights $w_{\mathcal{C}}(\mathbf{z},\mathbf{x}^n)$ measures the 'similarity' between data points $\mathbf{z}$ and $\mathbf{x}^n$, and will vary depending on the choice of $\mathcal{C}$. Note that unlike in some popular linear

[4]Results to be shown in Chapter 6

smoothing methods such as k-Nearest Neighbours and Kernel Smoothing, here the weight $w_{\mathcal{C}}(\mathbf{z}, \mathbf{x}^n)$ can take negative values.

In a predictive setting, *degree of freedom* is often used to compare linear smoothing methods with different parameterizations. Roughly speaking, degree of freedom of a linear smoother describes the effective number of parameters and thus provides a quantitative measure of estimator complexity. On the other hand, degree of freedom is closely connected with the gap between training and generalization errors. Below we provide a formal definition of degree of freedom and compute its value for the linear smoother in Equation (3.22). The treatment closely follows the lecture notes of Tibshirani [Spring 2014].

Definition 10. *Suppose we observe $y^n = r(\mathbf{x}^n) + \epsilon^n$, $n = 1, 2, \cdots, N$, where the errors ϵ^n's are uncorrelated with common variance $\sigma^2 > 0$. Treat $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N$ as fixed. Given an estimator $\hat{r}(\cdot)$ that fits $\hat{y}^n = \hat{r}(\mathbf{x}^n)$, $n = 1, 2, \cdots, N$, the degree of freedom of $\hat{r}(\cdot)$ is defined as:*

$$\mathrm{df}(\hat{r}) \triangleq \frac{1}{\sigma^2} \sum_{n=1}^{N} \mathrm{Cov}(\hat{y}^n, y^n) = \frac{1}{\sigma^2} \mathrm{tr}(\mathrm{Cov}(\hat{\mathbf{y}}, \mathbf{y})) \tag{3.23}$$

where $\hat{\mathbf{y}} = (\hat{y}^1, \hat{y}^2, \cdots, \hat{y}^N)$ and $\mathbf{y} = (y^1, y^2, \cdots, y^N)$.

The uncorrelated errors with common variance are only required for the definition of degree of freedom. We do not assume the training data is generated this way.

Proposition 16. *Let $\hat{r}(y^m) = \frac{1}{N}\sum_{n=1}^{N} f(\mathbf{x}^n) w_{\mathcal{C}}(\mathbf{x}^m, \mathbf{x}^n)$, $\forall m \in \{1, 2, \cdots, N\}$. Then*

$$\mathrm{df}(\hat{r}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{S \in \mathcal{C}} \prod_{i \in S} \frac{p(-x_i^n | \mathbf{x}_{\pi_i}^n)}{p(x_i^n | \mathbf{x}_{\pi_i}^n)} \tag{3.24}$$

The proof of the above proposition can be found in Appendix A. Note that if $p(\mathbf{x})$ is uniform, the degree of freedom will be $|\mathcal{C}|$. For an arbitrary $p(\mathbf{x})$, the exact value of degree of freedom depends on the choice of training data, but its expected value is $|\mathcal{C}|$. For comparison, Linear Regression has a degree of freedom of d, and k-Nearest

Neighbours algorithm has a degree of freedom of N/k.

A well-known limitation of linear smoothers is their poor performance on functions with different level of smoothness in different regions of the input space, which is a direct result of the commonly used Lipschitz continuity assumption in the analysis[5]. However, this limitation does not show up in the Boolean function context, as Boolean functions are fundamentally much easier than functions defined on a continuous domain. For example, assume $f : \{-1, 1\} \mapsto \mathbb{R}$, $g : [-1, 1] \mapsto \mathcal{R}$. Given the function values at two points $f(-1)$, $f(1)$, $g(-1)$ and $g(1)$, $f(x)$ is now fully defined, while $g(x)$ still has infinitely many possibilities that satisfy the requirements even under Lipschitz assumptions.

The rest of this section attempts to offer some intuition about the weights $w_{\mathcal{C}}(\cdot, \cdot)$ under different choices of $p(\mathbf{x})$ and $\mathcal{C}$. Let us first consider a few extreme cases.

If $\mathcal{C} = \{S : S \subseteq [d]\}$, i.e. no smoothness/low degree assumptions are made, then

$$w_{\mathcal{C}}(\mathbf{x}, \mathbf{z}) = \frac{1}{p(\mathbf{z})} \mathbb{1}(\mathbf{x} = \mathbf{z}) \tag{3.25}$$

In other words, observing a particular training sample provides no useful information at all for predicting the output of any other data point. More rigorously, 'no assumption on $f(\cdot)$' can be thought of as $f(\cdot)$ is chosen uniformly randomly from the 2^{2^d} possible Boolean-valued Boolean functions. Let $\mathbf{x}$ and $\mathbf{z}$ be two independent random variables uniformly drawn from $\{-1, 1\}^d$. Then the value $f(\mathbf{x})$ and $f(\mathbf{z})$ are two random variables, and the conditional mutual information $I(f(\mathbf{x}); f(\mathbf{z}) | \mathbf{x} \neq \mathbf{z}) = 0$.

At the other extreme, if $\mathcal{C} = \{\varnothing\}$, we have $w_{\mathcal{C}}(\mathbf{x}, \mathbf{z}) \equiv 1$ for all pairs of $\mathbf{x}$ and $\mathbf{z}$. That is, we output the average of all training labels as our prediction for any test

[5] There are research works that attempt to overcome this problem by focusing on the class of functions with bounded total variation in the derivatives, instead of functions with bounded Lipschitz continuity in the derivatives.

data. This also makes sense because by choosing $\mathcal{C} = \{\varnothing\}$ the assumption is that $f(\cdot)$ is a constant function.

For more general choices of $\mathcal{C}$, it seems hard to directly deal with individual weights without making restricting assumptions on $p(\mathbf{x})$. However, for a fixed test point $\mathbf{z}$, we can naturally characterize the mean and variance of the weights over certain collection of $\mathbf{x}$ values by taking the sequential data generation perspective. In particular, Proposition 17 shows that one natural intuition can be made rigorous: *the more similar* $\mathbf{x}$ *and* $\mathbf{z}$ *are, the higher the weight* $w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})$ *is in expectation.*

Proposition 17. *Given* $p(\mathbf{x})$ *(assuming* $1, 2, \cdots, d$ *is a topological ordering), collection of sets* $\mathcal{C}$*, a test point* $\mathbf{z} \in \{-1, 1\}^d$ *and some* $k \in \{0, 1, \cdots, d\}$*, we generate* $\mathbf{x}$ *by (1) let* $\mathbf{x}_1^k = \mathbf{z}_1^k$ *and (2) randomly sample* $\mathbf{x}_{k+1}^d$ *from* $p(\mathbf{x}_{k+1}^d | \mathbf{x}_1^k = \mathbf{z}_1^k)$*. Then*

$$\mathbb{E}[w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})] = \sum_{S \in \mathcal{C}, S \subseteq [k]} \left[\phi_S^p(\mathbf{z})\right]^2 \tag{3.26}$$

$$\mathrm{Var}(w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})) = \sum_{\substack{S,T \subseteq [k] \\ \varnothing \neq R \subseteq [d] \setminus [k] \\ S \cup R \in \mathcal{C}, T \cup R \in \mathcal{C}}} \left[\phi_S^p(\mathbf{z})\right]^2 \left[\phi_T^p(\mathbf{z})\right]^2 \left[\phi_R^p(\mathbf{z})\right]^2 \tag{3.27}$$

Proof of Proposition 17 can be found in Appendix A. Let us make a few comments.

1. Note that we assume a fixed k for both Equations (3.26) and (3.27), which measures how similar $\mathbf{x}$ is to $\mathbf{z}$. In particular, $\mathbf{x}$ and $\mathbf{z}$ share the same first k coordinates, and the coordinates $k + 1$ to d of $\mathbf{x}$ are randomly generated, conditioned on the values of the first k coordinates. As a direct result, the subsets whose elements are all contained in $[k]$ and the subsets that contain element outside $[k]$ contribute to the weight differently

2. The more similar $\mathbf{x}$ and $\mathbf{z}$ are, the larger the weight $w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})$ should be. As k increases, the expectation in Equation (3.26) is taken over a smaller collection of $\mathbf{x}$'s that are more similar to the fixed $\mathbf{z}$. It is not hard to verify that the right hand side of Equation (3.26) is non-decreasing with increasing k.

3. The variance of the weight (Equation (3.27)), however, in general is not a monotonic function of k.

To get more intuition about Proposition 17, we examine a few concrete examples.

Example 3. *Let $\mathcal{C} = \{S : S \subseteq [d]\}$, i.e. no assumptions are made on the spectrum of the target function. For any $p(\mathbf{x})$,*

$$\mathbb{E}[w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})] = \sum_{S \subseteq [k]} \left[\phi_S^p(\mathbf{z})\right]^2 = \prod_{i=1}^{k} (1 + \left[\phi_{\{i\}}^p(\mathbf{z})\right]^2) = \frac{1}{p(\mathbf{z}_1^k)}$$

$$\begin{aligned}\mathrm{Var}(w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})) &= \Bigg(\sum_{\substack{R \subseteq [d] \setminus [k] \\ R \neq \varnothing}} \left[\phi_R^p(\mathbf{z})\right]^2 \Bigg) \Bigg(\sum_{S \subseteq [k]} \left[\phi_S^p(\mathbf{z})\right]^2 \Bigg)^2 \\ &= \Bigg(\prod_{i=k+1}^{d} \frac{1}{p(z_i | \mathbf{z}_{\pi_i})} - 1 \Bigg) \Bigg(\prod_{j=1}^{k} \frac{1}{p(z_j | \mathbf{z}_{\pi_j})} \Bigg)^2 = \frac{1 - p(\mathbf{z}_{k+1}^d | \mathbf{z}_1^k)}{p(\mathbf{z}) p(\mathbf{z}_1^k)}\end{aligned}$$

Example 4. *Let $\mathcal{C} = \{S : S \subseteq [d] \text{ and } |S| \leq 1\}$, i.e. we make the low degree assumption with $d_0 = 1$. For any $p(\mathbf{x})$,*

$$\mathbb{E}[w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})] = 1 + \sum_{i=1}^{k} \frac{p(-z_i | \mathbf{z}_{\pi_i})}{p(z_i | \mathbf{z}_{\pi_i})}$$

$$\mathrm{Var}(w_{\mathcal{C}}(\mathbf{x}, \mathbf{z})) = \sum_{\substack{\varnothing \neq R \subseteq [d] \setminus [k] \\ R \in \mathcal{C}}} \left[\phi_R^p(\mathbf{z})\right]^2 = \sum_{i=k+1}^{d} \left[\phi_R^p(\mathbf{z})\right]^2 = \sum_{i=k+1}^{d} \frac{p(-z_i | \mathbf{z}_{\pi_i})}{p(z_i | \mathbf{z}_{\pi_i})}$$

3.5 Spectrum-Based Learning via Feature Selection

Although Algorithm 1 has nice interpretation and is amenable to analysis, it does not offer competitive performance in practice. Recall that Section 2.2.2 pointed out three problems with Low Degree algorithm that prevent it from being practical. Algorithm 1 relaxes the assumption of uniform input distribution, but still suffer from lack of proper regularization[6] and designed specifically for regression tasks. In response to these problems, we propose to *use the Fourier spectrum information*

[6]Here we use 'regularization' in a very broad sense to refer to any step or process whose goal is not minimizing training error.

to perform feature selection and feed the selected features to appropriate downstream learning methods. The downstream learning method should take care of regularization, and when given a classification task, a classifier can be used.

Algorithm 2 and 3 describe two variants of spectrum-based learning where the spectral information is only used for feature selection. A few comments are in order.

Algorithm 2 Spectrum-Based Feature Selection

Input: training data $\mathcal{D} = \{\mathbf{x}^n, y^n\}_{n=1}^N$, test data $\mathcal{D} = \{\mathbf{z}^m\}_{m=1}^M$, threshold ϵ, a collection of candidate subsets $\mathcal{C} \triangleq \{S \subseteq [d] : |S| \leq d_0\}$, choice of downstream learning method $A(\cdot)$

Output: predicted output for each $\mathbf{z}^m$ in the test set

Steps:

1. Estimate input distribution from data:

$$\{\pi_i\}_{i=1}^d, \{q(x_i|\mathbf{x}_{\pi_i})\}_{i=1}^d = \textbf{LearnInputDistribution}(\mathcal{D})$$

2. Let $\mathcal{C}' = \varnothing$. For each $S \in \mathcal{C}$:

$$\tilde{f}(S) = \frac{1}{N}\sum_{n=1}^N y^n \prod_{i \in S} x_i^n \sqrt{\frac{q(-x_i^n|\mathbf{x}_{\pi_i}^n)}{q(x_i^n|\mathbf{x}_{\pi_i}^n)}}$$

If $|\tilde{f}(S)| > \epsilon, \mathcal{C}' = \mathcal{C}' \cup \{S\}$

3. Use $\{(\{\phi_S^q(\mathbf{x}^n)\}_{S\in\mathcal{C}'}, y^n)\}_{n=1}^N$ as training data to train the downstream learning algorithm $A(\cdot)$

4. For each $\mathbf{z}^m \in \mathcal{D}'$, the predicted output is $A(\{\phi_S^q(\mathbf{z}^m)\}_{S\in\mathcal{C}'})$

Firstly, the two variants perform feature selection exactly the same and will choose the same collection of subsets $\mathcal{C}'$. They only differ in what features they feed into the downstream learning algorithm: $\{\phi_S^p(\mathbf{x})\}_{S\in\mathcal{C}'}$ for Algorithm 2 and $\{\mathcal{X}_S(\mathbf{x})\}_{S\in\mathcal{C}'}$

Algorithm 3 Spectrum-Based Subset Selection

Input: training data $\mathcal{D} = \{\mathbf{x}^n, y^n\}_{n=1}^N$, test data $\mathcal{D} = \{\mathbf{z}^m\}_{m=1}^M$, threshold ϵ, a collection of candidate subsets $\mathcal{C} \triangleq \{S \subseteq [d] : |S| \leq d_0\}$, choice of downstream learning method $A(\cdot)$

Output: predicted output for each $\mathbf{z}^m$ in the test set

Steps:

```
1. Estimate input distribution from data:
       $\{\pi_i\}_{i=1}^d, \{q(x_i|\mathbf{x}_{\pi_i})\}_{i=1}^d = \textbf{LearnInputDistribution}(\mathcal{D})$
2. Let $\mathcal{C}' = \varnothing$. For each $S \in \mathcal{C}$:
       $\tilde{f}(S) = \frac{1}{N}\sum_{n=1}^N y^n \prod_{i \in S} x_i^n \sqrt{\frac{q(-x_i^n|\mathbf{x}_{\pi_i}^n)}{q(x_i^n|\mathbf{x}_{\pi_i}^n)}}$
       If $|\tilde{f}(S)| > \epsilon, \mathcal{C}' = \mathcal{C}' \cup \{S\}$
3. Use $\{(\{\mathcal{X}_S(\mathbf{x}^n)\}_{S\in\mathcal{C}'}, y^n)\}_{n=1}^N$ as training data to train the downstream learning
   algorithm $A(\cdot)$
4. For each $\mathbf{z}^m \in \mathcal{D}'$, the predicted output is $A(\{\mathcal{X}_S(\mathbf{z}^m)\}_{S\in\mathcal{C}'})$
```

for Algorithm 3. The difference is visually illustrated in Figure 3-1. If the Low Degree assumption is approximately satisfied under the choice of d_0, Algorithm 2 is clearly the 'correct' version, at least asymptotically. However, it is valuable to study Algorithm 3 as well. Consider a naive baseline where features $\{\mathcal{X}_S(\mathbf{x})\}_{S\subseteq[d],|S|\leq d_0}$ are passed through some general purpose feature selection methods. Algorithm 2 differ from this baseline in two major ways: (1) the features take different form; and (2) features are chosen according to a different criterion. Algorithm 3 in a sense provides a middle ground: it uses the same feature selection criterion as Algorithm 2 but its features take the same form as in the baseline. Thus we can separate the effect of those two factors by studying Algorithm 3.

In practice, Algorithm 3 brings further benefits. In terms of learning performance, Algorithm 3 actually provides more **robustness** than Algorithm 2 under limited training data. We discuss this point in more detail in Section 6.2.1.4. Algorithm 3 also

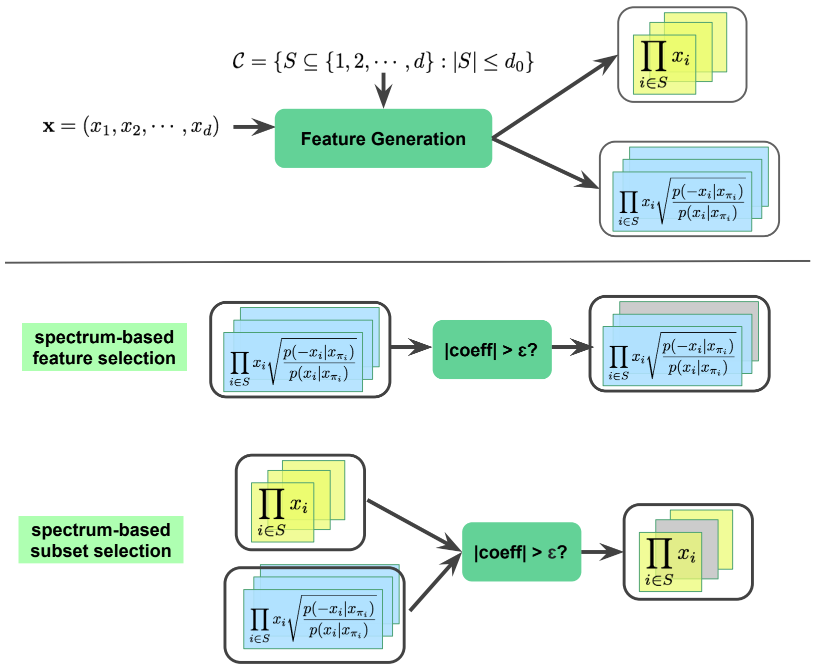

$\mathcal{C} = \{S \subseteq \{1, 2, \cdots, d\} : |S| \leq d_0\}$
$\mathbf{x} = (x_1, x_2, \cdots, x_d)$
Feature Generation
$\prod_{i \in S} x_i$
$\prod_{i \in S} x_i \sqrt{\frac{p(-x_i|x_{\pi_i})}{p(x_i|x_{\pi_i})}}$
spectrum-based feature selection
|coeff| > ε?
spectrum-based subset selection

Figure 3-1: Each $S \in \mathcal{C}$ can be viewed as corresponding to two features: $\mathcal{X}_S(\mathbf{x})$ (shown in yellow) and $\phi_S^p(\mathbf{x})$ (shown in blue). Both Algorithm 2 and 3 perform feature selection in the same way. But while Algorithm 2 feeds the blue features into downstream learning algorithm, Algorithm 3 will output the yellow features.

enjoys a better **interpretability**. As long as the raw features are human readable, so are the features Algorithm 3 outputs: they are simply the raw features themselves, or XOR of a few raw features.

In general, which variant of the algorithms should be used depends on the problem at hand: how many raw features are there, how much training data is available, what the generalized spectrum look like etc. In particular, we show in Chapter 6 that what downstream learning method is used will also play a role.

Secondly, Algorithms 2 and 3 are essentially proposing ways to generate non-linear combinations of the raw features and choose useful ones to augment the raw features. Thus there are two baselines we examine: (1) learning using the raw features directly vs learning using the new features; and (2) as mentioned above, the features generated by the proposed methods vs non-linear combinations $\{\mathcal{X}_S(\mathbf{x})\}_{S \subseteq [d], |S| \leq d_0}$ passed through a general purpose feature selection method.

One major advantage of the proposed methods is that due to the special structure of the generalized Fourier spectrum, one can legitimately make independent decisions for each of the features in parallel. This offers great reduction in computational cost. A brief overview of popular feature selection methods is included in Section 3.5.1 and simulation results are presented in Chapter 6.

Thirdly, so far we have argued that the benefit of having access to extra information by including higher order interaction terms leads to better prediction accuracy. However, even when $d_0 = 1$, the proposed methods can generate features that improve learning performance compared to the raw features, as they allow better feature selection. Such an example is provided in Section 6.3.2.1.

Fourthly, as for the downstream learning algorithm $A(\cdot)$, we put an emphasis on Linear Regression (for regression tasks) and Logistic Regression (for classification tasks) due to their simplicity and good interpretability. Moreover, it is not difficult to show that under mild conditions, both Linear Regression and Logistic Regression produce the Maximum Likelihood estimation of the Fourier coefficients. But the new features can certainly be fed into other learning algorithms and we show in Chapter 6 that the learning performance will generally be improved.

3.5.1 Brief Survey of Feature Selection Methods

This section provides a brief survey over feature selection methods, discusses where the proposed methods fit and lists baseline methods used in experiments.

Feature selection methods can broadly be divided into two categories: *filter methods* and *wrapper methods* (Chandrashekar and Sahin [2014]).

Filter methods deal with each feature individually (e.g. compute its variance, or mutual information with the output, or p-value), and rank them. Only the highly ranked features are selected. The filter methods do not involve learning as subroutine. Thus they are computationally light and very robust to overfitting. But they also have obvious drawbacks: there is no guarantee on the optimality of the chosen features, and higher order influences cannot be captured, i.e. informative but highly correlated features will be chosen together, while features that are not very informative on their own but are informative when combined with others will likely be discarded.

Wrapper methods require a learning method (treated as a blackbox) and use the prediction accuracy of the learning method to measure how good a subset of the features are. Evaluating all subsets of features is likely to be NP-hard, thus wrapper methods usually employ some searching algorithms to find a subset heuristically, e.g. start with an empty set and sequentially select the 'optimal' feature to add in. Wrapper methods generally do a better job taking into account higher order influence but are much more expensive computationally. The learning method in the blackbox needs to be trained every time a new subset of features is considered.

Chandrashekar and Sahin [2014] also discussed *embedded methods*, which tries to compensate for the drawbacks in filter methods and wrapper methods by incorporating feature selection as part of the training process. Examples include using an objective function to encourage larger relevancy between chosen features and output and smaller redundancy among chosen features, or training a model on all features and then use the 'weight' of each feature in the trained model to determine whether to keep it or to remove it.

The following representative feature selection methods are chosen as baselines to which we compare the proposed methods:

- **Individual p-value** (filter method): compute the p-value of each feature, and choose the ones with lowest p-values

- **Optimal Matching Pursuit** (wrapper method): start from an empty set, iteratively find the feature that best aligned with the residual of the signal[7]. After each step, the coefficients extracted so far are updated by computing the orthogonal projection of the signal onto the subspace spanned by the set of atoms selected so far.

- **Recursive Feature Elimination** (wrapper method): start with all the features, iteratively train a model on all the remaining features, identify the 'least relevant' feature according to the model and discard it.

- **LASSO** (embedded method): fit a linear model to the training data with L_1 regularization, which promotes sparsity and can be efficiently optimized, thus a popular feature selection method in practice.

As for the proposed methods, just like filter methods, they do not involve model training and are thus light in computation. Yet by design, we have taken into account the higher order influences and thus have optimality guarantees[8]. But note that the proposed methods require the candidate features to be generated according to the Fourier basis and cannot be applied to general feature selection problems.

Extra comment on p-value: Note that apart from the fact that it cannot deal with joint effect from several features, p-value based feature selection is fundamentally looking at a different problem. P-value of a feature measures "how easily we can detect its corresponding coefficient is not zero", while the actual question we care about is "whether the coefficient is large (in absolute value)". Asymptotically, if the variance of the coefficient estimation is 0, those two questions coincide. But with finite data, they are different and thus the p-valued based feature selection method is 'incorrect'.

[7]To use signal processing terminology, the vector of training labels $\mathbf{y} = (y^{(1)}, y^{(2)}, \cdots, y^{(N)})^T$ is the signal, and each feature $(\mathcal{X}_S(\mathbf{x}^{(1)}), \mathcal{X}_S(\mathbf{x}^{(2)}), \cdots, \mathcal{X}_S(\mathbf{x}^{(N)}))^T$ is an atom in the dictionary.

[8]Of course in practice we often need to estimate $p(\mathbf{x})$ from data and more complications arise. But that is a different story.

Chapter 4

Data Dependent Function Complexity

Chapter 5 will analyse how well the spectrum of a Boolean function can be estimated from training data under a given input distribution $p(\mathbf{x})$. The current chapter deviates from the practical learning problem and takes a more theoretical perspective, focusing on *understanding the spectrum itself and how it is a result of the interplay between* $p(\mathbf{x})$ *and* $f(\mathbf{x})$. In particular, the same $f(\mathbf{x})$ will have different spectra under different $p(\mathbf{x})$, how are they related? Are there some input distributions under which the spectrum is 'simpler'? What exactly do we mean by 'simple'?

We begin with a discussion on various notions of function complexity in Section 4.1, emphasizing "spectrally simple functions", which provides a *data-dependent, algorithm-independent* measure of function complexity. Section 4.2 shows how the spectra of the same function under different input distributions are related. Section 4.3 validates the intuition that *uniform distribution is the most difficult to learn from* by rigorous statements as well as intuitive examples.

4.1 Notions of Function Complexity

In total, there are 2^{2^d} Boolean-valued Boolean functions with d input bits. Yet in practice, one only cares about a small subset of them that are highly structured and 'simple'[1]. From a slightly different point of view, if the goal is to explain the N training samples, where $N \ll 2^d$, one can always find a 'simple' explanation (i.e. a 'simple' function that fits all N training points well).

The arguments above are quite vague: what does "simple functions" mean exactly and how do we measure complexity of Boolean functions? We first review some commonly used notions of Boolean function complexity in Section 4.1.1 and then emphasize what we mean by spectrally simple functions in Section 4.1.2.

4.1.1 Commonly Used Complexity Notions

4.1.1.1 Kolmogorov Complexity

Fix an ordering to iterate over $\{-1,1\}^d$ and assume the bijective mapping from $\{-1,1\}$ to $\{0,1\}$ ($-1 \to 1$, $1 \to 0$), a Boolean-valued Boolean function $f(\mathbf{x})$ can be viewed as a binary string of length 2^d, and we can talk about its **Kolmogorov complexity**. This is probably the most fundamental measure of all.

Definition 11 (Cover and Thomas [2006][2], Chapter 14)**.** *The Kolmogorov complexity of a binary string* $\mathbf{x}$ *w.r.t universal computer* $\mathcal{U}$ *is the minimum length over all programs that print* $\mathbf{s}$ *and halts:* $K_{\mathcal{U}}(\mathbf{s}) = \min_{p:\mathcal{U}(p)=\mathbf{s}} l(p)$*. We also define conditional Kolmogorov complexity as* $K_{\mathcal{U}}(\mathbf{s}|l(\mathbf{s})) = \min_{p:\mathcal{U}(p,l(\mathbf{s}))=\mathbf{s}} l(p)$*, i.e. the universal computer has access to the length of* $\mathbf{s}$*.*

We cannot hope to efficiently learn a function with large Kolmogorov complexity. To see this point, note that if there is a polynomial-time algorithm that learns $f(\mathbf{x})$, then we have a polynomial size program that can output the function: simply describe the learning algorithm and the training samples.

[1]A similar argument holds for real-valued Boolean functions: only a vanishingly small fraction of all such functions is of practical interests.

[2]This definition and the Proposition below are directly taken from the Cover and Thomas textbook. The original ideas of Kolmogorov complexity are developed almost simultaneously by Kolmogorov, Solomonoff and Chaitin. Please refer to the Historical Notes section of Chapter 14 in Cover and Thomas [2006] for more bibliographical details.

The following proposition states that most strings have a Kolmogorov complexity close to their actual length, which in turn means that most Boolean functions cannot be learned exactly in polynomial time (w.r.t. the number of features d).

Proposition 18 (Cover and Thomas [2006], Chapter 14)**.** *If we uniformly randomly pick $f : \{-1,1\}^d \mapsto \{-1,1\}$ and let $\mathbf{s}$ be the corresponding string, then*

$$\mathbb{P}(K(\mathbf{s}|l(\mathbf{s}) = 2^d) < 2^d - k) < 2^{-k}, \quad \forall k \in \mathbb{N} \tag{4.1}$$

The definition of Kolmogorov complexity is very general and does not depend on any particular choice of learning algorithm or structure pattern. But as a result, it usually cannot be directly computed or estimated.

4.1.1.2 Circuit Size and Circuit Depth

One of the most popular complexity measures for Boolean function is the **circuit complexity**, i.e. the total number of gates (circuit size) or the length of longest path (circuit depth) in the smallest circuit that computes the Boolean function. The study of circuit complexity is an important field in theoretical computer science.

Depending on the choice of basis (i.e. what gates can be used), fan-in, fan-out etc., circuit complexity is actually a family of measures. The importance of circuit complexity is derived from its direct connections to the space and time required to perform the computation on a Turing machine.

It is proved that most Boolean functions on d variables require circuits of size $\Theta(2^d/d)$ and depth of $O(d)$, although this result says nothing about the circuit complexity of any specific function. Upper bound on circuit complexity is often provided by construction, while good lower bound is generally very hard to derive. Circuit complexity is mostly relevant in proofs and rarely used in a practical setting, since it is challenging to compute, let alone optimize for.

4.1.1.3 Size of Normal Forms

Another family of complexity measures for Boolean functions that is closely related to circuit complexity is the **size of the normal forms**. Common normal forms include:

- **Disjunctive Normal Form (DNF)**: sum (OR) of product (AND) terms, e.g. $(x_1 \wedge x_2 \wedge \bar{x_3}) \vee (\bar{x_2} \wedge x_4 \wedge \bar{x_5}) \vee (x_2 \wedge \bar{x_4})$

- **Conjunctive Normal Form (CNF)**: product (AND) of summation (OR) terms, e.g. $(x_1 \vee x_2 \vee \bar{x_3}) \wedge (\bar{x_2} \vee x_4 \vee \bar{x_5}) \wedge (x_2 \vee \bar{x_4})$

- **Ring Sum Expansion (RSE)**: XOR of a constant and products of un-negated variables, e.g. $1 \oplus x_1x_2 \oplus x_2x_3$

Complexity can then be measured by *number of terms* in the normal form. The different normal forms use different "building blocks" and thus do not generally agree on which functions are "simple". For example, the OR function over d variables has linear size in DNF representation but exponential size in RSE, while parity functions has linear size in RSE but exponential size in DNF or CNF.

Normal forms provide standard ways of constructing circuits from truth tables. A small normal form size does imply a small circuit (with appropriate assumptions on basis, fan-in, fan-out etc), but not vice versa. For example, $f(\mathbf{x}) = \mathbb{1}(\sum_{i=1}^{d} x_i \equiv 0 (\text{mod } 3))$ can be computed by a linear-size circuit, but has exponential size in CNF, DNF, and RSE representations.

4.1.1.4 Complexity Class

A Boolean-valued Boolean function can be viewed as a decision problem and therefore we can talk about their complexity classes. For example, P vs NP or DTIME vs NTIME in terms of time constraint, or PSPACE vs NPSPACE in terms of space constraint. Of course, in a learning setting, we implicitly assume the target function to be in P and PSPACE.

Note that the notions listed so far share a property: $f(\mathbf{x})$ *needs to be computed exactly for any* $\mathbf{x} \in \{-1, 1\}^d$ *and no error is allowed.* In other words, all inputs are equally important and these complexity measures purely depend on the target function itself. If a small fraction of error is allowed, which is almost always the case in a learning scenario in practice, some inputs become more important than others based on the choice of $p(\mathbf{x})$. Below are a few commonly used complexity notions that take $p(\mathbf{x})$ into account.

4.1.1.5 Model-Dependent Complexity

For most learning algorithms, there is some natural notion measuring the complexity of the model learned by the algorithm, such as the size of a Decision Tree, or the norm of the weight vector in Support Vector Machines, just to name a few. Obviously, different algorithms define different sets of 'simple' functions.

Such complexity measures are often easy to compute and can be readily incorporated into practical learning algorithms, e.g. as a regularization term in the objective function or as an extra step such as pruning. Since most practical learning methods attempt to minimize some form of empirical error (as an approximation of what actually matters, i.e. the generalization error), the input distribution is automatically taken care of. The cost to pay is that the model dependent complexity measures are mostly heuristic and usually hard to reason about beyond intuitive arguments.

4.1.1.6 Sample Complexity/Time Complexity

In the PAC learning model, *Sample Complexity* and *Time Complexity*, i.e. the amount of training samples or running time necessary to ensure the target function is learned by the algorithm up to a required accuracy with high probability, are both popular complexity measures. Note that the time complexity is lower-bounded by sample complexity, as each sample is touched at least once. As for sample complexity, there are two variants: the weak variant assumes a particular $p(\mathbf{x})$, while the strong variant considers the worst-case sample complexity over all possible $p(\mathbf{x})$.

For Boolean valued functions, the strong variant of the sample complexity is fully characterized by the *VC-dimension* of the class of target functions. In fact, Hanneke [2016] shows that a class of Boolean valued functions $\mathcal{H}$ is (ϵ, δ)-PAC learnable with a sample size $N = O\big(\frac{1}{\epsilon}[\mathrm{VC}(\mathcal{H}) + \ln\frac{1}{\delta}]\big)$ while Ehrenfeucht et al. [1989] proves that any (ϵ, δ)-PAC learning algorithm for $\mathcal{H}$ must have sample complexity $N = \Omega\big(\frac{1}{\epsilon}[\mathrm{VC}(\mathcal{H}) + \ln\frac{1}{\delta}]\big)$. To derive results for distribution-dependent sample complexity (i.e. the weak variant), one often rely on distribution-sensitive quantities such as *Rademacher complexity* instead of the VC-dimension.

Just as with model-dependent complexity, time and sample complexity are defined with respect to a certain learning algorithm. Note that while all previous complexity notions are defined for a single function, time and sample complexity are defined for a class of functions and measure the richness of the class, i.e. how difficult it is to identify the target function from the function class. Richness of a class depends on both the size of the class and the complexity of functions in the class.

4.1.2 Spectrally Simple Boolean Functions

Complexity Measure	Data Dependent?	Algorithm Dependent?	Efficiently Computable?
Kolmogorov Complexity	No	No	No
Circuit Size	No	No	No
Normal Form Size	No	No	Yes
Complexity Class	No	No	Depends
Model-Dependent Complexity	Yes	Yes	Yes
Time/Sample Complexity[3]	Yes	Yes	Depends

Table 4.1: Properties of Popular Complexity Notions for Boolean Functions

[3]Measures the complexity of function classes rather than single functions.

The complexity measures in Section 4.1.1 all make perfect sense, but are not equivalent. We summarize some of their properties in Table 4.1. A function may be considered simple under a certain notion but not under another. In fact, there is no single notion that can capture all forms of structure/regularity.

This section proposes two complexity measures in the context of spectrum-based learning of Boolean functions. They provide a nice middle ground between theory and application: these spectrum-based complexity measures can be efficiently estimated, but they are also closely related to quantities such as the circuit size or normal form size through Boolean function analysis. Moreover, it turns out that these measures are not tied to any specific learning algorithm, but can naturally take into account the input distribution, thus providing appropriate language for discussing how $p(\mathbf{x})$ affect the complexity of a function.

4.1.2.1 L1 Spectral Norm

A natural measure of spectral complexity is its **sparsity**, $\sum_{S\subseteq[d]} \mathbb{1}(\hat{f}^p(S) \neq 0)$. However, L_0 norm is often hard to work with and following common practice, we instead consider the L_1 relaxation, namely the **spectral L_1 norm**

$$\hat{\|}f\hat{\|}_1 \triangleq \sum_{S\subseteq[d]} |\hat{f}^p(S)| \quad \text{where } f(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}^p(S)\phi_S^p(\mathbf{x}) \tag{4.2}$$

Note that for a Boolean valued function $f(\mathbf{x})$, $\hat{\|}f\hat{\|}_1 \leq 2^{d/2}\sqrt{\sum_{S\subseteq[d]} \hat{f}^2(S)} = 2^{d/2}$, thus in general the spectral L_1 norm can be exponentially large in d. In fact, the Majority function $f(\mathbf{x}) = \text{sgn}(\sum_{i=1}^d x_i)$ has $\hat{\|}f\hat{\|}_1 \sim 2^{d/2+1}/\sqrt{\pi d}$. But the following proposition shows that given N training samples where $N \ll 2^d$, we can always find a function that explains all the training data, yet has a small spectral L_1 norm.

Proposition 19. *Given i.i.d. samples $\mathcal{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$ where $\mathbf{x}^n \in \{-1,1\}^d$ are generated from uniform distribution and $y^n \in \{-1,1\}$ chosen arbitrarily, one can always find some function $f : \{-1,1\}^d \mapsto \{-1,1\}$ such that $\forall n \in \{1, 2, \cdots, N\}$,*

$f(\mathbf{x}^n) = y^n$, and

$$\hat{\|}f\hat{\|}_1 \leq 1 + \sqrt{2N} \tag{4.3}$$

The proof for Proposition 19 can be found in Appendix B.

4.1.2.2 Weighted Cardinality

The other spectrum-based complexity measure we consider derives naturally from the *low degree assumption*, which, as discussed in Chapter 3, is essentially a smoothness assumption. A Boolean function with spectrum concentrated at lower degrees can be learned more efficiently by spectrum-based learning algorithm. In particular, we define the **weighted cardinality** to be

$$\mathrm{Card}(f, p) \triangleq \sum_{S \subseteq [d]} |S| [\hat{f}^p(S)]^2 \tag{4.4}$$

If $p(\mathsf{x})$ is uniform, the weighted cardinality coincides with the *total influence* $\mathrm{I}[f]$, an important concept in Boolean function analysis literature (see Definition 3). Existing results in Boolean function analysis states that functions computable by small size DNF or by small size constant-depth circuit have bounded total influence.

Proposition 20 (Theorem 4.20 in O'Donnell [2014]). *Let $f : \{-1, 1\}^d \mapsto \{-1, 1\}$ be computable by a DNF of size s. Then $\mathrm{I}[f] \leq O(\log s)$.*

Proposition 21 (Theorem 4.30 in O'Donnell [2014]). *Let $f : \{-1, 1\}^d \mapsto \{-1, 1\}$ be computable by a depth-k circuit of size s. Then $\mathrm{I}[f] \leq O([\log s]^{k-1})$.*

4.2 Spectrum under Different Distributions

A function $f : \{-1, 1\}^d \mapsto \mathbb{R}$ can be Fourier-decomposed according to any distribution $p(\mathsf{x})$ we choose. The following proposition shows how the spectra of the same function under different distributions are related.

Proposition 22. *Given $f : \{-1, 1\}^d \mapsto \mathbb{R}$ and $p(\mathsf{x})$, we decompose f under uniform distribution as $f(\mathsf{x}) = \sum_{S \subseteq [d]} \hat{f}(S) \mathcal{X}_S(\mathsf{x})$ and under $p(\mathsf{x})$ as $f(\mathsf{x}) = \sum_{S \subseteq [d]} \hat{f}^p(S) \phi^p_S(\mathsf{x})$.*

Let $\hat{\boldsymbol{f}}$ *and* $\hat{\boldsymbol{f}}^p$ *denote the vector of Fourier coefficients* $\{\hat{f}(S)\}_{S\subseteq[d]}$ *and* $\{\hat{f}^p(S)\}_{S\subseteq[d]}$ *respectively, using the same ordering when iterating over the subsets. Then*

$$\hat{\boldsymbol{f}}^p = M^p\hat{\boldsymbol{f}}, \quad \text{where } M^p \in \mathbb{R}^{2^d\times 2^d} \text{ and } M^p_{S,T} = \langle \phi^p_S, \mathcal{X}_T\rangle_p, \forall S,T\subseteq[d] \tag{4.5}$$

That is, the vector of Fourier coefficients under any $p(\mathbf{x})$ can be obtained from the vector of Fourier coefficients under uniform distribution using a linear transformation. The proof of Proposition 22 is fairly simple and can be found in Appendix B.

Of course, Proposition 22 is nothing more than a change of basis. The rest of this section attempts to provide some intuitive understanding about the linear mapping M^p. In some special cases M^p is well-understood. For instance, if $p(\mathbf{x})$ is a product distribution, we know all the eigenvalues and corresponding eigenvectors of M^p. In general, however, M^p can be quite complicated.

For the purpose of spectrum-based learning of $f(\mathbf{x})$, there is one particular question about M^p we are interested in: if the spectrum of $f(\mathbf{x})$ is concentrated on terms with cardinality up to d_0 under uniform distribution, how about its spectrum under some non-uniform $p(\mathbf{x})$? Low degree concentration is necessary to apply the proposed methods. Of course we can simply take the assumption on faith that any combination of $p(\mathbf{x})$ and $f(\mathbf{x})$ encountered in real applications approximately satisfies the Low Degree assumption. But this seems a bit arbitrary. Fortunately, it turns out that *a function satisfying the low degree assumption under uniform distribution is also low degree under 'reasonable'* $p(\mathbf{x})$, justifying the Low Degree assumption with the same d_0 under non-uniform $p(\mathbf{x})$.

Below we prove for two families of distributions that the low-degree-ness is carried over exactly. It should become clear in the discussion that the intuition applies to more general choices of $p(\mathbf{x})$. A small simplification is made: technically the Low Degree assumption is defined as $\sum_{S\subseteq[d],|S|>d_0}\hat{f}^2(S)\le\epsilon$, and we assume $\epsilon=0$ here to simplify presentation. All results can be carried over to the case where ϵ is non-zero but very small.

Product Distributions It is quite straightforward to prove that the low-degree property of a function is retained under *product* $p(\mathbf{x})$. We begin with the following proposition, which relates the Fourier coefficients under $p(\mathbf{x})$ to the coefficients under uniform distribution:

Proposition 23. *Let $p(\mathbf{x})$ be a product distribution, and let p_i denote $\mathbb{P}(x_i = -1)$. Then for any $S \subseteq [d]$, we have*

$$\hat{f}^p(S) = \Big[\prod_{i \in S} 2\sqrt{p_i(1-p_i)}\Big]\Big[\sum_{T: S \subseteq T} \hat{f}(T) \prod_{i \in T \setminus S} (1-2p_i)\Big] \tag{4.6}$$

The proof of Proposition 23 is included in Appendix B. Consider a function $f(\mathbf{x})$ that satisfies Low Degree assumption of degree d_0 under uniform distribution, i.e. $\hat{f}(T) = 0$ for all T with $|T| > d_0$, then Proposition 23 states that $\hat{f}^p(S) = 0$ for any $|S| > d_0$ because there is no T s.t. $S \subseteq T$ and $\hat{f}(T) \neq 0$. In other words, $f(\mathbf{x})$ *satisfies the low degree assumption with the same d_0 under product $p(\mathbf{x})$.*

Another interesting observation from Proposition 23 is that for S s.t. $|S| = d_0$, $\hat{f}^p(S) = \prod_{i \in S} 2\sqrt{p_i(1-p_i)}\hat{f}(S)$. Since $2\sqrt{p_i(1-p_i)} \leq 1$, the weights on these 'boundary sets' shrink under a non-uniform product distribution.

Tree/Forest with Symmetric Edges It turns out that the low-degree property is retained for a much broader class of distributions than the product ones. We say $p(\mathbf{x})$ is a tree/forest with symmetric edges if:

1. Each node x_i has at most 1 parent ('tree/forest')
2. $\forall i \in [d]$, if x_i has a parent, then $\mathbb{P}(x_i = -1|x_{\pi_i} = 1) = \mathbb{P}(x_i = 1|x_{\pi_i} = -1)$ ('symmetric edges')

Obviously product distribution is a special case of this family. It should be commented that both assumptions here are made in order to obtain a clean result, but not fundamental intuitively. The following proposition is the main result used to prove conservation of low-degree-ness:

Proposition 24. *Let $p(\mathbf{x})$ be a tree/forest with symmetric edges. Then $\forall S, T \subseteq [d]$ s.t. $|S| > |T|$, we have*

$$M^p_{S,T} = 0 \tag{4.7}$$

Note that Proposition 24 implies that, $\forall f : \{-1,1\}^d \mapsto \mathbb{R}$ that satisfies the Low Degree assumption up to degree d_0 under uniform distribution also satisfies the Low Degree assumption under $p(\mathbf{x})$: $\forall S \subseteq [d]$, s.t. $|S| > d_0$

$$\hat{f}^p(S) = \sum_{T \subseteq [d]} \hat{f}(T) M^p_{S,T} = \sum_{T : |T| \leq d_0} \hat{f}(T) M^p_{S,T} = 0$$

The second equality is based on the Low Degree assumption under uniform distribution and the last step comes from Proposition 24.

We show the proof of Proposition 24 inline because the intuition in the proof holds for more general choices of $p(\mathbf{x})$.

Proof of Proposition 24.

$$\begin{aligned} M^p_{S,T} &= \sum_{\mathbf{x}} p(\mathbf{x}) \phi^p_S(\mathbf{x}) \mathcal{X}_T(\mathbf{x}) = \sum_{\mathbf{x}} \prod_{i=1}^{d} p(x_i | x_{\pi_i}) \prod_{j \in S} x_j \sqrt{\frac{p(-x_j | x_{\pi_j})}{p(x_j | x_{\pi_j})}} \prod_{k \in T} x_k \\ &= \sum_{x_1} h_1(x_1) \sum_{x_2} h_2(\mathbf{x}_1^2) \cdots \sum_{x_d} h_d(\mathbf{x}_1^d) \end{aligned} \tag{4.8}$$

$$\text{where } h_i(\mathbf{x}_1^i) \triangleq p(x_i | x_{\pi_i}) \left[x_i \sqrt{\frac{p(-x_i | x_{\pi_i})}{p(x_i | x_{\pi_i})}} \right]^{\mathbb{1}(i \in S)} \left[x_i \right]^{\mathbb{1}(i \in T)}, \quad \forall i \in [d].$$

Assume $1, 2, \cdots, d$ is a topological ordering. Then the computation in Equation (4.8) can be viewed as performing message passing from leaf to root. Each leaf node x_i falls into one of the following four cases:

- if $i \notin S \cup T$, $\sum_{x_i} h_i(\mathbf{x}_1^i) = \sum_{x_i} p(x_i | x_{\pi_i}) = 1$. The node x_i can be safely removed from the directed graph and a message of 1 is sent to its parent node.

- if $i \in S \setminus T$, $\sum_{x_i} h_i(\mathbf{x}_1^i) = \sum_{x_i} x_i \sqrt{p(x_i|x_{\pi_i})p(-x_i|x_{\pi_i})} = 0$. Thus $M_{S,T}^p = 0$ and message passing can stop here. We can equivalently think a message of 0 is sent to the parent node.

- if $i \in S \cap T$, $\sum_{x_i} h_i(\mathbf{x}_1^i) = \sum_{x_i} \sqrt{p(x_i|x_{\pi_i})p(-x_i|x_{\pi_i})} = 2\sqrt{p_i(1-p_i)} \triangleq \Delta_i$. This can be viewed as taking out a factor Δ_i and then remove the node x_i from the directed graph, which in turn can be seen as an element in T cancelling out an element in S. Note here $\Delta_i \in [0,1]$ and we have used the assumptions of symmetric edges and single parent.

- if $i \in T \setminus S$, $\sum_{x_i} h_i(\mathbf{x}_1^i) = \sum_{x_i} p(x_i|x_{\pi_i})x_i = (1-2p_i)x_{\pi_i}$. This can be viewed as taking out a factor $1-2p_i$ and then sending a message of x_{π_i} to the parent node x_{π_i}. Given the form of $h_i(\mathbf{x}_1^i)$, the message x_{π_i} can be viewed as flipping the node x_{π_i}'s membership in set T. Also note $1-2p_i \in [-1,1]$ and we again used the assumptions on $p(\mathbf{x})$.

If a node has multiple children, the messages from each child are multiplied together. In particular, if any of the messages is 0, the total message will be 0. Any message of 1 essentially has no effect. Any two messages of x_{π_i} will cancel each other.

The message passing algorithm will stop if 0 is encountered or if all remaining nodes are neither in S nor in T. Notice how *each element in T can only cancel at most one element in S*, thus if $|S| > |T|$, (at some point in the message passing procedure) we will always have a leaf node in S but not in T, which leads to $M_{S,T}^p = 0$. □

Let us make some more comments on the factors Δ_i and $1-2p_i$.

(1) Δ_i appears if and only if a 'cancel out' happens, and its magnitude is maximized (to 1) if $p_i = 0.5$, which actually corresponds to the absence of the edge. The stronger an edge is (i.e. p_i closer to 0 or 1), the smaller Δ_i will be and thus more attenuation is introduced.

(2) $1-2p_i$ appears if and only if a node in T 'moves' towards the root without meeting a node from S. The magnitude of $1-2p_i$ is also upper-bounded by 1

but in contrast to Δ_i, stronger edges result in larger magnitudes and a weaker edge will attenuate the influence more.

In summary, if many 'cancel out' happens and/or nodes in T need to 'travel' long distances to root or to meet an element in S, we expect the corresponding $M^p_{S,T}$ to be close to 0 in magnitude.

For an arbitrary $p(\mathbf{x})$, it is generally not true that if $\hat{f}^p(S) = 0$ for all $|S| > d_0$ given $\hat{f}(T) = 0$ for all $|T| > d_0$. However, much of the intuition from the proof of Proposition 24 still holds. For example, any leaf node not in $S \cup T$ can be directly removed from the graph and any leaf node is in $S \setminus T$ results in $M^p_{S,T} = 0$. The latter is likely to happen regardless of the structure of $p(\mathbf{x})$ given a small set T and a larger set S. The 'attenuation' will also happen if long distances need to be 'travelled' in the message passing, but if the edges are not symmetric, the messages will depend on the structure of $p(\mathbf{x})$ and are no longer as simple as $1 - 2p_i$. Yet it is not hard to show that the magnitudes of the messages are still upper-bounded by 1.

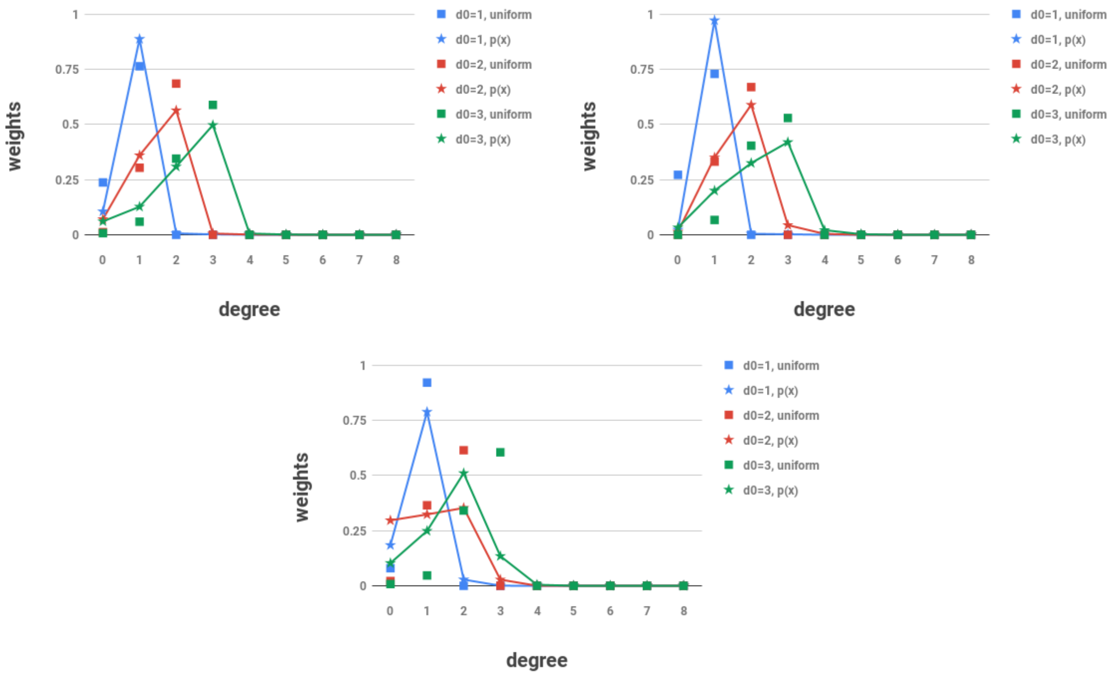


Figure 4-1: The spectrum of a low-degree function under uniform distribution roughly satisfies the low degree assumption under general $p(\mathbf{x})$ as well: a few typical examples.

Simulation results shown in Figure 4-1 support the above arguments. In particular, we plot the weight at degree k, defined as $W^k \triangleq \sum_{S:|S|=k} \left[\hat{f}^p(S)\right]^2$ for different choices of k, when the function $f(\mathbf{x})$ is generated such that it satisfies the Low Degree assumption up to d_0 under uniform distribution. Note that in the plots, the weights W^k for $k > d_0$ are not exactly zero, but quite negligible.

Each of the plots in Figure 4-1 correspond to one particular choice of $p(\mathbf{x})$. $p(\mathbf{x})$ is generated as follows: for each coordinate i (following the topological ordering), randomly generate the number of its parents using a Poisson distribution and then randomly choose the parents; then generate the corresponding parameters i.i.d from a Beta distribution. Note that randomly picking parent nodes for each node in turn in the topological ordering is perfectly general, but the exact way in which we determine the parent nodes and the corresponding parameters is somewhat arbitrary. However, we played with other possibilities of how to generate $p(\mathbf{x})$ and the results are consistent with what we present here.

Function $f(\mathbf{x})$ is generated by randomly picking values for $\hat{f}(S)$, $\forall S$ s.t. $|S| \le d_0$. The blue, red and green curves correspond to $d_0 = 1$, 2 and 3 respectively. Each curve is the result of averaging over 1000 independent choices of $f(\mathbf{x})$. The weights are normalized s.t. $\sum_{k=0}^{d} W^k = 1$. We also included the average of W^k for the same functions under uniform distribution (shown in squares) for reference: they share the same colour codes as the curves and are also normalized.

The plots in Figure 4-1 are some typical examples for $d = 8$. We tried varying different hyperparameters in how we generate $p(\mathbf{x})$ and $f(\mathbf{x})$ and similar trends can be observed, i.e. if $f(\mathbf{x})$ satisfies Low Degree assumption under uniform distribution, and the directed graph of $p(\mathbf{x})$ is relatively sparse, the Fourier spectrum of $f(\mathbf{x})$ under $p(\mathbf{x})$ is likely to be concentrated on terms up to d_0 as well.

4.3 Uniform Distribution is the Most Difficult

Fix a Boolean function $f : \{-1,1\}^d \mapsto \mathbb{R}$, its spectrum under some choices of $p(\mathbf{x})$ can be simpler than under others in terms of the *spectral* L_1 *norm* and/or *weighted cardinality*. Intuitively, we expect uniform distribution to be the most difficult to learn from, as non-uniform distributions can be viewed as offering information on what aspects of the input space to focus on: making an error on inputs with very small probabilities under $p(\mathbf{x})$ is not as costly as on inputs with high probabilities. This intuition also justifies the fact that in learning theory, learnability results are usually proved assuming uniform distribution, i.e. the worst case scenario is considered.

Ideally, we want to prove that for any Boolean function, its spectral L_1 norm and weighted cardinality are both maximized at uniform distribution. Unfortunately this argument is not true and counter examples are not difficult to construct. However, we can show that the argument holds in an average sense, i.e. if the function is randomly chosen from some ensemble, the expected spectral L_1 norm and weighted cardinality are both maximized when $p(\mathbf{x})$ is uniform.

In particular, consider $f(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}(S)\mathcal{X}_S(\mathbf{x})$, where each coefficient $\hat{f}(S)$ is independently generated from $\mathcal{N}(0,1)$, $S \subseteq [d]$. Note that this defines an ensemble of functions as well as a distribution over the functions in the ensemble. Denote the Fourier decomposition of such an $f(\mathbf{x})$ under some $p(\mathbf{x})$ to be $f(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}^p(S)\phi^p_S(\mathbf{x})$. Then we have the following two propositions.

Proposition 25. *Under the setup described above,*

$$\mathbb{E}\big[\sum_{S\subseteq[d]} |\hat{f}(S)| - \sum_{S\subseteq[d]} |\hat{f}^p(S)|\big] = 2^d\sqrt{\frac{2}{\pi}}\left[1-\frac{1}{2^d}\sum_{S\subseteq[d]}\sqrt{2^d\sum_{\mathbf{x}} p^2(\mathbf{x})[\phi^p_S(\mathbf{x})]^2}\right] \geq 0 \quad (4.9)$$

Proposition 26. *With the same setup as in Proposition 25, we have*

$$\mathbb{E}\Big[\sum_{S\subseteq[d]} |S|\hat{f}^2(S) - \sum_{S\subseteq[d]} |S|[\hat{f}^p(S)]^2\Big] = 2^{d-1}\sum_{i=1}^{d}\mathbb{E}_{\mathbf{x}_{\pi_i}}\Big[1 - 4p(\mathsf{x}_i|\mathbf{x}_{\pi_i})p(-\mathsf{x}_i|\mathbf{x}_{\pi_i})\Big] \geq 0 \tag{4.10}$$

The proof of Propositions 25 and 26 are included in Appendix B. We just want to comment here that both proofs make no assumption on $p(\mathbf{x})$ but do rely on the definition of this particular ensemble and cannot be easily generalized to other ensembles. This can be limiting. For instance, in the learning context, we might care about a 'low-degree ensemble', where only $\{\hat{f}(S)\}_{S:|S|\leq d_0}$ are generated i.i.d from $\mathcal{N}(0,1)$ and the other coefficients are zero.

The rest of this section attempts to argue that the intuition of 'uniform distribution is the most difficult to learn from' holds for more general ensembles. First, we relax the requirement on the ensemble but restrict our attention to specific choices of $p(\mathbf{x})$. In particular, we consider *local perturbations* from the uniform distribution and show how uniform is locally 'the worst' for various choices of ensembles. Second, we present some simulations results for low-degree ensembles under some families of $p(\mathbf{x})$.

4.3.1 Local Perturbation of Uniform Distribution

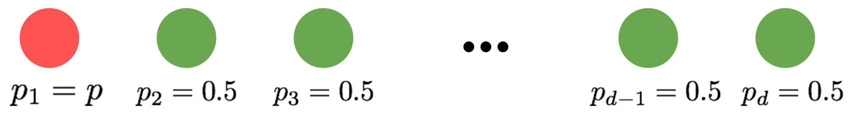


Figure 4-2: Local perturbation of uniform: vary the parameter of a single coordinate.

Consider modifying a single *parameter*, as shown in Figure 4-2. Without loss of generality, we assume that $p_1 \triangleq p(\mathsf{x}_1 = -1)$ is set to some p s.t. $p \neq 0.5$. It turns out the Fourier coefficients are affected in the following way:

$$\hat{f}^p(S) = \begin{cases} 2\sqrt{p(1-p)}\hat{f}(S) & \text{if } 1 \in S \\ \hat{f}(S) + (1-2p)\hat{f}(S \cup \{1\}) & \text{if } 1 \notin S \end{cases} \tag{4.11}$$

Note how the 2^d subsets can be partitioned into pairs and only subsets within the same pair can affect each other. The difference in spectral L1 norm or weighted cardinality can also be decomposed into sum of terms, each term corresponding to a pair. Let us start with spectral L1 norm:

$$\begin{aligned} &\sum_{S\subseteq[d]} |\hat{f}(S)| - \sum_{S\subseteq[d]} |\hat{f}^p(S)| \\ =& \sum_{S:1\notin S} \left[|\hat{f}(S\cup\{1\})|\big(1-2\sqrt{p(1-p)}\big) + |\hat{f}(S)| - |\hat{f}(S) + (1-2p)\hat{f}(S\cup\{1\})| \right] \\ \triangleq& \sum_{S:1\notin S} \left[|v_S|\big(1-2\sqrt{p(1-p)}\big) + |u_S| - |u_S + (1-2p)v_S| \right] \triangleq \sum_{S:1\notin S} \mathrm{Diff}(S) \end{aligned}$$

We have denoted $\hat{f}(S)$ as u_S and $\hat{f}(S \cup \{1\})$ as v_S, $\forall S \subseteq [d] \setminus \{1\}$. Exactly how u_S's and v_S's are generated is dictated by the definition of the ensemble.

We now have much more flexibility in choosing the ensemble. Let us examine a few examples. First consider a low-degree Gaussian ensemble, i.e. $\hat{f}(S) \sim \mathcal{N}(0,1)$ if $|S| \leq d_0$ and $\hat{f}(S) = 0$ otherwise. For $S \subseteq [d] \setminus \{1\}$, if $|S| = d_0$, $v_S = 0$ by definition and $\mathrm{Diff}(S) = |u_S| - |u_S| = 0$. If $|S| < d_0$, $u_S + (1-2p)v_S \sim \mathcal{N}(0, 1 + (1-2p)^2)$ and thus $\mathbb{E}[\mathrm{Diff}(S)] = \sqrt{\pi/2} \cdot [2 - 2\sqrt{p(1-p)} - \sqrt{1+(1-2p)^2}] \leq 0$.

As a second example, consider the ensemble where each $\hat{f}(S)$ with $|S| \leq d_0$ is independently generated from Uniform$[-1,1]$, and the other coefficients are set to 0. If $|S| = d_0$, $\mathrm{Diff}(S) = |u_S| - |u_S| = 0$ as before. If $|S| < d_0$,

$$\mathbb{E}[\mathrm{Diff}(S)] = \frac{1}{2}[1 - 2\sqrt{p(1-p)} + 1] - \int_{v=-1}^{1}\int_{u=-1}^{1} p(u,v)|u + (1-2p)v| \mathrm{d}u\mathrm{d}v$$

$$= 1 - \sqrt{p(1-p)} - \frac{1}{2} - \frac{1}{6}(2p-1)^2 = \frac{1}{3} - \sqrt{p(1-p)} + \frac{2}{3}p(1-p)$$
$$= \frac{1}{3}(1 - 2\sqrt{p(1-p)}(1 - \sqrt{p(1-p)}) \geq 0$$

Similar arguments can be made regarding the weighted degree.

$$\sum_{S\subseteq[d]} |S|\hat{f}^2(S) - \sum_{S\subseteq[d]} |S|\big[\hat{f}^p(S)\big]^2$$
$$= \sum_{S:1\notin S} (|S|+1)\hat{f}^2(S\cup\{1\})(1-4p(1-p)) + |S|\hat{f}^2(S) - |S|\big(\hat{f}(S) + (1-2p)\hat{f}(S\cup\{1\})\big)^2$$
$$= \sum_{S:1\notin S} (|S|+1)v_S^2(1-4p(1-p)) + |S|u_S^2 - |S|\big(u_S + (1-2p)v_S\big)^2$$
$$= v_S^2(1-2p)^2 - 2(1-2p)|S|v_S u_S \triangleq \mathrm{Diff}(S)$$

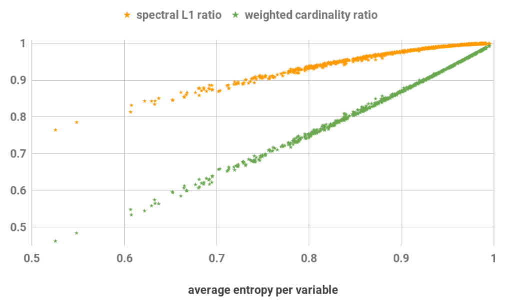
Here a sufficient condition for $\mathbb{E}[\mathrm{Diff}(S)] \geq 0$ is $u_S \perp\!\!\!\perp v_S$ and $\mathbb{E}[u_S] = \mathbb{E}[v_S] = 0$.

4.3.2 Simulation Results for Low Degree Ensembles

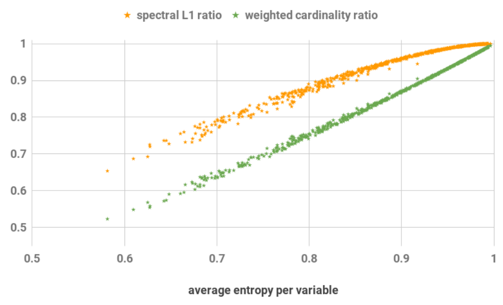
So far we have proved the intuition that "uniform distribution is the most difficult" in two setups: (1) no assumption on the distribution but assuming a specific Gaussian ensemble; and (2) more flexible choices of ensembles but only considering certain deviations from uniform distribution. Rigorous analysis that works for arbitrary distributions and ensembles seems hard, but we provide some simulations results below, which suggest that the validity of the intuition hold in more general settings.

Figure 4-3 shows how the spectral L_1 norm (shown in orange) and weighted cardinality (shown in green) under $p(\mathbf{x})$ compare to that under uniform distribution when $f(\mathbf{x})$ is chosen from the low degree Gaussian ensemble. The plots are generated assuming a product $p(\mathbf{x})$, d = 15 and d0 = 1, 2, or 3. The horizontal axis represents the average entropy per variable, measuring how 'non-uniform' the distribution is. The vertical axis shows the ratio of the expected L_1 norm (weighted cardinality) under $p(\mathbf{x})$ over the expected L_1 norm (weighted cardinality) under uniform distribution, which we expect to be less than 1. Each point in the plot corresponds to one $p(\mathbf{x})$

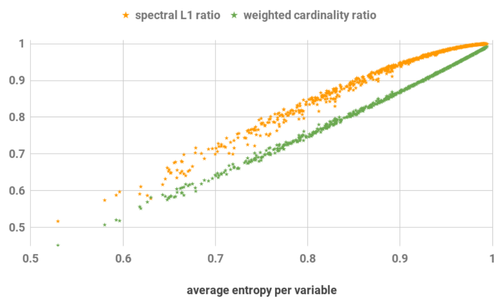
and the corresponding 'expected L_1 norm/weighted cardinality' are approximated by the sample mean from 50000 functions randomly generated from the ensemble.



(a) d=15, d0=1

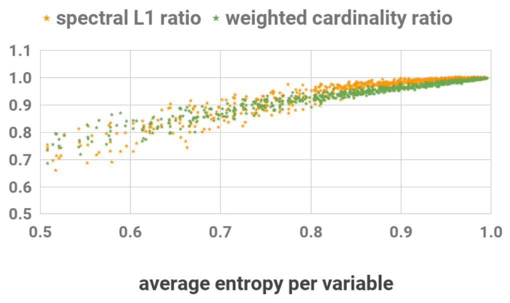


(b) d=15, d0=2

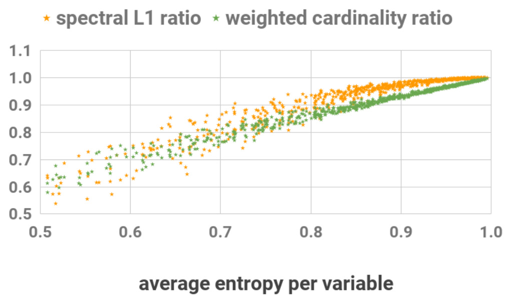


(c) d=15, d0=3

Figure 4-3: Expected value of spectral L_1 norm and weighted cardinality are both maximized at uniform distribution when $f(\mathbf{x})$ is generated from a low degree ensemble.



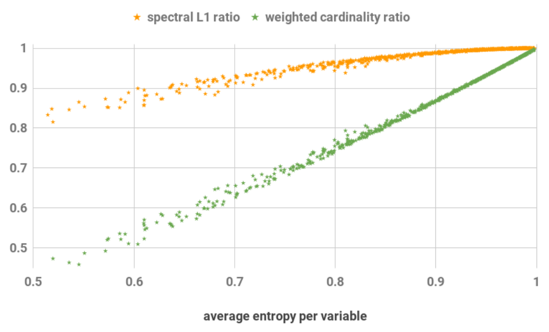
(a) d=8, d0=2



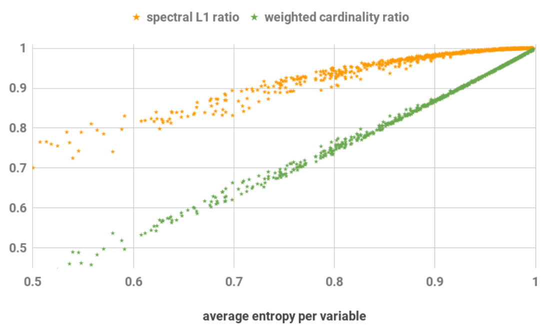
(b) d=8, d0=3

Figure 4-4: Ratio of the expected spectral L_1 norm and weighted cardinality, Markov Chain $p(\mathbf{x})$

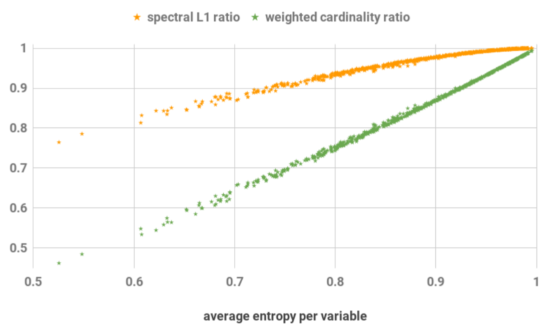
As can be seen from the plots, the ratios are always below 1, as expected. Also, there is a clear trend that the more 'non-uniform' $p(\mathbf{x})$ is (i.e. lower entropy), the spectrum of $f(\mathbf{x})$ is simpler both in terms of spectral L_1 norm and weighted cardinality, which makes intuitive sense. Similar trends can be observed when other families of $p(\mathbf{x})$ are considered (shown in Figure 4-4) and other choices of input dimension d is used (shown in Figure 4-5).



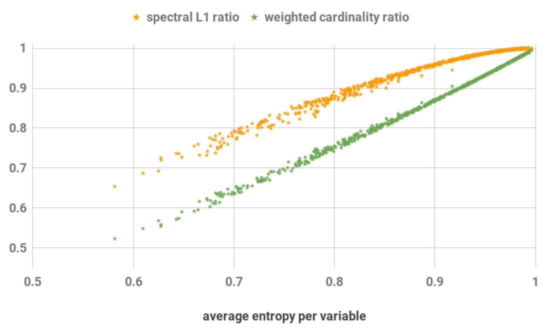
(a) d=5, d0=1



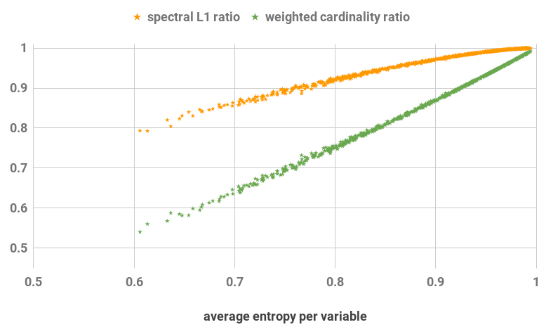
(b) d=5, d0=2



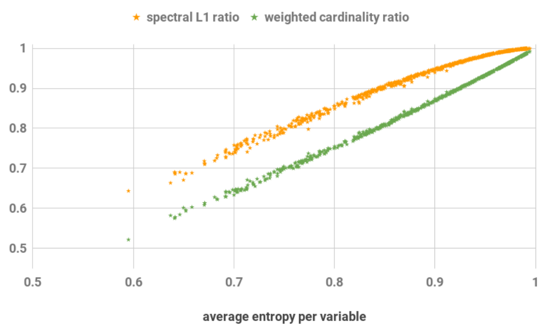
(c) d=15, d0=1



(d) d=15, d0=2



(e) d=25, d0=1



(f) d=25, d0=2

Figure 4-5: Ratio of expected spectral L_1 norm and weighted cardinality, varying d

Chapter 5

Analysis of the Spectrum-Based Learning Algorithms

Chapter 3 presents three spectrum-based algorithms for learning Boolean functions. *Direct Spectrum Estimation* (Algorithm 1) is a straightforward generalization of the famous Low Degree algorithm to arbitrary input distribution, while *Spectrum-Based Feature Selection* (Algorithm 2) and *Spectrum-Based Subset Selection* (Algorithm 3) use the spectral information to generate new features and subsequently carry out feature selection. Before examining their practical performance in experiments in Chapter 6, the current chapter aims to provide some theoretical guarantees.

Section 5.1 focuses on how well we can estimate a single Fourier coefficient under various assumptions of our prior knowledge of the input distribution. It is generally hard to analyse Algorithms 2 and 3 because their performances heavily rely on what downstream learning method is used. Section 5.2 analyses Algorithm 1 assuming known structure of the input distribution. Since Algorithm 1 is out-performed by the other two variants in practice, the analysis can be regarded as a lower bound.

5.1 Estimate a Single Fourier Coefficient

How well can we estimate a particular Fourier coefficient? We consider the case where the true $p(\mathbf{x})$ is known in Section 5.1.1 and the case where the structure of $p(\mathbf{x})$ is known but the parameters need to be estimated from data in Section 5.1.2.

5.1.1 Given Correct Input Distribution

Assume $p(\mathbf{x})$ is fully known. Then given training samples $\mathcal{D} = \{(\mathbf{x}^n, y^n)\}_{n=1}^N$ where $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N \overset{i.i.d}{\sim} p(\mathbf{x})$ and $y^n = f(\mathbf{x}^n)$, $\forall n$. For any $S \subseteq [d]$, the Fourier coefficient $\hat{f}^p(S)$ can be approximated by sample mean $\sum_{n=1}^N f(\mathbf{x}^n)\phi_S^p(\mathbf{x}^n)/N$. Note that if $v_n \triangleq f(\mathbf{x}^n)\phi_S^p(\mathbf{x}^n)$, $v_1, v_2, \cdots, v_N$ are also i.i.d, thus:

Proposition 27. *For any given $S \subseteq [d]$, $\exists \epsilon = O\Big(\sqrt{\log(1/\delta)/N}\Big)$, such that*

$$\mathbb{P}\Big(\Big|\frac{1}{N}\sum_{n=1}^{N} y^n \phi_S^p(\mathbf{x}^n) - \hat{f}^p(S)\Big| > \epsilon\Big) < \delta \tag{5.1}$$

The proof for Proposition 27 follows standard concentration results and we omit the proof here. In fact, *Central Limit Theorem* can be applied and the distribution of $\sum_{n=1}^N f(\mathbf{x}^n)\phi_S^p(\mathbf{x}^n)/N$ converges in distribution to a Gaussian distribution with mean $\mathbb{E}_p[f(\mathbf{x})\phi_S^p(\mathbf{x})]$ and variance $\mathrm{Var}_p(f(\mathbf{x})\phi_S^p(\mathbf{x}))/N$. Note that by definition we have $\mathbb{E}_p[f(\mathbf{x})\phi_S^p(\mathbf{x})] = \hat{f}^p(S)$, and thus the estimation is *unbiased*. In the special case where $f(\mathbf{x})$ is Boolean-valued, i.e. $f(\mathbf{x}) \in \{-1, 1\}$, we have a fairly nice interpretation of the variance, as shown in Proposition 28.

Proposition 28. *If $f(\mathbf{x})$ is Boolean-valued, then*

$$\mathrm{Var}_p(f(\mathbf{x})\phi_S^p(\mathbf{x})) = 1 - \big[\hat{f}^p(S)\big]^2 = \sum_{T \neq S} \big[\hat{f}^p(T)\big]^2 \tag{5.2}$$

The proof of Proposition 28 can be found in Appendix C. Note that Proposition 28 basically states that *the more important coefficients are estimated more accurately.*

The discussion so far assumed noiseless observations. It turns out that considering random noise does not significantly change the results, as shown in Proposition 29.

Proposition 29. *Given training samples* $\{(\mathbf{x}^n, y^n)\}_{n=1}^N$ *where* $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N \overset{i.i.d}{\sim} p(\mathbf{x})$.

*(1) [**Additive Noise**] If* $y^n = f(\mathbf{x}^n) + \epsilon^n$*, where* $\epsilon^1, \epsilon^2, \cdots, \epsilon^N \overset{i.i.d}{\sim} \mathcal{N}(0, \sigma^2)$ *and* ϵ^n*'s are independent of the* $\mathbf{x}^n$*'s. Then* $\frac{1}{N}\sum_{n=1}^N y^n \phi_S^p(\mathbf{x}^n)$ *converges in distribution to a Gaussian with mean* $\hat{f}^p(S)$ *and variance* $\big(\mathrm{Var}(f(\mathbf{x})\phi_S^p(\mathbf{x})) + \sigma^2\big)/N$.

*(2) [**Random Sign Flip**] If* $y^n = s_n f(\mathbf{x}^n)$*, where* $s_n = 1$ *with probability* ρ *and* $s_n = -1$ *otherwise.* $s_1, s_2, \cdots, s_N$ *are i.i.d and independent of the input. Then* $\frac{1}{N}\sum_{n=1}^N y^n \phi_S^p(\mathbf{x}^n)$ *converges in distribution to a normal distribution with mean* $(2\rho - 1)\hat{f}^p(S)$ *and variance* $\Big(\mathrm{Var}(f(\mathbf{x})\phi_S^p(\mathbf{x})) + (1 - (2\rho-1)^2)\big[\hat{f}^p(S)\big]^2\Big)/N$.

Proof of Proposition 29 is included in Appendix C. A more realistic but much more challenging setup is to allow ϵ^n to depend on $\mathbf{x}^n$. This is known as *agnostic learning* and is beyond the scope of this thesis.

5.1.2 Given Correct Structure of Input Distribution

In practice, it is rarely the case that the learning algorithm has full access to the true input distribution $p(\mathbf{x})$. In this section, we relax the assumption to knowing only the **correct structure** of $p(\mathbf{x})$ and allow the parameters to be estimated from training data. Let us denote the resulting distribution as $q(\mathbf{x})$. If not otherwise stated, we use the Maximum-Likelihood estimates of the parameters $p(x_i = -1 | \mathbf{x}_{\pi_i} = \mathbf{x}_{\pi_i})$. It is easy to show that the ML estimates can be obtained by frequency counting.

As a simple concrete example, consider the subset $S = \{1\}$ and assume $\pi_1 = \varnothing$. We estimate $q(x_1 = b)$ by $\frac{1}{N}\sum_{n=1}^N \mathbb{1}(x_1^n = b)$, $b \in \{-1, 1\}$. Thus the corresponding Fourier coefficient $\hat{f}^q(\{1\})$ is approximated by

$$\frac{1}{N}\sum_{n=1}^N y^n \phi_S^q(\mathbf{x}^n) = \frac{1}{N}\sum_{n=1}^N y^n x_1^n \sqrt{\frac{q(x_1 = -x_1^n)}{q(x_1 = x_1^n)}} = \frac{1}{N}\sum_{n=1}^N y^n x_1^n \sqrt{\frac{\sum_{m=1}^N \mathbb{1}(x_1^m = -x_1^n)}{\sum_{m=1}^N \mathbb{1}(x_1^m = x_1^n)}} \quad (5.3)$$

Again let $v_n \triangleq y^n \phi_S^q(\mathbf{x}^n)$, and thus $\sum_{n=1}^N y^n \phi_S^q(\mathbf{x}^n)/N = \sum_{n=1}^N v_n/N$. But note that v_n's are no longer independent because parameters of $q(\mathbf{x})$ depend on all training samples. As a result, classic Central Limit Theorem (CLT) can no longer be directly applied.

It turns out that concentration results very similar to the case of fully given $p(\mathbf{x})$ can be achieved, but more powerful machinery is needed. Below we introduce *the generalized perturbative approach to Stein's method* (Chatterjee [2008]), which can be used to prove CLT-like results for general function of i.i.d samples. In contrast, the classic CLT assumes a special form of the function, namely the sample mean. The introduction below closely follows Section 3 of Chatterjee [2014].

Let $\mathcal{X}$ be a measure space and let $\mathbf{X} = (\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N)$ be a sequence of i.i.d random variables each taking value from $\mathcal{X}$. Let $h : \mathcal{X}^N \mapsto \mathbb{R}$ be an arbitrary measurable function and let $W \triangleq h(\mathbf{X})$. Without loss of generality, assume W is appropriately shifted and scaled s.t. $\mathbb{E}[W] = 0$ and $\text{Var}(W) = 1$. The goal is to obtain an upper bound for the distance between W and a standard normal random variable.

The generalized perturbative approach, as the name suggests, gets the upper bound by considering how $h(\mathbf{X})$ changes when its input is perturbed, i.e. some $\mathbf{x}^n$'s are modified. In particular, consider an independent copy of $\mathbf{X}$, which we denote as $\tilde{\mathbf{X}} = (\tilde{\mathbf{x}}^1, \tilde{\mathbf{x}}^2, \cdots, \tilde{\mathbf{x}}^N)$. For any $A \subseteq [N]$, let $\mathbf{X}^A$ denotes the random sequence whose n^{th} variable is $\tilde{\mathbf{x}}^n$ if $n \in A$ and $\mathbf{x}^n$ if $n \notin A$. $\forall A \subseteq [N]$, $n \notin A$,

$$\Delta_n h^A \triangleq h(\mathbf{X}^A) - h(\mathbf{X}^{A \cup \{n\}}) \tag{5.4}$$

Let us write $\Delta_n h^{\varnothing}$ simply as $\Delta_n h$. $\forall A \subseteq [d], A \neq [N]$, define a weight

$$\nu(A) \triangleq \frac{1}{N\binom{N-1}{|A|}} \tag{5.5}$$

Then we can define a weighted sum of 'perturbations':

$$T \triangleq \frac{1}{2} \sum_{n=1}^{N} \sum_{A \subseteq [N] \backslash \{n\}} \nu(A) \Delta_n h \Delta_n h^A \tag{5.6}$$

Proposition 30 (Theorem 3.1 of Chatterjee [2014]). *Let $Z \sim \mathcal{N}(0,1)$ be the standard normal random variable. Let W, T, and $\Delta_n h$ be defined as above. Then*

$$\sup_{t \in \mathbb{R}} \left| \mathbb{P}(W \leq t) - \mathbb{P}(Z \leq t) \right| \leq 2 \left(\sqrt{\operatorname{Var}(\mathbb{E}[T|W])} + \frac{1}{4} \sum_{n=1}^{N} \mathbb{E}[\left| \Delta_n h \right|^3] \right)^{1/2} \tag{5.7}$$

The proof of Proposition 30 can be found in Chatterjee [2014]. Let us make a few comments before moving on to apply it on our problem:

- In practice, the term $\operatorname{Var}(\mathbb{E}[T|W])$ is often replaced by an upper-bounded $\operatorname{Var}(\mathbb{E}[T|X])$ or $\operatorname{Var}(T)$, which are easier to compute.

- Roughly speaking, Equation (5.7) states that W is asymptotically normal as long as (1) the influence of each variable to $h(\mathbf{X})$ is relatively small (controlled by the terms $\sum_{n=1}^{N} \mathbb{E}[\left|\Delta_n h\right|^3]$) , and (2) the influences from single variables are not too dependent (controlled by the term $\operatorname{Var}(\mathbb{E}[T|W])$).

- The rate of convergence to normal distribution is not necessarily tight. It is known that variants of Stein's method often provide suboptimal rates. In fact, if we apply Equation (5.7) to the classic CLT setup (i.e. $W = h(\mathbf{X}) = \sum_{n=1}^{N} \mathbf{x}^n / N$ and assume $\mathbf{x}^n \in \mathbb{R}$), the right hand side is $O(N^{-1/4})$, while the optimal convergence rate obtained by Berry-Esseen Theorem is $O(N^{-1/2})$.

The following proposition specializes Proposition 30 to our problem, i.e. the estimation of Fourier coefficients assuming the correct structure of $p(\mathbf{x})$ is known.

Proposition 31. *Given training data $\{(\mathbf{x}^n, y^n)\}_{n=1}^{N}$ where $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N \overset{i.i.d}{\sim} p(\mathbf{x})$ and $y^n = f(\mathbf{x}^n), \forall n$. Let $q(\mathbf{x})$ be the distribution that has the same directed graph representation as $p(\mathbf{x})$ and the parameters are estimated empirically*[1] *from training*

[1] Since we are doing asymptotic analysis here, we assume N is large enough that all parameters that are not exactly 0 or 1 will not be estimated to 0 or 1 from training data.

data. Then $\forall S \subseteq [d]$, our estimate of the Fourier coefficient of S

$$W_S \triangleq \frac{1}{N}\sum_{n=1}^{N} \mathsf{y}^n \phi_S^q(\mathbf{x}^n) \tag{5.8}$$

converges in distribution to a normal $\mathcal{N}(\mu_S, \sigma_S^2)$, where $\mu_S = \hat{f}^p(S) + O(1/N)$, and $\sigma_S^2 = O(1/N)$. The rate of convergence[2] of the cdf is at least $O(n^{-1/4})$.

This proposition is the direct result of applying the *generalized perturbative approach of Stein's method* to our problem. Here we choose $\mathcal{X} = \{-1, 1\}^d$ and

$$h_S(\mathbf{X}, \mathbf{y}) = \frac{W_S - \mu_S}{\sigma_S}$$

The proof of Proposition 31 is somewhat technical and involved, we provide a proof sketch in Appendix C.

5.2 Analysis of the Direct Method

Section 5.1 provides detailed analysis for how well we can estimate a single Fourier coefficient from training data. Of course what one actually cares about is to have good predictive performance on test data rather than computing these coefficients.

As mentioned before, Algorithm 2 and Algorithm 3 are hard to analyse in general. Instead, this section provides theoretical guarantees for Algorithm 1, which can be thought of as a 'lower bound' on performance.

We focus on the case where $p(\mathbf{x})$ is fully known and $f(\mathbf{x})$ takes value from $\{-1, 1\}$ (i.e. classification). Again assume we have training data $\{(\mathbf{x}^n, y^n)\}_{n=1}^N$ where $\mathbf{x}^n \overset{\text{i.i.d}}{\sim} p(\mathbf{x})$ and $y^n = f(\mathbf{x}^n), \forall n$. Given a fixed test point $\mathbf{z}$ generated from the same $p(\mathbf{x})$,

[2]The constant depends exponentially on the cardinality of S.

Algorithm 1 outputs the sign of

$$\sum_{S\in\mathcal{C}} \left[\frac{1}{N}\sum_{n=1}^{N} y^n \phi_S^p(\mathbf{x}^n)\right]\phi_S^p(\mathbf{z}) \tag{5.9}$$

as the prediction of the label for $\mathbf{z}$. Without loss of generality, let us assume the true label is -1. The prediction will be correct as long as the quantity in Equation (5.9) is negative. Thus we allow an estimation error of about magnitude 1. Proposition 27 states that the error on each coefficient estimation is with high probability $O(1/\sqrt{N})$. Naively applying the union bound, with N samples, we can well-approximate functions whose spectrum is concentrated on $O(\sqrt{N})$ subsets, i.e. $|\mathcal{C}| = O(\sqrt{N})$.

However, union bound is fairly pessimistic. We expect the estimation errors on some coefficients to be smaller than the upper bound suggests. Moreover, we expect positive and negative errors to cancel out each other to some extent. It turns out we can in fact accommodate $|\mathcal{C}| = O(N)$ coefficient estimations, as shown by Proposition 2.

Theorem 2. *The distribution of* $\sum_{S\in\mathcal{C}}\left[\sum_{n=1}^{N} y^n\phi_S^p(\mathbf{x}^n)\right]\phi_S^p(\mathbf{z})/N$ *converges to a normal distribution* $\mathcal{N}(\mu(\mathbf{z}),\sigma^2(\mathbf{z}))$*, where*

$$\mu(\mathbf{z}) = \sum_{S\in\mathcal{C}} \hat{f}^p(S)\phi_S^p(\mathbf{z}) \tag{5.10}$$

$$\sigma^2(\mathbf{z}) \le \frac{1}{N}\sum_{S\in\mathcal{C}}\prod_{i\in S}\frac{p(-z_i|\mathbf{z}_{\pi_i})}{p(z_i|\mathbf{z}_{\pi_i})} \tag{5.11}$$

Proof of Theorem 2. Note that with fixed $\mathbf{z}$, full knowledge of $p(\mathbf{x})$ and independent $\mathbf{x}^n$, $v_n \triangleq y^n\sum_{S\in\mathcal{C}}\phi_S^p(\mathbf{x}^n)\phi_S^p(\mathbf{z})$ are also i.i.d.

$$\mathbb{E}[\sum_{S\in\mathcal{C}}\big[\frac{1}{N}\sum_{n=1}^{N} y^n\phi_S^p(\mathbf{x}^n)\big]\phi_S^p(\mathbf{z})] = \sum_{S\in\mathcal{C}}\phi_S^p(\mathbf{z})\mathbb{E}[f(\mathbf{x})\phi_S^p(\mathbf{x})]$$
$$= \sum_{S\in\mathcal{C}}\phi_S^p(\mathbf{z})\mathbb{E}\Big[\big(\sum_{T\in\mathcal{C}}\hat{f}^p(T)\phi_T^p(\mathbf{x})\big)\phi_S^p(\mathbf{x})\Big] = \sum_{S,T\in\mathcal{C}}\phi_S^p(\mathbf{z})\hat{f}^p(T)\langle\phi_S^p,\phi_T^p\rangle_p = \sum_{S\in\mathcal{C}}\hat{f}^p(S)\phi_S^p(\mathbf{z})$$

$$\mathrm{Var}\bigg(\sum_{S\in\mathcal{C}}\big[\frac{1}{N}\sum_{n=1}^{N}y^n\phi_S^p(\mathbf{x}^n)\big]\phi_S^p(\mathbf{z})\bigg) = \mathrm{Var}\bigg(\frac{1}{N}\sum_{n=1}^{N}y^n\sum_{S\in\mathcal{C}}\phi_S^p(\mathbf{x}^n)\phi_S^p(\mathbf{z})\bigg)$$
$$= \frac{1}{N}\mathrm{Var}_{\mathbf{x}\sim p}\bigg(f(\mathbf{x})\sum_{S\in\mathcal{C}}\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{z})\bigg) \leq \frac{1}{N}\mathbb{E}\bigg[\bigg(f(\mathbf{x})\sum_{S\in\mathcal{C}}\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{z})\bigg)^2\bigg]$$
$$= \frac{1}{N}\mathbb{E}\bigg[f^2(\mathbf{x})\sum_{S,T\in\mathcal{C}}\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{z})\phi_T^p(\mathbf{x})\phi_T^p(\mathbf{z})\bigg] = \frac{1}{N}\sum_{S,T\in\mathcal{C}}\phi_S^p(\mathbf{z})\phi_T^p(\mathbf{z})\mathbb{E}[\phi_S^p(\mathbf{x})\phi_T^p(\mathbf{x})]$$
$$= \frac{1}{N}\sum_{S,T\in\mathcal{C}}\phi_S^p(\mathbf{z})\phi_T^p(\mathbf{z})\mathbb{1}(S=T) = \frac{1}{N}\sum_{S\in\mathcal{C}}\big[\phi_S^p(\mathbf{z})\big]^2 = \frac{1}{N}\sum_{S\in\mathcal{C}}\prod_{i\in S}\frac{p(-z_i|\mathbf{z}_{\pi_i})}{p(z_i|\mathbf{z}_{\pi_i})}$$

□

Let us make a few comments on Proposition 2 before moving on.

Firstly, consider the special case where $p(\mathbf{x})$ is indeed uniform over $\{-1,1\}^d$. In this case the right hand side of Equation (5.11) equals $|\mathcal{C}|/N$, thus we can have $|\mathcal{C}| = N$ and still with high probability have the correct prediction of the label. In general the bound on $|\mathcal{C}|$ depends on parameters of $p(\mathbf{x})$, but we still have $|\mathcal{C}| = O(N)$.

Secondly, notice that the variance bound depends on the test point $\mathbf{z}$. In particular, the right hand side of Equation (5.11) is smaller with when $p(\mathbf{z})$ is larger and vice versa. Intuitively, we are more confident about predicting for a test point that is from a "highly-populated" region because we have seen many "similar" points in training. Of course the exact meaning of "similarity" depends on assumptions made on $f(\mathbf{x})$ (e.g. smoothness) and on $p(\mathbf{x})$ (e.g. structural assumption on the directed graph representing $p(\mathbf{x})$).

Chapter 6

Experiment Results

This chapter demonstrates the practical performance of the proposed spectrum-based learning algorithms on various synthetic and real datasets.

As mentioned in Chapter 3, the direct generalization of Low Degree Algorithm (i.e. Algorithm 1) does not yield competitive performance. This argument is experimentally validated in Section 6.1. The rest of this chapter only focuses on the feature selection variants, namely Algorithm 2, Spectrum-Based Feature Selection and Algorithm 3, Spectrum-Based Subset Selection. Several popular downstream learning methods are considered, including *Support Vector Machine (SVM) with RBF kernels or polynomial kernels*, *Random Forest (RF)*, *Gradient Boosted Random Trees (GBRT)*, *Multi-Layer Perceptron*[1] *(MLP)* and *k-Nearest Neighbours (KNN)*. But we emphasize linear models, in particular **Logistic Regression** and **Linear Regression**, due to their good interpretability.

The experiments are divided into two categories: results on *synthetic datasets* are presented in Section 6.2 and results on *real datasets* are presented in Section 6.3. Synthetic datasets allow full control of the target function and input distribution, thus can help us better understand how various factors impact the learning performance. On the other hand, real datasets present more realistic scenarios and challenges, and

[1] 2-3 layers, not deep ones.

thus are what we actually care about. To provide both depth and breadth, we study one particular dataset in detail (*Tribes functions* for synthetic data and *MNIST* for real data) and also present summarized results for several other datasets.

Finally, Section 6.4 offers some practical guidelines and tips on how to apply the proposed methods, e.g. how to encode the raw features, how to set various hyper-parameters, how to choose structural assumptions on input distribution $p(\mathbf{x})$ etc.

The performance measure we use for classification (binary or multi-class) tasks, if not otherwise stated, is prediction accuracy[2]. For regression tasks, MSE (mean square error) is reported.

There are three types of baselines with which we compare our proposed methods.

1. **Raw features**. We compare the results from proposed methods with learning using the raw features, assuming the same learning method. This provides an idea of how the inclusion of non-linear features impacts learning performance.

2. **Generic feature selection methods**. It is a fairly natural idea to consider non-linear combinations of raw features. As a baseline, one can always naively generate *low degree features* up to a given d_0: $\{\mathbf{x}_S \triangleq \prod_{i\in S} x_i\}_{S\subseteq[d]:|S|\le d_0}$ and then resort to some general purpose feature selection techniques[3]. We show the proposed methods turn out to be a better feature selection scheme.

3. **Previous works**. For the real datasets, we also compare our results with those reported in previous works that cite the datasets.

We use the Python machine learning library *scikit-learn* (Pedregosa et al. [2011]) for the implementations of downstream learning methods as well as the general

[2]Of course classification accuracy is not a suitable measure if the class labels are very imbalanced. Imbalanced classification is itself an active area of research (see, e.g. Garcia and He [2008]), but this thesis does not attempt to deal with it.

[3]Section 3.5.1 has briefly discussed available feature selection methods and which ones we plan to use as baseline.

purpose feature selection methods. In particular, the learning methods are used with *essentially no parameter tuning*, i.e. the default values are used for hyperparameters such as degree in the polynomial kernel, number of trees in the Random Forest, and number of hidden layers and hidden units in Multi-Layer Perceptron. The only thing we control is what features are fed into these learning methods.

6.1 Feature Selection Variants are Preferred

Through three concrete examples, this section shows that among the proposed methods, the direct generalization of Low Degree Algorithm (i.e. Algorithm 1) is consistently out-performed by the feature selection variants (i.e. Algorithms 2 and 3). As a result, the rest of this chapter focuses on the latter.

6.1.1 Tribes Function

This is a *binary classification* task on *synthetic data*. Let us first define the Tribes function, which is studied in much greater details in Section 6.2.1.

Definition 12. *Let* $d = w \cdot s$*, then the* ***Tribes*** *function of width* w *and size* s*,* $\text{Tribes}_{w,s} : \{-1,1\}^d \mapsto \{-1,1\}$*, is defined by*

$$\text{Tribes}_{w,s}(\mathbf{x}) = (x_1 \wedge x_2 \wedge \cdots \wedge x_w) \vee \cdots \vee (x_{w(s-1)+1} \wedge x_{w(s-1)+2} \wedge \cdots \wedge x_{ws}) \qquad (6.1)$$

Intuitively, Tribes function can be viewed as a voting scheme: there are s 'tribes', each with the same number of members. A tribe collectively votes 'yes' if all its members unanimously voted 'yes', and the final outcome is 'yes' as long as one of the tribes voted 'yes'. Tribes function is first introduced in Ben-Or and Linial [1985] and with the notable feature of (with suitable choices of w and s) being essentially unbiased and yet all the individual variable influence is quite tiny. For our purposes, this is a family of non-trivial Boolean functions for which we can readily understand and control the hyperparameters.

Consider training data $\{(\mathbf{x}^n, y^n)\}_{n=1}^N$ where $\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^N$ are i.i.d. generated from some product distribution $p(\mathsf{x}) = \prod_{i=1}^d p(x_i)$ and $y^n = \text{Tribes}_{w,s}(\mathbf{x}^n)$, $n = 1, 2, \cdots, N$. The parameters $p_i \triangleq \mathbb{P}(x_i = -1)$ are randomly generated from a symmetric Beta distribution Beta(α, α), and we vary the choice of α to control how 'non-uniform' $p(\mathsf{x})$ is. If α is large, $p_i \approx 0.5$ with high probability and $p(\mathsf{x})$ is essentially the uniform distribution; the smaller α is, the more likely p_i takes values closer to 0 or 1. Figure 6-1 shows the probability density function of Beta(α, α) for a few choices of α.

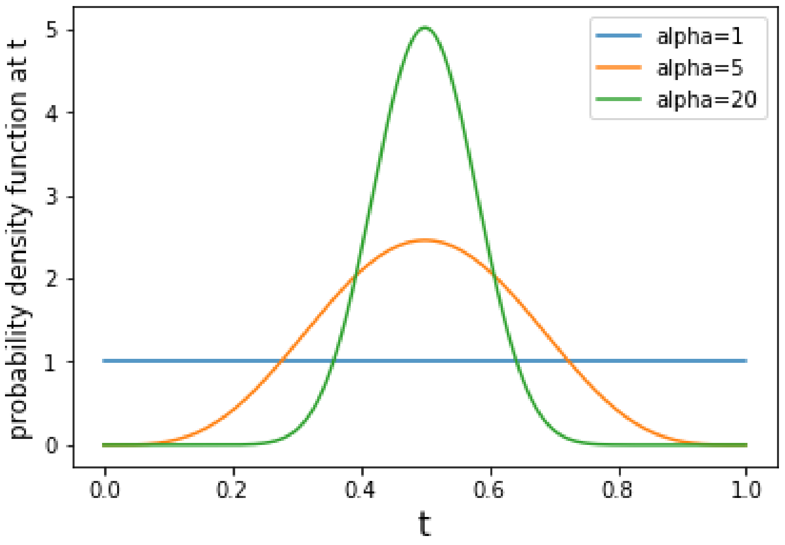


Figure 6-1: The pdf of Beta(α, α), for a few choices of α.

Figure 6-2 shows the performance of the three proposed algorithms on $\text{Tribes}_{3,5}(\mathsf{x})$ function with varying dataset size N. The vertical axis represents accuracy, thus higher bars indicate better performance. The values plotted are averaged over 10 independent runs on the same choice of parameters (α, N) and in each run we perform a 5-fold cross validation. Note that each run has a different choice of $p(\mathsf{x})$: for each run a new set of parameters is generated, but all the parameters are generated from the same distribution Beta(α, α). All three algorithms used $d_0 = 2$ and we individually optimize the number of features to use for each variant based on a coarse grid search. Both Algorithm 2 and Algorithm 3 used Logistic Regression as downstream classifier.

As is evident from the plots, Algorithm 2 and Algorithm 3 have a clear advantage over Algorithm 1. The same trend holds for other structural choices of $p(\mathbf{x})$, and a few examples are shown in Figure 6-3.

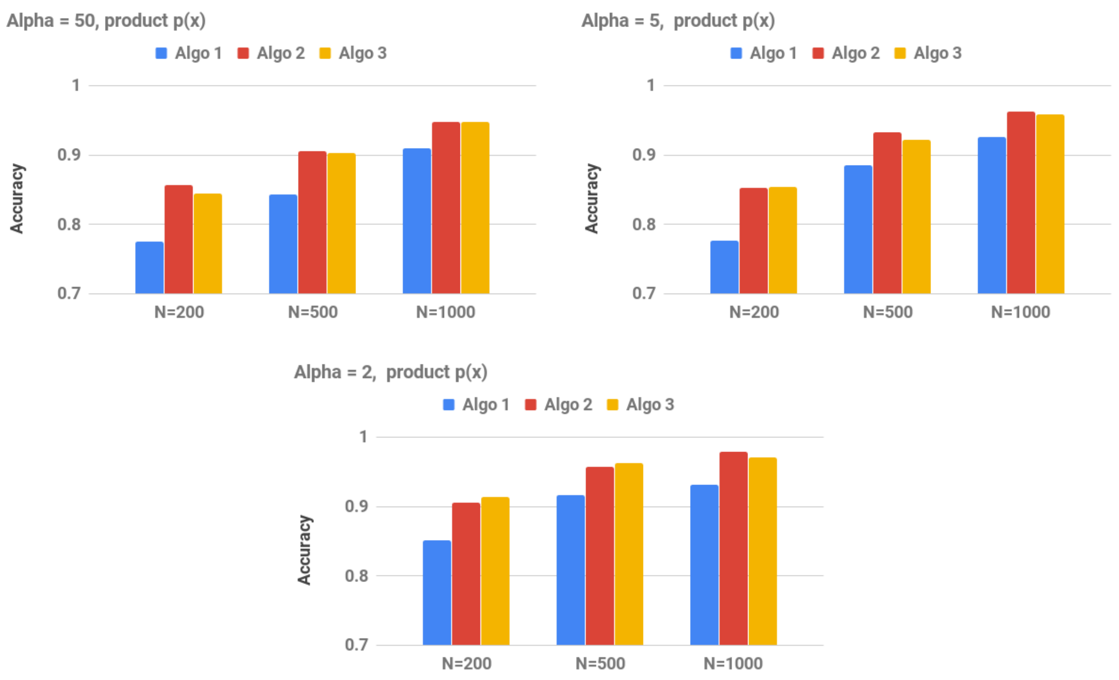
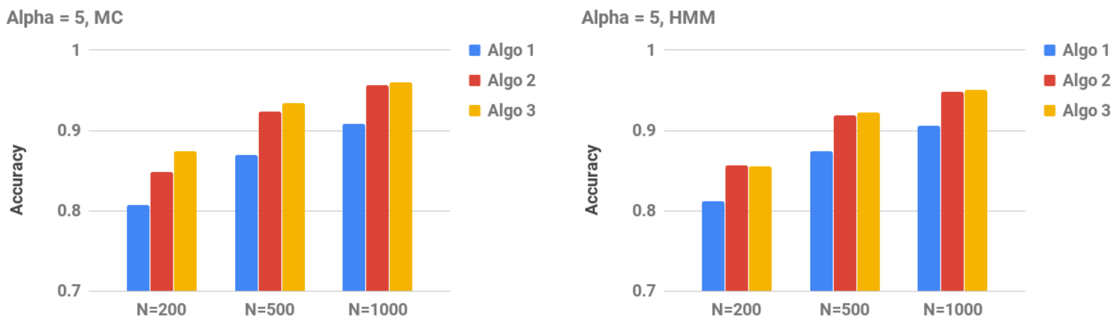


Figure 6-2: Comparison of the proposed algorithms for Tribes function under product distribution, with various choices of α and N. The plots assumed $w = 3$, $s = 5$ and thus $d = 15$.



(a) Markov Chain $p(\mathbf{x})$ (b) Hidden Markov Model $p(\mathbf{x})$

Figure 6-3: Comparison of the proposed algorithms under other structural assumptions of $p(\mathbf{x})$.

6.1.2 Low Degree Polynomial

This is a *regression* task on *synthetic data*. This synthetic dataset is also used in Section 6.2.2. We generate a polynomial as follows. Fix some positive integer t, and for $i = 1, 2, \cdots, t$: first draw a k_i from a Poisson(λ); then uniformly randomly pick a $S_i \subseteq [d]$ among all sets of cardinality k_i; generate a coefficient $a_i \sim \mathcal{N}(0, 1)$. The resulting polynomial is

$$f(\mathbf{x}) = \sum_{i=1}^{t} a_i \prod_{j \in S_i} x_j \tag{6.2}$$

We then generate dataset $\{(\mathbf{x}^n, f(\mathbf{x}^n))\}_{n=1}^{N}$ where $\mathbf{x}^n$'s are generated i.i.d from some $p(\mathbf{x})$. Figure 6-4 compares the learning performance of the three proposed algorithms under different choices of $p(\mathbf{x})$ structure and different dataset sizes. Note that the vertical axis here is the Mean Square error, thus lower bars are preferred.

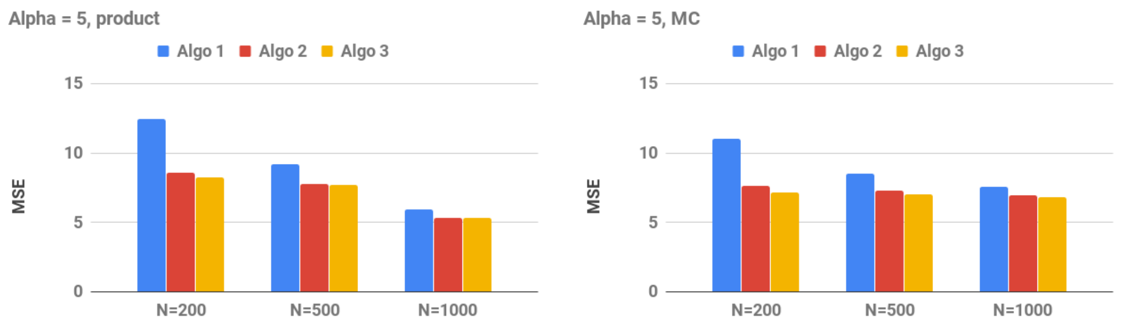


Figure 6-4: Comparison of the proposed algorithms for a synthetic *regression* dataset. The plots are generated with $d = 15$, $t = 20$, and $d_0 = 2$.

Similar to the Tribes function case, 5-fold cross validation is performed. We use $d_0 = 2$ and optimize over the number of features for each variant separately. Linear Regression is used as downstream learning method for Algorithms 2 and 3. The parameters in $p(\mathbf{x})$ are all generated independently from a Beta(α, α) distribution with $\alpha = 5$, and we show the averaged results from 50 independent runs.

Clearly, Algorithm 1 is again out-performed by Algorithms 2 and 3, especially at relatively small training data size. Note that asymptotically as $N \to \infty$, Algorithms 1 and 2 are expected to converge to the same solution.

6.1.3 Congressional Voting Records Dataset

This is a *binary classification* task on a *real dataset* from the UCI repository (Dua and Taniskidou [2017]). There are 435 entries in total and 16 binary features with some feature values missing. Each entry represents a congress representative and the features are his/her votes on 16 key issues. The label represents the representative's party (2 classes, democrat or republican).

We use 2 bits to represent each original feature, one indicating whether the value is missing and if it's not, the other bit stores the value. Thus $d = 16 \times 2 = 32$. Again cross validation is used and we show the results for various choices of N in Figure 6-5.

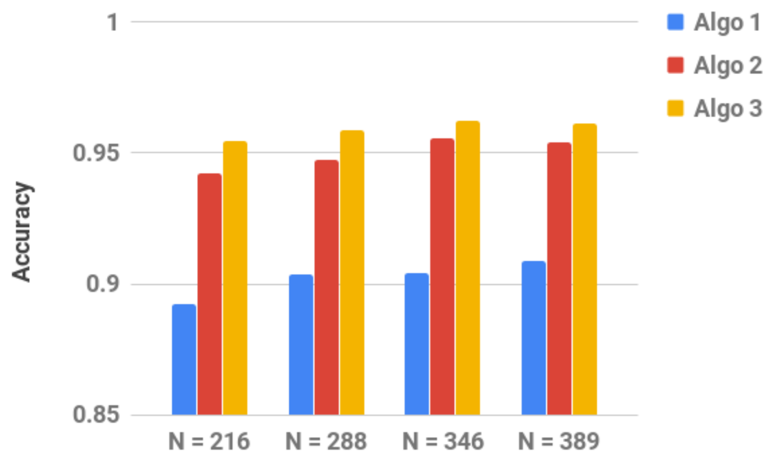


Figure 6-5: Comparison of the proposed algorithms on a real dataset: Congressional Vote dataset from UCI repository. Vertical axis represents accuracy.

The results reported are averaged over 5 runs, each time randomly permuting the ordering of the samples. As before, Logistic Regression is used after the feature selection step in both Algorithm 2 and 3. Note that unlike synthetic datasets, we do not have access to 'ground truth' of the structure of $p(\mathbf{x})$ here. We assumed product $p(\mathbf{x})$ to produce Figure 6-5. More results on this dataset, such as the performance under different structural assumptions, can be found in Section 6.3.2.

6.2 Synthetic Data

In this section and Section 6.3, the feature selection variants (Algorithms 2 and 3) among the proposed methods are applied on various synthetic and real datasets. Of course, the performance of any learning algorithm depends strongly on properties of the dataset at hand, but we do believe it is valuable to study some specific problems in detail to gain high level insights. In particular, we carefully study the family of Tribes function[4] in Section 6.2.1. Section 6.2.2 then verify the insights gained from the case study on various other synthetic datasets.

6.2.1 Case Study: Tribes Function

6.2.1.1 Simulation Setup

We carried out systematic simulations on three problem sizes[5]: (1) $w = 3, s = 5, d = 15$; (2) $w = 4, s = 6, d = 24$; and (3) $w = 4, s = 10, d = 40$. The observations are fairly consistent under different choices of d. These choices of (w, s, d) lead to classification problems with roughly balanced classes under uniform distribution. To be more precise, assuming uniform $p(\mathbf{x})$, cases (1) and (3) have $\mathbb{P}(y = -1) \approx 0.5$ and case (2) has $\mathbb{P}(y = -1) \approx 0.33$. In each case, a range of dataset sizes N are tested, roughly from $N = 100$ to $N = 5000$. If not otherwise stated, N includes both training and testing examples and 5-fold cross validation is used.

Several structural assumptions on $p(\mathbf{x})$ are considered: *Product*, *Markov Chain* (MC), *Hidden Markov Model* (HMM) and *Grid*. The schematics of the structures are shown in Figure 6-6. All the parameters $p(x_i|\mathbf{x}_{\pi_i})$ are independently generated from a symmetric Beta distribution, Beta(α, α), where α controls how 'non-uniform' $p(\mathbf{x})$ is. For example, a large α essentially leads to a uniform $p(\mathbf{x})$ while with α close to 0,

[4]Definition can be found in Section 6.1.1.

[5]The reason several choices of (w, s, d) are considered here is that due to hardware constraints, we will stick to a dataset of N at most a few tens of thousands and $d_0 \leq 3$ in the simulations. Varying problem size enables us to explore the full spectrum of 'small data' to 'large data' and also serves as a knob to control the difficulty of the problem.

one can simulate having sparse features. Note that it is only meaningful to compare α values within the same distribution structure, not among different structures. For example, on average, product distribution generated using α is more 'non-uniform' than a Markov Chain distribution generated from the same α in, say, total variation distance sense[6].

The labelling of the nodes are randomized in each run, i.e. the variable x_1 in Equation (6.1) can appear anywhere in the graph representing $p(\mathbf{x})$.

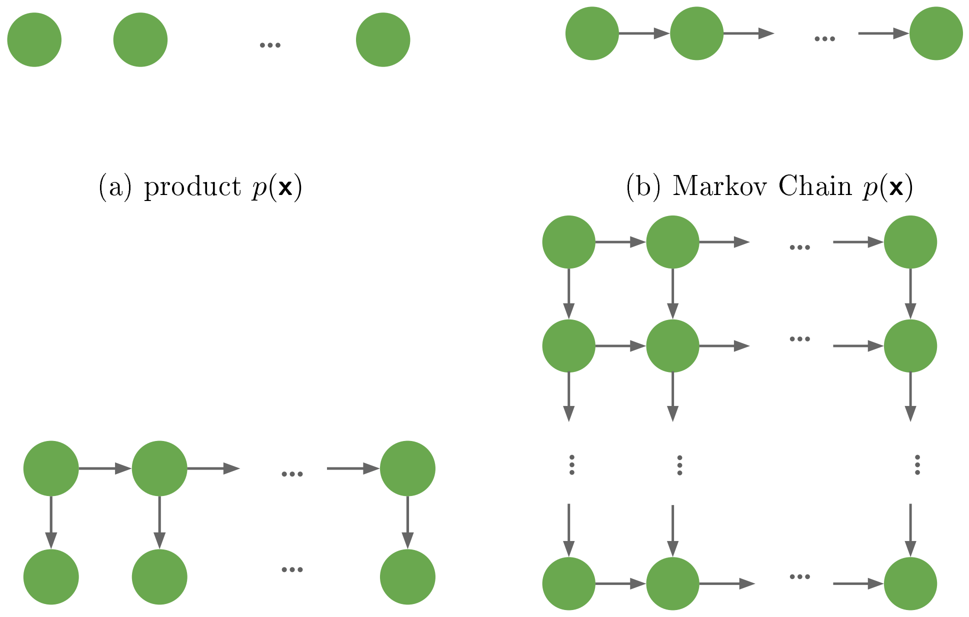

(a) product $p(\mathbf{x})$ (b) Markov Chain $p(\mathbf{x})$ (c) Hidden Markov Model $p(\mathbf{x})$ (d) Grid $p(\mathbf{x})$

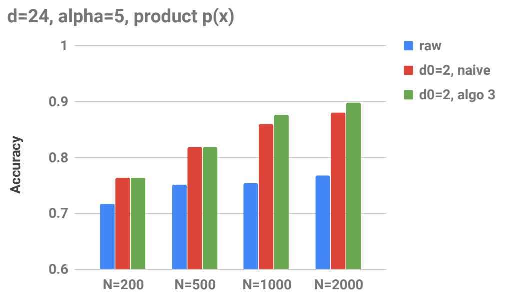
Figure 6-6: Schematics of structural assumptions on the input distribution

6.2.1.2 First Baseline: Raw Features

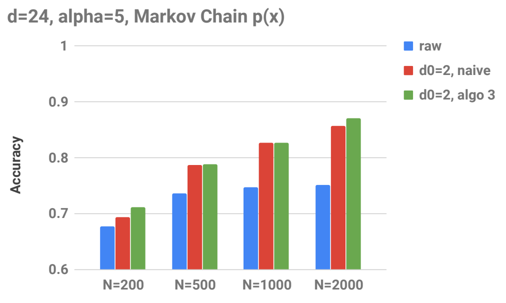
The proposed methods introduce non-linear combinations of the raw features. This can bring about two competing factors for the learning task at hand: access to extra information, and increased risk of overfitting. The two factors are clearly competing and which one dominates depends on the amount of available data, as well as the

[6]The same argument holds for other commonly used distance measures too, e.g. divergence, local TV distance, etc.

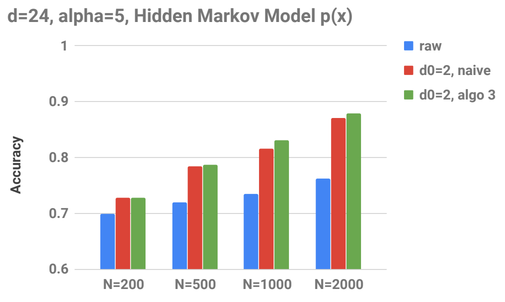
downstream learning method used. For example, the extra information is more helpful for the less powerful methods such as Logistic Regression. In fact, the more powerful methods are expected to learn such non-linear representations on their own from training data.



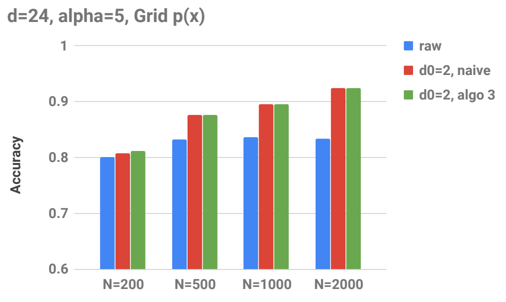
(a) product $p(\mathbf{x})$



(b) Markov Chain $p(\mathbf{x})$



(c) Hidden Markov Model $p(\mathbf{x})$



(d) Grid $p(\mathbf{x})$

Figure 6-7: Learning performance of Logistic Regression on the Tribes function under various structural assumptions. Note that the prediction accuracy improves significantly after including non-linear combinations of raw features for a wide range of dataset size. In the plots $\alpha = 5$, $w = 4, s = 6, d = 24$, $d_0 = 2$.

Figure 6-7 demonstrates the effect of including pairwise non-linear combinations of raw features (i.e. $d_0 = 2$) for various dataset sizes and input distributions. Three sets of features are considered in the comparison: the *blue* bars show the performance using raw features $x_1, x_2, \cdots, x_d$; the *red* bars represent results using 'naive features', i.e. the raw features as well as new features of the form $x_i x_j$, $\forall 1 \leq i < j \leq d$; the *green* bars correspond to Algorithm 3, i.e. feature selection is performed on the 'naive features' based on the estimated spectrum[7]. *Logistic Regression* is used as the

[7]In these experiments, we observe Algorithm 3 to provide similar or better performance than Algorithm 2. We defer a more detailed comparison of the two variants to Section 6.2.1.4.

learning algorithm in all three cases and vertical axis represents prediction accuracy, thus higher bars indicate better performance. The results shown are averaged over 10 runs, each run with independently generated $p(\mathbf{x})$ and training data. 5-fold cross validation is used.

Note that including the non-linear terms improved prediction accuracy in all cases shown in Figure 6-7 and the improvement is often quite significant. In some cases feature selection provides further improvement, but such improvement is quite small. Apparently, with Logistic Regression and $d_0 = 2$, the benefit of obtaining extra information dominates the potential risk for overfitting. If we keep including more non-linear features (i.e. increase d_0), overfitting will happen and feature selection becomes necessary, as shown in Figure 6-8. Note how the gap between red and green bars is larger for $d_0 = 3$. Figure 6-8 also confirms how a larger d_0 is helpful only with larger training dataset.

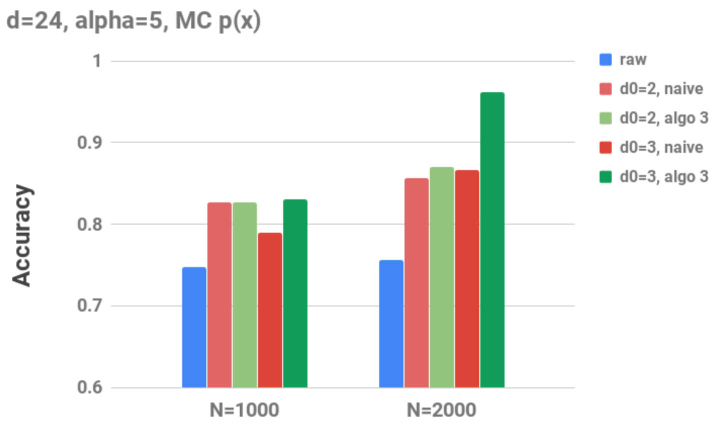
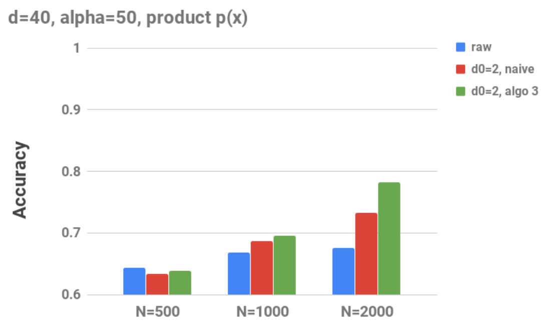
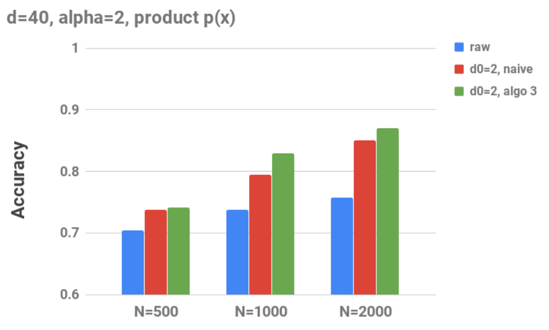


Figure 6-8: Overfitting starts to happen at $d_0 = 3$ and feature selection becomes important. $w = 4, s = 6, d = 24$ and $\alpha = 5$, as in Figure 6-7, and Markov Chain structured $p(\mathbf{x})$ is used. Observations under other structural assumptions are qualitatively very similar.

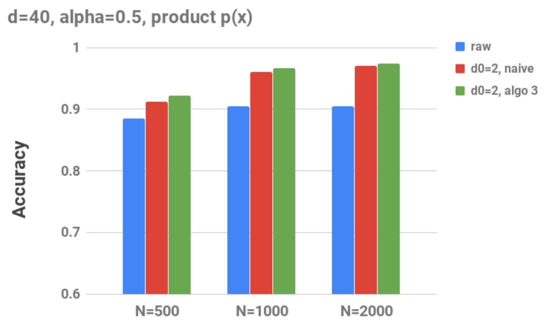
So far we have fixed $\alpha = 5$ and $d = 24$, but the above discussion applies to other choices of hyperparameters as well. For example, Figure 6-9 presents the effect of including non-linear features and subsequently performing feature selection for various choices of α and a different model size ($w = 4, s = 10, d = 40$). Note that Figure 6-9 also supports our intuition of *uniform distribution is the most difficult to learn from, and the learning is easier with more 'non-uniform'* $p(\mathbf{x})$.



(a) $\alpha = 50$



(b) $\alpha = 2$



(c) $\alpha = 0.5$

Figure 6-9: Performance with decreasing α, i.e. $p(\mathbf{x})$ becomes more 'non-uniform'. Here $p(\mathbf{x})$ is assumed to be a product distribution, $d = 40$, $d_0 = 2$. The interpretation of the colour bars are the same with Figure 6-7.

Although we focus on Logistic Regression for its simplicity and good interpretability, the features generated and selected by the proposed algorithms can be fed into any downstream learning algorithms. Figure 6-10 compares the performance of various classifiers: the first plot shows how the performance of each learning algorithm varies with number of features used, and the second plot summarizes the 'optimal' performances. Here Algorithm 2 is used to perform the feature selection[8]. Let us make a few more comments before moving on:

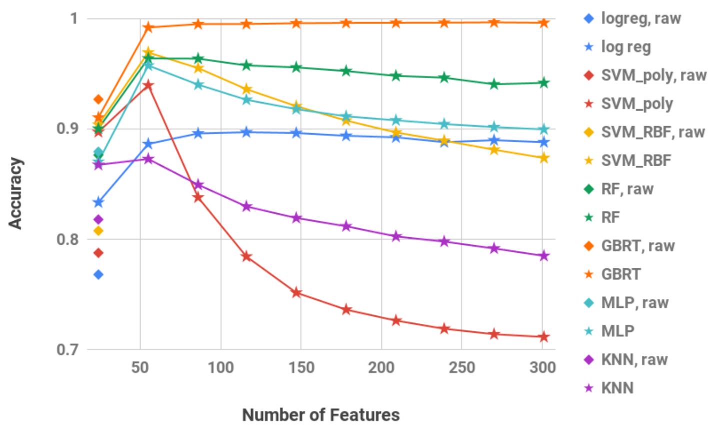
(1) All the learning methods considered here benefit from including non-linear

[8]With the choice of N, d and α here, Algorithms 3 has a clear advantage on Random Forest and GBRT compared to Algorithm 3, and the two variants perform similarly on the rest of the classifiers. More discussion to come in Section 6.2.1.4.

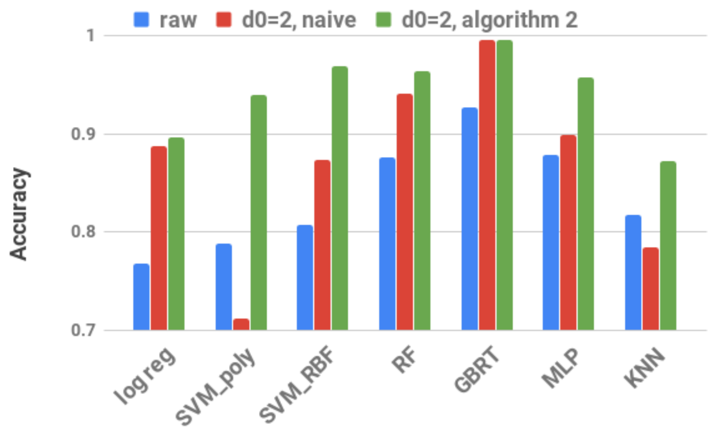
features with appropriate feature selection.

(2) As shown by Figure 6-8, overfitting will eventually kick in. But given the same amount of training data, some classifiers (e.g. Logistic Regression or GBRT) are relatively robust, while others are more prone to overfitting (e.g. SVM).

(3) With raw features, the performance of Logistic Regression is much worse than the more powerful methods such as RF, GBRT or MLP. However, by including the nonlinear features, Logistic Regression catches up quite a bit to the other classifiers but remains to be a linear model with intelligible features.



(a) How prediction accuracy varies with number of features.



(b) Effect of including and selecting nonlinear features.

Figure 6-10: Performance of other downstream learning algorithms. The plots are generated for product $p(\mathbf{x})$, $\alpha = 5$, $d = 24$, $N = 2000$ and $d_0 = 2$.

6.2.1.3 Second Baseline: Generic Feature Selection

Section 6.2.1.2 has shown that *given enough training data, adding properly chosen non-linear combinations of the raw features generally improves learning performance regardless of the learning algorithm used.* Note that including non-linear features to improve learning performance is not a new idea, and our contribution is proposing systematic methods to design the non-linear features in such a way that feature selection can be performed *'correctly'* and *efficiently.* This section aims to demonstrate the superiority of the proposed spectrum-based feature selection methods over general purpose feature selection techniques. Specifically, we consider the following baseline: generate 'naive' features up to degree d_0, i.e. $\mathsf{x}_S = \prod_{i \in S} \mathsf{x}_i$ for all $S \subseteq [d]$ with $|S| \leq d_0$ and apply general purpose feature selection methods.

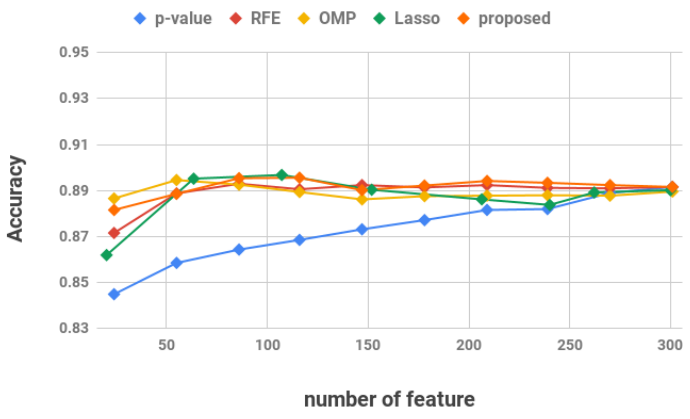
Four typical, commonly used feature selection methods are considered: (1) selection based on individual feature's p-value; (2) recursive feature elimination (RFE); (3) LASSO and (4) optimal matching pursuit (OMP). A more detailed discussion of these methods and why we choose to compare to them can be found in Section 3.5.1.

Figure 6-11 illustrates the comparison between the proposed methods[9] with the general purpose feature selection schemes listed above. The plots are generated for Tribes function with $d = 24$, $\alpha = 2$, and assuming a product input distribution $p(\mathsf{x})$. The x-axis represents accuracy, while the y-axis shows number of features chosen.

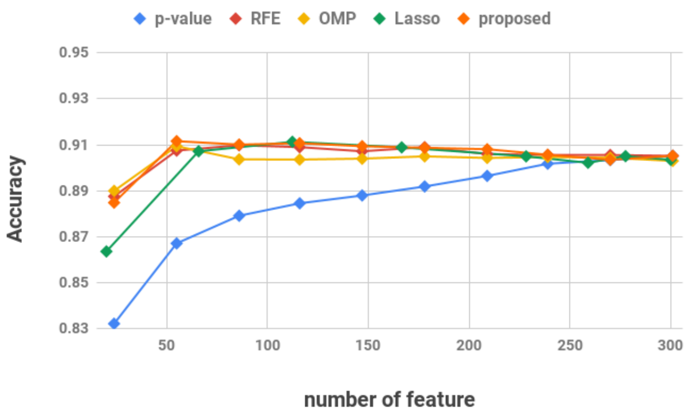
The plots clearly shows that all methods have very similar performance except for the p-value based one, which is significantly worse and the performance gap increases as the number of features decreases. This is expected because by making independent decisions for each feature, the p-value based method implicitly assumes *the features are independent.* This is in general not true, and becomes more problematic as α decreases, i.e. as $p(\mathsf{x})$ becomes less 'uniform'. Figure 6-12 demonstrates how the

[9]We take the better of Algorithms 2 and 3. We defer a more detailed comparison of the two variants to Section 6.2.1.4.

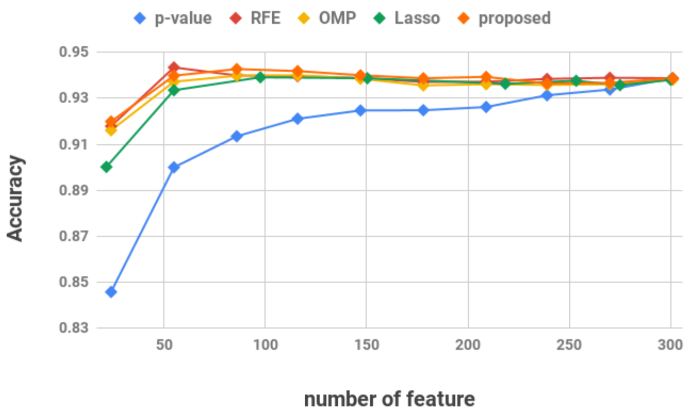
performance gap becomes more significant for smaller α.



(a) N=500



(b) N=1000



(c) N=2000

Figure 6-11: Comparing the proposed methods with p-value based feature selection. The plots are generated using $w = 4, s = 6, d = 24$ and product $p(\mathbf{x})$, whose parameters p_i's are i.i.d. generated from a Beta(2, 2) distribution.

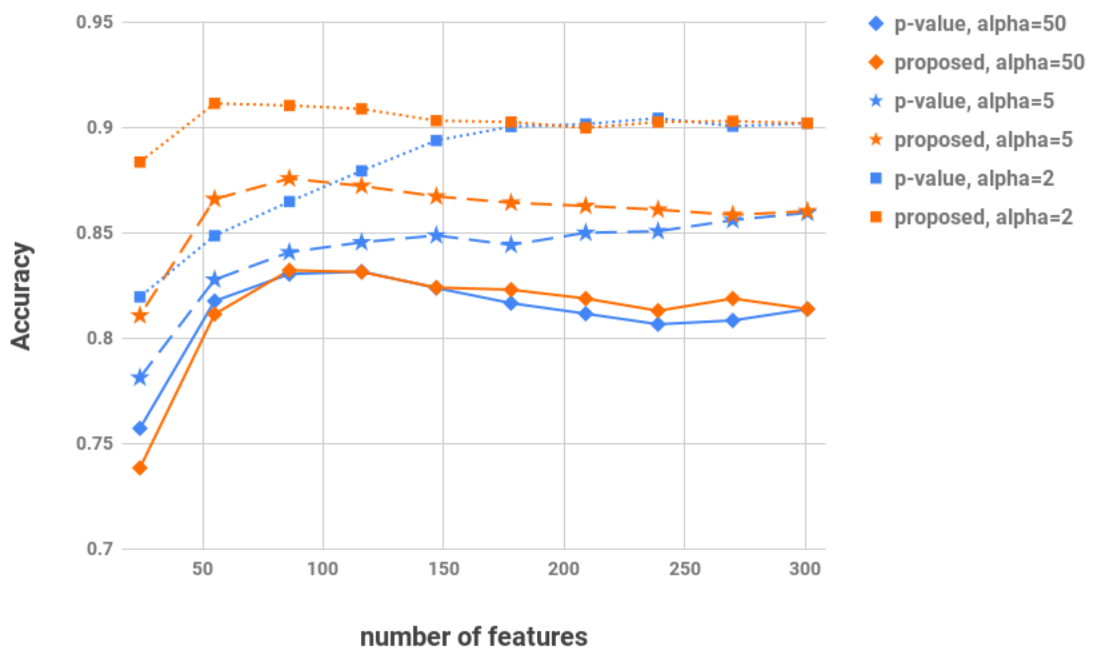


Figure 6-12: The performance gap between p-value based feature selection and more sophisticated feature selection methods is smaller when $p(\mathbf{x})$ is closer to uniform. This plot is generated for Tribes function with $d = 24$, $N = 1000$, $d_0 = 2$.

Table 6.1 summarizes the comparison of the feature selection methods in terms of the computational cost as well as some other properties. Note that we start with $O(d^{d_0})$ features and aim to reduce the number of features to K. As always, N denotes the amount of available data.

Method	Computational Complexity	Parallelizable?
p-value	$O(d^{d_0}N)$	Yes
RFE	$O(d^{d_0}T)$, where T is the cost to train the model of choice	No
LASSO	$O(d^{3d_0} + Nd^{2d_0})$ (Efron et al. [2004])	No
OMP	$O(d^{d_0}KN)$ (Rubinstein et al. [2008])	Partly
proposed	$O(d^{d_0}N)$	Yes

Table 6.1: Comparison of various feature selection methods.

We have a few more comments on the comparison of these feature selection methods:

(1) The proposed methods and the p-value based one share the lowest computational

cost and both can be fully parallelized, i.e. each candidate feature can be dealt with independently.

(2) The RFE method starts with all the features, and iteratively discards features, one at a time, by training a model and finding the 'least important feature' in the model. Thus it is computationally very expensive and starts to be infeasible even for a relatively small learning problem.

(3) For the LASSO method, the number of chosen features (i.e. with non-zero parameters) is indirectly controlled by the regularization constant. In other words, one cannot exactly specify the model size.

(4) The OMP method starts with an empty set and iteratively adds features that best explain the 'residual signal', one at a time. Within each iteration, the step of determining which feature to add next is parallelizable, but the method as a whole is not.

(5) The results shown here assumed a somewhat decent dataset size. If very little data is available, the proposed methods are probably not a good idea as the estimation of neither the parameters of $p(\mathbf{x})$ nor the Fourier coefficients will be accurate. Figure 6-13 shows results for $N = 200$. Note that at small N, the performance varies dramatically for all the feature selection methods across different runs, thus the curves in Figure 6-13 seem all over the place even after averaging over 10 independent runs.

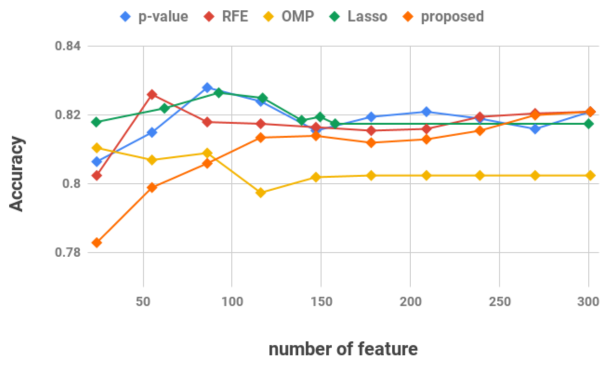
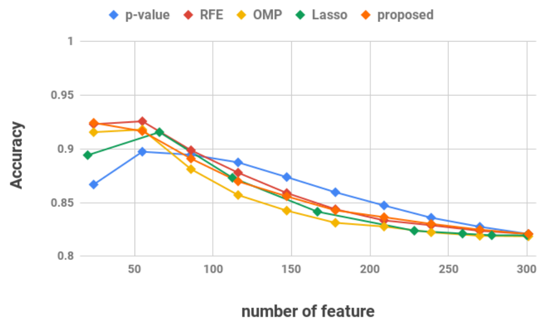
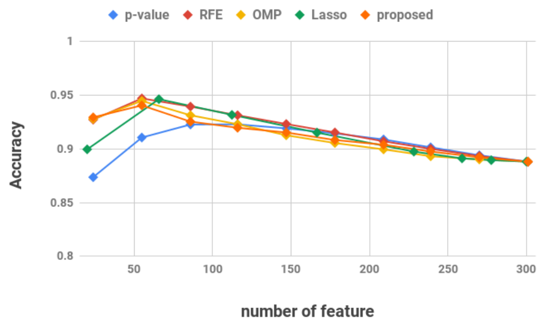


Figure 6-13: Tribes function with $d = 24$, $N = 200$, $d_0 = 2$, $\alpha = 2$.

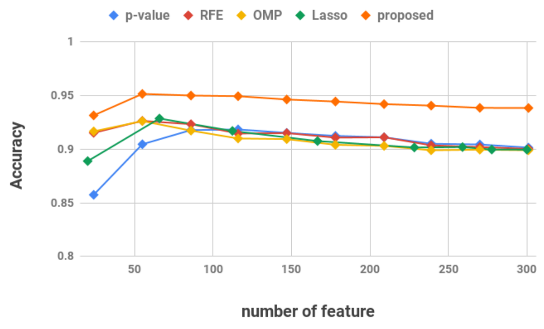
Despite our emphasis on Logistic Regression, we observe that the proposed methods provide improvement for some other downstream learning methods that was not expected a priori. Figure 6-14 compares different feature selection schemes using classifiers other than Logistic Regression.



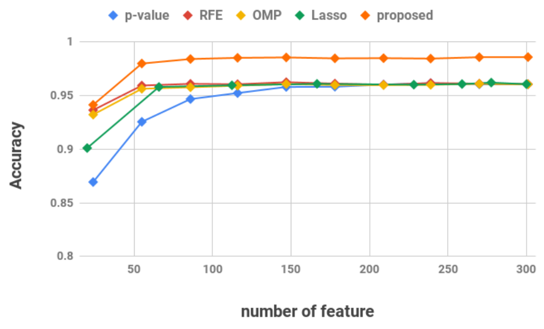
(a) SVM-poly



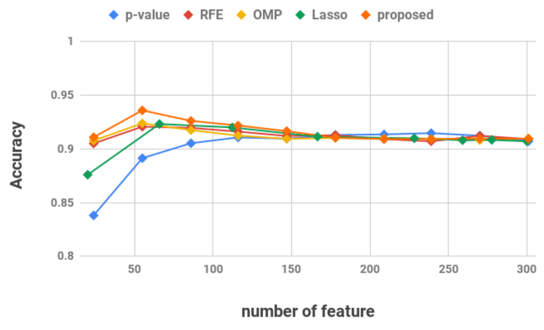
(b) SVM-RBF



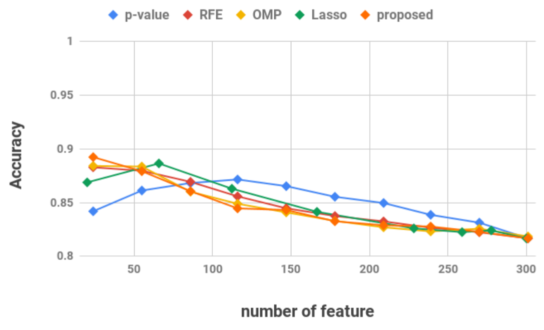
(c) Random Forest



(d) GBRT



(e) MLP



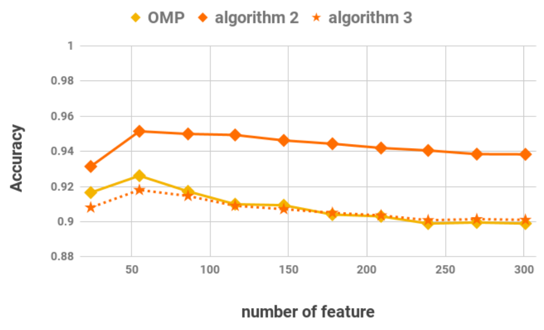
(f) KNN

Figure 6-14: Proposed method turns out to be a better feature selection scheme than p-value based one, regardless of the downstream learning algorithm. The plots are generated for $d = 24$, $\alpha = 2$, $N = 1000$, and a product $p(\mathbf{x})$.

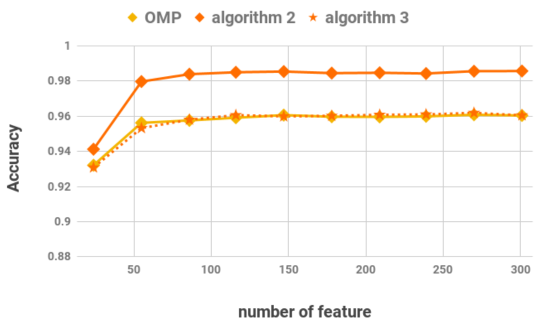
For non-tree-based methods (e.g. the SVM variants, Multi-Layer Perceptron and k-

Nearest Neighbours), the observations are fairly consistent with the Logistic Regression case: all feature selection schemes offer similar performances except the p-value based one, which is clearly worse. But note that for SVM+polynomial kernel and KNN, p-value based scheme actually provides better robustness to overfitting.

If the downstream learning method is tree-based (Random Forest or Gradient-Based Random Trees here), the proposed method demonstrates a significant improvement over all the feature selection schemes we considered. It turns out that the improvement is achieved by Algorithm 2 while the performance of Algorithm 3 is very similar to the baseline schemes, as shown in Figure 6-15. Note that Algorithm 2 is different from the baseline feature selection schemes in two major ways: (1) it chooses from candidate features using a different criterion, i.e. based on the estimated magnitude of the corresponding coefficient; and (2) the features are $\phi_S^p(\mathbf{x})$ instead of $\mathbf{x}_S$. In contrast, Algorithm 3 shares (1) but sticks to using the $\mathbf{x}_S$ features. Thus it is reasonable to conclude that the improvement comes from the use of the features $\phi_S^p(\mathbf{x})$.



(a) RF



(b) GBRT

Figure 6-15: For tree-based methods, Algorithm 2 is significantly better than the baseline feature selection schemes but Algorithm 3 offers similar performance to the baselines. Plots are generated assuming $d = 24$, $N = 1000$, $\alpha = 2$ and product $p(\mathbf{x})$.

Why is the improvement only observed for the tree-based methods? The non-tree-based methods considered here treat the feature values as continuous but the tree-based ones treat them in a more 'discrete' manner. As a concrete example, consider the feature corresponding to $S = \{1, 2\}$. Let x_1 and x_2 be independent and p_1, p_2 be close to 0.5 but not exactly 0.5 such that $\phi^p_S(\mathbf{x})$ takes value in $\{1.03, -0.99, -1.01, 0.97\}$. For a method like Logistic Regression, this is not very different from x_S; but for a decision tree, $\phi^p_S(\mathbf{x})$ has better resolution than x_S: the cases $x_1 = 1, x_2 = -1$ and $x_1 = -1, x_2 = 1$ can be distinguished by the former but not the latter. Moreover, when the parameters of $p(\mathbf{x})$ are estimated from data, tree-based methods tend to be more robust to estimation errors: while any error will be directly reflected in a method like Logistic Regression, in a Decision Tree, a small error that does not change the choice of split has no effect on the learning performance.

6.2.1.4 Comparison of Algorithm 2 and Algorithm 3

So far we have presented result for the 'proposed method', by which we mean the better of Algorithm 2 and Algorithm 3. This section aims to understand which variant is expected to be better under what situations, and thus should be used.

Recall that both variants select features based on the estimated Fourier coefficients, thus the same collection of subsets $\mathcal{C}$ will be chosen. The only difference lies in what features they feed into the downstream learning algorithms: $\{\phi^p_S(\mathbf{x})\}_{S\in\mathcal{C}}$ for Algorithm 2 and $\{\mathcal{X}_S(\mathbf{x})\}_{S\in\mathcal{C}}$ for Algorithm 3. These two sets of features differ in two major ways: (1) $\phi^p_S(\mathbf{x})$ can have varying amplitudes while the magnitude of $\mathcal{X}_S(\mathbf{x})$ is always 1; and (2) $\phi^p_S(\mathbf{x})$ can take many different values[10] while $\mathcal{X}_S(\mathbf{x})$ can only take one of two values (i.e. 1 or -1).

If the Low Degree Assumption is approximately satisfied, a classification problem is indeed linearly separable and a regression problem can be well approximated by a linear function, both in terms of $\{\phi^p_S(\mathbf{x})\}_{S\in\mathcal{C}}$. Thus as $N \to \infty$, we expect Algorithm 2 to be the correct thing to do. Also remember that the feature selection is performed to $\{\phi^p_S(\mathbf{x})\}_{S\in\mathcal{C}}$, not $\{\mathcal{X}_S(\mathbf{x})\}_{S\in\mathcal{C}}$. However, with finite N, Algorithm 3 can provide some more robustness. There are at least two sources for such robustness.

[10]The exact number of 'quantization levels' depends on $|S|$ as well as the number of parents of elements in S.

Firstly, depending on $p(\mathbf{x})$, the magnitudes of $\phi_S^p(\mathbf{x})$ for different subsets S may vary dramatically and this poses difficulty to the learning algorithm downstream, especially with relatively few training data available. In general, dramatically varying scales of features can be harmful because the ones with small magnitudes will essentially be neglected. This is why *standardization*, i.e. making sure all features have roughly the same mean and variance, is a common pre-processing step in machine learning.

Secondly, in practice the parameters of $p(\mathbf{x})$ are usually estimated from data. Consider a parameter $p(x_i = b|\mathbf{x}_{\pi_i})$ close to 0. Because it's small, there will be relatively few samples based on which this parameters is estimated and therefore the variance will be high. However, it will appear on the denominator in $\phi_S^p(\mathbf{x})$, and therefore a small error in the estimation of the parameter can lead to big errors in the feature magnitude. By using constant-magnitude features $\mathcal{X}_S(\mathbf{x})$, Algorithm 3 alleviates such errors: such an error can still cause a feature to be incorrectly included or discarded, but it will not affect the downstream learning algorithm as long as the feature is correctly chosen. Thus in theory we expect Algorithm 3 to dominate at small N but Algorithm 2 to take over as N increases.

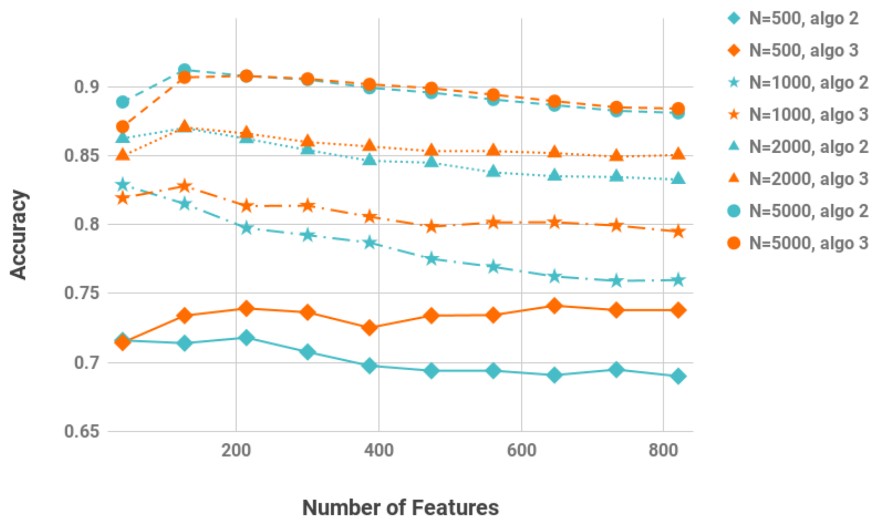


Figure 6-16: Performance of Algorithms 2 vs Algorithm 3. The plots are generated using product $p(\mathbf{x})$, $w = 4, s = 10, d = 40,\ d_0 = 2$ and $\alpha = 2$.

Figure 6-16 shows the result from Algorithms 2 and 3 on a typical experiment. The vertical axis represents accuracy and horizontal axis represents number of features, as usual. The cyan curves show results for Algorithms 2 and orange ones for Algorithm 3. Different line styles and markers indicate different choices of dataset size N.

A few observations from Figure 6-16, which are consistently found in results under other structural assumptions and choices of hyperparameters as well:

(1) As we expected, Algorithm 3 shows a clear advantage at smaller N, and the gap between the two variants shrinks as the dataset size increases.

(2) Given the same N, Algorithm 3 is better at larger number of features, probably due to its robustness to overfitting. In contrast, Algorithm 2 is better when only a very small set of features is used. This should not be very surprising because the feature selection is indeed tailored to $\phi_S^p(\mathbf{x})$.

(3) When Algorithm 3 out-performs Algorithm 2, the gap can be quite large, but when Algorithm 2 out-performs Algorithm 3, the margin is often quite small. In some cases (e.g. with more complicated structural assumptions), Algorithm 2 never takes over in the range of dataset sizes we consider, possibly due to the larger number of parameters to estimate. Figure 6-17 shows such an example. As a result, *when the downstream learning algorithm is Logistic Regression, it seems advisable to stick to Algorithm 3.*

(4) It should be emphasized the results shown in Figure 6-16 and 6-17 (and many other figures in this chapter) are averaged over many independent runs, each with freshly generated $p(\mathbf{x})$ and training data, to allow smoother curves and thus improve readability of the results. What the observation "Algorithm 3 is slightly better than Algorithm 2" on the plot usually corresponds to is there are some runs in which Algorithm 3 is better and some runs in which Algorithm 2 is better, but the former happens more often and/or by a larger margin on average. Figure 6-18 offers a rough idea of how the performance in a single run looks like before taking the average.

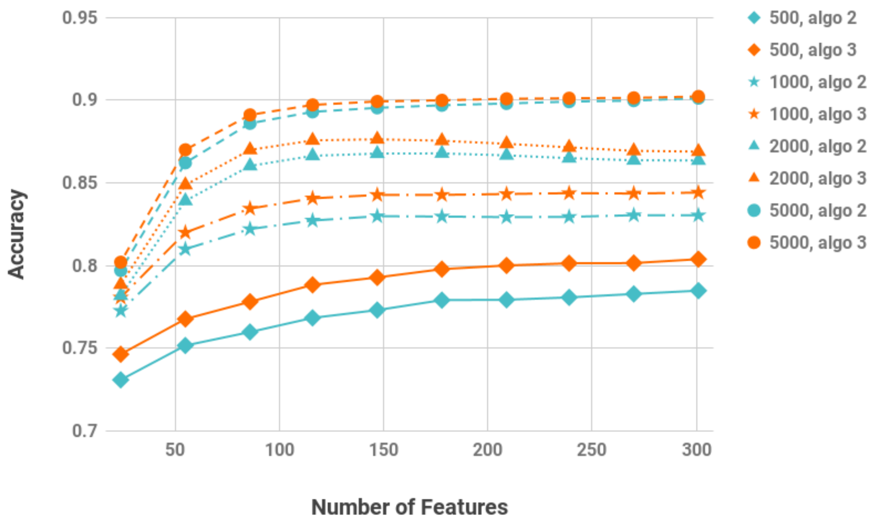


Figure 6-17: Performance of Algorithms 2 vs Algorithm 3. The plots are generated using Markov Chain $p(\mathbf{x})$, $w = 4, s = 6, d = 24$, $d_0 = 2$ and $\alpha = 2$.

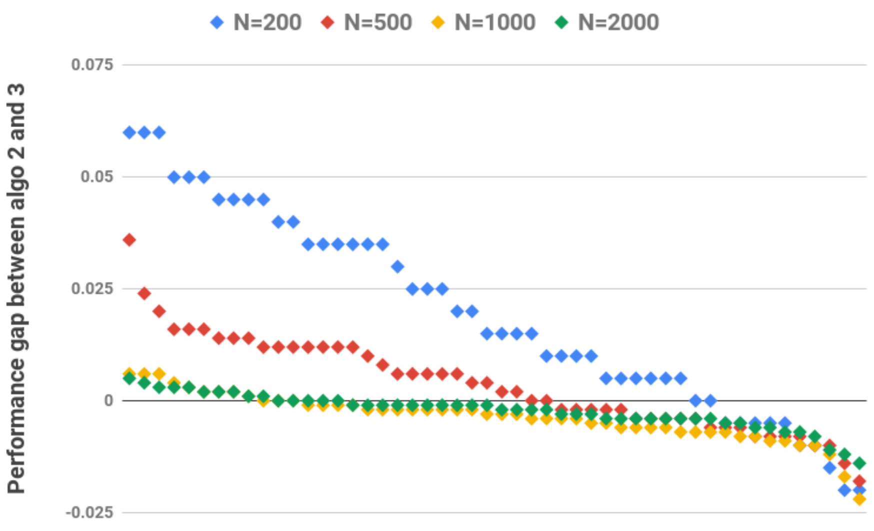
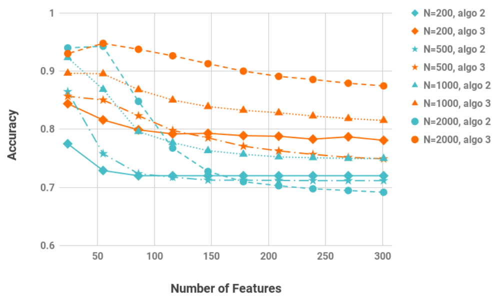


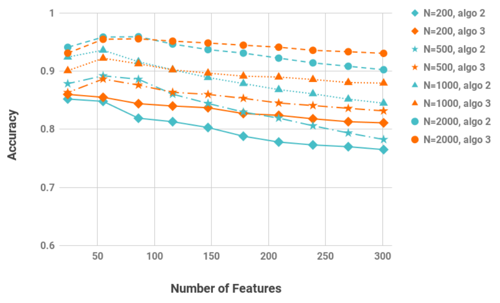
Figure 6-18: Performance comparison of Algorithms 2 vs Algorithm 3 on individual runs. The plots are generated using product $p(\mathbf{x})$, $w = 4, s = 6, d = 24$, $d_0 = 2$ and $\alpha = 5$. The y-axis represents the difference between the optimal accuracies (optimized over number of features) achieved by the two variants, and positive value indicates Algorithm 3 is better and vice versa. The horizontal axis simply represents the index of the run, and the runs are sorted to improve readability. The different colored markers show results for different dataset sizes.

From a slightly different perspective, $\phi_S^p(\mathbf{x})$ contains more information than $\mathcal{X}_S(\mathbf{x})$.

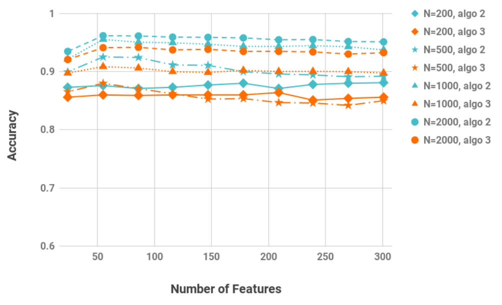
In an extreme case, the feature $x_d\sqrt{p(-x_d|\mathbf{x}_1^{d-1})/p(x_d|\mathbf{x}_1^{d-1})}$ can take on 2^d different values and there is no loss of information to train a classifier on this feature only for any classification problem – of course, most likely the resulting classifier is quite complicated. What extra information is revealed depends solely on $p(\mathbf{x})$ and has no consideration to the target function. Depending on the assumption of the form of the classifier, the extra information can be assistance or hindrance, and different downstream learning algorithms respond quite differently.



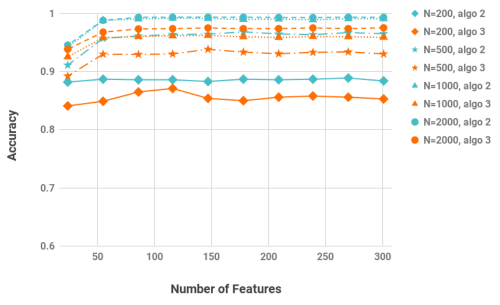
(a) SVM-poly



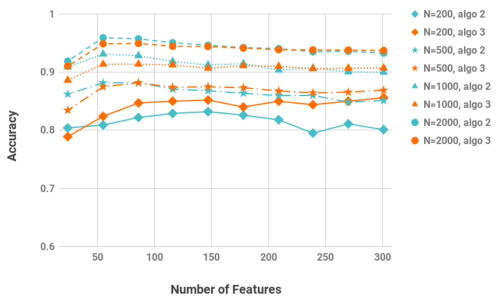
(b) SVM-RBF



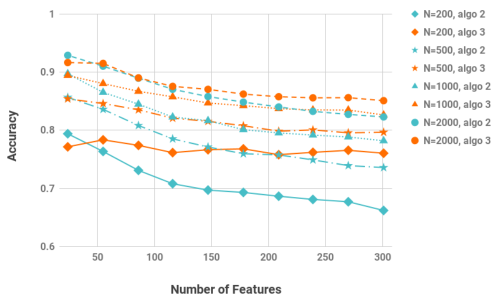
(c) Random Forest



(d) GBRT



(e) MLP



(f) KNN

Figure 6-19: Comparison of Algorithm 2 and 3 with various downstream learning methods. Plots are generated for product $p(\mathbf{x})$, $\alpha = 2$, $d = 24$ and $d_0 = 2$.

Figure 6-19 shows how the two variants compare using learning algorithms other than Logistic Regression. Some comments:

(1) Following similar discussions as in Section 6.2.1.3, the tree-based learning methods (Random Forest and GBRT) respond very well to the new features $\{\phi_S^p(\mathbf{x})\}_{S\in\mathcal{C}}$ and thus Algorithm 2 show significant advantage even at relatively small N.

(2) The non-tree based methods show similar general trends with Logistic Regression, i.e. at small N, Algorithm 3 is more robust and Algorithm 2 gradually catches up with increasing N.

(3) Algorithm 2 with polynomial kernel SVM is particularly susceptible to overfitting. This is expected because the problem of dramatically varying feature magnitudes and/or incorrectly estimated feature magnitudes deteriorates in a polynomial kernel, where powers of such features are used.

6.2.1.5 Structural Assumption

So far the proposed algorithms are assumed to have access to the correct structural information about $p(\mathbf{x})$. If such information is unavailable, there is a variance-bias trade-off in choosing the structural assumption. It turns out that *with limited amount of training data, it is not always optimal to use the correct structure.* We validate such arguments experimentally in this section.

Figure 6-20 compares the performance of Algorithm 3 assuming a product $p(\mathbf{x})$ (ground truth) vs assuming a uniform $p(\mathbf{x})$ (simplified assumption, no parameter to be estimated). The training data is actually generated from a product $p(\mathbf{x})$, with parameters i.i.d. generated from Beta(5, 5) distribution. The green curves correspond to correct structure while the yellow ones correspond to simplified ones. Horizontal and vertical axis shows number of features used and learning accuracy respectively. Different markers and line styles indicate different dataset size examined. The plot clearly shows how at very small dataset size (e.g. $N = 100$), incorrectly assuming uniform distribution actually has an advantage over attempting to learn the correct $p(\mathbf{x})$. But such advantage vanishes as N increases.

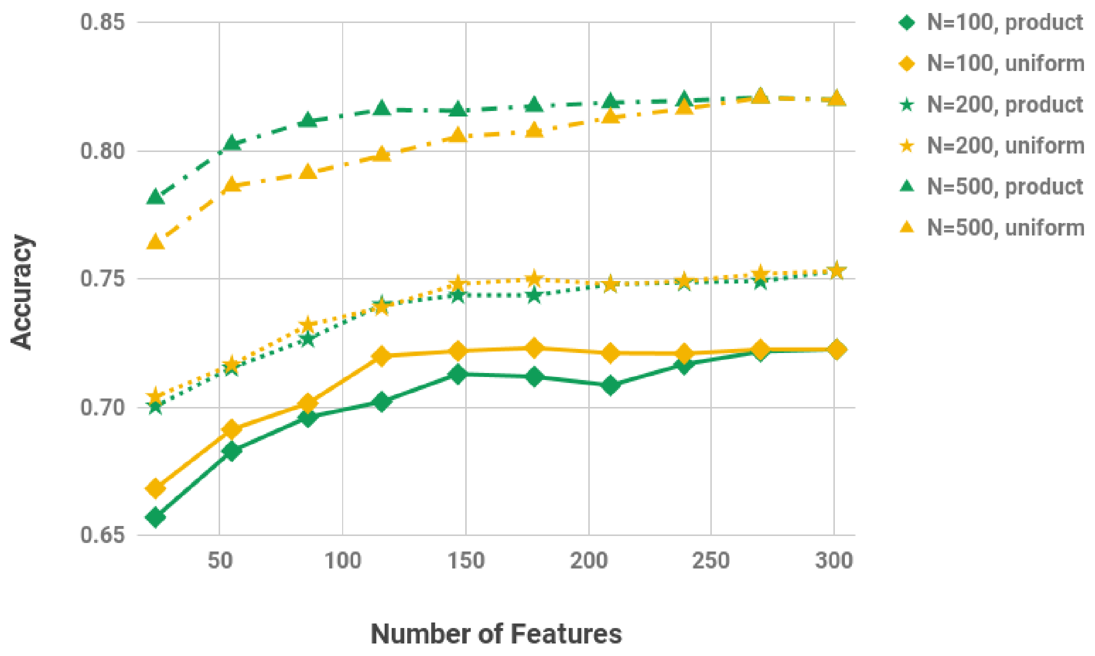
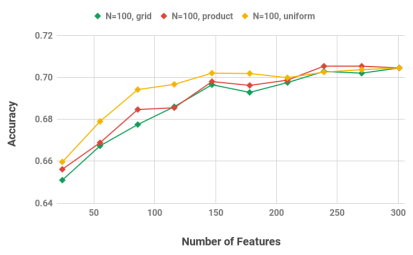


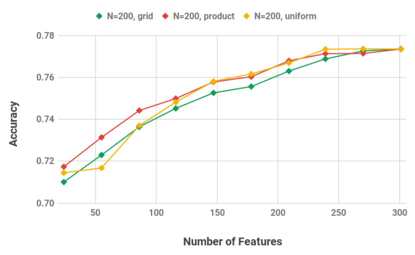
Figure 6-20: Comparison between assuming a product $p(\mathbf{x})$ (ground truth) and a uniform $p(\mathbf{x})$ (incorrect, simplified assumption). The plot is generated for $d = 24$, $\alpha = 5$, $d_0 = 2$, and various choices of N. The results reported are obtained using Algorithm 3 and the downstream learning algorithm is Logistic Regression.

Figure 6-21 shows another example of such comparison. This time a more complex ground truth structure is considered (4×6 grid), which allows us to examine multiple levels of simplified structural assumptions. In particular, we consider assuming the correct grid assumption (shown as green curves in the plots), a product $p(\mathbf{x})$ (shown as red curves) and a uniform $p(\mathbf{x})$ (shown as yellow curves).

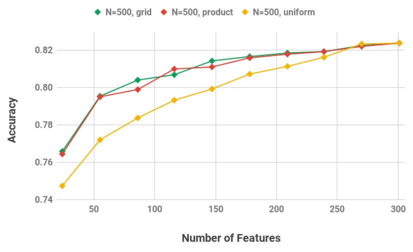
The plots in Figure 6-21 agree well with our intuition: at very small N, uniform assumption offers the best learning performance; as N increases, product assumption catches up and takes the lead; knowing the correct structure of $p(\mathbf{x})$ is useful when the dataset size eventually becomes large enough.



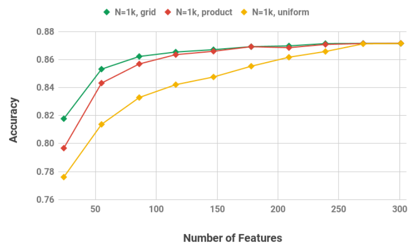
(a) N=100



(b) N=200

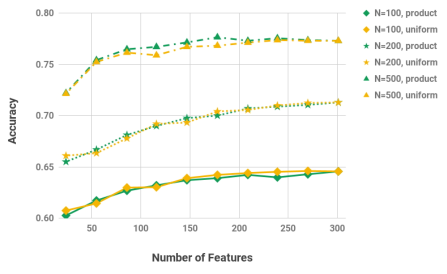


(c) N=500

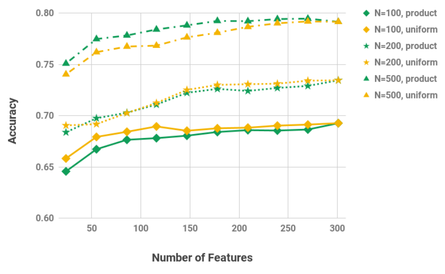


(d) N=1000

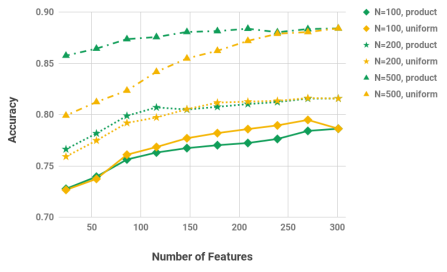
Figure 6-21: Comparison among assuming a grid $p(\mathbf{x})$ (ground truth), a product $p(\mathbf{x})$ (simplified assumption) and a uniform $p(\mathbf{x})$ (further simplified assumption). The plots are generated for $d = 24$, $\alpha = 2$, $d_0 = 2$, and various choices of N. Algorithm 3 and Logistic Regression are used.



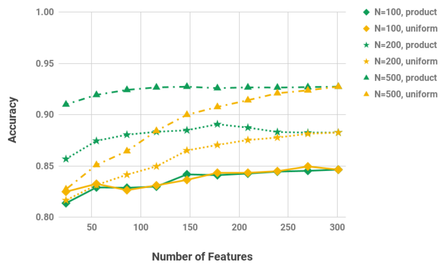
(a) $\alpha = 50$



(b) $\alpha = 10$



(c) $\alpha = 2$

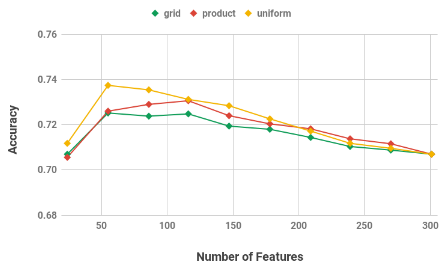


(d) $\alpha = 1$

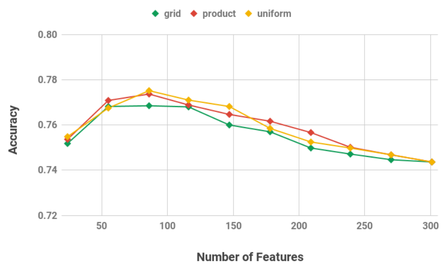
Figure 6-22: The real measure of quality of a structural assumption is whether a distribution can be found satisfying the assumption, and at the same time providing a good approximation of the low degree marginals of the true input distribution. The plots are generated for $d = 24$, $\alpha = 2$, $d_0 = 2$ and a grid $p(\mathbf{x})$.

Of course what really matters is not whether or not a certain edge is included in the structural assumption. There should be little error incurred if there is a distribution $q(\mathbf{x})$ satisfying the incorrect structural assumption but close to the true $p(\mathbf{x})$ in terms of low degree marginals. This intuition is validated by the results shown in Figure 6-22, where the true distribution has a grid structure but the simplified assumption of product distribution is made. Clearly the incorrect assumption becomes more problematic as α decreases, i.e. when the input distribution $p(\mathbf{x})$ becomes harder to approximate by a product distribution.

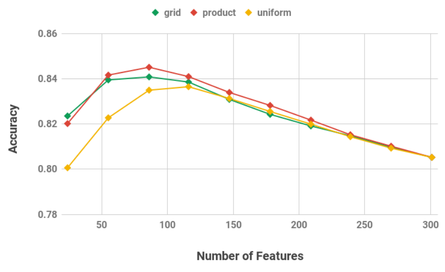
For other non-tree-based downstream learning methods (i.e. SVM variants, MLP, KNN), we observe very similar trends with Logistic Regression: simpler structural assumptions lead to better performance at small N, and more complex (and closer to ground truth) assumptions gradually catches up and takes over as N increases. As an example, Figure 6-23 shows some results for SVM with RBF kernel.



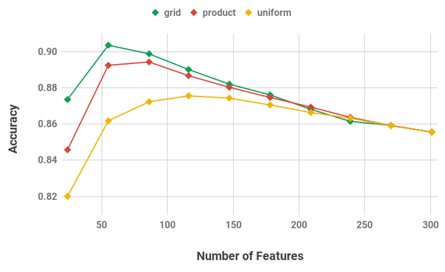
(a) N=100



(b) N=200



(c) N=500

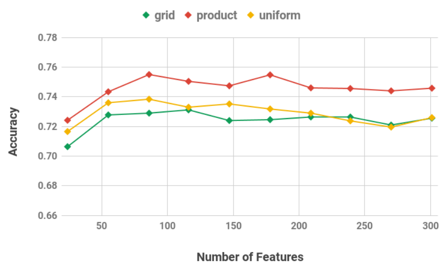


(d) N=1000

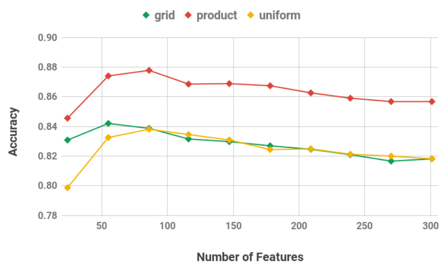
Figure 6-23: Comparison among assuming a grid $p(\mathbf{x})$ (ground truth), a product $p(\mathbf{x})$ (simplified assumption) and a uniform $p(\mathbf{x})$ (further simplified assumption). The plots are generated for $d = 24$, $\alpha = 2$, $d_0 = 2$, and various choices of N. Algorithm 3 and *SVM with RBF kernel* are used.

For tree-based methods (RF and GBRT), however, we consistently observe product assumption offers better learning performance than using the true structure for the range of dataset sizes we tested, as shown in Figure 6-24.

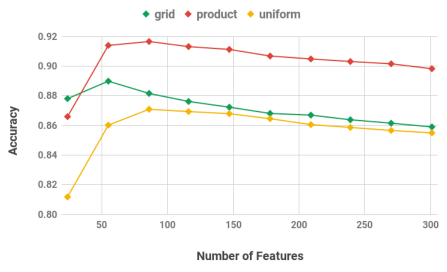
Section 6.2.1.3 discussed how learning performance of tree-based methods can be facilitated by the improved 'resolution' (i.e. more quantization levels) of the features $\phi_S^p(\mathbf{x})$. But it also lead to higher risk of overfitting. Recall each node in the tree is essentially looking for a split that can explain the incoming data points well. For each feature $\mathcal{X}_S(\mathbf{x})$, there is only one possible split since $\mathcal{X}_S(\mathbf{x})$ takes only two values. For each feature $\phi_S^p(\mathbf{x})$ that takes 2^t possible values, there are $2^t - 1$ possible cuts. Given the large number of splits to choose from, it is more likely to choose a cut that works well with the training set but hurts generalization performance. Figure 6-24 suggests that the product assumption, in which case each feature $\phi_S^p(\mathbf{x})$ takes 4 values, offer a good balance for the range of d and N we consider here.



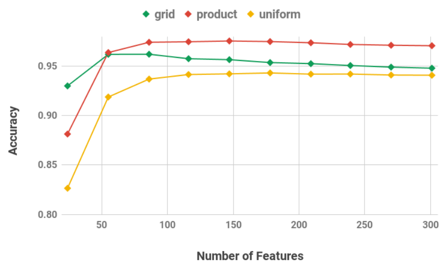
(a) N=100



(b) N=500



(c) N=1000



(d) N=5000

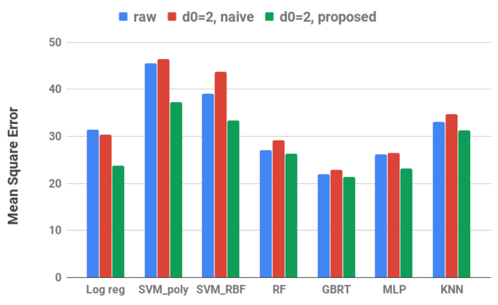
Figure 6-24: Comparison among assuming a grid $p(\mathbf{x})$ (ground truth), a product $p(\mathbf{x})$ (simplified assumption) and a uniform $p(\mathbf{x})$ (further simplified assumption). The plots are generated for $d = 24$, $\alpha = 2$, $d_0 = 2$, and various choices of N. Algorithm 2 and *Random Forest* are used.

6.2.2 Other Synthetic Datasets

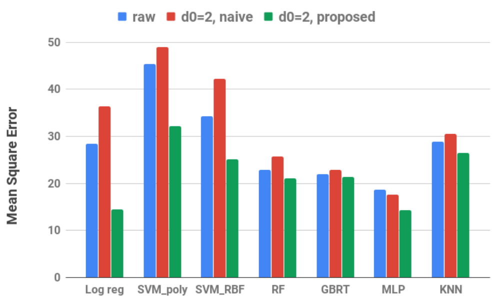
After studying the family of Tribes functions in Section 6.2.1 in great details, this section tests the proposed algorithms on several other synthetic datasets. It turns out that much of the insight gained from the case study holds quite generally, e.g. how to choose the structural assumption and decide which variant of the proposed algorithm to use depending on the dataset size and downstream learning method. We briefly describe these additional datasets here and report the corresponding results.

6.2.2.1 Low Degree Polynomial

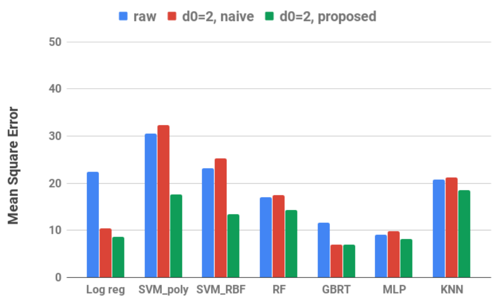
First let us consider the *regression* task described in Section 6.1.2. Figure 6-25 demonstrates the learning performance of the new features generated and chosen by the proposed methods versus that of the raw features or 'naive' features (i.e. $\mathcal{X}_S(\mathbf{x})$ for all S with $|S| \leq d_0$). The results shown are collected from a particular choice of the set of hyperparameters, but are quite representative.



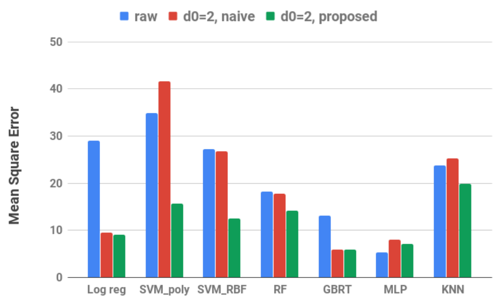
(a) N=200



(b) N=500



(c) N=1000



(d) N=2000

Figure 6-25: Polynomial functions: comparison of raw features, naive features, and new features generated and selected by the proposed methods.

In particular, the plots are generated assuming a grid-structured $p(\mathbf{x})$ with parameters generated i.i.d from Beta(2, 2) distribution, $d = 24$, $d_0 = 2$, and a range of dataset sizes ($N = 200$ to $N = 2000$) are examined. The target function is

$$f(\mathbf{x}) = \sum_{i=1}^{t} a_i \prod_{j \in S_i} x_j$$

where $t = 48$, a_i's are generated i.i.d from a standard Gaussian distribution, and each subset S_i is chosen by first drawing an integer k_i from Poisson(λ) distribution with $\lambda = 1$, and then uniformly randomly picking from all subsets of cardinality k_i.

The vertical axis in the plots of Figure 6-25 shows Mean Square Error, and thus lower bars indicates better performance. The blue, red, and green bars correspond to raw features, 'naive' features, and proposed features (i.e. features generated and chosen by the proposed methods). The results shown for the proposed features are optimized over the number of features chosen, as well as the structural assumption to use. It turns out that for the range of N considered here and this particular target function, grid structure and product structure are the best structural assumption for non-tree-based and tree-based methods, respectively. We choose the variant of the algorithm a priori based on discussion in Section 6.2.1.4: Algorithm 3 is used for non-tree-based learning methods and Algorithm 2 for the tree-based ones.

It should be clear from Figure 6-25 that (1) the proposed features provides significant improvement in learning performance, particularly for Linear Regression and SVM-based regression methods; and (2) at $N = 500$ and larger and combined with proposed features, Linear Regression becomes one of the best methods while remaining to be a simple model and easy to interpret.

Figure 6-26 compares the proposed features with features chosen among naive features by other feature selection methods, including p-value based method, Optimal Matching Pursuit, and LASSO. The plot is generated using the same hyperparameters as in

Figure 6-25 and $N = 1000$, and the results is obtained using Linear Regression as downstream learning method. A few quick comments:

- The results for Recursive Feature Elimination method are not reported here due to the very expensive computational cost of the method.

- The p-value based method is clearly worse in this experiment, and the other three offer very similar performance.

- as discussed before, for LASSO, we can only control the number of features indirectly through the regularization constant.

- the proposed method enjoys a slight advantage at relatively large model size, but is inferior to OMP at very small model size. Note that the greedy nature of OMP makes it particularly suitable if only a few features are chosen. The proposed methods, however, make independent decisions for each feature without overall consideration and this strategy can be particularly problematic if only a small number of features can be chosen.

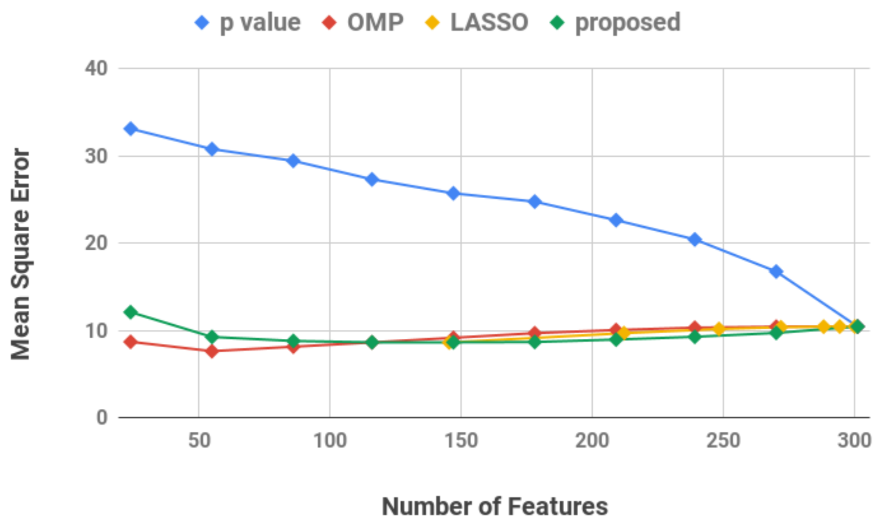


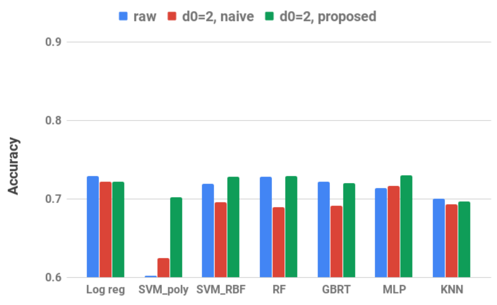
Figure 6-26: Polynomial functions: comparison of proposed features and features selected from naive features by general purpose feature selection methods. Linear Regression is used and $N = 1000$.

6.2.2.2 Polynomial Threshold Functions

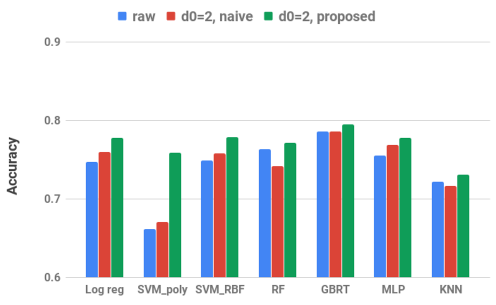
This is essentially the same target function as the Low Degree Polynomial, except we threshold the output at 0 to turn it into a *binary classification* task. More concretely,

$$f(\mathbf{x}) = \operatorname{sgn}\left(\sum_{i=1}^{t} a_i \prod_{j \in S_i} x_j\right)$$

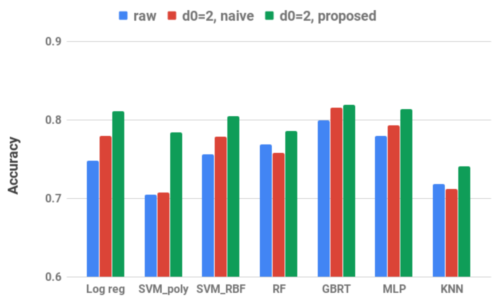
Figure 6-27 compares the performance of raw features, naive features and proposed features on Polynomial Threshold Functions with Markov Chain structured $p(\mathbf{x})$ whose parameters are i.i.d generated from a Beta(2, 2) distribution, and the target function is generated with $d = 30$, $t = 60$ and $\lambda = 1$. Note that the horizontal axis shows accuracy here and thus higher bars indicate better performance.



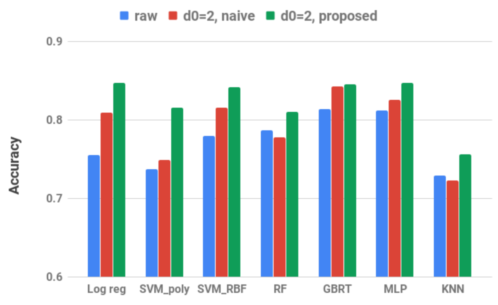
(a) N=200



(b) N=500



(c) N=1000



(d) N=2000

Figure 6-27: Polynomial Threshold Functions: comparison of raw features, naive features, and new features generated and selected by the proposed methods.

The moral of the story is very similar to the previous datasets considered: (1) the proposed features start to offer significant improvement when dataset size is reasonably large (in this case $N = 1000$ or larger); (2) Logistic Regression, when trained on the proposed features, can achieve similar or even better performance than more complicated methods such as SVM, GBRT or MLP.

Figure 6-28 makes the comparison of the proposed features with other feature selection methods on the Polynomial Threshold Functions, and the results are quite consistent with those from Low Degree Polynomials: (1) the p-value based method is clearly worse than the others (although the gap seems to be less dramatic); (2) OMP is the clear winner when the number of features is very small.

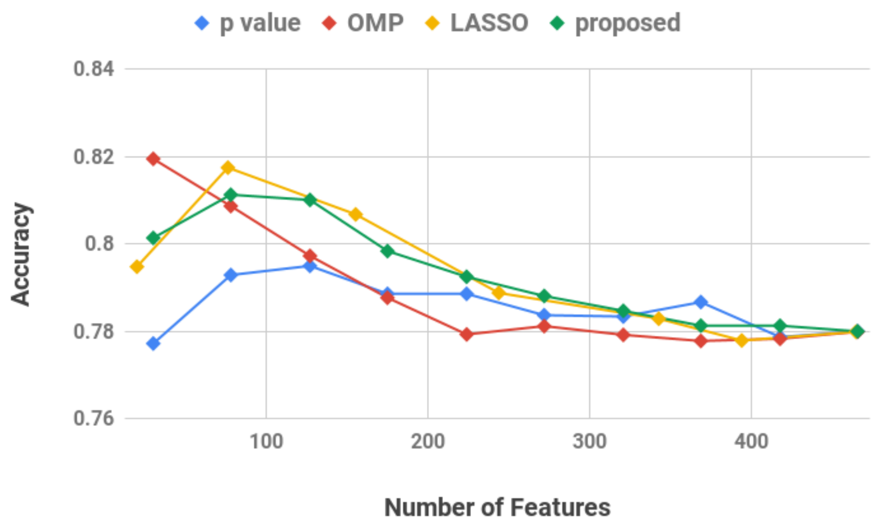


Figure 6-28: Polynomial Threshold Functions: comparison of proposed features and features selected from naive features by general purpose feature selection methods. Logistic Regression is used and $N = 1000$.

6.2.2.3 Tic Tac Toe

The last synthetic dataset we examined is a *multi-class classification* task adapted from the simple game Tic Tac Toe. In this game, two players take turns drawing circles (player A) or crosses (player B) into the empty spots in a 3×3 grid. The player first filled a whole row/column/diagonal claims a win.

We use two binary features representing each square: first bit indicating whether the square is filled, and if it is, the second indicating what shape is in the square. A

3×3 grid with any number of squares filled with any shape is considered legitimate input, and there are three possible output values:

- output 1 if Player A has filled a row/column/diagonal and Player B has not
- output -1 if Player B has filled a row/column/diagonal and Player A has not
- output 0 if both players have filled a row/column/diagonal or neither has, in other words, this is not a legitimate endgame

Note that we have not imposed constraints on the numbers of each shape, which can be interpreted as a player can choose to skip a round. This may or may not make sense in the game, but for our purpose, this is still a well defined Boolean function we can generate training samples and learn from.

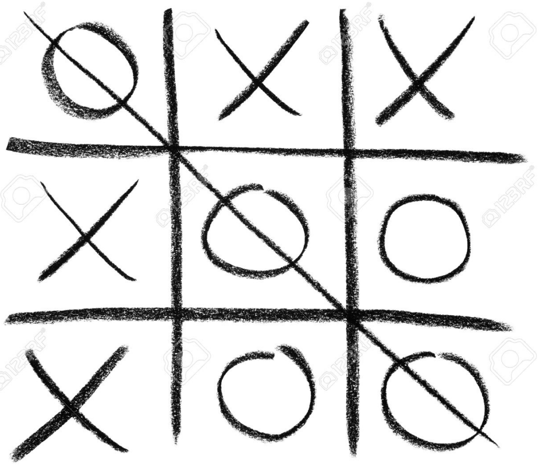
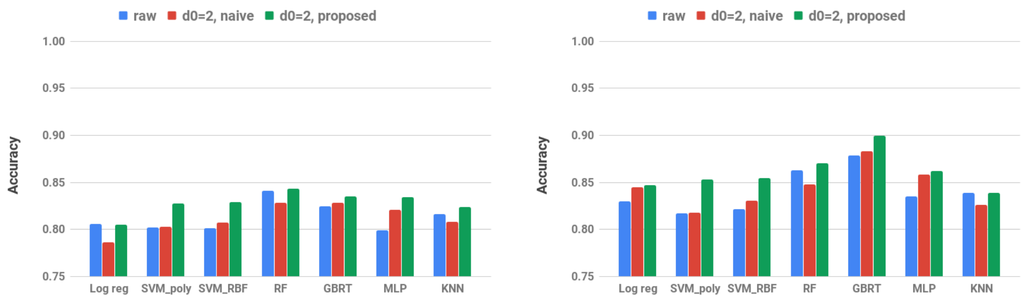


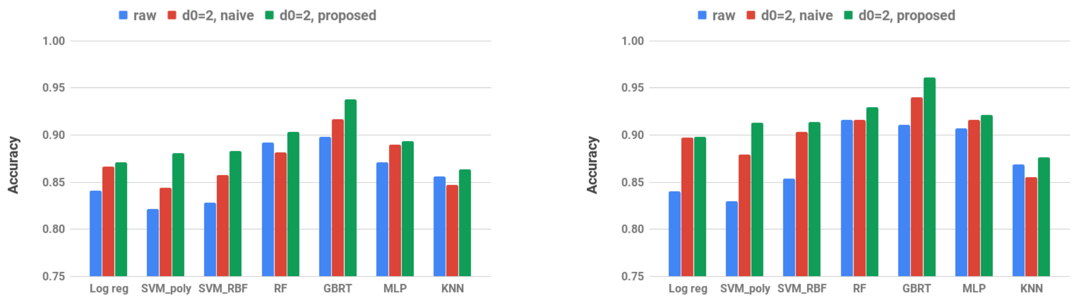
Figure 6-29: An example end-game of Tic Tac Toe, in which case the circle side wins.

The performance of raw features, naive features and proposed features are compared in Figure 6-30. The choice of hyperparameters for the plots are: Hidden Markov Model structured $p(\mathbf{x})$, with parameters i.i.d. generated from Beta(2, 2) distribution; $d = 18$, $d_0 = 2$, and $N = 200, 500, 1000, 2000$.

Again, we observe improvement in prediction accuracy after replacing the raw features by proposed features, and the improvement is more significant as N increases.



(a) N=200 (b) N=500



(c) N=1000 (d) N=2000

Figure 6-30: Tic tac toe function: comparison of raw features, naive features, and new features generated and selected by the proposed methods.

Figure 6-31 compares the various feature selection methods for the Tic Tac Toe function. In this case, the performance are actually fairly similar to each other.

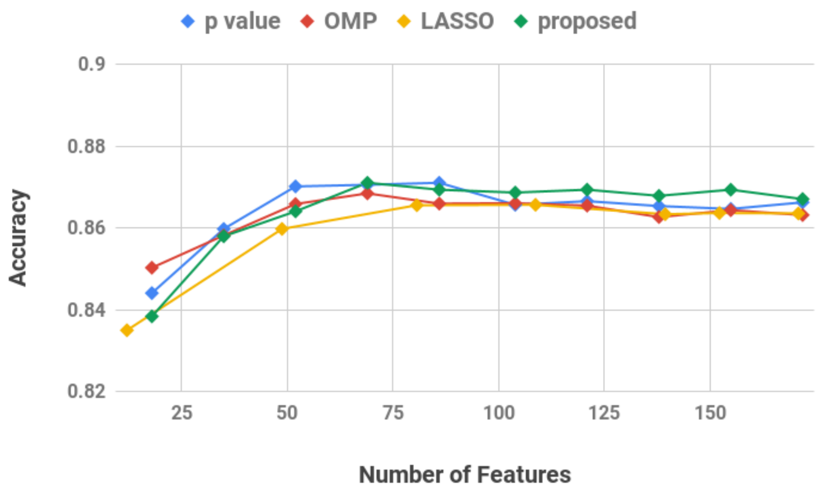


Figure 6-31: Polynomial Threshold Functions: comparison of proposed features and features selected from naive features by general purpose feature selection methods. Logistic Regression is used and $N = 1000$.

6.3 Real Data

The proposed algorithms are also tested on real datasets to validate its usefulness in practice. With synthetic datasets, since we get to choose $f(\mathbf{x})$, $p(\mathbf{x})$, d and N, the emphasis is on understanding how the learning performance responds to changes in assumptions and/or hyperparameters. With real datasets, however, we have much less flexibility about the dataset itself and will focus more on how to optimize the hyper-parameters, including structural assumptions on $p(\mathbf{x})$, choice of d_0 and number of features to use etc. Similar to the synthetic data case, we will treat one problem (MNIST dataset) in greater details in Section 6.3.1 and then summarized results for some other real datasets are briefly presented in Section 6.3.2.

6.3.1 MNIST Digit Recognition Dataset

The MNIST dataset is a database of 70000 (60000 training + 10000 testing) images of handwritten digits $(0, 1, \cdots, 9)$, each contains 28×28 pixels. Each pixel takes value from 256 grey scale levels. The goal is to do *multi-class classification*, i.e. to predict which digit it is given the image. It is probably one of the most widely used datasets in the machine learning community today.

We use the MNIST dataset in the standard way, except for two small changes. Firstly, while the original dataset contains a fixed training set and testing set, we randomize the ordering of the samples and use cross validation. The number of folds is set to 7 so that we still have 60000 training samples. Secondly, while the original pixels take values in [0, 255], we quantized each pixel to 2 levels and represent each pixel by one binary variable. After the quantization step, some pixel values will be (almost) constant in all the images in the dataset. Those pixels are removed and we ended up with $d = 557$ rather than $28 \times 28 = 784$. In practice, we find this pre-processing step causes little loss of information but significantly reduces computational cost.

6.3.1.1 Structural Assumptions on Input Distribution

One major decision to make is what structural assumption to choose for the input distribution $p(\mathbf{x})$. Based on considerations on computational feasibility and our understanding of the dataset, we propose the following structures:

- *Product Structure*, i.e. $p(\mathbf{x}) = \prod_{i=1}^{d} p(x_i)$
- *Tree Structure*, i.e. $p(\mathbf{x}) = \prod_{i=1}^{d} p(x_i|\mathbf{x}_{\pi_i})$ where $|\pi_i| \leq 1, \forall i$. Under this assumption, we can efficiently learn an optimal tree/forest from training data using Chow-Liu algorithm.
- *Grid Structure*, each pixel takes the pixel above it and the pixel to the left (if that exists) as its parents. Here we've used the knowledge that the input data form an image and essentially making smoothness assumption.

Given the structural assumption, the corresponding parameters are estimated from training data. We present the performance of Algorithms 2 and Algorithm 3 under the different structural assumptions in Figure 6-32. *Multinomial Logistic Regression* is used as the downstream classifier.

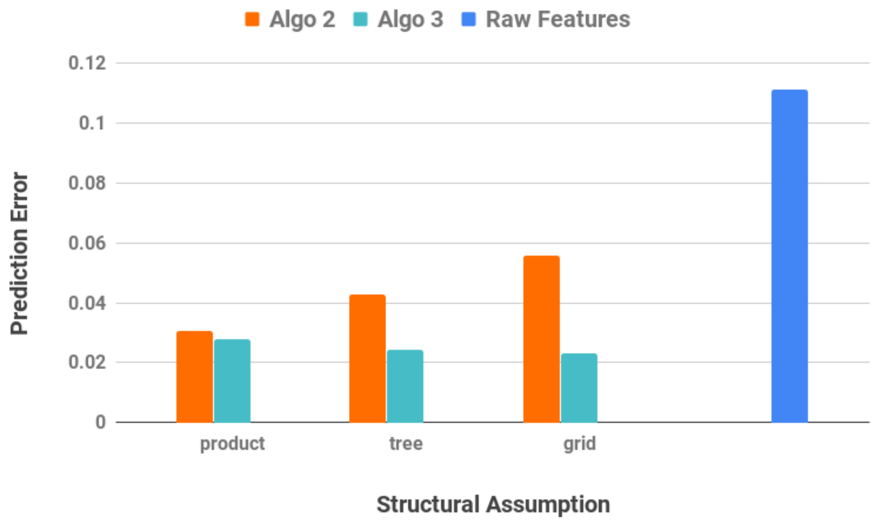


Figure 6-32: Performance of Algorithms 2 and Algorithm 3 on the MNIST dataset under different structural assumptions. The results in this plot uses $d_0 = 2$ and the number of features used is 50000.

The vertical axis represents prediction error, thus lower bars correspond to better performance. The orange bars show the performance of Algorithm 2, i.e. using the 'correct features' $\{\phi_S^p(\mathbf{x})\}_{S\in\mathcal{C}}$, and cyan bars show the performance of Algorithm 3 where the more robust $\{\mathcal{X}_S(\mathbf{x})\}_{S\in\mathcal{C}}$ are used. But both algorithms share the choice of the collection of subsets $\mathcal{C}$. As a baseline, we show the prediction error of Multinomial Logistic Regression trained on the raw features by the blue bar.

Note that the three structures, i.e. *product*, *tree* and *grid*, are in increasing order of complexity and probably increasingly a better approximation of the actual $p(\mathbf{x})$. Performance of Algorithm 3 is improving with more complex structure, but the performance of Algorithm 2 demonstrates an opposite trend. We believe the latter observation is the result of having relatively 'biased' parameters (some values of $p(x_i|\mathbf{x}_{\pi_i})$ are estimated to be very close to 1), which in turn leads to dramatically varying feature magnitudes in Algorithm 2 that is harmful to the learning performance. Regardless of the structural assumption, the features chosen by the proposed methods provides significant improvement over the raw features when the downstream learning algorithm is Multinomial Logistic Regression.

6.3.1.2 Number of Features

Figure 6-33 shows how the prediction error changes as we increase K, the number of features chosen to feed into the downstream classifier. Grid structure is assumed and Algorithm 3 is used, which, according to Figure 6-32, is the 'optimal' combination on MNIST dataset. Multinomial Logistic Regression is used as the downstream classifier.

The performance improves with increasing number of features at smaller values of K and the performance flattens around $K \sim$ 40,000 to 45,000. Recall the training dataset has $N = 60000$ images, thus this is roughly in agreement with the intuition $|\mathcal{C}| = O(N)$ discussed in Section 5.2. All the results in this section used $d_0 = 2$ and it offers significant improvement over $d_0 = 1$. $d_0 = 3$ is also tested but clearly suboptimal: it not only dramatically increases computational cost, but also has little or even negative effects in prediction accuracy due to widely varying feature magnitude caused by parameters close to 0. Since $\phi^p_S(\mathbf{x}) = \prod_{i\in S} \phi^p_{\{i\}}(\mathbf{x})$, the magnitude problem deteriorates as d_0 increases.

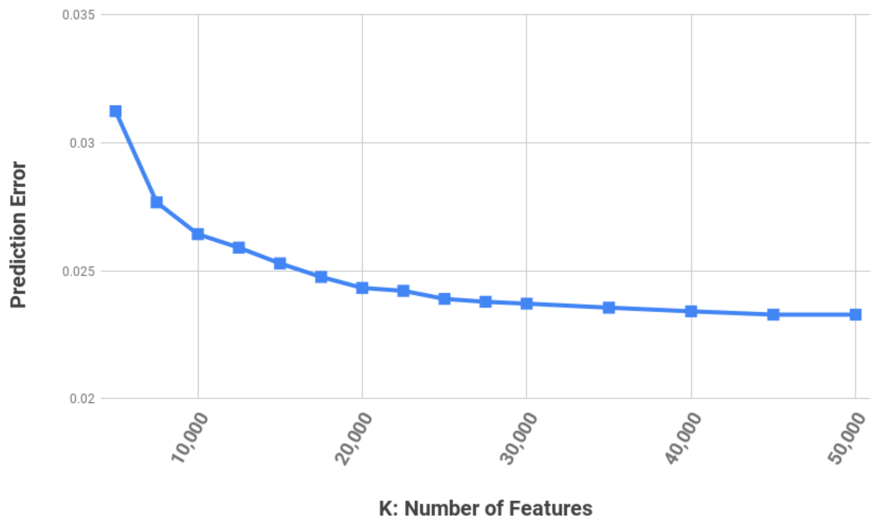


Figure 6-33: Performance of Algorithm 3 under grid structured $p(\mathbf{x})$ as a function of number of features used in the Multinomial Logistic Regression. $d_0 = 2$.

6.3.1.3 Combined with Other Classification Methods

So far Multinomial Logistic Regression is used as the downstream learning method due to their desired property of simplicity and good interpretability. Of course other classification methods could be used. Figure 6-34 demonstrates the learning performance on a few other classifiers. In particular, we consider Random Forest and Multi-Layer Perceptron. The other downstream methods are not considered here either due to inferior performance (e.g. KNN) or much higher computational costs (e.g. the SVM variants).

The vertical axis again represents the prediction error. Blue bars show performance on raw features and green bars show that of the features chosen by the optimal variant of the proposed methods: Algorithm 3 with grid structural assumption on $p(\mathbf{x})$ for Logistic Regression and MLP, and Algorithm 2 with product structural assumption for Random Forest. Note that Random Forest doesn't seem to benefit from having access to the new features at all but Logistic Regression improved dramatically after considering the second order terms. Also note that the relative simple Logistic Regression method with the new features outperforms more powerful methods such as Random Forest and MLP with raw features by $\sim 1\%$ (i.e. from $\sim 3.3\%$ to $\sim 2.3\%$).

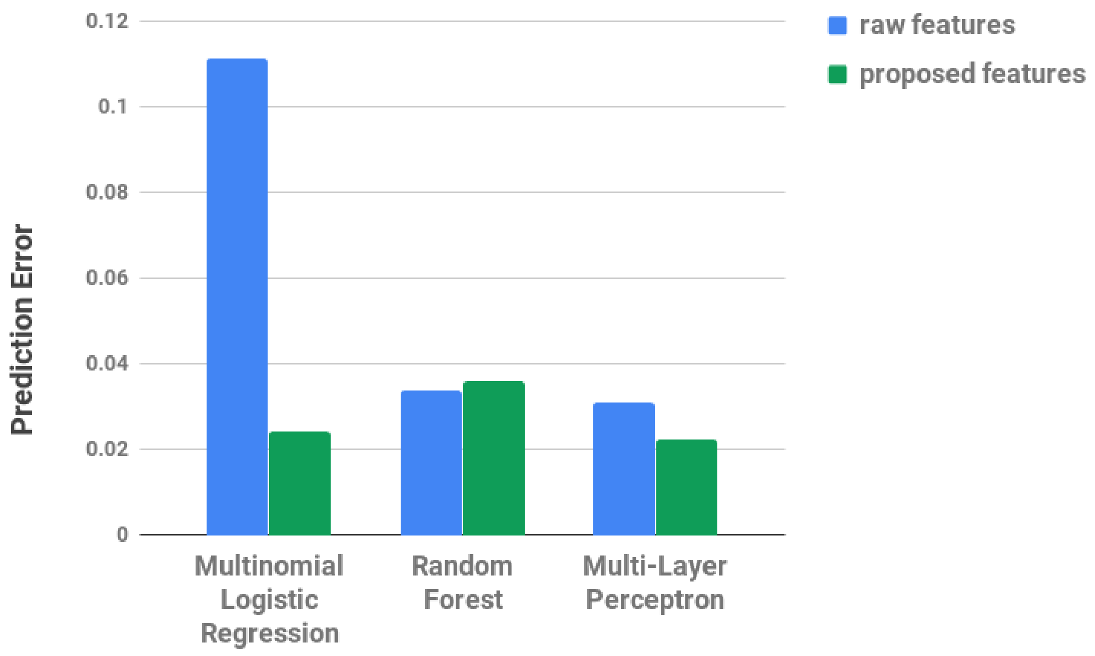


Figure 6-34: Using different downstream learning methods.

6.3.1.4 Comparison to Baselines

MNIST is such a well-studied problem that results of numerous algorithms applied to the dataset have been reported. In this part we present some of these results to get a sense how well our proposed method is doing[11].

In the deep learning community, MNIST is considered easy and basically a solved problem. For example, Ciresan et al. [2012] achieved an error of 0.23% using a committee of 35 convolutional neural networks (CNNs). Various other architectures of CNNs are reported to achieve $0.27\% \sim 1.7\%$.

[11]The results are quoted from Yann Lecun's website: http://yann.lecun.com/exdb/mnist/

Recall that the lowest error we achieved with the proposed methods using Multinomial Logistic Regression is 2.3%. This is worse than even the more primitive forms of CNNs. But this is not too surprising as CNNs (or committees of CNNs) are expected to be much more powerful than the proposed methods. We achieved 2.3% error with essentially a **linear classifier** where **each feature is human readable: either the (quantized) value of a pixel, or the XOR of two pixels**.

Tables 6.2, 6.3, 6.4 and 6.5 present the reported results for families of non-deep-learning-based methods. Generally speaking, *the proposed methods outperform those methods unless some fancy tricks or fancy pre-processing steps are implemented.* Most of such tricks/pre-processing can be viewed as feeding prior knowledge into the learning algorithm. For the proposed method, the only such information we utilized is the grid structure.

One exception is SVM, where even with the generic RBF kernel and no pre-processing, an error of 1.4% can be achieved. However, the computational cost of SVM is significantly higher ($O(dN^2)$ to $O(dN^3)$ depending on the dataset). The computational complexity of the proposed method is $O(d^2N)$, and with parallelization, $O(N)$ can be achieved. Here $d = 557$ and $N = 60000$.

Method	Pre-Processing	Error (%)
2-layer NN, 300-1000 hidden units, with or without distortion	none	3.6 - 4.7
2-layer NN	de-skewing	1.6
3-layer NN, varying # of hidden units	none	2.45 - 3.05
2 or 3 layer NN, cross entropy loss	none	0.7 - 1.53
fancy NN methods (more layers, elastic distortions, committee of NNs, pre-training)	none	0.35 - 1.0

Table 6.2: Prediction Error reported on MNIST using Neural Networks (not deep)

Method	Pre-Processing	Error (%)
KNN, Euclidean distance (L2)	none	5.0
KNN, L3 distance	none	2.83
KNN, L2 distance	de-skewing	2.4
KNN, L3 distance	de-skewing, noise removal, blurring, pixel shift	1.22 - 1.73
fancy KNN methods (non-linear deformation, tangent distance, shape context matching)	none	0.52 - 1.1

Table 6.3: Prediction Error reported on MNIST using K Nearest Neighbours

Method	Pre-Processing	Error (%)
Linear Classifier	none	12.0
Linear Classifier	de-skewing	8.4
Pairwise Linear Classifier	de-skewing	7.6
40 PCA + Quadratic Classifier	none	3.3
1000 RBF + Linear Classifier	none	3.6

Table 6.4: Prediction Error reported on MNIST using linear or quadratic classifier combined with generic feature selection methods.

Method	Pre-Processing	Error (%)
SVM + RBF Kernel	none	1.4
SVM + Degree 4 Polynomial Kernel	de-skewing	1.1
Virtual SVM, Degree 9 Polynomial Kernel	none	0.8
Virtual SVM, Degree 9 Polynomial Kernel, 1 or 2 pixels jittered	de-skewing	0.56 - 0.68

Table 6.5: Prediction Error reported on MNIST using Support Vector Machines

6.3.2 Other Real Datasets

Several other real datasets from UCI Machine Learning Repository (Dua and Taniskidou [2017]) are examined and we report a summary of the results here.

6.3.2.1 Hepatitis Dataset

The Hepatitis Dataset[12] is a classification task containing 155 data entries, each representing a Hepatitis patient. There are 19 input features (including age, gender, and various medical symptoms), and the the output is a binary label: LIVE or DIE. Some features are binary-valued while others take continuous values. But the continuous-valued features are already grouped into brackets in the dataset.

To apply the proposed methods, we use one binary variable to describe each binary-valued feature, and one-hot encode the others, which result in $d = 49$. Due to the very small dataset size, we use $d_0 = 1$ and assume a product $p(\mathbf{x})$. As downstream learning methods, Logistic Regression, Random Forest, and Multi-Layer Perceptron are considered. Note that with $d_0 = 1$, we do not introduce 'new information' via interaction terms, yet it turns out the appropriate feature selection provided by the proposed methods lead to better learning performance.

Feature Type	Log Reg	RF	MLP	LazyDT
Raw Features	85.00 ± 1.44	84.84 ± 0.85	81.55 ± 1.40	85.87 ± 1.10
Proposed Features	86.07 ± 1.19	85.47 ± 1.20	83.60 ± 1.72	N/A

Table 6.6: Results on Hepatitis Dataset. The numbers shown are accuracy in percentage.

To the best of our knowledge, the best results reported on this dataset is from Fern and Brodley [2003], reporting an accuracy of 85.87 ± 1.10 by the Lazy Decision Tree algorithm. To ensure fair comparison, we follow the practice in Fern and Brodley [2003] by using 10-fold cross validation and report the results on 10 independent runs,

[12]https://archive.ics.uci.edu/ml/datasets/Hepatitis

each randomly permuting the data. Table 6.6 summarizes the accuracy achieved by different combinations of feature type and learning algorithm. The best performance is provided by a Logistic Regression classifier trained on about 25 features chosen by Algorithm 3. Figure compares the performance of Algorithm 3 with other feature selection techniques as the number of features varies, assuming the downstream classifier is Logistic Regression.

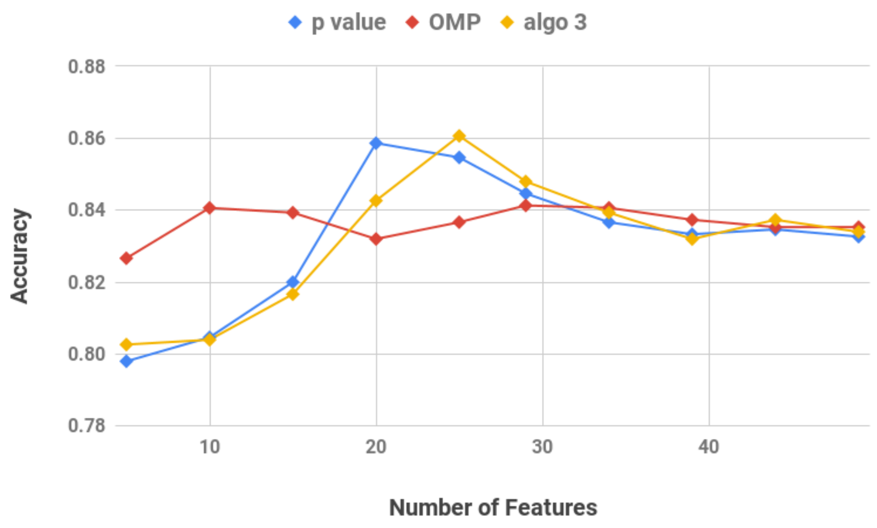


Figure 6-35: On the Hepatitis dataset, p-value based feature selection and Algorithm 3 has similar performance. Orthonormal Matching Pursuit is preferred at small model size but its best performance is not as good as the other two.

6.3.2.2 Congressional Voting Records Dataset

The Congressional Vote Dataset[13] has been described in Section 6.1.3. Recall that $N = 435$, $d = 32$ (we use 2 bits to represent each of the 16 key issues voted on due to existence of missing values in the dataset). Thus $d_0 = 2$ is used. As for the structural assumption, we put an edge between each pair of binary variables that correspond to the same issue, and assume all pairs are independent. The learning performance is summarized in Table 6.7, and we compare the proposed methods with other feature selection techniques in Figure 6-36. 10-fold cross validation is used, and the results shown are averaged over 10 independent runs, each time randomly permuting the

[13]https://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

entries.

Feature Type	Log Reg	RF	MLP
Raw Features	95.88 ± 0.41	95.90 ± 0.17	94.60 ± 0.60
Proposed Features	96.23 ± 0.3	96.34 ± 0.24	95.27 ± 0.61

Table 6.7: Results on Congressional Voting Records Dataset. The numbers shown are accuracy in percentage.

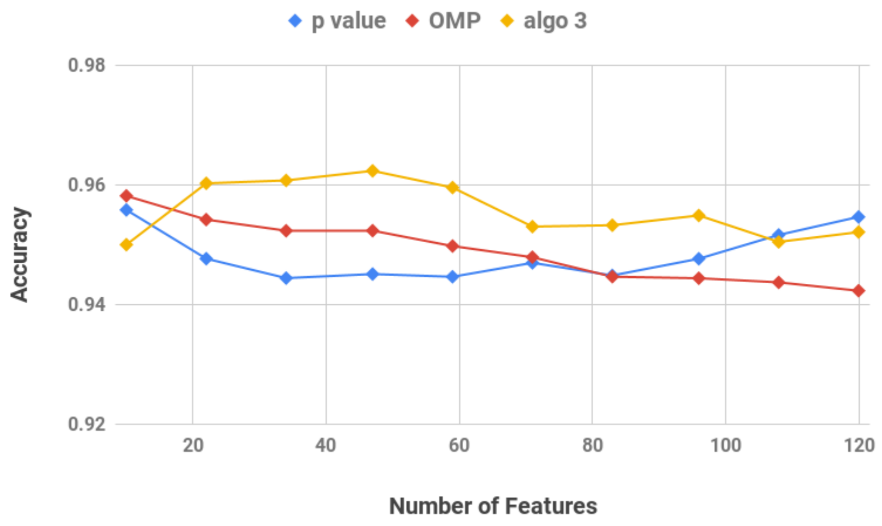


Figure 6-36: On the Congressional Voting Records dataset, Algorithm 3 enjoys a clear advantage over the baseline feature selection methods when the number of features is relatively small. At very small model size (K<10), Algorithm 3 is worse than the baselines, but this is expected.

6.3.2.3 Tic Tac Toe Endgame Dataset

The Tic Tac Toe Endgame dataset[14] considers the game discussed in Section 6.2.2.3, but unlike the synthetic version, the entries in this dataset are all legitimate endgame configurations. Another difference is that here we have no control over $p(\mathbf{x})$.

This dataset has $N = 958$ entries. Each of the 9 square can take 3 different values (empty, circle, or cross) and we again use 2 bits to represent each square, thus

[14]https://archive.ics.uci.edu/ml/datasets/Tic-Tac-Toe+Endgame

$d = 18$. Similar to the Congressional Voting Records Dataset, we make the structural assumption that each pair of binary features related to the same square are dependent, but all pairs are independent. Given the values of d and N, we tested both $d_0 = 2$ and $d_0 = 3$. As shown by Table 6.8 and Table 6.9, it turns out $d_0 = 3$ is the right choice and it offers near perfect learning performance. This should be expected, since the rules determining the result of the game each involves 3 squares.

Feature Type	Log Reg	RF	MLP
Raw Features	81.78 ± 0.36	91.81 ± 0.34	96.77 ± 0.12
Proposed Features	99.00 ± 0.27	96.34 ± 0.24	96.10 ± 0.59

Table 6.8: Results on Tic Tac Toe Endgame Dataset, assuming $d_0 = 2$.

Feature Type	Log Reg	RF	MLP
Raw Features	81.78 ± 0.36	91.81 ± 0.34	96.77 ± 0.12
Proposed Features	100 ± 0.00	99.52 ± 0.20	99.50 ± 0.09

Table 6.9: Results on Tic Tac Toe Endgame Dataset, assuming $d_0 = 3$.

As a baseline, the same dataset is considered in Esmeir and Markovitch [2004] using variants of Decision Trees. The reported best accuracy among the variants is 87.7 ± 3.2. Bain [2002] tackled the dataset with several variants of rule-based systems and one of the variants also achieves perfect performance – not particularly surprising as the target function in this dataset is indeed governed by well-defined and relatively simple rules. Generally speaking, rule-based systems are not very scalable and do not work well if the target function is complicated or noisy. In contrast, we are able to achieve perfect performance with the proposed methods + Logistic Regression, which is scalable and robust to noise.

Figure 6-37 compares the performance of various feature selection methods. In this case, when the number of features is relatively large, all the feature selection methods are doing fairly. The performance of p-value based method starts to deteriorate earlier than the others as number of features decreases.

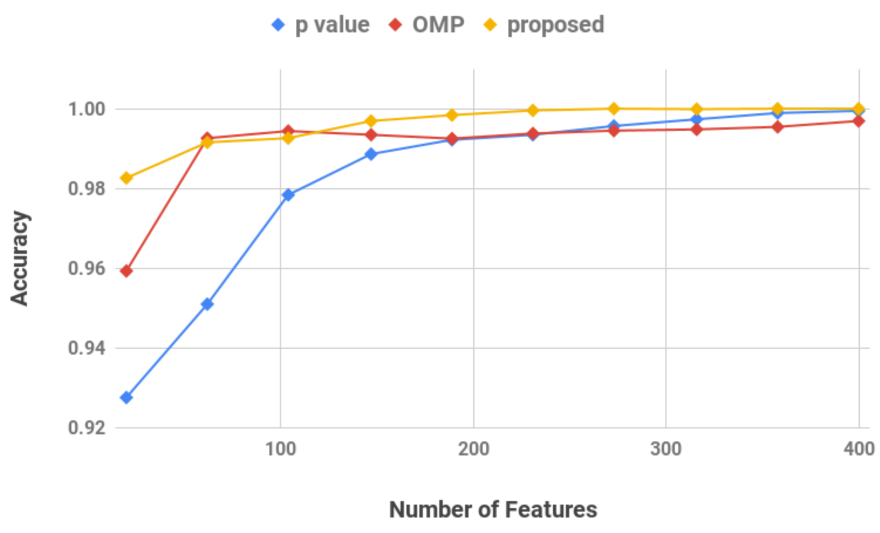


Figure 6-37: Comparison of the feature selection methods on the Tic Tac Toe Endgame dataset. $d_0 = 3$ and Algorithm 3 are used. The downstream learning method is Logistic Regression.

6.3.2.4 Chess (King-Rook vs King-Pawn) Dataset

The Chess (King-Rook vs King-Pawn) dataset[15] contains entries that describe chess board configurations where the white side has a King and a Pawn while the black side has a King and a Rook. The pawn is on a7 (i.e. one square away from queening) and it is the black side's turn to move. There are 36 features describing various attributes of the configurations, e.g. whether the Black King can attack the White Pawn, whether the White King is in stalemate etc.[16]. The output is a binary variable indicating whether the white side can win.

Among the 36 features, only one of them takes 3 possible values, which we one-hot encode into 2 bits, and the other 35 features are binary, thus $d = 37$. There are a total of $N = 3196$ entries and we consider to use $d_0 = 2$. As for the structural assumption, we simply assume $p(\mathbf{x})$ is a product distribution.

[15]https://archive.ics.uci.edu/ml/datasets/Chess+%28King-Rook+vs.+King-Pawn%29
[16]A complete description of the features can be found in Shapiro [1987].

Table 6.10 summarizes the learning performance on raw and proposed features, while Figure 6-38 compares the various feature selection methods. This is another classic case where Logistic Regression is not as powerful as the more complicated methods using the raw features, but catches up with the proposed features.

Feature Type	Log Reg	RF	MLP
Raw Features	96.12 ± 0.13	99.15 ± 0.09	99.20 ± 0.04
Proposed Features	99.61 ± 0.07	99.54 ± 0.05	99.32 ± 0.08

Table 6.10: Results on Chess (King-Rook vs King-Pawn) Dataset, assuming $d_0 = 2$.

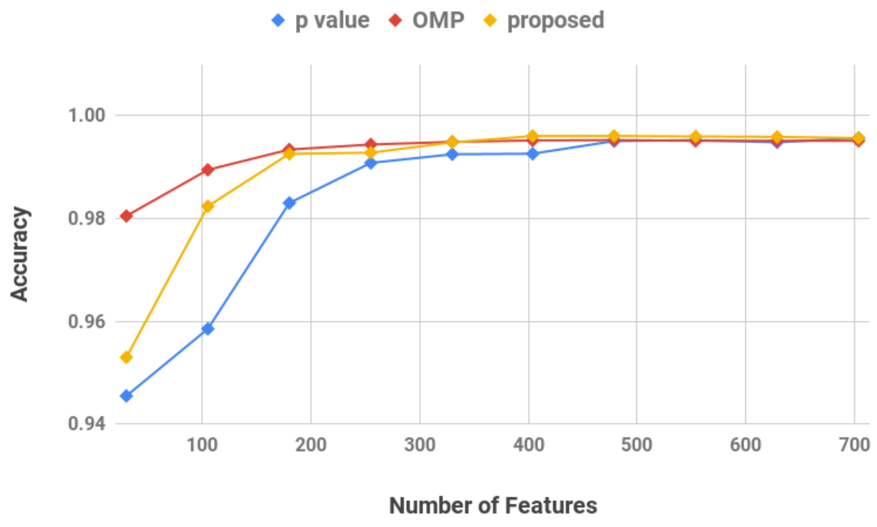


Figure 6-38: Comparison of the feature selection methods on the Chess (King-Rook vs King-Pawn) dataset. $d_0 = 2$ and Algorithm 3 is used. The downstream learning method is Logistic Regression.

6.4 Practical Guidelines and Tips

Every dataset is different and even for experienced practitioners, it is usually hard to tell whether a certain method will work well on a dataset before carrying out extensive exploratory data analysis. This section attempts to provide some tips that are summarized from the extensive simulations performed for this thesis about when the proposed methods may be useful (Section 6.4.1) and how to choose which variant to use and set various hyperparameters (Section 6.4.2).

6.4.1 When to Use the Proposed Methods

Need for Explanation Keep in mind that one of the major advantages of the proposed methods is its interpretability. If good prediction accuracy is the only concern, the proposed methods may not be the first choice. For instance, consider an Object Recognition application where the input features are pixel values of an image, and the goal is to output whether the image contains a dog. Most likely the user does not care which pixels are responsible for the decision as long as the decision can be reliably made. In contrast, in a Medical Diagnosis application where input are symptoms and test results, both the doctor and the patient are probably interested in how the final diagnosis is reached.

Quite conveniently, the datasets for which interpretability is needed in addition to predictive accuracy usually have human readable features, and tend to be problems for which humans have some intuitive understanding about. Thus we can make informed approximation about things such as the input distribution structure or the difficulty of the target function. The proposed methods should be considered in these scenarios.

Difficulty of the Target Function The proposed methods are NOT designed to compete with complex models such as deep neural nets. Rather, we target applications that are being solved by linear models and claim that the proposed methods offer improvement in predictive power while retaining the interpretability. As computational cost grows as $O(Nd^{d_0})$, most datasets can only allow a relatively small d_0. If we expect the target function to be much more complicated than what can be captured by second or third order interaction terms, the proposed methods may not be suitable.

For example, again consider the Object Recognition application mentioned above. As the shape and orientation of the dog in the image can vary, a correct decision probably requires features based on way more than a few pixels and we expect the proposed methods to perform poorly. In contrast, the MNIST dataset contains images of digits, and all the digits have roughly the same size and orientation. Thus it is not surprising that small patches of pixels can lead to correct decisions and even with $d_0 = 2$ the proposed methods achieve very good accuracy.

Feature Type The proposed methods only deal with categorical features. Discrete features can be directly treated as categorical ones, at the cost of losing the order information. Empirically, if the discrete feature only takes a few different values, this practice is usually not very costly.

If the dataset contains continuous-valued features, we can always find ways to quantize it into discrete values. Obviously information loss will occur during such quantization process, but whether the information loss is vital to the learning performance really depends on the actual problem at hand. Generally speaking, (1) it probably makes sense to quantize a few real-valued features when most others are categorical or discrete, but if most are real-valued to begin with, the proposed methods probably is not the right algorithm to use; (2) the quantization is likely to be less problematic for classification tasks than regression ones.

Amount of Available Data As simulation results indicate, the proposed methods can work well for a fairly wide range of dataset sizes. For example, for several dozens of raw features, the proposed methods offer competitive learning performance from a couple hundred training samples to much larger training set, provided the appropriate d_0 is used. However, since we usually need to estimate the parameters of the input distribution from data, N much smaller than a couple hundred is probably not enough.

6.4.2 How to Use the Proposed Methods

Encoding Scheme Given a categorical feature with more than two categories, we will need to encode it into binary features before applying the proposed methods.

Consider an example where a feature x_i takes 4 possible values $\{a, b, c, d\}$. We can one-hot encode this feature, which require 3 bits: $\mathbb{1}(x_i = a)$, $\mathbb{1}(x_i = b)$ and $\mathbb{1}(x_i = c)$. If all three bits equal to 0, we have $x_i = d$. Alternatively, we can use two bits to encode the feature: e.g. 00 for a, 01 for b, 10 for c and 11 for d. The latter scheme obviously requires fewer binary features. However, in practice, it is almost always observed that one-hot encoding allows for better learning performance because the resulting target function is 'simpler' and involving lower-order interactions.

Choice of d_0 The choice of d_0 is usually determined by the relative size of d and N. Recall that roughly $|\mathcal{C}| = O(N)$ coefficient estimates can be accommodated and since $|\{S \subseteq [d] : |S| \leq d_0\}| = O(d^{d_0})$, we roughly expect $d^{d_0} = O(N)$.

Sometimes understanding about the problem can guide the choice of d_0 as well. For instance, in the Tic Tac Toe dataset, we choose $d_0 = 3$ because each rule of winning the game involves 3 squares.

Which Variant to Use Following discussion in Section 6.2.1.4 and consistent observations on other synthetic or real datasets, we should use the Algorithm 3 (where features $\{\mathcal{X}_S(\mathbf{x})\}_{S\in\mathcal{C}}$ are used) if the downstream learning algorithm to use is not tree based and use Algorithm 2 (where features $\{\phi^p_S(\mathbf{x})\}_{S\in\mathcal{C}}$ are used) if it is.

Choice of Structural Assumption If the downstream learning algorithm is a tree-based method, product $p(\mathbf{x})$ is usually a safe option.

If the downstream algorithm is not tree-based, what structural assumption to use becomes a design problem. Usually we rely on some understanding of the problem. For example, if the input features are pixels of an image, a grid structure encodes the prior knowledge of 'near-by pixels are likely to take similar values'. For another example, if several binary features are the result of one-hot encoding of one original feature, we might want to connect the binary features in a Markov chain because we know that only one of the binary features can take value 1.

A useful strategy is to design a series of structural assumptions in increasing order of complexity, and select the sweet spot on the bias-variance trade-off using some validation data.

Chapter 7

Conclusions and Future Work

7.1 Conclusion Remarks

This thesis has developed **generalized Fourier spectrum**, a framework to discuss Boolean functions under arbitrary input distribution.

From a *theoretical* perspective, the framework is indeed a natural extension of the classic Boolean function Fourier spectrum, as many intuitions and important results can be carried over and meaningfully generalized from the uniform case. It offers the tools to explore the interaction between input distributions and target functions and how they jointly affect the difficulty of the learning problem.

From a *practical* perspective, the generalized Fourier spectrum allows systematic generation and efficient selection of new features based on the raw data, which in turn lead to models for learning categorical data that are both accurate in predictive performance and easy to understand.

7.2 Broader Context and Future Directions

Beyond regression and classification This thesis has focused on simple supervised learning tasks, namely classification and regression. However, the framework can potentially be useful in much more general settings.

For example, *structured prediction*, also known as *structured output learning* is a supervised learning problem where we need to simultaneously predict many labels y_1, y_2, ..., y_K and there is known structure among the target labels. All target labels share the same input features and thus the same input distribution. Consider the Fourier decomposition of each target label, any structure on the target labels can always be translated to structure on the corresponding Fourier coefficients. The interesting question is, *are there ways to translate commonly used structures on the output labels into easy-to-work-with conditions on the Fourier coefficients and thus tackling the structured prediction problem from the spectral perspective?*

Optimization over input distribution So far we assumed random access model, i.e. the learning algorithms are given i.i.d. training and testing examples. The proposed methods approximate the input distribution from which the samples are generated, and then perform spectral decomposition accordingly.

However, if we have more control on how training data can be generated, there is an interesting optimization problem to be asked: how can we find the 'best' input distribution which on one hand makes the target function easy to learn, and on the other hand covers most interesting use cases (in other words, test points likely to appear have relatively high probability under this distribution)?

Finding suitable features In some sense, finding the right set of features, i.e. finding a suitable representation of the data, is the most important step in any machine learning problem. Philosophically, anything that is 'learnable' should be highly structured and thus simple when viewed from the right perspective. Of course the real challenge is how to find that representation or perspective.

One extreme of searching for suitable data representation is the traditional *feature* *engineering*, which can easily make use of available domain knowledge but can be very time consuming. Moreover, domain knowledge can also be a double-edged sword and the model might be limited by human understanding of the problem. At the other extreme, deep neural networks are known to have the power of automatically learn such representations from data but usually cannot be made sense of.

The methods proposed in this thesis seem to get the best of both worlds, both automatically generated and selected, and highly interpretable. However, this is only possible because we restricted our attention to a much simpler setup where input features are binary and heavily utilized their special structures.

Appendices

Appendix A

Proof of Propositions in Chapter 3

A.1 Proof of Proposition 10

Proof. By assumption $p(\mathbf{x})$ is not a product distribution. There exists at least a pair of coordinates that are dependent. Without loss of generality, assume $x_1 \not\perp x_2$. Also assume the three conditions are all satisfied.

Denote $p_1 \triangleq \mathbb{P}(x_1 = -1)$, $p_2 \triangleq \mathbb{P}(x_2 = -1)$, $p \triangleq \mathbb{P}(x_1 = -1, x_2 = -1)$. By Condition 1, $h_{\{1\}}(\mathbf{x}) = h_{\{1\}}(x_1)$ and $h_{\{2\}}(\mathbf{x}) = h_{\{2\}}(x_2)$. Denote $h_{\{1\}}(-1), h_{\{1\}}(1), h_{\{2\}}(-1), h_{\{2\}}(1)$ as a, b, c, d respectively. By Condition 2, $h_{\{1,2\}}(-1,-1)$, $h_{\{1,2\}}(-1,1)$, $h_{\{1,2\}}(1,-1)$, $h_{\{1,2\}}(1,1)$ take values ac, ad, bc, bd respectively[1]. By Condition 3:

$$\begin{aligned}
\langle h_\varnothing, h_{\{1\}}\rangle_p &= ap_1 + b(1-p_1) = 0 \Rightarrow b = -\frac{p_1}{1-p_1}a \\
\langle h_\varnothing, h_{\{2\}}\rangle_p &= cp_2 + d(1-p_2) = 0 \Rightarrow d = -\frac{p_2}{1-p_2}c \\
\langle h_\varnothing, h_{\{1,2\}}\rangle_p &= acp + ad(p_1-p) + bc(p_2-p) + bd(1-p_1-p_2+p) \\
&= ac\big[p - \frac{p_2}{1-p_2}(p_1-p) - \frac{p_1}{1-p_1}(p_2-p) \\
&\quad + \frac{p_1p_2}{(1-p_1)(1-p_2)}(1-p_1-p_2+p)\big] \\
&= ac\big[p\frac{1}{(1-p_1)(1-p_2)} - \frac{p_1p_2}{(1-p_1)(1-p_2)}\big] = 0 \Rightarrow p = p_1p_2
\end{aligned}$$

[1]There is some abuse of notation to equate $h_S(\mathbf{x})$ and $h_S(\mathbf{x}_S)$.

With $p = p_1 p_2$, we can easily check that $\mathbb{P}(\mathsf{x}_1 = x_1, \mathsf{x}_2 = x_2) = \mathbb{P}(\mathsf{x}_1 = x_1)\mathbb{P}(\mathsf{x}_2 = x_2)$ for any choice of $(x_1, x_2) \in \{-1, 1\}^2$, contradicting the assumption that $\mathsf{x}_1 \not\perp \mathsf{x}_2$. $\square$

A.2 Proof of Proposition 11

Proof of Proposition 11. It is obvious that $\phi^p_{\varnothing} \equiv 1$ is unique up to sign regardless of choice of $p(\mathbf{x})$. Since decomposability is required, we only need to prove that $\phi^p_{\{k\}}(\mathbf{x})$ is unique up to sign, $\forall k \in [d]$. Also note that causality is required, $\phi^p_{\{k\}}(\mathbf{x}) = \phi^p_{\{k\}}(\mathbf{x}_1^k)$. As usual, we assume $1, 2, \cdots d$ is a topological ordering.

Consider $k = 1$,

$$1 = \langle \phi^p_{\{1\}}, \phi^p_{\{1\}} \rangle_p = \sum_{\mathbf{x}} p(\mathbf{x}) \left[\phi^p_{\{1\}}(\mathbf{x})\right]^2 = p_{\mathsf{x}_1}(1) \left[\phi^p_{\{1\}}(\mathsf{x}_1 = 1)\right]^2 + p_{\mathsf{x}_1}(-1) \left[\phi^p_{\{1\}}(\mathsf{x}_1 = -1)\right]^2$$

$$0 = \langle \phi^p_{\varnothing}, \phi^p_{\{1\}} \rangle_p = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot 1 \cdot \phi^p_{\{1\}}(\mathbf{x}) = p_{\mathsf{x}_1}(1)\phi^p_{\{1\}}(\mathsf{x}_1 = 1) + p_{\mathsf{x}_1}(-1)\phi^p_{\{1\}}(\mathsf{x}_1 = -1)$$

Solving these two equations, we have

$$\left[\phi^p_{\{1\}}(\mathsf{x}_1 = 1)\right]^2 = \frac{p_{\mathsf{x}_1}(-1)}{p_{\mathsf{x}_1}(1)}, \qquad \left[\phi^p_{\{1\}}(\mathsf{x}_1 = -1)\right]^2 = \frac{p_{\mathsf{x}_1}(1)}{p_{\mathsf{x}_1}(-1)}$$

in other words $\phi^p_{\{1\}}(\mathsf{x}_1) = \mathsf{x}_1\sqrt{p(-\mathsf{x}_1)/p(\mathsf{x}_1)}$ or $\phi^p_{\{1\}}(\mathsf{x}_1) = -\mathsf{x}_1\sqrt{p(-\mathsf{x}_1)/p(\mathsf{x}_1)}$. Without loss of generality, let us choose $\phi^p_{\{1\}}(\mathsf{x}_1) = \mathsf{x}_1\sqrt{p(-\mathsf{x}_1)/p(\mathsf{x}_1)}$.

Now consider $k = 2$,

$$0 = \langle \phi^p, \phi^p_{\{2\}} \rangle_p = \sum_{\mathbf{x}} p(\mathbf{x}) \cdot 1 \cdot \phi^p_{\{2\}}(\mathbf{x}) = \sum_{x_1} p_{\mathsf{x}_1}(x_1) \sum_{x_2} \phi^p_{\{2\}}(\mathbf{x}_1^2) p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1) \tag{A.1}$$

$$0 = \langle \phi^p_{\{1\}}, \phi^p_{\{2\}} \rangle_p = \sum_{\mathbf{x}} p(\mathbf{x}) \phi^p_{\{1\}}(\mathbf{x}) \phi^p_{\{2\}}(\mathbf{x}) = \sum_{x_1} x_1 \sqrt{p_{\mathsf{x}_1}(1)p_{\mathsf{x}_1}(-1)} \sum_{x_2} \phi^p_{\{2\}}(\mathbf{x}_1^2) p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1)$$

$$\Rightarrow 0 = \sum_{x_1} x_1 \sum_{x_2} \phi^p_{\{2\}}(\mathbf{x}_1^2) p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1) \tag{A.2}$$

Combining Equations (A.1) and (A.2), we have

$$\sum_{x_2} \phi^p_{\{2\}}(\mathbf{x}_1^2) p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1) = 0, \quad \forall x_1 \in \{-1, 1\} \tag{A.3}$$

$$0 = \langle \phi^p_{\{2\}}, \phi^p_{\{1,2\}} \rangle_p = \sqrt{p_{\mathsf{x}_1}(1) p_{\mathsf{x}_1}(-1)} \sum_{x_1} x_1 \sum_{x_2} \left[\phi^p_{\{2\}}(\mathbf{x}_1^2)\right]^2 p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1)$$

$$\Rightarrow 0 = \sum_{x_1} x_1 \sum_{x_2} \left[\phi^p_{\{2\}}(\mathbf{x}_1^2)\right]^2 p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1) \tag{A.4}$$

$$1 = \langle \phi^p_{\{2\}}, \phi^p_{\{2\}} \rangle_p = \sum_{x_1} p_{\mathsf{x}_1}(x_1) \sum_{x_2} \left[\phi^p_{\{2\}}(\mathbf{x}_1^2)\right]^2 p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1) \tag{A.5}$$

Combining Equations (A.4) and (A.5), we have

$$\sum_{x_2} \left[\phi^p_{\{2\}}(\mathbf{x}_1^2)\right]^2 p_{\mathsf{x}_2|\mathsf{x}_1}(x_2|x_1) = 1, \quad \forall x_1 \in \{-1, 1\} \tag{A.6}$$

Solving for Equations (A.3) and (A.6), and without loss of generality we choose $\phi^p_{\{2\}}(x_1 = 1, x_2 = 1)$ to be positive, then

$$\phi^p_{\{2\}}(\mathsf{x}) = \mathsf{x}_2 \sqrt{\frac{p(-\mathsf{x}_2|\mathsf{x}_1)}{p(\mathsf{x}_2|\mathsf{x}_1)}}$$

Note that $\mathsf{x}_1\mathsf{x}_2\sqrt{p(-\mathsf{x}_2|\mathsf{x}_1)/p(\mathsf{x}_2|\mathsf{x}_1)}$ will also satisfy both Equations (A.3) and (A.6). However, it will depend on both x_1 and x_2 even if $\mathsf{x}_1 \perp \mathsf{x}_2$, thus not satisfying the causality requirement.

For general $k \in [d]$, we can similarly deduce from the orthonormality conditions that $\forall \mathbf{x}_1^{k-1} \in \{-1, 1\}^{k-1}$,

$$\sum_{x_k} \phi^p_{\{k\}}(\mathbf{x}_1^k) p(x_k|\mathbf{x}_1^{k-1}) = 0$$

$$\sum_{x_k} \left[\phi^p_{\{k\}}(\mathbf{x}_1^k)\right]^2 p(x_k|\mathbf{x}_1^{k-1}) = 1$$

Assume $\phi^p_{\{k\}}(\mathbf{x}_1^k = \mathbf{1})$ is positive and combine with causality requirement, we have

$$\phi^p_{\{k\}}(\mathbf{x}_1^k = x_k \sqrt{\frac{p(-x_k|\mathbf{x}_1^{k-1})}{p(x_k|\mathbf{x}_1^{k-1})}}$$

□

A.3 Proof of Proposition 12

Proof of Proposition 12.

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})] &= \sum_{\mathbf{x}} p(\mathbf{x})f(\mathbf{x}) = \sum_{\mathbf{x}} p(\mathbf{x}) \sum_{S\subseteq[d]} \hat{f}^p(S)\phi^p_S(\mathbf{x}) = \sum_{S\subseteq[d]} \hat{f}^p(S) \sum_{\mathbf{x}} p(\mathbf{x})\phi^p_S(\mathbf{x}) \\
&= \sum_{S\subseteq[d]} \hat{f}^p(S)\langle 1, \phi^p_S\rangle_p = \sum_{S\subseteq[d]} \hat{f}^p(S)\mathbb{1}(S=\varnothing) = \hat{f}^p(\varnothing)
\end{aligned}$$

$$\begin{aligned}
\mathbb{E}_{\mathbf{x}\sim p}[f^2(\mathbf{x})] &= \sum_{\mathbf{x}} p(\mathbf{x})f^2(\mathbf{x}) = \sum_{\mathbf{x}} p(\mathbf{x})\Bigg(\sum_{S\subseteq[d]} \hat{f}^p(S)\phi^p_S(\mathbf{x})\Bigg)\Bigg(\sum_{T\subseteq[d]} \hat{f}^p(T)\phi^p_T(\mathbf{x})\Bigg) \\
&= \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{f}^p(T)\sum_{\mathbf{x}} p(\mathbf{x})\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{x}) \\
&= \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{f}^p(T)\langle\phi^p_S,\phi^p_T\rangle_p = \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{f}^p(T)\mathbb{1}(S=T) \\
&= \sum_{S\subseteq[d]} [\hat{f}^p(S)]^2 \\
\mathrm{Var}_{\mathbf{x}\sim p}(f(\mathbf{x})) &= \mathbb{E}_{\mathbf{x}\sim p}[f^2(\mathbf{x})] - (\mathbb{E}_{\mathbf{x}\sim p}[f(\mathbf{x})])^2 = \sum_{S\subseteq[d]} [\hat{f}^p(S)]^2 - (\hat{f}^p(\varnothing))^2 = \sum_{S\neq\varnothing} [\hat{f}^p(S)]^2 \\
\langle f, g\rangle_p &= \sum_{\mathbf{x}} p(\mathbf{x})f(\mathbf{x})g(\mathbf{x}) = \sum_{\mathbf{x}} p(\mathbf{x})\Bigg(\sum_{S\subseteq[d]} \hat{f}^p(S)\phi^p_S(\mathbf{x})\Bigg)\Bigg(\sum_{T\subseteq[d]} \hat{g}^p(T)\phi^p_T(\mathbf{x})\Bigg) \\
&= \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{g}^p(T)\sum_{\mathbf{x}} p(\mathbf{x})\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{x}) \\
&= \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{g}^p(T)\langle\phi^p_S,\phi^p_T\rangle_p = \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{g}^p(T)\mathbb{1}(S=T) \\
&= \sum_{S\subseteq[d]} \hat{f}^p(S)\hat{g}^p(S)
\end{aligned}$$

□

A.4 Proof of Proposition 14

Proof. Let K denote the last coordinate where $\mathbf{x}$ and $\mathbf{y}$ are coupled. It's not hard to see from the process defined above that $K = k$ with probability $\rho^k(1-\rho)$ for $k = 0, 1, \cdots, d-1$ and $K = d$ with probability ρ^d

$$\mathbb{E}_{\mathbf{x},\mathbf{y}}[f(\mathbf{x})f(\mathbf{y})] = \mathbb{E}_{\mathbf{x},\mathbf{y}}\Big[\Big(\sum_{S\subseteq[d]} \hat{f}^p(S)\phi^p_S(\mathbf{x})\Big)\Big(\sum_{T\subseteq[d]} \hat{f}^p(T)\phi^p_T(\mathbf{y})\Big)\Big]$$

$$= \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{f}^p(T)\mathbb{E}_{\mathbf{x},\mathbf{y}}[\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{y})]$$

$$= \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{f}^p(T)\mathbb{E}_K[\mathbb{E}_{\mathbf{x},\mathbf{y}}[\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{y})|K]] \tag{A.7}$$

$$\mathbb{E}_{\mathbf{x},\mathbf{y}}[\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{y})|K=k] = \mathbb{E}_{\mathbf{x}_1^k}\big[\mathbb{E}_{\mathbf{x}_{k+1}^d,\mathbf{y}_{k+1}^d}[\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{y})\big|\mathbf{x}_1^k]]$$

$$= \mathbb{E}_{\mathbf{x}_1^k}\Big[\prod_{i\in S, i\le k}\phi^p_{\{i\}}(\mathbf{x}_1^k)\prod_{j\in T, j\le k}\phi^p_{\{j\}}(\mathbf{x}_1^k)\mathbb{E}_{\mathbf{x}_{k+1}^d}[\prod_{i\in S:i>k}\phi^p_{\{i\}}(\mathbf{x})|\mathbf{x}_1^k]\mathbb{E}_{\mathbf{y}_{k+1}^d}[\prod_{j\in T:j>k}\phi^p_{\{j\}}(\mathbf{y})|\mathbf{x}_1^k]\Big]$$

$$= \mathbb{E}_{\mathbf{x}_1^k}\Big[\prod_{i\in S, i\le k}\phi^p_{\{i\}}(\mathbf{x}_1^k)\prod_{j\in T, j\le k}\phi^p_{\{j\}}(\mathbf{x}_1^k)\mathbb{1}(S\subseteq[k])\mathbb{1}(T\subseteq[k])\Big]$$

$$= \mathbb{1}(S=T)\mathbb{1}(S\subseteq[k]) \tag{A.8}$$

Plug Equation (A.8) into Equation (A.7), we get

$$\mathbb{E}_{\mathbf{x},\mathbf{y}}[f(\mathbf{x})f(\mathbf{y})] = \sum_{S,T\subseteq[d]} \hat{f}^p(S)\hat{f}^p(T)\big\{(1-\rho)\mathbb{1}(S=T=\varnothing) + (1-\rho)\rho\mathbb{1}(S=T\subseteq[1])$$
$$+ \cdots + (1-\rho)\rho^{d-1}\mathbb{1}(S=T\subseteq[d-1]) + \rho^d\mathbb{1}(S=T\subseteq[d])\big\}$$
$$= \sum_{S\subseteq[d]}[\hat{f}^p(S)]^2\rho^{\max(S)}$$

□

A.5 Proof of Proposition 15

Proof.

$$\mathbb{E}_{\mathbf{x},\mathbf{y}}[f(\mathbf{x})f(\mathbf{y})] = \mathbb{E}_{\mathbf{x},\mathbf{y}}\Big[\Big(\sum_{S\subseteq[d]}\hat{f}^p(S)\phi^p_S(\mathbf{x})\Big)\Big(\sum_{T\subseteq[d]}\hat{f}^p(T)\phi^p_T(\mathbf{y})\Big)\Big]$$

$$= \sum_{S,T\subseteq[d]}\hat{f}^p(S)\hat{f}^p(T)\mathbb{E}_{\mathbf{x},\mathbf{y}}[\phi^p_S(\mathbf{x})\phi^p_T(\mathbf{y})]$$

$$= \sum_{S,T\subseteq[d]}\hat{f}^p(S)\hat{f}^p(T)\sum_{\mathbf{x}}p(\mathbf{x})\phi^p_S(\mathbf{x})\sum_{\mathbf{y}}p(\mathbf{y}|\mathbf{x})\phi^p_T(\mathbf{y}) \triangleq \sum_{S,T\subseteq[d]}\hat{f}^p(S)\hat{f}^p(T)A(S,T)$$

Now let us figure out what $A(S,T)$ is for each pair (S,T).
If $T=\varnothing$, $A(S,T)=\sum_{\mathbf{x}}p(\mathbf{x})\phi^p_S(\mathbf{x})=\langle 1,\phi^p_S\rangle_p=\mathbb{1}(S=\varnothing)$.
If $T\neq\varnothing$, by definition T must contain at least one element. Let us denote by k the element in T that comes last in the topological ordering (we assume $1,2,\cdots,d$ is the topological ordering, as always). Then

$$A(S,T) = \sum_{\mathbf{x}}p(\mathbf{x})\phi^p_S(\mathbf{x})\sum_{\mathbf{y}_1^k}p(\mathbf{y}_1^k|\mathbf{x}_1^d)\phi^p_T(\mathbf{y}_1^k)$$

$$= \sum_{\mathbf{x}}p(\mathbf{x})\phi^p_S(\mathbf{x})\sum_{\mathbf{y}_1^{k-1}}p(\mathbf{y}_1^{k-1}|\mathbf{x}_1^d)\phi^p_{T\setminus\{k\}}(\mathbf{y}_1^{k-1})\sum_{y_k}p(y_k|\mathbf{y}_1^{k-1},\mathbf{x}_1^d)\phi^p_{\{k\}}(\mathbf{y}_1^k) \tag{A.9}$$

Recall the sequential data generation procedure defined, given $\mathbf{x}$, there are two possible ways $\mathbf{y}_k$ is generated:

(1) if all the parents of node k have been set to equal to the corresponding bits in $\mathbf{x}$, then with probability ρ, y_k is set to equal to x_k. In this case, Equation (A.9) can be written as:

$$\sum_{\mathbf{x}}p(\mathbf{x})\phi^p_S(\mathbf{x})\left[\phi^p_{\{k\}}(\mathbf{x}_1^k)\right]\sum_{\mathbf{y}_1^{k-1}}p(\mathbf{y}_1^{k-1}|\mathbf{x}_1^d)\phi^p_{T\setminus\{k\}}(\mathbf{y}_1^{k-1})$$

If we keep doing the same thing to the node in $T \setminus k$ etc., we can get:

$$A(S,T) = \sum_{\mathbf{x}} p(\mathbf{x})\phi_S^p(\mathbf{x})\phi_T^p(\mathbf{x}) = \langle \phi_S^p, \phi_T^p \rangle_p = \mathbb{1}(S = T)$$

And this will happen if and only if $\forall i \in [d]$ s.t. $i \in S$ or i is the ancestor of a node in S, we have $x_i = y_i$.

(2) if y_k is randomly generated, then $p(y_k|\mathbf{y}_1^{k-1}, \mathbf{x}_1^d) = p(y_k|\mathbf{y}_1^{k-1})$, thus Equation (A.9) equals to 0.

Combining the discussions so far, we have

$$\mathbb{E}_{\mathbf{x},\mathbf{y}}[f(\mathbf{x})f(\mathbf{y})] = \sum_{S \subseteq [d]} \left[\hat{f}^p(S)\right]^2 \mathbb{P}_{\mathbf{x},\mathbf{y}}(\mathbf{x}, \mathbf{y} \text{ share values on all nodes in } S \text{ and their ancestors})$$

Given the data generation process, it's not hard to see that the probability is $\phi^{\sigma(S)}$. □

A.6 Proof of Proposition 16

Proof. A linear smoother has the form $\hat{y}(\mathbf{z}) = \sum_{n=1}^N w(\mathbf{z}, \mathbf{x}^n) \cdot y^n$, where $\{(\mathbf{x}^n, y^n)\}_{n=1}^N$ are training samples and $\mathbf{z}$ is a test data point. Thus $\hat{\mathbf{y}} = S\mathbf{y}$, where the matrix $S \in \mathbb{R}^{N \times N}$ is defined as $S_{m,n} = w(\mathbf{x}^m, \mathbf{x}^n)$. Then

$$\mathrm{df}(\hat{\mathbf{y}}) = \frac{1}{\sigma^2}\mathrm{tr}\big(\mathrm{Cov}(S\mathbf{y}, \mathbf{y})\big) = \frac{1}{\sigma^2}\mathrm{tr}\big(S\mathrm{Cov}(\mathbf{y}, \mathbf{y})\big) = \mathrm{tr}(S) = \sum_{n=1}^N w(\mathbf{x}^n, \mathbf{x}^n)$$

Plug in our weights $w_{\mathcal{C}}(\cdot,\cdot)/N$, we get

$$\mathrm{df}(\hat{\mathbf{y}}) = \frac{1}{N}\sum_{n=1}^N w_{\mathcal{C}}(\mathbf{x}^n, \mathbf{x}^n) = \frac{1}{N}\sum_{n=1}^N \sum_{S \in \mathcal{C}} \left[\phi_S^p(\mathbf{x}^n)\right]^2 = \frac{1}{N}\sum_{n=1}^N \sum_{S \in \mathcal{C}} \prod_{i \in S} \frac{p(-x_i^n|\mathbf{x}_{\pi_i}^n)}{p(x_i^n|\mathbf{x}_{\pi_i}^n)}$$

□

A.7 Proof of Proposition 17

Proof.

$$\mathbb{E}[w_{\mathcal{C}}(\mathbf{x},\mathbf{z})] = \mathbb{E}[\sum_{S\in\mathcal{C}} \phi_S^p(\mathbf{x})\phi_S^p(\mathbf{z})] = \sum_{S\in\mathcal{C}} \phi_S^p(\mathbf{z}) \sum_{\mathbf{x}_{k+1}^d} p(\mathbf{x}_{k+1}^d | \mathbf{x}_1^k = \mathbf{z}_1^k)\phi_S^p(\mathbf{x})$$
$$= \sum_{S\in\mathcal{C}} \phi_S^p(\mathbf{z}) \sum_{\mathbf{x}_{k+1}^d} \Big(\prod_{i=k+1}^{d} p(x_i|\mathbf{x}_{\pi_i})\Big)\Big(\prod_{j\in S} x_j \sqrt{\frac{p(-x_i|\mathbf{x}_{\pi_i})}{p(x_i|\mathbf{x}_{\pi_i})}}\Big)$$
$$= \sum_{S\in\mathcal{C}} \phi_S^p(\mathbf{z})\mathbb{1}(S\cap\{k+1,\cdots,d\}=\varnothing)\phi_S^p(\mathbf{x}) = \sum_{S\in\mathcal{C}, S\subseteq[k]} \big[\phi_S^p(\mathbf{z})\big]^2$$

The last equality above used the assumption that $\mathbf{x}_1^k = \mathbf{z}_1^k$.

$$\mathrm{Var}(w_{\mathcal{C}}(\mathbf{x},\mathbf{z})) = \mathbb{E}[w_{\mathcal{C}}^2(\mathbf{x},\mathbf{z})] - \big(\mathbb{E}[w_{\mathcal{C}}(\mathbf{x},\mathbf{z})]\big)^2$$
$$= \sum_{S,T\in\mathcal{C}} \phi_S^p(\mathbf{z})\phi_T^p(\mathbf{z}) \sum_{\mathbf{x}_{k+1}^d} p(\mathbf{x}_{k+1}^d|\mathbf{x}_1^k = \mathbf{z}_1^k)\phi_S^p(\mathbf{x})\phi_T^p(\mathbf{x}) - \Big(\sum_{\substack{S\in\mathcal{C}\\ S\subseteq[k]}} \big[\phi_S^p(\mathbf{z})\big]^2\Big)^2$$
$$= \sum_{S,T\in\mathcal{C}} \phi_S^p(\mathbf{z})\phi_T^p(\mathbf{z}) \sum_{\mathbf{x}_{k+1}^d} \prod_{i=k+1}^{d} p(-x_i|\mathbf{x}_{\pi_i}) \prod_{j\in S\cap T} \big[\phi_{\{j\}}^p(\mathbf{x})\big]^2 \prod_{k\in S\triangle T} \phi_{\{k\}}^p(\mathbf{x})$$
$$- \sum_{\substack{S,T\in\mathcal{C}\\ S,T\subseteq[k]}} \big[\phi_S^p(\mathbf{z})\big]^2\big[\phi_T^p(\mathbf{z})\big]^2$$
$$= \sum_{S,T\in\mathcal{C}} \phi_S^p(\mathbf{z})\phi_T^p(\mathbf{z})\Big(\prod_{j\in(S\cap T)\cap[k]} \big[\phi_{\{j\}}^p(\mathbf{z})\big]^2\Big)\Big(\prod_{k\in(S\triangle T)\cap[k]} \phi_{\{k\}}^p(\mathbf{z})\Big)\mathbb{1}(S\cap A_k = T\cap A_k)$$
$$- \sum_{\substack{S,T\in\mathcal{C}\\ S,T\subseteq[k]}} \big[\phi_S^p(\mathbf{z})\big]^2\big[\phi_T^p(\mathbf{z})\big]^2 \tag{A.10}$$
$$= \sum_{\substack{S,T\subseteq[k]\\ \varnothing\neq R\subseteq[d]\setminus[k]\\ S\cup R\in\mathcal{C}, T\cup R\in\mathcal{C}}} \big[\phi_S^p(\mathbf{z})\big]^2\big[\phi_T^p(\mathbf{z})\big]^2\big[\phi_R^p(\mathbf{z})\big]^2$$

The A_k in Equation (A.10) denotes the set $\{k+1, k+2, \cdots, d\}$. □

Appendix B

Proof of Propositions in Chapter 4

B.1 Proof of Proposition 19

Proof. Without loss of generality, assume there are more training data labeled as '1' than '-1'. We will label all $\mathbf{x}$ that hasn't appeared in training as 1 and argue the resulting function has small spectral L_1 norm. Let us denote the set of input $\mathbf{x}$ that appeared in training and has label -1 as $\mathcal{D}^-$.

$$\begin{aligned}
\hat{\|f\|}_1 &= \sum_{S\subseteq[d]} \frac{1}{2^d}\Big|\sum_{\mathbf{x}} f(\mathbf{x})\mathcal{X}_S(\mathbf{x})\Big| = \sum_{S\subseteq[d]} \frac{1}{2^d}\Big|\sum_{\mathbf{x}\notin\mathcal{D}^-} \mathcal{X}_S(\mathbf{x}) - \sum_{\mathbf{x}\in\mathcal{D}^-} \mathcal{X}_S(\mathbf{x})\Big| \\
&= \sum_{S\subseteq[d]} \frac{1}{2^d}\Big|\sum_{\mathbf{x}\in\{-1,1\}^d} \mathcal{X}_S(\mathbf{x}) - 2\sum_{\mathbf{x}\in\mathcal{D}^-} \mathcal{X}_S(\mathbf{x})\Big| \\
&\le 1 + \sum_{S\neq\varnothing} \frac{1}{2^{d-1}}\Big|\sum_{\mathbf{x}\in\mathcal{D}^-} \mathcal{X}_S(\mathbf{x})\Big| = 1 + \frac{1}{2^{d-1}}\sum_{\mathbf{z}\neq\mathbf{1}}\Big|\sum_{T\in\mathcal{D}^-} \mathcal{X}_T(\mathbf{z})\Big|
\end{aligned}$$

The steps above are pretty self-explanatory, except for the last equality, which we discuss in more detail below:

Note that under the mapping $-1 \leftrightarrow 1, 1 \leftrightarrow 0$, all possible inputs $\mathbf{x} \in \{-1,1\}^d$ and all subsets of $[d]$ are in 1-1 correspondence. For example, the empty set corresponds to $\mathbf{1} \triangleq (1,1,\cdots,1)$. Now consider a $2^d \times 2^d$ table, where rows correspond to inputs and columns to subsets. The entry at the row corresponding to $\mathbf{x}$ and the column corresponding to S is $\mathcal{X}_S(\mathbf{x}) = \prod_{i=1}^d (-1)^{\mathbb{1}(x_i=-1 \text{ and } i\in S)}$. Let $\mathbf{z}$ be the input corresponding to S and let T be the subset corresponding to input $\mathbf{x}$. Then

$$\mathcal{X}_S(\mathbf{x}) = \prod_{i=1}^{d} (-1)^{\mathbb{1}(x_i=-1, z_i=-1)} = \mathcal{X}_T(\mathbf{z})$$

Hence the last equality holds. Let $g(\mathbf{z}) \triangleq \sum_{T\in\mathcal{D}^-} \mathcal{X}_T(\mathbf{z})$. Apply Parseval's theorem,

$$\mathbb{E}[g^2(\mathbf{z})] = \frac{1}{2^d}\sum_{\mathbf{z}} g^2(\mathbf{z}) = \sum_{T\subseteq[d]} \hat{g}^2(T) = \sum_{T\in\mathcal{D}^-} 1 = \left|\mathcal{D}^-\right| \le \frac{N}{2}$$

Thus we have

$$\begin{aligned}\hat{\|}f\hat{\|}_1 &\le 1 + \frac{1}{2^{d-1}}\sum_{\mathbf{z}\neq\mathbf{1}}\Big|\sum_{T\in\mathcal{D}^-}\mathcal{X}_T(\mathbf{z})\Big| \le 1 + \frac{1}{2^{d-1}}\sum_{\mathbf{z}}\left|g(\mathbf{z})\right| \\ &\le 1 + \frac{1}{2^{d-1}}\sqrt{2^d\sum_{\mathbf{z}} g^2(\mathbf{z})} \le 1 + \frac{1}{2^{d-1}}\sqrt{2^{2d}\frac{N}{2}} = 1 + \sqrt{2N}\end{aligned}$$

□

B.2 Proof of Proposition 22

Proof. For any $S \subseteq [d]$, we have

$$\begin{aligned}\hat{f}^p(S) &= \mathbb{E}_p[f(\mathbf{x})\phi^p_S(\mathbf{x})] = \mathbb{E}_p\Big[\Big(\sum_{T\subseteq[d]}\hat{f}(T)\mathcal{X}_T(\mathbf{x})\Big)\phi^p_S(\mathbf{x})\Big] \\ &= \sum_{T\subseteq[d]}\hat{f}(T)\mathbb{E}_p[\phi^p_S(\mathbf{x})\mathcal{X}_T(\mathbf{x})]\end{aligned}$$

□

B.3 Proof of Proposition 23

Proof.

$$\begin{aligned}
\hat{f}^p(S) &= \sum_{T\subseteq[d]} \hat{f}(T) \sum_{\mathbf{x}} p(\mathbf{x})\phi_S^p(\mathbf{x})\mathcal{X}_T(\mathbf{x}) \\
&= \sum_{T\subseteq[d]} \hat{f}(T) \sum_{\mathbf{x}} \prod_{i\in S\cap T} \sqrt{p_i(1-p_i)} \prod_{j\in S\setminus T} x_j\sqrt{p_j(1-p_j)} \prod_{k\in T\setminus S} x_i\mathbb{P}(x_k) \prod_{l\notin S\cup T} \mathbb{P}(x_l) \\
&= \sum_{T\subseteq[d]} \hat{f}(T) \prod_{i\in S\cap T} 2\sqrt{p_i(1-p_i)} \prod_{j\in S\setminus T} 0 \prod_{k\in T\setminus S} (1-2p_i) \prod_{l\notin S\cup T} 1 \\
&= \sum_{T\subseteq[d]:S\setminus T=\varnothing} \hat{f}(T) \prod_{i\in S\cap T} 2\sqrt{p_i(1-p_i)} \prod_{k\in T\setminus S} (1-2p_i) \\
&= \Big[\prod_{i\in S} 2\sqrt{p_i(1-p_i)}\Big]\Big[\sum_{T\subseteq[d]:S\subseteq T} \hat{f}(T) \prod_{k\in T\setminus S} (1-2p_i)\Big]
\end{aligned}$$

□

B.4 Proof of Proposition 25

Proof. Given how $f(\mathbf{x})$ is generated and $\hat{f}^p(S) = \sum_{T\subseteq[d]} M_{S,T}^p \hat{f}(T)$, we have $\hat{f}^p(S) \sim \mathcal{N}(0, \sum_{T\subseteq[d]}[M_{S,T}^p]^2)$. Thus

$$\mathbb{E}\big[\sum_{S\subseteq[d]} |\hat{f}^p(S)|\big] = \sqrt{\frac{2}{\pi}} \sum_{S\subseteq[d]} \sqrt{\sum_{T\subseteq[d]} [M_{S,T}^p]^2} = \sqrt{\frac{2}{\pi}} \sum_{S\subseteq[d]} \sqrt{2^d \sum_{\mathbf{x}} p^2(\mathbf{x})[\phi_S^p(\mathbf{x})]^2}$$

The last equality follows from

$$\begin{aligned}
\sum_{T\subseteq[d]} [M_{S,T}^p]^2 &= \sum_{\mathbf{x},\mathbf{x}'} p(\mathbf{x})p(\mathbf{x}')\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{x}') \sum_{T\subseteq[d]} \mathcal{X}_T(\mathbf{x})\mathcal{X}_T(\mathbf{x}') \\
&= \sum_{\mathbf{x},\mathbf{x}'} p(\mathbf{x})p(\mathbf{x}')\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{x}')\big[2^d\mathbb{1}(\mathbf{x}=\mathbf{x}')\big] = 2^d \sum_{\mathbf{x}} p^2(\mathbf{x})[\phi_S^p(\mathbf{x})]^2
\end{aligned}$$

As a special case, $\mathbb{E}\big[\sum_{S\subseteq[d]}|\hat{f}(S)|\big] = \sqrt{2/\pi}\sum_{S\subseteq[d]}\sqrt{2^d\sum_{\mathbf{x}}(1/2^d)^2} = 2^d\sqrt{2/\pi}$. Thus

$$\mathbb{E}\big[\sum_{S\subseteq[d]}|\hat{f}(S)| - \sum_{S\subseteq[d]}|\hat{f}^p(S)|\big] = 2^d\sqrt{\frac{2}{\pi}}\left[1-\frac{1}{2^d}\sum_{S\subseteq[d]}\sqrt{2^d\sum_{\mathbf{x}}p^2(\mathbf{x})[\phi^p_S(\mathbf{x})]^2}\right]$$

Finally, the inequality in Equation (4.9) follows from

$$\begin{aligned}
&\sum_{S\subseteq[d]}\sqrt{2^d\sum_{\mathbf{x}}p^2(\mathbf{x})[\phi^p_S(\mathbf{x})]^2} \le \sqrt{2^d\sum_{S\subseteq[d]}[2^d\sum_{\mathbf{x}}p^2(\mathbf{x})[\phi^p_S(\mathbf{x})]^2]}\\
&= 2^d\sqrt{\sum_{\mathbf{x}}p^2(\mathbf{x})\sum_{S\subseteq[d]}[\phi^p_S(\mathbf{x})]^2} = 2^d\sqrt{\sum_{\mathbf{x}}p^2(\mathbf{x})\sum_{S\subseteq[d]}[\prod_{i\in S}\phi^p_{\{i\}}(\mathbf{x})]^2}\\
&= 2^d\sqrt{\sum_{\mathbf{x}}p^2(\mathbf{x})\prod_{i=1}^{d}(1+[\phi^p_{\{i\}}(\mathbf{x})]^2)} = 2^d\sqrt{\sum_{\mathbf{x}}p^2(\mathbf{x})\prod_{i=1}^{d}\frac{1}{p(x_i|\mathbf{x}_{\pi_i})}} = 2^d\sqrt{\sum_{\mathbf{x}}p(\mathbf{x})} = 2^d
\end{aligned}$$

□

B.5 Proof of Proposition 26

Proof.

$$\begin{aligned}
&\sum_{S\subseteq[d]}|S|\hat{f}^2(S) - \sum_{S\subseteq[d]}|S|\big[\hat{f}^p(S)\big]^2 = \sum_{S\subseteq[d]}|S|\hat{f}^2(S) - \sum_{S\subseteq[d]}|S|\Big(\sum_{T\subseteq[d]}\hat{f}(T)M^p_{S,T}\Big)^2\\
&= \sum_{S\subseteq[d]}|S|\hat{f}^2(S) - \sum_{S\subseteq[d]}|S|\sum_{T,R\subseteq[d]}\hat{f}(T)\hat{f}(R)M^p_{S,T}M^p_{S,R}\\
&= \sum_{T\subseteq[d]}\hat{f}^2(T)|T| - \sum_{T,R\subseteq[d]}\hat{f}(T)\hat{f}(R)\sum_{S\subseteq[d]}|S|M^p_{S,T}M^p_{S,R} \triangleq \hat{\mathbf{f}}^T A^p\hat{\mathbf{f}}
\end{aligned}$$

where $A^p \in \mathbb{R}^{2^d\times 2^d}$, $A^p_{T,T} = |T| - \sum_{S\subseteq[d]}|S|\big[M^p_{S,T}\big]^2$ and for $\forall T \neq R$, $T,R\subseteq[d]$, $A^p_{T,R} = -\sum_{S\subseteq[d]}|S|M^p_{S,T}M^p_{S,R}$.

Note that A^p is a real, symmetric matrix, and thus we have $A^p = PDP^T$, where P is an orthonormal matrix and D is a diagonal matrix whose diagonal entries correspond to the eigenvalues of A^p (denote as $\lambda_1,\lambda_2,\cdots,\lambda_{2^d}$). Therefore $\hat{\mathbf{f}}^T A^p\hat{\mathbf{f}} = (P^T\hat{\mathbf{f}})^T D(P^T\hat{\mathbf{f}})$.

Also note by definition $\hat{\mathbf{f}} \sim \mathcal{N}(\mathbf{0}, I_{2^d})$, and thus $P^T\hat{\mathbf{f}} \triangleq \mathbf{z} \sim \mathcal{N}(\mathbf{0}, I_{2^d})$ as a result of the orthonormality of P. Since D is diagonal, we have $\hat{\mathbf{f}}^T A^p \hat{\mathbf{f}} = \sum_{i=1}^{2^d} \lambda_i z_i^2$ where z_i^2's are i.i.d. Chi-squared variables of degree 1.

$$\begin{aligned}
\therefore \quad \mathbb{E}[\hat{\mathbf{f}}^T A^p \hat{\mathbf{f}}] &= \sum_{i=1}^{2^d} \lambda_i \mathbb{E}[z_i^2] = \sum_{i=1}^{2^d} \lambda_i = \text{Trace}(A^p) = \sum_{T\subseteq[d]} \left(|T| - \sum_{S\subseteq[d]} |S| \left[M_{S,T}^p\right]^2 \right) \\
&= d \cdot 2^{d-1} - \sum_{S,T\subseteq[d]} |S| \sum_{\mathbf{x},\mathbf{x}'} p(\mathbf{x})p(\mathbf{x}')\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{x}')\mathcal{X}_T(\mathbf{x})\mathcal{X}_T(\mathbf{x}') \\
&= d \cdot 2^{d-1} - \sum_{\mathbf{x},\mathbf{x}'} p(\mathbf{x})p(\mathbf{x}') \sum_{S\subseteq[d]} |S|\phi_S^p(\mathbf{x})\phi_S^p(\mathbf{x}') \sum_{T\subseteq[d]} \mathcal{X}_T(\mathbf{x})\mathcal{X}_T(\mathbf{x}') \\
&= d \cdot 2^{d-1} - 2^d \sum_{\mathbf{x}} p^2(\mathbf{x}) \sum_{S\subseteq[d]} |S| \left[\phi_S^p(\mathbf{x})\right]^2 \\
&= d \cdot 2^{d-1} - 2^d \sum_{\mathbf{x}} p^2(\mathbf{x}) \sum_{i=1}^{d} \frac{p(-x_i|\mathbf{x}_{\pi_i})}{p(x_i|\mathbf{x}_{\pi_i})} \prod_{j\neq i} \frac{1}{p(x_j|\mathbf{x}_{\pi_j})} \qquad \text{(B.1)} \\
&= d \cdot 2^{d-1} - 2^d \sum_{i=1}^{d} \sum_{\mathbf{x}} p(\mathbf{x})p(-x_i|\mathbf{x}_{\pi_i}) \\
&= 2^{d-1} \sum_{i=1}^{d} \sum_{\mathbf{x}_{\pi_i}} p(\mathbf{x}_{\pi_i}) \Big[1 - 4p(x_i|\mathbf{x}_{\pi_i})p(-x_i|\mathbf{x}_{\pi_i}) \Big] \geq 0
\end{aligned}$$

Equation (B.1) used the following identity: given real numbers $a_1, a_2, \cdots, a_d$

$$\sum_{S\subseteq[d]} |S| \prod_{i\in S} a_i = \sum_{i=1}^{d} a_i \prod_{j\neq i} (1 + a_j) \qquad \text{(B.2)}$$

□

Appendix C

Proof of Propositions in Chapter 5

C.1 Proof of Proposition 28

Proof.

$$\begin{aligned}\mathrm{Var}(f(\mathbf{x})\phi_S^p(\mathbf{x})) &= \mathbb{E}\Big[\big(f(\mathbf{x})\phi_S^p(\mathbf{x}) - \hat{f}^p(S)\big)^2\Big] = \sum_{\mathbf{x}\in\{-1,1\}^d} p(\mathbf{x}) f^2(\mathbf{x})\big[\phi_S^p(\mathbf{x})\big]^2 - \big[\hat{f}^p(S)\big]^2 \\ &= \sum_{\mathbf{x}\in\{-1,1\}^d} p(\mathbf{x})\big[\phi_S^p(\mathbf{x})\big]^2 - \big[\hat{f}^p(S)\big]^2 = \langle \phi_S^p, \phi_S^p\rangle_p - \big[\hat{f}^p(S)\big]^2 \\ &= \sum_{T\neq S}\big[\hat{f}^p(T)\big]^2\end{aligned}$$

The last equality used Parseval's theorem. □

C.2 Proof of Proposition 29

Proof. For additive noise:

$$\begin{aligned}&\mathrm{Var}\Big(\frac{1}{N}\sum_{n=1}^{N} y^n \phi_S^p(\mathbf{x}^n)\Big) = \frac{1}{N}\mathrm{Var}\big[(f(\mathbf{x}^n) + \epsilon^n)\phi_S^p(\mathbf{x}^n)\big] \\ &= \frac{1}{N}\big[\mathrm{Var}(f(\mathbf{x}^n)\phi_S^p(\mathbf{x}^n)) + \mathrm{Var}(\epsilon^n\phi_S^p(\mathbf{x}^n))\big] = \frac{1}{N}\big[\mathrm{Var}(f(\mathbf{x})\phi_S^p(\mathbf{x})) + \mathrm{Var}(\epsilon)\mathrm{Var}(\phi_S^p(\mathbf{x}))\big] \\ &= \frac{1}{N}\big(\mathrm{Var}(f(\mathbf{x})\phi_S^p(\mathbf{x})) + \sigma^2\big)\end{aligned}$$

The last equality follows from $\mathrm{Var}(\phi_S^p(\mathbf{x})) = \mathbb{E}[(\phi_S^p(\mathbf{x}))^2] - \Big(\mathbb{E}[\phi_S^p(\mathbf{x})]\Big)^2 = 1 - 0 = 1$. For random sign flips:

$$\begin{aligned}
&\mathrm{Var}(\frac{1}{N}\sum_{n=1}^{N} y^n \phi_S^p(\mathbf{x}^n)) \\
&= \frac{1}{N}\mathrm{Var}\big[s f(\mathbf{x})\phi_S^p(\mathbf{x})\big] = \frac{1}{N}\Big(\mathbb{E}[s^2 f^2(\mathbf{x})\big(\phi_S^p(\mathbf{x})\big)^2] - \Big(\mathbb{E}[s f(\mathbf{x})\phi_S^p(\mathbf{x})]\Big)^2\Big) \\
&= \frac{1}{N}\Big(\mathbb{E}[f^2(\mathbf{x})\big(\phi_S^p(\mathbf{x})\big)^2] - \Big(\mathbb{E}[s]\mathbb{E}[f(\mathbf{x})\phi_S^p(\mathbf{x})]\Big)^2\Big) \\
&= \frac{1}{N}\Big(\mathrm{Var}(f(\mathbf{x})\phi_S^p(\mathbf{x})) + \big[\hat{f}^p(S)\big]^2 - (2\rho-1)^2\big[\hat{f}^p(S)\big]^2\Big)
\end{aligned}$$

□

C.3 Proof of Proposition 31

Proof sketch: The general proof for this proposition is long and somewhat tedious. Here we present the proof sketch for a special case where $S = \{1\}$ and $y^n \in \{-1, 1\}$, $\forall n = 1, 2, \cdots, N$, in order to minimize notations. The techniques will apply more generally.

$$W_{\{1\}} \triangleq \frac{1}{N}\sum_{n=1}^{N} y^n x_1^n \sqrt{\frac{\sum_{m=1}^{N} \mathbb{1}(x_1^m = -x_1^n)}{\sum_{m=1}^{N} \mathbb{1}(x_1^m = x_1^n)}}$$

Let us first compute the mean $\mu_{\{1\}}$:

$$\begin{aligned}
\mathbb{E}[W_{\{1\}}] &= \mathbb{E}\Big[y^N x_1^N \sqrt{\frac{\sum_{m=1}^{N} \mathbb{1}(x_1^m = -x_1^N)}{\sum_{m=1}^{N} \mathbb{1}(x_1^m = x_1^N)}}\Big] \\
&= \mathbb{E}_{x_1^N}\Big[x_1^N \mathbb{E}\Big[y^N \sqrt{\frac{\sum_{m=1}^{N} \mathbb{1}(x_1^m = -x_1^N)}{\sum_{m=1}^{N} \mathbb{1}(x_1^m = x_1^N)}}\Big| x_1^N\Big]\Big] \\
&= -p_{x_1}(-1)\mathbb{E}\Big[y^N \sqrt{\frac{N-1-K}{K+1}}\Big| x_1^N = -1\Big] + p_{x_1}(1)\mathbb{E}\Big[y^N \sqrt{\frac{K}{N-K}}\Big| x_1^N = 1\Big] \\
&= \sqrt{p_{x_1}(-1)p_{x_1}(1)}\Big(- \mathbb{E}[y^N | x_1^N = -1] + \mathbb{E}[y^N | x_1^N = 1]\Big) + O(\frac{1}{N}) \\
&= \hat{f}^p(\{1\}) + O(\frac{1}{N})
\end{aligned}$$

Here K denotes the number of -1's in $x_1^1, x_1^2, \cdots, x_1^{N-1}$ and K is distributed as a binomial($p_{x_1}(-1)$, $N-1$).

We follow similar steps to obtain the variance of $W_{\{1\}}$ and can get

$$\mathrm{Var}(W_{\{1\}}) = \mathbb{E}[W_{\{1\}}^2] - \big(\mathbb{E}[W_{\{1\}}]\big)^2 = O(\frac{1}{N})$$

Next we need to establish that $h_S(\mathbf{X}, \mathbf{y})$ converges in distribution to a Gaussian. In particular, we will prove $\sum_{n=1}^N \mathbb{E}[\big|\Delta_n h\big|^3] = O(1/\sqrt{N})$ and $\mathrm{Var}(T) = O(1/N)$.

$$\begin{aligned}\big|\Delta_N h\big| = \frac{1}{N}\Bigg| \sum_{n=1}^{N-1} x_1^n y^n \Bigg(&\sqrt{\frac{\sum_{m=1}^{N-1} \mathbb{1}(x_1^m = -x_1^n) + \mathbb{1}(x_1^m = -x_1^N)}{\sum_{m=1}^{N-1} \mathbb{1}(x_1^m = x_1^n) + \mathbb{1}(x_1^m = x_1^N)}} \\ &- \sqrt{\frac{\sum_{m=1}^{N-1} \mathbb{1}(x_1^m = -x_1^n) + \mathbb{1}(x_1^m = -\tilde{x}_1^N)}{\sum_{m=1}^{N-1} \mathbb{1}(x_1^m = x_1^n) + \mathbb{1}(x_1^m = \tilde{x}_1^N)}}\Bigg) \\ + \Bigg(y^N x_1^N \sqrt{\frac{\sum_{m=1}^{N} \mathbb{1}(x_1^m = -x_1^N)}{\sum_{m=1}^{N} \mathbb{1}(x_1^m = x_1^N)}} &- \tilde{y}^N \tilde{x}_1^N \sqrt{\frac{\sum_{m=1}^{N} \mathbb{1}(x_1^m = -\tilde{x}_1^N)}{\sum_{m=1}^{N} \mathbb{1}(x_1^m = \tilde{x}_1^N)}}\Bigg)\Bigg|\end{aligned}$$

Again let K denote the number of -1's in $x_1^1, x_1^2, \cdots, x_1^{N-1}$. Then it is not difficult to show that

$$\big|\Delta_N h\big| \leq \begin{cases} \frac{c}{N^{1/2}}\sqrt{\frac{N-1-K}{K+1}} & \text{if } x_1^N = \tilde{x}_1^N = -1 \\ \frac{c}{N^{1/2}}\left(\sqrt{\frac{N-1-K}{K+1}} + \sqrt{\frac{K}{N-K}}\right) & \text{if } x_1^N \neq \tilde{x}_1^N \\ \frac{c}{N^{1/2}}\sqrt{\frac{K}{N-K}} & \text{if } x_1^N = \tilde{x}_1^N = 1 \end{cases}$$

$\therefore \sum_{n=1}^N \mathbb{E}[\big|\Delta_n h\big|^3] = N\mathbb{E}\Big[\big|\Delta_N h\big|^3\Big] \leq N \cdot O(1/N^{3/2}) = O(1/\sqrt{N})$

The proof of $\mathrm{Var}(T) = O(1/N)$ uses very similar methods. An important observation is that due to symmetry, the value $\mathbb{E}[\Delta_n h \Delta_n h^A]$ where $n \notin A$ only depends on the cardinality of A.

Bibliography

M. Bain. Structured features from concept lattices for unsupervised learning and classification. *McKay, B., Slaney, J.K. (eds.) Canadian AI 2002. LNCS (LNAI)*, pages 557–568, 2002.

I. Bayer. fastfm: A library for factorization machines. *Journal of Machine Learning Research*, 17(184):1–5, 2016.

M. Ben-Or and N. Linial. Collective coin flipping, robust voting schemes and minima of Banzhaf values. *Proceedings of the 26th Annual IEEE Symposium on Foundations of Computer Science*, pages 408–416, 1985.

E. Blais, R. O'Donnell, and K. Wimmer. Polynomial regression under arbitrary product distributions. *Proc. 21st Conf. on Learning Theory*, pages 193–204, 2008.

L. Breiman. Statistical modelling: The two cultures. *Statistical Science*, 16:199–231, 2001.

G. Bresler and M. Karzand. Learning a tree-structured ising model in order to make predictions. *arXiv preprint arXiv:1604.06749*, 2016.

G. Chandrashekar and F. Sahin. A survey on feature selection methods. *Computers and Electrical Engineering*, 40:16–28, 2014.

S. Chatterjee. A new method of normal approximation. *The Annals of Probability*, 2008.

S. Chatterjee. A short survey of stein's method. *Proceedings of ICM 2014*, 2014.

C. Chow and C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transaction on Information Theory*, 1968.

D. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural network for image classification. *CVPR 2012 Proceedings*, pages 3642–3649, 2012.

T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley-Interscience, New York, 2006.

D. Dua and K. Taniskidou. UCI machine learning repository, 2017.

B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Ann. Stat.*, pages 407–499, 2004.

A. Ehrenfeucht, D. Haussler, and L. Valiant. A general lower bound on the number of examples needed for learning. *information and computation*, 82:247 – 261, 1989.

S. Esmeir and S. Markovitch. Lookahead-based algorithms for anytime induction of decision trees. *Proceedings of the 21st International Conference on Machine Learning*, 2004.

X. Z. Fern and C. E. Brodley. Boosting lazy decision trees. *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.

C. Freudenthaler, L. Schmidt-Thieme, and S. Rendle. Bayesian factorization machines. *Proceedings of the NIPS Workshop on Sparse Representation and Low-Rank Approximation*, 2011.

E.A. Garcia and H. He. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, pages 1263–1284, 2008.

S. Hanneke. The optimal sample complexity of PAC learning. *Journal of Machine Learning Research*, 2016.

A.T. Kalai, A.R. Klivans, Y.Mansour, and R.A. Servedio. Agnostically learning halfspaces. *Proceedings of the 46th Annual Symposium on the Foundations of Computer Science (FOCS)*, 2005.

M. Karzand and G. Bresler. Inferning trees. *Allerton Conf. on Communication, Control and Computing*, 2015.

M. Kearns, R. Schapire, and L. Sellie. Toward efficient agnostic learning. *Machine Learning, 17(2/3)*, pages 115–141, 1994.

A. R. Klivans, R. O'Donnell, and R. A. Servedio. Learning intersections and thresholds of halfspaces. *Journal of Comput. System Sci. 68*, pages 808–840, 2004.

W.S. Lee, P. L. Bartlett, and R.C. Williamson. On efficient agnostic learning of linear combinations of basis functions. *Proceedings of the Eighth Annual Conference on Computational Learning Theory*, pages 369–376, 1995.

N. Linial and Y. Mansour. Constant depth circuits, fourier transform, and learnability. *Journal of the Association for Computing Machinery 40*, pages 607–620, 1993.

N. Merhav and M. Feder. Universal prediction. *IEEE Trans. Inform. Theory 44*, pages 2124–2147, 1998.

A. Ng and M. Jordan. On discriminative vs generative classifiers: A comparison of logistic regression and naive bayes. *Proceedings of Neural Information Processing Systems, 1*, 2002.

R. O'Donnell. *Analysis of Boolean Functions*. Cambridge University Press, 2014.

S. Rendle. Factorization machines. *Proceedings of the 2010 IEEE International Conference on Data Mining*, pages 995 – 1000, 2010.

S. Rendle, Z. Gantner, C. Freudenthaler, and L. Schmidt-Thieme. Fast context-aware recommendations with factorization machines. *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 635–644, 2011.

R. Rubinstein, M. Zibulevsky, and E. Elad. Efficient implementation of the k-svd algorithm using batch orthononal matching pursuit. *CS Technion*, pages 1–15, 2008.

J.E. Savage. *Models of Computation: Exploring the Power of Computing.* Addison-Wesley, 1998.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

A.D. Shapiro. *Structured Induction in Expert Systems.* Addison-Wesley, 1987.

G. Shmueli. To explain or to predict. *Statistical Science*, 25, No.3:289–310, 2010.

R. Tibshirani. Lecture notes for Advanced Methods for Data Analysis, Spring 2014.