Interpretability by Design: New Interpretable Machine Learning Models and Methods

by

Chaofan Chen

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Cynthia Rudin, Advisor

_____
Ronald Parr

_____
Sayan Mukherjee

_____
Hai Li

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2020

Abstract

Interpretability by Design:
New Interpretable Machine Learning Models and Methods

by

Chaofan Chen

Department of Computer Science
Duke University

Date: _____

Approved:

_____
Cynthia Rudin, Advisor

_____
Ronald Parr

_____
Sayan Mukherjee

_____
Hai Li

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2020

Abstract

As machine learning models are playing increasingly important roles in many real-life scenarios, interpretability has become a key issue for whether we can trust the predictions made by these models, especially when we are making some high-stakes decisions. Lack of transparency has long been a concern for predictive models in criminal justice and in healthcare. There have been growing calls for building interpretable, human understandable machine learning models, and "opening the black box" has become a debated issue in the media. My dissertation research addresses precisely the demand for interpretability and transparency in machine learning models. The key problem of this dissertation is: "Can we build machine learning models that are both accurate and interpretable?"

To address this problem, I will discuss the notion of interpretability as it relates to machine learning, and present several new interpretable machine learning models and methods I developed during my dissertation research. In Chapter 1, I will discuss two types of model interpretability – predicate-based and case-based interpretability. In Chapters 2 and 3, I will present novel predicate-based interpretable models and methods, and their applications to understanding low-dimensional structured data. In particular, Chapter 2

presents falling rule lists, which extend regular decision lists by requiring the probabilities of the desired outcome to be monotonically decreasing down the list; Chapter 3 presents two-layer additive models, which are hybrids of predicate-based additive scoring models and small neural networks. In Chapter 4, I will present case-based interpretable deep models, and their applications to computer vision. Given the empirical evidence, I conclude in Chapter 5 that, by designing novel model architectures or regularization techniques, we can build machine learning models that are both accurate and interpretable.

Acknowledgements

This work was supported by a dissertation committee consisting of Professor Cynthia Rudin (advisor) of the Department of Computer Science, Professor Ronald Parr of the Department of Computer Science, Professor Sayan Mukherjee of the Department of Statistics, and Professor Hai Li of the Department of Electrical and Computer Engineering. The experiments and visualizations in Chapter 3 were conducted in part by Kangcheng Lin, Sijia Wang, Professor Yaron Shaposhnik (the University of Rochester), and Professor Tong Wang (the University of Iowa). The experiments in Chapter 4 were conducted in part by Oscar Li, Hao Liu, and Chaofan Tao. Some figures in Chapter 4 were made in part by Alina Jade Barnett. All other work conducted for the dissertation was completed by me independently. This dissertation work was supported in part by a grant from MIT Lincoln Laboratory to Professor Cynthia Rudin.

I am especially indebted to my advisor Professor Cynthia Rudin, who has been supportive of my career goals and who worked actively to provide guidance during my dissertation research. I am grateful to all of my collaborators with whom I had the pleasure to work during my graduate study. I would especially like to thank Oscar Li, who worked closely

with me in coding and running experiments.

I would also like to thank the staff at the Department of Computer Science. I am indebted to Marilyn Butler, who helped me through the most difficult times during my graduate study.

Finally, I would like to thank my parents, Xinrong Chen and Rong Chen, who have showered me with love and supported me generously, especially during my undergraduate study. I am grateful to my loving wife, Songyao Ren, who provides unending love and inspiration for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As machine learning models are playing increasingly important roles in many real-life scenarios, interpretability has become a key issue for whether we can trust the predictions made by these models, especially when we are making some high-stakes decisions. In the realm of law and justice, there have been cases where incorrect data fed into black box models have gone unnoticed, leading to unfairly long prison sentences (e.g., prisoner Glen Rodriguez was denied parole due to an incorrect COMPAS score, [Wex17]). In healthcare, doctors rarely make risk assessments based on results from "black box" models – they often want to know the conditions that signify a high risk of stroke, for example, so that patients with such conditions can be prioritized in receiving treatment [WR15]. Lack of transparency has long been a concern for clinical decision support systems based on machine learning, in particular, deep learning models – the U.S. Food and Drug Administration (FDA) now requires all such systems to provide "the rationale or support for the recommendation" [Edw17]. In light of these issues, there have been growing calls for building interpretable, human understandable

machine learning models [AVW+18, Fre14, HDM+11, Kod94, MB10], and "opening the black box" has become a debated issue in the media [Cit16, Smi16, ALMK16, Wes17].

My dissertation research addresses precisely the demand for interpretability and transparency in machine learning models. The key problem of this dissertation is: "Can we build machine learning models that are both accurate and interpretable?" To address this problem, I will begin with a discussion of the notion of interpretability as it relates to machine learning. In the course of this discussion, I will survey the traditional approach to interpretability, and highlight the novelty and the contributions of my dissertation research to the field of interpretable machine learning.

1.1 Model Interpretability: What Is It?

An interpretable machine learning model is one whose reasoning process can be explained and understood by humans. In general, there are two types of interpretable models: the first type makes its predictions based on *predicates* (i.e., logical expressions that evaluate to true or false); the second type makes its predictions based on known *similar cases*. I call the first type *predicate-based interpretable models*, and the second type *case-based interpretable models*.

1.1.1 Predicate-based Interpretability

A predicate-based interpretable model makes its predictions using predicates, which are logical expressions that evaluate to true or false. Examples of predicate-based interpretable

models include decision trees, decision lists, and additive scoring models. Given an unseen test instance, (1) a decision tree classifies the instance by tracing the path of nodes whose predicates are true until reaching a leaf node; (2) a decision list classifies the instance by evaluating the predicate in each decision rule until finding a rule whose predicate evaluates to true; (3) an additive scoring model classifies the instance by summing the scores corresponding to the predicates that evaluate to true. As we can see, evaluations of predicates are integral to decision trees, decision lists, and additive scoring models.

Traditionally, the methods of learning decision trees and decision lists use greedy algorithms (e.g., CART [BFSO84], ID3 [Qui86], C4.5 [Qui93], and C5.0 [Qui04], and greedy decision list learning [Riv87]). In my dissertation research, I developed an optimization approach to learning a special class of (probabilistic) decision lists (called falling rule lists) [CR18], as an alternative to greedy algorithms. A falling rule list is a probabilistic decision list for binary classification, consisting of a series of *if-then* rules with predicates (also called *antecedents*) in the *if* clauses and probabilities of the desired outcome ("1") in the *then* clauses, where the probabilities of the desired outcome ("1") are monotonically decreasing down the list (hence the name "falling" rule list). The monotonicity constraint in the probabilities provides added interpretability for falling rule lists, compared to regular decision lists or decision trees: in the case of a falling rule list, users only need to check a small number of conditions to determine whether an observation is in a high-risk or high-probability subgroup; however, in the case of a regular decision list or decision tree,

it is possible that high-risk subgroups are in different parts of the list/tree, so that users may have to check many conditions in order to find high-risk subgroups. The form of a falling rule list was proposed by Wang and Rudin [WR15], and they designed a Bayesian framework to learning falling rule lists. In my research, I proved a set of tight bounds that can effectively limit the size of the search space, whereas in the original Bayesian approach, there are no tight bounds on optimal solutions to restrict the search space – even if we constructed bounds for the original Bayesian approach, they would involve loose approximations to gamma functions. In particular, the bounds I proved include a *necessary condition of optimality* and a *prefix bound*. The former tells us what kind of falling rule lists cannot be optimal solutions and can therefore be eliminated during search. The latter tells us the best objective value we can achieve with a partially constructed falling rule list (called a *prefix*), so that we can apply branch-and-bound techniques (e.g., if the best objective value we can achieve with a prefix is worse than the objective value of the current best solution, we can safely exclude any falling rule list that begins with this prefix from the search space). The result of my paper is a computationally more efficient algorithm for falling rule list learning, compared to the original Bayesian approach.

Additive scoring models are widely used in risk assessment. In the past, such models were often created by domain experts. For example, the $CHADS_2$ score for predicting the atrial fibrillation stroke risk was created by a group of medical professionals [GWS+01]. There are recent works that used machine learning techniques to create medical scoring

systems [UR16, UR17]. In my dissertation research, I extended traditional additive scoring models by partitioning features into meaningful subgroups (called *subscales*) and inserting more nonlinearities. The result is a *two-layer additive risk model* that is more expressive than traditional scoring models. In particular, a two-layer additive risk model is a hybrid of traditional additive scoring models and a small (two-layer) neural network. In terms of model architecture, a two-layer additive model partitions the original features into meaningful subgroups (called *subscales*), and transforms the original features into piecewise-constant risk-scoring functions that are combined within each subscale to form the subscale models in the first layer of the model. Each subscale model can function like a "mini-scoring model" that produces its own probability of risk (much like a traditional scoring model). At the same time, the combination of the subscale probabilities in the second layer resembles the computation in a neural network layer with sigmoid nonlinearity.

1.1.2 Case-based Interpretability

A case-based interpretable model makes its predictions using similarities to known cases. Traditional case-based interpretable models include k-nearest neighbors [MPP09, WS09] and various prototype models [PMDS03, BT11, WT17, KRS14]. Given an unseen test instance, (1) a k-nearest neighbors model classifies the instance by considering the class labels of similar training instances; (2) a prototype model classifies the instance by considering the class labels of similar prototypical instances. As we can see, comparisons with known similar cases are integral to k-nearest neighbors and prototype models.

Computer vision poses specific challenges when it comes to interpretability. Predicate-based interpretable models (e.g., decision trees, decision lists) are often insufficient for handling high-dimensional image data, particularly because the dimensions (the pixel values) themselves do not mean much to human beings. On the other hand, it is more natural for humans to reason about images using visual similarities. For example, when we try to identify the species of a bird, we would often say that "this bird is a sparrow because its head looks like a sparrow's head." However, visual similarities are far from similarities in terms of pixel values, and we cannot directly apply traditional case-based interpretable models to images, because these models would compare images using similarities in terms of pixel values.

In my dissertation research, I quantified the notion of visual similarities, and designed deep neural networks – *prototype network* (PrototypeNet) and *prototypical part network* (ProtoPNet), that are able to reason about images by saying "this looks like that." These networks are able to learn a meaningful latent embedding space that captures the notion of visual similarities, and a set of prototypical cases for comparison. Given a new image, they are able to identify similar prototypical cases using distances in the latent space, and make predictions according to the known class labels of those prototypical cases.

1.2 Organization of This Dissertation

In Chapter 1, I have discussed two types of model interpretability – predicate-based and case-based interpretability, surveyed traditional approaches to both types of model interpretability, and highlighted the contributions of my dissertation research. The remaining of my dissertation is organized as follows. In Chapters 2 and 3, I will present novel predicate-based interpretable models and methods, and their applications to understanding low-dimensional structured data. In particular, Chapter 2 presents falling rule lists, which extend regular decision lists by requiring the probabilities of the desired outcome to be monotonically decreasing down the list; Chapter 3 presents two-layer additive models, which are hybrids of predicate-based additive scoring models and small neural networks. In Chapter 4, I will present case-based interpretable deep models, and their applications to computer vision. Given the empirical evidence, I conclude in Chapter 5 that, by designing novel model architectures or regularization techniques, we can build machine learning models that are both accurate and interpretable.

Chapter 2

Predicate-based Interpretability: Falling Rule Lists

In many real-life scenarios, we want to learn a predictive model that allows us to easily identify the most significant conditions that are predictive of a certain outcome. For example, in healthcare, doctors often want to know the conditions that signify a high risk of stroke, so that patients with such conditions can be prioritized in receiving treatment. A falling rule list, whose form was first proposed by Wang and Rudin [WR15], is a predicate-based interpretable model that serves this purpose.

Table 2.1 shows a falling rule list we learned from the bank-full dataset, which was used by Moro et al. [MLC11] in their study of applying data mining techniques to direct marketing. As we can see, a falling rule list is a probabilistic decision list for binary classification, consisting of a series of **if-then** rules with antecedents in the **if** clauses and probabilities of the desired outcome ("1") in the **then** clauses, where the probabilities of the desired outcome ("1") are monotonically decreasing down the list (hence the name "falling"

Table 2.1: Falling rule list for bank-full dataset

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND default=no	THEN success prob. is	0.28	434	1113
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1539
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.23	242	794
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.14	658	4092
ELSE IF	previous ≥ 2 AND education=tertiary	THEN success prob. is	0.13	108	707
ELSE		success prob. is	0.07	2365	31146

rule list). The falling rule list in Table 2.1 has identified clients for whom the previous marketing campaign was successful ("poutcome=success"), and who have no credit in default ("default=no"), as individuals who are most likely to subscribe to a term deposit in the current marketing campaign. Their probability of subscribing is 0.65. Of the remaining clients, those who are next most likely to sign up for a term deposit are older people (aged between 60 and 100) with no credit in default. Their probability of subscribing is 0.28. The two rightmost columns in Table 2.1, labeled + and −, show the number of positive training examples (i.e. clients who subscribe to a term deposit in the current campaign) and of negative training examples, respectively, that satisfy the antecedent in each rule of the falling rule list.

Falling rule lists can provide valuable insight into data – if we know how to construct them well. In this chapter, I will present a novel optimization approach to learning falling rule lists and "softly" falling rule lists, which I developed during my dissertation research.

The materials in this chapter were presented at the 21st International Conference on Artificial Intelligence and Statistics (AISTATS 2018), and have been published in the *Proceedings of Machine Learning Research*, Volume 84 [CR18].

2.1 Related Work

My dissertation work on falling rule lists lives within several well-established fields, but is the first work to use an optimization approach to handling monotonicity constraints in rule-based models. It relates closely to associative classification (e.g. the RIPPERk algorithm [Coh95] and the CBA algorithm [LHM98]; see [Tha07] for a comprehensive review) and inductive logic programming [MDR94]. The proposed algorithms are competitors for decision tree methods like CART [BFSO84], ID3 [Qui86], C4.5 [Qui93], and C5.0 [Qui04], and decision list learning [Riv87]. Almost all methods from this class build decision trees from the top down using greedy splitting criteria. Greedy splitting criteria do not lend naturally to constrained models like falling rule lists. There are some works on machine learning methods that enforce monotonicity constraints (e.g., [ARD05, BD95, FP03]), but they focus mostly on enforcing the monotonic relationship between certain attributes and ordinal class labels.

Wang and Rudin [WR15] proposed the form of a falling rule list, and a Bayesian approach to learning falling rule lists (extending the ideas of [LRMM15] and [YRS17]). The Bayesian approach offers some advantages: e.g. a full posterior over rule lists allows

model averaging. However, the optimization perspective has an important computational advantage: the search space is made substantially smaller by the tight bounds presented here. The concept of softly falling rule lists is novel to my dissertation work and has not been done in the Bayesian setting.

2.2 Problem Formulation

We first formalize the notion of an antecedent, of a rule list, of a falling rule list, and of a prefix.

Definition 2.2.1. An *antecedent* a on an input domain $\mathcal{X}$ is a Boolean function that outputs true or false. Given an input $\mathbf{x} \in \mathcal{X}$, we say that $\mathbf{x}$ satisfies the antecedent a if $a(\mathbf{x})$ evaluates to true. For example, (poutcome=success AND default=no) in Table 2.1 is an antecedent.

Definition 2.2.2. A *rule list* $d : \mathcal{X} \to [0, 1]$ on an input domain $\mathcal{X}$ is a probabilistic decision list of the following form:

IF $\mathbf{x}$ satisfies $a_0^{(d)}$, THEN $\Pr(y = 1|\mathbf{x}) = \hat{\alpha}_0^{(d)}$
ELSE IF $\mathbf{x}$ satisfies $a_1^{(d)}$, THEN $\Pr(y = 1|\mathbf{x}) = \hat{\alpha}_1^{(d)}$
...
ELSE IF $\mathbf{x}$ satisfies $a_{|d|-1}^{(d)}$, THEN $\Pr(y = 1|\mathbf{x}) = \hat{\alpha}_{|d|-1}^{(d)}$
ELSE $\Pr(y = 1|\mathbf{x}) = \hat{\alpha}_{|d|}^{(d)}$

where $a_j^{(d)}$ is the j-th antecedent in d, $j \in \{0, 1, ..., |d| - 1\}$, and $|d|$ denotes the size of the

rule list, which is defined as the number of rules, excluding the final else clause, in the rule list. We can denote the rule list d as follows:

$$d = \{(a_0^{(d)}, \hat{\alpha}_0^{(d)}), (a_1^{(d)}, \hat{\alpha}_1^{(d)}), ..., (a_{|d|-1}^{(d)}, \hat{\alpha}_{|d|-1}^{(d)}), \hat{\alpha}_{|d|}^{(d)}\}. \tag{2.1}$$

The rule list d of Equation (2.1) is a *falling rule list* if the following inequalities hold:

$$\hat{\alpha}_0^{(d)} \geq \hat{\alpha}_1^{(d)} \geq ... \geq \hat{\alpha}_{|d|-1}^{(d)} \geq \hat{\alpha}_{|d|}^{(d)}. \tag{2.2}$$

For convenience, we sometimes refer to the final else clause in d as the $|d|$-th antecedent $a_{|d|}^{(d)}$ in d, which is satisfied by all $\mathbf{x} \in \mathcal{X}$. We denote the space of all possible rule lists on $\mathcal{X}$ by $\mathcal{D}(\mathcal{X})$.

Definition 2.2.3. A *prefix* e on an input domain $\mathcal{X}$ is a rule list without the final else clause. We can denote the prefix e as follows:

$$e = \{(a_0^{(e)}, \hat{\alpha}_0^{(e)}), (a_1^{(e)}, \hat{\alpha}_1^{(e)}), ..., (a_{|e|-1}^{(e)}, \hat{\alpha}_{|e|-1}^{(e)})\}. \tag{2.3}$$

where $a_j^{(e)}$ is the j-th antecedent in e, $j \in \{0, 1, ..., |e| - 1\}$, and $|e|$ denotes the size of the prefix, which is defined as the number of rules in the prefix.

Definition 2.2.4. Given the rule list d of Equation (2.1) (or the prefix e of Equation (2.3)), we say that an input $\mathbf{x} \in \mathcal{X}$ is *captured* by the j-th antecedent in d (or e) if $\mathbf{x}$ satisfies $a_j^{(d)}$ (or $a_j^{(e)}$, respectively), and for all $k \in \{0, 1, ..., |d|\}$ (or $k \in \{0, 1, ..., |e| - 1\}$, respectively) such that $\mathbf{x}$ satisfies $a_k^{(d)}$ (or $a_k^{(e)}$, respectively), $j \leq k$ holds – in other words, $a_j^{(d)}$ (or $a_j^{(e)}$, respectively) is the first antecedent that $\mathbf{x}$ satisfies. We define the function

capt by $\text{capt}(\mathbf{x}, d) = j$ (or $\text{capt}(\mathbf{x}, e) = j$) if $\mathbf{x}$ is captured by the j-th antecedent in d (or e). Moreover, given the prefix e of Equation (2.3), we say that an input $\mathbf{x} \in \mathcal{X}$ is captured by the prefix e if $\mathbf{x}$ is captured by some antecedent in e, and we define $\text{capt}(\mathbf{x}, e) = |e|$ if $\mathbf{x}$ is not captured by the prefix e.

Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ be the training data, with $\mathbf{x}_i \in \mathcal{X}$ and $y_i \in \{1, -1\}$ for each $i \in \{1, 2, ..., n\}$. We now define the empirical positive proportion of an antecedent, and introduce the notion of a rule list (or a prefix) that is compatible with D.

Definition 2.2.5. Given the training data D and the rule list d of Equation (2.1) (or the prefix e of Equation (2.3)), we denote by $n^{+}_{j,d,D}$, $n^{-}_{j,d,D}$, $n_{j,d,D}$ (or $n^{+}_{j,e,D}$, $n^{-}_{j,e,D}$, $n_{j,e,D}$), the number of positive, negative, and all training inputs captured by the j-th antecedent in d (or e), respectively. We define the *empirical positive proportion* of the j-th antecedent in d, denoted by $\alpha_j^{(d,D)}$, as:

$$\alpha_j^{(d,D)} = \frac{n^{+}_{j,d,D}}{n_{j,d,D}}.$$

The empirical positive proportion of the j-th antecedent in e, denoted by $\alpha_j^{(e,D)}$, is defined analogously by:

$$\alpha_j^{(e,D)} = \frac{n^{+}_{j,e,D}}{n_{j,e,D}}.$$

Moreover, given the training data D and the prefix e of Equation (2.3), we denote by $\tilde{n}^{+}_{e,D}$, $\tilde{n}^{-}_{e,D}$, $\tilde{n}_{e,D}$, the number of positive, negative, and all training inputs that are not captured by the prefix e, and define the *empirical positive proportion after the prefix* e, denoted by $\tilde{\alpha}_{e,D}$, as $\tilde{\alpha}_{e,D} = \tilde{n}^{+}_{e,D}/\tilde{n}_{e,D}$.

Definition 2.2.6. Given the training data D and the rule list d of Equation (2.1) (or the prefix e of Equation (2.3)), we say that the rule list d (or the prefix e) is *compatible* with D if for all $j \in \{0, 1, ..., |d|\}$ (or $j \in \{0, 1, ..., |e| - 1\}$, respectively), the equation $\hat{\alpha}_j^{(d)} = \alpha_j^{(d,D)}$ ($\hat{\alpha}_j^{(e)} = \alpha_j^{(e,D)}$, respectively) holds. We denote the space of all possible rule lists on $\mathcal{X}$ that are compatible with the training data D by $\mathcal{D}(\mathcal{X}, D)$.

To formulate the problem of learning falling rule lists from data as an optimization program, we first observe that, given a threshold τ, the rule list d of Equation (2.1) can be viewed as a classifier $\tilde{d}_\tau : \mathcal{X} \to \{1, -1\}$ that predicts 1 for an input $\mathbf{x} \in \mathcal{X}$ only if the inequality $\hat{\alpha}^{(d)}_{\text{capt}(\mathbf{x},d)} > \tau$ holds. Hence, we can define the empirical risk of misclassification by the rule list d on the training data D as that by the classifier $\tilde{d}_\tau$. More formally, we have the following definition.

Definition 2.2.7. Given the training data D, the rule list d of Equation (2.1), a threshold τ, and the weight w for the positive class, the *empirical risk of misclassification by the rule list* d on the training data D with threshold τ and with weight w for the positive class, denoted by $R(d, D, \tau, w)$, is:

$$R(d, D, \tau, w) = \frac{1}{n} \left(w \sum_{i:y_i=1} \mathbb{1}[\hat{\alpha}^{(d)}_{\text{capt}(\mathbf{x}_i,d)} \leq \tau] + \sum_{i:y_i=-1} \mathbb{1}[\hat{\alpha}^{(d)}_{\text{capt}(\mathbf{x}_i,d)} > \tau] \right). \tag{2.4}$$

If d is compatible with D, we can replace $\hat{\alpha}^{(d)}_{\text{capt}(\mathbf{x}_i,d)}$ in Equation (2.4) with $\alpha^{(d,D)}_{\text{capt}(\mathbf{x}_i,d)}$. We define the *empirical risk of misclassification by the prefix* e on the training data D with

threshold τ and with weight w for the positive class, denoted by $R(e, D, \tau, w)$, analogously:

$$R(e, D, \tau, w) = \frac{1}{n}\left(w \sum_{\substack{i:y_i=1\wedge \\ \text{capt}(\mathbf{x}_i,e)\neq|e|}} \mathbb{1}[\hat{\alpha}^{(e)}_{\text{capt}(\mathbf{x}_i,e)} \leq \tau] + \sum_{\substack{i:y_i=-1\wedge \\ \text{capt}(\mathbf{x}_i,e)\neq|e|}} \mathbb{1}[\hat{\alpha}^{(e)}_{\text{capt}(\mathbf{x}_i,e)} > \tau]\right). \tag{2.5}$$

If e is compatible with D, we can replace $\hat{\alpha}^{(e)}_{\text{capt}(\mathbf{x}_i,e)}$ in Equation (2.5) with $\alpha^{(e,D)}_{\text{capt}(\mathbf{x}_i,e)}$. Note that for any rule list d that begins with a given prefix e, $R(e, D, \tau, w)$ is the contribution by the prefix e to $R(d, D, \tau, w)$.

We can formulate the problem of learning falling rule lists as a minimization program of the empirical risk of misclassification, given by Equation (2.4), with a regularization term $C|d|$ that penalizes each rule in d with a cost of C to limit the number of rules, subject to the monotonicity constraint (2.2). For now, we focus on the problem of learning falling rule lists that are compatible with the training data D.

Let $L(d, D, \tau, w, C) = R(d, D, \tau, w) + C|d|$ and $L(e, D, \tau, w, C) = R(e, D, \tau, w) + C|e|$ be the regularized empirical risk of misclassification by the rule list d and by the prefix e, respectively, on the training data D. The former defines the objective of the minimization program, and the latter gives the contribution by the prefix e to $L(d, D, \tau, w, C)$ for any rule list d that begins with e. The following theorem provides a motivation for setting the threshold τ to $1/(1+w)$ in the minimization program – the empirical risk of misclassification by a given rule list d is minimized when τ is set in this way.

Theorem 2.2.8. *Given the training data D, a rule list d that is compatible with D, and*

the weight w for the positive class, we have $R(d, D, 1/(1+w), w) \leq R(d, D, \tau, w)$ for all $\tau \geq 0$.

Proof. Suppose $\tau > 1/(1+w)$. Consider the j-th rule $(a_j^{(d)}, \alpha_j^{(d,D)})$ in d, whose antecedent captures $\alpha_j^{(d,D)} n_{j,d,D}$ positive training inputs and $(1-\alpha_j^{(d,D)}) n_{j,d,D}$ negative training inputs. Let $R_j(d, D, \tau, w)$ denote the contribution by the j-th rule to $R(d, D, \tau, w)$, i.e.

$$\begin{aligned} R_j(d, D, \tau, w) &= \frac{1}{n} \left(w \sum_{\substack{i: y_i = 1 \wedge \\ \text{capt}(\mathbf{x}_i, d) = j}} \mathbb{1}[\alpha_j^{(d,D)} \leq \tau] + \sum_{\substack{i: y_i = -1 \wedge \\ \text{capt}(\mathbf{x}_i, d) = j}} \mathbb{1}[\alpha_j^{(d,D)} > \tau] \right) \\ &= \begin{cases} \frac{1}{n} n_{j,d,D}^- & \text{if } \alpha_j^{(d,D)} > \tau \\ \frac{w}{n} n_{j,d,D}^+ & \text{otherwise.} \end{cases} \end{aligned} \tag{2.6}$$

Case 1. $1/(1+w) < \alpha_j^{(d,D)} \leq \tau$. In this case, we have

$$\begin{aligned} & R_j(d, D, 1/(1+w), w) \\ &= \frac{1}{n} n_{j,d,D}^- \quad \text{(by the definition of } R_j \text{ in Equation (2.6))} \\ &= \frac{1}{n} (n_{j,d,D} - n_{j,d,D}^+) \quad \text{(by the definition of } n_{j,d,D}^+, n_{j,d,D}^-, n_{j,d,D} \text{ in Definition 2.2.5)} \\ &= \frac{1}{n} (n_{j,d,D} - \alpha_j^{(d,D)} n_{j,d,D}) \quad \text{(by the definition of } \alpha_j^{(d,D)} \text{ in Definition 2.2.5)} \\ &= \frac{1}{n} (1 - \alpha_j^{(d,D)}) n_{j,d,D} \\ &< \frac{1}{n} \left(1 - \frac{1}{1+w} \right) n_{j,d,D} \\ &= \frac{w}{n} \frac{1}{1+w} n_{j,d,D} \\ &< \frac{w}{n} \alpha_j^{(d,D)} n_{j,d,D} \\ &= \frac{w}{n} n_{j,d,D}^+ \quad \text{(by the definition of } \alpha_j^{(d,D)} \text{ in Definition 2.2.5)} \end{aligned}$$

$= R_j(d, D, \tau, w)$. (by the definition of R_j in Equation (2.6))

Case 2. $\alpha_j^{(d,D)} > \tau$. In this case, both $R_j(d, D, 1/(1+w), w)$ and $R_j(d, D, \tau, w)$ are equal to $\frac{1}{n} n^-_{j,d,D}$.

Case 3. $\alpha_j^{(d,D)} \leq 1/(1+w)$. In this case, both $R_j(d, D, 1/(1+w), w)$ and $R_j(d, D, \tau, w)$ are equal to $\frac{w}{n} n^+_{j,d,D}$.

Hence, given $\tau > 1/(1+w)$, we have

$$\begin{aligned} R(d, D, 1/(1+w), w) &= \sum_{j=0}^{|d|} R_j(d, D, 1/(1+w), w) \\ &\leq \sum_{j=0}^{|d|} R_j(d, D, \tau, w) = R(d, D, \tau, w). \end{aligned}$$

The proof for $R(d, D, 1/(1+w), w) \leq R(d, D, \tau, w)$ given $\tau < 1/(1+w)$ is similar. □

For reasons of computational tractability and model interpretability, we further restrict our attention to learning compatible falling rule lists whose antecedents must come from a pre-determined set of antecedents $A = \{A_l\}_{l=1}^m$. We now present the optimization program for learning falling rule lists, which forms the basis of the rest of this paper.

Program 2.2.9 (Learning compatible falling rule lists)**.**

$$\min_{d \in \mathcal{D}(\mathcal{X}, D)} L(d, D, 1/(1+w), w, C) \text{ subject to}$$

$$\alpha_0^{(d,D)} \geq \alpha_1^{(d,D)} \geq ... \geq \alpha_{|d|-1}^{(d,D)} \geq \alpha_{|d|}^{(d,D)}, \tag{2.7}$$

$$a_j^{(d)} \in A, \text{ for all } j \in \{0, 1, ..., |d| - 1\}. \tag{2.8}$$

The constraint (2.7) is exactly the monotonicity constraint (2.2) for the falling rule lists that are compatible with D. The constraint (2.8) limits the choice of antecedents. An instance of Program 2.2.9 is defined by the tuple (D, A, w, C).

2.3 Algorithm FRL

In this section, we present a Monte-Carlo search algorithm, Algorithm FRL, based on Program 2.2.9, for learning compatible falling rule lists from data. Given an instance (D, A, w, C) of Program 2.2.9, the algorithm constructs a compatible falling rule list d in each iteration, while keeping track of the falling rule list d^* that has the smallest objective value $L_{\text{best}} = L(d^*, D, \tau, w, C)$ among all the falling rule lists that the algorithm has constructed so far. At the end of T iterations, the algorithm outputs the falling rule list that has the smallest objective value out of the T lists it has constructed.

In the process of constructing a falling rule list d, the algorithm chooses the antecedents successively: first for the antecedent $a_0^{(d)}$ in the top rule, then for the antecedent $a_1^{(d)}$ in the next rule, and so forth. For each antecedent $a_j^{(d)}$ chosen, the algorithm also computes its empirical positive proportion $\alpha_j^{(d,D)}$. After p rules have been constructed so that d currently holds the prefix $e = \{(a_0^{(d)}, \alpha_0^{(d,D)}), (a_1^{(d)}, \alpha_1^{(d,D)}), ..., (a_{p-1}^{(d)}, \alpha_{p-1}^{(d,D)})\}$, the algorithm either: (1) terminates the construction of d by computing the empirical positive proportion after e, $\tilde{\alpha}_{e,D}$, and then adding to d the final else clause with probability estimate $\tilde{\alpha}_{e,D}$, or (2) randomly picks an antecedent from a candidate set S of possible next antecedents, computes

its empirical positive proportion, and uses these as the next rule $(a_p^{(d)}, \alpha_p^{(d,D)})$ for d.

The algorithm uses various properties of Program 2.2.9, which are presented in Section 2.4, to prune the search space. More specifically, the algorithm terminates the construction of d if Inequality (2.11) in Theorem 2.4.6 holds. Otherwise it either terminates the construction of d with some probability, or proceeds to construct a candidate set S of possible next antecedents, as follows. For every antecedent $A_l \in A$ that has not been chosen before, it constructs a candidate next rule $(a_p^{(d)}, \alpha_p^{(d,D)})$ by setting $a_p^{(d)} = A_l$ and computing $\alpha_p^{(d,D)}$ using Definition 2.2.5. The algorithm then checks if the monotonicity constraint $\alpha_p^{(d,D)} \leq \alpha_{p-1}^{(d,D)}$ and the necessary condition for optimality $\alpha_p^{(d,D)} > 1/(1+w)$ (Corollary 2.4.5) are satisfied, if the prefix $e' = \{e, (a_p^{(d)}, \alpha_p^{(d,D)})\}$ is feasible under Program 2.2.9 (i.e. whether there exists a compatible falling rule list that begins with the prefix e') using Proposition 2.4.2, and if the best possible objective value $L^*(e', D, w, C)$ achievable by any falling rule list that begins with e' and is compatible with D (Theorem 2.4.6) is less than the current best objective value $L_{\text{best}} = L(d^*, D, 1/(1+w), w, C)$. If all of the above conditions are satisfied, the algorithm adds A_l to S. Once the construction of S is complete, the algorithm randomly chooses an antecedent $A_l \in S$ with probability $P(A_l|S, e, D)$ and uses this antecedent, together with its empirical positive proportion, as the next rule $(a_p^{(d)}, \alpha_p^{(d,D)})$ for d. If S is empty, the algorithm terminates the construction of d.

In practice, we define the probability $P(A_l|S, e, D)$ for $A_l \in S$ by first defining a curiosity

function $f_{S,e,D} : S \to \mathbb{R}_{\geq 0}$ and then normalizing it:

$$P(A_l|S, e, D) = \frac{f_{S,e,D}(A_l)}{\sum_{A_{l'}} f_{S,e,D}(A_{l'})}.$$

A possible choice of the curiosity function $f_{S,e,D}$ for use in Algorithm FRL is given by

$$f_{S,e,D}(A_l) = \lambda\alpha(A_l, e, D) + (1 - \lambda)\frac{n^+(A_l, e, D)}{\tilde{n}^+_{e,D}}, \tag{2.9}$$

where $\alpha(A_l, e, D)$ is the empirical positive proportion of A_l, and $n^+(A_l, e, D)$ is the number of positive training inputs captured by A_l, should A_l be chosen as the next antecedent after the prefix e. The curiosity function $f_{S,e,D}$ given by (2.9) is a weighted sum of $\alpha(A_l, e, D)$ and $n^+(A_l, e, D)/\tilde{n}^+_{e,D}$ for each $A_l \in S$: the former encourages the algorithm to choose antecedents that have large empirical positive proportions, and the latter encourages the algorithm to choose antecedents that have large positive supports in the training data not captured by e. We used this curiosity function for Algorithm FRL in our experiments.

The pseudocode of Algorithm FRL is shown in Algorithm 1.

2.4 Prefix Bound

The goal of this section is to find a lower bound on the objective value of any compatible falling rule list that begins with a given compatible prefix, which we call a *prefix bound*, and to prove the various results used in the algorithm. To derive this prefix bound, we first introduce the concept of a feasible prefix, with which it is possible to construct a compatible falling rule list from data.

Input: an instance (D, A, w, C) of Program 2.2.9
Result: a falling rule list d^* that are compatible with D and whose antecedents come from A
initialize $d^* = \emptyset$, $L_{\text{best}} = \infty$;
for $t = 1, ..., T$ **do**
  set $p = -1$, $\alpha_p = 1$, $d = e = \emptyset$;
  while *Inequality (9) in Theorem 2.4.6 does not hold* **do**
    go to Terminate with some probability;
    set $p = p + 1$, $S = \emptyset$;
    for *every antecedent* $A_l \in A$ *that is not in* d **do**
      set $a_p^{(d)} = A_l$, compute $\alpha_p^{(d,D)}$, and let $e' = \{e, (a_p^{(d)}, \alpha_p^{(d,D)})\}$;
      if $\alpha_p^{(d,D)} \leq \alpha_{p-1}^{(d,D)}$, $\alpha_p^{(d,D)} > 1/(1+w)$, *and* e' *is feasible under Program 2.2.9* **then**
        compute $L^*(e', D, w, C)$ using Theorem 2.4.6;
        if $L^*(e', D, w, C) < L(d^*, D, 1/(1+w), w, C)$ **then**
          add A_l to S;
        end
      end
    end
    if $S \neq \emptyset$ **then**
      choose an antecedent $A_l \in S$ with probability $P(A_l|S, e, D)$ according to a discrete probability distribution over S;
      set $a_p^{(d)} = A_l$ and add $(a_p^{(d)}, \alpha_p^{(d,D)})$ to d;
      set $e = d$;
      // save the partially constructed list d as the prefix e
    else
      go to Terminate
    end
  end
  Terminate: terminate the construction of d, and compute $L(d, D, 1/(1+w), w, C)$;
  if $L(d, D, 1/(1+w), w, C) < L_{best}$ **then**
    set $d^* = d$, $L_{\text{best}} = L(d, D, 1/(1+w), w, C)$;
  end
end

Algorithm 1: Algorithm FRL

Figure 2.1: Algorithm FRL

Definition 2.4.1. Given the training data D and the set of antecedents A, a prefix e is feasible for Program 2.2.9 under the training data D and the set of antecedents A if e is compatible with D, and there exists a falling rule list d such that d is compatible with D, the antecedents of d come from A, and d begins with e.

The following proposition gives necessary and sufficient conditions for a prefix e to be feasible.

Proposition 2.4.2. *Given the training data D, the set of antecedents A, and a prefix e that is compatible with D and satisfies $a_j^{(e)} \in A$ for all $j \in \{0, 1, ..., |e| - 1\}$ and $\alpha_{k-1}^{(e,D)} \geq \alpha_k^{(e,D)}$ for all $k \in \{1, 2, ..., |e| - 1\}$, the following statements are equivalent: (1) e is feasible for Program 2.2.9 under D and A; (2) $\tilde{\alpha}_{e,D} \leq \alpha_{|e|-1}^{(e,D)}$ holds; (3) $\tilde{n}_{e,D}^- \geq ((1/\alpha_{|e|-1}^{(e,D)}) - 1)\tilde{n}_{e,D}^+$ holds.*

Proof. (1) $\Rightarrow$ (3): Suppose that Statement (1) holds. Then there exists a falling rule list

$$d = \{e, (a_{|e|}^{(d)}, \alpha_{|e|}^{(d,D)}), ..., (a_{|d|-1}^{(d)}, \alpha_{|d|-1}^{(d,D)}), \alpha_{|d|}^{(d,D)}\}$$

that is compatible with D, and we have

$$\begin{aligned}
\tilde{n}_{e,D}^- &= \tilde{n}_{e,D} - \tilde{n}_{e,D}^+ \\
&= n_{|e|,d,D} + ... + n_{|d|,d,D} - \tilde{n}_{e,D}^+ \\
&= \frac{1}{\alpha_{|e|}^{(d,D)}} n_{|e|,d,D}^+ + ... + \frac{1}{\alpha_{|d|}^{(d,D)}} n_{|d|,d,D}^+ - \tilde{n}_{e,D}^+ \quad \text{(by Definition 2.2.5)} \\
&\geq \frac{1}{\alpha_{|e|-1}^{(d,D)}} n_{|e|,d,D}^+ + ... + \frac{1}{\alpha_{|e|-1}^{(d,D)}} n_{|d|,d,D}^+ - \tilde{n}_{e,D}^+ \quad \text{(by the monotonicity constraint)}
\end{aligned}$$

$$= \frac{1}{\alpha_{|e|-1}^{(d,D)}}(n_{|e|,d,D}^{+} + ... + n_{|d|,d,D}^{+}) - \tilde{n}_{e,D}^{+}$$

$$= \frac{1}{\alpha_{|e|-1}^{(d,D)}}\tilde{n}_{e,D}^{+} - \tilde{n}_{e,D}^{+}$$

$$= ((1/\alpha_{|e|-1}^{(d,D)}) - 1)\tilde{n}_{e,D}^{+}$$

$$= ((1/\alpha_{|e|-1}^{(e,D)}) - 1)\tilde{n}_{e,D}^{+}.$$

(3) $\Rightarrow$ (2): Suppose that Statement (3) holds. Then we have

$$\tilde{\alpha}_{e,D} = \frac{\tilde{n}_{e,D}^{+}}{\tilde{n}_{e,D}} \quad \text{(by Definition 2.2.5)}$$

$$= \frac{\tilde{n}_{e,D}^{+}}{\tilde{n}_{e,D}^{+} + \tilde{n}_{e,D}^{-}}$$

$$\leq \frac{\tilde{n}_{e,D}^{+}}{\tilde{n}_{e,D}^{+} + ((1/\alpha_{|e|-1}^{(d,D)}) - 1)\tilde{n}_{e,D}^{+}} \quad \text{(by Statement (3))}$$

$$= \frac{\tilde{n}_{e,D}^{+}}{(1 + (1/\alpha_{|e|-1}^{(d,D)}) - 1)\tilde{n}_{e,D}^{+}} = \alpha_{|e|-1}^{(d,D)}.$$

(2) $\Rightarrow$ (1): Suppose that Statement (2) holds. Then the falling rule list $d = \{e, \tilde{\alpha}_{e,D}\}$ begins with e and is compatible with D. By Definition 2.4.1, e is feasible for Program 2.2.9 under the training data D. □

We now introduce the concept of a hypothetical rule list, whose antecedents do not need to come from the pre-determined set of antecedents A.

Definition 2.4.3. Given a pre-determined set of antecedents A, a hypothetical rule list with respect to A is a rule list that contains an antecedent that is not in A.

We need the following lemma to prove the necessary condition for optimality (Corollary 2.4.5), and to derive a prefix bound (Theorem 2.4.6).

Lemma 2.4.4. *Suppose that we are given an instance (D, A, w, C) of Program 2.2.9, a prefix e that is feasible for Program 2.2.9 under D and A, and a (possibly hypothetical) falling rule list d that begins with e and is compatible with D. Then there exists a falling rule list d', possibly hypothetical with respect to A, such that d' begins with e, has at most one more rule (excluding the final else clause) following e, is compatible with D, and satisfies*

$$L(d', D, 1/(1+w), w, C) \leq L(d, D, 1/(1+w), w, C).$$

As a special case, if either $\alpha_j^{(d,D)} > 1/(1+w)$ holds for all $j \in \{|e|, |e|+1, ..., |d|\}$, or $\alpha_j^{(d,D)} \leq 1/(1+w)$ holds for all $j \in \{|e|, |e|+1, ..., |d|\}$, then the falling rule list $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$ (i.e. the falling rule list in which the final else clause follows immediately the prefix e, and the probability estimate of the final else clause is $\tilde{\alpha}_{e,D}$) is compatible with D and satisfies $L(\bar{e}, D, 1/(1+w), w, C) \leq L(d, D, 1/(1+w), w, C)$.

A consequence of the above lemma is that an optimal solution for a given instance (D, A, w, C) of Program 2.2.9 should not have any antecedent whose empirical positive proportion falls below $1/(1+w)$.

Corollary 2.4.5. *If d^* is an optimal solution for a given instance (D, A, w, C) of Program 2.2.9, then we must have $\alpha_j^{(d^*,D)} > 1/(1+w)$ for all $j \in \{0, 1, ..., |d^*| - 1\}$.*

Another implication of Lemma 2.4.4 is that the objective value of any compatible falling rule list that begins with a given prefix e cannot be less than a lower bound on the objective

value of any compatible falling rule list that begins with the same prefix e, and has at most one more rule (excluding the final else clause) following e. This leads to the following theorem.

Theorem 2.4.6. *Suppose that we are given an instance (D, A, w, C) of Program 2.2.9 and a prefix e that is feasible for Program 2.2.9 under D and A. Then any falling rule list d that begins with e and is compatible with D satisfies*

$$L(d, D, 1/(1+w), w, C) \geq L^*(e, D, w, C),$$

where

$$\begin{aligned} L^*(e, D, w, C) &= L(e, D, 1/(1+w), w, C) + \\ &\min\left(\frac{1}{n}\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+} + C, \frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right) \end{aligned} \tag{2.10}$$

is a lower bound on the objective value of any compatible falling rule list that begins with e, under the instance (D, A, w, C) of Program 2.2.9. We call $L^*(e, D, w, C)$ the prefix bound for e. Further, if

$$C \geq \min\left(\frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right) - \frac{1}{n}\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+} \tag{2.11}$$

holds, then the compatible falling rule list $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$, where the prefix e is followed directly by the final else clause, satisfies $L(\bar{e}, D, 1/(1+w), w, C) = L^*(e, D, w, C)$.

The results presented in this section are used in Algorithm FRL to prune the search space. The proofs of Lemma 2.4.4 and Corollary 2.4.5 are presented in Section 2.4.1, and the proof of Theorem 2.4.6 is presented in Section 2.4.2.

2.4.1 Proofs of Lemma 2.4.4 and Corollary 2.4.5

Before we proceed with proving Lemma 2.4.4, we make the following observation.

Observation 2.4.7. *For any rule list*

$$d' = \{e, (a_{|e|}^{(d')}, \hat{\alpha}_{|e|}^{(d')}), ..., (a_{|d'|-1}^{(d')}, \hat{\alpha}_{|d'|-1}^{(d')}), \hat{\alpha}_{|d'|}^{(d')}\}$$

that begins with a given prefix e, we have

$$\tilde{n}_{e,D}^{+} = n_{|e|,d',D}^{+} + ...n_{|d'|,d',D}^{+}, \tag{2.12}$$

$$\tilde{n}_{e,D}^{-} = n_{|e|,d',D}^{-} + ...n_{|d'|,d',D}^{-}, \tag{2.13}$$

and

$$\tilde{n}_{e,D} = n_{|e|,d',D} + ...n_{|d'|,d',D}. \tag{2.14}$$

Proof. Any positive training input $\mathbf{x}_i$ that is not captured by the prefix e must be captured by some antecedent $a_j^{(d')}$ with $|e| \leq j < |d'|$ in d', or the final else clause in d'. Conversely, any positive training input $\mathbf{x}_i$ that is captured by some antecedent $a_j^{(d')}$ with $|e| \leq j < |d'|$ in d', or the final else clause in d', must not satisfy any antecedent in the prefix e and is consequently not captured by the prefix e. This means that the set of positive training inputs that are not captured by e is exactly the set of positive training inputs that are captured by some antecedent $a_j^{(d')}$ with $|e| \leq j < |d'|$ in d', or the final else clause in d'. It then follows that these two sets have the same number of elements. The former set has $\tilde{n}_{e,D}^{+}$ number of elements, and the latter has $n_{|e|,d',D}^{+} + ... + n_{|d'|,d',D}^{+}$ number of elements. This establishes Equation (2.12).

We can establish Equations (2.13) and (2.14) using essentially the same argument. □

We now prove Lemma 2.4.4.

Proof of Lemma 2.4.4. **Case 1.** There exists some $k \in \{|e|+1, ..., |d|\}$ that satisfies $\alpha_{k-1}^{(d,D)} > 1/(1+w)$ but $\alpha_k^{(d,D)} \leq 1/(1+w)$. For any $j \in \{|e|, ..., k-1\}$, we have $\alpha_j^{(d,D)} > 1/(1+w)$, and the contribution $R_j(d, D, 1/(1+w), w)$ by the j-th rule to $R(d, D, 1/(1+w), w)$, defined by Equation (2.6) with $\tau = 1/(1+w)$, is given by

$$R_j(d, D, 1/(1+w), w) = \frac{1}{n} n_{j,d,D}^{-}. \tag{2.15}$$

For any $j \in \{k, ..., |d|\}$, we have $\alpha_j^{(d,D)} \leq 1/(1+w)$, and the contribution $R_j(d, D, 1/(1+w), w)$ by the j-th rule to $R(d, D, 1/(1+w), w)$ is given by

$$R_j(d, D, 1/(1+w), w) = \frac{w}{n} n_{j,d,D}^{+}. \tag{2.16}$$

The rest of the proof for this case proceeds in three steps.

Step 1. Construct a hypothetical falling rule list d' that begins with e, has exactly one more rule (excluding the final else clause) following e, and is compatible with D. In later steps, we shall show that the falling rule list d' constructed in this step satisfies $L(d', D, 1/(1+w), w, C) \leq L(d, D, 1/(1+w), w, C)$.

Let $d' = \{e, (a_{|e|}^{(d')}, \hat{\alpha}_{|e|}^{(d')}), \hat{\alpha}_{|e|+1}^{(d')}\}$ be the falling rule list of size $|d'| = |e| + 1$ that is compatible with D, such that

$$a_{|e|}^{(d')} = a_{|e|}^{(d)} \vee ... \vee a_{k-1}^{(d)}$$

is the antecedent given by the logical **or**'s of the antecedents $a_{|e|}^{(d)}$ through $a_{k-1}^{(d)}$ in d.

Step 2. Show that the empirical risk of misclassification by the falling rule list d' is the same as that by the falling rule list d.

To see this, we observe that the training instances captured by $a_{|e|}^{(d')}$ in d' are exactly those captured by the antecedents $a_{|e|}^{(d)}$ through $a_{k-1}^{(d)}$ in d, and the training instances captured by $a_{|e|+1}^{(d')}$ (i.e. the final else clause) in d' are exactly those captured by the antecedents $a_{k}^{(d)}$ through $a_{|d|}^{(d)}$ in d. This observation implies

$$n^{+}_{|e|,d',D} = n^{+}_{|e|,d,D} + ... + n^{+}_{k-1,d,D}, \tag{2.17}$$

$$n^{-}_{|e|,d',D} = n^{-}_{|e|,d,D} + ... + n^{-}_{k-1,d,D}, \tag{2.18}$$

$$n_{|e|,d',D} = n_{|e|,d,D} + ... + n_{k-1,d,D}, \tag{2.19}$$

$$n^{+}_{|e|+1,d',D} = n^{+}_{k,d,D} + ... + n^{+}_{|d|,d,D}, \tag{2.20}$$

and

$$n_{|e|+1,d',D} = n_{k,d,D} + ... + n_{|d|,d,D}. \tag{2.21}$$

Since d' is compatible with D, using the definition of a compatible rule list in Definition 2.2.6 and the definition of the empirical positive proportion in Definition 2.2.5, together with (2.17), (2.19), (2.20), and (2.21), we must have

$$\begin{aligned}\hat{\alpha}_{|e|}^{(d')} = \alpha_{|e|}^{(d',D)} = \frac{n^{+}_{|e|,d',D}}{n_{|e|,d',D}} &= \frac{n^{+}_{|e|,d,D} + ... + n^{+}_{k-1,d,D}}{n_{|e|,d,D} + ... + n_{k-1,d,D}} \\ &= \frac{\alpha_{|e|}^{(d,D)} n_{|e|,d,D} + ... + \alpha_{k-1}^{(d,D)} n_{k-1,d,D}}{n_{|e|,d,D} + ... + n_{k-1,d,D}} > \frac{1}{1+w},\end{aligned}$$

and

$$\hat{\alpha}^{(d')}_{|e|+1} = \alpha^{(d',D)}_{|e|+1} = \frac{n^+_{|e|+1,d',D}}{n_{|e|+1,d',D}} = \frac{n^+_{k,d,D} + ... + n^+_{|d|,d,D}}{n_{k,d,D} + ... + n_{|d|,d,D}}$$

$$= \frac{\alpha^{(d,D)}_k n_{k,d,D} + ... + \alpha^{(d,D)}_{|d|} n_{|d|,d,D}}{n_{k,d,D} + ... + n_{|d|,d,D}} \leq \frac{1}{1+w}.$$

This means that the contribution $R_{|e|}(d', D, 1/(1+w), w)$ by the $|e|$-th rule to $R(d', D, 1/(1+w), w)$ is given by

$$R_{|e|}(d', D, 1/(1+w), w) = \frac{1}{n} n^-_{|e|,d',D} = \frac{1}{n}(n^-_{|e|,d,D} + ... + n^-_{k-1,d,D}),$$

where we have used (2.18), and the contribution $R_{|e|+1}(d', D, 1/(1+w), w)$ by the $(|e|+1)$-st "rule" (i.e. the final else clause) to $R(d', D, 1/(1+w), w)$ is given by

$$R_{|e|+1}(d', D, 1/(1+w), w) = \frac{w}{n} n^+_{|e|+1,d',D} = \frac{w}{n}(n^+_{k,d,D} + ... + n^+_{|d|,d,D}),$$

where we have used (2.20).

It then follows that the empirical risk of misclassification by the rule list d' is the same as that by the rule list d:

$$\begin{aligned}
&R(d', D, 1/(1+w), w) \\
&= R(e, D, 1/(1+w), w) + R_{|e|}(d', D, 1/(1+w), w) + R_{|e|+1}(d', D, 1/(1+w), w) \\
&= R(e, D, 1/(1+w), w) + \frac{1}{n}(n^-_{|e|,d,D} + ... + n^-_{k-1,d,D}) + \frac{w}{n}(n^+_{k,d,D} + ... + n^+_{|d|,d,D}) \\
&= R(e, D, 1/(1+w), w) + \sum_{j=|e|}^{|d|} R_j(d, D, 1/(1+w), w) \\
&= R(d, D, 1/(1+w), w). \qquad (2.22)
\end{aligned}$$

Step 3. Put everything together.

Using (2.22), together with the observation $|d'| = |e| + 1 \leq |d|$, we must also have

$$L(d', D, 1/(1+w), w, C) = R(d', D, 1/(1+w), w) + C|d'|$$

$$\leq R(d, D, 1/(1+w), w) + C|d| = L(d, D, 1/(1+w), w, C),$$

as desired.

Case 2. $\alpha_j^{(d,D)} > 1/(1+w)$ holds for all $j \in \{|e|, |e|+1, ..., |d|\}$. Then the contribution $R_j(d, D, 1/(1+w), w)$ by the j-th rule to $R(d, D, 1/(1+w), w)$, for all $j \in \{|e|, |e|+1, ..., |d|\}$, is given by Equation (2.15). Let $d' = \{e, \hat{\alpha}_{|e|}^{(d')}\}$ be the falling rule list of size $|d'| = |e|$ that is compatible with D. Then the instances captured by $a_{|e|}^{(d')}$ (i.e. the final else clause) in d' are exactly those that are not captured by e, or equivalently, those that are captured by $a_{|e|}^{(d)}$ through $a_{|d|}^{(d)}$. This implies

$$n^+_{|e|,d',D} = n^+_{|e|,d,D} + ... + n^+_{|d|,d,D}, \tag{2.23}$$

$$n^-_{|e|,d',D} = n^-_{|e|,d,D} + ... + n^-_{|d|,d,D}, \tag{2.24}$$

and

$$n_{|e|,d',D} = n_{|e|,d,D} + ... + n_{|d|,d,D}. \tag{2.25}$$

Since d' is compatible with D, using the definition of a compatible rule list in Definition 2.2.6 and the definition of the empirical positive proportion in Definition 2.2.5 , together with (2.23) and (2.25), we must have

$$\hat{\alpha}_{|e|}^{(d')} = \alpha_{|e|}^{(d',D)} = \frac{n^+_{|e|,d',D}}{n_{|e|,d',D}}$$

$$= \frac{n^+_{|e|,d,D} + ... + n^+_{|d|,d,D}}{n_{|e|,d,D} + ... + n_{|d|,d,D}} \tag{2.26}$$

$$= \frac{\alpha^{(d,D)}_{|e|} n_{|e|,d,D} + ... + \alpha^{(d,D)}_{|d|} n_{|d|,d,D}}{n_{|e|,d,D} + ... + n_{|d|,d,D}} > \frac{1}{1+w}. \tag{2.27}$$

Note that the right-hand side of Equality (2.26) is equal to $\tilde{n}^+_{e,D}/\tilde{n}_{e,D} = \tilde{\alpha}_{e,D}$, by Equations (2.12) and (2.14) in Observation 2.4.7. Therefore, we also have $\hat{\alpha}^{(d')}_{|e|} = \tilde{\alpha}_{e,D}$.

Inequality (2.27) implies that the contribution $R_{|e|}(d', D, 1/(1+w), w)$ by the $|e|$-th "rule" (i.e. the final else clause) to $R(d', D, 1/(1+w), w)$ is given by

$$R_{|e|}(d', D, 1/(1+w), w) = \frac{1}{n} n^-_{|e|,d',D} = \frac{1}{n}(n^-_{|e|,d,D} + ... + n^-_{|d|,d,D}),$$

where we have used (2.24).

It then follows that the empirical risk of misclassification by the rule list d' is the same as that by the rule list d:

$$\begin{aligned}
&R(d', D, 1/(1+w), w) \\
&= R(e, D, 1/(1+w), w) + R_{|e|}(d', D, 1/(1+w), w) \\
&= R(e, D, 1/(1+w), w) + \frac{1}{n}(n^-_{|e|,d,D} + ... + n^-_{|d|,d,D}) \\
&= R(e, D, 1/(1+w), w) + \sum_{j=|e|}^{|d|} R_j(d, D, 1/(1+w), w) \\
&= R(d, D, 1/(1+w), w).
\end{aligned}$$

Since we clearly have $|d'| = |e| \leq |d|$, we must also have

$$\begin{aligned}
L(d', D, 1/(1+w), w, C) &= R(d', D, 1/(1+w), w) + C|d'| \\
&\leq R(d, D, 1/(1+w), w) + C|d| = L(d, D, 1/(1+w), w, C),
\end{aligned}$$

as desired.

Case 3. $\alpha_j^{(d,D)} \leq 1/(1+w)$ holds for all $j \in \{|e|, |e|+1, ..., |d|\}$. The proof is similar to Case 2, with $R_j(d, D, 1/(1+w), w)$ for all $j \in \{|e|, |e|+1, ..., |d|\}$ given by Equation (2.16), the "greater than" in Inequality 2.27 replaced by "less than or equal to", and $R_{|e|}(d', D, 1/(1+w), w)$ given by

$$R_{|e|}(d', D, 1/(1+w), w) = \frac{w}{n} n^+_{|e|,d',D} = \frac{w}{n}(n^+_{|e|,d,D} + ... + n^+_{|d|,d,D}).$$

□

Corollary 2.4.5 follows from Lemma 2.4.4.

Proof of Corollary 2.4.5. Suppose that d^* were an optimal solution for a given instance (D, A, w, C) of Program 2.2.9, such that $\alpha_k^{(d^*,D)} \leq 1/(1+w)$ form some $k \in \{0, 1, ..., |d^*|-1\}$. Let

$$e = \{(a_0^{(d^*)}, \alpha_0^{(d^*,D)}), ..., (a_{k-1}^{(d^*)}, \alpha_{k-1}^{(d^*,D)})\}$$

be a prefix consisting of the top k rules in d^*. By Lemma 2.4.4, the falling rule list $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$ satisfies $L(\bar{e}, D, 1/(1+w), w, C) \leq L(d^*, D, 1/(1+w), w, C)$. In fact, the inequality is strict because the size of $\bar{e}$ is strictly less than that of d^*. This contradicts the optimality of d^*. □

2.4.2 Proof of Theorem 2.4.6

Before we proceed with proving Theorem 2.4.6, we make two other observations.

Observation 2.4.8. *For any rule list d', we have*

$$n^{-}_{|e|,d',D} = \left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 \right) n^{+}_{|e|,d',D}, \tag{2.28}$$

Proof. By Definition 2.2.5, we have

$$\alpha^{(d',D)}_{|e|} = n^{+}_{|e|,d',D} / n_{|e|,d',D}.$$

Since $n_{|e|,d',D}$ denotes the total number of training inputs captured by the $|e|$-th antecedent in d', which is exactly the sum of the number of positive training inputs captured by that antecedent (denoted $n^{+}_{|e|,d',D}$), and the number of negative training inputs captured by the same antecedent (denoted $n^{-}_{|e|,d',D}$), we have

$$\alpha^{(d',D)}_{|e|} = \frac{n^{+}_{|e|,d',D}}{n^{+}_{|e|,d',D} + n^{-}_{|e|,d',D}}.$$

The desired equation follows from rearranging the terms. □

Observation 2.4.9. *For any rule list*

$$d' = \{e, (a^{(d')}_{|e|}, \hat{\alpha}^{(d')}_{|e|}), \hat{\alpha}^{(d')}_{|e|+1}\}$$

that has exactly one rule (excluding the final else clause) following a given prefix e, we have

$$n^{+}_{|e|+1,d',D} = \tilde{n}^{+}_{e,D} - n^{+}_{|e|,d',D}, \tag{2.29}$$

$$n^{-}_{|e|+1,d',D} = \tilde{n}^{-}_{e,D} - n^{-}_{|e|,d',D}, \tag{2.30}$$

and

$$n_{|e|+1,d',D} = \tilde{n}_{e,D} - n_{|e|,d',D}. \tag{2.31}$$

Note that since $n^+_{|e|+1,d',D}$, $n^-_{|e|+1,d',D}$, *and* $n_{|e|+1,d',D}$ *are non-negative, Equations (2.29), (2.30), and (2.31) imply* $n^+_{|e|,d',D} \leq \tilde{n}^+_{e,D}$, $n^-_{|e|,d',D} \leq \tilde{n}^-_{e,D}$, *and* $n_{|e|,d',D} \leq \tilde{n}_{e,D}$.

Proof. Applying Observation 2.4.7 with $|d'| = |e| + 1$, we have

$$\tilde{n}^+_{e,D} = n^+_{|e|,d',D} + n^+_{|e|+1,d',D},$$

$$\tilde{n}^-_{e,D} = n^-_{|e|,d',D} + n^-_{|e|+1,d',D},$$

and

$$\tilde{n}_{e,D} = n_{|e|,d',D} + n_{|e|+1,d',D}.$$

Equations (2.29), (2.30), and (2.31) follow from rearranging the terms in the above equations.

□

We now prove Theorem 2.4.6.

Proof of Theorem 2.4.6. Let $\mathcal{F}(\mathcal{X}, D, e)$ be the set of falling rule lists (including both hypothetical and non-hypothetical falling rule lists) that begin with e and are compatible with D, and let $\mathcal{F}(\mathcal{X}, D, e, k)$ be the subset of $\mathcal{F}(\mathcal{X}, D, e)$, consisting of those falling rule lists in $\mathcal{F}(\mathcal{X}, D, e)$ that have exactly k rules (excluding the final else clause) following the prefix e.

Let $d \in \mathcal{F}(\mathcal{X}, D, e)$.

Case 1. $\alpha^{(e,D)}_{|e|-1} > 1/(1+w)$.

In this case, Lemma 2.4.4 implies

$$L(d, D, 1/(1+w), w, C) \geq \inf_{d' \in \mathcal{F}(\mathcal{X},D,e,1) \cup \mathcal{F}(\mathcal{X},D,e,0)} L(d', D, 1/(1+w), w, C). \tag{2.32}$$

Note that we have $\mathcal{F}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$, where $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$ is the falling rule list in which the final else clause immediately follows the prefix e, and the probability estimate of the final else clause is $\tilde{\alpha}_{e,D}$. To see this, we first observe $\bar{e} \in \mathcal{F}(\mathcal{X}, D, e, 0)$. This is because:

(i) $\bar{e}$ clearly begins with e, and has no additional rules (excluding the final else clause) following the prefix e;

(ii) the feasibility of e implies $\alpha_{k-1}^{(e,D)} \geq \alpha_{k}^{(e,D)}$ for all $k \in \{1, 2, ..., |e| - 1\}$ (otherwise we could not possibly have a falling rule list that begins with e, and we would violate Definition 2.4.1), and $\tilde{\alpha}_{e,D} \leq \alpha_{|e|-1}^{(e,D)}$ (by Proposition 2.4.2), which together imply that $\bar{e}$ is indeed a falling rule list; and

(iii) we have

$$\begin{aligned}
\tilde{\alpha}_{e,D} &= \frac{\tilde{n}^{+}_{e,D}}{\tilde{n}_{e,D}} \quad \text{(by the definition of } \tilde{\alpha}_{e,D} \text{ in Definition 2.2.5)} \\
&= \frac{n^{+}_{|\bar{e}|,\bar{e},D}}{n_{|\bar{e}|,\bar{e},D}} \quad \text{(by Equations (2.12) and (2.14) in Observation 2.4.7, applied to } \bar{e}) \\
&= \alpha_{|\bar{e}|}^{(\bar{e},D)}, \quad \text{(by definition of empirical positive proportion in Definition 2.2.5)}
\end{aligned}$$

which implies that $\bar{e}$ is indeed compatible with D.

Conversely, for any $d_0 = \{e, \hat{\alpha}_{|e|}^{(d_0)}\} \in \mathcal{F}(\mathcal{X}, D, e, 0)$, we must have

$$\begin{aligned}
\hat{\alpha}_{|e|}^{(d_0)} &= \alpha_{|e|}^{(d_0,D)} \quad \text{(because } d_0 \text{ must be compatible with } D) \\
&= \frac{n^{+}_{|e|,d_0,D}}{n_{|e|,d_0,D}} \quad \text{(by definition of empirical positive proportion in Definition 2.2.5)} \\
&= \frac{\tilde{n}^{+}_{e,D}}{\tilde{n}_{e,D}} \quad \text{(by Equations (2.12) and (2.14) in Observation 2.4.7, applied to } d_0) \\
&= \tilde{\alpha}_{e,D},
\end{aligned}$$

which implies $d_0 = \bar{e}$. This establishes $\mathcal{F}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$.

Let $\mathcal{F}'(\mathcal{X}, D, e, 1)$ be the subset of $\mathcal{F}(\mathcal{X}, D, e, 1)$, consisting of those falling rule lists

$$d' = \{e, (a_{|e|}^{(d')}, \alpha_{|e|}^{(d',D)}), \alpha_{|e|+1}^{(d',D)}\} \in \mathcal{F}(\mathcal{X}, D, e, 1)$$

with $\alpha_{|e|}^{(d',D)} > 1/(1+w)$ and $\alpha_{|e|+1}^{(d',D)} \leq 1/(1+w)$. Note that for any

$$d_1 = \{e, (a_{|e|}^{(d_1)}, \alpha_{|e|}^{(d_1,D)}), \alpha_{|e|+1}^{(d_1,D)}\} \in \mathcal{F}(\mathcal{X}, D, e, 1) - \mathcal{F}'(\mathcal{X}, D, e, 1),$$

we have either $\alpha_{|e|}^{(d_1,D)} \geq \alpha_{|e|+1}^{(d_1,D)} > 1/(1+w)$ or $\alpha_{|e|+1}^{(d_1,D)} \leq \alpha_{|e|}^{(d_1,D)} \leq 1/(1+w)$, and Lemma 2.4.4 implies $L(d_1, D, 1/(1+w), w, C) \geq L(\bar{e}, D, 1/(1+w), w, C)$. This means

$$\inf_{d' \in \mathcal{F}(\mathcal{X},D,e,1) - \mathcal{F}'(\mathcal{X},D,e,1)} L(d', D, 1/(1+w), w, C) \geq L(\bar{e}, D, 1/(1+w), w, C). \tag{2.33}$$

Using $\mathcal{F}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$ and (2.33), we can write the right-hand side of (2.32) as

$$\begin{aligned}
&\inf_{d' \in \mathcal{F}(\mathcal{X},D,e,1) \cup \mathcal{F}(\mathcal{X},D,e,0)} L(d', D, 1/(1+w), w, C) \\
&= \inf_{d' \in \mathcal{F}'(\mathcal{X},D,e,1) \cup (\mathcal{F}(\mathcal{X},D,e,1) - \mathcal{F}'(\mathcal{X},D,e,1)) \cup \{\bar{e}\}} L(d', D, 1/(1+w), w, C) \\
&= \min \Bigg(\inf_{d' \in \mathcal{F}'(\mathcal{X},D,e,1)} L(d', D, 1/(1+w), w, C), \\
&\qquad \inf_{d' \in \mathcal{F}(\mathcal{X},D,e,1) - \mathcal{F}'(\mathcal{X},D,e,1)} L(d', D, 1/(1+w), w, C), L(\bar{e}, D, 1/(1+w), w, C) \Bigg) \\
&= \min \left(\inf_{d' \in \mathcal{F}'(\mathcal{X},D,e,1)} L(d', D, 1/(1+w), w, C), L(\bar{e}, D, 1/(1+w), w, C) \right).
\end{aligned} \tag{2.34}$$

The rest of the proof for this case proceeds in three steps.

Step 1. Compute $L(\bar{e}, D, 1/(1+w), w, C)$.

Since the contribution by the final else clause to $L(\bar{e}, D, 1/(1+w), w, C)$ is given by

$$R_{|e|}(\bar{e}, D, 1/(1+w), w) = \begin{cases} \frac{1}{n} n^-_{|e|,\bar{e},D} & \text{if } \tilde{\alpha}_{e,D} > 1/(1+w) \\ \frac{w}{n} n^+_{|e|,\bar{e},D} & \text{otherwise,} \end{cases}$$

where we have used Equation (2.6), and since Observation 2.4.7 implies $\tilde{n}^+_{e,D} = n^+_{|e|,\bar{e},D}$ and $\tilde{n}^-_{e,D} = n^-_{|e|,\bar{e},D}$, it is not difficult to see

$$L(\bar{e}, D, 1/(1+w), w, C) = \begin{cases} L(e, D, 1/(1+w), w, C) + \frac{1}{n}\tilde{n}^-_{e,D} & \text{if } \tilde{\alpha}_{e,D} > 1/(1+w) \\ L(e, D, 1/(1+w), w, C) + \frac{w}{n}\tilde{n}^+_{e,D} & \text{otherwise.} \end{cases}$$

Since $\tilde{\alpha}_{e,D} > 1/(1+w)$ is equivalent to $\tilde{n}^+_{e,D}/(\tilde{n}^+_{e,D} + \tilde{n}^-_{e,D}) > 1/(1+w)$, or $w\tilde{n}^+_{e,D} > \tilde{n}^-_{e,D}$, and similarly $\tilde{\alpha}_{e,D} \leq 1/(1+w)$ is equivalent to $w\tilde{n}^+_{e,D} \leq \tilde{n}^-_{e,D}$, we can write

$$\begin{aligned} L(\bar{e}, D, 1/(1+w), w, C) = \min \Bigg(& L(e, D, 1/(1+w), w, C) + \frac{1}{n}\tilde{n}^-_{e,D}, \\ & L(e, D, 1/(1+w), w, C) + \frac{w}{n}\tilde{n}^+_{e,D} \Bigg). \end{aligned} \tag{2.35}$$

Step 2. Determine a lower bound of $L(d', D, 1/(1+w), w, C)$ for all $d' \in \mathcal{F}'(\mathcal{X}, D, e, 1)$.

Let $d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{F}'(\mathcal{X}, D, e, 1)$. Since the contribution by both the $|e|$-th rule and the final else clause to $L(d', D, 1/(1+w), w, C)$ is given by $R_{|e|}(d', D, 1/(1+w), w) + R_{|e|+1}(d', D, 1/(1+w), w) + C$, where $R_{|e|}(d', D, 1/(1+w), w)$ and $R_{|e|+1}(d', D, 1/(1+w), w)$ are defined by Equation (2.6) and are given by

$$R_{|e|}(d', D, 1/(1+w), w) = \frac{1}{n} n^-_{|e|,d',D} \quad \text{and} \quad R_{|e|+1}(d', D, 1/(1+w), w) = \frac{w}{n} n^+_{|e|+1,d',D}$$

(because we have $\alpha^{(d',D)}_{|e|} > 1/(1+w)$ and $\alpha^{(d',D)}_{|e|+1} \leq 1/(1+w)$ for $d' \in \mathcal{F}'(\mathcal{X}, D, e, 1)$), it is

not difficult to see

$$L(d', D, 1/(1+w), w, C) = L(e, D, 1/(1+w), w, C) + \frac{1}{n} n^{-}_{|e|,d',D} + \frac{w}{n} n^{+}_{|e|+1,d',D} + C. \quad (2.36)$$

Substituting (2.28) in Observation 2.4.8 and (2.29) in Observation 2.4.9 into Equation (2.36), we have

$$\begin{aligned}
& L(d', D, 1/(1+w), w, C) \\
&= L(e, D, 1/(1+w), w, C) + \frac{1}{n} \left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 \right) n^{+}_{|e|,d',D} + \frac{w}{n} (\tilde{n}^{+}_{e,D} - n^{+}_{|e|,d',D}) + C \\
&= L(e, D, 1/(1+w), w, C) + \frac{1}{n} \left(\left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 - w \right) n^{+}_{|e|,d',D} + w \tilde{n}^{+}_{e,D} \right) + C. \quad (2.37)
\end{aligned}$$

Note that Equation (2.37) shows that given the prefix e, $L(d', D, 1/(1+w), w, C)$ is a function of $\alpha^{(d',D)}_{|e|}$ and of $n^{+}_{|e|,d',D}$. Since we have

$$\frac{\partial L(d', D, 1/(1+w), w, C)}{\partial n^{+}_{|e|,d',D}} = \frac{1}{n} \left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 - w \right) < 0$$

because $\alpha^{(d',D)}_{|e|} > 1/(1+w)$ holds for any $d' \in \mathcal{F}'(\mathcal{X}, D, e, 1)$, and

$$\frac{\partial L(d', D, 1/(1+w), w, C)}{\partial \alpha^{(d',D)}_{|e|}} = -\frac{n^{+}_{|e|,d',D}}{n} \frac{1}{(\alpha^{(d',D)}_{|e|})^2} \leq 0,$$

we see that $L(d', D, 1/(1+w), w, C)$ is indeed a monotonically decreasing function of both $n^{+}_{|e|,d',D}$ and $\alpha^{(d',D)}_{|e|}$. Thus, we can obtain a lower bound of $L(d', D, 1/(1+w), w, C)$ by substituting $n^{+}_{|e|,d',D}$ and $\alpha^{(d',D)}_{|e|}$ with their respective upper bound. The inequality $n^{+}_{|e|,d',D} \leq \tilde{n}^{+}_{e,D}$ in Observation 2.4.9 gives an upper bound for $n^{+}_{|e|,d',D}$, and the inequality

$\alpha_{|e|}^{(d',D)} \leq \alpha_{|e|-1}^{(d',D)} = \alpha_{|e|-1}^{(e,D)}$ from d' being a falling rule list gives an upper bound for $\alpha_{|e|}^{(d',D)}$. Substituting these upper bounds into (2.37), we obtain the following inequality, which gives a lower bound of $L(d', D, 1/(1+w), w, C)$:

$$\begin{aligned}
&L(d', D, 1/(1+w), w, C) \\
&\geq L(e, D, 1/(1+w), w, C) + \frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1 - w\right)\tilde{n}_{e,D}^{+} + w\tilde{n}_{e,D}^{+}\right) + C \\
&= L(e, D, 1/(1+w), w, C) + \frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C.
\end{aligned}$$

This means

$$\begin{aligned}
&\inf_{d' \in \mathcal{F}'(\mathcal{X}, D, e, 1)} L(d', D, 1/(1+w), w, C) \\
&\geq L(e, D, 1/(1+w), w, C) + \frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C.
\end{aligned} \tag{2.38}$$

Step 3. Put everything together.

Using (2.32), (2.34), (2.35), and (2.38), we have

$$\begin{aligned}
&L(d, D, 1/(1+w), w, C) \\
&\geq \min\left(\inf_{d' \in \mathcal{F}'(\mathcal{X}, D, e, 1)} L(d', D, 1/(1+w), w, C), L(\bar{e}, D, 1/(1+w), w, C)\right) \\
&\geq \min\left(L(e, D, 1/(1+w), w, C) + \frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C,\right. \\
&\qquad \left.\min\left(L(e, D, 1/(1+w), w, C) + \frac{1}{n}\tilde{n}_{e,D}^{-}, L(e, D, 1/(1+w), w, C) + \frac{w}{n}\tilde{n}_{e,D}^{+}\right)\right) \\
&= L(e, D, 1/(1+w), w, C) + \min\left(\frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C, \frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right),
\end{aligned}$$

as desired.

Case 2. $\alpha_{|e|-1}^{(e,D)} \leq 1/(1+w)$.

This implies $\alpha_j^{(d,D)} \leq 1/(1+w)$ for all $j \in \{|e|, ..., |d|\}$. By Lemma 2.4.4, we have

$$L(d, D, 1/(1+w), w, C) \geq L(\bar{e}, D, 1/(1+w), w, C).$$

Since $L(\bar{e}, D, 1/(1+w), w, C)$ is given by Equation (2.35), we have

$$L(d, D, 1/(1+w), w, C) \geq L(e, D, 1/(1+w), w, C) + \min\left(\frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right). \tag{2.39}$$

Given $\alpha_{|e|-1}^{(e,D)} \leq 1/(1+w)$, we must also have

$$\frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(d',D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C \geq \frac{w}{n}\tilde{n}_{e,D}^{+} + C \geq \frac{w}{n}\tilde{n}_{e,D}^{+},$$

which means

$$\min\left(\frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right) = \min\left(\frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(d',D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C, \frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right). \tag{2.40}$$

Substituting (2.40) into (2.39) completes the proof for Case 2.

Finally, if Inequality (2.11) holds, then we have

$$\frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|-1}^{(d',D)}} - 1\right)\tilde{n}_{e,D}^{+}\right) + C \geq \min\left(\frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right),$$

which implies

$$L^*(e, D, w, C) = L(e, D, 1/(1+w), w, C) + \min\left(\frac{w}{n}\tilde{n}_{e,D}^{+}, \frac{1}{n}\tilde{n}_{e,D}^{-}\right)$$

$$= L(\bar{e}, D, 1/(1+w), w, C).$$

□

2.5 Softly Falling Rule Lists

Program 2.2.9 and Algorithm FRL have some limitations. Let us consider a toy example, where we have a training set D of 19 instances, with 14 positive and 5 negative instances. Suppose that we have an antecedent A_1 that is satisfied by 8 positive and 3 negative training instances. If A_1 were to be the first rule of a falling rule list d that is compatible with D, we would obtain a prefix $e = \{(A_1, 8/11)\}$. However, the empirical positive proportion after the prefix e is $\tilde{\alpha}_{e,D} = 6/8 > 8/11$. This violates (2) in Proposition 2.4.2, so e is not a feasible prefix for Program 2.2.9 under the training data D. In fact, if every antecedent in A is satisfied by 8 positive and 3 negative instances in the training set D, then the only possible compatible falling rule list we can learn using Algorithm FRL is the trivial falling rule list, which has only the final else clause. At the same time, if we consider the rule list $d = \{(A_1, 8/11), 6/8\}$, which is compatible with the given toy dataset D but is not a falling rule list, we may notice that the two probability estimates in d are quite close to each other – it is very likely that the difference between them is due to sampling variability in the dataset itself.

The two limitations of Program 2.2.9 and Algorithm FRL – the potential non-existence of a feasible non-trivial solution and the rigidness of using empirical positive proportions as probability estimates – motivate us to formulate a new optimization program for learning "softly" falling rule lists, where we remove the monotonicity constraint and instead introduce a penalty term in the objective function that penalizes violations of the monotonicity

constraint (2.7) in Program 2.2.9. More formally, define a softly falling rule list as a rule list of Equation (2.1) with $\hat{\alpha}_j^{(d)} = \min_{k \leq j} \alpha_k^{(d,D)}$. Note that any rule list d that is compatible with the given training data D can be turned into a softly falling rule list by setting $\hat{\alpha}_j^{(d)} = \min_{k \leq j} \alpha_k^{(d,D)}$. Hence, we can learn a softly falling rule list by first learning a compatible rule list with the "softly falling objective" (denoted by $\tilde{L}$ below), and then transforming the rule list into a softly falling rule list. Let

$$\tilde{L}(d, D, \tau, w, C, C_1) = L(d, D, \tau, w, C) + C_1 \sum_{j=0}^{|d|} \lfloor \alpha_j^{(d,D)} - \min_{k<j} \alpha_k^{(d,D)} \rfloor_+$$

$$\text{and } \tilde{L}(e, D, \tau, w, C, C_1) = L(e, D, \tau, w, C) + C_1 \sum_{j=0}^{|e|-1} \lfloor \alpha_j^{(e,D)} - \min_{k<j} \alpha_k^{(e,D)} \rfloor_+$$

be the regularized empirical risk of misclassification by a rule list d and by a prefix e, respectively, with a penalty term that penalizes violations of monotonicity in the empirical positive proportions of the antecedents in d and in e, respectively. We call $\tilde{L}$ the softly falling objective function, set the threshold $\tau = 1/(1+w)$ as before, and obtain the following optimization program:

Program 2.5.1 (Learning compatible rule lists with the softly falling objective)**.**

$$\min_{d \in \mathcal{D}(\mathcal{X}, D)} \tilde{L}(d, D, 1/(1+w), w, C, C_1)$$

$$\text{subject to} \quad a_j^{(d)} \in A, \text{ for all } j \in \{0, 1, ..., |d| - 1\}.$$

An instance of Program 2.5.1 is defined by the tuple (D, A, w, C, C_1). Similarly, we have a Monte-Carlo search algorithm, Algorithm softFRL, based on Program 2.5.1, for learning softly falling rule lists from data. Given an instance (D, A, w, C, C_1) of Program 2.5.1, this

algorithm searches through the space of rule lists that are compatible with D and finds a compatible rule list whose antecedents come from A, and whose objective value is the smallest among all the rule lists that the algorithm explores. It then turns this compatible rule list into a softly falling rule list. In the search phase, the algorithm uses the following prefix bound (Theorem 2.5.2) to prune the search space of compatible rule lists. The details of Algorithm softFRL are presented in Section 2.5.1, and the proof of Theorem 2.5.2 is given in Section 2.5.2.

Theorem 2.5.2. *Suppose that we are given an instance (D, A, w, C, C_1) of Program 5.1 and a prefix e that is compatible with D. Then any rule list d that begins with e and is compatible with D satisfies*

$$\tilde{L}(d, D, 1/(1+w), w, C, C_1) \geq \tilde{L}^*(e, D, w, C, C_1),$$

where

$$\begin{aligned}
&\tilde{L}^*(e, D, w, C, C_1) = \tilde{L}(e, D, 1/(1+w), w, C, C_1) \\
&+ \min\left(\frac{1}{n}\left(\frac{1}{\alpha_{\min}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+} + C + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_+ + \frac{w}{n}\tilde{n}_{e,D}^{+}\mathbb{1}[\tilde{\alpha}_{e,D} \geq \alpha_{\min}^{(e,D)}],\right. \\
&\quad \inf_{\beta:\zeta<\beta\leq 1} g(\beta), \frac{w}{n}\tilde{n}_{e,D}^{+} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_+, \\
&\quad \left.\frac{1}{n}\tilde{n}_{e,D}^{-} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_+\right)
\end{aligned} \tag{2.41}$$

is a lower bound on the objective value of any compatible rule list that begins with e, under the instance (D, A, w, C, C_1) of Program 2.5.1. In Equation (2.41), $\alpha_{\min}^{(e,D)}$, ζ, and g are

defined by

$$\alpha_{\min}^{(e,D)} = \min_{k<|e|} \alpha_k^{(e,D)},$$

$$\zeta = \max(\alpha_{\min}^{(e,D)}, \tilde{\alpha}_{e,D}, 1/(1+w)),$$

$$g(\beta) = \frac{1}{n}\left(\frac{1}{\beta} - 1\right)\tilde{n}_{e,D}^{+} + C + C_1(\beta - \alpha_{\min}^{(e,D)}).$$

Note that $\inf_{\beta:\zeta<\beta\leq 1} g(\beta)$ *can be computed analytically:* $\inf_{\beta:\zeta<\beta\leq 1} g(\beta) = g(\beta^*)$ *if* $\beta^* = \sqrt{\tilde{n}_{e,D}^{+}/(C_1 n)}$ *satisfies* $\zeta < \beta^* \leq 1$*, and* $\inf_{\beta:\zeta<\beta\leq 1} g(\beta) = \min(g(\zeta), g(1))$ *otherwise.*

2.5.1 Algorithm softFRL

In this section, we present Algorithm softFRL in detail.

Given an instance (D, A, w, C, C_1) of Program 2.5.1, the algorithm searches through the space of rule lists that are compatible with D and finds a compatible rule list whose antecedents come from A, and whose objective value is the smallest among all the rule lists that the algorithm explores. It does so by iterating over T steps, in each of which the algorithm constructs a compatible rule list d, while keeping track of the rule list d^* that has the smallest objective value $\tilde{L}_{\text{best}} = \tilde{L}(d^*, D, 1/(1+w), w, C, C_1)$ among all the rule lists that the algorithm has constructed so far. At the end of T iterations, the algorithm transforms the rule list d^* that has the smallest objective value out of the T lists it has constructed, into a falling rule list by setting $\hat{\alpha}_j^{(d^*)} = \min_{k\leq j} \alpha_k^{(d^*,D)}$.

In the process of constructing a rule list d, the algorithm chooses the antecedents successively: first for the antecedent $a_0^{(d)}$ in the top rule, then for the antecedent $a_1^{(d)}$ in the

next rule, and so forth. For each antecedent $a_j^{(d)}$ chosen, the algorithm also computes its empirical positive proportion $\alpha_j^{(d,D)}$. After p rules have been constructed so that d currently holds the prefix $e = \{(a_0^{(d)}, \alpha_0^{(d,D)}), (a_1^{(d)}, \alpha_1^{(d,D)}), ..., (a_{p-1}^{(d)}, \alpha_{p-1}^{(d,D)})\}$, the algorithm either: (1) terminates the construction of d by computing the empirical positive proportion after e, $\tilde{\alpha}_{e,D}$, and then adding to d the final else clause with probability estimate $\tilde{\alpha}_{e,D}$, or (2) randomly picks an antecedent from a candidate set S of possible next antecedents, computes its empirical positive proportion, and use these as the next rule $(a_p^{(d)}, \alpha_p^{(d,D)})$ for d.

The algorithm uses Theorem 2.5.2 to prune the search space. More specifically, the algorithm terminates the construction of d if $\tilde{L}^*(e, D, w, C, C_1)$ defined by Equation (2.41) in Theorem 2.5.2 is equal to $\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1)$, where $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$ is the compatible rule list in which the prefix e is followed directly by the final else clause. The condition $\tilde{L}^*(e, D, w, C, C_1) = \tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1)$ implies that $\bar{e}$ is an optimal compatible rule list that begins with e. If we have $\tilde{L}^*(e, D, w, C, C_1) < \tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1)$ instead, the algorithm either terminates the construction of d with some probability, or it proceeds to construct a candidate set S of possible next antecedents, as follows. For every antecedent $A_l \in A$ that has not been chosen before, it constructs a candidate next rule $(a_p^{(d)}, \alpha_p^{(d,D)})$ by setting $a_p^{(d)} = A_l$ and computing $\alpha_p^{(d,D)}$ using Definition 2.2.5. The algorithm then checks if the best possible objective value $\tilde{L}^*(e', D, w, C, C_1)$ achievable by any rule list that begins with $e' = \{e, (a_p^{(d)}, \alpha_p^{(d,D)})\}$ and is compatible with D (Theorem 2.5.2) is less than the current best objective value $\tilde{L}_{\text{best}} = \tilde{L}(d^*, D, 1/(1+w), w, C, C_1)$. If so, the algorithm

adds A_l to S. Once the construction of S is complete, the algorithm randomly chooses an antecedent $A_l \in S$ with probability $P(A_l|S, e, D)$ and uses this antecedent, together with its empirical positive proportion, as the next rule $(a_p^{(d)}, \alpha_p^{(d,D)})$ for d. If S is empty, the algorithm terminates the construction of d.

In practice, we define the probability $P(A_l|S, e, D)$ for $A_l \in S$ by first defining a curiosity function $f_{S,e,D} : S \to \mathbb{R}_{\geq 0}$ and then normalizing it:

$$P(A_l|S, e, D) = \frac{f_{S,e,D}(A_l)}{\sum_{A_{l'}} f_{S,e,D}(A_{l'})}.$$

A possible choice of the curiosity function $f_{S,e,D}$ for use in Algorithm softFRL is given by

$$f_{S,e,D}(A_l) = \lambda \lfloor \min(\alpha(A_l, e, D), \frac{1.01}{0.01}\alpha_{\min}^{(e,D)} - \frac{1}{0.01}\alpha(A_l, e, D)) \rfloor_+ + (1 - \lambda)\frac{n^+(A_l, e, D)}{\tilde{n}_{e,D}^+}, \tag{2.42}$$

where $\alpha_{\min}^{(e,D)} = \min_{k<|e|} \alpha_k^{(e,D)}$ is the minimum empirical positive proportion of the antecedents in the prefix e, $\alpha(A_l, e, D)$ is the empirical positive proportion of A_l, and $n^+(A_l, e, D)$ is the number of positive training inputs captured by A_l, should A_l be chosen as the next antecedent after the prefix e. The curiosity function $f_{S,e,D}$ given by (2.42) is a weighted sum of $\lfloor \min(\alpha(A_l, e, D), (1.01/0.01)\alpha_{|e|-1}^{(e,D)} - (1/0.01)\alpha(A_l, e, D)) \rfloor_+$ and $n^+(A_l, e, D)/\tilde{n}_{e,D}^+$ for each $A_l \in S$: the former encourages the algorithm to choose antecedents that have large empirical positive proportions but do not violate the monotonicity constraint $\alpha(A_l, e, D) \leq \alpha_{\min}^{(e,D)}$ by more than 1%, and the latter encourages the algorithm to choose antecedents that have large positive supports in the training data not captured by e. We used this curiosity function for Algorithm softFRL in our experiments.

The pseudocode of Algorithm softFRL is shown in Algorithm 2.

Input: an instance (D, A, w, C, C_1) of Program 2.5.1
Result: a falling rule list d^* whose antecedents come from A
initialize $d^* = \emptyset$, $\tilde{L}_{\text{best}} = \infty$;
for $t = 1, ..., T$ **do**
  set $p = -1$, $\alpha_p = 1$, $d = e = \emptyset$;
  while $\tilde{L}^*(e, D, w, C, C_1) < \tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1)$ **do**
    go to Terminate with some probability;
    set $p = p + 1$, $S = \emptyset$;
    for *every antecedent* $A_l \in A$ *that is not in* d **do**
      set $a_p^{(d)} = A_l$, compute $\alpha_p^{(d,D)}$, and let $e' = \{e, (a_p^{(d)}, \alpha_p^{(d,D)})\}$;
      compute $\tilde{L}^*(e', D, w, C, C_1)$ using Theorem 2.5.2;
      if $\tilde{L}^*(e', D, w, C, C_1) < \tilde{L}(d^*, D, 1/(1+w), w, C, C_1)$ **then**
        add A_l to S;
      end
    end
    if $S \neq \emptyset$ **then**
      choose an antecedent $A_l \in S$ with probability $P(A_l|S, e, D)$
       according to a discrete probability distribution over S;
      set $a_p^{(d)} = A_l$ and add $(a_p^{(d)}, \alpha_p^{(d,D)})$ to d;
      set $e = d$;
      // save the partially constructed list d as the prefix e
    else
      go to Terminate
    end
  end
  Terminate: terminate the construction of d, and compute
   $\tilde{L}(d, D, 1/(1+w), w, C, C_1)$;
  if $\tilde{L}(d, D, 1/(1+w), w, C, C_1) < \tilde{L}_{best}$ **then**
    set $d^* = d$, $\tilde{L}_{\text{best}} = \tilde{L}(d, D, 1/(1+w), w, C, C_1)$;
  end
end
transform d^* into a falling rule list by setting $\hat{\alpha}_j^{(d^*)} = \min_{k \leq j} \alpha_k^{(d^*, D)}$;

Algorithm 2: Algorithm softFRL

Figure 2.2: Algorithm softFRL

2.5.2 Proof of Theorem 2.5.2

To prove Theorem 2.5.2, we need the following lemma:

Lemma 2.5.3. *Suppose that we are given an instance (D, A, w, C, C_1) of Program 2.5.1, a prefix e that is compatible with D, and a (possibly hypothetical) rule list d that begins with e and is compatible with D. Then there exists a rule list d', possibly hypothetical with respect to A, such that d' begins with e, has at most one more rule (excluding the final else clause) following e, is compatible with D, and satisfies*

$$\tilde{L}(d', D, 1/(1+w), w, C, C_1) \leq \tilde{L}(d, D, 1/(1+w), w, C, C_1). \tag{2.43}$$

Moreover, if either $\alpha_j^{(d,D)} > 1/(1+w)$ holds for all $j \in \{|e|, |e|+1, ..., |d|\}$, or $\alpha_j^{(d,D)} \leq 1/(1+w)$ holds for all $j \in \{|e|, |e|+1, ..., |d|\}$, then the rule list $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$ (i.e. the rule list in which the final else clause follows immediately the prefix e, and the probability estimate of the final else clause is $\tilde{\alpha}_{e,D}$) is compatible with D and satisfies $\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1) \leq \tilde{L}(d, D, 1/(1+w), w, C)$.

Proof. **Case 1.** There exists some $k \in \{|e|, ..., |d|\}$ that satisfies $\alpha_k^{(d,D)} > 1/(1+w)$ and some $k' \in \{|e|, ..., |d|\}$ that satisfies $\alpha_{k'}^{(d,D)} \leq 1/(1+w)$. For any $j \in \{|e|, ..., |d|\}$ with $\alpha_j^{(d,D)} > 1/(1+w)$, the contribution $R_j(d, D, 1/(1+w), w)$ by the j-th rule to $R(d, D, 1/(1+w), w)$, defined by the right-hand side of Equation (2.6) with $\tau = 1/(1+w)$, is given by

$$R_j(d, D, 1/(1+w), w) = \frac{1}{n} n^-_{j,d,D}.$$

For any $j \in \{|e|, ..., |d|\}$ with $\alpha_j^{(d,D)} \leq 1/(1+w)$, the contribution $R_j(d, D, 1/(1+w), w)$ by the j-th rule to $R(d, D, 1/(1+w), w)$ is given by

$$R_j(d, D, 1/(1+w), w) = \frac{w}{n} n^+_{j,d,D}.$$

The rest of the proof for this case proceeds in four steps.

Step 1. Construct a hypothetical rule list d' that begins with e, has exactly one more rule (excluding the final else clause) following e, and is compatible with D. In later steps, we shall show that the rule list d' constructed in this step satisfies (2.43).

Let $d' = \{e, (a_{|e|}^{(d')}, \hat{\alpha}_{|e|}^{(d')}), \hat{\alpha}_{|e|+1}^{(d')}\}$ be the hypothetical rule list of size $|d'| = |e| + 1$ that is compatible with D, and whose $|e|$-th antecedent $a_{|e|}^{(d')}$ is defined by

$$a_{|e|}^{(d')}(\mathbf{x}) = \mathbb{1}[\alpha_{\mathrm{capt}(\mathbf{x},d)}^{(d,D)} > 1/(1+w)] \cdot \mathbb{1}[|e| \leq \mathrm{capt}(\mathbf{x}, d) \leq |d|].$$

Step 2. Show that the empirical risk of misclassification by the rule list d' is the same as that by the rule list d.

To see this, we observe that the training instances in D captured by $a_{|e|}^{(d')}$ in d' are exactly those captured by the antecedents $a_j^{(d)}$, $|e| \leq j \leq |d|$, in d whose empirical positive proportion satisfies $\alpha_j^{(d,D)} > 1/(1+w)$, and the training instances in D captured by $a_{|e|+1}^{(d')}$ (i.e. the final else clause) in d' are exactly those captured by the antecedents $a_j^{(d)}$, $|e| \leq j \leq |d|$, in d whose empirical positive proportion satisfies $\alpha_j^{(d,D)} \leq 1/(1+w)$. This observation implies

$$n_{|e|,d',D}^{+} = \sum_{j:|e|\leq j\leq|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n_{j,d,D}^{+}, \tag{2.44}$$

$$n_{|e|,d',D}^{-} = \sum_{j:|e|\leq j\leq|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n_{j,d,D}^{-}, \tag{2.45}$$

$$n_{|e|,d',D} = \sum_{j:|e|\leq j\leq|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n_{j,d,D}, \tag{2.46}$$

$$n^+_{|e|+1,d',D} = \sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}\le 1/(1+w)} n^+_{j,d,D} \tag{2.47}$$

and

$$n_{|e|+1,d',D} = \sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}\le 1/(1+w)} n_{j,d,D}. \tag{2.48}$$

Since d' is compatible with D, using the definition of a compatible rule list in Definition 2.2.6 and the definition of the empirical positive proportion in Definition 2.2.5, together with (2.44), (2.46), (2.47), and (2.48), we must have

$$\begin{aligned}\hat{\alpha}^{(d')}_{|e|} = \alpha^{(d',D)}_{|e|} = \frac{n^+_{|e|,d',D}}{n_{|e|,d',D}} &= \frac{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n^+_{j,d,D}}{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n_{j,d,D}} \\ &= \frac{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} \alpha_j^{(d,D)} n_{j,d,D}}{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n_{j,d,D}} > \frac{1}{1+w},\end{aligned}$$

and

$$\begin{aligned}\hat{\alpha}^{(d')}_{|e|+1} = \alpha^{(d',D)}_{|e|+1} = \frac{n^+_{|e|+1,d',D}}{n_{|e|+1,d',D}} &= \frac{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}\le 1/(1+w)} n^+_{j,d,D}}{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}\le 1/(1+w)} n_{j,d,D}} \\ &= \frac{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}\le 1/(1+w)} \alpha_j^{(d,D)} n_{j,d,D}}{\sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}\le 1/(1+w)} n_{j,d,D}} \le \frac{1}{1+w}.\end{aligned}$$

This means that the contribution $R_{|e|}(d', D, 1/(1+w), w)$ by the $|e|$-th rule to $R(d', D, 1/(1+w), w)$ is given by

$$R_{|e|}(d', D, 1/(1+w), w) = \frac{1}{n} n^-_{|e|,d',D} = \frac{1}{n} \sum_{j:|e|\le j\le|d|\wedge\alpha_j^{(d,D)}>1/(1+w)} n^-_{j,d,D},$$

where we have used (2.45), and the contribution $R_{|e|+1}(d', D, 1/(1+w), w)$ by the $(|e|+1)$-st

"rule" (i.e. the final else clause) to $R(d', D, 1/(1+w), w)$ is given by

$$R_{|e|+1}(d', D, 1/(1+w), w) = \frac{w}{n} n^+_{|e|+1,d',D} = \frac{w}{n} \sum_{j:|e|\le j\le |d| \wedge \alpha_j^{(d,D)} \le 1/(1+w)} n^+_{j,d,D},$$

where we have used (2.47).

It then follows that the empirical risk of misclassification by the rule list d' is the same as that by the rule list d:

$$\begin{aligned}
&R(d', D, 1/(1+w), w) \\
&= R(e, D, 1/(1+w), w) + R_{|e|}(d', D, 1/(1+w), w) + R_{|e|+1}(d', D, 1/(1+w), w) \\
&= R(e, D, 1/(1+w), w) \\
&\quad + \frac{1}{n} \sum_{j:|e|\le j\le |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} n^-_{j,d,D} + \frac{w}{n} \sum_{j:|e|\le j\le |d| \wedge \alpha_j^{(d,D)} \le 1/(1+w)} n^+_{j,d,D} \\
&= R(e, D, 1/(1+w), w) + \sum_{j=|e|}^{|d|} R_j(d, D, 1/(1+w), w) \\
&= R(d, D, 1/(1+w), w). \qquad (2.49)
\end{aligned}$$

Step 3. Show that the monotonicity penalty of the rule list d' is at most that of d.

Let $S(d, D) = \sum_{j=0}^{|d|} \lfloor \alpha_j^{(d,D)} - \min_{k<j} \alpha_k^{(d,D)} \rfloor_+$ be the monotonicity penalty of the rule list d. We now show $S(d', D) \le S(d, D)$. Let $S_j(d, D) = \lfloor \alpha_j^{(d,D)} - \min_{k<j} \alpha_k^{(d,D)} \rfloor_+$ be the monotonicity penalty for the j-th rule in d.

Let $l \in \{|e|, ..., |d|\}$ be any integer with

$$\alpha_l^{(d,D)} = \max_{j:|e|\le j\le |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} \alpha_j^{(d,D)}. \qquad (2.50)$$

Then the total monotonicity penalty for all the rules $(a_j^{(d)}, \alpha_j^{(d,D)})$ in d with $|e| \leq j \leq |d|$ and $\alpha_j^{(d,D)} > 1/(1+w)$ satisfies

$$\begin{aligned}
&\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} S_j(d,D) \\
&\geq S_l(d,D) \quad \text{(because } S_l(d,D) \text{ is included in the sum on the left)} \\
&= \lfloor \alpha_l^{(d,D)} - \min_{k<l} \alpha_k^{(d,D)} \rfloor_+ \\
&\geq \lfloor \alpha_l^{(d,D)} - \min_{k<|e|} \alpha_k^{(d,D)} \rfloor_+. \qquad (2.51)
\end{aligned}$$

On the other hand, the monotonicity penalty for the $|e|$-th rule in d' satisfies

$$S_{|e|}(d',D) = \lfloor \alpha_{|e|}^{(d',D)} - \min_{k<|e|} \alpha_k^{(d',D)} \rfloor_+ \leq \lfloor \alpha_l^{(d,D)} - \min_{k<|e|} \alpha_k^{(d,D)} \rfloor_+, \qquad (2.52)$$

because we have $\min_{k<|e|} \alpha_k^{(d',D)} = \min_{k<|e|} \alpha_k^{(d,D)}$ (d and d' begin with the same prefix e), and

$$\begin{aligned}
&\alpha_{|e|}^{(d',D)} \\
&= \frac{n^+_{|e|,d',D}}{n_{|e|,d',D}} \quad \text{(by definition of empirical positive proportion in Definition 2.2.5)} \\
&= \frac{\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} n^+_{j,d,D}}{\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} n_{j,d,D}} \quad \text{(by Equations (2.44) and (2.46))} \\
&= \frac{\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} \alpha_j^{(d,D)} n_{j,d,D}}{\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} n_{j,d,D}} \quad \text{(by definition of } \alpha_j^{(d,D)} \text{ in Definition 2.2.5)} \\
&\leq \frac{\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} \alpha_l^{(d,D)} n_{j,d,D}}{\sum_{j:|e|\leq j\leq |d| \wedge \alpha_j^{(d,D)} > 1/(1+w)} n_{j,d,D}} \quad \text{(by the definition of } l \text{ in (2.50))} \\
&= \alpha_l^{(d,D)}.
\end{aligned}$$

Combining (2.51) and (2.52), we have

$$S_{|e|}(d', D) \leq \sum_{j:|e|\leq j\leq |d|\wedge\alpha_j^{(d,D)}>1/(1+w)} S_j(d, D). \tag{2.53}$$

A similar argument will show

$$S_{|e|+1}(d', D) \leq \sum_{j:|e|\leq j\leq |d|\wedge\alpha_j^{(d,D)}\leq 1/(1+w)} S_j(d, D). \tag{2.54}$$

It then follows from (2.53) and (2.54) that the monotonicity penalty of d' is at most that of d:

$$\begin{aligned} S(d', D) &= \left(\sum_{j=0}^{|e|-1} S_j(d', D)\right) + S_{|e|}(d', D) + S_{|e|+1}(d', D) \\ &\leq \left(\sum_{j=0}^{|e|-1} S_j(d, D)\right) + \sum_{j:|e|\leq j\leq |d|\wedge\alpha_j^{(d,D)}>1/(1+w)} S_j(d, D) \qquad (2.55) \\ &\quad + \sum_{j:|e|\leq j\leq |d|\wedge\alpha_j^{(d,D)}\leq 1/(1+w)} S_j(d, D) \\ &= S(d, D). \qquad (2.56) \end{aligned}$$

Step 4. Put everything together.

Using (2.49) and (2.56), together with the observation $|d'| = |e| + 1 \leq |d|$, we must also have

$$\begin{aligned} \tilde{L}(d', D, 1/(1+w), w, C, C_1) &= R(d', D, 1/(1+w), w) + C|d'| + C_1 S(d', D) \\ &\leq R(d, D, 1/(1+w), w) + C|d| + C_1 S(d, D) \\ &= \tilde{L}(d, D, 1/(1+w), w, C, C_1). \end{aligned}$$

Case 2. Either $\alpha_j^{(d,D)} > 1/(1+w)$ holds for all $j \in \{|e|, ..., |d|\}$, or $\alpha_j^{(d,D)} \leq 1/(1+w)$ holds for all $j \in \{|e|, ..., |d|\}$. The construction of $d' = \bar{e}$ and the proof for $R(d', D, 1/(1+w), w) = R(d, D, 1/(1+w), w)$ is similar to those given in the proof of Lemma 2.4.4. The proof for $S(d', D) \leq S(d, D)$ is similar to that in Case 1. The desired inequality then follows from $|d'| = |e| \leq |d|$. □

Before we proceed with proving Theorem 2.5.2, we make the following four observations. Observations 2.5.4, 2.5.5, and 2.5.6 are the same as Observations 2.4.7, 2.4.8 and 2.4.9. They are repeated here for convenience.

Observation 2.5.4. *For any rule list*

$$d' = \{e, (a_{|e|}^{(d')}, \hat{\alpha}_{|e|}^{(d')}), ..., (a_{|d'|-1}^{(d')}, \hat{\alpha}_{|d'|-1}^{(d')}), \hat{\alpha}_{|d'|}^{(d')}\}$$

that begins with a given prefix e, we have

$$\tilde{n}_{e,D}^{+} = n_{|e|,d',D}^{+} + ...n_{|d'|,d',D}^{+}, \tag{2.57}$$

$$\tilde{n}_{e,D}^{-} = n_{|e|,d',D}^{-} + ...n_{|d'|,d',D}^{-}, \tag{2.58}$$

and

$$\tilde{n}_{e,D} = n_{|e|,d',D} + ...n_{|d'|,d',D}. \tag{2.59}$$

Proof. Same as Observation 2.4.7. □

Observation 2.5.5. *For any rule list d', we have*

$$n_{|e|,d',D}^{-} = \left(\frac{1}{\alpha_{|e|}^{(d',D)}} - 1 \right) n_{|e|,d',D}^{+}, \tag{2.60}$$

Proof. Same as Observation 2.4.8. □

Observation 2.5.6. *For any rule list*

$$d' = \{e, (a_{|e|}^{(d')}, \hat{\alpha}_{|e|}^{(d')}), \hat{\alpha}_{|e|+1}^{(d')}\}$$

that has exactly one rule (excluding the final else clause) following a given prefix e, we have

$$n_{|e|+1,d',D}^{+} = \tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}, \tag{2.61}$$

$$n_{|e|+1,d',D}^{-} = \tilde{n}_{e,D}^{-} - n_{|e|,d',D}^{-}, \tag{2.62}$$

and

$$n_{|e|+1,d',D} = \tilde{n}_{e,D} - n_{|e|,d',D}. \tag{2.63}$$

Note that since $n_{|e|+1,d',D}^{+}$, $n_{|e|+1,d',D}^{-}$, and $n_{|e|+1,d',D}$ are non-negative, Equations (2.61), (2.62), and (2.63) imply $n_{|e|,d',D}^{+} \leq \tilde{n}_{e,D}^{+}$, $n_{|e|,d',D}^{-} \leq \tilde{n}_{e,D}^{-}$, and $n_{|e|,d',D} \leq \tilde{n}_{e,D}$.

Proof. Same as Observation 2.4.9. □

Observation 2.5.7. *For any rule list*

$$d' = \{e, (a_{|e|}^{(d')}, \hat{\alpha}_{|e|}^{(d')}), \hat{\alpha}_{|e|+1}^{(d')}\}$$

that has exactly one rule (excluding the final else clause) following a given prefix e, we have

$$\alpha_{|e|+1}^{(d',D)} = \frac{\tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}}{\tilde{n}_{e,D}^{+} + \tilde{n}_{e,D}^{-} - \frac{1}{\alpha_{|e|}^{(d',D)}} n_{|e|,d',D}^{+}}. \tag{2.64}$$

Proof. By Definition 2.2.5, we have

$$\alpha_{|e|+1}^{(d',D)} = \frac{n_{|e|+1,d',D}^{+}}{n_{|e|+1,d',D}} = \frac{n_{|e|+1,d',D}^{+}}{n_{|e|+1,d',D}^{+} + n_{|e|+1,d',D}^{-}}.$$

Applying Equations (2.61) and (2.62) in Observation 2.5.6, we have

$$\alpha_{|e|+1}^{(d',D)} = \frac{\tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}}{(\tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}) + (\tilde{n}_{e,D}^{-} - n_{|e|,d',D}^{-})}$$

$$= \frac{\tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}}{\tilde{n}_{e,D}^{+} + \tilde{n}_{e,D}^{-} - n_{|e|,d',D}^{+} - n_{|e|,d',D}^{-}}.$$

Applying Equation (2.60) in Observation 2.5.5, we have

$$\alpha_{|e|+1}^{(d',D)} = \frac{\tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}}{\tilde{n}_{e,D}^{+} + \tilde{n}_{e,D}^{-} - n_{|e|,d',D}^{+} - \left(\frac{1}{\alpha_{|e|}^{(d',D)}} - 1\right) n_{|e|,d',D}^{+}}$$

$$= \frac{\tilde{n}_{e,D}^{+} - n_{|e|,d',D}^{+}}{\tilde{n}_{e,D}^{+} + \tilde{n}_{e,D}^{-} - \frac{1}{\alpha_{|e|}^{(d',D)}} n_{|e|,d',D}^{+}}.$$

□

We are now ready to prove Theorem 2.5.2.

Proof of Theorem 2.5.2. Let $\mathcal{D}(\mathcal{X}, D, e)$ be the set of rule lists (including both hypothetical and non-hypothetical rule lists) that begin with e and are compatible with D, and let $\mathcal{D}(\mathcal{X}, D, e, k)$ be the subset of $\mathcal{D}(\mathcal{X}, D, e)$, consisting of those rule lists in $\mathcal{D}(\mathcal{X}, D, e)$ that have exactly k rules (excluding the final else clause) following the prefix e. Let $\mathcal{S}(\mathcal{X}, D, e, 1)$ be the subset of $\mathcal{D}(\mathcal{X}, D, e, 1)$, consisting of those rule lists

$$d' = \{e, (a_{|e|}^{(d')}, \alpha_{|e|}^{(d',D)}), \alpha_{|e|+1}^{(d',D)}\} \in \mathcal{D}(\mathcal{X}, D, e, 1)$$

with $\alpha_{|e|}^{(d',D)} > 1/(1+w)$ and $\alpha_{|e|+1}^{(d',D)} \leq 1/(1+w)$.

Note that we have $\mathcal{D}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$, where $\bar{e} = \{e, \tilde{\alpha}_{e,D}\}$ is the rule list in which the final else clause immediately follows the prefix e, and the probability estimate of the final

else clause is $\tilde{\alpha}_{e,D}$, by a similar argument as that given in the proof of Theorem 2.4.6 for $\mathcal{F}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$.

Let $d \in \mathcal{D}(\mathcal{X}, D, e)$.

Lemma 2.5.3, along with its proof, implies

$$\tilde{L}(d, D, 1/(1+w), w, C, C_1) \geq \inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1) \bigcup \mathcal{D}(\mathcal{X},D,e,0)} \tilde{L}(d', D, 1/(1+w), w, C, C_1). \quad (2.65)$$

This is because if d obeys Case 1 in the proof of the lemma, then using the same argument as in the proof of the lemma we can construct a rule list $d_1 = \{e, (a_{|e|}^{(d_1)}, \alpha_{|e|}^{(d_1,D)}), \alpha_{|e|+1}^{(d_1,D)}\} \in \mathcal{S}(\mathcal{X}, D, e, 1)$ that satisfies

$$\tilde{L}(d, D, 1/(1+w), w, C, C_1) \geq \tilde{L}(d_1, D, 1/(1+w), w, C, C_1). \quad (2.66)$$

Since d_1 must also obey

$$\begin{aligned} \tilde{L}(d_1, D, 1/(1+w), w, C, C_1) &\geq \inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\ &\geq \inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1) \bigcup \mathcal{D}(\mathcal{X},D,e,0)} \tilde{L}(d', D, 1/(1+w), w, C, C_1), \end{aligned} \quad (2.67)$$

combining the inequalities in (2.66) and (2.67) gives us (2.65). On the other hand, if d obeys Case 2 in the proof of the lemma, then by the lemma itself we know

$$\tilde{L}(d, D, 1/(1+w), w, C, C_1) \geq \tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1). \quad (2.68)$$

Since we have $\mathcal{D}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$, it is straightforward to see

$$\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1) = \inf_{d' \in \mathcal{D}(\mathcal{X},D,e,0)} \tilde{L}(d', D, 1/(1+w), w, C, C_1)$$

$$\geq \inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1) \bigcup \mathcal{D}(\mathcal{X},D,e,0)} \tilde{L}(d', D, 1/(1+w), w, C, C_1). \quad (2.69)$$

Combining the inequalities in (2.68) and (2.69) again gives us (2.65).

Note that if $\mathcal{S}(\mathcal{X}, D, e, 1)$ is not empty, then the right-hand side of (2.65) can be expressed as

$$\begin{aligned}
&\inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1) \bigcup \mathcal{D}(\mathcal{X},D,e,0)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&= \inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1) \bigcup \{\bar{e}\}} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&= \min \left(\inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1), \tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1) \right). \quad (2.70)
\end{aligned}$$

The rest of the proof proceeds in six steps.

Step 1. Compute $\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1)$.

Since the contribution by the final else clause to $\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1)$ is given by $R_{|e|}(\bar{e}, D, 1/(1+w), w) + \lfloor \tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)} \rfloor_+$, where $R_{|e|}(\bar{e}, D, 1/(1+w), w)$ is defined by Equation (2.6) and is given by

$$R_{|e|}(\bar{e}, D, 1/(1+w), w) = \begin{cases} \frac{1}{n} n^-_{|e|,\bar{e},D} & \text{if } \tilde{\alpha}_{e,D} > 1/(1+w) \\ \frac{w}{n} n^+_{|e|,\bar{e},D} & \text{otherwise,} \end{cases}$$

and since Observation 2.5.4 implies $\tilde{n}^+_{e,D} = n^+_{|e|,\bar{e},D}$ and $\tilde{n}^-_{e,D} = n^-_{|e|,\bar{e},D}$, it is not difficult to see

$$\begin{aligned}
&\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1) \\
&= \begin{cases} \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\tilde{n}^-_{e,D} + C_1 \lfloor \tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)} \rfloor_+ & \text{if } \tilde{\alpha}_{e,D} > 1/(1+w) \\ \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{w}{n}\tilde{n}^+_{e,D} + C_1 \lfloor \tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)} \rfloor_+ & \text{otherwise.} \end{cases}
\end{aligned}$$

Since $\tilde{\alpha}_{e,D} > 1/(1+w)$ is equivalent to $\tilde{n}^+_{e,D}/(\tilde{n}^+_{e,D}+\tilde{n}^-_{e,D}) > 1/(1+w)$, or $w\tilde{n}^+_{e,D} > \tilde{n}^-_{e,D}$, and similarly $\tilde{\alpha}_{e,D} \leq 1/(1+w)$ is equivalent to $w\tilde{n}^+_{e,D} \leq \tilde{n}^-_{e,D}$, we can write

$$\begin{aligned}
&\tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1) \\
&= \min\Bigg(\tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\tilde{n}^-_{e,D} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha^{(e,D)}_{\min}\rfloor_+, \\
&\qquad \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{w}{n}\tilde{n}^+_{e,D} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha^{(e,D)}_{\min}\rfloor_+\Bigg) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \min\left(\frac{w}{n}\tilde{n}^+_{e,D}, \frac{1}{n}\tilde{n}^-_{e,D}\right) + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha^{(e,D)}_{\min}\rfloor_+.
\end{aligned} \tag{2.71}$$

Step 2. Partition the set $\mathcal{S}(\mathcal{X}, D, e, 1)$ into three subsets based on how the softly falling objective is computed.

For any $d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{S}(\mathcal{X}, D, e, 1)$, the softly falling objective is given by

$$\begin{aligned}
&\tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}n^-_{|e|,d',D} + \frac{w}{n}n^+_{|e|+1,d',D} + C \\
&\quad + C_1\lfloor\alpha^{(d',D)}_{|e|} - \alpha^{(e,D)}_{\min}\rfloor_+ + C_1\lfloor\alpha^{(d',D)}_{|e|+1} - \alpha^{(e,D)}_{\min}\rfloor_+.
\end{aligned} \tag{2.72}$$

This is because for any $d' \in \mathcal{S}(\mathcal{X}, D, e, 1)$, the contribution by both the $|e|$-th rule and the final else clause to $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ is given by

$$\begin{aligned}
&R_{|e|}(d', D, 1/(1+w), w) + R_{|e|+1}(d', D, 1/(1+w), w) + C \\
&\quad + C_1\lfloor\alpha^{(d',D)}_{|e|} - \alpha^{(e,D)}_{\min}\rfloor_+ + C_1\lfloor\alpha^{(d',D)}_{|e|+1} - \alpha^{(e,D)}_{\min}\rfloor_+,
\end{aligned}$$

where $R_{|e|}(d', D, 1/(1+w), w)$ and $R_{|e|+1}(d', D, 1/(1+w), w)$ are defined by Equation (2.6)

and are given by

$$R_{|e|}(d', D, 1/(1+w), w) = \frac{1}{n} n^{-}_{|e|,d',D} \text{ and } R_{|e|+1}(d', D, 1/(1+w), w) = \frac{w}{n} n^{+}_{|e|+1,d',D}$$

(because we have $\alpha^{(d',D)}_{|e|} > 1/(1+w)$ and $\alpha^{(d',D)}_{|e|+1} \leq 1/(1+w)$ for $d' \in \mathcal{S}(\mathcal{X}, D, e, 1)$).

Let

$$\mathcal{S}_1(\mathcal{X}, D, e, 1)$$

$$= \{d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{S}(\mathcal{X}, D, e, 1) : \alpha^{(e,D)}_{\min} \geq \alpha^{(d',D)}_{|e|} > \alpha^{(d',D)}_{|e|+1}\},$$

$$\mathcal{S}_2(\mathcal{X}, D, e, 1)$$

$$= \{d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{S}(\mathcal{X}, D, e, 1) : \alpha^{(d',D)}_{|e|} > \alpha^{(e,D)}_{\min} \geq \alpha^{(d',D)}_{|e|+1}\},$$

and

$$\mathcal{S}_3(\mathcal{X}, D, e, 1)$$

$$= \{d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{S}(\mathcal{X}, D, e, 1) : \alpha^{(d',D)}_{|e|} > \alpha^{(d',D)}_{|e|+1} > \alpha^{(e,D)}_{\min}\},$$

It is easy to see

$$\mathcal{S}(\mathcal{X}, D, e, 1) = \mathcal{S}_3(\mathcal{X}, D, e, 1) \cup \mathcal{S}_1(\mathcal{X}, D, e, 1) \cup \mathcal{S}_2(\mathcal{X}, D, e, 1).$$

We observe here that given the prefix e, we can write $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ as a function of $n^{+}_{|e|,d',D}$ and $\alpha^{(d',D)}_{|e|}$, by substituting (2.60), (2.61), and (2.64) in Observations 2.5.5, 2.5.6, and 2.5.7 into (2.72).

Step 3. Determine a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for all $d' \in \mathcal{S}_1(\mathcal{X}, D, e, 1)$.

Let $d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{S}_1(\mathcal{X}, D, e, 1)$.

By the definition of $\mathcal{S}_1(\mathcal{X}, D, e, 1)$, we have

$$\alpha^{(e,D)}_{\min} \geq \alpha^{(d',D)}_{|e|} > \frac{1}{1+w} \geq \alpha^{(d',D)}_{|e|+1} \tag{2.73}$$

We first prove the following inequality

$$\alpha_{\min}^{(e,D)} \geq \alpha_{|e|}^{(d',D)} > \max(1/(1+w), \tilde{\alpha}_{e,D}), \tag{2.74}$$

which will be useful later.

To prove (2.74), we use Definition 2.5 as well as (2.61) and (2.63) in Observation 2.5.6 to obtain

$$\tilde{\alpha}_{e,D} = \frac{\tilde{n}_{e,D}^{+}}{\tilde{n}_{e,D}} = \frac{n_{|e|,d',D}^{+} + n_{|e|+1,d',D}^{+}}{n_{|e|,d',D} + n_{|e|+1,d',D}} = \frac{\alpha_{|e|}^{(d',D)} n_{|e|,d',D} + \alpha_{|e|+1}^{(d',D)} n_{|e|+1,d',D}}{n_{|e|,d',D} + n_{|e|+1,d',D}}. \tag{2.75}$$

Substituting $\alpha_{|e|+1}^{(d',D)} < \alpha_{|e|}^{(d',D)}$ from (2.73) into (2.75), we obtain $\tilde{\alpha}_{e,D} < \alpha_{|e|}^{(d',D)}$. Combining this inequality with $\alpha_{\min}^{(e,D)} \geq \alpha_{|e|}^{(d',D)} > \frac{1}{1+w}$ from (2.73), we obtain (2.74), as desired.

Note that since (2.74) has to hold for any $d' \in \mathcal{S}_1(\mathcal{X}, D, e, 1)$, if $\alpha_{\min}^{(e,D)} \leq \max(1/(1+w), \tilde{\alpha}_{e,D})$ is true for the given prefix e, then $\mathcal{S}_1(\mathcal{X}, D, e, 1)$ is empty.

We now show that given the prefix e, the softly falling objective $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for d' is a monotonically decreasing function of both $n_{|e|,d',D}^{+}$ and $\alpha_{|e|}^{(d',D)}$.

To do so, we substitute (2.60) and (2.61) in Observations 2.5.5 and 2.5.6 into (2.72) to obtain

$$\begin{aligned}
&\tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|}^{(d',D)}} - 1 - w\right) n_{|e|,d',D}^{+} + w\tilde{n}_{e,D}^{+}\right) + C.
\end{aligned} \tag{2.76}$$

Note that Equation (2.76) shows that given the prefix e, $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ is a

function of $n^+_{|e|,d',D}$ and $\alpha^{(d',D)}_{|e|}$. Since we have

$$\frac{\partial \tilde{L}(d', D, 1/(1+w), w, C, C_1)}{\partial n^+_{|e|,d',D}} = \frac{1}{n}\left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 - w\right) < 0$$

because $\alpha^{(d',D)}_{|e|} > 1/(1+w)$ holds for any $d' \in \mathcal{S}_1(\mathcal{X}, D, e, 1)$, and

$$\frac{\partial \tilde{L}(d', D, 1/(1+w), w, C, C_1)}{\partial \alpha^{(d',D)}_{|e|}} = -\frac{n^+_{|e|,d',D}}{n}\frac{1}{(\alpha^{(d',D)}_{|e|})^2} \leq 0,$$

we see that $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ is indeed a monotonically decreasing function of both $n^+_{|e|,d',D}$ and $\alpha^{(d',D)}_{|e|}$. Thus, we can obtain a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ by substituting $n^+_{|e|,d',D}$ and $\alpha^{(d',D)}_{|e|}$ with their respective upper bound. The inequality $n^+_{|e|,d',D} \leq \tilde{n}^+_{e,D}$ in Observation 2.5.6 gives an upper bound for $n^+_{|e|,d',D}$, and the inequality $\alpha^{(d',D)}_{|e|} \leq \alpha^{(e,D)}_{\min}$ from (2.73) gives an upper bound for $\alpha^{(d',D)}_{|e|}$. Substituting these upper bounds into (2.76), we obtain the following inequality, which gives a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$:

$$\begin{aligned}
&\tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\left(\left(\frac{1}{\alpha^{(e,D)}_{\min}} - 1 - w\right)\tilde{n}^+_{e,D} + w\tilde{n}^+_{e,D}\right) + C \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\left(\frac{1}{\alpha^{(e,D)}_{\min}} - 1\right)\tilde{n}^+_{e,D} + C.
\end{aligned}$$

Step 4. Determine a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for all $d' \in \mathcal{S}_2(\mathcal{X}, D, e, 1)$.

Let $d' = \{e, (a^{(d')}_{|e|}, \alpha^{(d',D)}_{|e|}), \alpha^{(d',D)}_{|e|+1}\} \in \mathcal{S}_2(\mathcal{X}, D, e, 1)$.

By the definition of $\mathcal{S}_2(\mathcal{X}, D, e, 1)$, we have

$$\alpha^{(d',D)}_{|e|} > \frac{1}{1+w} \tag{2.77}$$

and

$$\alpha_{|e|}^{(d',D)} > \alpha_{\min}^{(e,D)} \geq \alpha_{|e|+1}^{(d',D)} \tag{2.78}$$

We first prove the following inequality

$$1 \geq \alpha_{|e|}^{(d',D)} > \max(\alpha_{\min}^{(e,D)}, \tilde{\alpha}_{e,D}, 1/(1+w)) = \zeta, \tag{2.79}$$

which will be useful later.

To prove (2.79), we use Definition 2.5 as well as (2.61) and (2.63) in Observation 2.5.6 to obtain (2.75). Substituting $\alpha_{|e|+1}^{(d',D)} < \alpha_{|e|}^{(d',D)}$ from (2.73) into (2.75), we obtain $\tilde{\alpha}_{e,D} < \alpha_{|e|}^{(d',D)}$. Combining this inequality with (2.77) and $\alpha_{|e|}^{(d',D)} > \alpha_{\min}^{(e,D)}$ from (2.78), we obtain (2.79), as desired.

We now show that given the prefix e and a particular value of $\alpha_{|e|}^{(d',D)}$ that obeys (2.79), the softly falling objective $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for d' is a decreasing function of $n_{|e|,d',D}^{+}$.

To do so, we substitute (2.60) and (2.61) in Observations 2.5.5 and 2.5.6 into (2.72) to obtain

$$\begin{aligned}
&\tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\left(\left(\frac{1}{\alpha_{|e|}^{(d',D)}} - 1 - w\right) n_{|e|,d',D}^{+} + w\tilde{n}_{e,D}^{+}\right) + C \\
&\quad + C_1(\alpha_{|e|}^{(d',D)} - \alpha_{\min}^{(e,D)}).
\end{aligned} \tag{2.80}$$

Note that Equation (2.80) shows that given the prefix e, $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ is a function of $n_{|e|,d',D}^{+}$ and $\alpha_{|e|}^{(d',D)}$. Differentiating $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ given in (2.80)

with respect to $n^+_{|e|,d',D}$, we obtain

$$\frac{\partial \tilde{L}(d', D, 1/(1+w), w, C, C_1)}{\partial n^+_{|e|,d',D}} = \frac{1}{n}\left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 - w\right). \tag{2.81}$$

Since $\alpha^{(d',D)}_{|e|}$ obeys (2.79), in particular, it obeys $\alpha^{(d',D)}_{|e|} > 1/(1+w)$, we have

$$\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 - w < 0,$$

which then gives $\partial \tilde{L}(d', D, 1/(1+w), w, C, C_1)/\partial n^+_{|e|,d',D} < 0$. This means that given the prefix e and a particular value of $\alpha^{(d',D)}_{|e|}$ that obeys (2.79), $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ is a decreasing function of $n^+_{|e|,d',D}$.

Thus, given the prefix e and a particular value of $\alpha^{(d',D)}_{|e|}$ that obeys (2.79), we can obtain a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ by substituting $n^+_{|e|,d',D}$ with its upper bound. The inequality $n^+_{|e|,d',D} \le \tilde{n}^+_{e,D}$ in Observation 2.5.6 gives an upper bound for $n^+_{|e|,d',D}$. Substituting $n^+_{|e|,d',D}$ with its upper bound $\tilde{n}^+_{e,D}$ into (2.80), we obtain a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$, denoted by $\tilde{g}(\alpha^{(d',D)}_{|e|})$, when $\alpha^{(d',D)}_{|e|}$ is held constant:

$$\begin{aligned}
&\tilde{g}(\alpha^{(d',D)}_{|e|}) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n}\left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1\right)\tilde{n}^+_{e,D} + C + C_1(\alpha^{(d',D)}_{|e|} - \alpha^{(e,D)}_{\min}) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + g(\alpha^{(d',D)}_{|e|})
\end{aligned}$$

where g is defined in the statement of the theorem. In other words, given the prefix e and a particular value of $\alpha^{(d',D)}_{|e|}$ that obeys (2.79), we have $\tilde{L}(d', D, 1/(1+w), w, C, C_1) \ge \tilde{g}(\alpha^{(d',D)}_{|e|})$.

Since (2.79) is true for any $d' \in \mathcal{S}_2(\mathcal{X}, D, e, 1)$, we always have $\tilde{L}(d', D, 1/(1+w), w, C, C_1) \geq \tilde{g}(\alpha_{|e|}^{(d',D)})$ for any $d' \in \mathcal{S}_2(\mathcal{X}, D, e, 1)$. This implies

$$\begin{aligned}
&\tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
\geq &\inf_{\alpha_{|e|}^{(d',D)}:\zeta<\alpha_{|e|}^{(d',D)}\leq 1} \tilde{g}(\alpha_{|e|}^{(d',D)}) \\
= &\tilde{L}(e, D, 1/(1+w), w, C, C_1) + \inf_{\alpha_{|e|}^{(d',D)}:\zeta<\alpha_{|e|}^{(d',D)}\leq 1} g(\alpha_{|e|}^{(d',D)}).
\end{aligned}$$

Step 5. Determine a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for all $d' \in \mathcal{S}_3(\mathcal{X}, D, e, 1)$.

Let $d' = \{e, (a_{|e|}^{(d')}, \alpha_{|e|}^{(d',D)}), \alpha_{|e|+1}^{(d',D)}\} \in \mathcal{S}_3(\mathcal{X}, D, e, 1)$.

By the definition of $\mathcal{S}_3(\mathcal{X}, D, e, 1)$, we have

$$\alpha_{|e|}^{(d',D)} > \frac{1}{1+w} \geq \alpha_{|e|+1}^{(d',D)} > \alpha_{\min}^{(e,D)}. \tag{2.82}$$

We first prove the following inequality

$$1 \geq \alpha_{|e|}^{(d',D)} > \max(\alpha_{\min}^{(e,D)}, \tilde{\alpha}_{e,D}, 1/(1+w)) = \zeta, \tag{2.83}$$

which will be useful later.

To prove (2.83), we use Definition 2.5 as well as (2.61) and (2.63) in Observation 2.5.6 to obtain (2.75). Substituting $\alpha_{|e|+1}^{(d',D)} < \alpha_{|e|}^{(d',D)}$ from (2.82) into (2.75), we obtain $\tilde{\alpha}_{e,D} < \alpha_{|e|}^{(d',D)}$. Combining this inequality with $\alpha_{|e|}^{(d',D)} > \frac{1}{1+w} > \alpha_{\min}^{(e,D)}$ from (2.82), we obtain (2.83), as desired.

To determine a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$, we observe

$\tilde{L}(d', D, 1/(1+w), w, C, C_1)$

$$\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n} n^{-}_{|e|,d',D} + \frac{w}{n} n^{+}_{|e|+1,d',D} + C + C_1 \lfloor \alpha^{(d',D)}_{|e|} - \alpha^{(e,D)}_{\min} \rfloor_+ \tag{2.84}$$

$$= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n} \left(\left(\frac{1}{\alpha^{(d',D)}_{|e|}} - 1 - w \right) n^{+}_{|e|,d',D} + w \tilde{n}^{+}_{e,D} \right) + C$$

$$+ C_1(\alpha^{(d',D)}_{|e|} - \alpha^{(e,D)}_{\min}) \tag{2.85}$$

where the last equality follows by substituting (2.60) and (2.61) in Observations 2.5.5 and 2.5.6 into (2.84). Using (2.83) and applying the same argument as in Step 4, the quantity labeled (2.85) is also lower-bounded by

$$\tilde{L}(e, D, 1/(1+w), w, C, C_1) + \inf_{\alpha^{(d',D)}_{|e|} : \zeta < \alpha^{(d',D)}_{|e|} \leq 1} g(\alpha^{(d',D)}_{|e|}),$$

so that we again have

$$\tilde{L}(d', D, 1/(1+w), w, C, C_1)$$

$$\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \inf_{\alpha^{(d',D)}_{|e|} : \zeta < \alpha^{(d',D)}_{|e|} \leq 1} g(\alpha^{(d',D)}_{|e|}).$$

Step 6. Put everything together.

Suppose, first, that $\mathcal{S}(\mathcal{X}, D, e, 1)$ is not empty.

In the case where $\mathcal{S}_1(\mathcal{X}, D, e, 1)$ is not empty, we observe the following inequality

$$\begin{aligned} &\inf_{d' \in \mathcal{S}_1(\mathcal{X}, D, e, 1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\ &\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \frac{1}{n} \left(\frac{1}{\alpha^{(e,D)}_{\min}} - 1 \right) \tilde{n}^{+}_{e,D} + C, \end{aligned} \tag{2.86}$$

which follows from the definition of inf being the greatest lower bound, as well as the lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for $d' \in \mathcal{S}_1(\mathcal{X}, D, e, 1)$, which we have derived in Step 3.

In the case where $\mathcal{S}_2(\mathcal{X}, D, e, 1) \cup \mathcal{S}_3(\mathcal{X}, D, e, 1)$ is not empty, we observe the following inequality

$$\begin{aligned} &\inf_{d' \in \mathcal{S}_2(\mathcal{X},D,e,1) \cup \mathcal{S}_3(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\ &\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \inf_{\beta:\zeta<\beta\leq 1} g(\beta), \end{aligned} \tag{2.87}$$

which follows from the definition of inf being the greatest lower bound, as well as the lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for $d' \in \mathcal{S}_2(\mathcal{X}, D, e, 1)$, which we have derived in Step 4, and the lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for $d' \in \mathcal{S}_3(\mathcal{X}, D, e, 1)$, which we have derived in Step 5.

To derive a lower bound of $\tilde{L}(d', D, 1/(1+w), w, C, C_1)$ for $d' \in \mathcal{S}(\mathcal{X}, D, e, 1)$, we further observe that if $\alpha_{\min}^{(e,D)} \leq \max(1/(1+w), \tilde{\alpha}_{e,D})$ holds, then by our remark in Step 3, $\mathcal{S}_1(\mathcal{X}, D, e, 1)$ is empty, and consequently, using (2.87), we have

$$\begin{aligned} &\inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\ &= \inf_{d' \in \mathcal{S}_2(\mathcal{X},D,e,1) \cup \mathcal{S}_3(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\ &\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \inf_{\beta:\zeta<\beta\leq 1} g(\beta). \end{aligned} \tag{2.88}$$

On the other hand, if $\alpha_{\min}^{(e,D)} > \max(1/(1+w), \tilde{\alpha}_{e,D})$ holds, then $\mathcal{S}_1(\mathcal{X}, D, e, 1)$ may or may not be empty. If, in addition, both $\mathcal{S}_1(\mathcal{X}, D, e, 1)$ and $\mathcal{S}_2(\mathcal{X}, D, e, 1) \cup \mathcal{S}_3(\mathcal{X}, D, e, 1)$ are not empty, then using (2.86) and (2.87), we have

$$\begin{aligned} &\inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\ &= \min \left(\inf_{d' \in \mathcal{S}_1(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1), \right. \end{aligned}$$

$$\left. \inf_{d' \in \mathcal{S}_2(\mathcal{X},D,e,1) \cup \mathcal{S}_3(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \right)$$

$$\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \min\left(\frac{1}{n}\left(\frac{1}{\alpha_{\min}^{(e,D)}} - 1 \right) \tilde{n}_{e,D}^{+} + C, \inf_{\beta:\zeta<\beta\leq 1} g(\beta) \right). \quad (2.89)$$

If either $\mathcal{S}_1(\mathcal{X}, D, e, 1)$ or $\mathcal{S}_2(\mathcal{X}, D, e, 1) \cup \mathcal{S}_3(\mathcal{X}, D, e, 1)$ is empty, then

$$\inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1)$$

is given by either

$$\inf_{d' \in \mathcal{S}_2(\mathcal{X},D,e,1) \cup \mathcal{S}_3(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1)$$

or

$$\inf_{d' \in \mathcal{S}_1(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1),$$

both of which are lower-bounded by the quantity labeled (2.89) because of (2.87) and (2.86).

Putting these cases together, we have

$$\inf_{d' \in \mathcal{S}(\mathcal{X},D,e,1)} \tilde{L}(d', D, 1/(1+w), w, C, C_1)$$

$$\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1)$$

$$+ \begin{cases} \min\left(\frac{1}{n}\left(\frac{1}{\alpha_{\min}^{(e,D)}} - 1 \right) \tilde{n}_{e,D}^{+} + C, \inf_{\beta:\zeta<\beta\leq 1} g(\beta) \right) & \text{if } \alpha_{\min}^{(e,D)} > \\ & \max(1/(1+w), \tilde{\alpha}_{e,D}), \\ \inf_{\beta:\zeta<\beta\leq 1} g(\beta) & \text{otherwise.} \end{cases} \quad (2.90)$$

Combining (2.65), (2.70), (2.71), and (2.90), we have

$$\tilde{L}(d, D, 1/(1+w), w, C, C_1)$$

$$\geq \tilde{L}(e, D, 1/(1+w), w, C, C_1)$$

$$+ \begin{cases} \min\left(\frac{1}{n}\left(\frac{1}{\alpha_{\min}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+} + C, \inf_{\beta:\zeta<\beta\leq 1} g(\beta), \frac{w}{n}\tilde{n}_{e,D}^{+} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_{+},\right. \\ \qquad \left.\frac{1}{n}\tilde{n}_{e,D}^{-} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_{+}\right) & \text{if } \alpha_{\min}^{(e,D)} > \max(1/(1+w), \tilde{\alpha}_{e,D}), \\ \min\left(\inf_{\beta:\zeta<\beta\leq 1} g(\beta), \frac{w}{n}\tilde{n}_{e,D}^{+} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_{+},\right. \\ \qquad \left.\frac{1}{n}\tilde{n}_{e,D}^{-} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_{+}\right) & \text{otherwise.} \end{cases} \tag{2.91}$$

Note that the quantity labeled (2.91) is precisely equal to $\tilde{L}^*(e, D, w, C, C_1)$ given by Equation (2.41) in the statement of the theorem, because:

(i) if $\alpha_{\min}^{(e,D)} > \max(1/(1+w), \tilde{\alpha}_{e,D})$ holds, then the first term in the minimum on the right-hand side of Equation (2.41) is precisely $\frac{1}{n}\left(\frac{1}{\alpha_{\min}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+} + C$;

(ii) if $\alpha_{\min}^{(e,D)} > \max(1/(1+w), \tilde{\alpha}_{e,D})$ does not hold, then we have $\alpha_{\min}^{(e,D)} \leq 1/(1+w)$ or $\alpha_{\min}^{(e,D)} \leq \tilde{\alpha}_{e,D}$: in the former case where $\alpha_{\min}^{(e,D)} \leq 1/(1+w)$ holds, we have

$$\frac{1}{n}\left(\frac{1}{\alpha_{\min}^{(e,D)}} - 1\right)\tilde{n}_{e,D}^{+} \geq \frac{w}{n}\tilde{n}_{e,D}^{+},$$

which implies that the first term in the minimum on the right-hand side of Equation (2.41) is bounded below by $\frac{w}{n}\tilde{n}_{e,D}^{+} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_{+}$, and thus has no influence over the computation of the minimum; in the latter case where $\alpha_{\min}^{(e,D)} \leq \tilde{\alpha}_{e,D}$ holds, the first term in the minimum on the right-hand side of Equation (2.41) is clearly bounded below by $\frac{w}{n}\tilde{n}_{e,D}^{+} + C_1\lfloor\tilde{\alpha}_{e,D} - \alpha_{\min}^{(e,D)}\rfloor_{+}$, and again has no influence over the computation of the minimum.

This proves that $\tilde{L}^*(e, D, w, C, C_1)$ given by Equation (2.41) is indeed a lower bound

of $\tilde{L}(d, D, 1/(1+w), w, C, C_1)$ for $d \in \mathcal{D}(\mathcal{X}, D, e)$, in the case where $\mathcal{S}(\mathcal{X}, D, e, 1)$ is not empty. In the case where $\mathcal{S}(\mathcal{X}, D, e, 1)$ is empty, using (2.65) and (2.71), along with the fact $\mathcal{D}(\mathcal{X}, D, e, 0) = \{\bar{e}\}$, we have

$$\begin{aligned}
&\tilde{L}(d, D, 1/(1+w), w, C, C_1) \\
&\geq \inf_{d' \in \mathcal{D}(\mathcal{X}, D, e, 0)} \tilde{L}(d', D, 1/(1+w), w, C, C_1) \\
&= \tilde{L}(\bar{e}, D, 1/(1+w), w, C, C_1) \\
&= \tilde{L}(e, D, 1/(1+w), w, C, C_1) + \min\left(\frac{w}{n}\tilde{n}^+_{e,D}, \frac{1}{n}\tilde{n}^-_{e,D}\right) + C_1 \lfloor \tilde{\alpha}_{e,D} - \alpha^{(e,D)}_{\min} \rfloor_+,
\end{aligned}$$

where the last quantity is clearly lower-bounded by $\tilde{L}^*(e, D, w, C, C_1)$ defined in Equation (2.41). We have now proven that $\tilde{L}^*(e, D, w, C, C_1)$ given by Equation (2.41) is a lower bound of $\tilde{L}(d, D, 1/(1+w), w, C, C_1)$ for $d \in \mathcal{D}(\mathcal{X}, D, e)$.

Finally, we compute $\inf_{\beta:\zeta<\beta\leq 1} g(\beta)$ analytically. Since the derivative of g is given by

$$g'(\beta) = -\frac{\tilde{n}^+_{e,D}}{n\beta^2} + C_1,$$

and β must be positive, the only stationary point β^* of g that could satisfy the constraint $\zeta < \beta^* \leq 1$ is given by $\beta^* = \sqrt{\tilde{n}^+_{e,D}/(C_1 n)}$, and the second derivative test confirms that β^* is a local minimum of g. It then follows that $\inf_{\beta:\zeta<\beta\leq 1} g(\beta)$ is given by

$$\inf_{\beta:\zeta<\beta\leq 1} g(\beta) = \begin{cases} g(\beta^*) & \text{if } \zeta < \beta^* \leq 1 \\ \min(g(\zeta), g(1)) & \text{otherwise.} \end{cases}$$

□

2.6 Experiments

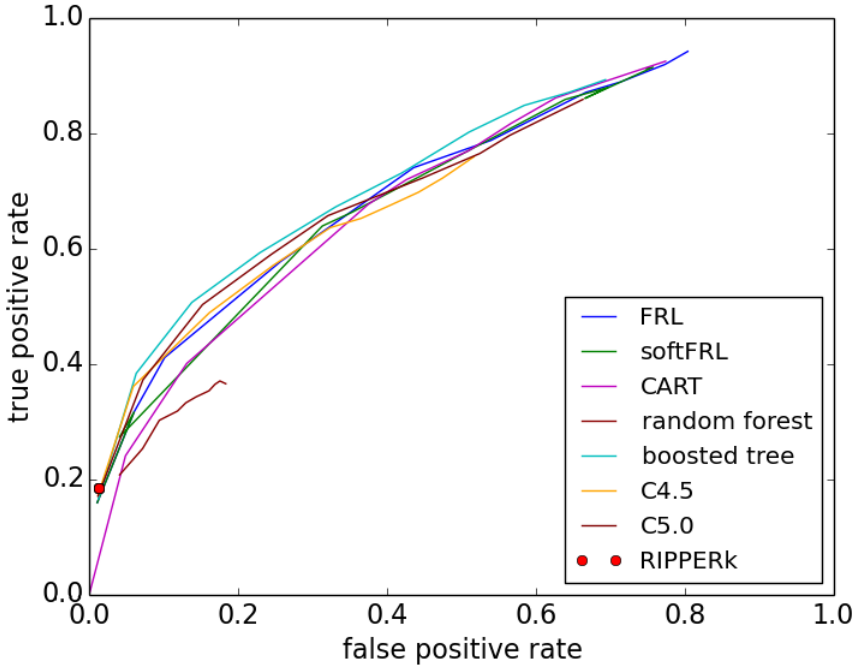
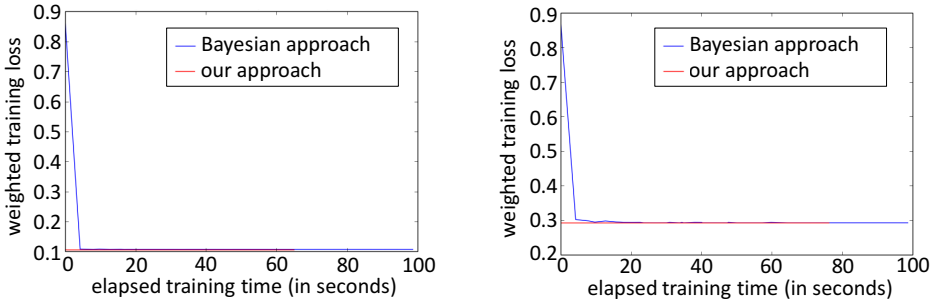


Figure 2.3: Comparison with other classification algorithms.

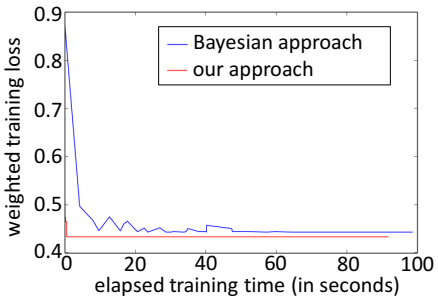
The comparison was performed using ROC curves on the test set using different w values for a random training-test split.

In this section, we demonstrate our algorithms for learning falling rule lists using a real-world application – learning the conditions that are predictive of the success of a bank marketing effort, from previous bank marketing campaign data. We used the public bank-full dataset [MLC11], which contains 45211 observations, with 12 predictor variables that were discretized. We used the frequent pattern growth (FP-growth) algorithm [HP00] to generate the set of antecedents A from the dataset. For reasons of model interpretability and generalizability, we included in A the antecedents that have at most 2 predicates, and have at least 10% support within the data that are labeled positive or within the data

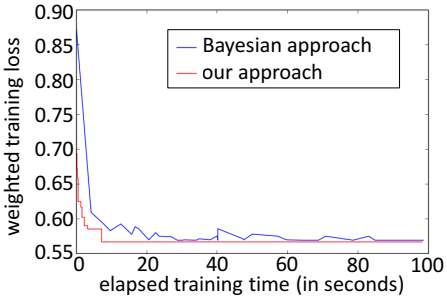


(a) Positive weight $w = 1$.

(b) Positive weight $w = 3$.



(c) Positive weight $w = 5$.

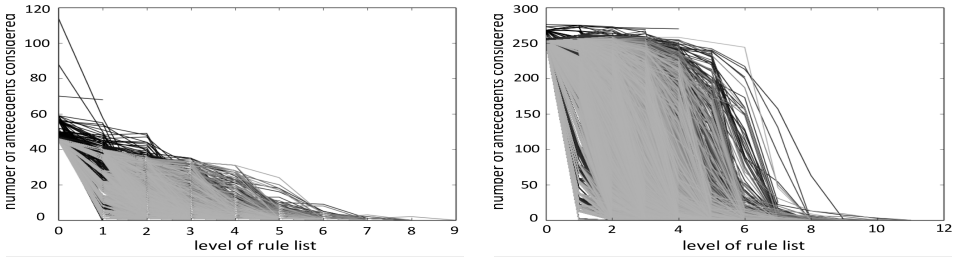


(d) Positive weight $w = 7$.

Figure 2.4: Weighted training loss over runtime for Bayesian and Algorithm FRL.

that are labeled negative. Besides the FP-growth algorithm, there is a vast literature on rule mining algorithms (e.g., [AS94, HPY00, LKDR05]), and any of these can be used to produce antecedents for our algorithms.

The bank-full dataset is imbalanced – there are only 5289 positive instances out of 45211 observations. A trivial model that always predicts the negative outcome for a bank marketing campaign will achieve close to 90% accuracy on this dataset, but it will not be useful for the bank to understand what makes a marketing campaign successful. Moreover, when predicting if a future campaign will be successful in finding a client, the bank cares more about "getting the positive right" than about "getting the negative right" – a false



(a) Number of antecedents considered by Algorithm FRL.

(b) Number of antecedents considered by Algorithm softFRL.

Figure 2.5: Effectiveness of prefix bounds.

negative means a substantial loss in revenue, while a false positive incurs little more than some phone calls.

2.6.1 Comparison with Other Classification Algorithms

We compared our algorithms with other classification algorithms in a cost-sensitive setting, where a false negative and a false positive have different costs of misclassification. We randomly split the dataset into a training and a test set, where 80% of the observations in the original dataset were placed into the training set. For each positive class weight $w \in \{1, 3, 5, 7, 9, 11, 13, 15, 17, 19\}$, we learned from the training set: (1) a falling rule list d, which is treated as a classifier $\tilde{d}_{1/(1+w)}$, using Algorithm FRL with $C = 0.000001$ (which is small enough so that no training accuracy will be sacrificed for sparsity), (2) a softly falling rule list d', which is treated as a classifier $\tilde{d}'_{1/(1+w)}$, using Algorithm softFRL with $C = 0.000001$ and $C_1 = 0.5$, (3) three decision trees using cost-sensitive CART [BFSO84], cost-sensitive C4.5 [Qui93], and cost-sensitive C5.0 [Qui04], respectively, (6) a random forest

[Bre01] of decision trees trained with cost-sensitive CART, (7) a boosted tree classifier using AdaBoost [FS96] on trees trained with cost-sensitive CART, and (8) a decision list using RIPPERk [Coh95], and we computed the true positive rate and the false positive rate on the test set for each classifier. For each split and for each algorithm, we plotted a receiver operating characteristic (ROC) curve on the test set using different values of w. Figure 2.3 shows the ROC curves for one of the training-test splits. Note that since RIPPERk is not a cost-sensitive algorithm, its ROC curve based on different w values has only a single point. As we can see, the curves in Figure 2.3 lie close to each other. This demonstrates the effectiveness of our algorithms in producing falling rule lists that, when used as classifiers, are comparable with classifiers produced by other widely used classification algorithms, in a cost-sensitive setting. This is possibly surprising since our models are much more constrained than other classification methods.

2.6.2 Comparison with Bayesian Falling Rule Lists

Since this paper was directly inspired by Wang and Rudin [WR15], who proposed a Bayesian approach to learning falling rule lists, we conducted a set of experiments comparing their work to ours. We trained falling rule lists on the entire bank-full dataset using both the Bayesian approach and our optimization approach, and plotted the weighted training loss over real runtime for each positive class weight $w \in \{1, 3, 5, 7\}$ with the threshold set to $1/(1 + w)$ (By Theorem 2.8, this is the threshold with the least weighted training loss for any given rule list). Since we want to focus our experiments on the efficiency of searching

the model space, the runtimes recorded do not include the time for mining the antecedents. Note that the Bayesian approach is not cost-sensitive, and does not optimize the weighted training loss directly. However, in many real-life applications such as predicting the success of a future marketing campaign, it is desirable to minimize the expected weighted loss. Therefore, it is reasonable to compare the two approaches using the weighted training loss to demonstrate the advantages of our optimization approach. We compared the Bayesian approach only with Algorithm FRL, because both methods strictly enforce the monotonicity constraint on the positive proportions of the training data that are classified into each rule. Softly falling rule lists do not strictly enforce the monotonicity constraint, and are therefore not used for comparison. Figure 2.4 shows the plots of the weighted training loss over real runtime. As shown in Figure 2.4, our optimization approach tends to find a falling rule list with a smaller weighted training loss faster than the Bayesian approach. This is not too surprising because in our approach, the search space is made substantially smaller by the tight bounds presented here, whereas in the original Bayesian approach, there are no tight bounds on optimal solutions to restrict the search space – even if we constructed bounds for the original Bayesian approach, they would involve loose approximations to gamma functions.

It is worth pointing out that both the Bayesian approach and our optimization approach produce similar falling rule lists. Table 2.2 shows a falling rule list for the bank-full dataset, obtained in a particular run of the Bayesian approach with 6000 iterations. Table 2.3 shows

a falling rule list for the same dataset, obtained in a particular run of Algorithm FRL with 3000 iterations and the positive class weight $w = 7$. As we can see, the top four rules in both falling rule lists are identical. Tables 2.4 and 2.5 show another pair of falling rule lists obtained using both approaches in different runs, and in this case, both approaches have identified some common rules for a high chance of marketing success. This means that both the Bayesian approach and our optimization approach tend to identify similar conditions that are significant, but our approach has the added advantage of faster training convergence over the Bayesian approach in general.

Table 2.2: Falling rule list for bank-full dataset, trained using the Bayesian approach.

	antecedent		prob.	+	−
IF	poutcome=success	THEN success prob. is	0.65	978	531
	AND default=no				
ELSE IF	$60 \leq$ age < 100	THEN success prob. is	0.29	426	1030
	AND loan=no				
ELSE IF	$17 \leq$ age < 30	THEN success prob. is	0.25	504	1539
	AND housing=no				
ELSE IF	campaign=1	THEN success prob. is	0.15	787	4471
	AND housing=no				
ELSE IF	education=tertiary	THEN success prob. is	0.12	460	3313
	AND housing=no				
ELSE IF	marital=single	THEN success prob. is	0.11	550	4331
	AND contact=cellular				
ELSE IF	contact=cellular	THEN success prob. is	0.08	1080	12709
ELSE		success prob. is	0.04	504	11998

2.6.3 Effectiveness of Prefix Bounds

We also plotted the number of antecedents considered by Algorithm FRL and Algorithm softFRL in the process of constructing a rule list at each iteration (Figures 2.5a and

Table 2.3: Falling rule list for bank-full dataset, trained using Algorithm FRL.

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1539
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.15	787	4471
ELSE		success prob. is	0.07	2594	32351

Table 2.4: Another falling rule list, trained using the Bayesian approach.

	antecedent		prob.	+	−
IF	poutcome=success AND housing=no	THEN success prob. is	0.70	729	311
ELSE IF	poutcome=success	THEN success prob. is	0.53	249	222
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1538
ELSE IF	education=tertiary AND housing=no	THEN success prob. is	0.14	790	4750
ELSE IF	marital=single AND contact=cellular	THEN success prob. is	0.12	648	4754
ELSE IF	$1000 \leq$ balance < 2000 AND housing=no	THEN success prob. is	0.11	135	1061
ELSE IF	campaign=1 AND contact=cellular	THEN success prob. is	0.10	571	4904
ELSE IF	contact=cellular AND loan=no	THEN success prob. is	0.08	587	6800
ELSE		success prob. is	0.04	650	14552

2.5b), when we applied the two algorithms to the entire dataset. Each curve in either plot corresponds to a rule list constructed in an iteration of the appropriate algorithm. The intensity of the curve is inversely proportional to the iteration number – the larger the iteration number, the lighter the curve is. The number of antecedents considered by

Table 2.5: Another falling rule list, trained using Algorithm FRL.

	antecedent		prob.	+	−
IF	poutcome=success AND housing=no	THEN success prob. is	0.70	729	311
ELSE IF	poutcome=success AND previous ≥ 2	THEN success prob. is	0.55	185	154
ELSE IF	poutcome=success AND default=no	THEN success prob. is	0.48	64	68
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.25	302	921
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.24	444	1413
ELSE IF	education=tertiary AND housing=no	THEN success prob. is	0.13	671	4435
ELSE		success prob. is	0.07	2468	31590

Algorithm FRL stays below 60 in all but a few early iterations (despite a choice of 276 antecedents available), and the number considered by either algorithm generally decreases drastically in each iteration after three or four antecedents have been chosen. The curves generally become lighter as we move vertically down the plots, indicating that as we find better rule lists, there are less antecedents to consider at each level. Algorithm softFRL needs to consider more antecedents in general since the search space is less constrained. All of these demonstrate that the prefix bounds we have derived for our algorithms are effective in excluding a large portion of the search space of rule lists. The supplementary material contains more rule lists created using our algorithms with different parameter values.

2.6.4 Effect of Varying Parameter Values

In this section, we include some additional rule lists created using Algorithm FRL and Algorithm softFRL with varying parameter values. The default parameter values we used in creating these rule lists are $w = 7$, $C = 0.000001$, and $C_1 = 0.5$. In each of the following subsections, the rule lists were created with default parameter values, other than the parameter that was being varied.

Effect of Varying w on Algorithm FRL

Running Algorithm FRL with $w = 1$ on the bank-full dataset produces the following falling rule list:

Table 2.6: Falling rule list created using Algorithm FRL with $w = 1$.

	antecedent		prob.	+	−
IF	poutcome=success AND loan=no	THEN success prob. is	0.65	934	495
ELSE IF	poutcome=success AND marital=married	THEN success prob. is	0.62	31	19
ELSE IF	poutcome=success AND campaign=1	THEN success prob. is	0.56	9	7
ELSE		success prob. is	0.10	4315	39401

Running Algorithm FRL with $w = 3$ on the bank-full dataset produces the following falling rule list:

Table 2.7: Falling rule list created using Algorithm FRL with $w = 3$.

	antecedent		prob.	+	−
IF	poutcome=success AND previous ≥ 2	THEN success prob. is	0.65	677	361
ELSE IF	poutcome=success AND campaign=1	THEN success prob. is	0.65	185	99
ELSE IF	poutcome=success AND loan=no	THEN success prob. is	0.63	111	65
ELSE IF	poutcome=success AND marital=married	THEN success prob. is	0.56	5	4
ELSE IF	$60 \leq$ age < 100 AND housing=no	THEN success prob. is	0.30	390	919
ELSE		success prob. is	0.09	3921	38474

Running Algorithm FRL with $w = 5$ on the bank-full dataset produces the following falling rule list:

Table 2.8: Falling rule list created using Algorithm FRL with $w = 5$.

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1539
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.23	242	796
ELSE		success prob. is	0.08	3139	36026

Running Algorithm FRL with $w = 7$ on the bank-full dataset produces the following falling rule list:

Table 2.9: Falling rule list created using Algorithm FRL with $w = 7$.

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND default=no	THEN success prob. is	0.28	434	1113
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1539
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.23	242	794
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.14	658	4092
ELSE IF	previous ≥ 2 AND education=tertiary	THEN success prob. is	0.13	108	707
ELSE		success prob. is	0.07	2365	31146

As the positive class weight w increases, the falling rule list created using Algorithm FRL tends to have rules whose probability estimates are smaller. This is not surprising – a larger value of w means a smaller threshold $\tau = 1/(1 + w)$, and by including rules whose probability estimates are not much larger than the threshold, the falling rule list produced by the algorithm will more likely predict positive, thereby reducing the (weighted) empirical risk of misclassification. Note that Algorithm FRL will never include rules whose probability estimates are less than the threshold (see Corollary 4.5).

Effect of Varying w on Algorithm softFRL

Running Algorithm softFRL with $w = 1$ on the bank-full dataset produces the following softly falling rule list:

Table 2.10: Softly falling rule list created using Algorithm softFRL with $w = 1$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success AND campaign=1	THEN prob. is	0.67	0.67	557	280
ELSE IF	poutcome=success AND marital=married	THEN prob. is	0.65	0.65	263	143
ELSE IF	poutcome=success AND loan=no	THEN prob. is	0.61	0.61	154	98
ELSE		prob. is	0.10	0.10	4315	39401

Note that there is an extra column "+ prop." (or *positive proportion*) in a table showing a softly falling rule list. This column gives the empirical positive proportion of each antecedent in the softly falling rule list. When the probability estimate of a rule is less than the positive proportion of the antecedent in the same rule, we know that the softly falling rule list has been transformed from a non-falling compatible rule list, and that the monotonicity penalty has been incurred in the process of running Algorithm softFRL.

Running Algorithm softFRL with $w = 3$ on the bank-full dataset produces the following softly falling rule list:

Table 2.11: Softly falling rule list created using Algorithm softFRL with $w = 3$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success AND marital=married	THEN prob. is	0.65	0.65	547	289
ELSE IF	poutcome=success AND loan=no	THEN prob. is	0.65	0.65	418	225
ELSE IF	poutcome=success AND campaign=1	THEN prob. is	0.56	0.56	9	7
ELSE IF	poutcome=success AND previous ≥ 2	THEN prob. is	0.33	0.33	4	8
ELSE IF	$60 \leq$ age < 100 AND housing=no	THEN prob. is	0.30	0.30	390	919
ELSE IF	previous ≥ 2 AND campaign=1	THEN prob. is	0.15	0.15	281	1559
ELSE		prob. is	0.09	0.09	3640	36915

Running Algorithm softFRL with $w = 5$ on the bank-full dataset produces the following softly falling rule list:

Table 2.12: Softly falling rule list created using Algorithm softFRL with $w = 5$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success	THEN prob. is	0.65	0.65	978	533
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN prob. is	0.29	0.29	426	1030
ELSE IF	poutcome=unknown AND contact=cellular	THEN prob. is	0.11	0.11	2380	18659
ELSE		prob. is	0.07	0.07	1505	19700

Running Algorithm softFRL with $w = 7$ on the bank-full dataset produces the following softly falling rule list:

Table 2.13: Softly falling rule list created using Algorithm softFRL with $w = 7$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success	THEN prob. is	0.65	0.65	978	533
ELSE IF	$60 \leq$ age < 100	THEN prob. is	0.28	0.28	435	1120
ELSE IF	marital=single AND housing=no	THEN prob. is	0.18	0.18	970	4504
ELSE IF	contact=cellular AND default=no	THEN prob. is	0.10	0.10	2255	19970
ELSE		prob. is	0.05	0.05	651	13795

As the positive class weight w increases, the softly falling rule list created using Algorithm softFRL also tends to have rules whose probability estimates are smaller. This is again not surprising – a larger value of w means a smaller threshold $\tau = 1/(1 + w)$, and by including rules whose probability estimates are not much larger than the threshold, the softly falling rule list produced by the algorithm will more likely predict positive, thereby reducing the (weighted) empirical risk of misclassification.

Effect of Varying C on Algorithm FRL

Running Algorithm FRL with $C = 0.000001$ on the bank-full dataset produces the following falling rule list:

Table 2.14: Falling rule list created using Algorithm FRL with $C = 0.000001$.

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND default=no	THEN success prob. is	0.28	434	1113
ELSE IF	$17 \leq$ age < 30 AND housing=no	THEN success prob. is	0.25	504	1539
ELSE IF	previous ≥ 2 AND housing=no	THEN success prob. is	0.23	242	794
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.14	658	4092
ELSE IF	previous ≥ 2 AND education=tertiary	THEN success prob. is	0.13	108	707
ELSE		success prob. is	0.07	2365	31146

Running Algorithm FRL with $C = 0.01$ on the bank-full dataset produces the following falling rule list:

Table 2.15: Falling rule list created using Algorithm FRL with $C = 0.01$.

	antecedent		prob.	+	−
IF	poutcome=success AND default=no	THEN success prob. is	0.65	978	531
ELSE IF	$60 \leq$ age < 100 AND loan=no	THEN success prob. is	0.29	426	1030
ELSE IF	$17 \leq$ age < 30 AND contact=cellular	THEN success prob. is	0.20	653	2621
ELSE IF	campaign=1 AND housing=no	THEN success prob. is	0.15	803	4634
ELSE		success prob. is	0.07	2429	31106

Running Algorithm FRL with $C = 0.1$ on the bank-full dataset produces the following falling rule list:

Table 2.16: Falling rule list created using Algorithm FRL with $C = 0.1$.

	antecedent		prob.	+	−
IF	housing=no	THEN success prob. is	0.20	2883	11799
	AND contact=cellular				
ELSE		success prob. is	0.08	2406	28123

As the cost C of adding a rule increases, the size of the falling rule list created by Algorithm FRL decreases, as expected.

Effect of Varying C on Algorithm softFRL

Running Algorithm softFRL with $C = 0.000001$ on the bank-full dataset produces the following softly falling rule list:

Table 2.17: Softly falling rule list created with $C = 0.000001$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success	THEN prob. is	0.65	0.65	978	533
ELSE IF	$60 \leq$ age < 100	THEN prob. is	0.28	0.28	435	1120
ELSE IF	marital=single	THEN prob. is	0.18	0.18	970	4504
	AND housing=no					
ELSE IF	contact=cellular	THEN prob. is	0.10	0.10	2255	19970
	AND default=no					
ELSE		prob. is	0.05	0.05	651	13795

Running Algorithm softFRL with $C = 0.01$ on the bank-full dataset produces the following softly falling rule list:

Table 2.18: Softly falling rule list created with $C = 0.01$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success AND loan=no	THEN prob. is	0.65	0.65	934	495
ELSE IF	housing=no AND contact=cellular	THEN prob. is	0.16	0.16	2245	11535
ELSE IF	housing=yes AND default=no	THEN prob. is	0.07	0.07	1677	22591
ELSE		prob. is	0.07	0.08	433	5301

Running Algorithm softFRL with $C = 0.1$ on the bank-full dataset produces the following softly falling rule list:

Table 2.19: Softly falling rule list created with $C = 0.1$.

	antecedent		prob.	+ prop.	+	−
IF	housing=no AND contact=cellular	THEN prob. is	0.20	0.20	2883	11799
ELSE		prob. is	0.08	0.08	2406	28123

As the cost C of adding a rule increases, the size of the softly falling rule list created by Algorithm softFRL decreases, as expected.

Effect of Varying C_1 on Algorithm softFRL

Running Algorithm softFRL with $C_1 \in \{0.005, 0.05, 0.5\}$ on the bank-full dataset produces the softly falling rule lists shown in Tables 2.20, 2.21, and 2.22.

Table 2.20: Softly falling rule list created using Algorithm softFRL with $C_1 = 0.005$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success	THEN prob. is	0.65	0.65	978	533
ELSE IF	$60 \leq$ age < 100 AND housing=no	THEN prob. is	0.30	0.30	599	1177
ELSE IF	marital=single AND housing=no	THEN prob. is	0.18	0.18	970	4504
ELSE IF	marital=single AND previous=0	THEN prob. is	0.08	0.08	456	4936
ELSE IF	campaign ≥ 3 AND education=secondary	THEN prob. is	0.06	0.06	323	5294
ELSE IF	$30 \leq$ age < 40 AND previous=0	THEN prob. is	0.06	0.08	568	6849
ELSE IF	education=tertiary AND housing=no	THEN prob. is	0.06	0.14	361	2237
ELSE IF	loan=yes AND previous=0	THEN prob. is	0.05	0.05	106	1972
ELSE IF	education=secondary AND default=no	THEN prob. is	0.05	0.09	595	5779
ELSE IF	campaign=1	THEN prob. is	0.05	0.08	233	2564
ELSE IF	housing=no AND previous=0	THEN prob. is	0.05	0.05	68	1176
ELSE IF	job=management AND contact=cellular	THEN prob. is	0.05	0.10	75	693
ELSE IF	job=technician AND poutcome=unknown	THEN prob. is	0.05	0.07	10	143
ELSE IF	marital=married	THEN prob. is	0.05	0.06	110	1841
ELSE IF	campaign ≥ 3 AND housing=yes	THEN prob. is	0.05	0.06	16	238
ELSE IF	marital=single AND housing=yes	THEN prob. is	0.05	0.13	13	91
ELSE IF	housing=yes AND contact=cellular	THEN prob. is	0.05	0.10	8	69
ELSE IF	job=blue-collar AND loan=no	THEN prob. is	0.05	0.16	4	21
ELSE		prob. is	0.05	0.07	5	63

Table 2.21: Softly falling rule list created using Algorithm softFRL with $C_1 = 0.05$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success AND default=no	THEN prob. is	0.65	0.65	978	531
ELSE IF	housing=yes	THEN prob. is	0.07	0.07	1686	22974
ELSE IF	$50 \leq$ age < 60 AND poutcome=unknown	THEN prob. is	0.07	0.09	367	3806
ELSE IF	contact=cellular AND default=no	THEN prob. is	0.07	0.18	1927	8961
ELSE IF	campaign=1 AND poutcome=unknown	THEN prob. is	0.07	0.08	126	1374
ELSE IF	campaign ≥ 3 AND loan=no	THEN prob. is	0.07	0.08	93	1110
ELSE IF	campaign=2 AND education=tertiary	THEN prob. is	0.07	0.09	18	192
ELSE IF	loan=no	THEN prob. is	0.07	0.10	72	648
ELSE		prob. is	0.06	0.06	22	326

Table 2.22: Softly falling rule list created using Algorithm softFRL with $C_1 = 0.5$.

	antecedent		prob.	+ prop.	+	−
IF	poutcome=success	THEN prob. is	0.65	0.65	978	533
ELSE IF	$60 \leq$ age < 100	THEN prob. is	0.28	0.28	435	1120
ELSE IF	marital=single AND housing=no	THEN prob. is	0.18	0.18	970	4504
ELSE IF	contact=cellular AND default=no	THEN prob. is	0.10	0.10	2255	19970
ELSE		prob. is	0.05	0.05	651	13795

When the monotonicity penalty C_1 is small, the softly falling rule list created by Algorithm softFRL exhibits the "pulling down" of the empirical positive proportion for a substantial number of rules, because with little monotonicity penalty the algorithm will more likely choose a rule list that frequently violates monotonicity but that has a small empirical risk on the training set, in the hope of getting more of the training instances

"right". This is also why the softly falling rule list tends to be longer when C_1 is small: in minimizing the empirical risk on the training set with little regularization (the default $C = 0.000001$ is very small), the algorithm tends to overfit the training data.

When C_1 becomes larger, the softly falling rule list created by Algorithm softFRL exhibits less "pulling down" of the empirical positive proportion. This is consistent with our expectation that when C_1 is larger, the penalty for violating monotonicity is higher and the algorithm will less likely choose a rule list that frequently violates monotonicity.

2.7 Discussion

We have proposed an optimization approach to learning falling rule lists and softly falling rule lists, along with Monte-Carlo search algorithms that use bounds on the optimal solution to prune the search space. A recent work by Angelino et al. [ALSA$^+$17] on (non-falling) rule lists showed that it is possible to exhaustively optimize an objective over rule lists, indicating that the space of lists is not as large as one might think. Our search space is a dramatically constrained version of their search space, allowing us to reasonably believe that it can be searched exhaustively. Unfortunately, almost none of the logic of [ALSA$^+$17] can be used here. Indeed, introducing the falling constraint or the monotonicity penalty changes the nature of the problem, and the bounds in our work are entirely different. The algorithm of [ALSA$^+$17] is not cost-sensitive, which led in this work to another level of complexity for the bounds.

Falling rule lists are optimized for ease-of-use – users only need to check a small number of conditions to determine whether an observation is in a high risk or high probability subgroup. As pointed out by Wang and Rudin [WR15], the monotonicity in probabilities in falling rule lists allows doctors to identify the most at-risk patients easily. Typical decision tree methods (CART, C4.5, C5.0) do not have the added interpretability that comes from the falling constraint in falling rule lists: one may have to check many conditions in a decision tree to determine whether an observation is in a high risk or high probability subgroup – even if the decision tree has a small depth, it is possible that high risk subgroups are in different parts of the tree, so that one still has to check many conditions in order to find high risk subgroups. In this sense, falling rule lists and softly falling rule lists are as sparse as we need them to be, and they can provide valuable insight into data.

Code: The code for Algorithm FRL and Algorithm softFRL is available at

`https://github.com/cfchen-duke/FRLOptimization`

Chapter 3

Predicate-based Interpretability: Two-layer Additive Models

In many real-life applications, we want to learn a predictive model that explains its decisions using predicates (i.e., decision factors). For example, when assessing the risk of cardiovascular diseases, we want a model that can explain why a particular patient is at high risk, using risk factors such as "high blood pressure" or "high blood sugar." At the same time, we also want the model to respect our prior knowledge of how a particular feature should be related to an outcome: in cardiovascular risk assessment, we expect our model to assign higher risk to patients with higher blood pressure, because high blood pressure generally increases the risk of cardiovascular diseases" [Kan96].

A two-layer additive risk model is a predicate-based interpretable model that is designed to explain its decisions using risk factors and to preserve the monotonic relationship between certain features and predicted risk. It resembles a traditional subscale model, where features are partitioned into meaningful subgroups (called *subscales*), and the subgroup scores are

later combined into a global model. Traditional subscale models are generally interpretable because they are decomposable into meaningful components, and because these models are usually linear with coefficients whose sign is positive for risk factors. Our model preserves these classical elements – it is decomposable into subscale models, and uses linear modeling with positive coefficients for risk factors, but inserts (interpretable) nonlinearities in several places to make the model more flexible and accurate. In particular, our model transforms each original feature into a piecewise-constant risk-scoring function that monotonically increases or decreases with respect to the original feature, if we constrain the risk-scoring function to behave in such a way. These piecewise-constant risk-scoring functions are combined within each subscale to form the subscale models in the first layer of our two-layer additive model. In particular, each subscale model uses sigmoid nonlinearity to transform the risk score computed from the sum of the risk-scoring functions whose underlying features belong to the subscale. This has the effect of adding more perceptron-like flexibility to the model, but also makes the output of each subscale model more meaningful – we can interpret each subscale model as a "mini-scoring model" that produces its own probability of risk, based on a subset of related features. In the second layer, the subscale probabilities are combined using a weighted sum and sent through the sigmoid function to produce the final probability of risk.

In this chapter, I will present the two-layer additive risk model in the context of credit risk prediction (i.e., risk of defaulting on a loan). The materials in this chapter were jointly

developed with Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang, and were presented at the NIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy [CLR+18].

3.1 Related Work

Our two-layer additive risk model is a globally interpretable model. This means that the entire reasoning process of our model is reflected in the full computation of the model, and can be easily explained to humans. In our credit risk application, we created an interactive display that shows the full computation of a two-layer additive risk model from beginning to end, without hiding any nonlinearities or computations from the user. Risk factors are colored according to their contributions to the model. The form of our model lends itself naturally to explaining predictions using risk factors, and understanding monotonicity constraints, through the visualization.

Attempts to create globally interpretable models for financial, healthcare, or other high-stakes applications use mainly standard machine learning approaches (e.g., decision trees and support vector machines were used for bank direct marketing [MLC11, MCR14]). Most of these standard methods cannot model the monotonic relationship between certain features and predicted risk. There are some works on decision tree learning methods that enforce (or improve) the monotonic relationship between features and outcome: these works either use

pruning to remove subtrees that violate the monotonic relationship [BDSP89, BD92, FP03], or they include a regularization term that penalizes potential violation of the monotonic relationship in greedy splitting criteria [BD95]. However, these methods either cannot optimize the accuracy while enforcing the monotonic relationship at the same time, or they cannot fully enforce the monotonic relationship. There are also works that find rule lists [LRMM15, WR15, ALSA$^+$17, CR18], but rules are not natural for datasets with many real-valued features. On the other hand, additive scoring models are natural for real-valued features and can easily preserve monotonicity between certain features and risk. In the past, such models were often created by domain experts. For example, the CHADS_2 score for predicting the atrial fibrillation stroke risk was created by a group of medical professionals [GWS$^+$01]. There are recent works that used machine learning techniques to create medical scoring systems [UR16, UR17]. Our two-layer additive risk model differs from traditional scoring models in that it partitions features into meaningful subgroups (called *subscales*) and includes a second layer that introduces more nonlinearities. The resulting model is a hybrid of additive scoring models and a small neural network, and is more expressive than traditional scoring models.

There are a number of recent works in explainable credit risk modeling, including [DGW18], [GCV$^+$18], and [GHYB20]. In particular, Dash et al. [DGW18] used column generation to efficiently search over an exponential number of candidate conjunctive or disjunctive clauses without the need for heuristic rule mining. Grath et al. [GCV$^+$18]

proposed positive counterfactuals and introduced several weighting strategies to generate more understandable counterfactuals. Gomez et al. [GHYB20] created an interactive visual analytics tool that generates counterfactual explanations to enhance the explainability of machine learning models.

3.2 Two-layer Additive Risk Model and Its Visualization

In this section, we present our two-layer additive risk model in the context of credit risk prediction. We work with a dataset released by the Fair Isaac Corporation (FICO) in 2018 for the Explainable Machine Learning Challenge [1]. We use $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ to denote the data, where $\mathbf{x}_i \in \mathbb{R}^P$ is a vector of the features in the dataset. The labels are indicators of defaulting on a loan: $y_i \in \{0, 1\}$. Let $\mathcal{P}$ represent the set of features with $|\mathcal{P}| = P$.

3.2.1 Model Design

The design of our two-layer additive risk model pursues interpretability by integrating predicates (i.e., risk factors) into the model and enforcing monotonicity constraints on certain features. Here we illustrate our design strategies.

First, to integrate predicates (i.e., risk factors) into the model, we discretize each feature into intervals. To ensure that our model respect the monotonic relationship between any given feature and predicted risk, we transform the feature into *one-sided* intervals and

[1] https://community.fico.com/s/explainable-machine-learning-challenge

constrain the risk scores assigned to these one-sided intervals to be non-negative. More concretely, for a monotonically decreasing feature $x_{\cdot,p}$, we create the following binary features of one-sided intervals:

$$b_{p,1}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} < \theta_1],$$

$$b_{p,2}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} < \theta_2],$$

$$...$$

$$b_{p,L_p}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} < \theta_{L_p}],$$

$$b_{p,0}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} \text{ is not missing}],$$

where $b_{p,0}(x_{\cdot,p})$ is a special indicator for non-missing values[2], L_p is the number of binary features created for feature p (not counting the special indicators for non-missing values and for missing values), and $\theta_1, \theta_2, ...,$ and θ_{L_p} satisfy

$$\theta_1 < \theta_2 < ... < \theta_{L_p}.$$

To choose the proper thresholds θ_l for creating the binary features, we used heuristic measures of "purity" of splitting the dataset by a specific candidate threshold and then choose a threshold yielding the "purest" group. In particular, we used information gain, which is a commonly used metric in classification algorithms such as decision trees, to measure how informative splitting at a particular threshold is to predicting a target variable. We split the data repeatedly using a greedy approach, each time finding a threshold θ^* that

[2] We handle missing values by creating binary indicators to indicate that the values are missing.

results in the largest information gain. Mathematically, this means that we set

$$\theta^* = \arg\min_{\theta} E(\theta),$$

where $E(\theta)$ is the entropy of splitting the subset of training data (that are currently being split) at θ:

$$\begin{aligned} E(\theta) =& \frac{n_1}{n}\left(-\frac{n_1^+}{n_1}\log\frac{n_1^+}{n_1} - \frac{n_1^-}{n_1}\log\frac{n_1^-}{n_1}\right) \\ &+ \frac{n_2}{n}\left(-\frac{n_2^+}{n_2}\log\frac{n_2^+}{n_2} - \frac{n_2^-}{n_2}\log\frac{n_2^-}{n_2}\right). \end{aligned}$$

In the equation above, n is the number of training examples that are currently being split, n_1 (or n_2) is the number of training examples that are currently being split and that satisfy $x_{\cdot,p} < \theta$ (or $x_{\cdot,p} \geq \theta$, respectively), n_1^+ (or n_1^-) is the number of positive (or negative, respectively) training examples that are currently being split and that satisfy $x_{\cdot,p} < \theta$, and n_2^+ (or n_2^-) is the number of positive (or negative, respectively) training examples that are currently being split and that satisfy $x_{\cdot,p} \geq \theta$. In this way, our method has the flexibility in deciding where to "bump up" the risk score for each individual feature.

Note that all of the binary features $b_{p,1}(x_{\cdot,p}), b_{p,2}(x_{\cdot,p}), ..., b_{p,L_p}(x_{\cdot,p})$ created for the monotonically decreasing feature $x_{\cdot,p}$ use one-sided intervals. This choice was made because assigning risk scores to these one-sided intervals is equivalent to a weighted sum of the one-sided binary features, which yields a *monotonic* piecewise constant risk-scoring function for the given feature if the risk scores for the one-sided intervals are non-negative. In other words, by enforcing that the constraints that the coefficients for $b_{p,1}(x_{\cdot,p}), b_{p,2}(x_{\cdot,p}), ...,$ and $b_{p,L_p}(x_{\cdot,p})$ must be non-negative, we shall guarantee a monotonically decreasing relationship

between the original continuous feature $x_{\cdot,p}$ and the predicted risk of default. To see this, let f_p denote the risk-scoring function for the feature $x_{\cdot,p}$. Note that

$$f_p(x_{\cdot,p}) = \beta_{p,1} b_{p,1}(x_{\cdot,p}) + \beta_{p,2} b_{p,2}(x_{\cdot,p}) + ... + \beta_{p,L_p} b_{p,L_p}(x_{\cdot,p}) + \beta_{p,0} b_{p,0}(x_{\cdot,p})$$

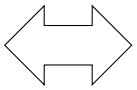
can be equivalently written as

$$\begin{aligned} f_p(x_{\cdot,p}) \quad = \quad & (\beta_{p,1} + \beta_{p,2} + ... + \beta_{p,L_p} + \beta_{p,0}) \mathbb{1}[x_{\cdot,p} < \theta_1] \\ & + (\beta_{p,2} + ... + \beta_{p,L_p} + \beta_{p,0}) \mathbb{1}[\theta_1 \leq x_{\cdot,p} < \theta_2] \\ & + ... \\ & + (\beta_{p,L_p} + \beta_{p,0}) \mathbb{1}[\theta_{L_p - 1} \leq x_{\cdot,p} < \theta_{L_p}] + \beta_{p,0} \mathbb{1}[\theta_{L_p} \leq x_{\cdot,p}], \end{aligned}$$

which can be displayed as a traditional scoring system (e.g., [UR16, Med18]). We show a transformation like this in Figure 3.1 for the ExternalRiskEstimate feature, which is monotonically decreasing with respect to credit risk.

If coefficients $\beta_{p,1}, \beta_{p,2}, ...,$ and β_{p,L_p} are nonnegative, it is easy to see that the risk-scoring function f_p for the feature $x_{\cdot,p}$ is monotonically decreasing – note that we do not need to enforce the non-negativity constraint on $\beta_{p,0}$ to make f_p monotonically decreasing. If instead, we would like to constrain f_p to be monotonically increasing, we reverse the above one-sided inequalities "$<$" into "$\geq$". Of course, if a feature $x_{\cdot,p}$ has no desired monotonicity, we discretize the feature into two-sided intervals, create binary features of the form $b_{p,l}(x_{\cdot,p}) = \mathbb{1}[\theta_{l-1} \leq x_{\cdot,p} < \theta_l]$, and drop the non-negativity constraints on the coefficients (i.e., the assigned risk scores) for the intervals during training.

Intervals	Points
< 64	+0.782
< 71	+0.748
< 76	+0.595
< 81	+0.768
< Inf	-1.094
-7	0
-8	0
-9	+1.094



Intervals	Points
0-63	+1.799
64-70	+1.017
71-75	+0.269
76-80	-0.326
81-Inf	-1.094
-7	0
-8	0
-9	+1.094

External Risk Estimate

Figure 3.1: "External Risk Estimate" feature.

If a person has an external risk estimate of $x = 33$, the person will satisfy the intervals $x < 64$, $x < 71$, $x < 76$, $x < 81$, and $x < \text{Inf}$. This means that the person will receive $0.782 + 0.748 + 0.595 + 0.768 + (-1.094) = 1.799$ points. -7, -8, and -9 are special values indicating that the feature value is missing, for three different reasons.

The FICO dataset comes with three special values: -7, -8, and -9, to denote that a particular feature value is missing, for three different reasons. For each feature $x_{\cdot,p}$, we also created three special binary features: $b_{p,-7}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} = -7]$, $b_{p,-8}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} = -8]$, and $b_{p,-9}(x_{\cdot,p}) = \mathbb{1}[x_{\cdot,p} = -9]$, and add the terms $\beta_{p,-7}b_{p,-7}(x_{\cdot,p}) + \beta_{p,-8}b_{p,-8}(x_{\cdot,p}) + \beta_{p,-9}b_{p,-9}(x_{\cdot,p})$ to $f_p(x_{\cdot,p})$, to handle the missing values.

Using domain knowledge obtained from the data description, we partition the features $\mathcal{P}$ into different subsets called subscales:

$$\mathcal{P} = \cup_{k=1}^{K} \mathcal{P}^{[k]},$$

where $\mathcal{P}^{[k]}$ is a subset of features assigned to the k-th subscale. In our experiments on the FICO dataset, we created 10 subscales, each containing one to four original features that

are related to each other. For each subscale, we build a mini-scoring model for predicting the probability of failure to repay a loan, using only the features designated for the subscale. Mathematically, each subscale model predicts a probability of default, denoted by $r^{[k]}$ for the k-th subscale, as follows:

$$r^{[k]}(\mathbf{x}) = \sigma\left(b_k + \sum_{p \in \mathcal{P}^{[k]}} f_p(x_{\cdot,p})\right)$$

$$= \sigma\left(b_k + \sum_{p \in \mathcal{P}^{[k]}} \sum_{l \in \{-7,-8,-9,0,1,\ldots,L_p\}} \beta_{p,l} b_{p,l}(x_{\cdot,p})\right),$$

where $\sigma(\cdot)$ represents the logistic sigmoid function, b_k is the bias term for the k-th subscale. Note that the equation above can be viewed as a scoring model that assigns $\beta_{p,l}$ points to the binary condition $b_{p,l}(x_{\cdot,p})$, for features $x_{\cdot,p}$ that are assigned to the k-th subscale. Alternatively, we can view the equation as a scoring model that assigns risk scores according to the risk-scoring functions f_p, for each original feature $x_{\cdot,p}$ belonging to the k-th subscale

Finally, the subscale probabilities are combined using a weighted sum and nonlinearly transformed into a final probability, denoted by r, of failure to repay a loan:

$$r(\mathbf{x}) = \sigma\left(b + \sum_{k=1}^{K} \alpha_k r^{[k]}(\mathbf{x})\right),$$

where b is the bias term and α_k is the coefficient for the probability predicted by the k-th subscale. The contribution of the k-th subscale to the final prediction can be easily observed by its weighted subscale probability $\alpha_k r^{[k]}(\mathbf{x})$.

Our model is called a "two-layer additive risk model" because we view the subscale models as the first layer that transforms input into subscale probabilities, and view the

transformation of subscale probabilities into a final predicted probability as the second layer. Conceptually, our two-layer additive risk model has a form of a small (two-layer) neural network. However, it differs from a regular neural network in multiple ways: unlike our model, a regular neural network does not partition features into meaningful subgroups; neither does it transform the original features into monotonic piecewise-constant risk-scoring functions.

3.2.2 Training the Model

The simplest way to train coefficients $\boldsymbol{\beta}^{[k]} = \{\beta_{p,l} \text{ for } p \in \mathcal{P}^{[k]}, l \in [0, L_p]\}$ is to treat each subscale $r^{[k]}$ as an independent logistic regression model with the $\{y_i\}_{i=1}^N$ being the target variables, using regularization (e.g., ℓ_2) to prevent overfitting, and non-negativity constraints on the coefficients to enforce monotonicity. Mathematically, we minimize the (regularized) logistic loss of the k-th subscale subject to non-negativity constraints, to train the k-th subscale:

$$\min_{b_k, \boldsymbol{\beta}^{[k]}} \frac{1}{N} \sum_{i=1}^{N} L(r^{[k]}(\mathbf{x}_i), y_i) + \|\boldsymbol{\beta}^{[k]}\|_2^2$$

subject to

$$\beta_{p,l} \geq 0 \quad \text{for all monotonic features } x_{\cdot,p} \text{ and } l \in \{1, ..., L_p\}.$$

L denotes the logistic loss function: $L(\hat{y}, y) = -y \log \hat{y} - (1 - y) \log \hat{y}$. After we have trained all the subscales, we train the final model similarly, by minimizing the (regularized) logistic

loss of the final model subject to non-negativity constraints:

$$\min_{b,\alpha_1,...,\alpha_K} \frac{1}{N}\sum_{i=1}^{N} L(r(\mathbf{x}_i), y_i) + \sum_{k=1}^{K} \alpha_k^2 \quad \text{subject to } \alpha_k \geq 0 \text{ for all } k \in \{1, ..., K\}.$$

We enforce the non-negativity constraints on the α_k's, so that the final risk probability monotonically increases with each subscale probability. After we have trained the subscales and the final model separately, it is possible to fine-tune the subscales and the final model jointly, using (stochastic) gradient descent with a small learning rate (so that the non-negativity constraints will not be violated).

3.2.3 Model Visualization and Accuracy

An image of the full model is shown in Figure 3.2, where the colors indicate the final contribution to the combined score. Red indicates more likely to default on the loan. The 23 feature values can be entered on the left, and clicking on any of the ten subscales (in the second colored layer) reveals a pop-up window with the calculation, as shown in Figure 3.3 for two subscales. The final combination of features is shown in Figure 3.2 (right panel). Figure 3.4 shows that our global model does not lose accuracy over other machine learning techniques, despite being constrained to be interpretable. The accuracy results were obtained by averaging test accuracy figures of five random 80% : 20% training-test splits. Our final model (for visualization purposes) was trained on the entire dataset.

Figure 3.2: Visualization of two-layer additive risk model.

Left: Snapshot of visualization tool showing the two-layer additive risk model. Colors indicate contribution to the final score. The 23 feature values are entered on the left. *Right*: Final combined risk score.

3.2.4 Explaining Predictions Using Risk Factors

Our two-layer additive risk model comes naturally with a way to identify a list of factors that contribute most heavily to the final prediction. Table 3.1 shows a list of four factors that are important for predicting observation "Demo 1" to have 95.2% risk of default (i.e., bad risk performance).

To identify the factors, we first identify the most important two subscales and then the

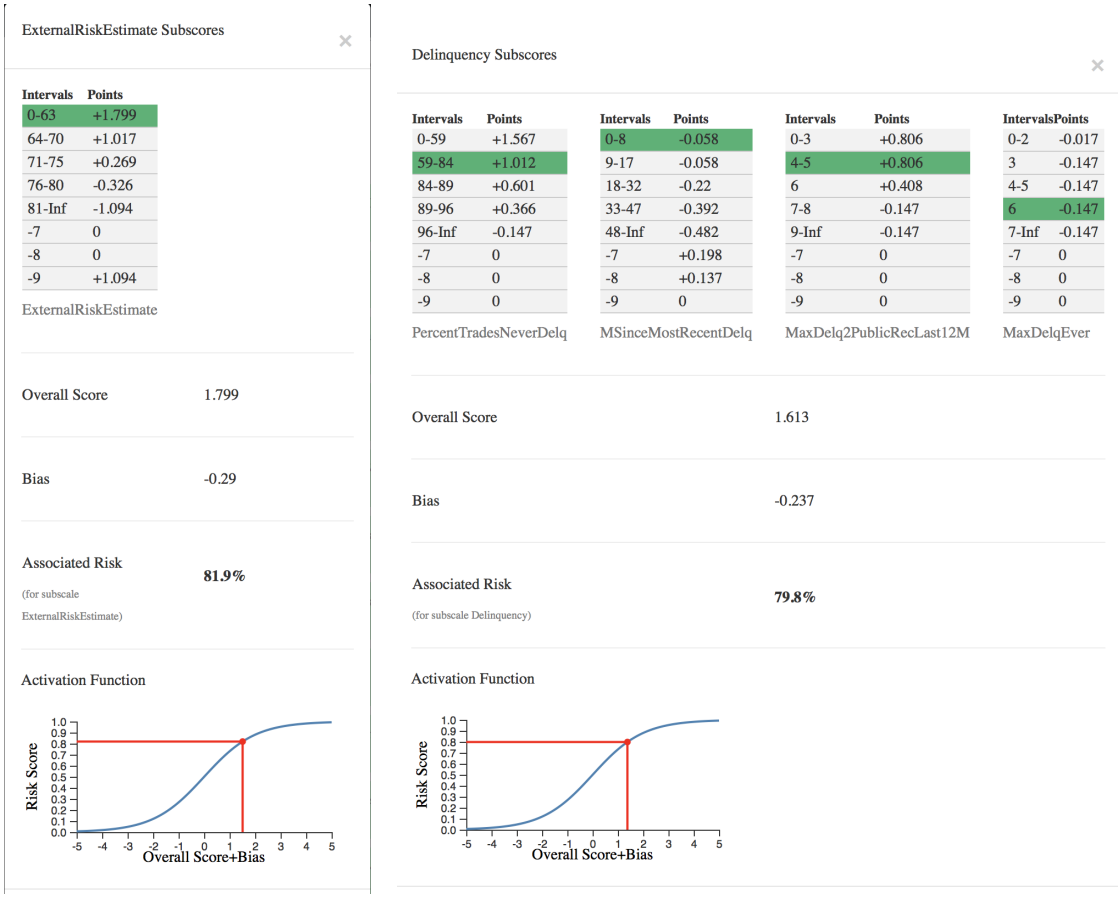


Figure 3.3: The "External Risk Estimate" and "Delinquency" subscales.

Left: The transformation of points into a risk estimate, which is 81.9% for this person. The risk estimate for the subscale is meaningful as its own "mini-model" of risk, based only on the ExternalRiskEstimate feature. *Right*: A different subscale, which uses multiple features.

most important factors within each subscale. The importance of each subscale in the final model is determined by its weighted score, which is the product of the subscale's output and its coefficient – the larger the product, the larger the contribution of the term in the final risk.

For example, for "Demo 1", the two most important subscales are Delinquency (with points of 1.973) and TradeOpenTime (with points of 1.947). Then, within each of the

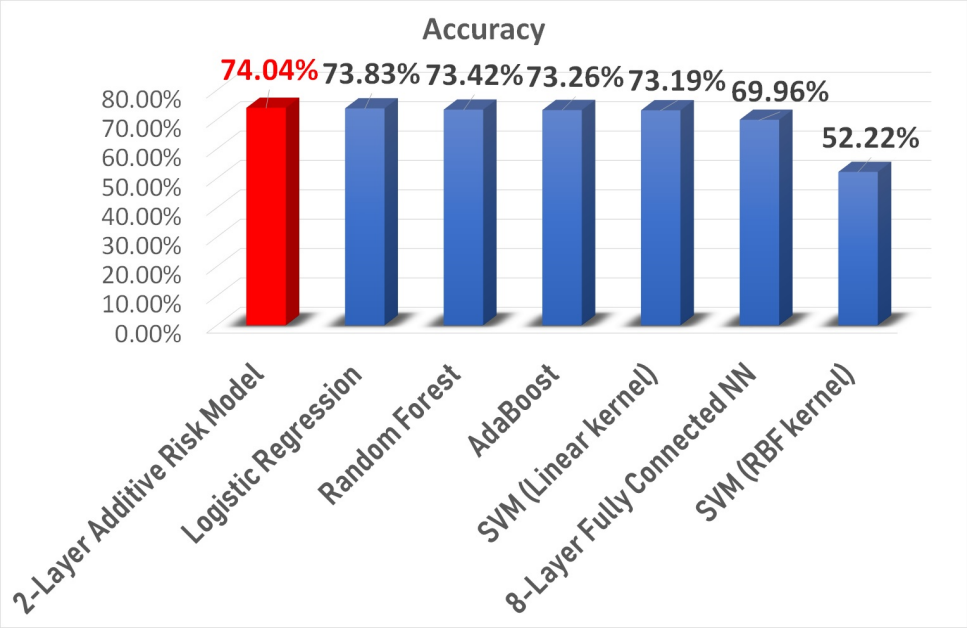


Figure 3.4: Accuracy compared to other common machine learning models.

Table 3.1: Most important risk factors for observation "Demo 1".

	Most important contributing factors
1	MaxDelq2PublicRecLast12M is 6 or less (from the most important subscale, Delinquency)
2	PercentTradesNeverDelq is 95 or less (from the most important subscale, Delinquency)
3	AverageMInFile is 48 or less (from the second most important subscale, TradeOpenTime)
4	AverageMInFile is 69 or less (from the second most important subscale, TradeOpenTime)

two subscales, we find two factors that contribute the most to that particular subscale's risk score. The two most important factors for each subscale are determined likewise by the product of the coefficient of each binary feature and the value of the binary feature itself. We finally output those binary features and their corresponding values as the most important contributing factors to the prediction made by our model.

The factors are grouped by subscales and are displayed in decreasing order of importance (within each important subscale) to the global model's predictions.

3.3 Discussion

In this chapter, we presented the two-layer additive risk model, and applied it to credit risk modeling. The novel elements of the work are (i) the form of the two-layer additive risk model, which lends naturally to sparsity, decomposability, monotonic relationship, and easy explainability using risk factors, (ii) the interactive visualization tool for the model, and (iii) the application to finance, indicating that black boxes may not be necessary in the case of credit-risk assessment.

Link: Our interactive display can be found here:

`http://dukedatasciencefico.cs.duke.edu`

Chapter 4

Case-based Interpretability: This Looks Like That

How would you describe why the image in Figure 4.1 looks like a clay colored sparrow? Perhaps the bird's head and wing bars look like those of a prototypical clay colored sparrow. When we describe how we classify images, we might compare images or image parts with prototypical cases from a given class. This method of reasoning is commonly used in difficult identification tasks: e.g., radiologists compare suspected tumors in X-ray scans with prototypical tumor images for diagnosis of cancer [HBSP05]. The question is whether we can ask a machine learning model to imitate this way of thinking, and to explain its reasoning process in a human-understandable way.

Existing predicate-based interpretable methods (e.g., decision trees, decision lists) are often insufficient for handling high-dimensional complex data such as images, particularly because the dimensions (the pixel values) themselves do not mean much to human beings. The complexity of image data calls for a new form of interpretability. The goal of this

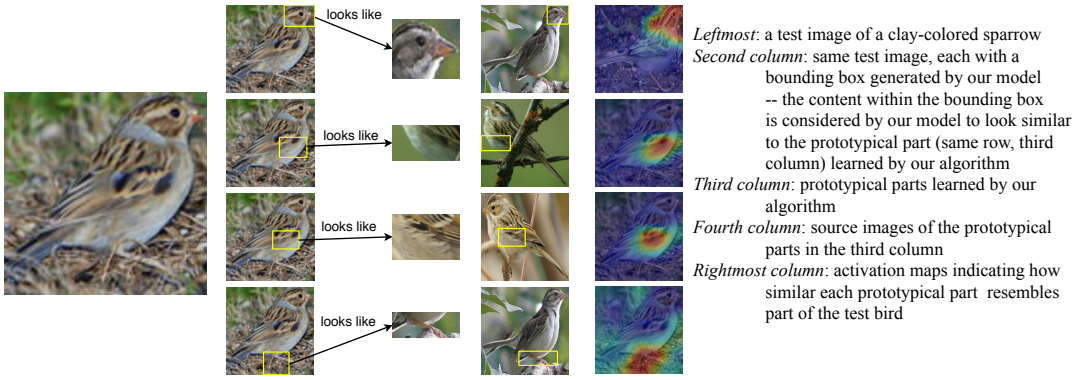


Figure 4.1: How parts of a clay colored sparrow look like some prototypical parts.

chapter is to define a form of *case-based interpretability – this looks like that* – for image recognition, that agrees with the way humans describe their own thinking in classification tasks.

In this chapter, we present two deep network architectures – *prototype network* (PrototypeNet) and *prototypical part network* (ProtoPNet), that accommodate this definition of interpretability, where comparison of entire images or image parts to learned prototypes is integral to the way our networks reason about new examples. PrototypeNet will be presented in Section 4.2, in the context of handwritten digit recognition. ProtoPNet will be presented in Section 4.3, in the context of bird species identification. The materials in Section 4.2 were jointly developed with Oscar Li, Hao Liu, and Cynthia Rudin, and were presented at the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018) and have been published in the *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence* [LLCR18]. The materials in other sections were jointly developed with Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin, and were presented

at the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019) and have been published in the *Advances in Neural Information Processing Systems 32* [CLT+19].

4.1 Related Work

Our work in case-based interpretable computer vision relates to (but contrasts with) those that perform *posthoc* interpretability analysis for a trained convolutional neural network (CNN). In posthoc analysis, one interprets a trained CNN by fitting explanations to how it performs classification. Examples of posthoc analysis techniques include activation maximization [EBCV09, Hin12, LGRN09, vdOKK16, NDY+16, SVZ14, YCFL15], deconvolution [ZF14], and saliency visualization [SVZ14, STY17, STK+17, SCD+17]. All of these posthoc visualization methods do not explain the reasoning process of how a network *actually* makes its decisions. In contrast, our PrototypeNet and ProtoPNet have built-in case-based reasoning processes, and the explanations generated by our networks are actually used during classification and are not created posthoc.

Our work relates closely to works that build attention-based interpretability into CNNs. These models aim to expose the parts of an input the network focuses on when making decisions. Examples of attention models include class activation maps [ZKL+16] and various part-based models (e.g., [ZFML17, ZDGD14, HXTZ16, ZSBT18, UVDSGS13, GDDM14, Gir15, RHGS15, SR15, XXY+15, FZM17]; see Table 4.3). All of these works build interpretability into neural networks by learning which parts of an input image are

important for classifying the image. Our PrototypeNet does not use attention to explain its predictions: it explains its predictions based on similarity to prototypical cases, rather than highlighting the most relevant parts of the input. Our ProtoPNet, on the other hand, uses both attention and similarity to prototypical cases to explain its predictions: it is not only able to expose the parts of the input it is looking at, but also point us to prototypical cases similar to those parts. Section 4.3.5 provides a comparison between attention-based models and our ProtoPNet.

Recently there have also been attempts to quantify the interpretability of visual representations in a CNN, by measuring the overlap between highly activated image regions and labeled visual concepts [BZK$^+$17, ZWZ18]. However, to quantitatively measure the interpretability of a convolutional unit in a network requires fine-grained labeling for a significantly large dataset specific to the purpose of the network. The existing Broden dataset for scene/object classification networks [BZK$^+$17] is not well-suited to measure the unit interpretability of a network trained for fine-grained classification (which is our main application), because the concepts detected by that network may not be present in the Broden dataset. Hence, in our work, we do not focus on quantifying unit interpretability of our network, but instead look at the *reasoning process* of our network which is qualitatively similar to that of humans.

Our work uses generalized convolution [GS19, NGSAT17] by including a prototype layer that computes squared L^2 distance instead of conventional inner product. In PrototypeNet,

a decoder is used to visualize prototypes. In ProtoPNet, we propose to constrain each prototype filter to be *identical* to some latent training patch. This added constraint allows us to interpret the prototype filters as visualizable prototypical image parts and also necessitates a novel training procedure.

Our work relates closely to other case-based classification techniques using k-nearest neighbors [WS09, SH07, PM18] or prototypes [PMDS03, BT11, WT17], and very closely, to the Bayesian Case Model [KRS14]. It relates to traditional "bag-of-visual-words" models used in image recognition [LSP06, FFP05, JNY07, SZ03, NS06]. These models (like our ProtoPNet) also learn a set of prototypical parts for comparison with an unseen image. However, the feature extraction in these models is performed by Scale Invariant Feature Transform (SIFT) [Low99], and the learning of prototypical patches ("visual words") is done separately from the feature extraction (and the learning of the final classifier). In contrast, our ProtoPNet uses a specialized neural network architecture for feature extraction and prototype learning, and can be trained in an *end-to-end* fashion. Our work also relates to works (e.g., [BVHBP14, LSLJ15]) that identify a set of prototypes for pose alignment. However, their prototypes are templates for warping images and similarity with these prototypes does not provide an explanation for why an image is classified in a certain way. Ming et al. [MXQR19] recently used the concepts in our work to develop prototype learning in recurrent neural networks for modeling sequential data.

4.2 Prototype Network (PrototypeNet): Written Digit Recognition

In this section, we introduce the architecture and the training objective of our PrototypeNet, and provide a detailed walk-through of how our network classifies an image of a handwritten digit and explains its prediction. We trained and evaluated our network on the MNIST dataset [LBBH98] of handwritten digits. The MNIST dataset is a benchmark dataset of gray-scale images of segmented and centered handwritten digits [LBBH98]. In our experiments, we used 55,000 training examples, 5,000 validation examples, and 10,000 testing examples, where every image is of size 28×28 pixels. We pre-process the images so that every pixel value is in $[0, 1]$.

4.2.1 PrototypeNet Architecture

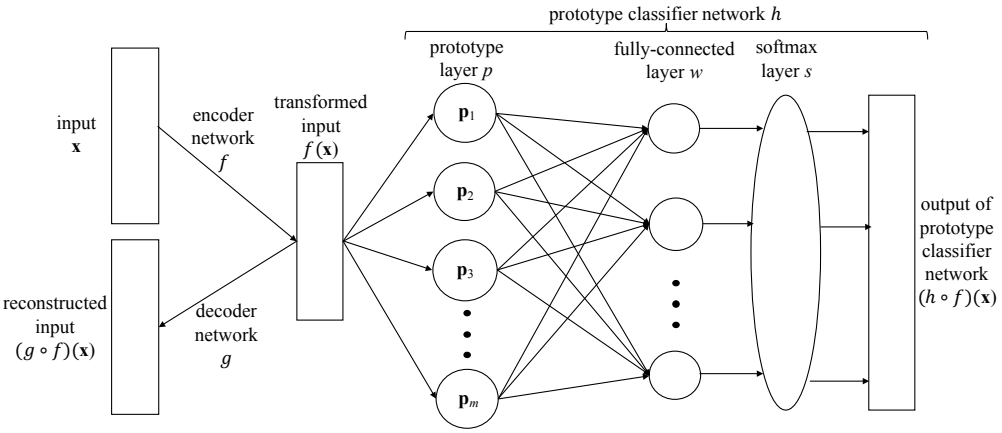


Figure 4.2: PrototypeNet architecture.

Let $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$ be the training dataset with $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{1, ..., K\}$ for each

$i \in \{1, ..., n\}$. Our model architecture consists of two components: an autoencoder (including an encoder, $f : \mathbb{R}^p \to \mathbb{R}^q$, and a decoder, $g : \mathbb{R}^q \to \mathbb{R}^p$) and a prototype classification network $h : \mathbb{R}^q \to \mathbb{R}^K$, illustrated in Figure 4.2. The network uses the autoencoder to reduce the dimensionality of the input and to learn useful features for prediction; then it uses the encoded input to produce a probability distribution over the K classes through the prototype classification network h. The network h is made up of three layers: a prototype layer, $p : \mathbb{R}^q \to \mathbb{R}^m$, a fully-connected layer $w : \mathbb{R}^m \to \mathbb{R}^K$, and a softmax layer, $s : \mathbb{R}^K \to \mathbb{R}^K$. The network learns m prototype vectors $\mathbf{p}_1, ..., \mathbf{p}_m \in \mathbb{R}^q$ (each corresponds to a *prototype unit* in the architecture) in the latent space. The prototype layer p computes the squared L^2 distance between the encoded input $\mathbf{z} = f(\mathbf{x}_i)$ and each of the prototype vectors:

$$p(\mathbf{z}) = \left[\|\mathbf{z} - \mathbf{p}_1\|_2^2, \quad \|\mathbf{z} - \mathbf{p}_2\|_2^2, \quad ... \quad \|\mathbf{z} - \mathbf{p}_m\|_2^2 \right]^\top. \tag{4.1}$$

In Figure 4.2, the *prototype unit* corresponding to $\mathbf{p}_j$ executes the computation $\|\mathbf{z} - \mathbf{p}_j\|_2^2$. The fully-connected layer w computes weighted sums of these distances $Wp(\mathbf{z})$, where W is a $K \times m$ weight matrix. These weighted sums are then normalized by the softmax layer s to output a probability distribution over the K classes. The k-th component of the output of the softmax layer s is defined by

$$s(\mathbf{v})_k = \frac{\exp(v_k)}{\sum_{k'=1}^{K} \exp(v_{k'})} \tag{4.2}$$

where v_k is the k-th component of the vector $\mathbf{v} = Wp(\mathbf{z}) \in \mathbb{R}^K$.

During prediction, the model outputs the class that it thinks is the most probable. In essence, our classification algorithm is distance-based on the low-dimensional learned feature

space. A special case is when we use one prototype for every class (let $m = K$) and set the weight matrix of the fully-connected layer to the negative identity matrix, $W = -I_{K\times K}$ (i.e. W is not learned during training). Then the data will be predicted to be in the same class as the nearest prototype in the latent space. More realistically, we typically do not know how many prototypes should be assigned to each class, and we may want a different number of prototypes from the number of classes, i.e., $m \neq K$. In this case, we allow W to be learned by the network, and, as a result, the distances to all the prototype vectors will contribute to the probability prediction for each class.

This network architecture has at least three advantages. First, unlike traditional case-based learning methods, the new method automatically learns useful features. For image datasets, which have dimensions equal to the number of pixels, if we perform classification using the original input space or use hand-crafted feature spaces, the methods tend to perform poorly (e.g., k-nearest neighbors). Second, because the prototype vectors live in the same space as the encoded inputs, we can feed these vectors into the decoder and visualize the learned prototypes throughout the training process. This property, coupled with the case-based reasoning nature of the prototype classification network h, gives users the ability to interpret how the network reaches its predictions and visualize the prototype learning process without *posthoc* analysis. Third, when we allow the weight matrix W to be learnable, we are able to tell from the strengths of the learned weight connections which prototypes are more representative of which class.

Architecture Details for Experiments on MNIST

Hinton and Salakhutdinov [HS06] showed that a multilayer fully connected autoencoder network can achieve good reconstruction on MNIST even when using a very low dimensional latent space. The autoencoder in our experiments on MNIST is a multilayer convolutional autoencoder with a symmetric architecture for the encoder and decoder to be our model's autoencoder; convolutional autoencoders are generally used to reduce dimensionality of image data and to learn useful hierarchical features for producing state-of-the-art classification results. Each convolutional layer consists of a convolution operation followed by a pointwise nonlinearity. We achieve down-sampling in the encoder through strided convolution, and use strided deconvolution in the corresponding layer of the decoder. After passing the original image through the encoder, the network flattens the output of the encoder, which is a series of feature maps, into a code vector and feeds it into the prototype layer. The resulting unflattened feature maps are fed into the decoder to reconstruct the original image. To visualize a prototype vector in the pixel space, we first reshape the vector to be in the same shape as the encoder output and then feed the reshaped vector (now a series of feature maps) into the decoder.

The autoencoder in our experiments has four convolutional layers in the encoder, and four deconvolutional layers in the decoder. All four convolutional layers in the encoder use kernels of size 3×3, same zero padding, and stride of size 2 in the convolution stage. The filters in the corresponding layers in the encoder and decoder are not constrained to

be transposes of each other. Each of the outputs of the first three layers has 32 feature maps, while the last layer has 10. Given an input image of dimension $28 \times 28 \times 1$, the shape of the encoder layers are thus: $14 \times 14 \times 32$; $7 \times 7 \times 32$; $4 \times 4 \times 32$; $2 \times 2 \times 10$, and therefore the network compresses every 784-dimensional image input to a 40-dimensional code vector ($2 \times 2 \times 10$). Every layer uses the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ as the nonlinear transformation. We specifically use the sigmoid function in the last encoder layer so that the output of the encoder is restricted to the unit hypercube $(0, 1)^{40}$. This allows us to initialize 15 prototype vectors uniformly at random in that hypercube. We do not use the rectified linear unit (ReLU – [KSH12]) in the last encoder layer because using it would make it more difficult to initialize the prototype vectors, as initial states throughout ${\mathbb{R}_{\geq 0}}^{40}$ would need to be explored, and the network would take longer to stabilize. We also specifically choose the sigmoid function for the last decoder layer to make the range of pixel values in the reconstructed output $(0, 1)$, roughly the same as the preprocessed image's pixel range.

4.2.2 Training Objective

The objective (i.e., cost function) for training a PrototypeNet reflects the needs for both accuracy and interpretability. In addition to the classification error, there is a (standard) term that penalizes the reconstruction error of the autoencoder. There are two new error terms that encourage the learned prototype vectors to correspond to meaningful points in the input space; in our case studies, these points are realistic images. All four terms are described mathematically below.

We use the standard cross-entropy loss for penalizing the misclassification. The cross-entropy loss on the training data D is denoted by E, and is given by

$$E(h \circ f, D) = \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} -\mathbb{1}[y_i = k] \log((h \circ f)_k(\mathbf{x}_i)) \tag{4.3}$$

where $(h \circ f)_k$ is the k-th component of $(h \circ f)$. We use the squared L^2 distance between the original and reconstructed input for penalizing the autoencoder's reconstruction error. The reconstruction loss, denoted by R, on the training data D is given by

$$R(g \circ f, D) = \frac{1}{n} \sum_{i=1}^{n} \|(g \circ f)(\mathbf{x}_i) - \mathbf{x}_i\|_2^2. \tag{4.4}$$

The two interpretability regularization terms are formulated as follows:

$$R_1(\mathbf{p}_1, ..., \mathbf{p}_m, D) = \frac{1}{m} \sum_{j=1}^{m} \min_{i \in [1,n]} \|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2, \tag{4.5}$$

$$R_2(\mathbf{p}_1, ..., \mathbf{p}_m, D) = \frac{1}{n} \sum_{i=1}^{n} \min_{j \in [1,m]} \|f(\mathbf{x}_i) - \mathbf{p}_j\|_2^2. \tag{4.6}$$

Here both terms are averages of minimum squared distances. The minimization of R_1 would require each prototype vector to be as close as possible to at least one of the training examples in the latent space. As long as we choose the decoder network to be a continuous function, we should expect two very close vectors in the latent space to be decoded to similar-looking images. Thus, R_1 will push the prototype vectors to have meaningful decodings in the pixel space. The minimization of R_2 would require every encoded training example to be as close as possible to one of the prototype vectors. This means that R_2 will cluster the training examples around prototypes in the latent space. We notice here that

although R_1 and R_2 involve a minimization function that is not differentiable everywhere, these terms are differentiable almost everywhere and many modern deep learning libraries support this type of differentiation. Ideally, R_1 would take the minimum distance over the entire training set for every prototype; therefore, the gradient computation would grow linearly with the size of the training set. However, this would be impractical during optimization for a large dataset. To address this problem, we relax the minimization to be over only the random minibatch used by the Stochastic Gradient Descent (SGD) algorithm. For the other three terms, since each of them is a summation over the entire training set, it is natural to apply SGD to randomly selected batches for gradient computation.

Putting everything together, the cost function, denoted by L, on the training data D with which we train our network (f, g, h), is given by

$$\begin{aligned} L((f,g,h),D) = E(h \circ f, D) + \lambda R(g \circ f, D) \\ + \lambda_1 R_1(\mathbf{p}_1, ..., \mathbf{p}_m, D) + \lambda_2 R_2(\mathbf{p}_1, ..., \mathbf{p}_m, D), \end{aligned} \tag{4.7}$$

where λ, λ_1, and λ_2 are real-valued hyperparameters that adjust the ratios between the terms.

Training Details for Experiments on MNIST

We set all the hyperparameters λ, λ_1, λ_2 to 0.05 and the learning rate to 0.0001. We minimize (4.7) as a whole: we do not employ a greedy layer-wise optimization for different layers of the autoencoder nor do we first train the autoencoder and then the prototype classification network.

Our goal in this work is not just to obtain reasonable accuracy, but also interpretability. We use only a few of the general techniques for improving performance in neural networks, and it is possible that using more techniques would improve accuracy. In particular, we use the data augmentation technique *elastic deformation* [SSP03] to improve prediction accuracy and reduce potential overfitting. The set of all elastic deformations is a superset of affine transformations. For every mini-batch of size 250 that we randomly sampled from the training set, we apply a random elastic distortion where a Gaussian filter of standard deviation equal to 4 and a scaling factor of 20 are used for the displacement field. Due to the randomness in the data augmentation process, the network sees a slightly different set of images during every epoch, which significantly reduces overfitting.

4.2.3 Experimental Results on Handwritten Digit Classification

In this section, we present the experimental results of training the PrototypeNet on the MNIST dataset. We first visualize the learned prototypes and the weight matrix W, then provide a detailed walk-through of how a specific image is classified, and finally compare the performance of our network model with other non-interpretible neural networks.

Visualization of Learned Prototypes and Weight Matrix

Let us first discuss the quality of the autoencoder, because good performance of the autoencoder will allow us to interpret the prototypes. After training, our network's

autoencoder achieved an average squared L^2 reconstruction error of 4.22 over the undeformed training set, where examples are shown in Figure 4.3. This reconstruction result assures us that the decoder can faithfully map the prototype vectors to the pixel space.

Figure 4.3: Reconstruction quality.

First row: Some random images from the training set. *Second row*: Reconstructions of the same images.

Figure 4.4: Prototype visualization: 15 learned prototypes visualized in pixel space.

We visualize the learned prototype vectors in Figure 4.4, by sending them through the decoder. The decoded prototype images are sharp-looking and mostly resemble real-life handwritten digits, owing to the interpretability terms R_1 and R_2 in the cost function. Note that there is not a one-to-one correspondence between classes and prototypes. Since we multiply the output of the prototype layer by a learnable weight matrix prior to feeding it into the softmax layer, the distances from an encoded image to each prototype have differing effects on the predicted class probabilities.

Table 4.1: Weight matrix between prototype layer and fully-connected layer.

Each row represents a prototype node whose decoded image is shown in the first column. Each column represents a digit class. The most negative weight is shaded for each prototype. In general, for each prototype, its most negative weight is towards its visual class except for the prototype in the last row.

	0	1	2	3	4	5	6	7	8	9
	-0.07	7.77	1.81	0.66	4.01	2.08	3.11	4.10	-20.45	-2.34
	2.84	3.29	1.16	1.80	-1.05	4.36	4.40	-0.71	0.97	-18.10
	-25.66	4.32	-0.23	6.16	1.60	0.94	1.82	1.56	3.98	-1.77
	-1.22	1.64	3.64	4.04	0.82	0.16	2.44	-22.36	4.04	1.78
	2.72	-0.27	-0.49	-12.00	2.25	-3.14	2.49	3.96	5.72	-1.62
	-5.52	1.42	2.36	1.48	0.16	0.43	-11.12	2.41	1.43	1.25
	4.77	2.02	2.21	-13.64	3.52	-1.32	3.01	0.18	-0.56	-1.49
	0.52	-24.16	2.15	2.63	-0.09	2.25	0.71	0.59	3.06	2.00
	0.56	-1.28	1.83	-0.53	-0.98	-0.97	-10.56	4.27	1.35	4.04
	-0.18	1.68	0.88	2.60	-0.11	-3.29	-11.20	2.76	0.52	0.75
	5.98	0.64	4.77	-1.43	3.13	-17.53	1.17	1.08	-2.27	0.78
	1.53	-5.63	-8.78	0.10	1.56	3.08	0.43	-0.36	1.69	3.49
	1.71	1.49	-13.31	-0.69	-0.38	4.55	1.72	1.59	3.18	2.19
	5.06	-0.03	0.96	4.35	-21.75	4.25	1.42	-1.27	1.64	0.78
	-1.31	-0.62	-2.69	0.96	2.36	2.83	2.76	-4.82	-4.14	4.95

We now look at the transposed weight matrix connecting the prototype layer to the softmax layer, shown in Table 4.1, to see the influence of the distance to each prototype on every class. We observe that each decoded prototype is visually similar to an image of a class for which the corresponding entry in the weight matrix has a significantly negative value. We will call the class to which a decoded prototype is visually similar the *visual class* of the prototype.

The reason for such a significantly negative value can be understood as follows. The prototype layer is computing the dissimilarity between an input image and a prototype through the squared L^2 distance between their representations in the latent space. Given

an image $\mathbf{x}_i$ and a prototype $\mathbf{p}_j$, if $\mathbf{x}_i$ does not belong to the visual class of $\mathbf{p}_j$, then the distance between $f(\mathbf{x}_i)$ and $\mathbf{p}_j$ will be large, so that when $\|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2$ is multiplied by the highly negative weight connection between the prototype $\mathbf{p}_j$ and its visual class, the product will also be highly negative and will therefore significantly reduce the activation of the visual class of $\mathbf{p}_j$. As a result, the image $\mathbf{x}_i$ will likely not be classified into the visual class of $\mathbf{p}_j$. Conversely, if $\mathbf{x}_i$ belongs to the visual class of $\mathbf{p}_j$, then when the small squared distance $\|\mathbf{p}_j - f(\mathbf{x}_i)\|_2^2$ is multiplied by the highly negative weight connection between $\mathbf{p}_j$ and its visual class, the product will not decrease the activation of $\mathbf{p}_j$'s visual class too much. In the end, the activations of every class that $\mathbf{x}_i$ does not belong to will be significantly reduced because of some non-similar prototype, leaving only the activation of $\mathbf{x}_i$'s actual class comparatively large. Therefore, $\mathbf{x}_i$ is correctly classified in general.

An interesting prototype learned by the network is the last prototype in Table 4.1. It is visually similar to an image of class 2; however, it has strong negative weight connections with class 7 and class 8 as well. Therefore, we can think of this prototype as being shared by these three classes, which means that an encoded input image that is far away from this prototype in latent space would be unlikely to be an image of 7, 8, or 2. This should not be too surprising: if we look at this decoded prototype image carefully, we can see that if we hide the tail of the digit, it would look like an image of 7; if we connect the upper-left endpoint with the lower-right endpoint, it would look like an image of 8.

Let us now look at the learned prototypes in Figure 4.4. The three prototypes for class

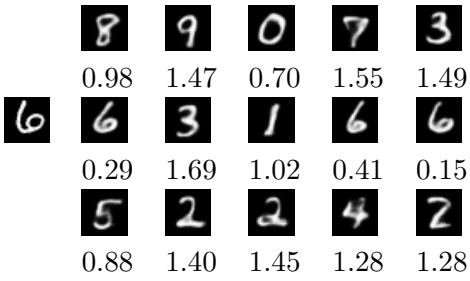
6 seem to represent different writing habits in terms of what the loop and angle of "6" looks like. The first and third 6's have their loops end at the bottom while the second 6's loop ends more on the side. The 2's show similar variation. As for the two 3's, the two prototypes correspond to different curvatures.

Reasoning Process of PrototypeNet

Let us look into the model as it produces a prediction for a specific image of digit 6, shown on the left of Table 4.2. The distances computed by the prototype layer between the encoded input image and each of the prototypes are shown below the decoded prototypes in Table 4.2, and the three smallest distances correspond to the three prototypes that resemble 6 after decoding. We observe here that these three distances are quite different, and the encoded input image is significantly closer to the third "6" prototype than the other two. This indicates that our model is indeed capturing the subtle differences within the same class.

After the prototype layer computes the 15-dimensional vector of distances shown in Table 4.2, it is multiplied by the weight matrix in Table 4.1, and the output is the unnormalized probability vector used as the logit for the softmax layer. The predicted probability of class 6 for this specific image is 99.99%.

Table 4.2: Distances between a test image 6 and every prototype in the latent space.



Comparison with Non-interpretible Neural Networks

After training for 1500 epochs, our model achieved a classification accuracy of 99.53% on the standard MNIST training set and 99.22% on the standard MNIST test set.

To examine how the two key elements of our interpretable network (the autoencoder and prototype layer) affect predictive power, we performed a type of ablation study. In particular, we trained two classification networks that are similar to ours, but removed some key pieces in both of the networks. The first network substitutes the prototype layer with a fully-connected layer whose output is a 15-dimensional vector, the same dimension as the output from the prototype layer; the second network also removes the decoder and changes the nonlinearity to ReLU. The second network is just a regular convolutional neural network that has similar architectural complexity to LeNet 5 [LBBH98]. After training both networks using elastic deformation for 1500 epochs, we obtained test accuracies of 99.24% and 99.23% respectively. These test accuracies, along with the test accuracy of 99.2% reported by [LBBH98], are comparable to the test accuracy of 99.22% obtained using our interpretable network. This result demonstrates that changing from a traditional

convolutional neural network to our interpretable network architecture does not hinder the predictive ability of the network (at least not in this case).

In general, it is not always true that accuracy needs to be sacrificed to obtain interpretability; there could be many models that are almost equally accurate. The extra terms in the cost function (and changes in architecture) encourage the model to be more interpretable among the set of approximately equally accurate models.

4.3 Prototypical Part Network (ProtoPNet): Bird Species Identification

It is certainly great to have the ability to classify a handwritten digit based on its similarity with how that digit is typically written. The question is: can we apply (or extend) our PrototypeNet, so that it can handle more complex images – for example, natural images with various backgrounds? It turns out that, when we train a PrototypeNet on datasets of natural images, the decoder of the PrototypeNet generally fails to produce realistic prototype images. Moreover, since the prototypes in a PrototypeNet represent entire images, we cannot use a PrototypeNet to perform fine-grained comparisons between parts of images. In addition, we have no control over the class identities of the prototypes in a PrototypeNet, which means that a trained PrototypeNet may not have any prototype for a particular class (this happens frequently when we have a large number of classes).

The limitations of our PrototypeNet prompt us to look for a new model architecture, one that can handle the classification of natural images while retaining the case-based

interpretability of our PrototypeNet. We call this new model architecture a *prototypical part network*, or ProtoPNet. Our ProtoPNet does not require a decoder for prototype visualization. Every prototype is the latent representation of some training image patch, which naturally and faithfully becomes the prototype's visualization. The removal of the decoder also facilitates the training of our network, leading to better explanations and better accuracy. Unlike a PrototypeNet, whose prototypes represent entire images, our ProtoPNet's prototypes can have much smaller spatial dimensions and represent *prototypical parts of images*. This allows for more fine-grained comparisons because different parts of an image can now be compared to different prototypes. In ProtoPNet, we also allocate a pre-determined number of prototypes for each class, so that every class will be represented by some prototypes in the final model.

In this section, we present the architecture and the training procedure of our ProtoPNet in the context of bird species identification, and provide a detailed walk-through of how our network classifies a new bird image and explains its prediction. We trained and evaluated our network on the CUB-200-2011 dataset [WBW$^+$11] of 200 bird species. Since the dataset has only about 30 images per class, we performed offline data augmentation using random rotation, skew, shear, distortion, and left-right flip to enlarge the training set, so that each class has approximately 1200 training images. We trained ProtoPNets on the training images that have been cropped using the bounding boxes provided with the dataset.

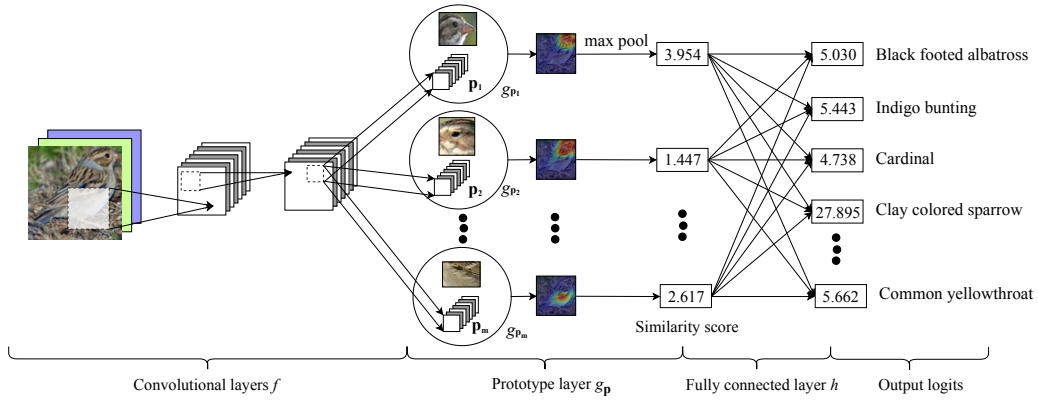


Figure 4.5: ProtoPNet architecture.

4.3.1 ProtoPNet Architecture

Figure 4.5 gives an overview of the architecture of our ProtoPNet. Our network consists of a regular convolutional neural network f, whose parameters are collectively denoted by w_{conv}, followed by a prototype layer $g_{\mathbf{p}}$ and a fully connected layer h with weight matrix w_h and no bias. For the regular convolutional network f, our model use the convolutional layers from models such as VGG-16, VGG-19 [SZ15], ResNet-34, ResNet-152 [HZRS16], DenseNet-121, or DenseNet-161 [HLvdMW17] (initialized with filters pretrained on ImageNet [DDS+09]), followed by two additional 1×1 convolutional layers (initialized using Kaiming uniform initialization [HZRS15]) in our experiments. We use ReLU as the activation function for all convolutional layers except the last for which we use the sigmoid activation function.

Given an input image $\mathbf{x}$ (such as the clay colored sparrow in Figure 4.5), the convolutional layers of our model extract useful features $f(\mathbf{x})$ to use for prediction. Let $H \times W \times D$ be the shape of the convolutional output $f(\mathbf{x})$. For the bird dataset with input images resized to $224 \times 224 \times 3$, the spatial dimension of the convolutional output is $H = W = 7$, and the

number of output channels D in the additional convolutional layers is chosen from three possible values: 128, 256, 512, using cross validation. The network learns m prototypes $\mathbf{P} = \{\mathbf{p}_j\}_{j=1}^{m}$, whose shape is $H_1 \times W_1 \times D$ with $H_1 \leq H$ and $W_1 \leq W$. In our experiments, we used $H_1 = W_1 = 1$. Since the depth of each prototype is the same as that of the convolutional output but the height and the width of each prototype is smaller than those of the whole convolutional output, each prototype will be used to represent some prototypical activation pattern in a *patch* of the convolutional output, which in turn will correspond to some prototypical image patch in the original pixel space. Hence, each prototype $\mathbf{p}_j$ can be understood as the latent representation of some prototypical *part* of some bird image in this case study. As a schematic illustration, the first prototype $\mathbf{p}_1$ in Figure 4.5 corresponds to the head of a clay colored sparrow, and the second prototype $\mathbf{p}_2$ the head of a Brewer's sparrow. Given a convolutional output $\mathbf{z} = f(\mathbf{x})$, the j-th prototype unit $g_{\mathbf{p}_j}$ in the prototype layer $g_{\mathbf{p}}$ computes the squared L^2 distances between the j-th prototype $\mathbf{p}_j$ and all patches of $\mathbf{z}$ that have the same shape as $\mathbf{p}_j$, and inverts the distances into similarity scores. The result is an activation map of similarity scores whose value indicates how strong a prototypical part is present in the image. This activation map preserves the spatial relation of the convolutional output, and can be upsampled to the size of the input image to produce a heat map that identifies which part of the input image is most similar to the learned prototype. The activation map of similarity scores produced by each prototype unit $g_{\mathbf{p}_j}$ is then reduced using global max pooling to a single similarity score, which can

be understood as how strongly a prototypical part is present in *some* patch of the input image. In Figure 4.5, the similarity score between the first prototype $\mathbf{p}_1$, a clay colored sparrow head prototype, and the most activated (upper-right) patch of the input image of a clay colored sparrow is 3.954, and the similarity score between the second prototype $\mathbf{p}_2$, a Brewer's sparrow head prototype, and the most activated patch of the input image is 1.447. This shows that our model finds that the head of a clay colored sparrow has a stronger presence than that of a Brewer's sparrow in the input image. Mathematically, the prototype unit $g_{\mathbf{p}_j}$ computes

$$g_{\mathbf{p}_j}(\mathbf{z}) = \max_{\tilde{\mathbf{z}} \in \text{patches}(\mathbf{z})} \log\left((\|\tilde{\mathbf{z}} - \mathbf{p}_j\|_2^2 + 1)/(\|\tilde{\mathbf{z}} - \mathbf{p}_j\|_2^2 + \epsilon)\right).$$

The function $g_{\mathbf{p}_j}$ is *monotonically decreasing* with respect to $\|\tilde{\mathbf{z}} - \mathbf{p}_j\|_2$ (if $\tilde{\mathbf{z}}$ is the closest latent patch to $\mathbf{p}_j$). Hence, if the output of the j-th prototype unit $g_{\mathbf{p}_j}$ is large, then there is a patch in the convolutional output that is (in 2-norm) very close to the j-th prototype in the latent space, and this in turn means that there is a patch in the input image that has a similar concept to what the j-th prototype represents.

In our ProtoPNet, we allocate a pre-determined number of prototypes m_k for each class $k \in \{1, ..., K\}$ (10 per class in our experiments), so that every class will be represented by some prototypes in the final model. Let $\mathbf{P}_k \subseteq \mathbf{P}$ be the subset of prototypes that are allocated to class k: these prototypes should capture the most relevant parts for identifying images of class k.

Finally, the m similarity scores produced by the prototype layer $g_{\mathbf{p}}$ are multiplied by

the weight matrix w_h in the fully connected layer h to produce the output logits, which are normalized using softmax to yield the predicted probabilities for a given image belonging to various classes.

Architecture Details for Experiments on CUB-200-2011

In our experiments on CUB-200-2011, we used the convolutional layers from VGG-16, VGG-19, ResNet-34, ResNet-152, DenseNet-121, and DenseNet-161 (initialized with filters pretrained on ImageNet), followed by two additional 1×1 convolutional layers, as the convolutional part of our ProtoPNet. The number of output channels in each of the two additional convolutional layers is chosen to be the same as the number of channels in a prototype. For each base architecture, we chose from three possible values: 128, 256, 512, for the number of channels in a prototype (using cross validation): for VGG-16, VGG-19, DenseNet-121, DenseNet-161, we used 128 as the number of channels in a prototype; for ResNet-34, we used 256 as the number of channels in a prototype; for ResNet-152, we used 512 as the number of channels in a prototype. We used 1×1 as the spatial dimension of each prototype (i.e., $H_1 = 1$ and $W_1 = 1$): given that the spatial dimension of the convolutional output for a 224×224 image is only 7×7, a 1×1 prototype is already large enough to represent a significant part of the original image in the pixel space (we want to learn prototypes **focused on specific parts**). The number of prototypes can be chosen with prior domain knowledge or hyperparameter search: we used 10 prototypes per class, because CUB-200-2011 provides (at most) 15 part locations per image, so we believe that

10 prototypes per class should be enough to capture a variety of bird parts. Note that the part locations (keypoint annotations) provided with the dataset were not used by our algorithm during training – we used only image-level labels.

4.3.2 Training Algorithm

The training of our ProtoPNet is divided into: (1) stochastic gradient descent (SGD) of layers before the last layer; (2) projection of prototypes; (3) convex optimization of last layer. It is possible to cycle through these three stages more than once. The entire training algorithm is summarized in the algorithm chart in Figure 4.6. In this chart, w_{base} and w_{add} denote the parameters of the base and additional convolutional layers; N_{SGD} and N_{convex} denote the number of training epochs in stage 1 and 3; L and L_{convex} denote the loss function of stage 1 and 3; $\eta_{\text{base}}^{(t)}$, $\eta_{\text{add}}^{(t)}$, $\eta_{\text{p}}^{(t)}$, η_{convex} are learning rates (t denotes epoch number). The choice of learning rates, as well as the coefficients of the terms in the loss function, is discussed at the end of this section (see "Training Details for Experiments on CUB-200-2011").

Stage 1: Stochastic Gradient Descent (SGD) of Layers before Last Layer

In the first training stage, we aim to learn a meaningful latent space, where the most important patches for classifying images are clustered (in L^2-distance) around semantically similar prototypes of the images' true classes, and the clusters that are centered at prototypes from different classes are well-separated. To achieve this goal, we jointly optimize the

initialize $t_0 \leftarrow 0$; $w_{\text{base}} \leftarrow$ weights pre-trained on ImageNet; $w_{\text{add}} \leftarrow$ Kaiming uniform initialization (He et al., 2015);
  $\forall j$: prototype $\mathbf{p}_j \leftarrow \text{Uniform}([0,1]^{H_1 \times W_1 \times D})$; $\forall k, j$: $w_h^{(k,j)} \leftarrow 1$ if $\mathbf{p}_j \in \mathbf{P}_k$, $w_h^{(k,j)} \leftarrow 0$ if $\mathbf{p}_j \notin \mathbf{P}_k$;
while NOT(converge AND Clst $< -$Sep) **do**
  `/* Stage 1: SGD of layers before the last */`
  for SGD training epoch $t = t_0 + 1, ..., t_0 + N_{\text{SGD}}$ **do**
    foreach batch $[\overline{\mathbf{X}}, \overline{\mathbf{Y}}]$ from $[\mathbf{X}, \mathbf{Y}]$ **do**
      if $t > 5$ **then** `/* Pretrained weights and biases are fixed during the warm-up period */`
        $w_{\text{base}} \leftarrow w_{\text{base}} - \eta_{\text{base}}^{(t)} \nabla_{w_{\text{base}}} L(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$;
      $w_{\text{add}} \leftarrow w_{\text{add}} - \eta_{\text{add}}^{(t)} \nabla_{w_{\text{add}}} L(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$; $\mathbf{P} \leftarrow \mathbf{P} - \eta_{\text{p}}^{(t)} \nabla_{\mathbf{P}} L(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$;
  $t_0 \leftarrow t_0 + N_{SGD}$;
  `/* Stage 2: projection of prototypes */`
  foreach prototype $\mathbf{p}_j$ **do**
    $k \leftarrow$ class of $\mathbf{p}_j$; $\mathbf{p}_j \leftarrow \arg\min_{\mathbf{z} \in \{\tilde{\mathbf{z}}: \tilde{\mathbf{z}} \in \text{patches}(f(\mathbf{x})) \ \forall (\mathbf{x}, y) \in [\mathbf{X}, \mathbf{Y}] \text{ s.t. } y = k\}} \|\mathbf{z} - \mathbf{p}_j\|_2$;
  `/* Stage 3: convex optimization of last layer */`
  for convex training epoch $t' = 1, ..., N_{\text{convex}}$ **do**
    foreach batch $[\overline{\mathbf{X}}, \overline{\mathbf{Y}}]$ from $[\mathbf{X}, \mathbf{Y}]$ **do** $w_h \leftarrow w_h - \eta_{\text{convex}} \nabla_{w_h} L_{\text{convex}}(\overline{\mathbf{X}}, \overline{\mathbf{Y}})$;

Figure 4.6: Overview of training algorithm.

convolutional layers' parameters w_{conv} and the prototypes $\mathbf{P} = \{\mathbf{p}_j\}_{j=1}^m$ in the prototype layer $g_{\mathbf{p}}$ using SGD, while keeping the last layer weight matrix w_h fixed. Let $[\mathbf{X}, \mathbf{Y}] = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be the set of training images. The optimization problem we aim to solve here is:

$$\min_{\mathbf{P}, w_{\text{conv}}} \frac{1}{n} \sum_{i=1}^{n} \text{CrsEnt}(h \circ g_{\mathbf{p}} \circ f(\mathbf{x_i}), \mathbf{y_i}) + \lambda_1 \text{Clst} + \lambda_2 \text{Sep},$$

where Clst and Sep are defined by

$$\text{Clst} = \frac{1}{n} \sum_{i=1}^{n} \min_{j: \mathbf{p}_j \in \mathbf{P}_{y_i}} \min_{\mathbf{z} \in \text{patches}(f(\mathbf{x}_i))} \|\mathbf{z} - \mathbf{p}_j\|_2^2, \quad \text{and}$$

$$\text{Sep} = -\frac{1}{n} \sum_{i=1}^{n} \min_{j: \mathbf{p}_j \notin \mathbf{P}_{y_i}} \min_{\mathbf{z} \in \text{patches}(f(\mathbf{x}_i))} \|\mathbf{z} - \mathbf{p}_j\|_2^2.$$

The cross entropy loss (CrsEnt) penalizes misclassification on the training data. The minimization of the cluster cost (Clst) encourages each training image to have some latent patch that is close to at least one prototype of its own class, while the minimization of the separation cost (Sep) encourages every latent patch of a training image to stay away from the prototypes *not* of its own class. These terms shape the latent space into a semantically

meaningful clustering structure, which facilitates the L^2-distance-based classification of our network.

In this training stage, we also fix the last layer h, whose weight matrix is w_h. Let $w_h^{(k,j)}$ be the (k,j)-th entry in w_h that corresponds to the weight connection between the output of the j-th prototype unit $g_{\mathbf{p}_j}$ and the logit of class k. Given a class k, we set $w_h^{(k,j)} = 1$ for all j with $\mathbf{p}_j \in \mathbf{P}_k$ and $w_h^{(k,j)} = -0.5$ for all j with $\mathbf{p}_j \notin \mathbf{P}_k$ (when we are in this stage for the first time). Intuitively, the positive connection between a class k prototype and the class k logit means that similarity to a class k prototype should increase the predicted probability that the image belongs to class k, and the negative connection between a non-class k prototype and the class k logit means that similarity to a non-class k prototype should decrease class k's predicted probability. By fixing the last layer h in this way, we can force the network to learn a meaningful latent space because if a latent patch of a class k image is too close to a non-class k prototype, it will decrease the predicted probability that the image belongs to class k and increase the cross entropy loss in the training objective. Note that both the separation cost and the negative connection between a non-class k prototype and the class k logit encourage prototypes of class k to represent semantic concepts that are characteristic of class k but not of other classes: if a class k prototype represents a semantic concept that is also present in a non-class k image, this non-class k image will highly activate that class k prototype, and this will be penalized by increased (i.e., less negative) separation cost and increased cross entropy (as a result of the

negative connection). The separation cost is new to this paper, and has not been explored by previous works of prototype learning (e.g., [BVHBP14, LLCR18]).

Stage 2: Projection of Prototypes

To be able to visualize the prototypes as training image patches, we project ("push") each prototype $\mathbf{p}_j$ onto the nearest latent training patch from the *same* class as that of $\mathbf{p}_j$. In this way, we can conceptually equate each prototype with a training image patch. (Section 4.3.3 discusses how we visualize the projected prototypes.) Mathematically, for prototype $\mathbf{p}_j$ of class k, i.e., $\mathbf{p}_j \in \mathbf{P}_k$, we perform the following update:

$$\mathbf{p}_j \leftarrow \arg\min_{\mathbf{z} \in \mathcal{Z}_j} \|\mathbf{z} - \mathbf{p}_j\|_2, \text{where } \mathcal{Z}_j = \{\tilde{\mathbf{z}} : \tilde{\mathbf{z}} \in \text{patches}(f(\mathbf{x}_i)) \; \forall i \text{ s.t. } y_i = k\}.$$

The following theorem provides some theoretical understanding of how prototype projection affects classification accuracy. We use another notation for prototypes $\mathbf{p}_l^k$, where k represents the class identity of the prototype and l is the index of that prototype among all prototypes of that class.

Theorem 4.3.1. *Let $h \circ g_{\mathbf{p}} \circ f$ be a ProtoPNet. For each k, l, we use $\mathbf{b}_l^k$ to denote the value of the l-th prototype for class k **before** the projection of $\mathbf{p}_l^k$ to the nearest latent training patch of class k, and use $\mathbf{a}_l^k$ to denote its value **after** the projection. Let $\mathbf{x}$ be an input image that is correctly classified by the ProtoPNet before the projection, $\mathbf{z}_l^k = \arg\min_{\tilde{\mathbf{z}} \in patches(f(\mathbf{x}))} \|\tilde{\mathbf{z}} - \mathbf{b}_l^k\|_2$ be the nearest patch of $f(\mathbf{x})$ to the prototype $\mathbf{p}_l^k$ before the projection (i.e., $\mathbf{b}_l^k$), and c be the correct class label of $\mathbf{x}$.*

Suppose *that:*

(A1) $\mathbf{z}_l^k$ *is also the nearest latent patch to prototype* $\mathbf{p}_l^k$ *after the projection* ($\mathbf{a}_l^k$)*, i.e.,* $\mathbf{z}_l^k = \arg\min_{\tilde{\mathbf{z}} \in patches(f(\mathbf{x}))} \|\tilde{\mathbf{z}} - \mathbf{a}_l^k\|_2$*;*

(A2) *there exists some* δ *with* $0 < \delta < 1$ *such that:*

(A2a) *for all incorrect classes' prototypes* $k \neq c$ *and* $l \in \{1, ..., m_k\}$*, we have* $\|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2 \leq \theta \|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \sqrt{\epsilon}$*, where we define* $\theta = \min\left(\sqrt{1+\delta} - 1, 1 - \frac{1}{\sqrt{2-\delta}}\right)$ *(*ϵ *comes from the prototype activation function* $g_{\mathbf{p}_j}$ *defined in Section 2.1);*

(A2b) *for the correct class* c *and for all* $l \in \{1, ..., m_c\}$*, we have* $\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2 \leq (\sqrt{1+\delta} - 1)\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2$ *and* $\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 \leq \sqrt{1-\delta}$*;*

(A3) *the number of prototypes is the same for each class, which we denote by* m'*.*

(A4) *for each class* k*, the weight connection in the fully connected last layer* h *between a class* k *prototype and the class* k *logit is* 1*, and that between a non-class* k *prototype and the class* k *logit is* 0 *(i.e.,* $w_h^{(k,j)} = 1$ *for all* j *with* $\mathbf{p}_j \in \mathbf{P}_k$ *and* $w_h^{(k,j)} = 0$ *for all* j *with* $\mathbf{p}_j \notin \mathbf{P}_k$*).*

Then *after projection, the output logit for the correct class* c *can decrease at most by* $\Delta_{\max} = m' \log((1+\delta)(2-\delta))$*, and the output logit for every incorrect class* $k \neq c$ *can increase at most by* $\Delta_{\max}$*. If the output logits between the top-*2 *classes are at least* $2\Delta_{\max}$ *apart, then the projection of prototypes to their nearest latent training patches does not change the prediction of* $\mathbf{x}$*.*

Intuitively speaking, the theorem states that, if prototype projection does not move the

prototypes by much (assured by the optimization of the cluster cost Clst), the prediction does not change for examples that the model predicted correctly with some confidence before the projection. The proof is in Section 4.3.8.

Note that prototype projection has the same time complexity as feedforward computation of a regular convolutional layer followed by global average pooling, a configuration common in standard CNNs (e.g., ResNet, DenseNet), because the former takes the minimum distance over all prototype-sized patches, and the latter takes the average of dot-products over all filter-sized patches. Hence, prototype projection does not introduce extra time complexity in training our network.

Stage 3: Convex Optimization of Last Layer

In this training stage, we perform a convex optimization on the weight matrix w_h of last layer h. The goal of this stage is to adjust the last layer connection $w_h^{(k,j)}$, so that for k and j with $\mathbf{p}_j \notin \mathbf{P}_k$, our final model has the sparsity property $w_h^{(k,j)} \approx 0$ (initially fixed at -0.5). This sparsity is desirable because it means that our model relies less on a *negative* reasoning process of the form "this bird is of class k' because it is *not* of class k (it contains a patch that is *not* prototypical of class k)." The optimization problem we solve here is:

$$\min_{w_h} \frac{1}{n} \sum_{i=1}^{n} \text{CrsEnt}(h \circ g_{\mathbf{p}} \circ f(\mathbf{x_i}), \mathbf{y_i}) + \lambda \sum_{k=1}^{K} \sum_{j:\mathbf{p}_j \notin \mathbf{P}_k} |w_h^{(k,j)}|.$$

This optimization is convex because we fix all the parameters from the convolutional and prototype layers. This stage further improves accuracy without changing the learned latent

space or prototypes.

Training Details for Experiments on CUB-200-2011

In our experiments on CUB-200-2011, we set the coefficient of the cluster cost to 0.8, and the coefficient of the separation cost to 0.08 during stochastic gradient descent of layers before the last layer, and we set the coefficient of the L^1-regularization term (on the weight connection between each prototype of class k and the logit of class $k' \neq k$) to 10^{-4} during convex optimization of the last layer. For the coefficient of the cluster cost and the coefficient of the separation cost, we considered three different settings: $(1, 0.1)$, $(0.8, 0.08)$, $(0.6, 0.06)$, and chose the pair $(0.8, 0.08)$ using cross validation. For the coefficient of the L^1-regularization term, we considered 10^{-3}, 10^{-4}, and 10^{-5}, and chose 10^{-4} also by cross validation.

In our experiments, we initialized the base convolutional layers with filters pretrained on ImageNet [DDS+09], and initialized the two additional convolutional layers using Kaiming uniform initialization [HZRS15]. The prototypes were initialized randomly from uniform distribution over $[0, 1]^{H_1 \times W_1 \times D}$ – the last convolutional layer uses sigmoid activation, so the convolutional features all lie in $[0, 1]$. We started our training with a "warm-up" stage, in which we loaded and froze the pre-trained weights and biases, and focused on training the two additional convolutional layers and the prototype layer (without requiring each prototype to be exactly some latent training patch) for 5 epochs. The learning rate we used in this sub-stage is 3×10^{-3}. Afterward, we trained all the convolutional layers and the

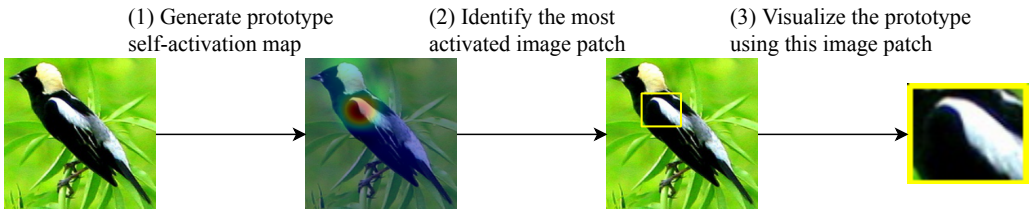


Figure 4.7: How to visualize a prototype.

prototype layer jointly, using 10^{-4} learning rate for those layers that were pretrained on ImageNet, and 3×10^{-3} learning rate for the two additional convolutional layers and the prototype layer. We reduced the learning rate by a factor of 0.1 every 5 epochs, and we performed prototype projection and convex optimization of the last layer (for 20 iterations) every two times we reduced the learning rate. We stopped training when training accuracy converged and the cluster cost became smaller than the separation cost on the training set.

We implemented our ProtoPNet using PyTorch. The experiments were run on 4 NVIDIA Tesla P100 GPUs or 8 NVIDIA Tesla K80 GPUs.

4.3.3 Prototype Visualization

In this section, we provide a detailed explanation of how we visualize prototypes of a trained ProtoPNet model. Recall that after the projection of prototypes to the closest latent patch of some training image, a prototype $\mathbf{p}_j$ is exactly equal to some patch of the latent representation $f(\mathbf{x})$, of some training image $\mathbf{x}$. Since the patch of $\mathbf{x}$ that corresponds to the prototype $\mathbf{p}_j$ should be the one that the prototype $\mathbf{p}_j$ activates the most strongly on, we visualize the prototype $\mathbf{p}_j$ by first obtaining the activation map of $\mathbf{x}$ by the prototype $\mathbf{p}_j$:

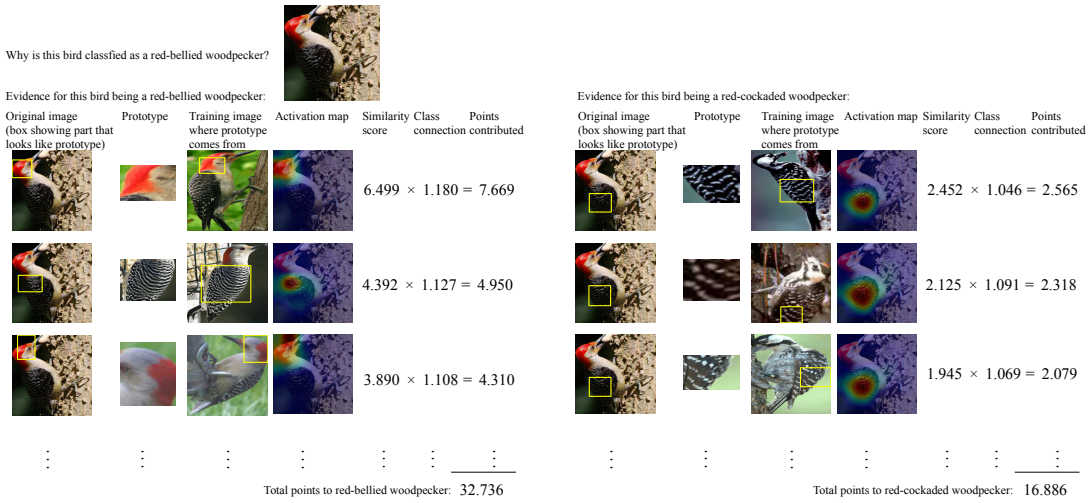


Figure 4.8: Reasoning process of ProtoPNet in deciding the species of a bird (top).

this can be done by forwarding $\mathbf{x}$ through the trained ProtoPNet model and upsampling the activation map produced by the prototype unit $g_{\mathbf{p}_j}$ to the size of the image $\mathbf{x}$ (Step (1) in Figure 4.7). After we obtain such an activation map, we can locate the patch of $\mathbf{x}$ on which $\mathbf{p}_j$ has the strongest activation by finding the high activation region in the (upsampled) activation map (Step (2) in Figure 4.7). In our experiments, we define the high activation region in an upsampled activation map as the smallest rectangular region that encloses pixels whose corresponding activation value in the aforementioned activation map is at least 95%-percentile of all activation values in that same map. Finally, we can visualize the prototype $\mathbf{p}_j$ using the image patch of $\mathbf{x}$ that corresponds to the high activation region (Step (3) in Figure 4.7).

4.3.4 Reasoning Process of ProtoPNet

Figure 4.8 shows the reasoning process of our ProtoPNet in reaching a classification decision on a test image of a red-bellied woodpecker at the top of the figure. Given this test image $\mathbf{x}$, our model compares its latent features $f(\mathbf{x})$ against the learned prototypes. In particular, for each class k, our network tries to find evidence for x to be of class k by comparing its latent patch representations with every learned prototype $\mathbf{p}_j$ of class k. For example, in Figure 4.8 (left), our network tries to find evidence for the red-bellied woodpecker class by comparing the image's latent patches with each prototype (visualized in "Prototype" column) of that class. This comparison produces a map of similarity scores towards each prototype, which was upsampled and superimposed on the original image to see which part of the given image is activated by each prototype. As shown in the "Activation map" column in Figure 4.8 (left), the first prototype of the red-bellied woodpecker class activates most strongly on the head of the testing bird, and the second prototype on the wing: the most activated image patch of the given image for each prototype is marked by a bounding box in the "Original image" column – this is the image patch that the network considers to look like the corresponding prototype. In this case, our network finds a high similarity between the head of the given bird and the *prototypical* head of a red-bellied woodpecker (with a similarity score of 6.499), as well as between the wing and the *prototypical* wing (with a similarity score of 4.392). These similarity scores are weighted and summed together to give a final score for the bird belonging to this class. The reasoning process is similar for

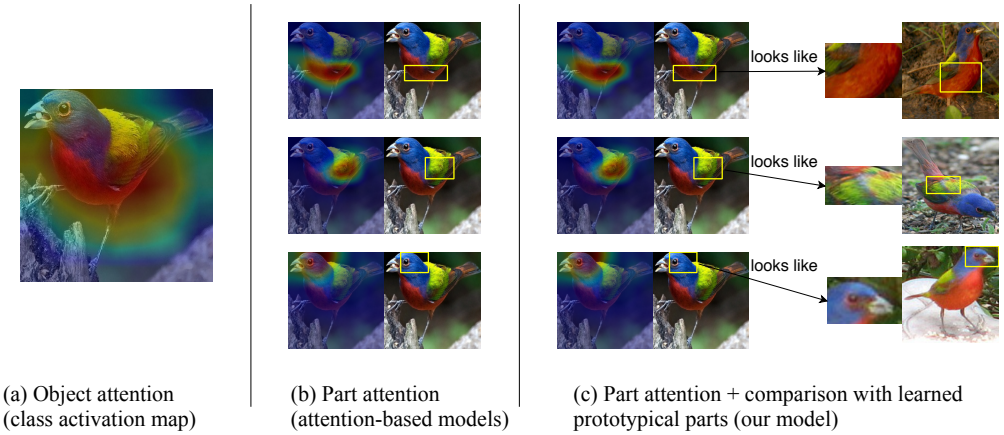


Figure 4.9: Visual comparison of different types of model interpretability.

(a) Object-level attention map (e.g., class activation map [ZKL+16]); (b) part attention (provided by attention-based interpretable models); and (c) part attention with similar prototypical parts (provided by our model).

all other classes (Figure 4.8 (right)). The network finally correctly classifies the bird as a red-bellied woodpecker.

4.3.5 Comparison with Baseline and Attention Models

The accuracy of our ProtoPNet (with various base CNN architectures) on cropped bird images is compared to that of the corresponding baseline model in the top of Table 4.3: the first number in each cell gives the mean accuracy, and the second number gives the standard deviation, over three runs. To ensure fairness of comparison, the baseline models (without the prototype layer) were trained on the same augmented dataset of cropped bird images as the corresponding ProtoPNet. As we can see, the test accuracy of our ProtoPNet is comparable with that of the corresponding baseline (non-interpretable) model: the loss of accuracy is at most 3.5% when we switch from the non-interpretable baseline model to

Table 4.3: Accuracy comparison of ProtoPNet with other models.

Top: Accuracy comparison of ProtoPNet with baseline models on cropped bird images of CUB-200-2011.
Bottom: Accuracy comparison of ProtoPNet with other deep models.

Base	ProtoPNet	Baseline	Base	ProtoPNet	Baseline
VGG16	76.1 ± 0.2	74.6 ± 0.2	VGG19	78.0 ± 0.2	75.1 ± 0.4
Res34	79.2 ± 0.1	82.3 ± 0.3	Res152	78.0 ± 0.3	81.5 ± 0.4
Dense121	80.2 ± 0.2	80.5 ± 0.1	Dense161	80.1 ± 0.3	82.2 ± 0.2

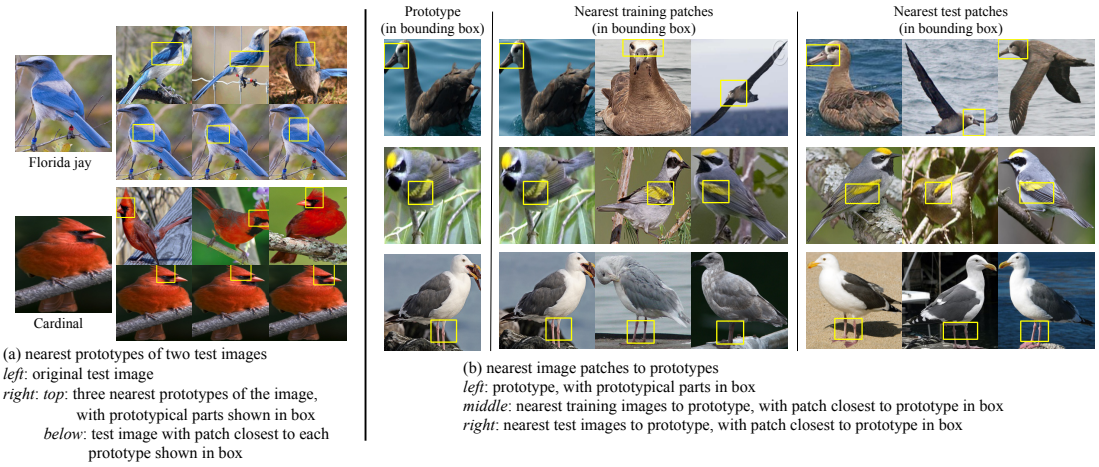
Interpretability	Model: accuracy
None	**B-CNN**[LRM15]: 85.1 (bb), 84.1 (full)
Object-level attn.	**CAM**[ZKL+16]: 70.5 (bb), 63.0 (full)
Part-level attention	**Part R-CNN**[ZDGD14]: 76.4 (bb+anno.); **PS-CNN** [HXTZ16]: 76.2 (bb+anno.); **PN-CNN** [BVHBP14]: 85.4 (bb+anno.); **DeepLAC**[LSLJ15]: 80.3 (anno.); **SPDA-CNN**[ZXE+16]: 85.1 (bb+anno.); **PA-CNN**[KJYFF15]: 82.8 (bb); **MG-CNN**[WSS+15]: 83.0 (bb), 81.7 (full); **ST-CNN**[JSZK15]: 84.1 (full); **2-level attn.**[XXY+15]: 77.9 (full); **FCAN**[LXW+16]: 82.0 (full); **Neural const.**[SR15]: 81.0 (full); **MA-CNN**[ZFML17]: 86.5 (full); **RA-CNN**[FZM17]: 85.3 (full)
Part-level attn. + prototypical cases	**ProtoPNet** (ours): 80.8 (full, VGG19+Dense121+Dense161-based) 84.8 (bb, VGG19+ResNet34+DenseNet121-based)

our interpretable ProtoPNet. We can further improve the accuracy of ProtoPNet by *adding the logits of several ProtoPNet models together*. Since each ProtoPNet can be understood as a "scoring sheet" (as in Figure 4.8) for each class, adding the logits of several ProtoPNet models is equivalent to creating a combined scoring sheet where (weighted) similarity with prototypes from all these models is taken into account to compute the total points for each class – the combined model will have the same interpretable form when we combine several ProtoPNet models in this way, though there will be more prototypes for each class. The test

accuracy on cropped bird images of combined ProtoPNets can reach 84.8%, which is on par with some of the best-performing deep models that were also trained on cropped images (see bottom of Table 4.3). We also trained a VGG19-, DenseNet121-, and DenseNet161-based ProtoPNet on full images: the test accuracy of the combined network can go above 80% – at 80.8%, even though the test accuracy of each individual network is 72.7%, 74.4%, and 75.7%, respectively. Section 4.3.9 includes an example that illustrates how combining several ProtoPNet models can improve accuracy while preserving interpretability.

Moreover, our ProtoPNet provides a level of interpretability that is absent in other interpretable deep models. In terms of the type of explanations offered, Figure 4.9 provides a visual comparison of different types of model interpretability. At the coarsest level, there are models that offer object-level attention (e.g., class activation maps [ZKL$^+$16]) as explanation: this type of explanation (usually) highlights the entire object as the "reason" behind a classification decision, as shown in Figure 4.9(a). At a finer level, there are numerous models that offer part-level attention: this type of explanation highlights the important parts that lead to a classification decision, as shown in Figure 4.9(b). Almost all attention-based interpretable deep models offer this type of explanation (see the bottom of Table 4.3). In contrast, our model not only offers part-level attention, but also provides similar prototypical cases, and uses similarity to prototypical cases of a particular class as justification for classification (see Figure 4.9(c)). This type of interpretability is absent in other interpretable deep models. In terms of how attention is generated, some

attention models generate attention with auxiliary part-localization models trained with part annotations (e.g., [ZDGD14, ZXE+16, BVHBP14, LSLJ15, HXTZ16]); other attention models generate attention with "black-box" methods – e.g., RA-CNN [FZM17] uses another neural network (attention proposal network) to decide where to look next; multi-attention CNN [ZFML17] uses aggregated convolutional feature maps as "part attentions." There is no explanation for why the attention proposal network decides to look at some region over others, or why certain parts are highlighted in those convolutional feature maps. In contrast, our ProtoPNet generates attention based on similarity with learned prototypes: it requires no part annotations for training, and explains its attention naturally – it is looking at *this* region of input because *this* region is similar to *that* prototypical example. Although other attention models focus on similar regions (e.g., head, wing, etc.) as our ProtoPNet, they cannot be made into a case-based reasoning model like ours: the only way to find prototypes on other attention models is to analyze *posthoc* what activates a convolutional filter of the model most strongly and think of that as a prototype – however, since such prototypes do not participate in the actual model computation, any explanations produced this way are not always faithful to the classification decisions. The bottom of Table 4.3 compares the accuracy of our model with that of some state-of-the-art models on this dataset: "full" means that the model was trained and tested on full images, "bb" means that the model was trained and tested on images cropped using bounding boxes (or the model used bounding boxes in other ways), and "anno." means that the model was



(a) nearest prototypes of two test images
left: original test image
right: *top*: three nearest prototypes of the image, with prototypical parts shown in box
below: test image with patch closest to each prototype shown in box

(b) nearest image patches to prototypes
left: prototype, with prototypical parts in box
middle: nearest training images to prototype, with patch closest to prototype in box
right: nearest test images to prototype, with patch closest to prototype in box

Figure 4.10: Nearest prototypes to images and nearest images to prototype.

trained with keypoint annotations of bird parts. Even though there is some accuracy gap between our (combined) ProtoPNet model and the best of the state-of-the-art, this gap may be reduced through more extensive training effort, and the added interpretability in our model already makes it possible to bring richer explanations and better transparency to deep neural networks.

4.3.6 Analysis of Latent Space

In this section, we analyze the structure of the latent space learned by our ProtoPNet. Figure 4.10(a) shows the three nearest prototypes to a test image of a Florida jay and of a cardinal. As we can see, the nearest prototypes for each of the two test images come from the same class as that of the image, and the test image's patch most activated by each prototype also corresponds to the same semantic concept as the prototype: in the case of the Florida jay, the most activated patch by each of the three nearest prototypes (all

wing prototypes) indeed localizes the wing; in the case of the cardinal, the most activated patch by each of the three nearest prototypes (all head prototypes) indeed localizes the head. Figure 4.10(b) shows the nearest (i.e., most activated) image patches in the entire training/test set to three prototypes. As we can see, the nearest image patches to the first prototype in the figure are all heads of black-footed albatrosses, and the nearest image patches to the second prototype are all yellow stripes on the wings of golden-winged warblers. The nearest patches to the third prototype are feet of some gull. It is generally true that the nearest patches of a prototype all bear the same semantic concept, and they mostly come from those images in the same class as the prototype. Section 4.3.10 gives more examples of the nearest prototypes of given test images, and Section 4.3.11 gives more examples of the nearest image patches of given prototypes. Those prototypes whose nearest training patches have mixed class identities usually correspond to background patches, and they can be automatically pruned from our model. Section 4.3.7 discusses prototype pruning in greater detail.

4.3.7 Prototype Pruning

Recall from our analysis of the nearest image patches of given prototypes, that it is generally true that the nearest (i.e., most activated) patches of a prototype mostly come from those images in the same class as that of the prototype. However, there are exceptions to this general observation. In our experiments, we find that for some prototypes in a trained ProtoPNet model, the nearest (training or test) image patches can all come from different

classes. This is often the case when the prototype corresponds to a "background" patch (e.g., a patch of the sky). This happens during the training process because a background prototype can often be useful in distinguishing different species of birds: for example, a prototype that corresponds to a patch of water can be useful in distinguishing water birds from others.

In this section, we describe an algorithm for prototype pruning, that can reduce the number of prototypes for each class, and at the same time, remove "background" prototypes automatically from the reasoning process of our ProtoPNet. The pruning algorithm starts by first finding the k-nearest latent patches of training images to each prototype $\mathbf{p}_j$. Since we know the labels of all training images, we know the class labels of the k training images where the k-nearest latent patches to $\mathbf{p}_j$ come from. Out of these k training images, if less than τ (a predefined pruning threshold) of them come from the designated class of the prototype $\mathbf{p}_j$, then we assume that the prototype $\mathbf{p}_j$ most likely corresponds to some background patch and we remove the prototype $\mathbf{p}_j$ (along with its last layer connections) from the ProtoPNet model. After pruning, we can again optimize the last layer (see Section 2.2: Training algorithm) to boost accuracy further.

In our experiments, we set $k = 6$ and $\tau = 3$. Table 4.4 shows the effect of pruning (and subsequent optimization of the last layer) on ProtoPNet models trained on cropped bird images of CUB-200-2011, as well as the number of prototypes pruned (recall that we used 10 prototypes per class, so there were a total of 2000 prototypes for 200 classes before

Table 4.4: Effect of pruning on ProtoPNet models.

The models were trained on cropped bird images of CUB-200-2011. The number of prototypes before pruning was 2000 for each model.

Base architecture of ProtoPNet	Accuracy before pruning	Accuracy after pruning	Accuracy after pruning and optimizing last layer	Number of prototypes pruned ($k = 6$, $\tau = 3$)
VGG16	76.3	71.8	76.0	651
VGG19	78.2	74.4	78.0	666
ResNet34	79.2	79.1	79.5	345
ResNet152	78.3	78.1	78.6	266
DenseNet121	80.4	77.0	79.2	524
DenseNet161	80.1	78.3	79.9	473

pruning). As we can see from Table 4.4, pruning has little effect on the accuracy of our ProtoPNet, if it is followed by the optimization of the last fully connected layer.

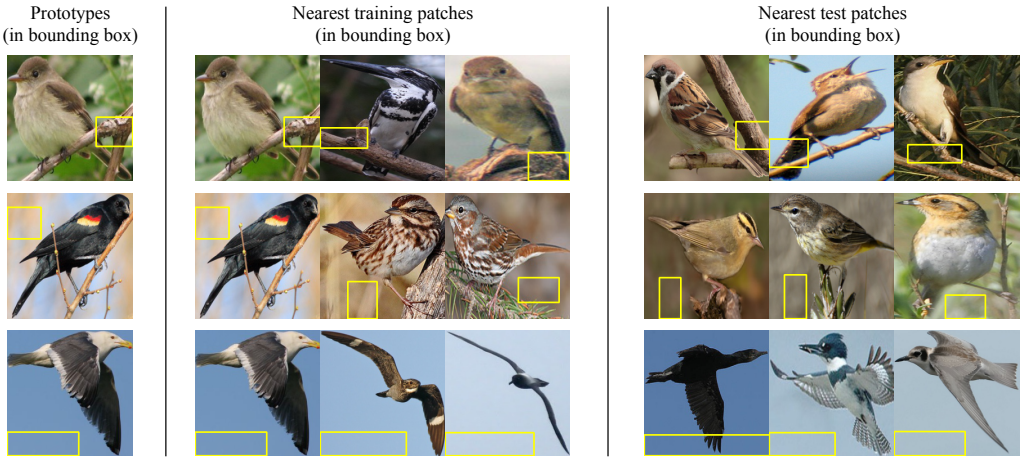


Figure 4.11: Examples of pruned prototypes and their nearest image patches.

Figure 4.11 shows some examples of pruned prototypes and their nearest image patches from the training and the test set. As we can see, the first pruned prototype in Figure 4.11 is a prototypical tree branch, the second some prototypical background color, and the third

corresponds to an image patch of sky.

4.3.8 Proof of Theorem 4.3.1

In this section, we provide a proof for Theorem 4.3.1 in the main paper.

We will introduce another notation, $\mathbf{p}_l^k$, for prototypes of a particular class: here, k represents the class identity of the prototype and l is the index of that prototype among all the prototypes of class k. In this way, the prototypes of class k can be easily denoted as: $\mathbf{P}^k = \{\mathbf{p}_l^k\}_{l=1}^{m_k}$.

Proof of Theorem 4.3.1. For any class k, let $L_k(\mathbf{x}, \{\mathbf{p}_l^k\}_{l=1}^{m'})$ denote its output logit for input image x with the values of class k prototypes being $\{p_l^k\}_{l=1}^{m'}$. By Assumption **(A4)**,

$$L_k(\mathbf{x}, \{\mathbf{p}_l^k\}_{l=1}^{m'}) = \sum_{l=1}^{m'} \log\left(\frac{\|\mathbf{z}_l^k - \mathbf{p}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{p}_l^k\|_2^2 + \epsilon}\right).$$

Let Δ_k denote the change of the output logit of class k as a result of the projection of prototypes $\{p_l^k\}_{l=1}^{m'}$ to their nearest latent training patches. This gives

$$\begin{aligned}
\Delta_k &= L_k(\mathbf{x}, \{\mathbf{a}_l^k\}_{l=1}^{m'}) - L_k(\mathbf{x}, \{\mathbf{b}_l^k\}_{l=1}^{m'}) \\
&= \sum_{l=1}^{m'} \left(\log\left(\frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon}\right) - \log\left(\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}\right)\right) \\
&= \sum_{l=1}^{m'} \log\left(\frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \cdot \frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon}\right).
\end{aligned}$$

For each class $k \in \{1, ..., K\}$ and its prototypes $l \in \{1, ..., m'\}$, let

$$\Psi_l^k = \frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \cdot \frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon}.$$

(**Correct class**) We now derive a lower bound of Ψ_l^c for the l-th prototype of the correct class c.

From the second inequality in **(A2b)**, we have

$$\frac{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + 1}{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + 1} \geq \frac{1}{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + 1} \geq \frac{1}{2 - \delta}. \tag{4.8}$$

Now we want to lower-bound $\frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + \epsilon}$ which is the second term in Ψ_l^c. We shall now prove

$$\frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + \epsilon} \geq \frac{1}{1 + \delta}. \tag{4.9}$$

First, by the triangle inequality, we know $\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2 \leq \|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2$. As a result, we know

$$\frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + \epsilon} \geq \frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{(\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2)^2 + \epsilon}.$$

Then by (**A2b**), we have

$$\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2 \leq (\sqrt{1 + \delta} - 1)\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2,$$

which implies

$$\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 \leq \sqrt{1 + \delta}\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2.$$

Squaring both sides of the above inequality, we have

$$(\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2)^2 \leq (1 + \delta)\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2.$$

Adding ϵ to both sides of the equation, we obtain

$$(\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2)^2 + \epsilon \leq (1 + \delta)\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon \leq (1 + \delta)\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + (1 + \delta)\epsilon.$$

Rearranging, we have

$$\frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{(\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2)^2 + \epsilon} \geq \frac{1}{1+\delta}.$$

Now we obtain the desired result:

$$\frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + \epsilon} \geq \frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{(\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 + \|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2)^2 + \epsilon} \geq \frac{1}{1+\delta}.$$

Combining inequalities (4.8) and (4.9), we have

$$\Psi_l^c = \frac{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + 1}{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + 1} \cdot \frac{\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2^2 + \epsilon}{\|\mathbf{z}_l^c - \mathbf{a}_l^c\|_2^2 + \epsilon} \geq \frac{1}{(1+\delta)(2-\delta)}.$$

This means that the change of the output logit of class c as a result of the prototype projection $\{\mathbf{p}_l^c\}_{l=1}^{m'}$ to their nearest latent training patches satisfies

$$\Delta_c = \sum_{l=1}^{m'} \log \Psi_l^c \geq \sum_{l=1}^{m'} \log\left(\frac{1}{(1+\delta)(2-\delta)}\right) = m' \log\left(\frac{1}{(1+\delta)(2-\delta)}\right),$$

or equivalently,

$$-\Delta_c \leq -m' \log\left(\frac{1}{(1+\delta)(2-\delta)}\right) = m' \log((1+\delta)(2-\delta)).$$

This means that the worst decrease of the output logit of class c as a result of prototype projection is $m' \log((1+\delta)(2-\delta))$, as desired.

(**Wrong class**) We now derive an upper bound of $\Psi_{k,l}$ for the l-th prototype of any incorrect class $k \neq c$. We shall first give an upper bound of $\frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1}$. We show

$$\frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \leq 1 + \delta. \tag{4.10}$$

Using the triangle inequality, we obtain

$$\frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \leq \frac{(\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2)^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1}. \tag{4.11}$$

By Assumption **(A2a)**, we have

$$\|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2 \leq (\sqrt{1+\delta} - 1)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \sqrt{\epsilon} \leq (\sqrt{1+\delta} - 1)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2,$$

which then gives

$$\begin{aligned}(\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2)^2 &\leq (\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + (\sqrt{1+\delta} - 1)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2)^2 \\ &= (\sqrt{1+\delta}\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2)^2 \\ &= (1+\delta)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2.\end{aligned} \tag{4.12}$$

Using the inequality (4.12), we obtain

$$\begin{aligned}\frac{(\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2)^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} &\leq \frac{(1+\delta)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \\ &\leq \frac{(1+\delta)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1 + \delta}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \\ &= 1 + \delta.\end{aligned} \tag{4.13}$$

Combining inequalities (4.11) and (4.13), we have the desired result

$$\frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \leq 1 + \delta.$$

Now we derive an upper bound for $\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon}$. In particular, we show

$$\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon} \leq 2 - \delta. \tag{4.14}$$

By the triangle inequality, we have

$$\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2 \geq (\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2).$$

Additionally, (**A2a**) implies $\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2 > 0$, so we can square both sides of the above inequality and get:

$$\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon} \leq \frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{(\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2)^2 + \epsilon} \leq \left(\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \sqrt{\epsilon}}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2}\right)^2. \tag{4.15}$$

We now only need to upper bound $\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \sqrt{\epsilon}}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2}$. Again, using Assumption **(A2a)**, we have

$$\|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2 \leq \left(1 - \frac{1}{\sqrt{2-\delta}}\right)\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \sqrt{\epsilon}.$$

Rearranging, we have:

$$\frac{1}{\sqrt{2-\delta}}\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \sqrt{\epsilon} \leq \|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2,$$

which leads us to conclude

$$\frac{1}{\sqrt{2-\delta}}\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \frac{\sqrt{\epsilon}}{\sqrt{2-\delta}} \leq \frac{1}{\sqrt{2-\delta}}\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \sqrt{\epsilon} \leq \|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2.$$

The above inequality yields

$$\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 + \sqrt{\epsilon}}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2} \leq \sqrt{2-\delta}. \tag{4.16}$$

Combining inequalities (4.15) and (4.16), we establish the desired inequality

$$\frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon} \leq (\sqrt{2-\delta})^2 = 2 - \delta.$$

Inequalities (4.10) and (4.14) then give us

$$\Psi_l^k = \frac{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + 1}{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + 1} \cdot \frac{\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2^2 + \epsilon}{\|\mathbf{z}_l^k - \mathbf{a}_l^k\|_2^2 + \epsilon} \leq (1+\delta)(2-\delta).$$

This means that the change of the output logit of class $k \neq c$ as a result of the projection of prototypes $\mathbf{p}_l^k$ to their nearest latent training patches satisfies

$$\Delta_k = \sum_{l=1}^{m'} \log \Psi_l^k \leq \sum_{l=1}^{m'} \log((1+\delta)(2-\delta)) = m' \log((1+\delta)(2-\delta)).$$

Since the increase of the output logit of class k is exactly Δ_k, we conclude that the worst increase of the output logit of class $k \neq c$ as a result of prototype projection is $m' \log((1+\delta)(2-\delta))$, as desired.

Finally, let $\Delta_{\max} = m' \log((1+\delta)(2-\delta))$. Suppose that the output logit $L_c(\mathbf{x}|\mathbf{b}_l^c)$ of the correct class c **before** prototype projection is at least $2\Delta_{\max}$ higher than the output logit $L_k(\mathbf{x}|\mathbf{b}_l^k)$ of any other class $k \neq c$, i.e.,

$$L_c(\mathbf{x}, \{\mathbf{b}_l^c\}_{l=1}^{m'}) \geq L_k(\mathbf{x}, \{\mathbf{b}_l^k\}_{l=1}^{m'}) + 2\Delta_{max}, \quad \forall k \neq c$$

Since the output logit of the correct class c satisfies

$$L_c(\mathbf{x}, \{\mathbf{a}_l^c\}_{l=1}^{m'}) \geq L_c(\mathbf{x}, \{\mathbf{b}_l^c\}_{l=1}^{m'}) - \Delta_{\max},$$

while the output logit of any incorrect class $k \neq c$ satifies

$$L_k(\mathbf{x}, \{\mathbf{a}_l^k\}_{l=1}^{m'}) \leq L_k(\mathbf{x}, \{\mathbf{b}_l^k\}_{l=1}^{m'}) + \Delta_{\max},$$

consequently, we have for any $k \neq c$,

$$
\begin{aligned}
L_c(\mathbf{x}, \{\mathbf{a}_l^c\}_{l=1}^{m'}) &\geq L_c(\mathbf{x}, \{\mathbf{b}_l^c\}_{l=1}^{m'}) - \Delta_{\max} \\
&\geq L_k(\mathbf{x}, \{\mathbf{b}_l^k\}_{l=1}^{m'}) + 2\Delta_{\max} - \Delta_{\max} \\
&= L_k(\mathbf{x}, \{\mathbf{b}_l^k\}_{l=1}^{m'}) + \Delta_{\max} \\
&\geq L_k(\mathbf{x}, \{\mathbf{a}_l^k\}_{l=1}^{m'}).
\end{aligned}
$$

Hence, in this case, the input image $\mathbf{x}$ will still be correctly classified as class c.

□

Interpretation of Theorem 4.3.1

Theorem 4.3.1 is presented in a general way in terms of the choice for δ. However, to get a concrete feeling of the assumption, we can for simplicity set $\delta = \frac{9}{16}$.

Then (**A2a**) becomes $\|\mathbf{a}_l^k - \mathbf{b}_l^k\|_2 \leq (1 - \frac{4}{\sqrt{23}})\|\mathbf{z}_l^k - \mathbf{b}_l^k\|_2 - \sqrt{\epsilon}$. And $(1 - \frac{4}{\sqrt{23}}) \approx 0.17$, while

(**A2b**) becomes $\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2 \leq \frac{1}{4}\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2$ and $\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 \leq \frac{\sqrt{7}}{4}$.

The requirement of $\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2 \leq \frac{\sqrt{7}}{4}$ is empirically always satisfied on our learned models. Regarding the relationship between $\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2$ and $\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2$, we can see that the requirement is tighter on the incorrect classes than the correct class. This is because for the wrong classes, we would expect that there exists no latent patch representation of the class-c image $\mathbf{x}$ that is **very close** to non-class-c prototypes. On the other hand, because our projection update pushes every non-class-c prototype to the closest representation from

its own class, the distance $\|\mathbf{a}_l^c - \mathbf{b}_l^c\|_2$ is generally much smaller than $\|\mathbf{z}_l^c - \mathbf{b}_l^c\|_2$. From this, we see that the assumptions made in the theorem are reasonable.

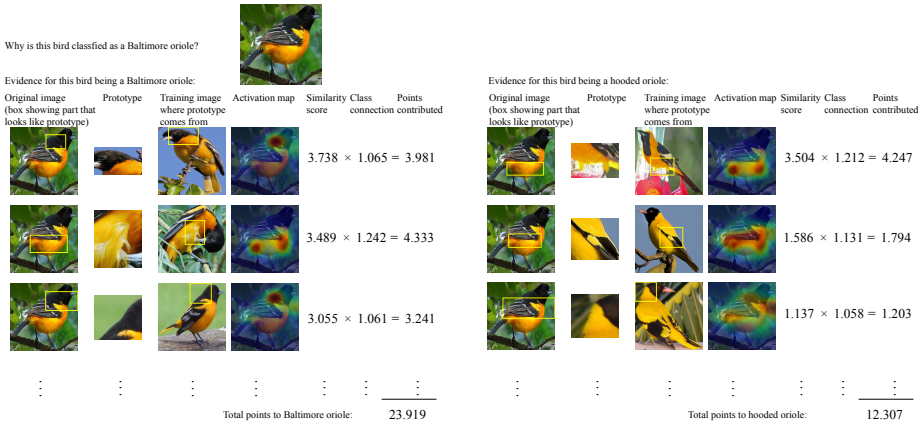
When the conditions are met, this theorem provably guarantees that the classifier's decision does not become worse on a large region in the image domain.

4.3.9 More Examples of How ProtoPNet Classifies Birds

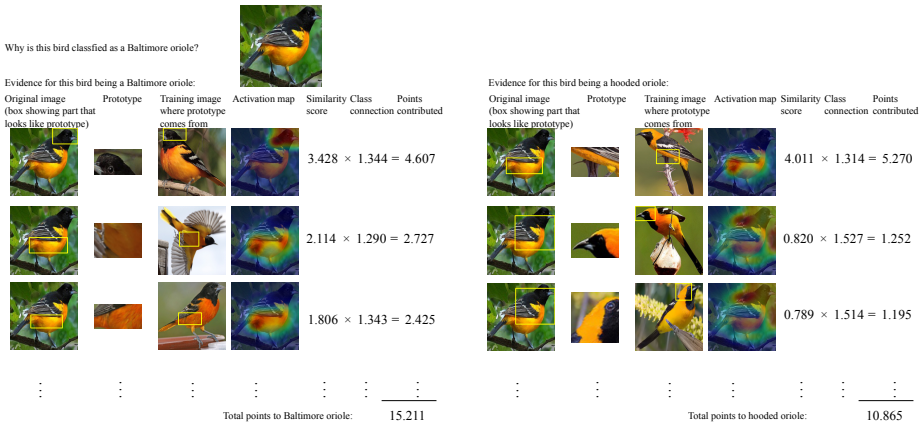
In this section, we provide more examples of how our ProtoPNet classifies previously unseen images of birds.

Figures 4.12 and 4.13 provide two examples of how our ProtoPNet (with various base architectures) correctly classifies a previously unseen image of a bird and how our network explains its prediction. In each of these figures, the left side presents evidence for the given bird belonging to the class with the highest logit, and the right side presents evidence for the given bird belonging to a closely related class. We shall give some general observations regarding the ways in which our network thinks that the given image is similar to the prototypical cases. The detailed reasoning process of our network has been explained in our main paper, and will not be repeated here.

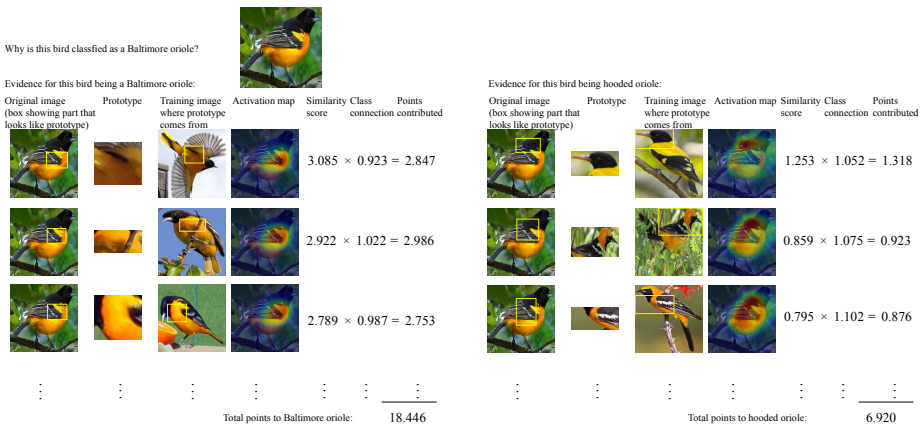
Figure 4.12 demonstrates how our ProtoPNet (with various base architectures) correctly classifies an image of a Baltimore oriole. In particular, every ProtoPNet is able to learn the prototypical golden chest/abdomen of a Baltimore oriole, and is able to associate the golden chest/abdomen of the (previously unseen) given image to the prototypical golden chest (or abdomen) of a Baltimore oriole: for each of those prototypes that correspond to



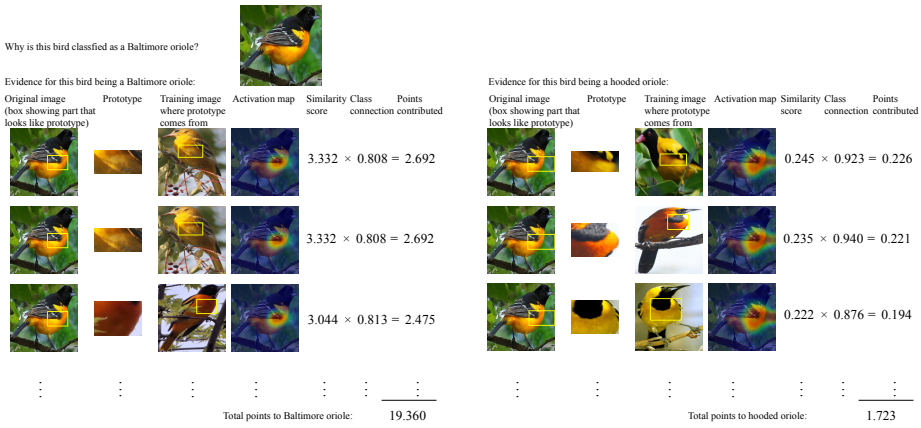
(a) VGG16-based ProtoPNet.



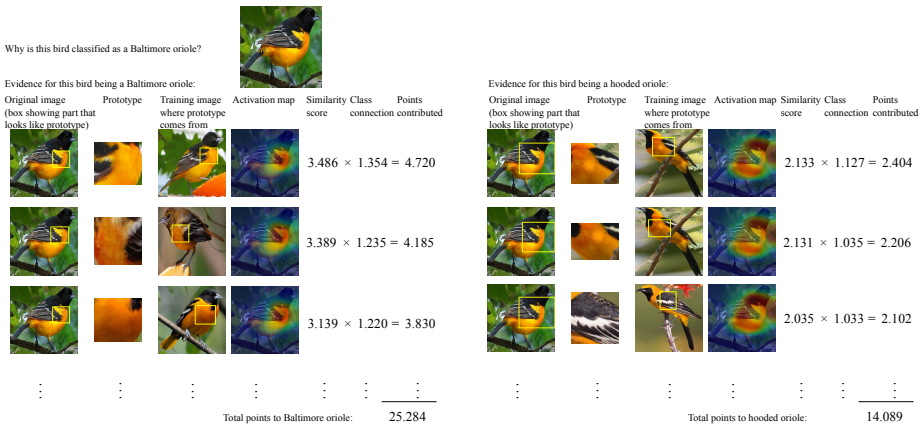
(b) VGG19-based ProtoPNet.



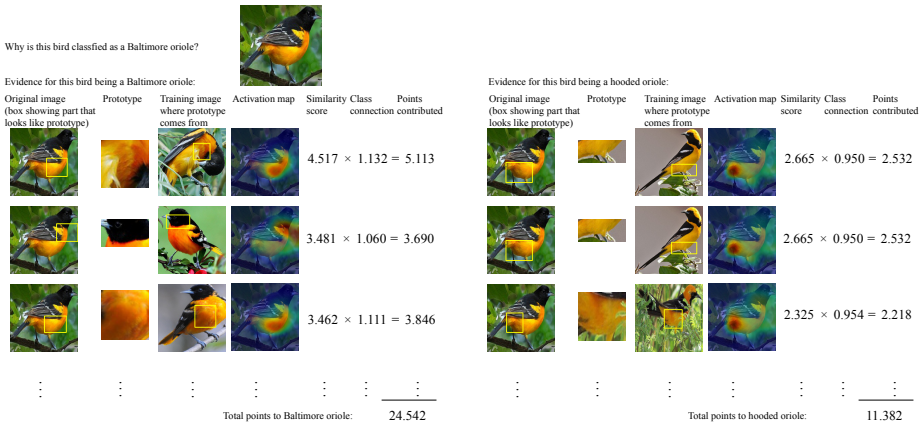
(c) ResNet34-based ProtoPNet.



(d) ResNet152-based ProtoPNet.

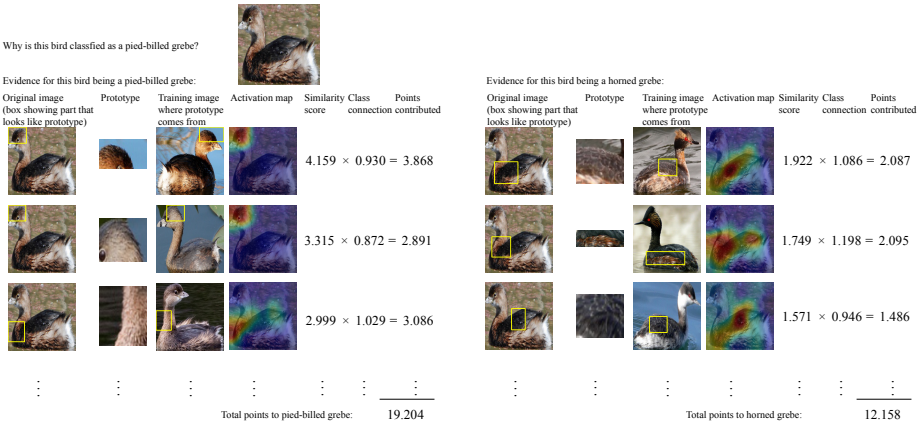


(e) DenseNet121-based ProtoPNet.

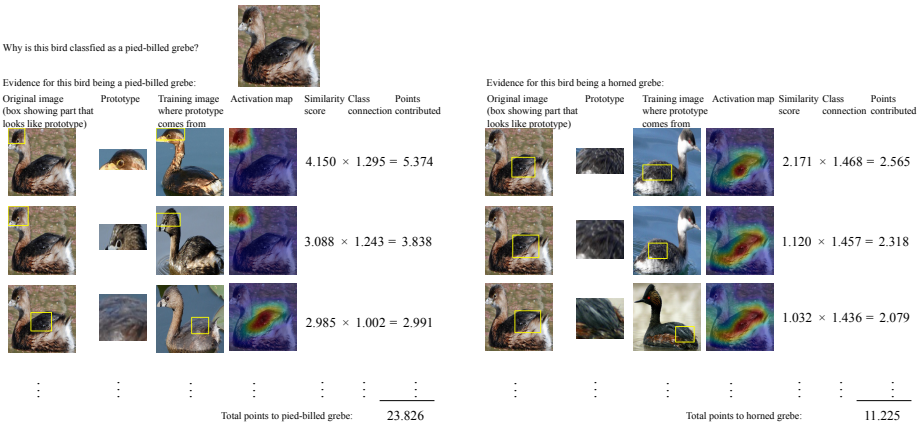


(f) DenseNet161-based ProtoPNet.

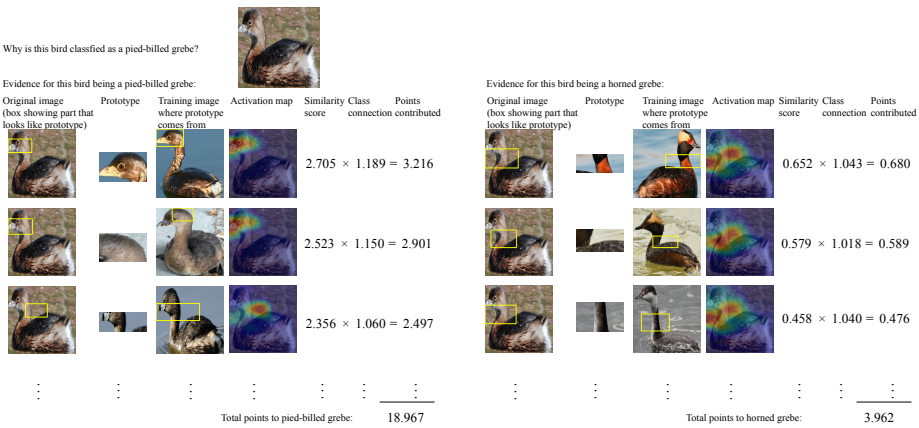
Figure 4.12: How ProtoPNet correctly classifies a Baltimore oriole.



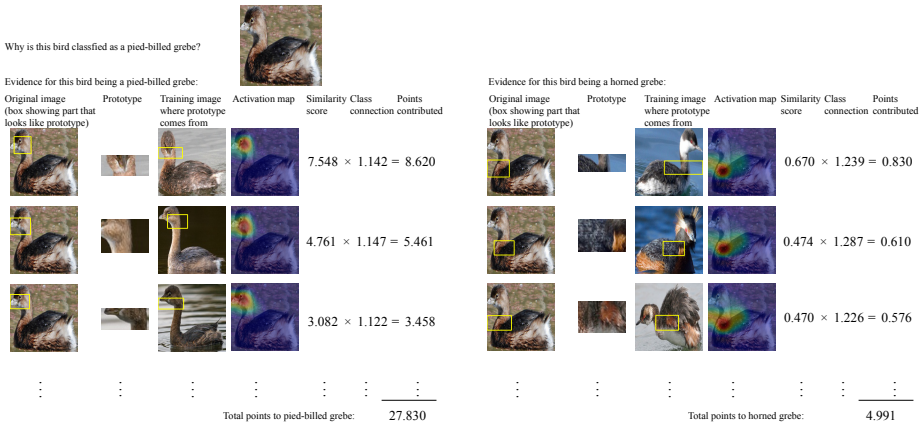
(a) VGG16-based ProtoPNet.



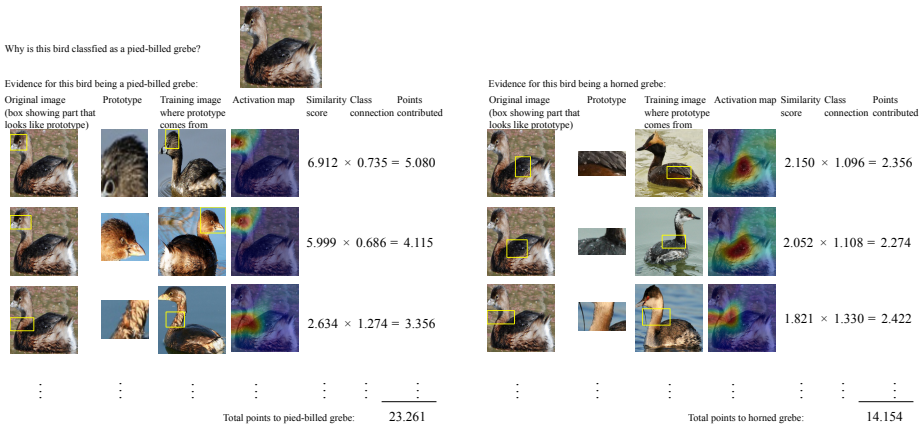
(b) VGG19-based ProtoPNet.



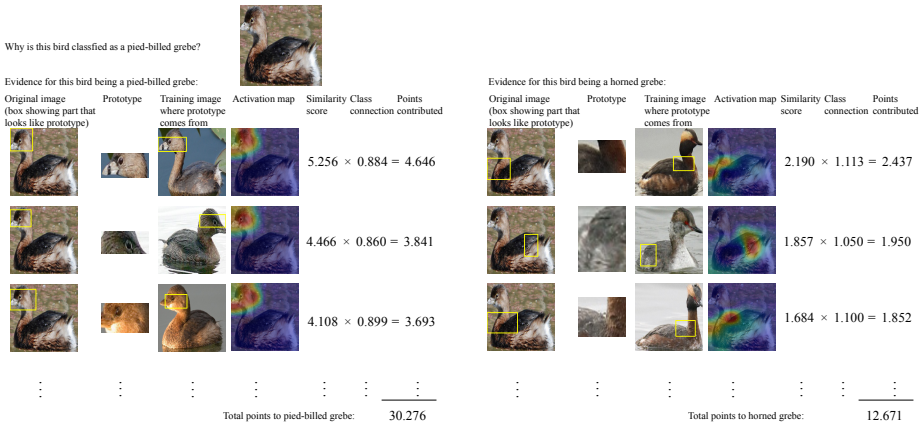
(c) ResNet34-based ProtoPNet.



(d) ResNet152-based ProtoPNet.

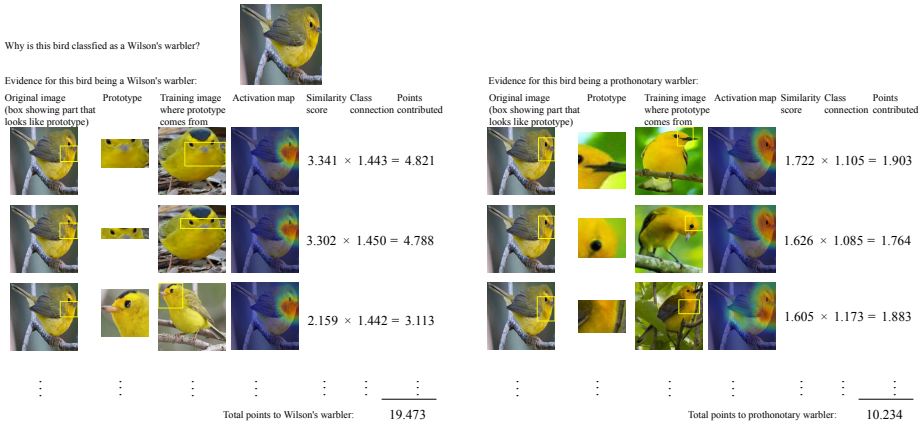


(e) DenseNet121-based ProtoPNet.

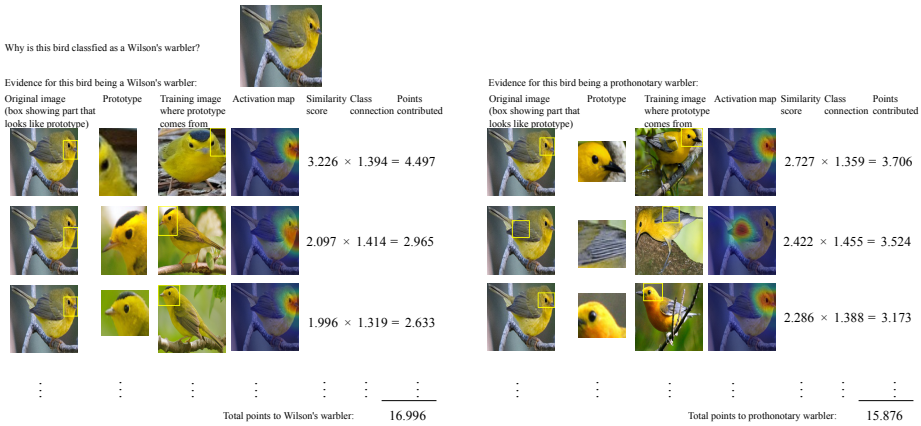


(f) DenseNet161-based ProtoPNet.

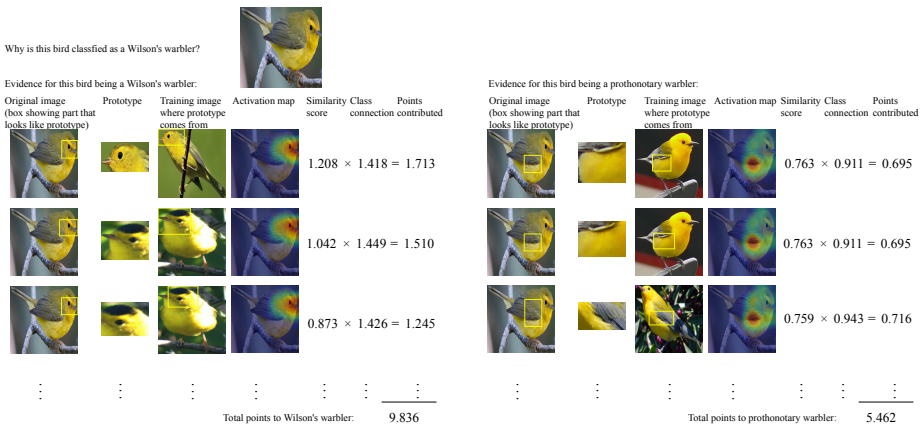
Figure 4.13: How ProtoPNet correctly classifies a pied-billed grebe.



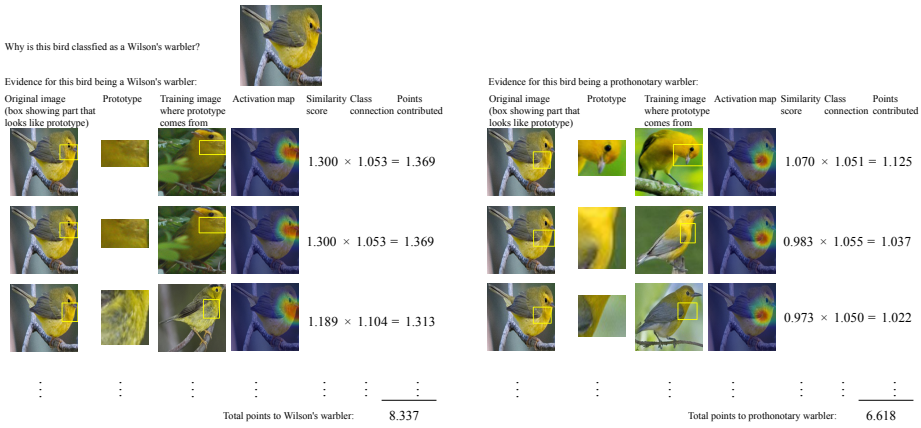
(a) VGG16-based ProtoPNet.



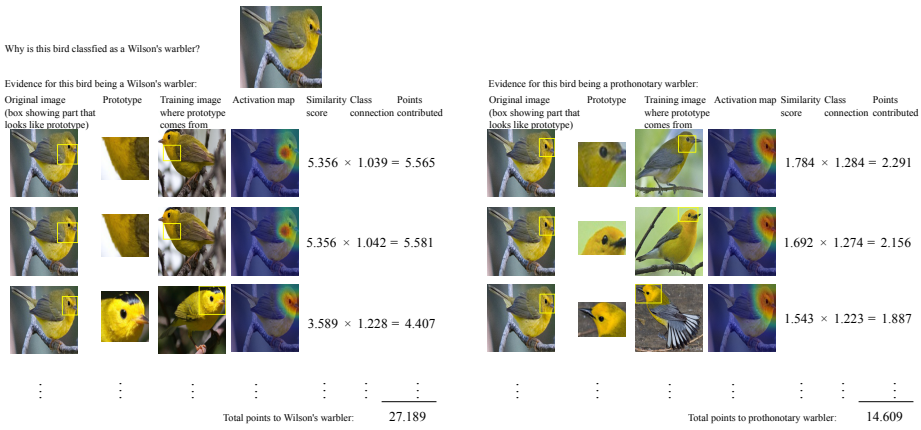
(b) VGG19-based ProtoPNet.



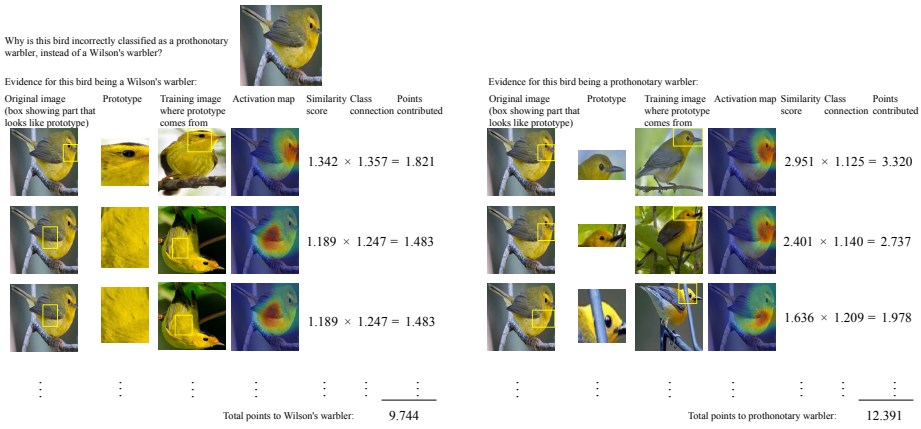
(c) ResNet34-based ProtoPNet.



(d) ResNet152-based ProtoPNet.



(e) DenseNet121-based ProtoPNet.



(f) DenseNet161-based ProtoPNet: this network mistakes the Wilson's warbler as a prothonotary warbler.

Figure 4.14: How ProtoPNet correctly and incorrectly classifies a Wilson's warbler.

the characteristic golden chest/abdomen of a Baltimore oriole, our network is able to pick out, on the previously unseen given image, a similar patch with a golden chest/abdomen and highlight the golden chest/abdomen on the previously unseen bird in its prototype activation map (e.g., the first prototype in Figure 4.12e (left), which corresponds to the characteristic golden chest of a Baltimore oriole, identifies and highlights the golden chest on the previously unseen bird). On the other hand, some of our networks also think that the the golden abdomen of the given bird looks like the prototypical golden abdomen of a hooded oriole (e.g., the first prototype in Figure 4.12a (left), which corresponds to the characteristic golden abdomen of a hooded oriole, identifies and highlights the golden abdomen on the given bird). It is worth pointing out that sometimes a network can "mistakenly" believe that a certain pattern in the given image looks like a prototypical part of some class of birds: for example, the second and the third prototype in Figure 4.12b (left) "mistakenly" think that the wing and the chest of the given bird looks like the prototypical head of a hooded oriole – this is, however, not too surprising because the orange-black coloration on the wing and the chest of the given bird does look somewhat like the same coloration on the prototypical head of a hooded oriole; however, such remote resemblance is reflected in our network by very small similarity scores between the given image and those two prototypes (0.820 and 0.789), which (fortunately) means that the network does not believe that the aforementioned similarity is strong. Not surprisingly, when our ProtoPNet accumulates the evidence presented by the comparison with all the prototypes, it sees that the evidence for

the given bird being a Baltimore oriole is the strongest, and concludes that the bird is a Baltimore oriole.

Note that sometimes a prototype can be duplicated in our network (e.g., the first and the second prototype in Figure 4.12d are the same): this results from the projection of each prototype onto the closest latent representation of training image patches from the prototype's designated class (described in Section 2.2: Training algorithm in the main paper) – in this case, the closest training patches to both prototypes are the same before the projection stage, and consequently both prototypes are projected onto the same patch in the latent space. This means that some of the learned prototypes in our network are repeated. However, this is not a problem because we can conceptually understand the repeated prototypes as one prototype, with its weight connection to each class in the fully connected last layer being the sum of the weight connections of those repeated prototypes to that class. Thus, we can understand the first and the second prototype in Figure 4.12d as one Baltimore oriole prototype with class connection $0.808 + 0.808 = 1.616$. This also means that the actual number of prototypes used by our ProtoPNet is in general *less* than the pre-determined number of prototypes when the network architecture is specified.

Figure 4.13 demonstrates how our ProtoPNet correctly classifies a previously unseen image of a pied-billed grebe. In particular, every ProtoPNet is able to learn the prototypical head of a pied-billed grebe, and is able to associate the head of the given bird to the prototypical head of a pied-billed grebe: for each of those prototypes that correspond to

the head of a pied-billed grebe, our network is able to pick out, on the previously unseen image, a similar patch that contains the head of the given bird, and also highlight the head of the given bird in its prototype activation map – this strong resemblance is also reflected in high similarity scores between the given image and those prototypes. This shows that our network thinks that the head of the given bird looks like the prototypical head of a pied-billed grebe. On the other hand, each of our networks also thinks that there is some resemblance between the neck/back of the given bird and that of a horned grebe, but such resemblance is not very strong.

How Model Combination Can Improve Accuracy

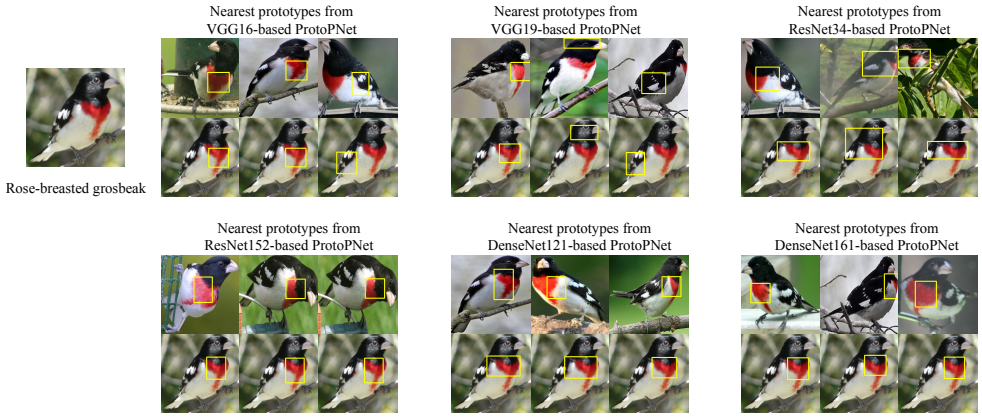
Figures 4.14a through 4.14e demonstrates how our ProtoPNet based on VGG16, VGG19, ResNet34, ResNet152, and DenseNet121 correctly classifies an image of a Wilson's warbler. On the other hand, Figure 4.14f shows why the DenseNet161-based ProtoPNet misclassifies the given bird as a prothonotary warbler instead of its true identity – a Wilson's warbler. As we can see in Figure 4.14f (right), the DenseNet161-based ProtoPNet (mistakenly) thinks that the head of the given bird is more similar to the prototypical head of a prothonotary warbler than to that of a Wilson's warbler: this is, for example, shown by the first prothonotary warbler prototype (in 4.14f (right)) having a higher similarity score than the first Wilson's warbler prototype (in 4.14f (left)). In the end, this network finds more evidence for the given bird being a prothonotary warbler than being a Wilson warbler, and misclassifies the bird as a prothonotary warbler.

Here we illustrate why combining several ProtoPNet models is a good idea: it can improve prediction accuracy while preserving interpretability. For simplicity, suppose that we combine the VGG16-based and the DenseNet161-based ProtoPNet together, by adding the logits (i.e., the total points to different classes) together. When the combined model makes a prediction on the Wilson's warbler, the total points to Wilson's warbler becomes $19.473 + 9.744 = 29.217$, while the total points to prothonotary warbler becomes $10.234 + 12.391 = 22.625$. Thus, the combined network will correctly classify the image as a Wilson's warbler. At the same time, adding the logits (the total points to different classes) together in this way **preserves the interpretability** of our ProtoPNet model, because it is conceptually equivalent to having a scoring sheet that makes comparison with more prototypes (in different latent spaces) and accumulating all the evidence together into the total points for each class.

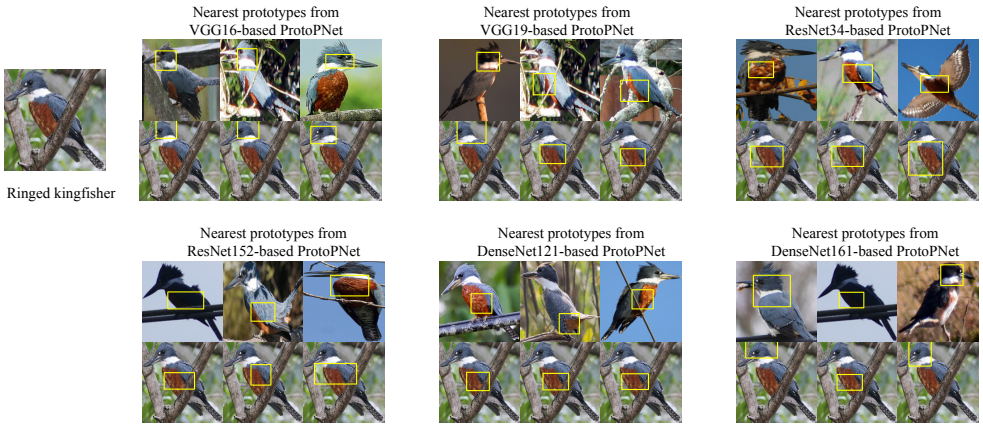
4.3.10 More Examples of Nearest Prototypes of Given Images

In this section, we provide more examples of the nearest prototypes of given test images.

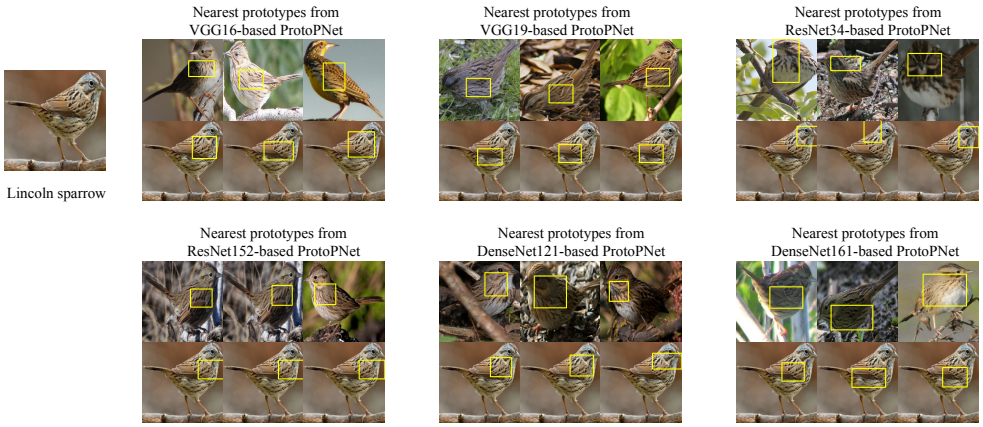
Figure 4.15 shows the three nearest prototypes to each of the five test bird images, from different ProtoPNet models (with various base architectures): for each ProtoPNet, the three nearest prototypes to a given test bird image are displayed, with prototypical parts shown in boxes, on the top row, and the same test image with the patch closest to each prototype shown in a bounding box, is displayed below the corresponding prototype. For a given



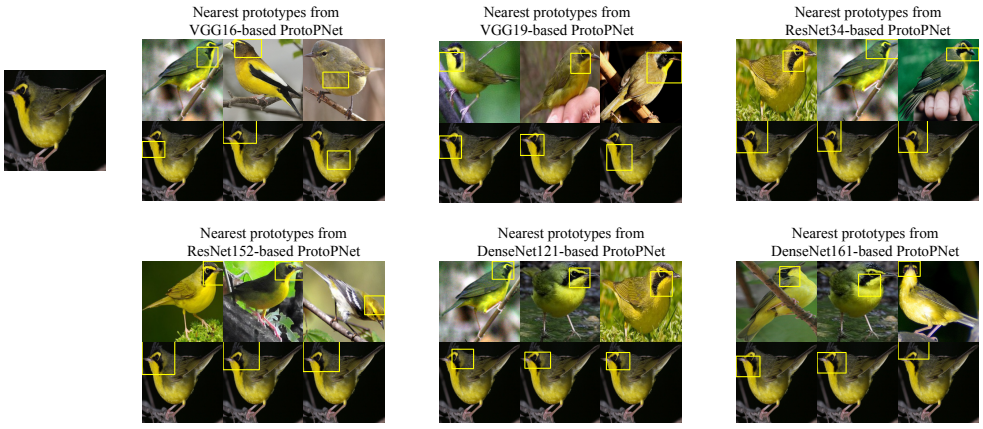
(a) Nearest prototypes of a rose-breasted grosbeak from ProtoPNet models.



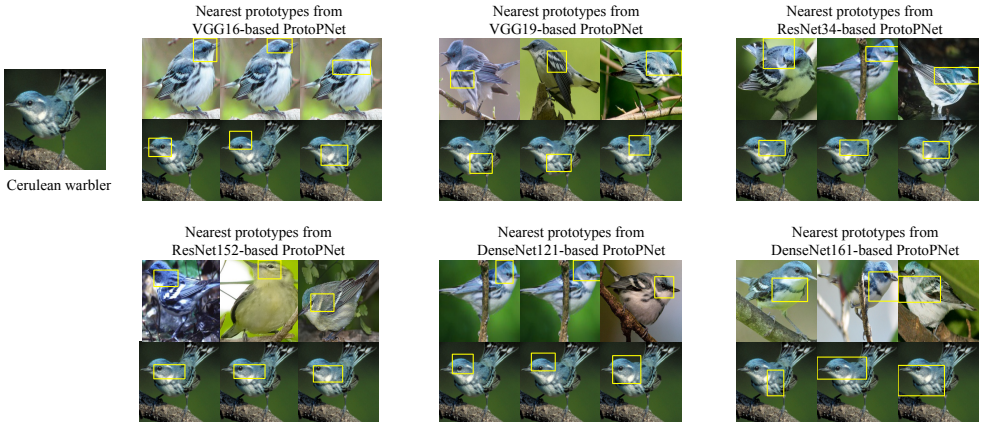
(b) Nearest prototypes of a ringed kingfisher from ProtoPNet models.



(c) Nearest prototypes of a Lincoln sparrow from ProtoPNet models.



(d) Nearest prototypes of a Kentucky warbler from ProtoPNet models.

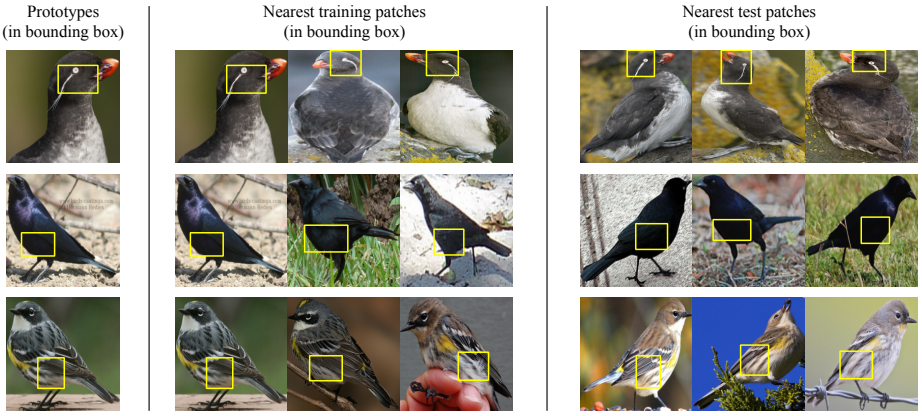


(e) Nearest prototypes of a Cerulean warbler from ProtoPNet models.

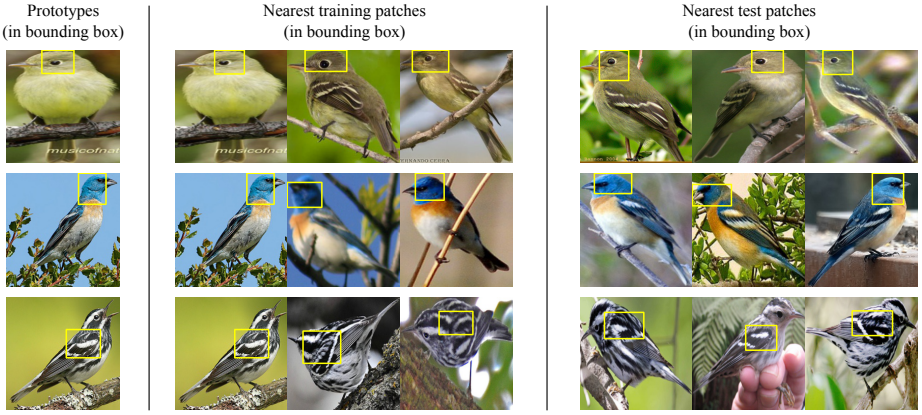
Figure 4.15: Nearest prototypes of five test images.

In each group of images, the three nearest prototypes of the corresponding test image are displayed, with prototypical parts shown in boxes, on the top row, and the same test image with the patch closest to each prototype shown in a bounding box, is displayed below the corresponding prototype.

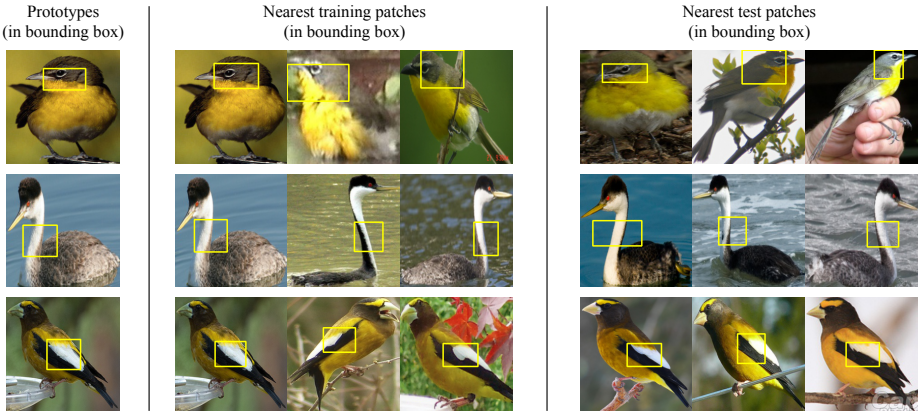
image, we define its nearest prototype as the one that forms the closest patch-prototype pair in the latent space, over all latent patches of the given image. As we can see from Figure 4.15, the nearest prototypes for each of these test images generally come from the same class as that of the image, and the patch that is closest to (i.e., most activated by) each prototype also corresponds to the same semantic concept. For example, Figure 4.15a shows the three nearest prototypes to a test image of a rose-breasted grosbeak, from different ProtoPNet models (with various base architectures): as we can see, the three nearest prototypes from each of the ProtoPNet models indeed all come from the rose-breasted grosbeak class, and moreover, the nearest prototype from each of the ProtoPNet models corresponds to the rose breast characteristic of the species, and the closest (i.e., most activated) patch to the prototype indeed localizes the rose breast of the given bird. There are some exceptions: for example, the third nearest prototype from the VGG16-based ProtoPNet for the Lincoln sparrow corresponds to the wing of a western meadowlark (see Figure 4.15c. This is understandable, because a Lincoln sparrow has wing stripes much like those of a western meadowlark. Hence, it is not too surprising that the wing of the Lincoln sparrow in Figure 4.15c is fairly close to the prototypical wing of a western meadowlark. This shows that the latent space learned by our ProtoPNet does have a clustering structure, where semantically similar patches that are relevant for classification are clustered together.



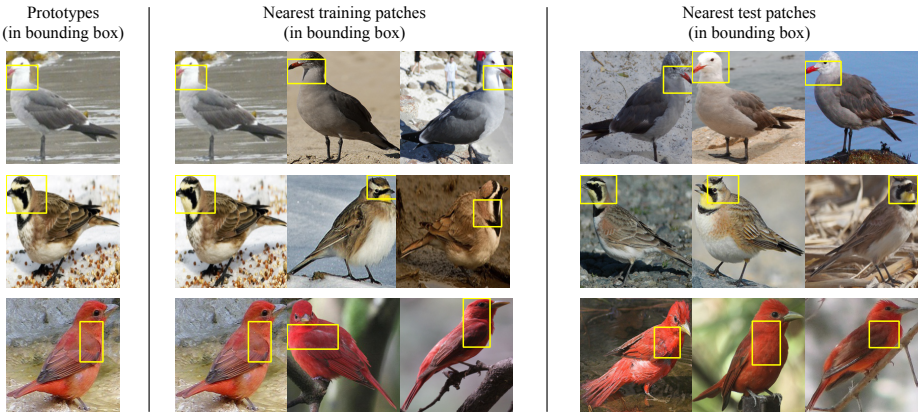
(a) Nearest image patches to prototypes from VGG16-based ProtoPNet.



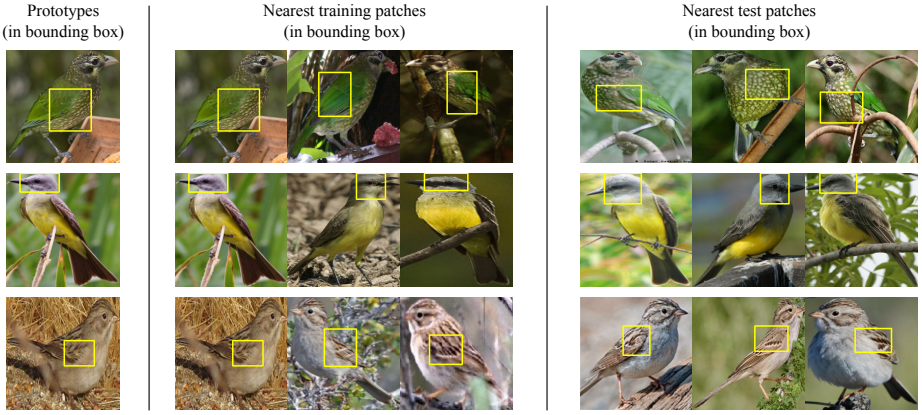
(b) Nearest image patches to prototypes from VGG19-based ProtoPNet.



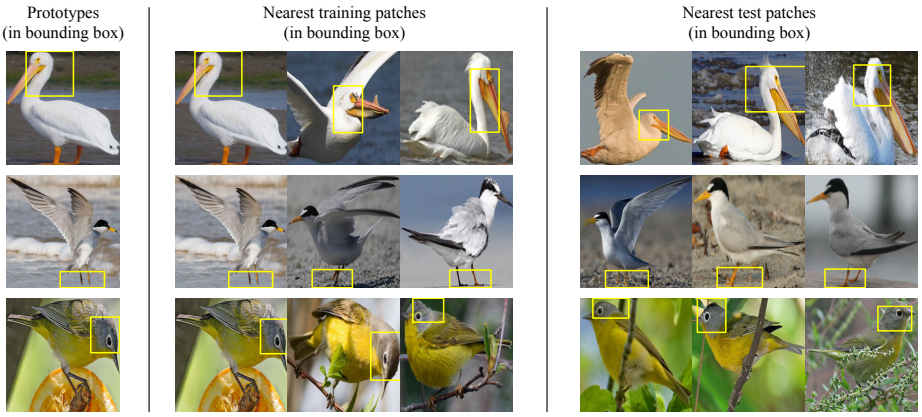
(c) Nearest image patches to prototypes from ResNet34-based ProtoPNet.



(d) Nearest image patches to prototypes from ResNet152-based ProtoPNet.



(e) Nearest image patches to prototypes from DenseNet121-based ProtoPNet.



(f) Nearest image patches to prototypes from DenseNet161-based ProtoPNet.

Figure 4.16: Nearest (most activated) image patches to prototypes.

4.3.11 More Examples of Nearest Training Patches of Given Prototypes

Figure 4.16 shows the nearest training and test image patches of three prototypes from different ProtoPNet models (with various base architectures). The prototypes are displayed with prototypical parts shown in bounding boxes, and the nearest training and test images to each prototype are displayed with the patch closest to that prototype in a bounding box. As we can see, the nearest image patches to each prototype in the figure all localize the same semantic part as the prototypical part of that prototype, and it is generally true that the nearest patches of a prototype mostly come from those images in the same class as that of the prototype.

4.4 Discussion

In this chapter, we have defined a form of interpretability in image processing (*this* looks like *that*) that agrees with the way humans describe their own reasoning in classification. We have presented two network architectures – PrototypeNet and ProtoPNet that accommodate this form of interpretability. We have described our specialized training objective/algorithm, and applied our technique to handwritten digit recognition and bird species identification.

Code: The code for PrototypeNet is available at

`https://github.com/OscarcarLi/PrototypeDL`

The code for ProtoPNet is available at

`https://github.com/cfchen-duke/ProtoPNet`

Chapter 5

Conclusion

As the popularity of machine learning models grow, model interpretability has become a more pressing issue than ever. There is an urgent need for predictive models that are not just accurate, but also transparent, so that we can trust the predictions made by these models, especially in high-stakes settings. In this dissertation, I have discussed the notion of interpretability, both predicate-based and case-based interpretability. I have presented novel predicate-based interpretable models and methods – optimized falling rule lists and two-layer additive models, and their applications to understanding low-dimensional structured data. I have also presented case-based interpretable deep models for high-dimensional image data. Given the empirical evidence, I conclude that, with novel model architectures or regularization techniques, machine learning models can be carefully crafted to incorporate interpretability and encourage transparency.

Bibliography

[ALMK16] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine Bias. https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing, 2016.

[ALSA+17] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. Learning Certifiably Optimal Rule Lists. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '17, pages 35–44, New York, NY, USA, 2017. ACM.

[ARD05] Eric E. Altendorf, Angelo C. Restificar, and Thomas G. Dietterich. Learning from Sparse Data by Exploiting Monotonicity Constraints. In *Proceedings of the Twenty-First Conference on Uncertainty in Artificial Intelligence*, pages 18–26. AUAI Press, 2005.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, pages 487–499, 1994.

[AVW+18] Ashraf Abdul, Jo Vermeulen, Danding Wang, Brian Y Lim, and Mohan Kankanhalli. Trends and Trajectories for Explainable, Accountable and Intelligible Systems: An HCI Research Agenda. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI'18*, 2018.

[BD92] Arie Ben-David. Automatic Generation of Symbolic Multiattribute Ordinal Knowledge-Based DSSs: Methodology and Applications. *Decision Sciences*, 23(6):1357–1372, 1992.

[BD95] Arie Ben-David. Monotonicity Maintenance in Information-Theoretic Machine Learning Algorithms. *Machine Learning*, 19:29–43, 1995.

[BDSP89] Arie Ben-David, Leon Sterling, and Yoh-Han Pao. Learning and classification of monotonic ordinal concepts. *Computational Intelligence*, 5(1):45–49, 1989.

[BFSO84] Leo Breiman, Jerome H. Friedman, Charles J. Stone, and Richard A. Olshen. Classification and Regression Trees. 1984.

[Bre01] Leo Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

[BT11] Jacob Bien and Robert Tibshirani. Prototype Selection for Interpretable Classification. *Annals of Applied Statistics*, 5(4):2403–2424, 2011.

[BVHBP14] Steve Branson, Grant Van Horn, Serge Belongie, and Pietro Perona. Bird Species Categorization Using Pose Normalized Deep Convolutional Nets. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[BZK+17] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network Dissection: Quantifying Interpretability of Deep Visual Representations. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 3319–3327. IEEE, 2017.

[Cit16] Danielle Citron. (Un)Fairness of Risk Scores in Criminal Sentencing. *Forbes, Tech section*, July 2016.

[CLR+18] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. An Interpretable Model with Globally Consistent Explanations for Credit Risk. *NIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy*, 2018.

[CLT+19] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *Advances in Neural Information Processing Systems (NeurIPS 2019)*, 2019.

[Coh95] William W. Cohen. Fast Effective Rule Induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.

[CR18] Chaofan Chen and Cynthia Rudin. An Optimization Approach to Learning Falling Rule Lists. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84. PMLR, 2018.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. IEEE, 2009.

[DGW18] Sanjeeb Dash, Oktay Gunluk, and Dennis Wei. Boolean Decision Rules via Column Generation. In *Advances in Neural Information Processing Systems (NIPS 2018)*, pages 4655–4665, 2018.

[EBCV09] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing Higher-Layer Features of a Deep Network. Technical Report 1341, the University of Montreal, June 2009. Also presented at the Workshop on Learning Feature Hierarchies at the 26th International Conference on Machine Learning (ICML 2009), Montreal, Canada.

[Edw17] Brian Edwards. FDA Guidance on Clinical Decision Support: Peering Inside the Black Box of Algorithmic Intelligence. https://www.chilmarkresearch.com/fda-guidance-clinical-decision-support/, December 2017. Online; accessed March 13, 2018.

[FFP05] Li Fei-Fei and Pietro Perona. A Bayesian Hierarchical Model for Learning Natural Scene Categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 524–531. IEEE, 2005.

[FP03] Ad Feelders and Martijn Pardoel. Pruning for Monotone Classification Trees. *Advances in Intelligent Data Analysis V, IDA 2003, Lecture Notes in Computer Science*, 2810:1–12, 2003.

[Fre14] Alex A. Freitas. Comprehensible Classification Models – a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[FS96] Yoav Freund and Robert E. Schapire. Experiments with a New Boosting Algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, volume 96, pages 148–156, 1996.

[FZM17] Jianlong Fu, Heliang Zheng, and Tao Mei. Look Closer to See Better: Recurrent Attention Convolutional Neural Network for Fine-grained Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4438–4446, 2017.

[GCV+18] Rory Mc Grath, Luca Costabello, Chan Le Van, Paul Sweeney, Farbod Kamiab, Zhao Shen, and Freddy Lecue. Interpretable credit application

predictions with counterfactual explanations. *NIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy*, 2018.

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[GHYB20] Oscar Gomez, Steffen Holter, Jun Yuan, and Enrico Bertini. Vice: Visual counterfactual explanations for machine learning models. In *Proceedings of the 25th International Conference on Intelligent User Interfaces (IUI'20)*, 2020.

[Gir15] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[GS19] Kamaledin Ghiasi-Shirazi. Generalizing the Convolution Operator in Convolutional Neural Networks. *Neural Processing Letters*, 2019.

[GWS+01] Brian F. Gage, Amy D. Waterman, William Shannon, Michael Boechler, Michael W. Rich, and Martha J. Radford. Validation of Clinical Classification Schemes for Predicting StrokeResults From the National Registry of Atrial Fibrillation. *JAMA*, 285(22):2864–2870, 06 2001.

[HBSP05] Alec Holt, Isabelle Bichindaritz, Rainer Schmidt, and Petra Perner. Medical applications in case-based reasoning. *The Knowledge Engineering Review*, 20:289–292, 09 2005.

[HCLR19] Peter Hase, Chaofan Chen, Oscar Li, and Cynthia Rudin. Interpretable Image Recognition with Hierarchical Prototypes. In *Proceedings of the Seventh AAAI Conference on Human Computation and Crowdsourcing (AAAI-HCOMP)*, 2019.

[HDM+11] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[Hin12] Geoffrey E. Hinton. A Practical Guide to Training Restricted Boltzmann Machines. In *Neural Networks: Tricks of the Trade*, pages 599–619. Springer, 2012.

[HLvdMW17] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely Connected Convolutional Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017.

[HP00] Jiawei Han and Jian Pei. Mining Frequent Patterns by Pattern-Growth: Methodology and Implications. *ACM SIGKDD explorations newsletter*, 2(2):14–20, 2000.

[HPY00] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns without Candidate Generation. *ACM SIGMOD Record*, 29(2):1–12, 2000.

[HS06] G E Hinton and R R Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, July 2006.

[HXTZ16] Shaoli Huang, Zhe Xu, Dacheng Tao, and Ya Zhang. Part-Stacked CNN for Fine-Grained Visual Categorization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1173–1182, 2016.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[JNY07] Yu-Gang Jiang, Chong-Wah Ngo, and Jun Yang. Towards Optimal Bag-of-Features for Object Categorization and Semantic Video Retrieval. In *Proceedings of the 6th ACM International Conference on Image and Video Retrieval*, pages 494–501. ACM, 2007.

[JSZK15] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 2017–2025, 2015.

[Kan96] William B. Kannel. Blood Pressure as a Cardiovascular Risk Factor: Prevention and Treatment. *JAMA*, 275(20):1571–1576, 05 1996.

[KJYFF15] Jonathan Krause, Hailin Jin, Jianchao Yang, and Li Fei-Fei. Fine-Grained Recognition without Part Annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5546–5555, 2015.

[Kod94] Yves Kodratoff. The comprehensibility manifesto. *KDD Nuggets*, 94(9), 1994.

[KRS14] Been Kim, Cynthia Rudin, and Julie Shah. The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification. In *Advances in Neural Information Processing Systems 27 (NIPS 2014)*, pages 1952–1960, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, pages 1097–1105. 2012.

[LBBH98] Yann Lecun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

[LGRN09] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, pages 609–616, 2009.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating Classification and Association Rule Mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86, 1998.

[LKDR05] Niels Landwehr, Kristian Kersting, and Luc De Raedt. nFOIL: Integrating Naïve Bayes and FOIL. In *Proceedings of the twentieth national conference on artificial intelligence (AAAI-05)*, pages 795–800, 2005.

[LLCR18] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018.

[Low99] David G. Lowe. Object Recognition from Local Scale-Invariant Features. In *Proceedings of the International Conference on Computer Vision (ICCV)*, volume 99, pages 1150–1157, 1999.

[LRM15] Tsung-Yu Lin, Aruni RoyChowdhury, and Subhransu Maji. Bilinear CNN Models for Fine-grained Visual Recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1449–1457, 2015.

[LRMM15] Benjamin Letham, Cynthia Rudin, Tyler H. McCormick, and David Madigan. Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model. *Annals of Applied Statistics*, 9(3):1350–1371, 2015.

[LSLJ15] Di Lin, Xiaoyong Shen, Cewu Lu, and Jiaya Jia. Deep LAC: Deep Localization, Alignment and Classification for Fine-grained Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1666–1674, 2015.

[LSP06] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2169–2178. IEEE, 2006.

[LXW+16] Xiao Liu, Tian Xia, Jiang Wang, Yi Yang, Feng Zhou, and Yuanqing Lin. Fully Convolutional Attention Networks for Fine-Grained Recognition. *arXiv preprint arXiv:1603.06765*, 2016.

[MB10] David Martens and Bart Baesens. Building Acceptable Classification Models. *Data Mining*, pages 53–74, 2010.

[MCR14] Sérgio Moro, Paulo Cortez, and Paulo Rita. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems*, 62:22–31, 2014.

[MDR94] Stephen Muggleton and Luc De Raedt. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*, 19:629–679, 1994.

[Med18] Mdcalc - medical calculators, equations, algorithms, and scores. `https://www.mdcalc.com`, 2018. Accessed: 2018-11-12.

[MLC11] Sergio Moro, Raul Laureano, and Paulo Cortez. Using Data Mining for Bank Direct Marketing: An Application of the CRISP-DM Methodology. In *Proceedings of European Simulation and Modelling Conference (ESM'2011)*, pages 117–121. Eurosis, 2011.

[MPP09] Antonio Mucherino, Petraq J. Papajorgji, and Panos M. Pardalos. *k-Nearest Neighbor Classification*, pages 83–106. Springer New York, New York, NY, 2009.

[MXQR19] Yao Ming, Panpan Xu, Huamin Qu, and Liu Ren. Interpretable and Steerable Sequence Learning via Prototypes. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*, pages 903–913. ACM, 2019.

[NDY+16] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems 29 (NIPS 2016)*, pages 3387–3395, 2016.

[NGSAT17] Keivan Nalaie, Kamaledin Ghiasi-Shirazi, and Modhammad-R Akbarzadeh-T. Efficient Implementation of a Generalized Convolutional Neural Networks based on Weighted Euclidean Distance. In *2017 7th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 211–216. IEEE, 2017.

[NS06] David Nister and Henrik Stewenius. Scalable Recognition with a Vocabulary Tree. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages 2161–2168. IEEE, 2006.

[PM18] Nicolas Papernot and Patrick McDaniel. Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning. *arXiv preprint arXiv:1803.04765*, 2018.

[PMDS03] Carey E. Priebe, David J. Marchette, Jason G. DeVinney, and Diego A. Socolinsky. Classification Using Class Cover Catch Digraphs. *Journal of Classification*, 20(1):003–023, 2003.

[Qui86] J. Ross Quinlan. Induction of Decision Trees. *Machine Learning*, 1(1):81–106, 1986.

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

[Qui04] J. Ross Quinlan. Data Mining Tools See5 and C5.0. `https://www.rulequest.com/see5-info.html`, 2004.

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28 (NIPS 2015)*, pages 91–99, 2015.

[Riv87] Ronald L. Rivest. Learning Decision Lists. *Machine Learning*, 2(3):229–246, 1987.

[SCD+17] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[SH07] Ruslan Salakhutdinov and Geoffrey Hinton. Learning a Nonlinear Embedding by Preserving Class Neighbourhood Structure. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 2 of *Proceedings of Machine Learning Research*, pages 412–419. PMLR, 2007.

[Smi16] Mitch Smith. In Wisconsin, a Backlash Against Using Data to Foretell Defendants' Futures. *New York Times*, June 2016.

[SR15] Marcel Simon and Erik Rodner. Neural Activation Constellations: Unsupervised Part Model Discovery with Convolutional Networks. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1143–1151, 2015.

[SSP03] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition (ICDAR), Volume 2*, 2003.

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

[STY17] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70 of *Proceedings of Machine Learning Research*, pages 3319–3328. PMLR, 2017.

[SVZ14] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency

Maps. In *Workshop at the 2nd International Conference on Learning Representations (ICLR Workshop)*, 2014.

[SZ03] Josef Sivic and Andrew Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision (ICCV)*, page 1470. IEEE, 2003.

[SZ15] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proceedings of the 3rd International Conference on Learning Representations (ICLR)*, 2015.

[Tha07] Fadi Thabtah. A review of associative classification mining. *The Knowledge Engineering Review*, 22(01):37–65, 2007.

[UR16] Berk Ustun and Cynthia Rudin. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning*, 102(3):349–391, 2016.

[UR17] Berk Ustun and Cynthia Rudin. Optimized risk scores. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

[UVDSGS13] Jasper R.R. Uijlings, Koen E.A. Van De Sande, Theo Gevers, and Arnold W.M. Smeulders. Selective Search for Object Recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[vdOKK16] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel Recurrent Neural Networks. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pages 1747–1756, 2016.

[WBW+11] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[Wes17] Eric Westervelt. Did A Bail Reform Algorithm Contribute To This San Francisco Man's Murder? *National Public Radio, Law*, August 2017.

[Wex17] Rebecca Wexler. When a Computer Program Keeps You in Jail: How Computers are Harming Criminal Justice. *New York Times*, June 2017.

[WR15] Fulton Wang and Cynthia Rudin. Falling Rule Lists. In *Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015.

[WS09] Kilian Q. Weinberger and Lawrence K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

[WSS+15] Dequan Wang, Zhiqiang Shen, Jie Shao, Wei Zhang, Xiangyang Xue, and Zheng Zhang. Multiple Granularity Descriptors for Fine-grained Categorization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2399–2406, 2015.

[WT17] Chenyue Wu and Esteban G. Tabak. Prototypal Analysis and Prototypal Regression. *arXiv preprint arXiv:1701.08916*, 2017.

[XXY+15] Tianjun Xiao, Yichong Xu, Kuiyuan Yang, Jiaxing Zhang, Yuxin Peng, and Zheng Zhang. The Application of Two-Level Attention Models in Deep Convolutional Neural Network for Fine-grained Image Classification. In *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on*, pages 842–850. IEEE, 2015.

[YCFL15] Jason Yosinski, Jeff Clune, Thomas Fuchs, and Hod Lipson. Understanding Neural Networks through Deep Visualization. In *Deep Learning Workshop at the 32nd International Conference on Machine Learning (ICML)*, 2015.

[YRS17] Hongyu Yang, Cynthia Rudin, and Margo Seltzer. Scalable Bayesian Rule Lists. In *International Conference on Machine Learning*, pages 3921–3930, 2017.

[ZDGD14] Ning Zhang, Jeff Donahue, Ross Girshick, and Trevor Darrell. Part-based R-CNNs for Fine-grained Category Detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 834–849. Springer, 2014.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 818–833, 2014.

[ZFML17] Heliang Zheng, Jianlong Fu, Tao Mei, and Jiebo Luo. Learning Multi-Attention Convolutional Neural Network for Fine-Grained Image Recognition. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 5209–5217, 2017.

[ZKL+16] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning Deep Features for Discriminative Localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929. IEEE, 2016.

[ZSBT18] Bolei Zhou, Yiyou Sun, David Bau, and Antonio Torralba. Interpretable Basis Decomposition for Visual Explanation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 119–134, 2018.

[ZWZ18] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[ZXE+16] Han Zhang, Tao Xu, Mohamed Elhoseiny, Xiaolei Huang, Shaoting Zhang, Ahmed Elgammal, and Dimitris Metaxas. SPDA-CNN: Unifying Semantic Part Detection and Abstraction for Fine-grained Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1143–1152, 2016.

Biography

Chaofan Chen attended the University of Chicago and graduated with a Bachelor of Science degree in Mathematics (with Honors). He began doctoral studies in computer science at Duke University in 2014. He was awarded the Outstanding Ph.D. Preliminary Exam Award in 2018 and the Outstanding Research Initiation Project Award in 2017, by the Department of Computer Science at Duke University. He pursued his research in the area of interpretable machine learning under the direction of Professor Cynthia Rudin. The following publications are the results of his work conducted during doctoral studies:

[CLT+19] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This Looks Like That: Deep Learning for Interpretable Image Recognition. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[HCLR19] Peter Hase, Chaofan Chen, Oscar Li, and Cynthia Rudin. Interpretable Image Recognition with Hierarchical Prototypes. In *Proceedings of the Seventh AAAI Conference on Human Computation and Crowdsourcing (AAAI-HCOMP)*, 2019.

[CLR+18] Chaofan Chen, Kangcheng Lin, Cynthia Rudin, Yaron Shaposhnik, Sijia Wang, and Tong Wang. An Interpretable Model with Globally Consistent Explanations for Credit Risk. *NIPS 2018 Workshop on Challenges and Opportunities for AI in Financial Services: the Impact of Fairness, Explainability, Accuracy, and Privacy*, 2018.

[CR18] Chaofan Chen and Cynthia Rudin. An Optimization Approach to Learning Falling Rule Lists. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 84. PMLR, 2018.

[LLCR18] Oscar Li, Hao Liu, Chaofan Chen, and Cynthia Rudin. Deep Learning for Case-Based Reasoning through Prototypes: A Neural Network that Explains Its Predictions. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI)*, 2018.