Project Report
LSP-205

Adaptable Interpretable Machine Learning—Recursive Bayesian Rule Lists: FY17 Line-Supported Information, Computation, and Exploitation Program

J.K. Su

12 December 2017

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001.

This report is the result of studies performed at Lincoln Laboratory, a federally funded research and development center operated by Massachusetts Institute of Technology. This material is based on work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

Massachusetts Institute of Technology
Lincoln Laboratory

Adaptable Interpretable Machine Learning—Recursive Bayesian Rule Lists: FY17 Line-Supported Information, Computation, and Exploitation Program

J.K. Su
Group 45

Project Report LSP-205

12 December 2017

Lexington Massachusetts

This page intentionally left blank.

ABSTRACT

Three important factors that influence user trust in automation and machine learning (ML) algorithm are good performance, interpretability—the ability for users to understand how the automation reaches its recommendation—and adaptability—the ability for the automation to learn from new data and user feedback. This report describes work conducted under the Adaptable Interpretable Machine Learning (AIM) program, whose goal is to create ML algorithms that users can understand and that keep learning so users will trust them and actually use them.

This report derives Recursive Bayesian Rule Lists (RBRL), a group of supervised-classification algorithms that are both interpretable and adaptable. RBRL is based on Bayesian Rule Lists (BRL), which are interpretable decision-list classifiers that perform competitively with state-of-the-art, non-interpretable classifiers on many problems, and on an analogy between classifier adaptation and recursive Bayesian tracking (RBT).

RBRL has several appealing properties. First, it can accept user feedback on the feature combinations that were important to a user, rather than only accept feedback in the form of corrected labels. Second, RBRL uses an efficient adaptation procedure that only involves the new data and user feedback; it does not require accumulating all past data and feedback. Adaptation is non-iterative and highly parallelizable, so it can be thousands or millions of times faster than the Markov Chain Monte Carlo sampling that would be required to retrain BRL from scratch. Third, RBRL can include an explicit model for dataset shift to account for a changing environment or differences between the training and testing data and the data encountered in actual use. Fourth, RBRL can accept noisy truth labels from multiple users, provided that the users' truthing-error distributions can be characterized. Together, these properties make RBRL well-suited for interactive machine learning.

Two algorithmic forms of RBRL are presented: grid-based RBRL and particle-filter RBRL. Grid-based RBRL is optimal in theory but might be impractical to realize in practice. However, it includes the special case in which dataset shift does not occur and more data is continuously acquired; static-case RBRL provides an efficient algorithm for this situation. Particle-filter RBRL includes a generic algorithm, for which the optimal importance distribution is theoretically available but might be practically infeasible. The sampling importance resampling RBRL algorithm offers an efficient, suboptimal alternative.

The derivation of the RBT analogy and the descriptions of the different algorithms lay the foundation for RBRL implementations and experiments planned for the next year of the AIM program.

This page intentionally left blank.

ACKNOWLEDGMENTS

The author thanks Professor Cynthia Rudin, Duke University Computer Science, Electrical Engineering, and Statistical Sciences, for a stimulating and engaging collaboration. The Bayesian Rule Lists developed at Dr. Rudin's Prediction Analysis Lab provided the inspiration and foundation for this work, and her involvement and participation in this project fueled this pursuit.

The author is also indebted to Mr. Mark Kozar, whose enthusiasm for this project and development expertise is taking it from concept to implementation.

This work was conducted as part of the Adaptable Interpretable Machine Learning (AIM) project under MIT Lincoln Laboratory's Line-Supported Information, Computation, and Exploitation (ICE) Program. The author gratefully acknowledges the Technology Office and the ICE leads, Dr. Paul Monticciolo and Dr. Vijay Gadepally, for their support.

The author also thanks the leadership of the Informatics and Decision Support group, Dr. Timothy Dasey, Dr. Kajal Claypool, and Dr. Jason Thornton, for their encouragement of and interest in this project.

This page intentionally left blank.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

This page intentionally left blank.

LIST OF FIGURES

This page intentionally left blank.

LIST OF TABLES

This page intentionally left blank.

LIST OF ALGORITHMS

This page intentionally left blank.

1. INTRODUCTION

Trust is a key component in whether or not an automation capability will be accepted by users. If users lose trust in automation, they quickly abandon it or only use it begrudgingly [1]. This behavior can be succinctly expressed as: "No trust, no use" [2].

Several different factors influence trust, including a user's natural propensity to accept or reject automation; the user's level of attention, stress or fatigue; the automation's performance; agreement between the user's mental model and automation's behavior; the interface through which the user and automation interact, if at all; and the purpose of the automation and context in which it is used [3] [2]. Three important factors, which are emphasized in this work, are:

1. Providing an acceptably small level of errors or mistakes;
2. Providing the user with an understanding of how the automation works or providing a rationale for its recommendation;
3. Providing cues to the user and/or accepting feedback from the user to modify the automation's behavior or performance.

Machine learning (ML) is a class of algorithms that have become ubiquitous in automa-tion, autonomy, and decision-support systems. ML is often a critical component of automation—some might say that it is *the* critical component because it is the component that chooses the recommendation—so the factors that affect trust in automation apply to trust in ML as well.

ML is used for myriad purposes, such as self-driving vehicles, screening of applicants for jobs or college admissions, estimating a person's risk of recidivism, recommending medical treatments or predicting outcomes, recognizing speech, faces, or vehicles, identifying potentially dangerous cargo or individuals, detecting cyberintrusion, processing medical images, and selecting targeted advertisements. There also exists a wide variety of ML algorithms, such as supervised classification, clustering or unsupervised classification, estimation or regression, ranking, dimensionality reduction, and reinforcement learning. Typically, the users, domain experts, and consumers who use automation or are affected by it have little or no knowledge of ML. If they do not trust it, it is unlikely that they will adopt or embrace it.

This work is part of the *Adaptable Interpretable Machine Learning* (AIM) project, which is part of MIT Lincoln Laboratory's Line-Supported Information, Computation, and Exploitation Program. The large number of factors that affect trust in automation and the large number of ML algorithms mean it is impossible to address them all in this work. AIM focuses on the three factors enumerated above. AIM's goal is to **create ML algorithms that users can understand and that keep learning so *users will trust them and actually use them.***

This report concentrates on supervised classification (Sec. 2.1). In supervised classification, each data sample contains a set of feature values, and it is assumed to belong to exactly one of a finite number of predefined classes. Training is the process of learning a mapping between the feature values and the classes from a training data set. The learned mapping is the trained classifier.

When a new, unlabeled sample of features is presented, the mapping is applied to predict the class or label for the sample. Testing is the process of assessing the predictive accuracy of the trained classifier; it is conducted with a test data set that was excluded during training.

We are interested in developing classifiers that provide a high degree of predictive accuracy while also being both *interpretable* and *adaptable*. By "interpretable," we mean that a human can understand how the classifier makes its predictions, and by "adaptable," we mean that the algorithm is able to learn from new data and user feedback. We choose this focus because these factors are important to user trust, because supervised classification is one of the most prevalent forms of ML, and because the vast majority of top-performing supervised classification algorithms are neither interpretable nor adaptable.

1.1 INTERPRETABILITY

Before discussing interpretability further, we mention some matters related to a lack of interpretability. Most ML algorithms are *black-box* algorithms, which just return a predicted label or number. They are non-interpretable because there is no way for a user to understand how an erroneous—or correct—prediction was determined. Examples include support vector machines and neural networks, which are known for achieving high predictive accuracy but in a way that is very difficult to comprehend [4]. The interpretability of naïve Bayes and logistic regression is debatable.[1] Ensemble methods, which use a battery of classifiers and include boosting, bootstrap aggregating or bagging, and Random Forests^{TM}, can also be difficult to interpret [5], even if each individual classifier is interpretable.

With black-box algorithms, one can obtain an answer, but one does not receive an explanation. Regarding insight and understanding, this situation corresponds to "giving a man a fish" rather than "teaching a man to catch a fish."[2] This deficiency can lead to faulty reasoning and cause a conflict between the user's mental model of how he or she thinks an algorithm operates and how it actually operates.

Even an ideal ML algorithm will make mistakes, yet ML experts and practitioners often cannot explain how an algorithm makes its predictions or how it will respond to different inputs. Unexpected, unexplainable behavior from a black-box algorithm can rapidly erode trust. Moreover, it is usually not feasible to test the response of a black-box algorithm to all possible inputs, which can raise concerns about its reliability during ordinary operation or its susceptibility to malicious or adversarial exploits.

In contrast, an algorithm is interpretable if a human user can understand how it arrives at its recommendations. Such algorithms are sometimes called *transparent-box* or *white-box* algorithms to contrast them with black-box algorithms. Freitas [4] makes a number of excellent arguments

[1] Naïve Bayes uses a sum of log-likelihood functions, and logistic regression computes an inner product and passes it through an S-shaped nonlinearity to approximate a probability. Some users are comfortable with the linear component of these algorithms, but the log-likelihood and nonlinear components can lead to confusion and misunderstanding.

[2] One form of this proverb is: "Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime."

in favor of interpretable classification, and we discuss some important aspects of interpretability below.

Justification Interpretability might be preferred or required, particularly in cases where the consequences of prediction— whether correct or incorrect— are significant. It can be necessary to explain a medical recommendation; to protect against legal repercussions like claims of bias or discrimination; to justify law-enforcement actions like detainment, search, and seizure; to fulfill policy requirements of accountability or transparency; and to explain a decision to use force, including lethal force [6] [7].

Performance There is sometimes a misconception that interpretability necessarily causes a reduction in model expressiveness or predictive accuracy.[3] However, a less-complicated model is less prone to overfitting than a more-complicated one, and interpretability can have a regularizing effect. For many problems, simple models perform well [8], a multitude of equally good models exists, and an interpretable algorithm is frequently among them [9]. In other words, *interpretability is often free.*

Availability Some ML packages offer interpretable classifiers. These classifiers should be included during training and testing, in which multiple algorithms are evaluated against the same data and then down-selected. In this way, one can find out if interpretability comes for free, and if not, one can decide if a small reduction in performance is offset by the additional benefit of interpretability.

Vetting An interpretable algorithm can be compared against the experience and expertise of domain experts. This property can minimize or eliminate mismatch between the algorithm and a user's mental model of it. Even if the application will not involve having a user check the algorithm's prediction, interpretability can increase confidence in the vetting process before the algorithm is deployed operationally.

Insight An interpretable algorithm can allow users, domain experts, and researchers to gain new insight, such as identifying important features or relationships between features that should be investigated further. In this way, interpretability can be viewed as "teaching a man to catch a fish," rather than just "giving a man a fish." It can also reveal biases that might otherwise remain hidden within a black-box algorithm.

Training Because an interpretable algorithm can be understood by humans, it can be used to teach novices so that they improve their proficiency rather than becoming reliant on a device to tell them what to do. In a similar way, it can prevent skill atrophy because users can review the algorithm to reinforce their understanding of how it works.

Behavior It is often possible to examine an interpretable algorithm and determine exactly how it will respond to any combination of features; hence, there will not be any surprises about

[3] This mistaken belief might occur because many ML packages rely on greedy algorithms to train decision trees, a well-known type of interpretable classifier. These algorithms are prone to becoming trapped near a local optimum, so the trained decision tree can perform worse than other classifiers.

how it will behave in all situations. Resources need not be diverted to improving it in areas where it is already strong. Knowledge of its weaknesses can indicate where additional data or more discriminative features are required, and one can develop procedures or protocols to protect against gaming the system.

Human calculation Users such as clinicians, first responders, and soldiers are already trained to follow prescribed procedures in certain situations. Examples include the $CHADS_2$ score for predicting stroke risk [10] and the flowchart for triaging individuals in a mass-casualty situation [11]. These procedures are often hand-crafted by humans rather than driven by data. An interpretable algorithm can use data to generate a simple, compact procedure that users can follow and that achieves greater performance than the hand-crafted approach [12] [13]. In other words, the procedure can be learned by a machine, but it will be calculated or carried out by a human.

These *machine-learned, human-calculated* algorithms can be deployed in cases where machine-calculated algorithms cannot. The resulting procedure can be memorized during training or printed on a card [14]. Such interpretable procedures can be used in austere, field-forward situations without requiring the additional bulk and weight of a computer, electronic device, or battery, and they can be performed without diverting the user's attention to a screen or other interface. Finally, these procedures do not risk electronic detection, either due to inadvertent RF emissions or connection to a communications network.

1.2 ADAPTABILITY

One of the fundamental assumptions in ML is that one has a set of training and testing data that is representative of the data that will be encountered in practice. This assumption can be violated for a number of reasons, such as geographic or demographic differences; sensor degradation or substitution; seasonal or environmental changes; biological evolution or mutation; or a change in an adversary's strategy or tactics. *Dataset shift* is the general term used when the ideal assumption of representativeness is not satisfied [15] [16].

When dataset shift occurs, an ML algorithm that does not adapt becomes brittle: It does not adjust to new conditions and repeats mistakes. Its performance degrades and can become unacceptable. When degraded performance is combined with a lack of interpretability, trust in the algorithm can decline precipitously. Users may stop using it, and they may also become skeptical of other automation.

Standard adaptation techniques for supervised classification are *relevance feedback* and *active learning*. In relevance feedback [17], the user indicates which algorithm predictions are correct and which are incorrect, and the algorithm is retrained. In active learning, an additional algorithm selects unlabeled samples that will help the classification algorithm learn most rapidly, and it prompts the user to provide labels for them [18] [19].

In both of these techniques, the user indicates the correct labels for additional data samples. There is no way for the user to provide feedback about which features led him or her to select a label. Additionally, adaptation often amounts to adding the newly-labeled samples to the training data

and retraining the classifier from scratch on all of the accumulated training data. Consequently, an ever-increasing amount of storage and computation is required.

1.3 OTHER APPLICATIONS

There are certainly applications for which interpretability and adaptability are not important. We mention them, but they are not the point of this work.

In some situations, such as collision avoidance, the user must follow a defined response or take a particular action based on an algorithm's prediction. There is neither a need nor an opportunity for the user to check the prediction or the reason behind it. In other cases, such as detecting manufacturing defects, automation replaces the human user entirely. Sometimes the data and features themselves are not naturally human-interpretable, and for some applications like targeted advertising or spam filtering, predictive accuracy is the only criterion of interest. Interpretability is not required here, and a black-box method is sufficient or called for.

For other situations, such as characterizing physical phenomena in a controlled environment, there is every reason to believe that the training and testing data are representative, so adaptation is unnecessary. Likewise, the rate of change might be so small as to obviate the need for adaptation. There are also cases where there are sufficient storage and computing resources to accumulate new data and retrain a classifier from scratch rather than adapt in a more efficient manner. In these cases, adaptability is either not needed at all or it can be accomplished by brute force.

1.4 RECURSIVE BAYESIAN RULE LISTS (RBRL)

This report describes and derives *Recursive Bayesian Rule Lists* (RBRL), which were discovered under AIM during FY17. RBRL is based on two components: *Bayesian Rule Lists* (BRL), which are interpretable decision-list classifiers that were published by Letham, Rudin, McCormick, and Madigan [12] and perform competitively with state-of-the-art classifiers on many problems; and an analogy between classifier adaptation and *recursive Bayesian tracking* (RBT), which includes state-estimation algorithms like Kalman filtering [20] [21] and particle filtering [22] [23].

Some key properties of RBRL are briefly listed here. First, RBRL can *accept user feedback on features and labels*, rather than on labels alone. Second, RBRL enables *efficient adaptation* that only involves the new data and user feedback and does not require accumulating all of the past data and feedback. The adaptation procedure is non-iterative and highly parallelizable. Third, RBRL can include an explicit model for *dataset shift*. Fourth, RBRL can accept *noisy truth labels from multiple users*, provided that the users' truthing-error distributions can be characterized. Finally, the combination of these properties make RBRL well-suited for *interactive machine learning*.[4]

The remainder of this report consists of the following. Sec. 2 provides background on supervised classification and on elements of BRL: frequent pattern mining, decision lists, association rules, and rule lists. Sec. 3 reviews BRL, and Sec. 4 derives the analogy between BRL adaptation and RBT. Sec. 5 and Sec. 6 describe some algorithms for implementing RBRL. Sec. 7 provides a

[4] Credit goes to Cynthia Rudin for this observation.

summary. Some reference material appears in the appendices. App. A describes certain random variables (RVs) used in BRL and RBRL, App. B contains derivations from BRL, and App. C reviews RBT.

2. BACKGROUND

This section presents some background material. We illustrate many of the ideas in this section using a version of the *Titanic* data set, which contains information about the 2,201 people aboard the ill-fated luxury ship. The data set is a 2,201-by-4 table, where each row contains a person's cabin (first-class, second-class, third-class, or crew), adulthood status (child or adult), gender (female or male), and whether or not the person survived the tragic accident; there were 711 survivors. The training set is a subset of this data that contains 1,761 rows and includes 582 survivors.

2.1 SUPERVISED CLASSIFICATION

This report assumes that the reader is familiar with *supervised classification*, so we present only a quick review. We have a finite-dimensional feature space $\mathcal{X}$ and a set $\mathcal{Y}$ of L classes or *labels*, where L is an integer greater than or equal to two. For the *Titanic* example, the *features* are cabin, adulthood, and gender, so we have a three-dimensional feature space $\mathcal{X} = \{\text{first-class, second-class, third-class, crew}\} \times \{\text{child, adult}\} \times \{\text{female, male}\}$. The labels are "died" and "survived," so the label set $\mathcal{Y} = \{\text{died, survived}\}$. Of course, the labels can also be represented as a discrete set of integers such as $\mathcal{L} = \{1, 2, \ldots, L\}$. *Binary classification* corresponds to $L = 2$, and *multi-class classification* to $L > 2$.

We wish to learn a mapping $g : \mathcal{X} \to \mathcal{Y}$ from the feature space to the set of labels. A *classifier* is an assumed form of the mapping g. If we are presented with a *feature vector* $\tilde{x} \in \mathcal{X}$ but do not know the associated *correct label* $\tilde{y} \in \mathcal{Y}$, then we can use the learned mapping g to compute a *predicted label* $\hat{y} = g(\tilde{x})$. If the correct label $\tilde{y}$ becomes available and $\hat{y} = \tilde{y}$, then the predicted label is correct; otherwise, it is incorrect.

The data is partitioned into a training set and test set. *Classifier training* is the process of learning the mapping from the training set, and the learned mapping is the *trained classifier*. *Classifier testing* applies the trained classifier to the feature vectors in the test set and compares its predicted labels against the correct labels in the test set to measure predictive accuracy. The training and test sets are disjoint to eliminate the possibility of cheating during training. *Cross-validation* is often applied to characterize performance variability due to different choices of the training and test partitions.

2.2 FREQUENT PATTERN MINING

This section describes some terms and concepts in frequent pattern mining.

2.2.1 Features, Feature Values, and Patterns

While the notion of a "feature" in classification is generally understood, we will make some distinctions between the set of values and a specific value. A *feature* is one of the dimensions of $\mathcal{X}$; it includes all possible values for that dimension. In our example, cabin and gender are two features: cabin = {first-class, second-class, third-class, crew}, and gender = {female, male}.

Next, a *feature value* is one of the values that a feature can take. For the example, one gender feature value is female, and the other is male.

Finally, a *pattern* is a combination of feature values for a subset of the dimensions of $\mathcal{X}$. Examples are "cabin = second-class," and "adulthood = child and gender = male." When there is no confusion, we drop the feature and just provide the feature values as a tuple, so (second-class) and (child, male) are equivalent forms of these example patterns. We refer to the length or dimensionality of a pattern as its *cardinality*, so $\emptyset$ has cardinality zero, (male) and (crew) each have cardinality one, and (third-class, child, male) has cardinality three.

As another example, suppose that the feature vector x = (second-class, adult, female). The first "feature" is cabin, which refers to the first dimension of $\mathcal{X}$ but not the value that $x(1)$ takes. The "feature value" $x(1)$ is second-class. The vector x contains "patterns" such as (female) and (adult) of cardinality one, (adult, female) and (second-class, adult) of cardinality two, and (second-class, adult, female) of cardinality three. It does not contain the patterns (male) or (crew, adult).

2.2.2 Frequent Pattern Mining

Frequent pattern mining (FPM) is a class of data-mining methods that explore a data set and extract combinations of feature values that occur often [24], [25], [26], [27]. Typically, one specifies the desired maximum pattern cardinality and minimum support (frequency of pattern occurrence).

For the *Titanic* training set, the pattern (first-class, adult) has cardinality two and occurs 254 times, so its support is 14.4%. The pattern (crew) occurs 705 times, yielding a support of 40.0%. Some other patterns are (crew, adult, male) with support 39.0% and (third-class, child) with support 3.7%. If the desired maximum cardinality and minimum support are 2 and 10%, respectively, then (first-class, adult) and (crew) would be returned, but (crew, adult, male) would be rejected because its cardinality exceeds 2, and (third-class, child) would be rejected because its support is below 10%.

2.3 DECISION TREES AND DECISION LISTS

This section provides some background on decision lists, which can be helpful in understanding the Bayesian Rule Lists used in this report.

2.3.1 Decision Trees

A *decision tree* is a tree structure that describes a sequence of tests applied to a sample $x \in \mathcal{X}$ to arrive at a final decision regarding x [28]. For classification, this decision is a predicted label $\hat{y}$, and such decision trees are often called "classification trees." Decision trees have several natural representations: tree, flowchart, and nested **if-then-else** statements. These representations make decision trees amenable to human interpretation. Indeed, they were initially conceived with interpretability in mind.

An example decision tree for the *Titanic* training set appears in Figure 1; its **if-then-else**-form is shown in Figure 2. It is a binary tree, in which each non-leaf node of the tree contains a

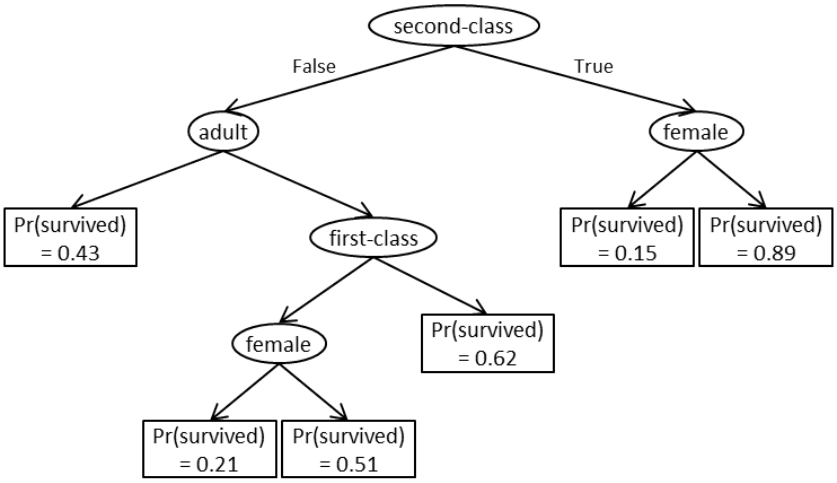

Figure 1. Graphical depiction of decision tree for Titanic *data*

boolean condition or test to apply to x. Typically, each condition tests a single feature in x. Each leaf node contains a decision about $\hat{y}$.

To classify x with a decision tree, processing starts at the root node and tests x against the root node's condition. Processing branches to the child node associated with the test outcome. If the next node has another condition, then testing and branching is performed at this node; the process continues until a leaf node is reached. When a leaf node is reached, a decision about $\hat{y}$ is made. For binary classification, only the probability or frequency for one label is needed, and it is compared against a threshold to select $\hat{y}$. All leaf nodes used the same threshold, which can be swept to generate a receiver operating characteristic (ROC) performance curve. For multi-class classification, one usually chooses the label with the highest probability.

For the *Titanic* example, suppose the threshold is 0.5, and $x =$ (first-class, adult, male). The outcome of the root node's test (**if** second-class) is false, so processing follows the false-branch to (**if** adult)-node. This node's test outcome is true, so processing follows the true-branch to the (**if** first-class)-node. The test outcome is true, so finally processing arrives at the leaf node with Pr(survived) = 0.62. Since $0.62 > 0.5$, the predicted label is $\hat{y} =$ "survived." Figure 3 shows the same **if-then-else**-form of the tree with a threshold of 0.5.

Decision-tree training is the process of constructing a tree with good predictive accuracy from a data set. It typically begins with a tree, such as a root-node-only tree, and applies splitting and pruning operations to modify the tree. Splitting takes a leaf node in the current tree, examines the training data that applies to the node, and chooses a condition for creating two new child nodes. Pruning takes a non-leaf node in the current tree and collapses its sub-trees to form a leaf node. Popular algorithms include CART [28] and C4.5 [29] or C5.0. Unfortunately, exploring all possible splits is computationally intractable, and optimal decision-tree training has been shown to be NP-complete [30]. Greedy algorithms and heuristics are used in practice, and they can become trapped near a local optimum. Consequently, the predictive accuracy of classification trees can lag

```
if second-class then
    if female then
        Pr(survived) = 0.89
    else
        Pr(survived) = 0.15
    end if
else
    if adult then
        if first-class then
            Pr(survived) = 0.62
        else
            if female then
                Pr(survived) = 0.51
            else
                Pr(survived) = 0.21
            end if
        end if
    else
        Pr(survived) = 0.43
    end if
end if
```

Figure 2. ***if-then-else****-form of decision tree for* Titanic *data*

```
if second-class then
    if female then
        ŷ = survived
    else
        ŷ = died
    end if
else
    if adult then
        if first-class then
            ŷ = survived
        else
            if female then
                ŷ = survived
            else
                ŷ = died
            end if
        end if
    else
        ŷ = died
    end if
end if
```

Figure 3. Decision tree from Figure 2 with a threshold of 0.5.

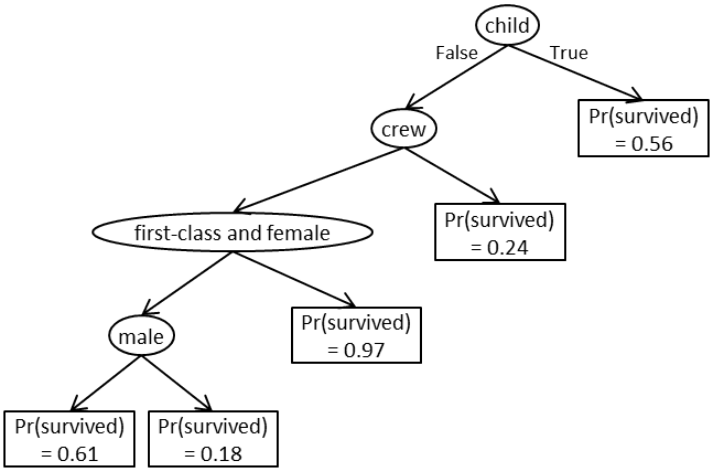

Figure 4. One-sided tree graphical depiction of decision list for Titanic *data*

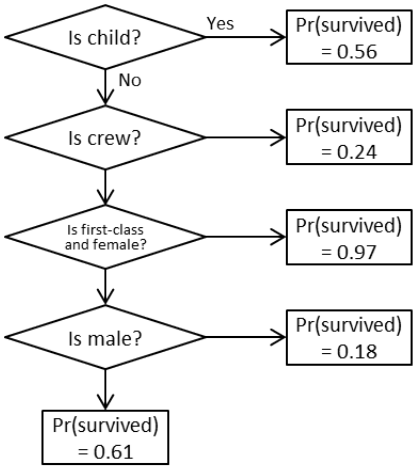

Figure 5. Flowchart for decision list for Titanic *data*

that of other state-of-the-art classifiers. Increasing tree depth can sometimes improve performance, but the tree can become much harder to interpret.

2.3.2 Decision Lists

A *decision list* [31] is a special kind of decision tree, namely a one-sided tree, such as the example in Figure 4. It can also be depicted as a one-sided flowchart (Figure 5), as **if-then-else**-statements with nesting only under the **else**-block (Figure 6), or as an **if-then-elseif**-statement with multiple **elseif**-blocks and without any nesting (Figure 7). The one-sided nature or absence of nesting can greatly simplify interpretation, provided that the conditions are not too complicated. Sec. 2.4 describes another representation, namely an ordered list of antecedents and consequents.

Any decision tree can also be mapped into a decision list by tracing the tree in reverse order from each leaf node to create a sequence of un-nested conditions. As an example, the decision tree from Figure 2 can also be written as the decision list in Figure 8. However, if the decision tree has several deep sub-trees, then the conditions in the resulting list can become difficult to comprehend.

2.4 ASSOCIATION RULES AND RULE LISTS

An *association rule*, written $a \rightarrow b$, is an implication between an *antecedent* a and the *consequent* b. Association rules were originally introduced in the context of data mining [24] [25], but in this report, it suffices to consider them in the context of decision-list classification. Here, the *antecedent* a corresponds to the condition and the *consequent* b to the action taken should the condition be satisfied. That is, $a \rightarrow b$ corresponds to "**if** a **then** b."

```
if child then
    Pr(survived) = 0.56
else
    if crew then
        Pr(survived) = 0.24
    else
        if first-class and female then
            Pr(survived) = 0.97
        else
            if male then
                Pr(survived) = 0.18
            else
                Pr(survived) = 0.61
            end if
        end if
    end if
end if
```

*Figure 6. Example decision list, with nesting only under **else**-blocks, for* Titanic *data*

```
if child then
    Pr(survived) = 0.56
else if crew then
    Pr(survived) = 0.24
else if first-class and female then
    Pr(survived) = 0.97
else if male then
    Pr(survived) = 0.18
else
    Pr(survived) = 0.61
end if
```

*Figure 7. Example decision list, as an **if-then-elseif**-statement without nesting, for* Titanic *data*

```
if second-class and female then
    Pr(survived) = 0.89
else if second-class then
    Pr(survived) = 0.15
else if adult and first-class then
    Pr(survived) = 0.62
else if adult and not first-class and female then
    Pr(survived) = 0.51
else if adult and not first-class then
    Pr(survived) = 0.21
else
    Pr(survived) = 0.43
end if
```

Figure 8. Example decision list created from the decision tree in Figure 2

A decision list can also be expressed as an ordered list of association rules. For Figure 7, the example decision list is ((child) $\rightarrow$ Pr(survived) = 0.56, (crew) $\rightarrow$ 0.24, (first-class, female) $\rightarrow$ 0.97, (male) $\rightarrow$ 0.18, $\emptyset \rightarrow$ 0.61), where $\emptyset \rightarrow 0.61$ is the final **else**-block, also known as the *default rule*.

In contrast, a *rule list* is an ordered list of antecedents only; *it does not contain any consequents* [12]. For the example in Figure 7, the rule list is ((child), (crew), (first-class, female), (male)); the final **else**-block does not have an antecedent and does not appear at the end of the list. Rule lists are of interest because BRL is based on a model for the distribution of rule lists. This model is combined with training data to learn a decision list for classification.

Finally, the sparsity of a decision list or rule list can be described by its *length*—i.e., number of antecedents—and the *average cardinality* of its antecedents. The example rule list of Figure 7 has a length of 4 and an average cardinality of 1.25.

This page intentionally left blank.

3. REVIEW OF BAYESIAN RULE LISTS

Bayesian Rule Lists (BRL), recently published by Letham, Rudin, McCormick, and Madigan [12], are decision-list classifiers that, for many data sets, demonstrate performance competitive with state-of-the-art, non-interpretable classifiers. A review of the main technical aspects of BRL appears here.

BRL cleverly combines a number of ideas during training. First, it uses FPM to find patterns that merit consideration as test conditions in a decision list. Second, it limits the training search space to ordered lists built from these patterns, rather than all possible splits. Third, it adopts a Bayesian approach—hence the name—in which the prior favors rule lists of a desired sparsity, and the likelihood relates the rule list to the training features and labels. Fourth, it employs Markov Chain Monte Carlo (MCMC) methods to sample from the rule list posterior distribution. Finally, it selects an optimal rule list from among the generated sample rule lists.

Hence, BRL learns a decision list, which is a restricted type of decision tree. Fundamentally, optimal decision-tree training remains NP-complete. BRL does not change this fact, but it avoids the combinatorial explosion in conventional decision-tree training and achieves a powerful balance between predictive accuracy, sparsity and interpretability, and tractable computation.[5]

3.1 OVERVIEW

Figure 9 shows that BRL training consists of two main components: frequent pattern mining and rule list optimization, where the latter consists of MCMC sampling followed by rule list selection. Let M be the size of the training set $(\mathbf{x}, \mathbf{y})$, where $\mathbf{x} = \{x_1, \ldots, x_M\}$ is the set of training feature vectors and $\mathbf{y} = \{y_1, \ldots, y_M\}$ is the corresponding set of training labels.

The pre-mined patterns from FPM comprise the set of antecedents, which provides a number of benefits. First, it greatly limits the training search space. Second, it means that computational complexity scales with the number of patterns rather than with the feature dimensionality. Third, the maximum pattern cardinality will usually be greater than one, so splits are not restricted to a single feature dimension.

The Bayesian formulation means that the worthiness of a candidate rule list is expressed as a posterior probability. The posterior combines the prior, which quantifies the rule list's agreement with the desired sparsity hyperparameters, and the likelihood, which quantifies the rule list's agreement with the observed training data.

The use of MCMC sampling allows stochastic exploration of the entire space $\mathbf{D}$, the set of all ordered lists that can be formed from $\mathcal{A}$. It eliminates the need for greedy algorithms, which are susceptible to becoming trapped in local optima. Sampling from the rule list posterior distribution generates candidate rule lists along with their probability or worthiness. Highly worthy rule lists

[5] A natural question is whether the BRL creators have considered lifting the decision-list restriction and allowing nested if-then statements. They have done so, but they did not find an improvement in predictive accuracy, and nesting can make it more difficult to interpret the resulting classifiers. For these reasons, they focused on decision lists.

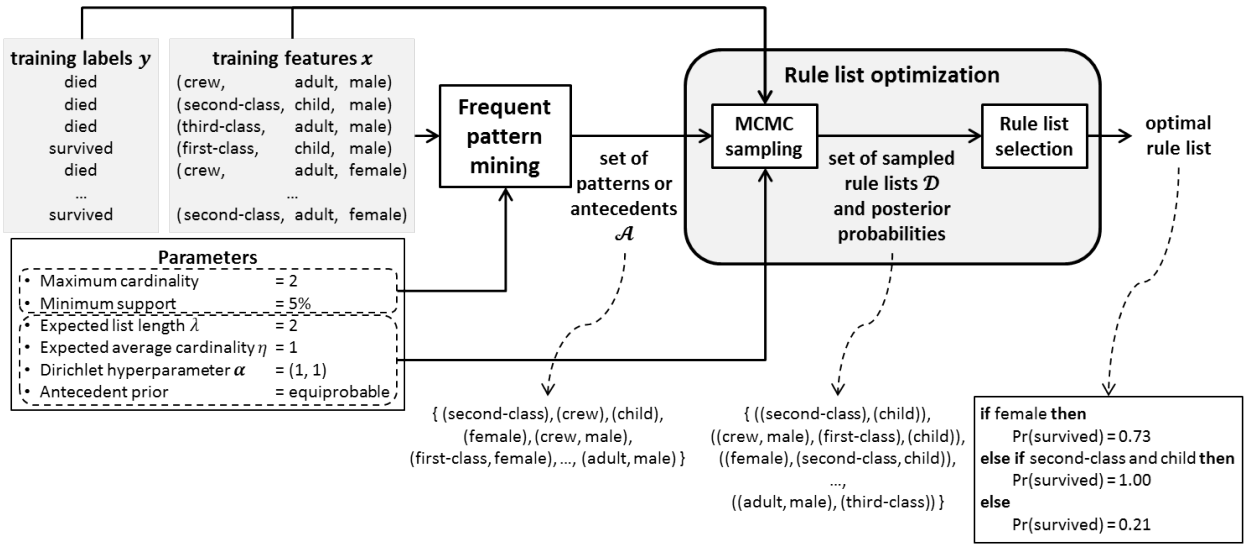

Figure 9. Block diagram of Bayesian Rule List training

also have a greater chance of being sampled compared to less worthy ones. Choosing a rule list from the generated samples is then a straightforward matter.

3.2 FREQUENT PATTERN MINING

FPM is applied only to the training *features* $\mathbf{x}$; the training labels $\mathbf{y}$ are not considered by this component. The class of FPM algorithms is known as "frequent itemset mining." The particular implementation used in [12] is FP-Growth [26], but algorithms such as Apriori [25] and Eclat [27] would return identical results.

One specifies the desired maximum cardinality and minimum support of a pattern, and the algorithm searches the training features $\mathbf{x}$ and returns the set of frequent patterns that meet these criteria. The mined patterns provide the antecedents in a rule list, so the set is called the *antecedent set* and is denoted by $\mathcal{A}$. The number of antecedents is $|\mathcal{A}|$.

Also, let $\mathbf{D}$ be the set of possible rule lists formed from $\mathcal{A}$. The number of rule lists of length m is $|\mathcal{A}|!/(|\mathcal{A}|-m)!$, for $m = 0, 1, \ldots, |\mathcal{A}|$, so the total number of rule lists is $|\mathbf{D}| = \sum_{m=0}^{|\mathcal{A}|} \frac{|\mathcal{A}|!}{(|\mathcal{A}|-m)!} \approx (|\mathcal{A}|!) \cdot e$. Even for modest values of $|\mathcal{A}|$, this number is enormous, making exhaustive search intractable.

3.3 ASSOCIATION RULES

BRL uses the language of association rules (Section 2.4). The antecedents are the pre-mined patterns in $\mathcal{A}$ produced by FPM, and the consequents are assumed to follow a Dirichlet-multinomial distribution. That is, $a \in \mathcal{A}$, and $b = \{y|\boldsymbol{\theta} \sim \text{Multinomial}(\boldsymbol{\theta}),\ \boldsymbol{\theta}|\boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha})\}$, where the multinomial parameter vector $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_L)$ and the Dirichlet hyperparameter vector $\boldsymbol{\alpha} =$

$(\alpha_1, \ldots, \alpha_L)$, $\alpha_\ell > 0$, $\forall \ell$. Hence, $a \to b$ reads as "a implies that the label y has a multinomial distribution with parameter $\boldsymbol{\theta}$, and $\boldsymbol{\theta}$ has a Dirichlet distribution with hyperparameter $\boldsymbol{\alpha}$."[6] App. A contains a brief review.

The Dirichlet-multinomial distribution is a compound distribution with some convenient properties [32]. First, the hyperparameter vector $\boldsymbol{\alpha}$ has a pseudocount interpretation: Each element α_ℓ can be interpreted as if α_ℓ examples of label ℓ have already been observed before any data is observed. Typically, $\alpha_\ell = 1$, $\forall \ell$, which results in $\boldsymbol{\theta}|\boldsymbol{\alpha}$ having a uniform distribution.

Second, the Dirichlet-multinomial distribution is a conjugate prior distribution, so it has a closed-form posterior update. For the observed data $(\mathbf{x}, \mathbf{y})$, let $N_\ell(\mathbf{x}, \mathbf{y})$ denote the number of times that $y_m = \ell$ when x_m satisfies the antecedent a. Then the posterior distribution of $\boldsymbol{\theta}$ is just another Dirichlet-multinomial RV, but with hyperparameter equal to $\boldsymbol{\alpha} + \mathbf{N}(\mathbf{x}, \mathbf{y})$, where $\mathbf{N}(\mathbf{x}, \mathbf{y}) = (N_1(\mathbf{x}, \mathbf{y}), \ldots, N_L(\mathbf{x}, \mathbf{y}))$. That is, $\boldsymbol{\theta}|\mathbf{x}, \mathbf{y}, \boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha} + \mathbf{N}(\mathbf{x}, \mathbf{y}))$ [see App. A.3.2]. Explicit computation of Bayes' rule and numerical integration are unnecessary to obtain the posterior.

3.4 GENERATIVE MODEL

BRL uses a generative model to enable MCMC rule list sampling. The model has three user-selectable hyperparameters: $\lambda > 0$, the expected rule list length; $\eta > 0$, the expected antecedent cardinality; and $\boldsymbol{\alpha}$, the Dirichlet hyperparameter vector for the labels. The hyperparameters λ and η control the sparsity of the desired rule list: λ specifies the desired list length, and η specifies the desired average cardinality.

This report contains a large amount of notation and indexing. Table 1 summarizes the indexes, and Table 2 summarizes elements that remain constant. Most of the entries appear in this discussion of BRL, but a few— marked with an asterisk— will be used later in the development of RBRL. Table 3 lists elements in BRL that will become time-varying in RBRL, where they appear again in Table 4.

Let the elements of $\mathcal{A}$ be indexed from 1 to $|\mathcal{A}|$, so $\mathcal{A} = \{a_1, a_1, \ldots, a_{|\mathcal{A}|}\}$, and let $a_0 \equiv \emptyset$ be reserved for the default rule. A rule list d is an ordered list of antecedents, so we express it as $d = (a_{k(1)}, a_{k(2)}, \ldots, a_{k(|d|)})$, where $k(j)$ is the index of the corresponding antecedent in $\mathcal{A}$, and the indexes are distinct: $k(j) \neq k(j')$, $j \neq j'$.[7] The default rule a_0 is not included in d, but for consistency, let $k(0) = 0$.

We also index the Dirichlet-multinomial consequents and define $\boldsymbol{\theta}_k$ and $\boldsymbol{\alpha}_k = \boldsymbol{\alpha}$, $k = 0, 1, \ldots, |\mathcal{A}|$. Let $\vec{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}_0, \ldots, \boldsymbol{\alpha}_{|\mathcal{A}|})$ be the vector of copies of the Dirichlet hyperparameters. We include an optional Dirichlet hyperparameter $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{|\mathcal{A}|})$ that can be used to encourage or

[6] Since y corresponds to a single trial, the Multinomial($\boldsymbol{\theta}$) distribution reduces to a Categorical($\boldsymbol{\theta}$) distribution, but these terms are often used without distinction, and we adhere to the "multinomial'' convention used by Letham *et al.* in [12].

[7] The rule list could also be represented as $(k(1), k(2), \ldots, k(|d|))$, an ordered list of antecedent indexes, but the previous form is more consistent with the notation of Letham *et al.*

TABLE 1

Indexing Conventions. An asterisk indicates an index used for Recursive Bayesian Rule Lists.

Symbol	Description
i	*index of rule list within a set of rule lists
j	index of antecedent in a rule list
$j(d, x)$	index of the first antecedent in rule list d that captures feature vector x
k	index of antecedent in antecedent set $\mathcal{A}$
$k(j(d, x))$	index of antecedent in antecedent set $\mathcal{A}$ that corresponds to the first antecedent in rule list d that captures feature vector x
ℓ	label index
m	sample index
n	*current time index
t	*truther index

discourage different antecedents in $\mathcal{A}$. A diagram using plate notation [33] [32] appears in Figure 10 for M samples.[8]

The model is constructed as follows.

1. The list length $|d|$ is sampled from a truncated Poisson distribution with parameter λ. It has support over $\{0, 1, \ldots, |\mathcal{A}|\}$ and mode close to λ, which encourages rule lists with length close to λ.

2. Given $|d|$, the antecedents are sequentially drawn from $\mathcal{A}$ without replacement (a_0 is not included). For $j = 1, \ldots, |d|$, let $a_{k(j)}$ be the jth antecedent in d and $c_{k(j)}$ be its cardinality. The following notation is used when $a_{k(j)}$ is being selected:

   - $a_{k(<j)} = \{a_{k(1)}, \ldots, a_{k(j-1)}\}$: the antecedents in d that precede the jth antecedent ($a_{k(j)}$) *in the rule list* d;
   - $c_{k(<j)} = \{c_{k(1)}, \ldots, c_{k(j-1)}\}$: the cardinalities of the preceding antecedents;
   - $\mathcal{A}(a_{k(<j)}) = \mathcal{A} \setminus a_{k(<j)}$: the set of remaining antecedents;
   - $\mathcal{C}(a_{k(<j)})$: the set of remaining cardinalities[9]—i.e., the cardinalities of the antecedents in $\mathcal{A}(a_{k(<j)})$, without duplication;

[8] Our notation introduces an additional layer of indexing—$k(j)$ rather than simply j—that does not appear in Letham *et al.* [12]. It is necessary because we will need to keep track of variables and parameters associated with different antecedents over time. We also make $|\mathcal{A}| + 1$ copies of $\boldsymbol{\alpha}$ because the hyperparameters for different antecedents might change over time. For the present review of BRL, one can safely replace $k(j)$ by j, $\alpha_{k(j),\ell}$ by α_ℓ, and $\vec{\boldsymbol{\alpha}}$ by $\boldsymbol{\alpha}$.

[9] $\mathcal{C}(a_{k(<j)})$ is equivalent to $R_{j-1}(c_{<j}, \mathcal{A})$ used by Letham *et al.* [12].

TABLE 2

Constant Elements. An asterisk indicates an element used for Recursive Bayesian Rule Lists.

Symbol	Type	Description
$\mathcal{A} = \{a_1, \ldots, a_{\lvert\mathcal{A}\rvert}\}$	set	antecedent set
a_k	antecedent	kth antecedent in $\mathcal{A}$ ($k \neq 0$)
$a_0 \equiv \emptyset$	antecedent	default antecedent
$\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_L)$	vector	user-specified Dirichlet parameters
α_ℓ	scalar	user-specified Dirichlet parameter for ℓth label
$\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{\lvert\mathcal{A}\rvert})$	vector	user-specified Dirichlet parameters for antecedents
β_k	scalar	user-specified Dirichlet parameter for kth antecedent in $\mathcal{A}$
$\mathbf{D}$	set	set of all possible rule lists from $\mathcal{A}$
$\mathcal{D} = \{d^{(1)}, \ldots, d^{(\lvert\mathcal{D}\rvert)}\}$	set	set of sampled rule lists
$d^{(i)}$	ordered list	ith rule list in $\mathcal{D}$
ζ	scalar	*process-noise parameter (Sec. 6.5.2)
κ	scalar	*exponential decay factor (Sec. 4.1)
λ	scalar	user-specified expected list length
η	scalar	user-specified expected average cardinality
L	scalar	number of labels or classes
$\mathcal{L} = \{1, \ldots, L\}$	set	set of labels
ν	scalar	*process-noise parameter (Sec. 6.5)
$N_\mathcal{T}$	scalar	*number of truthers (Sec. 4.1, Sec. 4.4.3)
$\mathcal{T} = \{1, \ldots, N_\mathcal{T}\}$	set	*set of truther indexes (Sec. 4.1, Sec. 4.4.3)

TABLE 3

Bayesian Rule List Elements

Symbol	Type	Description
$a_{k(j)}$	antecedent	jth antecedent in rule list d or $k(j)$th antecedent in antecedent set $\mathcal{A}$
$\vec{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}_0, \ldots, \boldsymbol{\alpha}_{\lvert\mathcal{A}\rvert})$	list of vectors	Dirichlet parameters for all antecedents
$\boldsymbol{\alpha}_k = (\alpha_{k,1}, \ldots, \alpha_{k,L})$	vector	Dirichlet parameters for kth antecedent
$\alpha_{k,\ell}$	scalar	Dirichlet parameter for kth antecedent, ℓth label
$d = (a_{k(1)}, \ldots, a_{k(\lvert d\rvert)})$	ordered list	rule list
$k(j)$	scalar	index into antecedent set $\mathcal{A}$ for jth antecedent in rule list d
M	scalar	number of samples
$\mathbf{N}_j(d, \mathbf{x}, \mathbf{y}) = (N_{j,1}(d, \mathbf{x}, \mathbf{y}), \ldots, N_{j,L}(d, \mathbf{x}, \mathbf{y}))$	vector	counts of all labels in $\mathbf{y}$ for samples in $\mathbf{x}$ captured by the jth antecedent in rule list d
$N_{j,\ell}(d, \mathbf{x}, \mathbf{y})$	scalar	count of ℓth label in $\mathbf{y}$ for samples in $\mathbf{x}$ captured by the jth antecedent in rule list d
$\vec{\boldsymbol{\theta}} = (\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_{\lvert\mathcal{A}\rvert})$	list of vectors	multinomial parameters for all consequents
$\boldsymbol{\theta}_k = (\theta_{k,1}, \ldots, \theta_{k,L})$	vector	multinomial parameters for kth consequent
$\theta_{k,\ell}$	scalar	multinomial parameter for kth consequent, ℓth label
$\mathbf{x} = (x_1, \ldots, x_M)$	list of vectors	all features
x_m	vector	features for mth sample
$\mathbf{y} = (y_1, \ldots, y_M)$	list of scalars	all labels
y_m	scalar	label for mth sample

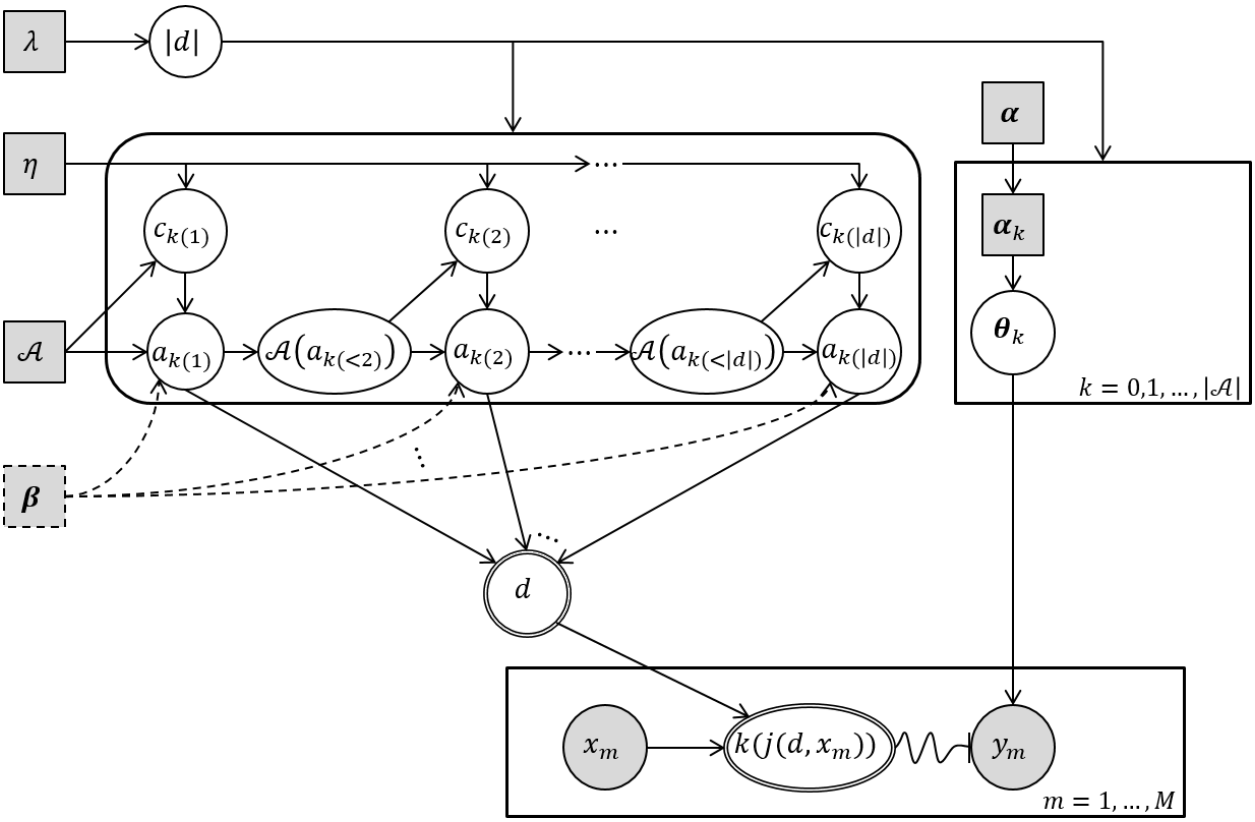

Figure 10. Generative model for Bayesian Rule Lists. Arrows show dependencies between variables. Squares indicate non-random variables, circles denote random entities, and double-outlined circles denote random entities that are completely determined by the variables they depend on. Shaded elements are known or observed, while unshaded ones are hidden or latent. The rectangles or plates indexed by k and m mean that the variables inside the plates are repeated as many times as indicated. The squiggly line indicates that $k(j(d, x_m))$ selects the index of the variable $\boldsymbol{\theta}_{k(j)}$ upon which y_m depends. The dashed lines for the parameter $\boldsymbol{\beta}$ denote an optional non-uniform prior over the antecedents $a_{k(1)}, \ldots, a_{k(|d|)}$.

- $\mathcal{A}(a_{k(<j)}, c) = \{a_k \in \mathcal{A}(a_{k(<j)}) : |a_k| = c\}$: the set of remaining antecedents of cardinality c;
- $\mathcal{K}(a_{k(<j)}, c) = \{k : a_k \in \mathcal{A}(a_{k(<j)}), |a_k| = c\}$: essentially the same set as $\mathcal{A}(a_{k(<j)}, c)$, except that it contains the antecedents' indexes rather than the antecedents themselves.

For $j = 1, 2, \ldots, |d|$, each antecedent $a_{k(j)}$ is sampled according to the following procedure.

(a) The cardinality $c_{k(j)}$ is drawn from a truncated Poisson distribution parameterized by η and with support over $\mathcal{C}(a_{k(<j)})$. The expected value of this distribution is close to η, so it favors antecedents with cardinality η. Figure 10 only shows dependence on $\mathcal{A}(a_{k(<j)})$ since it determines $\mathcal{C}(a_{k(<j)})$.

(b) Given $\mathcal{A}(a_{k(<j)})$ and $c_{k(j)}$, the antecedent $a_{k(j)}$ itself is distributed over $\mathcal{A}(a_{k(<j)}, c_{k(j)})$, the set of remaining antecedents of cardinality $c_{k(j)}$. We allow for the optional use of a Dirichlet-multinomial distribution parameterized by $\boldsymbol{\beta}$. The figure only shows dependence on $\mathcal{A}(a_{k(<j)})$ and $c_{k(j)}$ since they determine $\mathcal{A}(a_{k(<j)}, c_{k(j)})$.

3. For $k = 0, 1, \ldots, |\mathcal{A}|$, sample the multinomial parameter vector $\boldsymbol{\theta}_k \sim \text{Dirichlet}(\boldsymbol{\alpha}_k)$.

   Alternatively, since BRL only involves the antecedents *in the rule list* d, for each antecedent $a_{k(j)}$ in d, sample $\boldsymbol{\theta}_{k(j)} \sim \text{Dirichlet}(\boldsymbol{\alpha})$ only for $a_{k(j)} \in d$, and sample $\boldsymbol{\theta}_0 \sim \text{Dirichlet}(\boldsymbol{\alpha})$.

4. Antecedent $a_{k(j)} \in d$ is said to *capture* x_m if x_m contains the pattern of antecedent $a_{k(j)}$ and does not contain the patterns of the preceding antecedents $a_{k(1)}, a_{k(2)}, \ldots, a_{k(j-1)}$. Let $j(d, x)$ denote the index of the antecedent *in the rule list* d that captures x, so $k(j(d, x))$ is the index of the antecedent *in the antecedent set* $\mathcal{A}$; $j(d, x) = 0$ and $k(j(d, x)) = 0$ if none of the antecedents in d capture x_m. Then the consequent is $y_m \sim \text{Multinomial}(\boldsymbol{\theta}_{k(j(d,x_m))})$.

3.4.1 Prior

Based on the above model, the *rule list prior* is [cf . [12, Eq. (2.1)]]

$$p(d|\mathcal{A}, \lambda, \eta, \boldsymbol{\beta}) = \underbrace{p(|d||\mathcal{A}, \lambda)}_{\text{list length}} \prod_{j=1}^{|d|} \underbrace{p(c_{k(j)}|c_{k(<j)}, \mathcal{A}, \eta)}_{\substack{\text{cardinality of} \\ j\text{th antecedent}}} \underbrace{p(a_{k(j)}|a_{k(<j)}, c_{k(j)}, \mathcal{A}, \boldsymbol{\beta})}_{j\text{th antecedent}}$$

$$= \frac{\left(\lambda^{|d|}/|d|!\right)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)} \prod_{j=1}^{|d|} \frac{\left(\eta^{c_{k(j)}}/c_{k(j)}!\right)}{\sum_{c' \in \mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)} p(a_{k(j)}|a_{k(<j)}, c_{k(j)}, \mathcal{A}, \boldsymbol{\beta}). \quad (1)$$

As described above, the list-length term is a truncated Poisson with parameter λ. The product indexed by j corresponds to iteration over the antecedents in d. On each iteration, the cardinality term is a truncated Poisson distribution with parameter η and support restricted to the remaining cardinalities, and the antecedent term is a distribution over the remaining antecedents with the selected cardinality.

We let the antecedent term be a Dirichlet-multinomial distribution with hyperparameters chosen from $\{\beta_k : k \in \mathcal{K}(a_{k(<j)}, c_{k(j)})\}$:

$$p(a_{k(j)}|a_{k(<j)}, c_{k(j)}, \mathcal{A}, \boldsymbol{\beta}) = \frac{\Gamma(\sum_{k\in\mathcal{K}(a_{k(<j)},c_{k(j)})}\beta_k)}{\Gamma(1+\sum_{k\in\mathcal{K}(a_{k(<j)},c_{k(j)})}\beta_k)}\frac{1+\Gamma(\beta_{k(j)})}{\Gamma(\beta_{k(j)})}, \quad a_{k(j)} \in \mathcal{A}(a_{k(<j)}, c_{k(j)}). \tag{2}$$

Then (1) becomes

$$p(d|\mathcal{A}, \lambda, \eta, \boldsymbol{\beta}) = \frac{\left(\lambda^{|d|}/|d|!\right)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}\prod_{j=1}^{|d|}\frac{\left(\eta^{c_{k(j)}}/c_{k(j)}!\right)}{\sum_{c'\in\mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)}\frac{\Gamma(\sum_{k\in\mathcal{K}(a_{k(<j)},c_{k(j)})}\beta_k)}{\Gamma(1+\sum_{k\in\mathcal{K}(a_{k(<j)},c_{k(j)})}\beta_k)}\frac{1+\Gamma(\beta_{k(j)})}{\Gamma(\beta_{k(j)})}. \tag{3}$$

If $\beta_k = 1$, $\forall k$, the right-hand side of (2) becomes $|\mathcal{A}(a_{k(<j)}, c_{k(j)})|^{-1}$ for the eligible antecedents, which gives the uniform distribution used in [12, Eq. (2.2)]. In this case, the rule list prior is

$$p(d|\mathcal{A}, \lambda, \eta, \boldsymbol{\beta} = (1, \ldots, 1)) = \frac{\left(\lambda^{|d|}/|d|!\right)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}\prod_{j=1}^{|d|}\frac{\left(\eta^{c_{k(j)}}/c_{k(j)}!\right)}{\sum_{c'\in\mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)}\frac{1}{|\mathcal{A}(a_{k(<j)}, c_{k(j)})|} \tag{4}$$

$$\propto \frac{\left(\lambda^{|d|}/|d|!\right)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}\prod_{j=1}^{|d|}\frac{\left(\eta^{c_{k(j)}}/c_{k(j)}!\right)}{\sum_{c'\in\mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)}. \tag{5}$$

The rule list prior does not depend directly on the training data $(\mathbf{x}, \mathbf{y})$, although $\mathcal{A}$ is determined by mining $\mathbf{x}$. Rather, the prior favors sampling of rule lists that conform to the desired sparsity settings $\{\lambda, \eta\}$. Its form makes rule lists whose desired lengths and average cardinalities are close to λ and η more probable than rule lists whose characteristics differ substantially from λ or η.

3.4.2 Likelihood Function

The *rule list likelihood function* introduces the influence of the training data $(\mathbf{x}, \mathbf{y})$. It quantifies how likely a given rule list d and the features $\mathbf{x}$ are to have produced the labels $\mathbf{y}$. Given $(\mathbf{x}, \mathbf{y})$, let $N_{j,\ell}(d, \mathbf{x}, \mathbf{y})$ be the number of samples that are captured by antecedent $a_{k(j)}$ and have label ℓ, and let $\mathbf{N}_j(d, \mathbf{x}, \mathbf{y}) = (N_{j,1}(d, \mathbf{x}, \mathbf{y}), \ldots, \mathbf{N}_{j,L}(d, \mathbf{x}, \mathbf{y}))$.[10] Hence, the prior consequent for the jth rule is $\{y|\boldsymbol{\theta}_{k(j)} \sim \text{Multinomial}(\boldsymbol{\theta}_{k(j)}), \boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)} \sim \text{Dirichlet}(\boldsymbol{\alpha}_{k(j)})\}$, and the posterior consequent is $\{y|\boldsymbol{\theta}_{k(j)} \sim \text{Multinomial}(\boldsymbol{\theta}_{k(j)}), \boldsymbol{\theta}_{k(j)}|d, \mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}_{k(j)} \sim \text{Dirichlet}(\boldsymbol{\alpha}_{k(j)} + \mathbf{N}_j(d, \mathbf{x}, \mathbf{y}))\}$.

[10] Since d and j are given, there is no need to show the extra indexing layer $k(j)$ on N and $\mathbf{N}$. $N_{j,\ell}(d, \mathbf{x}, \mathbf{y})$ corresponds to $N_{j,\ell}$ in [12], and $\mathbf{N}_j(d, \mathbf{x}, \mathbf{y})$ corresponds to $\mathbf{N}_j$ in [12]. We emphasize the dependence on the rule list d and training data $(\mathbf{x}, \mathbf{y})$.

The likelihood function can be derived from the generative model; see [12] or App. B.1. The result is

$$p(\mathbf{y}|d,\mathbf{x},\vec{\boldsymbol{\alpha}}) \overset{(a)}{=} \prod_{j=0}^{|d|} \frac{\Gamma(\sum_{\ell=1}^{L}\alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L}N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})} \prod_{\ell=1}^{L} \frac{\Gamma(N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}{\Gamma(\alpha_{k(j),\ell})} \tag{6}$$

$$\propto \prod_{j=0}^{|d|} \frac{\prod_{\ell=1}^{L}\Gamma(N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L}N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}. \tag{7}$$

3.4.3 Posterior

The *rule list posterior* follows from Bayes' rule, which gives

$$\begin{aligned}
&p(d|\mathbf{x},\mathbf{y},\mathcal{A},\vec{\boldsymbol{\alpha}},\lambda,\eta,\boldsymbol{\beta}) \\
&\quad\propto p(\mathbf{y}|d,\mathbf{x},\mathcal{A},\vec{\boldsymbol{\alpha}},\lambda,\eta,\boldsymbol{\beta})p(d|\mathbf{x},\mathcal{A},\vec{\boldsymbol{\alpha}},\lambda,\eta,\boldsymbol{\beta}) \\
&\quad\overset{(a)}{\propto} p(\mathbf{y}|d,\mathbf{x},\vec{\boldsymbol{\alpha}})p(d|\mathcal{A},\lambda,\eta,\boldsymbol{\beta}) \\
&\quad\propto \left(\prod_{j=0}^{|d|}\frac{\prod_{\ell=1}^{L}\Gamma(N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L}N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}\right)\frac{\left(\lambda^{|d|}/|d|!\right)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}\prod_{j=1}^{|d|}\frac{\left(\eta^{c_{k(j)}}/c_{k(j)}!\right)}{\sum_{c'\in\mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)} \\
&\qquad\times\frac{\Gamma(\sum_{k\in\mathcal{K}(a_{k(<j)},c_{k(j)})}\beta_k)}{\Gamma(1+\sum_{k\in\mathcal{K}(a_{k(<j)},c_{k(j)})}\beta_k)}\frac{1+\Gamma(\beta_{k(j)})}{\Gamma(\beta_{k(j)})},
\end{aligned} \tag{8}$$

where (a) follows from the generative model. When $\boldsymbol{\beta}=(1,\ldots,1)$, it reduces to

$$\begin{aligned}
&p(d|\mathbf{x},\mathbf{y},\mathcal{A},\vec{\boldsymbol{\alpha}},\lambda,\eta,\boldsymbol{\beta}=(1,\ldots,1)) \\
&\qquad\propto\left(\prod_{j=0}^{|d|}\frac{\prod_{\ell=1}^{L}\Gamma(N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L}N_{j,\ell}(d,\mathbf{x},\mathbf{y})+\alpha_{k(j),\ell})}\right)\frac{\left(\lambda^{|d|}/|d|!\right)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}\prod_{j=1}^{|d|}\frac{\left(\eta^{c_{k(j)}}/c_{k(j)}!\right)}{\sum_{c'\in\mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)}.
\end{aligned} \tag{9}$$

The posterior describes how probable rule list d is, given the pre-mined antecedent set $\mathcal{A}$, the training data $(\mathbf{x},\mathbf{y})$, the sparsity settings $\{\lambda,\eta\}$, and the hyperparameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

3.5 MARKOV CHAIN MONTE CARLO RULE LIST SAMPLING

Markov chain Monte Carlo (MCMC) is a collection of methods for generating samples from a desired posterior distribution $p(d|\ldots)$ that cannot be directly sampled. One of the main MCMC methods is the Metropolis-Hastings algorithm, an iterative, stochastic sampling method [34].

Beginning at iteration $t=1$, the algorithm start with an initial sample d^1, which is usually randomly selected. On iteration $t+1$, it generates a new sample d^* $q(d^*|d^t)$, where $q(d^*|d^t)$ is called the *proposal distribution*. The sample d^* only depends upon d^t, the sample from the previous iteration. The algorithm then uses rejection sampling to determine whether to accept d^* and set

$d^{t+1} = d^*$ or to reject it and set $d^{t+1} = d^t$. The acceptance-rejection threshold is calculated from $q(d^*|d^t)$ and $p(d^*|\ldots)$. Finally, the algorithm repeats.

The procedure just described conducts a random walk over a Markov chain. If the proposal distribution $q(d^*|d^t)$ is chosen such that the chain is aperiodic and irreducible[11], then the chain will have $p(d|\ldots)$ as its unique equilibrium distribution. Hence, after a sufficient number of iterations, the chain will converge, and the subsequently generated samples will be distributed according to $p(d|\ldots)$. The required number of iterations is called the *burn-in period*, and different methods are available to test for convergence.

The Metropolis-Hastings algorithm has several appealing properties. First, although it is hard to sample from $p(d^*|\ldots)$, it is usually straightforward to calculate $p(d^*|\ldots)$ for the sample d^*. Second, $p(d^*|\ldots)$ only needs to be known up to a proportionality constant, which can be difficult to calculate for complicated distributions. Third, it is usually not difficult to construct a suitable proposal distribution to ensure that the chain has the necessary properties.

BRL uses the Metropolis-Hastings algorithm to sample from the rule list posterior $p(d|\mathbf{x}, \mathbf{y}, \mathcal{A}, \lambda, \eta)$. Three types of proposals are used [12]: *move* – exchange the positions of two antecedents already in d; *add* – select an antecedent from $\mathcal{A}$ that is not already in d and insert it into d; and *remove* – delete an antecedent from d. For the data sets considered in [12], the algorithm was run for 20,000, 50,000, or 100,000 iterations.

3.6 RULE LIST SELECTION AND POSTERIOR PREDICTIVE

Different classifiers can be chosen from $\mathbf{D}$. The rule list posterior is a probability distribution over $\mathbf{D}$, so choosing a classifier can be considered an estimation problem. Of course, in practice one uses $\mathcal{D}$, the set of sampled rule lists after burn-in. It is much smaller than $\mathbf{D}$ and serves as an approximation of $\mathbf{D}$.

3.6.1 Rule List Selection

In some Bayesian estimation problems, like minimum mean-square error estimation, the conditional posterior mean is the optimum estimate. However, it is not obvious how to define the mean of a collection of rule lists, so the conditional mean is not particularly useful. Another possible alternative is the *maximum a posteriori* (MAP) rule list, but the MAP rule list might be much shorter than the mean list length of the sampled rule lists in $\mathcal{D}$.

For these reasons, Letham *et al.* introduce *BRL-point*, which is a constrained MAP estimate. Namely, BRL-point is the most-probable rule list whose length and average cardinality agree with the mean length and mean average cardinality (rounded up or down) of the sampled rule lists. It is a single, human-readable rule list and the preferred point estimate in this report. Alg. 1 provides pseudocode; it returns BRL-point and its index, which will be useful for computing the posterior predictive.

[11] Roughly speaking, an aperiodic Markov chain does not contain any loops, and an irreducible Markov chain is a chain for which it is possible to get from any state to any other state.

Algorithm 1. *BRL-point. The inputs are a set of distinct rule lists and their corresponding posterior probabilities or non-negative weights.*

function $\left[\hat{d}, \hat{\imath}\right] \leftarrow$ BRL-Point$(\{d^{(i)}\}_{i=1}^{N}, \{w^{(i)}\}_{i=1}^{N})$
  function $\bar{c} \leftarrow$ AvgCard(d) ▷ Calculate average cardinality of a rule list
    if $d = \emptyset$ **then**
      $\bar{c} \leftarrow 0$
    else
      $\bar{c} \leftarrow |d|^{-1} \sum_{j=1}^{|d|} c_{k(j)}$
    end if
  end function
  $\bar{m} \leftarrow N^{-1} \sum_{i=1}^{N} |d^{(i)}|$ ▷ Mean length of rule lists
  $\bar{\bar{c}} \leftarrow N^{-1} \sum_{i=1}^{N}$ AvgCard$(d^{(i)})$ ▷ Mean average cardinality of rule lists
  $\mathcal{I} \leftarrow \{i : |d^{(i)}| \in \{\lfloor \bar{m} \rfloor, \lceil \bar{m} \rceil\}, \lfloor \bar{\bar{c}} \rfloor \leq$ AvgCard$(d^{(i)}) \leq \lceil \bar{\bar{c}} \rceil\}$
  $\hat{w} \leftarrow \max\{w^{(i)}\}_{i \in \mathcal{I}}$ ▷ Posterior probability or weight of BRL-point
  $\hat{\imath} \leftarrow$ index of $\hat{w}$ in $\{w^{(i)}\}_{i=1}^{N}$ ▷ Index of BRL-point
  $\hat{d} \leftarrow d^{(i)}$ ▷ BRL-point
end function

3.6.2 Posterior Predictive

Given a single rule list d, such as BRL-point, the *posterior predictive* can be applied to use d to classify a feature vector $\tilde{x}$ and predict the unknown label $\tilde{y}$. From [12, Sec. 2.7] or App. B.2, the posterior predictive is

$$p(\tilde{y} = \ell | \tilde{x}, d, \mathbf{x}, \mathbf{y}, \vec{\boldsymbol{\alpha}}) = \frac{\alpha_{k(j(d,\tilde{x})),\ell} + N_{j(d,\tilde{x}),\ell}(d, \mathbf{x}, \mathbf{y})}{\sum_{\ell'=1}^{L} \alpha_{k(j(d,\tilde{x})),\ell'} + N_{j(d,\tilde{x}),\ell'}(d, \mathbf{x}, \mathbf{y})}, \quad \ell = 1, \ldots, L. \tag{10}$$

This distribution gives the consequent for $a_{k(j(d,\tilde{x}))}$; hence, d can be expressed as a decision list like in Sec. 2.3.2.

A trivial algorithm for calculating the posterior predictives for all antecedents and labels appears in Alg. 2. It returns a $(|d| + 1) \times L$ array P, where $P_{j,\ell}$ contains the posterior predictive for the jth rule and ℓth label. This array can be used to implement a classifier. For example, in binary classification, $P_{j,1}$ can be compared to the decision threshold to determine the predicted label associated with the jth antecedent. For multi-class classification, the predicted label for the jth antecedent corresponds to $\arg\max_{\ell \in \mathcal{L}} P_{j,\ell}$.

Algorithm 2. *Calculation of posterior predictives for BRL point estimate. The optional input* $\mathbf{r}$ *is rule feedback that acts like additional pseudo-observations [cf. Sec. 4.4]. It defaults to the empty set if omitted.*

function $P \leftarrow$ TABULATEPOSTERIORPREDICTIVES$(d, \mathbf{x}, \mathbf{y}, \vec{\boldsymbol{\alpha}}, \mathbf{r} = \emptyset)$
  for $j \leftarrow 0 : |d|$ **do** ▷ Antecedents
    for $\ell \leftarrow 1 : L$ **do** ▷ Labels
      $P_{j,\ell} \leftarrow \frac{\alpha_{k(j),\ell} + N_{j,\ell}(d,\mathbf{x},\mathbf{y}) + N_{j,\ell}(d,\mathbf{r})}{\sum_{\ell'=1}^{L} \alpha_{k(j),\ell'} + N_{j,\ell'}(d,\mathbf{x},\mathbf{y}) + N_{j,\ell}(d,\mathbf{r})}$ ▷ Posterior predictive for point estimate (10)
    end for
  end for
end function

Another way to classify $\tilde{x}$ is to marginalize over all sampled rule lists. This classifier is called *BRL-post* and is given by [12, Sec. 2.7]

$$\begin{aligned} p(\tilde{y} = \ell | \tilde{x}, \mathbf{x}, \mathbf{y}, \mathcal{A}, \vec{\boldsymbol{\alpha}}, \lambda, \eta, \boldsymbol{\beta}) &= \sum_{d \in \mathcal{D}} p(\tilde{y} = \ell | \tilde{x}, d, \mathbf{x}, \mathbf{y}, \vec{\boldsymbol{\alpha}})\, p(d | \mathbf{x}, \mathbf{y}, \mathcal{A}, \vec{\boldsymbol{\alpha}}, \lambda, \eta, \boldsymbol{\beta}) \\ &= \sum_{d \in \mathcal{D}} \frac{\alpha_{k(j(d,\tilde{x})),\ell} + N_{j(d,\tilde{x}),\ell}(d, \mathbf{x}, \mathbf{y})}{\sum_{\ell'=1}^{L} \alpha_{k(j(d,\tilde{x})),\ell'} + N_{j(d,\tilde{x}),\ell'}(d, \mathbf{x}, \mathbf{y})} \underbrace{p(d | \mathbf{x}, \mathbf{y}, \mathcal{A}, \vec{\boldsymbol{\alpha}}, \lambda, \eta, \boldsymbol{\beta})}_{\text{rule list posterior}}, \end{aligned} \tag{11}$$

where the rule list posterior is available from (9). Since BRL-post is computed over many rule lists, it is not human-interpretable; however, Letham *et al.* suggest that a few point estimates could be used as examples.

3.7 REMARKS

Many classifiers are specified by a collection of unknown, non-random parameters, and classifier training is the process of estimating these parameters. For example, a discriminative classifier hypothesizes a model for $p(\mathbf{y}|\mathbf{x}, \psi)$, and training seeks the maximum-likelihood estimate of the parameters ψ. Hence, the labels $\mathbf{y}$ are random variables conditioned on the features $\mathbf{x}$ and the parameters ψ.

In contrast, BRL treats the rule list d itself as being random. Classifier training uses the rule list posterior $p(d|\mathbf{x}, \mathbf{y}, \mathcal{A}, \vec{\boldsymbol{\alpha}}, \lambda, \eta, \boldsymbol{\beta})$ to compute an optimal estimate; we consider a point estimate $\hat{d}$ such as BRL-point.

There are three posterior distributions of interest in BRL. First, the generative model introduces the posterior consequent $\{y|\boldsymbol{\theta}_{k(j)} \sim \text{Multinomial}(\boldsymbol{\theta}_{k(j)}),\ \boldsymbol{\theta}_{k(j)}|d, \mathbf{x}, \mathbf{y}, \boldsymbol{\alpha}_{k(j)} \sim \text{Dirichlet}(\boldsymbol{\alpha}_{k(j)} + \mathbf{N}_j(d, \mathbf{x}, \mathbf{y}))\}$ for each antecedent in d. Second, the rule list posterior $p(d|\mathbf{x}, \mathbf{y}, \mathcal{A}, \vec{\boldsymbol{\alpha}}, \lambda, \eta, \boldsymbol{\beta})$ describes the distribution of rule lists given the training data $(\mathbf{x}, \mathbf{y})$, antecedent set $\mathcal{A}$, and hyperparameters. Third, the selected estimate $\hat{d}$ is an ordered list of antecedents; it does not contain any consequents. To classify a new feature vector $\tilde{x}$ with $\hat{d}$, the posterior predictive $p(\tilde{y} = \ell|\tilde{x}, \hat{d}, \mathbf{x}, \mathbf{y}, \vec{\boldsymbol{\alpha}})$ is required to obtain the Dirichlet-multinomial consequents.

if female **then**
    Pr(survived) = 0.73
else if child **then**
    Pr(survived) = 0.50
else if crew **then**
    Pr(survived) = 0.23
else
    Pr(survived) = 0.18
end if

Figure 11. Example falling rule list for Titanic *data*

3.8 FALLING RULE LISTS

A special kind of decision list, introduced by Wang and Rudin [14], is a *falling rule list* (FRL). It is a decision list in which the label probability decreases monotonically down the list. Figure 11 shows an example.

Falling rule lists are not the focus of this report, but they deserve mention for several reasons. First, the monotonicity property distinguishes FRL from ordinary decision lists and makes FRL ideally suited for applications like triage or sorting, which ordinary classifiers are not designed for. Monotonicity is an additional requirement that is not even considered—and generally not satisfied—by most decision lists.

Second, the monotonicity requirement acts as a strong constraint, so the predictive accuracy of FRL might be lower than that of other classifiers that do not attempt to satisfy it. However, in some cases, there is little or no reduction in performance. For some applications, monotonicity might be a necessity, and in other cases, gaining monotonicity might be worth losing a small amount of predictive accuracy.

Third, FRL was developed from BRL, so it shares an academic and technical genealogy with BRL. FRL bears many mathematical similarities to BRL: It is a Bayesian approach to rule list training based on FPM and MCMC sampling. FRL uses Gibbs sampling and collapsed Metropolis-Hastings sampling to sample rule lists from a desired posterior distribution. Consequently, virtually all of the ideas that this report uses to derive RBRL from BRL can also be used to create recursive algorithms for FRL.

Finally, AIM has supported development of new Optimized Falling Rule Lists and Softly Falling Rule Lists [35]. These training algorithms use a non-Bayesian optimization approach that relaxes the monotonicity requirement from a hard constraint to a soft one.

4. RECURSIVE BAYESIAN RULE LISTS (RBRL)

This section shows how BRL can be made analogous to RBT to form *recursive Bayesian rule lists* (RBRL), which map the problem of classifier adaptation to the problem of tracking a moving target: *RBRL essentially tracks the underlying rule list.*

4.1 TEMPORAL MODEL

We extend the generative model of Sec. 3.4 and Figure 10 by adding a discrete-time dimension. Unfortunately, doing so requires adding to the already dense notation. Recall that indexing and constant elements are listed in Tables 1 and 2. For elements that can change with time, we add a subscript for the *discrete-time index* $n = 1, 2, \ldots$; it precedes all other subscripts. For example, for time n, M_n denotes the number of samples, $\mathbf{x}_n = \{x_{n,1},\, x_{n,2},\, \ldots,\, x_{n,M_n}\}$ denotes the features, and $\mathbf{y}_n = \{y_{n,1},\, y_{n,2},\, \ldots,\, y_{n,M_n}\}$ denotes the labels. The time-varying elements are organized in Table 4, which closely corresponds to Table 3. New elements introduced for RBRL are briefly described here and explained later in this section. To reduce the notation slightly, BRL constant elements such as $\mathcal{A}$, λ, η, $\boldsymbol{\beta}$ and new RBRL constant elements such as κ, ζ, and ν will not be shown in conditional distributions.

A graphical model for the first three time indexes appears in Figure 12. At time $n = 1$, regular BRL training generates the rule list d_1 and updated hyperparameters $\vec{\boldsymbol{\alpha}}_2$. The extension of the BRL model allows the rule list and hyperparameters to evolve on subsequent time steps; it is shown for times $n = 2$ and 3 and repeats indefinitely. It has four components.

First, the rule list is permitted to change over time. The rule list d_n depends on three independent variables: the previous rule list d_{n-1}, fully-observed *antecedent feedback* $\mathbf{a}_n$ provided by users, and unobserved *process noise* $\mathbf{v}_{n-1}$. The antecedent feedback and process noise can directly change the rule list. This part of the model gives RBRL a way to model dataset shift; it is developed in Sec. 4.3, which determines the rule list transition distribution $p(d_n|d_{n-1}, \mathbf{a}_n)$. The process noise can be modeled either as independent of d_{n-1} or dependent on it (dashed lines). The parameters ν and ζ are discussed in Sec. 6.5.1 and Sec. 6.5.2. At time $n = 1$, there is no antecedent feedback, so $\mathbf{a}_1 = \emptyset$, and there is no process noise.

Second, at each time, the correct labels $\mathbf{y}_n$ depend on the rule list d_n, the observed features $\mathbf{x}_n$, and fully-observed *rule feedback* $\mathbf{r}_n$ from the users. The dependence on d_n and $\mathbf{x}_n$ occurs through the capture mechanism $k(j(d_n, x_{n,m}))$. These variables are independent for different time or sample indexes. This model component is described in Sec. 4.4.1, which gives the rule list likelihood function with feedback $p(\mathbf{y}_n|d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)$. The features and rule feedback indirectly influence the rule list through the likelihood function, which modifies the rule list posterior. There is no rule feedback at time $n = 1$, so $\mathbf{r}_1 = \emptyset$.

Third, the model allows for the possibility that the correct labels $\mathbf{y}_n$ are not available and *noisy truth labels* $\mathbf{z}_n$ are observed instead. We assume there is a fixed set of $N_{\mathcal{T}}$ *truthers*, indexed by the set $\mathcal{T}$. For each sample $z_{n,m} \in \mathbf{z}_n$, let $\mathcal{T}_{n,m}$ be the set containing the indexes of the truthers who provided a truth label, and let $z_{n,m,t}$ denote the noisy truth label from the tth truther,

$t \in \mathcal{T}_{n,m}$. The noisy truth label $z_{n,m,t}$ is assumed to depend on the unseen correct label $y_{n,m}$ and unobserved, independent *truthing-error noise* or *truthing errors* $w_{n,m,t}$. The truthing errors for the mth sample at time n are $w_{n,m} = (w_{n,m,t})_{t \in \mathcal{T}_{n,m}}$, and the collective truthing errors at time n are $\mathbf{w}_n = (w_{n,1}, \ldots, w_{n,M_n})$. Sec. 4.4.2 and Sec. 4.4.3 develop this part of the model, which uses fractional truth labels $\check{\mathbf{y}}_n$, $\check{y}_{n,m}$, $\check{y}_{n,m,\ell}$ calculated from $\mathbf{z}_n$ to obtain an approximation for the rule list likelihood function $p(\mathbf{z}_n | d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)$.

Finally, the model assumes that the consequent for each antecedent $a_{k_n(j)} \in d_n$ has a Dirichlet-multinomial distribution, just as in BRL. However, the Dirichlet hyperparameters $\boldsymbol{\alpha}_{n+1,k_n(j)}$ now depend on the previous hyperparameters $\boldsymbol{\alpha}_{n,k_n(j)}$, the features and labels $(\mathbf{x}_n, \mathbf{y}_n)$ captured by the antecedent, and the rule feedback $\mathbf{r}_n$. Because the rule list might change over time, the model considers all antecedents $a_k \in \mathcal{A}$, not just the antecedents in the rule list at the current time. The hyperparameter update is given in Sec. 4.4.1 and Sec. 4.4.3. It includes a decay constant $0 < \kappa \leq 1$, so the influence of past data decreases exponentially if $\kappa \neq 1$.

Figure 13 shows the detailed graphical model for two time steps, which is a first-order *hidden Markov model* (HMM). By dropping many of the details, we arrive at the simplified HMM in Figure 14.

4.2 ANALOGY WITH RECURSIVE BAYESIAN TRACKING

As mentioned previously, RBRL is analogous to *recursive Bayesian tracking* (RBT), which is reviewed in App. C. RBT is a recursive procedure for estimating the state of a dynamical system as it changes over time. The state space is usually a continuous space $\mathbf{S}'$, such as $\mathbb{R}^N$, and the system state at time n is $s'_n \in \mathbf{S}'$. The system is assumed to evolve according to a state-space model that is driven by a known control input a'_n and unknown process noise v'_{n-1}. The system output y'_n is determined by the state and known direct-feed control inputs x'_n and r'_n. However, y'_n is not observed directly because of interference by measurement noise w'_n, so only noisy observations z'_n are available.

In RBRL, the state space is the set of all possible rule lists $\mathbf{D}$ formed from $\mathcal{A}$, so it is a finite—and hence discrete—space, and *the state is the rule list* d_n. Table 5 shows many of the correspondences between RBRL and RBT. A minor difference between RBRL and RBT is that RBRL must maintain the hyperparameters $\vec{\boldsymbol{\alpha}}_n$ and use them to obtain the posterior predictive from the selected rule list $\hat{d}_n$.

A block diagram of the RBRL state-space model appears in Figure 15; the corresponding block diagram for RBT is Figure C.1. The two components of the RBRL state-space model—the feedback and dynamics model and the truthing model—are described in Sec. 4.3 and Sec. 4.4.

4.3 ANTECEDENT FEEDBACK AND RULE LIST DYNAMICS

Let $d_n \in \mathbf{D}$ be a rule list at time n. To represent how it changes over time, we introduce a *feedback and dynamics model* of the form:

$$d_n = f(d_{n-1}, \mathbf{a}_n, \mathbf{v}_{n-1}). \tag{12}$$

TABLE 4

Recursive Bayesian Rule Lists Elements.
An asterisk indicates new elements that do not appear in Table 3.

Symbol	Type	Description
$a_{k_n(j)}$	antecedent	jth antecedent in rule list d_n or $k_n(j)$th antecedent in antecedent set $\mathcal{A}$
$\mathbf{a}_n$	abstract	*antecedent feedback
$\vec{\boldsymbol{\alpha}}_n = (\boldsymbol{\alpha}_{n,0}, \ldots, \boldsymbol{\alpha}_{n,\lvert\mathcal{A}\rvert})$	list of vectors	Dirichlet parameters for all antecedents
$\boldsymbol{\alpha}_{n,k} = (\alpha_{n,k,1}, \ldots, \alpha_{n,k,L})$	vector	Dirichlet parameters for kth antecedent
$\alpha_{n,k,\ell}$	scalar	Dirichlet parameter for kth antecedent, ℓth label
$d_n = (a_{k_n(1)}, \ldots, a_{k_n(\lvert d_n\rvert)})$	ordered list	rule list
$k_n(j)$	scalar	index into antecedent set $\mathcal{A}$ for jth antecedent in rule list d_n
M_n	scalar	number of samples
$\mathbf{N}_j(d_n, \mathbf{r}_n) = (N_{j,1}(d_n, \mathbf{r}_n), \ldots, N_{j,L}(d_n, \mathbf{r}_n))$	vector	*pseudocounts of all labels in $\mathbf{r}_n$ for patterns in $\mathbf{r}_n$ captured by the jth antecedent in rule list d_n
$N_{j,\ell}(d_n, \mathbf{r}_n)$	scalar	*pseudocount of ℓth label in $\mathbf{r}_n$ for patterns in $\mathbf{r}_n$ captured by the jth antecedent in rule list d_n
$\mathbf{N}_j(d_n, \mathbf{x}_n, \mathbf{y}_n) = (N_{j,1}(d_n, \mathbf{x}_n, \mathbf{y}_n), \ldots, N_{j,L}(d_n, \mathbf{x}_n, \mathbf{y}_n))$	vector	counts of all labels in $\mathbf{y}_n$ for samples in $\mathbf{x}_n$ captured by the jth antecedent in rule list d_n
$N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{y}_n)$	scalar	count of ℓth label in $\mathbf{y}_n$ for samples in $\mathbf{x}_n$ captured by the jth antecedent in rule list d_n
$\mathbf{r}_n$	abstract	*rule feedback
$\mathcal{T}_{n,m}$	set of indexes	*indexes of truthers for mth sample
$\vec{\boldsymbol{\theta}}_n = (\boldsymbol{\theta}_{n,0}, \ldots, \boldsymbol{\theta}_{n,\lvert\mathcal{A}\rvert})$	list of vectors	multinomial parameters for all consequents
$\boldsymbol{\theta}_{n,k} = (\theta_{n,k,1}, \ldots, \theta_{n,k,L})$	vector	multinomial parameters for kth consequent
$\theta_{n,k,\ell}$	scalar	multinomial parameter for kth consequent, ℓth label
$\mathbf{v}_n$	abstract	*process noise
$\mathbf{w}_n = (w_{n,1}, \ldots, w_{n,M_n})$	list of sets	*all truthing errors
$w_{n,m} = (w_{n,m,t})_{t \in \mathcal{T}_{n,m}}$	set	*truthing errors for mth sample
$w_{n,m,t}$	abstract	*truthing error for mth sample, tth truther
$\mathbf{x}_n = (x_{n,1}, \ldots, x_{n,M_n})$	list of vectors	all features
$x_{n,m}$	vector	features for mth sample
$\mathbf{y}_n = (y_{n,1}, \ldots, y_{n,M_n})$	list of scalars	all correct labels
$y_{n,m}$	scalar	correct label for mth sample
$\check{\mathbf{y}}_n = (\check{y}_{n,1}, \ldots, \check{y}_{n,M_n})$	list of vectors	*all fractional truth labels
$\check{y}_{n,m} = (\check{y}_{n,m,1}, \ldots, \check{y}_{n,m,L})$	vector	*fractional truth labels for mth sample
$\check{y}_{n,m,\ell}$	scalar	*fractional truth label for mth sample, ℓth label
$\mathbf{z}_n = (z_{n,1}, \ldots, z_{n,M_n})$	list of sets	*all noisy truth labels
$z_{n,m} = (z_{n,m,t})_{t \in \mathcal{T}_{n,m}}$	set	*noisy truth labels for mth sample
$z_{n,m,t}$	scalar	*noisy truth label for mth sample from tth truther

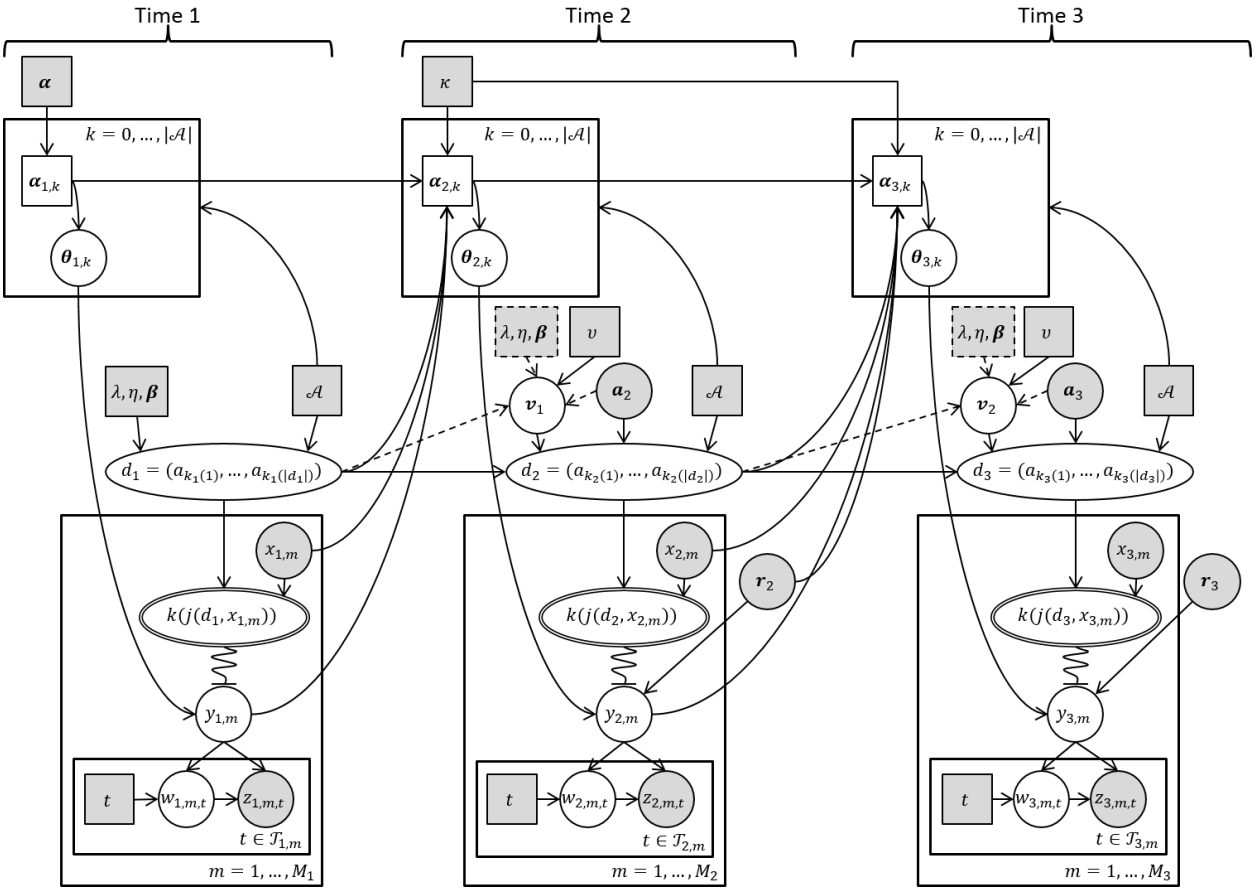

Figure 12. Generative model for BRL initialization and RBRL temporal evolution

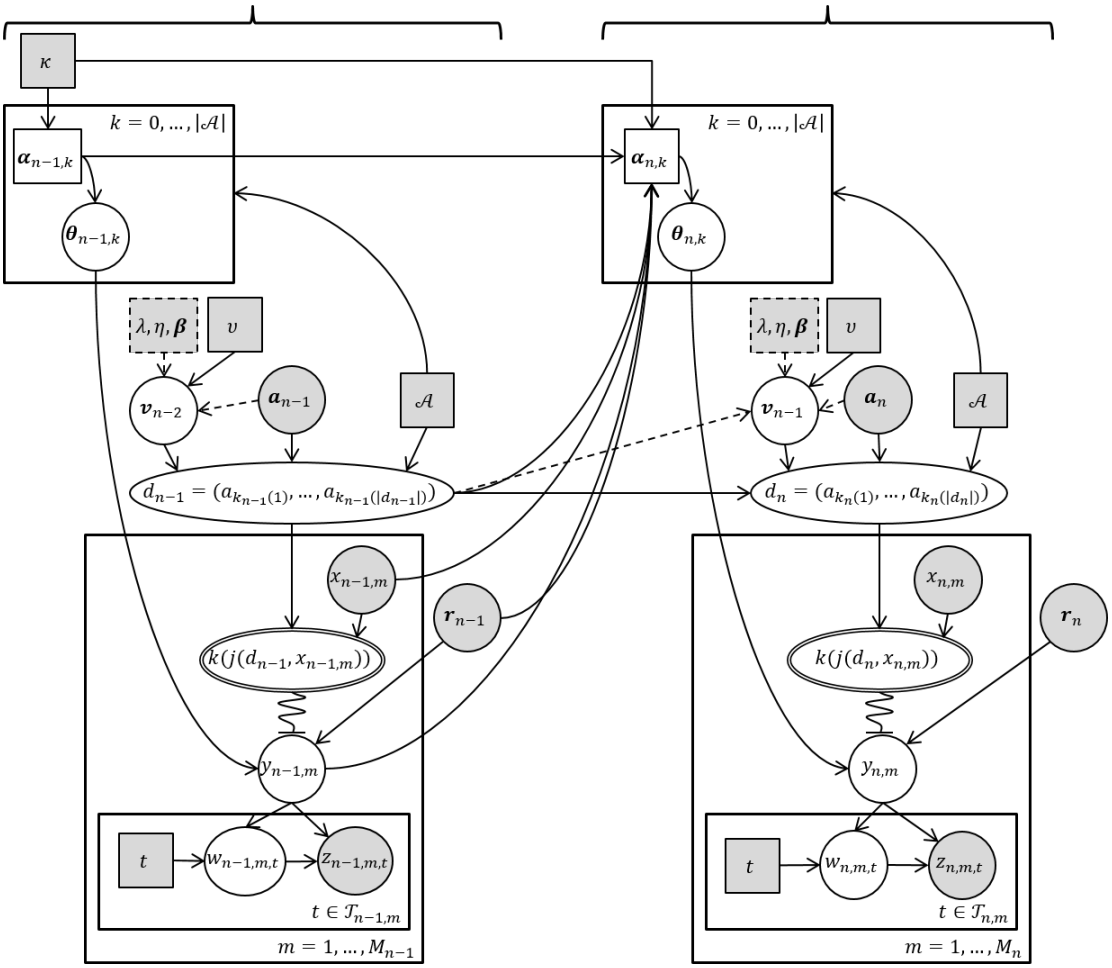

Figure 13. Generative model for RBRL over two time steps

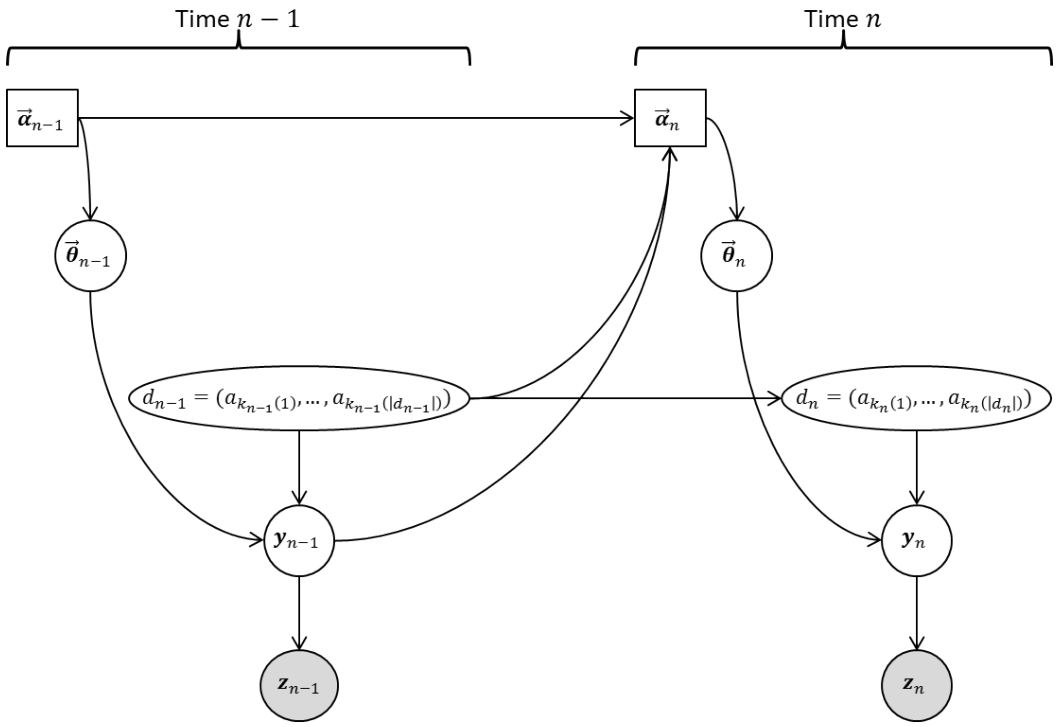

Figure 14. Simplified hidden Markov model for recursive Bayesian rule lists

We assume that f can be written as a composition of functions:

$$\tilde{d}_n = f_a(d_{n-1}, \mathbf{a}_n), \tag{13}$$

$$d_n = f_v(\tilde{d}_n, \mathbf{v}_{n-1}), \tag{14}$$

so that

$$d_n = f_v(f_a(d_{n-1}, \mathbf{a}_n), \mathbf{v}_{n-1}). \tag{15}$$

We choose this composition because the user's antecedent feedback $\mathbf{a}_n$ will be based on the previous rule list d_{n-1}, so it should be applied before the process noise.

The antecedent feedback $\mathbf{a}_n$ behaves like a control input in RBT that modifies d_{n-1} to yield the intermediate rule list $\tilde{d}_n$, while the process noise $\mathbf{v}_{n-1}$ causes random changes to $\tilde{d}_n$ to produce d_n. Examples of antecedent feedback could be the addition or removal of an antecedent or a reordering of antecedents. The specific types of antecedent feedback will depend on the application and interface, so $\mathbf{a}_n$ is described as "abstract" in Table 5.

4.3.1 Rule List Transition Distribution

We require the *rule list transition distribution* $p(d_n|d_{n-1}, \mathbf{a}_n)$, which is analogous to the state transition distribution in RBT [cf. (C.9)]. It will be useful to show explicit rule list indexes in the discrete state space $\mathbf{D}$. Suppose that d_{n-1} is the jth rule list in $\mathbf{D}$ and d_n is the ith rule list in $\mathbf{D}$;

TABLE 5

Analogous Entities in RBRL and RBT

Recursive Bayesian Rule Lists	Recursive Bayesian Tracking
rule list set $\mathbf{D}$	state space $\mathbf{S}'$
time index n	time index n
rule list d_n	state s'_n
antecedent feedback $\mathbf{a}_n$	control input a'_n
rule feedback $\mathbf{r}_n$	direct-feed control input r'_n
features $\mathbf{x}_n$	direct-feed control input x'_n
correct labels $\mathbf{y}_n$	system output y'_n
noisy truth labels $\mathbf{z}_n$	noisy observations z'_n
hyperparameters $\vec{\boldsymbol{\alpha}}_n$	—
feedback and dynamics model f (12)	control and dynamics model f' (C.1)
feedback model f_a (13)	control model $f'_{a'}$ (C.3)
dynamics model f_v (14)	dynamics model $f'_{v'}$ (C.4)
process noise $\mathbf{v}_{n-1}$	process noise v'_{n-1}
rule list transition distribution $p(d_n \mid d_{n-1}, \mathbf{a}_n)$	state transition distribution $p(s'_n \mid s'_{n-1}, a'_n)$
truthing model h (24)	measurement model h' (C.2)
correct-label model h_{xr} (25)	system-output model $h'_{x'r'}$ (C.6)
truthing-error model h_w (26)	measurement-error model $h'_{w'}$ (C.7)
truthing errors $\mathbf{w}_n$	measurement noise w'_n
rule list likelihood function $p(\mathbf{z}_n \mid d_n, \mathbf{x}_n, \mathbf{r}_n)$	state likelihood function $p(z'_n \mid s'_n, x'_n, r'_n)$
feedback and dynamics update (34)	control and dynamics update (C.9)
rule list prior at n: $p(d_n \mid \mathbf{z}_{1:n-1}, \mathbf{x}_{1:n-1}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n-1})$	state prior at n: $p(s'_n \mid z'_{1:n-1}, x'_{1:n-1}, a'_{1:n}, r'_{1:n-1})$
training update (35)	measurement update (C.10)
rule list posterior at n: $p(d_n \mid \mathbf{z}_{1:n}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n})$	state posterior at n: $p(s'_n \mid z'_{1:n}, x'_{1:n}, a'_{1:n}, r'_{1:n})$
trained rule list $\hat{d}_n$	filtered state estimate $\hat{s}'_n$
posterior predictive	—

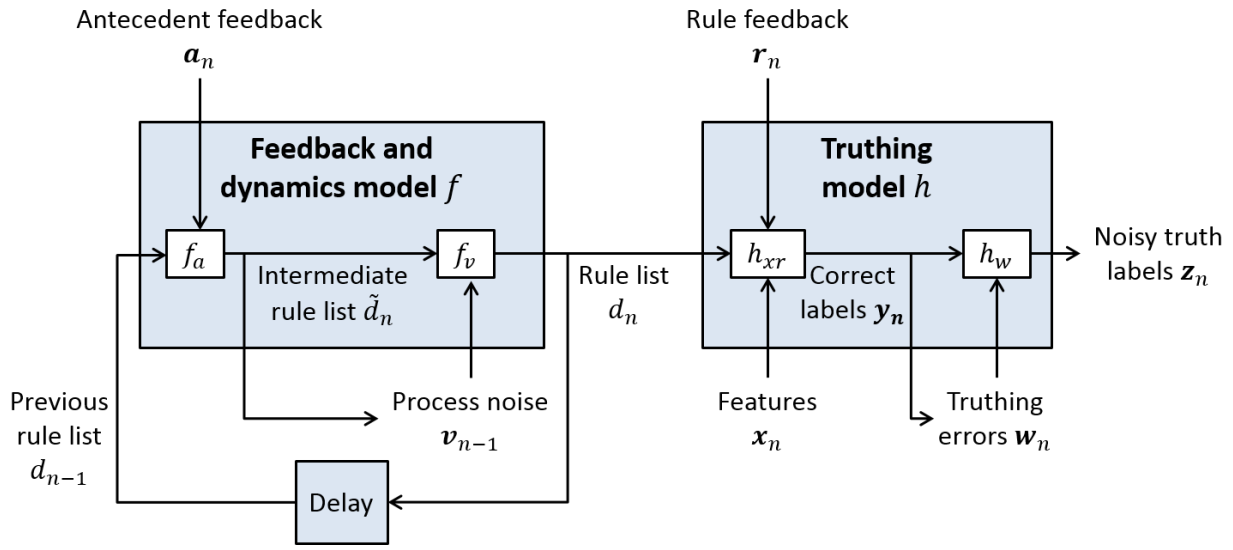

Figure 15. Detailed state-space model in recursive Bayesian rule lists

that is, $d_{n-1} = d^{(j)} \in \mathbf{D}$ and $d_n = d^{(i)} \in \mathbf{D}$, where d_{n-1} and d_n are random, and $d^{(i)}$ and $d^{(j)}$ are fixed, non-random elements of $\mathbf{D}$.

Then from (13),

$$\begin{aligned} p(d_n|d_{n-1}, \mathbf{a}_n) &= p(d_n^{(i)}|d_{n-1}^{(j)}, \mathbf{a}_n) \\ &= p(d_n = d^{(i)}|d_{n-1} = d^{(j)}, \mathbf{a}_n) \\ &\overset{(a)}{=} p(d_n = d^{(i)}|\tilde{d}_n = f_a(d^{(j)}, \mathbf{a}_n)) \\ &\overset{(b)}{=} \sum_{\mathbf{v}_{n-1} \in \{\mathbf{v}'_{n-1}: f_v(f_a(d^{(j)}, \mathbf{a}_n), \mathbf{v}'_{n-1}) = d^{(i)}\}} p(\mathbf{v}_{n-1}). \end{aligned}$$

Here, (a) is from (13): $d^{(j)}$ and $\mathbf{a}_n$ are passed into f_a, which yields the intermediate rule list $f_a(d^{(j)}, \mathbf{a}_n)$. Then (b) indicates that the desired conditional probability is equal to the probability that the process noise will drive the intermediate rule list to $d^{(i)}$ when (14) is applied.

We can drop the indexing to write this relationship as two steps:

$$\tilde{d}_n = f_a(d_{n-1}, \mathbf{a}_n), \tag{16}$$

and

$$\underbrace{p(d_n|d_{n-1},\mathbf{a}_n)}_{\substack{\text{rule list transition}\\ \text{distribution}}} = p(d_n|f_a(d_{n-1},\mathbf{a}_n)) \tag{17}$$

$$= \underbrace{p(d_n|\tilde{d}_n)}_{\substack{\text{switching}\\ \text{distribution}}} \tag{18}$$

$$= \sum_{\mathbf{v}_{n-1}\in\{\mathbf{v}'_{n-1}:f_v(\tilde{d}_n,\mathbf{v}'_{n-1})=d_n\}} p(\mathbf{v}_{n-1}|\tilde{d}_n). \tag{19}$$

We refer to $p(d_n|\tilde{d}_n)$ as the *switching distribution* since it gives the conditional probability of $\tilde{d}_n$ switching to d_n. The equations show that the process noise could depend on the intermediate rule list $\tilde{d}_n$. Although the last expression requires a summation over all possible process noise realizations that yield d_n from $\tilde{d}_n$, the implementations we consider will not requite such prohibitive computation.

The process noise and dynamics model (14) are mainly shown to complete the analogy with RBT. In RBT, they are typically based on a physically-motivated dynamical systems model and used to determine the state transition distribution [cf. (C.9)]. For RBRL, the rule list transition distribution $p(d_n|d_{n-1},\mathbf{a}_n)$ or switching distribution $p(d_n|\tilde{d}_n)$ is the important relationship. As long as one of these distributions can be computed, RBRL can be implemented without explicitly specifying the process-noise distribution and dynamics model (e.g., Alg. 6 and Sec. 6.5.2).

4.3.2 Static Case

One simple but important situation is the *static case*, in which there is neither antecedent feedback nor process noise. Then (12) reduces to $d_n \equiv d_{n-1}$, and the transition distribution is just

$$p(d_n|d_{n-1},\mathbf{a}_n) = \begin{cases} 1, & \text{if } d_n = d_{n-1};\\ 0, & \text{otherwise.}\end{cases} \tag{20}$$

Because the rule list remains constant in this case, the problem of estimating the rule list is parameter estimation rather than tracking [22]. The particle filter-based algorithms discussed in Sec. 6 are therefore not applicable. However, the discrete nature of RBRL means the grid-based algorithms of Sec. 5 are still valid; Sec. 5.2 and Alg. 7 provide a simple, efficient algorithm.

4.4 RULE FEEDBACK AND TRUTHING ISSUES

This section covers the RBRL truthing model, which handles rule feedback and various truthing issues.

4.4.1 Correct Truth and Rule Feedback

This section assumes that $\mathbf{z}_n \equiv \mathbf{y}_n$, meaning that the *correct labels* $\mathbf{y}_n$ are available; they correspond to the pristine system output y'_n in RBT. Sec. 4.4.2 and Sec. 4.4.3 discuss a model for

truthing issues in which the correct labels are unavailable, and noisy truth labels must be used instead.

Rule feedback, denoted by $\mathbf{r}_n$, allows users to encourage or discourage antecedent-consequent pairs in the likelihood function (Sec. 3.4.2). The BRL likelihood function is already based on a Dirichlet-multinomial model, so for each antecedent $a_{k_n(j)} \in d_n$, we can apply rule feedback through an additional length-L pseudocount vector $\mathbf{N}_j(d_n, \mathbf{r}_n) = (N_{j,1}(d_n, \mathbf{r}_n), \ldots, N_{j,L}(d_n, \mathbf{r}_n))$ that behaves just like $\mathbf{N}_j(d_n, \mathbf{x}_n, \mathbf{y}_n)$. The posterior consequent for $a_{n,k_n(j)}$ becomes $\{y_n | \boldsymbol{\theta}_{n,k_n(j)} \sim \text{Multinomial}(\boldsymbol{\theta}_{n,k_n(j)}),\ \boldsymbol{\theta}_{n,k_n(j)} \sim \text{Dirichlet}(\boldsymbol{\alpha}_{n,k_n(j)} + \mathbf{N}_j(d_n, \mathbf{x}_n, \mathbf{y}_n) + \mathbf{N}_j(d_n, \mathbf{r}_n))\}$.

Including the rule-feedback pseudocounts in (6) or (7) gives the *rule list likelihood function with feedback*:

$$p(\mathbf{y}_n | d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n) = \prod_{j=0}^{|d_n|} \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_{n,k_n(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{y}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})} \times \prod_{\ell=1}^{L} \frac{\Gamma(N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{y}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})}{\Gamma(\alpha_{n,k_n(j),\ell})} \tag{21}$$

$$\propto \prod_{j=0}^{|d_n|} \frac{\prod_{\ell=1}^{L} \Gamma(N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{y}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{y}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})}. \tag{22}$$

Here, $N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{y}_n)$ and $N_{j,\ell}(d_n, \mathbf{r}_n)$ highlight that the former originates from the training data $(\mathbf{x}_n, \mathbf{y}_n)$, and the latter arises from the rule feedback $\mathbf{r}_n$.

We must also update the Dirichlet hyperparameters $\vec{\boldsymbol{\alpha}}_n$ for the next time index. For antecedent $a_k \in \mathcal{A}$, we assume that $\boldsymbol{\alpha}_{n+1,k}$ is equal to $\boldsymbol{\alpha}_{n,k}$ from the posterior consequent, scaled by a decay constant $0 < \kappa \leq 1$ and bounded below by unity. For the antecedents in d_n, the Dirichlet-multinomial posterior applies, and for the other antecedents, the hyperparameters do not change. Thus,

$$\boldsymbol{\alpha}_{n+1,k} = \begin{cases} \max\{1, \kappa \cdot (\boldsymbol{\alpha}_{n,k} + \mathbf{N}_k(d_n, \mathbf{x}_n, \mathbf{y}_n) + \mathbf{N}_k(d_n, \mathbf{r}_n))\}, & \text{if } a_k \in d_n; \\ \max\{1, \kappa \cdot \boldsymbol{\alpha}_{n,k}\}, & \text{if } a_k \notin d_n. \end{cases} \tag{23}$$

There are many ways rule feedback from users can be mapped into pseudocounts. A direct mapping would require that the user say something like, "The pattern (first-class, child) should act like ten more observations of the label 'survived,'" for the Titanic example. More sophisticated mappings specific to the problem domain, user base, and human-machine interface will likely be needed for different applications, so $\mathbf{r}_n$ is described as "abstract" in Table 4, and the details of the mapping are left as an implementation detail.

4.4.2 Truthing Model

Classifier training normally requires the correct label for each training feature vector. However, RBRL can account for certain truthing issues by introducing a *truthing model* of the form

$$\mathbf{z}_n = h(d_n, \mathbf{x}_n, \mathbf{r}_n, \mathbf{w}_n), \tag{24}$$

which is analogous to the measurement model (C.2) in RBT. Here, $\mathbf{w}_n$ models *truthing errors* and corresponds to measurement noise w'_n in RBT, and $\mathbf{z}_n$ represents *noisy truth labels* and is analogous to noisy observations z'_n in RBT.

We assume that the model can be expressed as a composition of functions. First, the correct labels $\mathbf{y}_n$ obey [cf. (C.6)]

$$\mathbf{y}_n = h_{xr}(d_n, \mathbf{x}_n, \mathbf{r}_n). \tag{25}$$

Second, the noisy truth labels $\mathbf{z}_n$ are a function of the correct labels and truthing errors [cf. (C.7)]:

$$\mathbf{z}_n = h_w(\mathbf{y}_n, \mathbf{w}_n). \tag{26}$$

Then the model becomes [cf. (C.8)]

$$\mathbf{z}_n = h_w(h_{xr}(d_n, \mathbf{x}_n, \mathbf{r}_n), \mathbf{w}_n). \tag{27}$$

In (25), the correct labels $\mathbf{y}_n$ correspond to the pristine system output y'_n in RBT. The features $\mathbf{x}_n$ and rule feedback $\mathbf{r}_n$ behave like direct-feed control inputs in RBT. In (26), the correct labels are corrupted by truthing errors, which produce in the observed noisy truth labels.

4.4.3 Multiple Truthers and Truthing Errors

This section further defines the truthing model to allow RBRL to handle incorrect truth labels and multiple, conflicting truth labels from different truthers.[12] The truthing model appears in Figures 12 and 13 as the plates indexed by t. It yields an approximate expression for the rule list likelihood function $p(\mathbf{z}_n|d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)$.

Recall that $x_{n,m}$ and $y_{n,m}$ are the features and the unobserved correct label for the mth sample. Also recall that $\mathcal{T} = \{1, 2, \ldots, N_{\mathcal{T}}\}$ is the set of indexes of the truthers, who independently provide noisy truth labels for the samples. Next, consider the sample $x_{n,m}$. Let the non-empty set $\mathcal{T}_{n,m} \subseteq \mathcal{T}$ contain the indexes of the truthers who labeled $x_{n,m}$. For $t \in \mathcal{T}_{n,m}$, define $z_{n,m,t} \in \mathcal{L}$ as the noisy truth label assigned to $x_{n,m}$ by the tth truther. Hence, a different combination of truthers can provide truth labels for different samples. The set of noisy truth labels for $x_{n,m}$ is $z_{n,m} = \{z_{n,m,t} : t \in \mathcal{T}_{n,m}\}$. Finally, the set of noisy truth labels for all samples at time n is denoted by $\mathbf{z}_n = \{z_{n,1}, \ldots, z_{n,M_n}\}$.

We assume that the *truthing-error distribution* $p(z_t|y)$ is known for all $t \in \mathcal{T}$, where here we consider a single sample. Dawid and Skene [36] describe two general methods for estimating the conditional distribution $p(z_t|y)$ for each truther. First, if one has a data set with known correct labels, then one can have each truther examine the features and assign labels, and finally one can compute the maximum-likelihood estimate of $p(z_t|y)$ for each truther. Second, if one does not have a data set with known correct labels, then one can have each truther assign labels and apply the *expectation-maximization* (EM) algorithm [37] to find a locally optimum estimate of $p(z_t|y)$ for each truther. Other authors [38] [39] [40] [41] have built on this technique and constructed different

[12] The approach described in this section could also be included in initial BRL training.

models for $p(z_t|y)$, but Dawid and Skene considered the most general case, in which $p(z_t|y)$ is an $L \times L$ array, say $A^{(t)}$, where $A^{(t)}_{ij} = p(z_t = j|y = i)$ for $(i, j) \in \mathcal{L} \times \mathcal{L}$.

We do not define h_{xr} as a deterministic function in (25). After all, if we could do so, then we would not need to train a classifier in the first place. Rather, we continue to use the Dirichlet-multinomial model of Sec. 4.4.1.

In RBT, the measurement noise and measurement model are part of a dynamical systems model and used to obtain the state likelihood function. For RBRL, we mainly need the rule list likelihood function $p(\mathbf{z}_n|d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)$. It is not necessary to define the truthing-error distribution and h_w explicitly in (26); instead, we make two assumptions that let us find the approximate rule list likelihood function. First, we assume that the noisy truth labels assigned by the tth truther are independent and distributed according to $p(z_t|y)$. Second, we assume that the noisy truth labels for $x_{n,m}$ are conditionally independent given $y_{n,m}$; that is, the truthers do not influence each other. Hence, $z_{n,m,t}$ depends only on the correct label $y_{n,m}$ and truther index t.

These assumptions mean that, for a given combination of y and t, the noisy truth labels are independent and identically distributed. However, if the correct label or truther index changes, then a different conditional distribution applies.[13] This dependence is shown by the dependence of $w_{n,m,t}$ on $y_{n,m}$ and t in Figures 12 and 13. We could also define a trivial function like $z_{n,m,t} = y_{n,m} + w_{n,m,t}$ and then specify $p(w_{n,m,t}|y_{n,m}, t)$ in terms of $p(z_t|y)$, but this step is unnecessary.

These assumptions give

$$\begin{aligned} p(z_{n,m}|y_{n,m}, d_n, x_{n,m}, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n) &= \prod_{t \in \mathcal{T}_{n,m}} p(z_{n,m,t}|y_{n,m}, d_n, x_{n,m}, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n) \\ &\overset{(a)}{=} \prod_{t \in \mathcal{T}_{n,m}} p(z_{n,m,t}|y_{n,m}), \end{aligned} \tag{28}$$

where (a) is from (26). If there are no truthing errors, then all truthers in $\mathcal{T}_{n,m}$ assign the same correct label $y_{n,m}$, and

$$p(z_{n,m,t}|y_{n,m}) = \begin{cases} 1, & \text{if } z_{n,m,t} = y_{n,m}; \\ 0, & \text{otherwise.} \end{cases} \tag{29}$$

When correct truth was assumed as in Sec. 4.4.1, the properties of the Dirichlet-multinomial distribution allowed us to express the likelihood function in terms of $N_{j,\ell}(d_n, \mathbf{x}_n, \mathbf{z}_n)$ and $N_{j,\ell}(d_n, \mathbf{r}_n)$. The presence of truthing errors does not allow such a solution because the posterior consequent no longer has a Dirichlet-multinomial distribution. Nevertheless, we can still exploit the pseudo-count interpretation of the Dirichlet hyperparameters to obtain an approximation to the likelihood function.

We assume that the unknown correct labels indeed have a Dirichlet-multinomial distribution, and for each sample $(x_{n,m}, z_{n,m})$, we use it to apportion the unit count for the unknown correct

[13] It is also straightforward to include dependence on additional attributes, such as sample difficulty [40] [41]. If it is denoted by δ, then $p(z_t|y)$ can be replaced by $p(z_t|y, \delta)$.

label $y_{n,m}$ among the possible labels. For the mth sample, we form a vector of *fractional truth labels* $\check{y}_{n,m} = (\check{y}_{n,m,1}, \ldots, \check{y}_{n,m,L})$ with $\check{y}_{n,m,\ell} \geq 0$, $\ell = 1, \ldots, L$, and $\sum_{\ell=1}^{L} \check{y}_{n,m,\ell} = 1$. Then

$$\begin{aligned}
\check{y}_{n,m,\ell} &= p(y_{n,m} = \ell | d_n, x_{n,m}, z_{n,m}, \vec{\boldsymbol{\alpha}}) \\
&\overset{(a)}{\propto} p(z_{n,m} | d_n, x_{n,m}, y_{n,m} = \ell, \vec{\boldsymbol{\alpha}}_n)\, p(y_{n,m} = \ell | d_n, x_{n,m}, \vec{\boldsymbol{\alpha}}_n) \\
&= p(z_{n,m} | y_{n,m} = \ell)\, p(y_{n,m} = \ell | d_n, x_{n,m}, \vec{\boldsymbol{\alpha}}_n) \\
&\overset{(b)}{=} \left(\prod_{t \in \mathcal{T}_{n,m}} p(z_{n,m,t} | y_{n,m} = \ell) \right) \frac{\alpha_{n,k(j(d_n,x_{n,m})),\ell}}{\sum_{\ell'=1}^{L} \alpha_{n,k(j(d_n,x_{n,m})),\ell'})}, \quad \ell = 1, \ldots, L; m = 1, \ldots, M_n, \quad (30)
\end{aligned}$$

where (a) is because of Bayes' rule; and in (b), the first term is from (28) and the second term is from (A.9) since $\boldsymbol{\theta}_{n,k(j(d_n,x_{n,m}))} \sim \text{Dirichlet}(\boldsymbol{\alpha}_{n,k(j(d_n,x_{n,m}))})$. We use this equation to construct $\check{y}_{n,m}$, and pseudocode appears in Alg. 3. If there are no truthing errors, then (29) applies, and $\check{y}_{n,m}$ becomes a vector of zeros except at element ℓ, where it equals one. Hence, this approach properly reduces to the case of correct truth.

Finally, we substitute $\check{\mathbf{y}}_n$ for $\mathbf{y}_n$ to obtain the *approximate rule list likelihood function* [cf. (21) and (22)]:

$$\begin{aligned}
p(\mathbf{z}_n | d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n) &\approx p(\mathbf{y}_n | d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)\big|_{\mathbf{y}_n = \check{\mathbf{y}}_n} \\
&= \prod_{j=0}^{|d_n|} \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_{n,k_n(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d_n, \mathbf{x}_n, \check{\mathbf{y}}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})} \\
&\quad \times \prod_{\ell=1}^{L} \frac{\Gamma(N_{j,\ell}(d_n, \mathbf{x}_n, \check{\mathbf{y}}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})}{\Gamma(\alpha_{n,k_n(j),\ell})} \quad (31) \\
&\propto \prod_{j=0}^{|d_n|} \frac{\prod_{\ell=1}^{L} \Gamma(N_{j,\ell}(d_n, \mathbf{x}_n, \check{\mathbf{y}}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d_n, \mathbf{x}_n, \check{\mathbf{y}}_n) + N_{j,\ell}(d_n, \mathbf{r}_n) + \alpha_{n,k_n(j),\ell})}. \quad (32)
\end{aligned}$$

We also get the approximate hyperparameter update [cf. (23)]

$$\boldsymbol{\alpha}_{n+1,k} = \begin{cases} \max\{1, \kappa \cdot (\boldsymbol{\alpha}_{n,k} + \mathbf{N}_k(d_n, \mathbf{x}_n, \check{\mathbf{y}}_n) + \mathbf{N}_k(d_n, \mathbf{r}_n))\}, & \text{if } a_k \in d_n; \\ \max\{1, \kappa \cdot \boldsymbol{\alpha}_{n,k}\}, & \text{if } a_k \notin d_n; \end{cases} \quad (33)$$

In the above equations, $N_{j,\ell}$ and $\mathbf{N}_k$ now allow for fractional count assignments for the different labels. The hyperparameter update is implemented in Alg. 4. If the decay factor $\kappa \equiv 1$ and all elements of $\boldsymbol{\alpha}$ are greater than or equal to one, then Alg. 5 can be used instead.

4.5 RECURSIVE ADAPTATION

Given the results of the preceding sections, the derivation of the RBRL adaptation procedure is identical to that for RBT (App. C.3), except that we use a summation instead of an integral because the state space is discrete rather than continuous. A block diagram appears in Figure 16, which is analogous to Figure C.4 for RBT.

Algorithm 3. *Apportionment of correct label counts. Whether or not correct truth is assumed is treated as a global setting, as is the set of truthing-error probabilities* $\{p(z_t|y)\}_{t=1}^{N_\mathcal{T}}$. *The set* $\mathcal{T}_m$ *can be determined from* z_m*, so it does not appear as an argument.*

function $\check{\mathbf{y}} \leftarrow$ ApportionLabels$(d, \mathbf{x}, \mathbf{z}, \vec{\boldsymbol{\alpha}})$
  if correct truth **then**
    $\check{\mathbf{y}} \leftarrow \mathbf{z}$
  else
    $M \leftarrow |\mathbf{x}|$ ▷ Number of samples
    for $m \leftarrow 1 : M$ **do** ▷ Loop over samples
      Determine $\mathcal{T}_m$ from z_m ▷ Truther indexes for mth sample
      $Z \leftarrow 0$ ▷ Normalization term
      for $\ell \leftarrow 1 : L$ **do** ▷ Loop over possible labels
        $\check{y}_{m,\ell} \leftarrow \Big(\prod_{t \in \mathcal{T}_m} p(z_{m,t}|y_m = \ell) \Big) \frac{\alpha_{n,k(j(d_n,x_{n,m})),\ell}}{\sum_{\ell'=1}^{L} \alpha_{n,k(j(d_n,x_{n,m})),\ell'}}$ ▷ Eq.(30)
        $Z \leftarrow Z + \check{y}_{m,\ell}$
      end for
      $\check{y}_m \leftarrow (\check{y}_{m,1}/Z, \ldots, \check{y}_{m,L}/Z)$
    end for
    $\check{\mathbf{y}} \leftarrow (\check{y}_1, \ldots, \check{y}_M)$
  end if
end function

Algorithm 4. *Dirichlet hyperparameter update. The decay factor* $0 < \kappa \leq 1$ *defaults to 1 if omitted.*

function $\vec{\boldsymbol{\alpha}}_{n+1} \leftarrow$ UpdateDirichlets$(d, \mathbf{x}, \check{\mathbf{y}}, \mathbf{r}, \vec{\boldsymbol{\alpha}}_n, \kappa = 1)$
  for $k \leftarrow 0 : |\mathcal{A}|$ **do** ▷ The length of $\vec{\boldsymbol{\alpha}}$ is $|\mathcal{A}| + 1$
    if $a_k \in d$ **then** ▷ Eq. (33)
      $\boldsymbol{\alpha}_{n+1,k} \leftarrow \max\{1, \kappa \cdot (\boldsymbol{\alpha}_{n,k} + \mathbf{N}_k(d, \mathbf{x}, \check{\mathbf{y}}) + \mathbf{N}_k(d, \mathbf{r}))\}$
    else
      $\boldsymbol{\alpha}_{n+1,k} \leftarrow \max\{1, \kappa \cdot \boldsymbol{\alpha}_{n,k}\}$
    end if
  end for
end function

Algorithm 5. *Simplified Dirichlet hyperparameter update when the decay factor* $\kappa \equiv 1$ *and all elements of the initial Dirichlet hyperparameter vector* $\boldsymbol{\alpha}$ *are greater than or equal to one.*

function $\vec{\boldsymbol{\alpha}}_{n+1} \leftarrow$ UpdateDirichlets$(d, \mathbf{x}, \check{\mathbf{y}}, \mathbf{r}, \vec{\boldsymbol{\alpha}}_n)$
  $\vec{\boldsymbol{\alpha}}_{n+1} \leftarrow \vec{\boldsymbol{\alpha}}_n$ ▷ Copy hyperparameters
  for $j \leftarrow 0 : |d|$ **do**
    $\boldsymbol{\alpha}_{n+1,k(j)} \leftarrow \boldsymbol{\alpha}_{n,k(j)} + \mathbf{N}_{k(j)}(d, \mathbf{x}, \check{\mathbf{y}}) + \mathbf{N}_{k(j)}(d, \mathbf{r})$ ▷ Eq. (33) for $a_k \in d$ only
  end for
end function

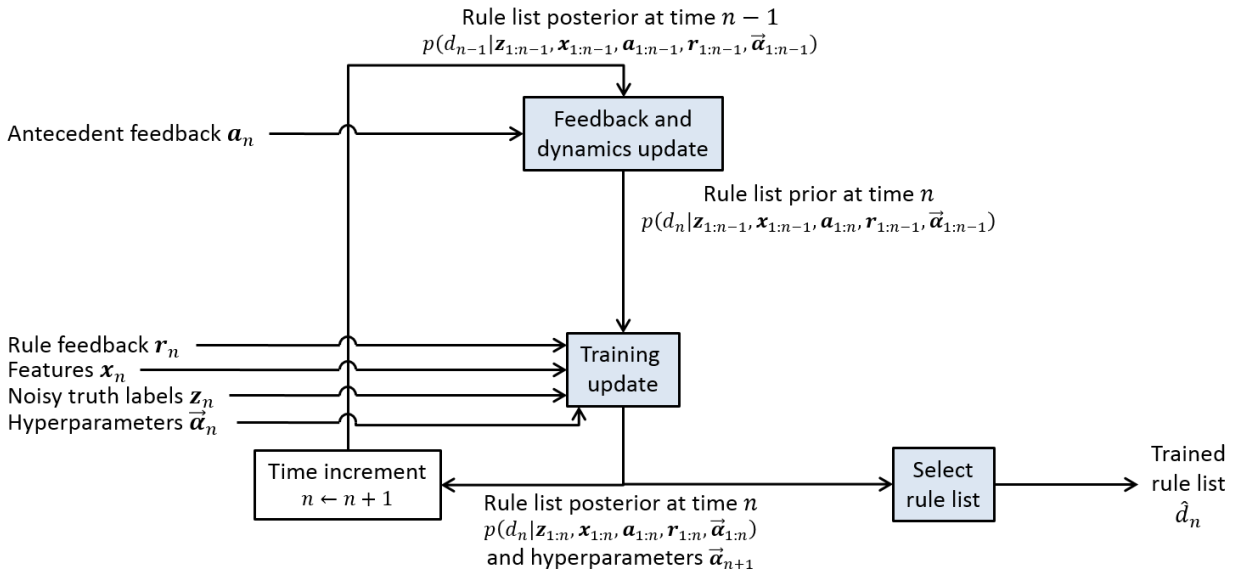

Figure 16. Adaptation procedure in recursive Bayesian rule lists

4.5.1 Feedback and Dynamics Update

The RBRL *feedback and dynamics update* corresponds to (C.9). It includes the new antecedent feedback $\mathbf{a}_n$ and permits the rule list to switch between time $n-1$ and time n. The derivation follows:

$$\begin{aligned}
&\underbrace{p(d_n|\mathbf{z}_{1:n-1}, \mathbf{x}_{1:n-1}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n-1}, \vec{\boldsymbol{\alpha}}_{1:n-1})}_{\text{rule list prior at time } n} \\
&\quad = \sum_{k=1}^{|\mathbf{D}|} p(d_n|d_{n-1}^{(k)}, \mathbf{z}_{1:n-1}, \mathbf{x}_{1:n-1}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n-1}, \vec{\boldsymbol{\alpha}}_{1:n-1}) \\
&\qquad \times p(d_{n-1}^{(k)}|\mathbf{z}_{1:n-1}, \mathbf{x}_{1:n-1}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n-1}, \vec{\boldsymbol{\alpha}}_{1:n-1}) \\
&\quad \overset{(a)}{=} \sum_{k=1}^{|\mathbf{D}|} \underbrace{p(d_n|d_{n-1}^{(k)}, \mathbf{a}_n)}_{\substack{\text{rule list transition} \\ \text{distribution}}} \underbrace{p(d_{n-1}^{(k)}|\mathbf{z}_{1:n-1}, \mathbf{x}_{1:n-1}, \mathbf{a}_{1:n-1}, \mathbf{r}_{1:n-1}, \vec{\boldsymbol{\alpha}}_{1:n-1})}_{\text{rule list posterior at time } n-1},
\end{aligned} \tag{34}$$

where (a) is because (12) applies to the first term and because $d_{n-1}^{(k)}$ does not depend on $\mathbf{a}_n$ in the second term. When $n = 2$, the rule list posterior at time 1 is available from initial BRL training, and when $n > 2$, it is available from the training update at time $n-1$, which is derived next.

4.5.2 Training Update

When the new training data $\mathbf{z}_n$ and $\mathbf{x}_n$ becomes available, RBRL applies its *training update*, which corresponds to (C.10). The derivation is:

$$\begin{aligned}
&\underbrace{p(d_n|\mathbf{z}_{1:n}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n})}_{\text{rule list posterior at time } n} \\
&\overset{(a)}{\propto} p(\mathbf{z}_n|d_n, \mathbf{z}_{1:n-1}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n}) p(d_n|\mathbf{z}_{1:n-1}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n}) \\
&\overset{(b)}{=} \underbrace{p(\mathbf{z}_n|d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)}_{\substack{\text{rule list likelihood} \\ \text{function}}} \underbrace{p(d_n|\mathbf{z}_{1:n-1}, \mathbf{x}_{1:n-1}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n-1}, \vec{\boldsymbol{\alpha}}_{1:n-1})}_{\text{rule list prior at time } n}, \qquad (35)
\end{aligned}$$

where (a) is from Bayes' rule w.r.t. $\mathbf{z}_n$, and (b) is because (27) applies to the first term and (12) applies to the second term. The rule list prior at time n is available from the feedback and dynamics update (34).

If we assume the truth labels are correct ($\mathbf{z}_n \equiv \mathbf{y}_n$) as in Sec. 4.4.1, then we use (21) or (22) for the rule list likelihood function, and we use (23) to update $\vec{\boldsymbol{\alpha}}_n$ for the next time index. If we instead adopt the truthing model in Sec. 4.4.3, then we apply (30) to apportion the unknown correct label occurrences and construct $\check{\mathbf{y}}_n$ from $\mathbf{z}_n$, and we use the approximate likelihood function (31) or (32) and the approximate hyperparameter update (33).

We point out the different ways the antecedent feedback, rule feedback, and features behave, and their relation to RBT. The antecedent feedback $\mathbf{a}_n$ (RBT control input) modifies the rule list (RBT state) through the rule list transition distribution (RBT state transition distribution). It forces a change to the rule list, but it does not influence the correct labels $\mathbf{y}_n$, so it does not have a pseudocount interpretation. In contrast, the rule feedback $\mathbf{r}_n$ (RBT direct-feed control input) drives the patterns and correct labels $\mathbf{y}_n$ (RBT system output), rather than the rule list. It indirectly affects the rule list through the likelihood function via artificial occurrences of patterns and correct labels, so it has the Dirichlet-multinomial pseudocount interpretation. Finally, the observed features $\mathbf{x}_n$ (another RBT direct-feed control input) act in the same manner as the rule feedback, but they are actual observations of patterns and associated correct labels $\mathbf{y}_n$, rather than pseudo-observations.

4.6 POSTERIOR PREDICTIVE

In ordinary RBT, once the state estimate has been computed, the answer is ready. In RBRL, the selected rule list $\hat{d}_n$ only contains antecedents; to classify an unlabeled feature vector $\tilde{x}_n$ by using $\hat{d}_n$ to predict $\tilde{y}_n$, the Dirichlet-multinomial consequents are needed, so the posterior predictive is required.

If the truth labels are assumed to be correct ($\mathbf{z}_n \equiv \mathbf{y}_n$), then the posterior predictive for a point estimate $\hat{d}_n$ is the same as (10) with rule feedback included:

$$\begin{aligned} p(\tilde{y}_n = \ell | \tilde{x}_n, \hat{d}_n, \mathbf{x}_{1:n}, \mathbf{y}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n}) \\ \stackrel{(a)}{=} p(\tilde{y}_n = \ell | \tilde{x}_n, \hat{d}_n, \mathbf{x}_n, \mathbf{y}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n) \\ = \frac{\alpha_{n,k(j(\hat{d}_n,\tilde{x}_n)),\ell} + N_{j(\hat{d}_n,\tilde{x}_n),\ell}(\hat{d}_n, \mathbf{x}_n, \mathbf{y}_n) + N_{j(\hat{d}_n,\tilde{x}_n),\ell}(\hat{d}_n, \mathbf{r}_n)}{\sum_{\ell'=1}^{L} \alpha_{n,k(j(\hat{d}_n,\tilde{x}_n)),\ell'} + N_{j(\hat{d}_n,\tilde{x}_n),\ell'}(\hat{d}_n, \mathbf{x}_n, \mathbf{y}_n) + N_{j(\hat{d}_n,\tilde{x}_n),\ell'}(\hat{d}_n, \mathbf{r}_n)}, \quad \ell = 1, \ldots, L, \end{aligned} \tag{36}$$

where (a) is from (27). If we use the model of Sec. 4.4.3, then we just replace $\mathbf{y}_n$ with $\check{\mathbf{y}}_n$ in this equation.

This page intentionally left blank.

5. GRID-BASED RBRL (GB-RBRL)

Grid-based RBRL (GB-RBRL) is theoretically optimal for a finite state space. It maintains the set $\mathbf{D}$ of all possible rule lists and updates the posterior probability of each one as more data becomes available. In most cases, $\mathbf{D}$ will be too large to make GB-RBRL tractable, so in practice, GB-RBRL will operate over $\mathcal{D}$, the set of rule lists from initial BRL training, instead of $\mathbf{D}$. This section covers the GB-RBRL algorithm because it could be useful as an optimal reference and because it includes the static case in which there are no dynamics.

5.1 RECURSION

The derivation of the GB-RBRL recursion is exactly the same as in [22, Sec. III-B], so we only present the main results. GB-RBRL maintains three sets of corresponding items:

- The set of sampled rule lists $\mathcal{D} = \{d^{(i)}\}_{i=1}^{|\mathcal{D}|}$ from initial BRL training.
- The corresponding set of non-negative weights $\vec{\mathcal{W}}_n = \{w_n^{(i)}\}_{i=1}^{|\mathcal{D}|}$. Each $w_n^{(i)} \propto p(d_n^{(i)}|\mathbf{z}_{1:n}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n}^{(i)})$, and the weights are normalized to sum to one.
- The corresponding set of Dirichlet hyperparameters $\vec{\mathcal{A}}_n = \{\vec{\boldsymbol{\alpha}}_n^{(i)}\}_{i=1}^{|\mathcal{D}|}$.

At time $n = 1$, regular BRL training is performed. It provides $\hat{d}_1$, $\mathcal{D}$, $\vec{\mathcal{W}}_1$, and $\vec{\mathcal{A}}_2$. For $n \geq 2$, GB-RBRL implements the recursion in Sec. 4.5 directly to update $\vec{\mathcal{W}}_n$ and $\vec{\mathcal{A}}_{n+1}$ from $\vec{\mathcal{W}}_{n-1}$, $\vec{\mathcal{A}}_n$, $\mathbf{z}_n$, $\mathbf{x}_n$, $\mathbf{a}_n$, and $\mathbf{r}_n$. Pseudocode appears in Alg. 6.

The rule list posterior at time n is

$$p(d_n|\mathbf{z}_{1:n}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n}) \approx \sum_{i=1}^{|\mathcal{D}|} w_n^{(i)} I(d_n = d_n^{(i)}),$$

where $I(x)$ is the *indicator function*: $I(x) = 1$ if x is true, and $I(x) = 0$ if not. Hence, $\mathcal{D}$ and $\vec{\mathcal{W}}_n$ can be used to compute BRL-point. The pseudocode returns $\hat{d}_n$ and its associated hyperparameters so the posterior predictive can be computed.

The loop that calculates $\tilde{w}^{(i)}$ requires a summation over all rule lists in $\mathcal{D}$:

$$\tilde{w}^{(i)} \leftarrow \sum_{j=1}^{|\mathcal{D}|} w_{n-1}^{(j)} p(d_n^{(i)}|\tilde{d}^{(j)}), \quad i = 1, 2, \ldots, |\mathcal{D}|,$$

so a simplistic implementation of GB-RBRL has complexity $O(|\mathcal{D}|^2)$. However, if only small rule list changes are possible between adjacent time steps, then for each $d^{(i)}$, there will only be a small number of intermediate rule lists that can transition to it. A more efficient implementation could

Algorithm 6. *Grid-Based Recursive Bayesian Rule Lists (GB-RBRL). The decay factor $0 < \kappa \leq 1$ defaults to one if omitted.*

function $\left[\hat{d}_n, \hat{\vec{\boldsymbol{\alpha}}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1}\right] \leftarrow$ GB-RBRL$(\mathcal{D}, \vec{\mathcal{W}}_{n-1}, \vec{\mathcal{A}}_n, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \kappa = 1)$
  for $j \leftarrow 1 : |\mathcal{D}|$ **do** ▷ Antecedent feedback
    $\tilde{d}^{(j)} \leftarrow f_a(d_{n-1}^{(j)}, \mathbf{a}_n)$ ▷ cf. (16)
  end for
  for $i \leftarrow 1 : |\mathcal{D}|$ **do** ▷ Feedback and dynamics update
    $\tilde{w}^{(i)} \leftarrow \sum_{j=1}^{|\mathcal{D}|} w_{n-1}^{(j)} p(d_n^{(i)} | \tilde{d}^{(j)})$ ▷ cf. (18) and (34)
  end for
  for $i \leftarrow 1 : |\mathcal{D}|$ **do** ▷ Training update (unnormalized)
    $\check{\mathbf{y}}^{(i)} \leftarrow$ ApportionLabels$(d_n^{(i)}, \mathbf{x}_n, \mathbf{z}_n, \vec{\boldsymbol{\alpha}}_n^{(i)})$
    $w_n^{(i)} \leftarrow \tilde{w}^{(i)} \underbrace{p(\check{\mathbf{y}}^{(i)} | d_n^{(i)}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{(i)})}_{\substack{\text{rule list likelihood} \\ \text{function}}}$ ▷ cf. (35)
  end for
  $Z \leftarrow \sum_{i=1}^{|\mathcal{D}|} w_n^{(i)}$ ▷ Normalization term
  for $i \leftarrow 1 : |\mathcal{D}|$ **do**
    $w_n^{(i)} \leftarrow w_n^{(i)} / Z$ ▷ Normalize and update weights $\vec{\mathcal{W}}_n$
    $\vec{\boldsymbol{\alpha}}_{n+1}^{(i)} \leftarrow$ UpdateDirichlets$(d_n^{(i)}, \mathbf{x}_n, \check{\mathbf{y}}^{(i)}, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{(i)}, \kappa)$ ▷ Update hyperparameters $\vec{\mathcal{A}}_{n+1}$
  end for
  $\left[\hat{d}_n, \hat{\imath}\right] \leftarrow$ BRLPoint$(\mathcal{D}, \vec{\mathcal{W}}_n)$
  $\hat{\vec{\boldsymbol{\alpha}}}_n \leftarrow \vec{\boldsymbol{\alpha}}_n^{(\hat{\imath})}$ ▷ Hyperparameters for $\hat{d}_n$
end function

reduce computations by pre-computing the set $\tilde{\mathcal{D}}^{(i)} = \{\tilde{d}^{(j)} \in \mathcal{D} : p(d^{(i)}|\tilde{d}^{(j)}) > 0\}$ and a look-up table with $p(d^{(i)}|\tilde{d}^{(j)})$. Then

$$\tilde{w}^{(i)} \leftarrow \sum_{\tilde{d}^{(j)} \in \tilde{\mathcal{D}}^{(i)}} w_{n-1}^{(j)} p(d_n^{(i)}|\tilde{d}^{(j)}), \quad i = 1, 2, \ldots, |\mathcal{D}|.$$

Since $|\tilde{D}^{(i)}| \ll |\mathcal{D}|$, $\forall i$, complexity could be reduced to $O(\max\{|\tilde{\mathcal{D}}^{(i)}|\}_{i=1}^{\mathcal{D}} \cdot |\mathcal{D}|) \ll O(|\mathcal{D}|^2)$.

It might also appear that $\vec{\mathcal{A}}_n$ must store $|\mathcal{D}|(|\mathcal{A}| + 1)$ Dirichlet parameter vectors, each of length L. However, the rule lists are fixed, so for each i, $\vec{\boldsymbol{\alpha}}_n^{(i)}$ only needs to store the Dirichlet parameters for the antecedents in $d^{(i)}$ and a_0.

5.2 STATIC-CASE RBRL (SC-RBRL)

An important special case of GB-RBRL is the static case from Sec. 4.3.2: There is no antecedent feedback $\mathbf{a}_n$ and no process noise. This case corresponds to the common situation in which one assumes that the generative model does not change, and one simply acquires more data over time and adjusts the posterior probabilities of the possible rule lists at each time step.

From (20), the feedback and dynamics update can be skipped, which gives the *static-case RBRL* (SC-RBRL) algorithm in Alg. 7. SC-RBRL has complexity $O(|\mathcal{D}|)$. Even though each rule list is assumed not to change, the decay factor κ can be used to attenuate the past data and give SC-RBRL a crude ability to address a slowly-changing environment.

Algorithm 7. *Static-Case Recursive Bayesian Rule Lists (SC-RBRL). The decay factor $0 < \kappa \leq 1$ defaults to one if omitted.*

function $\left[\hat{d}_n, \hat{\vec{\boldsymbol{\alpha}}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1}\right] \leftarrow$ SC-RBRL$(\mathcal{D}, \vec{\mathcal{W}}_{n-1}, \vec{\mathcal{A}}_n, \mathbf{z}_n, \mathbf{x}_n, \mathbf{r}_n, \kappa = 1)$
  for $i \leftarrow 1 : |\mathcal{D}|$ **do** ▷ Training update only (unnormalized)
    $\check{\mathbf{y}}^{(i)} \leftarrow$ ApportionLabels$(d_n^{(i)}, \mathbf{x}_n, \mathbf{z}_n, \vec{\boldsymbol{\alpha}}_n^{(i)})$
    $w_n^{(i)} \leftarrow w_{n-1}^{(i)} \underbrace{p(\check{\mathbf{y}}^{(i)}|d_n^{(i)}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{(i)})}_{\substack{\text{rule list likelihood} \\ \text{function}}}$ ▷ cf. (35)
  end for
  $Z \leftarrow \sum_{i=1}^{|\mathcal{D}|} w_n^{(i)}$ ▷ Normalization term
  for $i \leftarrow 1 : |\mathcal{D}|$ **do**
    $w_n^{(i)} \leftarrow w_n^{(i)}/Z$ ▷ Normalize and update weights $\vec{\mathcal{W}}_n$
    $\vec{\boldsymbol{\alpha}}_{n+1}^{(i)} \leftarrow$ UpdateDirichlets$(d_n^{(i)}, \mathbf{x}_n, \check{\mathbf{y}}^{(i)}, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{(i)}, \kappa)$ ▷ Update hyperparameters $\vec{\mathcal{A}}_{n+1}$
  end for
  $\left[\hat{d}_n, \hat{\imath}\right] \leftarrow$ BRLPoint$(\mathcal{D}, \vec{\mathcal{W}}_n)$
  $\hat{\vec{\boldsymbol{\alpha}}}_n \leftarrow \vec{\boldsymbol{\alpha}}_n^{(\hat{\imath})}$ ▷ Hyperparameters for $\hat{d}_n$
end function

This page intentionally left blank.

6. PARTICLE-FILTER RBRL (PF-RBRL)

This section shows how RBRL can be implemented as a *particle filter* (PF). Particle filtering is a *sequential Monte Carlo* (SMC) method [22] [23] [42]; it is one of the most successful nonlinear, non-Gaussian tracking methods available, is broadly used, and has a well-studied theory. Whereas a grid-based method like GB-RBRL maintains a fixed set of distinct sample states (rule lists) and updates their probabilities over time, PF methods maintain a collection of (possibly duplicated) sample states, each of which evolves through the dynamical model, and importance sampling is used to approximate the state distribution (rule list distribution).

6.1 REVIEW

We quickly review some aspects of PF in the usual RBT context with a continuous state space. PF maintains a set of N_p *particles* $\{s'^{[i]}_n\}_{i=1}^{N_p}$ and weights $\{w^{[i]}_n\}_{i=1}^{N_p}$, where $s'^{[i]}_n$ is a sample state and $w^{[i]}_n \geq 0$ is its associated weight. We use superscript bracketed indexes instead of parenthesized indexes because there could be multiple copies of the same state; that is, $s'^{[i]}_n$ and $s'^{[j]}_n$ could be identical, although their weights might differ.

First, the most basic PF form is *sequential importance sampling* (SIS) [22, Sec. V-A & Alg. 1] [23], which draws new particles from the *importance density* $q(s'_n|s'_{n-1}, u'_n)$ and updates the weights via [22, Eq. (48)]

$$w^{[i]}_n = w^{[i]}_{n-1} \frac{p(z'_n|s'^{[i]}_n, u'_n) p(s'^{[i]}_n|s'^{[i]}_{n-1}, u'_n)}{q(s'^{[i]}_n|s'^{[i]}_{n-1}, u'_n)}. \tag{37}$$

For background, pseudocode appears in Alg. 8, although we will not use SIS because it suffers from *degeneracy*: after a small number of cycles, only one particle will have non-negligible weight, and all other particles will have weights close to zero.

Algorithm 8. *Sequential Importance Sampling*

function $\left[\hat{s'}_n, \{s'^{[i]}_n\}_{i=1}^{N_p}, \{w^{[i]}_n\}_{i=1}^{N_p}\right] \leftarrow \text{SIS}(\{s'^{[i]}_n\}_{i=1}^{N_p}, \{w^{[i]}_n\}_{i=1}^{N_p}, u'_n)$
  for $i \leftarrow 1 : N_p$ **do** ▷ Importance sampling
    Draw $s'^{[i]}_n \sim q(s'_n|s'^{[i]}_{n-1}, u'_n)$
    $w^{[i]}_n \leftarrow w^{[i]}_{n-1} \dfrac{p(z'_n|s'^{[i]}_n, u'_n) p(s'^{[i]}_n|s'^{[i]}_{n-1}, u'_n)}{q(s'^{[i]}_n|s'^{[i]}_{n-1}, u_n)}$ ▷ Eq. (37)
  end for
  $\hat{s}'_n \leftarrow \text{SelectEstimate}(\{s'^{[i]}_n, w^{[i]}_n\}_{i=1}^{N_p})$ ▷ e.g.: conditional mean or MAP estimate
end function

Degeneracy is measured by the *effective sample size* N_{eff} [22] [23], which reflects the fact that the variance of an estimate formed from the N_p particles is approximately equivalent to that formed from N_{eff} samples drawn from the actual distribution. In practice, N_{eff} is approximated by

the *estimated effective sample size* $\hat{N}_{\text{eff}}$, which lies between 1 and N_p [22, Eq. (51)]:

$$\hat{N}_{\text{eff}} = \left(\sum_{i=1}^{N_p} w_n^{[i]} \right)^{-1}. \tag{38}$$

Second, most PF forms perform *resampling* to protect against degeneracy. When $\hat{N}_{\text{eff}}$ falls below a pre-defined threshold N_T, the particles are resampled according to the distribution (39), and all weights are reset to $w_n^{[i]} = 1/N_p$. $N_T = N_p/2$ is suggested in [23]. A significant amount of research in particle filtering is concerned with resampling methods [43].

Third, the resampled particles can become highly redundant, a phenomenon known *sample impoverishment*. Since particles are resampled according to the posterior, sample states with large weights are drawn more frequently than those with small weights, and the resampled particles lose diversity [22, Sec. V-A3] [23, Sec. 3.5]. Sample impoverishment is particularly problematic when the process noise is small [22, Sec. V-A3]. A variety of techniques have been proposed to combat sample impoverishment; they include the resample-move algorithm [44], the auxiliary PF [45], and the regularized PF [46].

Resampling and the variations above allow PF to protect against the effects of degeneracy and sample impoverishment. PF represents some of the most successful non-linear and/or non-Gaussian tracking algorithms, and the generic PF (described next for RBRL) works well for many problems. Under certain mixing conditions, the bias and variance of the PF estimation error can be uniformly bounded with n [23, Secs. 3.6 & 3.7].

6.2 GENERIC ALGORITHM

Particle-filter RBRL (PF-RBRL) offers a practical, suboptimal SMC tracking algorithm. It maintains:

- A set of *particles* $\vec{\mathcal{D}}_n = \{d_n^{[i]}\}_{i=1}^{N_p}$, where $d_n^{[i]}$ is a candidate rule list that can evolve over time.
- The corresponding set of non-negative weights $\vec{\mathcal{W}}_n = \{w_n^{[i]}\}_{i=1}^{N_p}$.
- The corresponding set of Dirichlet hyperparameters $\vec{\mathcal{A}}_n = \{\vec{\boldsymbol{\alpha}}_n^{[i]}\}_{i=1}^{N_p}$.

We re-use the symbols $\vec{\mathcal{W}}_n$ and $\vec{\mathcal{A}}_n$ from GB-RBRL, but we use superscript bracketed indexes instead of parenthesized indexes because $d_n^{[i]}$ and $d_n^{[j]}$ could be identical, unlike in the grid-based method. The rule list posterior at time n is approximated by [cf. (C.11)]

$$p(d_n | \mathbf{z}_{1:n}, \mathbf{x}_{1:n}, \mathbf{a}_{1:n}, \mathbf{r}_{1:n}, \vec{\boldsymbol{\alpha}}_{1:n}) \approx \sum_{i=1}^{N_p} w_n^{[i]} I(d_n = d_n^{[i]}). \tag{39}$$

Given the analogy between RBT and RBRL, derivation of PF-RBRL follows exactly the steps in [22, Sec. V-A]. At time $n = 1$, regular BRL training produces a set $\mathcal{D}$ of candidate rule lists, their

posterior probabilities, and their Dirichlet hyperparameter sets. The initial sets $\vec{\mathcal{D}}_1$, $\vec{\mathcal{W}}_1$, and $\vec{\mathcal{A}}_2$ can be created by replicating the BRL outputs C times and renormalizing the posterior probabilities, which means $N_p = C|\mathcal{D}|$. Alternatively, $\vec{\mathcal{D}}_1$, $\vec{\mathcal{W}}_1$, and $\vec{\mathcal{A}}_2$ can be initialized by sampling N_p times from the BRL outputs according to the BRL-generated posterior probabilities.

Alg. 9 gives the generic PF-RBRL algorithm, which is based on [22, Alg. 3] and [23, SIR/SMC for Filtering]. The algorithm relies upon an *importance distribution* $q(d_n|d_{n-1}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n)$, which must be specified and from which it must be possible to draw new rule lists:

$$d_n^{[i]} \sim q(d_n|d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}). \tag{40}$$

This sampling step provides PF-RBRL with a way to introduce new antecedents and rule lists.

The weight update is the same as (37):

$$w_n^{[i]} \propto w_{n-1}^{[i]} \frac{p(\mathbf{z}_n|d_n^{[i]}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) p(d_n^{[i]}|d_{n-1}^{[i]}, \mathbf{a}_n)}{q(d_n^{[i]}|d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}. \tag{41}$$

The algorithm substitutes $\check{\mathbf{y}}_n$ for $\mathbf{z}_n$ into the rule list likelihood function in case the truthing model (Sec. 4.4.3) is used.

The trained rule list $\hat{d}_n$ is selected based on the approximate posterior (39). There can be multiple particles for the same rule list, so the algorithm uses Alg. 10 to aggregate the weights and hyperparameters for each unique rule list before computing BRL-point. It returns the aggregated hyperparameters for $\hat{d}_n$ for computing the posterior predictive.

If $\hat{N}_{\text{eff}}$ falls below the threshold N_T, indicating potential degeneracy, then the algorithm resamples the candidate rule lists and resets the weights to $w_n^{[i]} = 1/N_p$, $i = 1, \ldots, N_p$. A simple resampling algorithm is given in [22, Alg. 2]; it has been modified to include the hyperparameters $\{\vec{\boldsymbol{\alpha}}_{n+1}^{[i]}\}_{i=1}^{N_p}$ and is included as Alg. 11.

6.3 OPTIMAL IMPORTANCE DISTRIBUTION

Selection of a good importance distribution is a critical element of successful PF development, and it must be possible to draw samples from the chosen distribution. The optimal importance distribution maximizes the effective sample size, and it is given by [22, Eq. (52)]:

$$\begin{aligned} q_{\text{opt}}(d_n|d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) &= p(d_n|d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) \\ &\overset{(a)}{=} \frac{p(\mathbf{z}_n|d_n, d_{n-1}^{[i]}, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) p(d_n|d_{n-1}^{[i]}, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}{p(\mathbf{z}_n|d_{n-1}^{[i]}, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})} \\ &\overset{(b)}{=} \frac{\overbrace{p(\mathbf{z}_n|d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}^{\substack{\text{rule list} \\ \text{likelihood function}}} \overbrace{p(d_n|d_{n-1}^{[i]}, \mathbf{a}_n)}^{\substack{\text{rule list transition} \\ \text{distribution}}}}{p(\mathbf{z}_n|d_{n-1}^{[i]}, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}, \end{aligned} \tag{42}$$

Algorithm 9. *Generic Particle-Filter Recursive Bayesian Rule Lists (PF-RBRL). The decay factor* $0 < \kappa \leq 1$ *defaults to one if omitted.*

function $\left[\hat{d}_n, \hat{\vec{\boldsymbol{\alpha}}}_n, \vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1}\right] \leftarrow$ PF-RBRL$(\vec{\mathcal{D}}_{n-1}, \vec{\mathcal{W}}_{n-1}, \vec{\mathcal{A}}_n, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \kappa = 1)$
  for $i \leftarrow 1 : N_p$ **do** ▷ Importance sampling
    $\check{\mathbf{y}}^{[i]} \leftarrow$ ApportionLabels$(d_n^{[i]}, \mathbf{x}_n, \mathbf{z}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})$
    Draw $d_n^{[i]} \sim q(d_n | d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})$ ▷ Eq. (40)
    $w_n^{[i]} \leftarrow w_{n-1}^{[i]} \dfrac{\overbrace{p(\check{\mathbf{y}}^{[i]} | d_n^{[i]}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}^{\text{rule list likelihood function}} \overbrace{p(d_n^{[i]} | d_{n-1}^{[i]}, \mathbf{a}_n)}^{\text{rule list transition distribution}}}{q(d_n^{[i]} | d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}$ ▷ Eq. (41)
  end for
  $Z \leftarrow \sum_{i=1}^{N_p} w_n^{[i]}$ ▷ Normalization term
  for $i \leftarrow 1 : N_p$ **do**
    $w_n^{[i]} \leftarrow w_n^{[i]} / Z$ ▷ Normalize and update weights $\vec{\mathcal{W}}_n$
    $\vec{\boldsymbol{\alpha}}_{n+1}^{[i]} \leftarrow$ UpdateDirichlets$(d_n^{[i]}, \mathbf{x}_n, \check{\mathbf{y}}^{[i]}, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}, \kappa)$ ▷ Update hyperparameters $\vec{\mathcal{A}}_{n+1}$
  end for
  $\left[\vec{\mathcal{D}}', \vec{\mathcal{W}}', \vec{\mathcal{A}}'\right] \leftarrow$ AggregateParticles$(\vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1})$
  $\left[\hat{d}_n, \hat{\imath}\right] \leftarrow$ BRLPoint$(\vec{\mathcal{D}}', \vec{\mathcal{W}}')$
  $\hat{\vec{\boldsymbol{\alpha}}}_n \leftarrow \hat{\imath}$th element of $\vec{\mathcal{A}}'$ ▷ Hyperparameters for $\hat{d}_n$
  $\hat{N}_{\text{eff}} \leftarrow \left(\sum_{i=1}^{N_p} w_n^{[i]}\right)^{-1}$ ▷ Eq. (38)
  if $\hat{N}_{\text{eff}} < N_T$ **then**
    $\left[\vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1}\right] \leftarrow$ Resample$(\vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1})$
  end if
end function

Algorithm 10. *Particle Aggregation.* FindUniqueParticles *returns the set of unique particles and, for each unique particle, the indexes of the particles in* $\vec{\mathcal{D}}$ *that match it.*

function $\left[\vec{\mathcal{D}}', \vec{\mathcal{W}}', \vec{\mathcal{A}}'\right] \leftarrow$ AggregateParticles$(\vec{\mathcal{D}}, \vec{\mathcal{W}}, \vec{\mathcal{A}})$
  $[\vec{\mathcal{D}}', \mathcal{I}] \leftarrow$ FindUniqueParticles$(\vec{\mathcal{D}})$
  for $j \leftarrow 1 : |\vec{\mathcal{D}}'|$ **do** ▷ Note that $\mathcal{I}_j = \{i : d^{[i]} \in \vec{\mathcal{D}}, d^{[i]} = d^{(j)} \in \vec{\mathcal{D}}'\}$ below
    $w'^{(j)} \leftarrow \sum_{i \in \mathcal{I}_j} w^{[i]}$ ▷ Aggregate weights $\vec{\mathcal{W}}'$
    $\vec{\boldsymbol{\alpha}}^{(j)} \leftarrow (w'^{(j)})^{-1} \sum_{i \in \mathcal{I}_j} w^{[i]} \vec{\boldsymbol{\alpha}}^{[i]}$ ▷ Aggregate hyperparameters $\vec{\mathcal{A}}'$
  end for
end function

Algorithm 11. *Resampling. This algorithm is from [22, Alg. 2] and includes the Dirichlet hyperparameters.*

function $\left[\{d_n^{[j]}\}_{j=1}^{N_p}, \{w_n^{[j]}\}_{j=1}^{N_p}, \{\vec{\boldsymbol{\alpha}}_{n+1}^{[j]}\}_{j=1}^{N_p}\right] \leftarrow$ RESAMPLE$(\{d_n^{[i]}\}_{j=1}^{N_p}, \{w_n^{[i]}\}_{j=1}^{N_p}, \{\vec{\boldsymbol{\alpha}}_{n+1}^{[i]}\}_{i=1}^{N_p})$
  $c_1 \leftarrow 0$ ▷ Initialize the CDF
  for $i \leftarrow 2 : N_p$ **do**
    $c_i \leftarrow c_{i-1} + w_n^{[i]}$ ▷ Construct CDF
  end for
  $i \leftarrow 1$ ▷ Start at the bottom of the CDF
  Draw $u_1 \sim \text{Uniform}[0, 1/N_p]$ ▷ Draw a starting point
  for $j \leftarrow 1 : N_p$ **do**
    $u_j \leftarrow u_1 + (j-1)/N_p$ ▷ Move along the CDF
    while $u_j > c_i$ **do**
      $i \leftarrow i + 1$
    end while
    $d_n^{[j]} \leftarrow d_n^{[i]}$ ▷ Assign sample
    $w_n^{[j]} \leftarrow 1/N_p$ ▷ Assign weight
    $\vec{\boldsymbol{\alpha}}_{n+1}^{[j]} \leftarrow \vec{\boldsymbol{\alpha}}_{n+1}^{[i]}$ ▷ Assign hyperparameters
  end for
end function

where (a) is Bayes' rule, and (b) is from (12) and (24). The numerator is the product of the rule list likelihood function and the rule list transition distribution.

For a continuous state space, the denominator requires integration and usually cannot be computed except for a few special cases [22]. For a discrete state space like RBRL, the denominator can be calculated as

$$p(\mathbf{z}_n|d_{n-1}^{[i]}, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) = \sum_{i'=1}^{N_p} p(\mathbf{z}_n|d_n^{[i']}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) p(d_n^{[i']}|d_{n-1}^{[i]}, \mathbf{a}_n), \quad i = 1, \ldots, N_p. \tag{43}$$

RBRL has a finite state space, so at least in theory, the optimal importance distribution could be employed. However, evaluating (43) for all particles has complexity $O(N_p^2)$, which could be impractical.

6.4 SAMPLING IMPORTANCE RESAMPLING RBRL (SIR-RBRL)

One popular PF form is the *sampling importance resampling* (SIR) filter, originally called the "bootstrap filter" [42] [22, Sec. V-B1]. It uses the rule list transition distribution $p(d_n|d_{n-1}, \mathbf{a}_n)$ as the importance distribution, so

$$q_{\text{SIR}}(d_n|d_{n-1}^{[i]}, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}) = p(d_n|d_{n-1}^{[i]}, \mathbf{a}_n). \tag{44}$$

Comparing this equation with (42), we see that the SIR filter neglects the influence of the rule list likelihood function $p(\mathbf{z}_n|d_n, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})$. However, as a practical matter, from Sec. 4.3, we

can readily sample from this importance distribution by first computing $\tilde{d}_n = f_a(d_{n-1}^{[i]}, \mathbf{a}_n)$, then drawing a process noise sample $\mathbf{v}$ distributed $p(\mathbf{v}_{n-1})$, and finally computing $d_n^{[i]} = f_v(\tilde{d}_n, \mathbf{v})$.

The weight update (41) becomes

$$w_n^{[i]} \propto w_{n-1}^{[i]} p(\mathbf{z}_n | d_n^{[i]}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}). \tag{45}$$

Alg. 12 presents the *sampling importance resampling RBRL* (SIR-RBRL) algorithm. It resamples when $\hat{N}_{\text{eff}}$ becomes sufficiently small, although the original SIR filter resamples on every iteration

The SIR importance distribution ignores $\mathbf{z}_n$ and $\mathbf{x}_n$, so the SIR filter "can be inefficient and is sensitive to outliers" [22, Sec. V-B1]. However, the complexity of the SIR filter is $O(N_p)$ rather than $O(N_p^2)$, which makes it tractable. In addition, the SIR filter has only mild requirements: One mainly needs to be able to draw realizations of the process noise and evaluate the likelihood function. The next section describes a process-noise sampling technique.

Algorithm 12. *Sampling Importance Resampling Recursive Bayesian Rule Lists (SIR-RBRL). The decay factor $0 < \kappa \leq 1$ defaults to one if omitted.*

function $\left[\hat{d}_n, \hat{\vec{\boldsymbol{\alpha}}}_n, \vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1}\right] \leftarrow$ SIR-RBRL$(\vec{\mathcal{D}}_{n-1}, \vec{\mathcal{W}}_{n-1}, \vec{\mathcal{A}}_n, \mathbf{z}_n, \mathbf{x}_n, \mathbf{a}_n, \mathbf{r}_n, \kappa = 1)$
  for $i \leftarrow 1 : N_p$ **do** ▷ Importance sampling
    $\check{\mathbf{y}}^{[i]} \leftarrow$ ApportionLabels$(d_n^{[i]}, \mathbf{x}_n, \mathbf{z}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})$
    $\tilde{d}_n \leftarrow f_a(d_{n-1}^{[i]}, \mathbf{a}_n)$ ▷ Eq. (13)
    Draw $\mathbf{v} \sim p(\mathbf{v}_{n-1} | \tilde{d}_n)$ ▷ See Alg. 13 or 14, for example
    $d_n^{[i]} \leftarrow f_v(\tilde{d}_n, \mathbf{v})$ ▷ Eq. (14)
    $w_n^{[i]} \leftarrow w_{n-1}^{[i]} \underbrace{p(\check{\mathbf{y}}^{[i]} | d_n^{[i]}, \mathbf{x}_n, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]})}_{\substack{\text{rule list likelihood} \\ \text{function}}}$ ▷ Eq. (45)
  end for
  $Z \leftarrow \sum_{i=1}^{N_p} w_n^{[i]}$ ▷ Normalization term
  for $i \leftarrow 1 : N_p$ **do**
    $w_n^{[i]} \leftarrow w_n^{[i]} / Z$ ▷ Normalize and update weights $\vec{\mathcal{W}}_n$
    $\vec{\boldsymbol{\alpha}}_{n+1}^{[i]} \leftarrow$ UpdateDirichlets$(d_n^{[i]}, \mathbf{x}_n, \check{\mathbf{y}}^{[i]}, \mathbf{r}_n, \vec{\boldsymbol{\alpha}}_n^{[i]}, \kappa)$ ▷ Update hyperparameters $\vec{\mathcal{A}}_{n+1}$
  end for
  $\left[\vec{\mathcal{D}}', \vec{\mathcal{W}}', \vec{\mathcal{A}}'\right] \leftarrow$ AggregateParticles$(\vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1})$
  $\left[\hat{d}_n, \hat{\imath}\right] \leftarrow$ BRLPoint$(\vec{\mathcal{D}}', \vec{\mathcal{W}}')$
  $\hat{\vec{\boldsymbol{\alpha}}}_n \leftarrow \hat{\imath}$th element of $\vec{\mathcal{A}}'$ ▷ Hyperparameters for $\hat{d}_n$
  $\hat{N}_{\text{eff}} \leftarrow \left(\sum_{i=1}^{N_p} w_n^{[i]}\right)^{-1}$ ▷ Eq. (38)
  if $\hat{N}_{\text{eff}} < N_T$ **then**
    $\left[\vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1}\right] \leftarrow$ Resample$(\vec{\mathcal{D}}_n, \vec{\mathcal{W}}_n, \vec{\mathcal{A}}_{n+1})$
  end if
end function

6.5 PROCESS-NOISE SAMPLING

Some approaches for sampling the process noise follow.

6.5.1 Independent Process Noise

The process noise $\mathbf{v}_{n-1}$ can be sampled in a straightforward manner by re-purposing the BRL MCMC proposals (move, add, remove) in Sec. 3.5. One possibility is to define $\mathbf{v}_{n-1}$ to be a sequence of N MCMC proposals, where $N \sim \text{Geometric}(\nu)$, and let each proposal element $\mathbf{v}_{n-1}(m)$, $m = 1, \ldots, N$, be uniformly distributed over {move, add, remove}. Then

$$\begin{aligned} p(\mathbf{v}_{n-1}|\nu) &= p(N|\nu) \prod_{m=1}^{N} p(\mathbf{v}_{n-1}(m)) \\ &= (1-\nu)^N \nu \times \prod_{m=1}^{N} (1/3). \end{aligned}$$

Here, the geometric RV has parameter $\nu \in [0, 1]$ and gives the number of failures prior to the first success for a series of independent Bernoulli(ν) trials. The distribution has support $\{0, 1, \ldots\}$, and larger values of ν make smaller values of N more probable than large ones. The distribution could also be truncated to limit the maximum length of $\mathbf{v}_{n-1}$. If a maximum length of one is desired, then a Bernoulli distribution could replace the geometric distribution.

A simple implementation is shown in Alg. 13. When $d_n = f_v(\tilde{d}_n, \mathbf{v}_{n-1})$ is computed, it applies the proposals in $\mathbf{v}_{n-1}$ sequentially. If a proposal is impossible (e.g., if the rule list becomes empty and the next proposal is "remove"), then it is skipped.

Algorithm 13. *Independent process-noise sampling using MCMC proposals*

function $\mathbf{v} \leftarrow$ SAMPLEPROCESSNOISE(ν)
  Draw $N \sim \text{Geometric}(\nu)$ ▷ Number of changes
  for $i \leftarrow 1 : N$ **do**
    Draw $p \sim \text{Categorical}((1/3, 1/3, 1/3))$ ▷ Equiprobable over {move, add, remove}
    if $p =$ move **then**
      $\mathbf{v}(i) \leftarrow$ move
    else if $p =$ add **then**
      $\mathbf{v}(i) \leftarrow$ add
    else
      $\mathbf{v}(i) \leftarrow$ remove
    end if
  end for
end function

6.5.2 Dependent Process Noise

The above model makes the distribution of $\mathbf{v}_{n-1}$ independent of the intermediate rule list $\tilde{d}_n$. It does not impose any constraints on the length or average cardinality of the resulting rule list d_n, so over time the rule list might become too complicated to interpret and might overfit the training data.

These problems can be addressed by making $\mathbf{v}_{n-1}$ dependent on $\tilde{d}_n$ [cf. Sec. 4.3.1] and the BRL parameters λ, η, and $\boldsymbol{\beta}$. Alg. 14 describes one possible procedure for doing so. Rather than explicitly sampling $\mathbf{v}_{n-1}$ and passing it into $f_v(\tilde{d}_n, \mathbf{v}_{n-1})$, it computes d_n directly given $\tilde{d}_n$, so it would replace Lines 5 and 6 in Alg. 12. This approach is shown by the dashed lines in Figure 12 and Figure 13. The models do not show $\tilde{d}_n$, but they show that $\mathbf{v}_{n-1}$ depends on d_{n-1} and $\mathbf{a}_{n-1}$, which together determine $\tilde{d}_n$.

The algorithm uses a geometric RV to sample the number of changes to make to $\tilde{d}_n$. For each change, it samples a list length using the same term from the BRL rule list prior (3). The difference between the sampled list length and the current list length is passed into a logistic function to select the probabilities of drawing "add" and "remove" changes. If the difference is negative, then "add" is favored; if it is positive, then "remove" is favored; and if it is zero, then these changes are equiprobable. The logistic function includes a non-negative parameter ζ; if $\zeta = 0$, then "add" and "remove" remain equiprobable regardless of the sample list length. The remaining probability is reserved for the "move" change.

Next, the change is sampled from a categorical RV with support over {move, add, remove}. If "move" or "remove" is drawn, then it is applied equiprobably over the antecedents in the current rule list. If "add" is drawn, then the cardinality and antecedent terms from the BRL rule list prior are applied to favor sampling an antecedent with the desired cardinality.

This processing encourages, but does not force, the rule list to conform to the parameters λ, η, and $\boldsymbol{\beta}$; these parameters could be made time-varying if desired.

6.6 REMARKS

Some remarks on GB-RBRL and PF-RBRL are merited.

6.6.1 Remarks on GB-RBRL

GB-RBRL is theoretically optimal if it operates over the entire set of rule lists $\mathbf{D}$. However, in practice it must operate over $\mathcal{D}$, the set of sampled rule lists. It does not suffer from degeneracy or sample impoverishment. As mentioned in Sec. 5.1, it has general complexity $O(|\mathcal{D}|^2)$, which could make it impractical, but when only small changes to a rule list can occur, the complexity could be greatly reduced.

GB-RBRL also includes the important static case (Sec. 4.3.2 and Sec. 5.2), in which the rule list does not change over time. Then GB-RBRL reduces to SC-RBRL (Alg. 7), which has complexity $O(|\mathcal{D}|)$. Particle filtering is not appropriate for the static case because the problem

Algorithm 14. *Applying dependent process-noise sampling. The parameter ζ must be non-negative.*

function $d \leftarrow$ ApplyDependentProcessNoise$(\tilde{d}, \nu, \mathcal{A}, \lambda, \eta, \boldsymbol{\beta}, \zeta)$
  $d \leftarrow \tilde{d}$
  Draw $N \sim \text{Geometric}(\nu)$ ▷ Number of changes
  for $i \leftarrow 1 : N$ **do**
    Draw $m \sim p(m|\mathcal{A}, \lambda) = \frac{(\lambda^m/m!)}{\sum_{j=0}^{|\mathcal{A}|}(\lambda^j/j!)}$ ▷ List-length term from BRL prior (3)
    $s \leftarrow \frac{1}{1+e^{-\zeta(m-|d|)}}$ ▷ Logistic function
    $\boldsymbol{\theta}_{\text{change}} \leftarrow \left(\frac{1}{3}, \frac{2}{3}s, \frac{2}{3}(1-s)\right)$ ▷ Categorical parameters over {move, add, remove}
    Draw $p \sim \text{Categorical}(\boldsymbol{\theta}_{\text{change}})$ ▷ Sample the change
    if $p =$ move **then**
      Draw j uniformly over $\{1, 2, \ldots, |d|\}$
      Draw j' uniformly over $\{1, 2, \ldots, |d|\} \setminus \{j\}$
      $d \leftarrow$ swap jth and j'th antecedents in d
    else if $p =$ add **then**
      Draw j uniformly over $\{1, 2, \ldots, |d| + 1\}$
      Draw $c \sim \frac{(\eta^c/c!)}{\sum_{c' \in \mathcal{C}(a_{k(<j)})}(\eta^{c'}/c'!)}$ ▷ Cardinality from BRL prior (3)
      Draw $a_{k'} \sim \frac{\Gamma(\sum_{k \in \mathcal{K}(a_{k(<j)}, c_{k(j)})} \beta_k)}{\Gamma(1+\sum_{k \in \mathcal{K}(a_{k(<j)}, c_{k(j)})} \beta_k)} \frac{1+\Gamma(\beta_{k(j)})}{\Gamma(\beta_{k(j)})}$ ▷ Antecedent from BRL prior (3)
      $d \leftarrow$ insert $a_{k'}$ before jth index of d
    else
      Draw j uniformly over $\{1, 2, \ldots, |d|\}$
      $d \leftarrow$ remove jth antecedent from d
    end if
  end for
end function

is actually parameter estimation rather than tracking, and it will experience degeneracy and/or sample impoverishment.

6.6.2 Remarks on PF-RBRL

If one assumes that the rule list can change over time, then PF-RBRL becomes suitable and may be preferable to GB-RBRL. Particle filtering actually requires a sufficient amount of process noise to work (Sec. 6.1); if the process noise is too small, then sample impoverishment can occur.[14] The SIR-RBRL version of PF-RBRL has complexity $O(N_p)$, and process-noise sampling can be realized by re-purposing the BRL MCMC proposals or components of the BRL rule list prior.

Although we have assumed that the antecedent set $\mathcal{A}$ and sampled rule list set $\mathcal{D}$ remain constant, they can easily be allowed to vary over time. For example, dynamic frequent pattern mining could discover new antecedents, so $\mathcal{A}$ could become time-varying. The importance sampling and process-noise sampling steps in PF-RBRL could introduce the new antecedents and rule lists. Of course, the user-specified parameters λ, η, and $\boldsymbol{\beta}$, and the other parameters like κ, ν, and ζ could also be made time-varying. It is not immediately clear how GB-RBRL could support these capabilities.

Next, RBT typically assumes a continuous state space and uses mean-squared error as a distance measure. RBRL uses a finite, and hence discrete, state space, and we have not defined a distance measure for rule lists.[15] RBT PF algorithms like the auxiliary PF [45] and regularized PF [46] were developed for a continuous state space [23], so they might not carry over to PF-RBRL. Similarly, PF convergence results [47] [48] have assumed a continuous state space, so they might not apply directly to PF-RBRL. For example, the study of bias and variance in PF requires the definition of a distance metric. This consideration is similar to the remark by Letham *et al.* in [12, Sec. 2.7] that it is difficult to interpret the conditional mean of a collection of sample rule lists.

Finally, sample impoverishment is a concern in conventional PF, but the discrete nature of RBRL might make it less of a concern in PF-RBRL. To maintain diversity, it might suffice to resample and then perturb the resampled rule lists, which might be similar to the resample-move algorithm [44].

[14] This situation is reminiscent of the linear Gaussian case, in which the Kalman filter is asymptotically unbiased if the process noise sufficiently excites the dynamics model [20] [21].

[15] Levenshtein distance or a similar edit distance could likely be adopted as a rule list distance measure.

7. SUMMARY

User trust in automation and machine learning depends on several factors. This work has emphasized three important factors: good performance, interpretability, and adaptability. These factors serve the goal of the Adaptable Interpretable Machine Learning (AIM) program: To create ML algorithms that users can understand and that keep learning so users will trust them and actually use them.

We have further focused on supervised classification with Bayesian Rule Lists (BRL), a recently-published, human-understandable decision-list classifier that is competitive with state-of-the-art, non-interpretable classifiers on many problems. We sought to enable BRL to adapt to new data and user feedback. We identified a powerful analogy between BRL adaptation and recursive Bayesian tracking (RBT). This analogy, called Recursive Bayesian Rule Lists (RBRL), has several appealing properties, some of which appear to be novel.

7.1 RBRL PROPERTIES

Some key properties of RBRL follow.

7.1.1 Feature and Label User Feedback

BRL's exploitation of pre-mined patterns means that, when providing truth labels for new training data, a user can tell RBRL which patterns led him or her to choose those labels. As a result, the user can potentially correct cases in which the classifier got "the right answer but for the wrong reason."

For conventional classifier adaptation methods like relevance feedback or active learning, such feedback is impossible: the user can only indicate the correct labels, but there is no way for the user to indicate the pattern that led him or her to make that decision. The user can only hope that the classifier figures out the relationship. Of course, RBRL also allows conventional, label-only feedback.

This capability allows the user to encourage preferred rule lists over competing ones. Letham *et al.* point out that BRL can produce many rule lists with nearly the same performance [12, Sec. 5]; user feedback can separate rule lists that contain preferred patterns from the rest.

Also, the rule feedback is incorporated into RBRL via Dirichlet-multinomial distributions, so it can be expressed in terms of pseudocounts or pseudo-observations. For example, the user might have years of experience, but the training data might only have been collected recently. The user can provide pseudocounts that reflect cases that are not sufficiently represented in the training data.

7.1.2 Efficient Adaptation

The RBRL adaptation procedure is highly efficient. First, it is recursive, in the sense that it only needs the new data and user feedback and the most recent set of candidate rule lists. There

is no need to store or accumulate past data and feedback. This aspect can make RBRL suitable for streaming applications, and it can give it a fixed memory and computational footprint.

Second, adaptation in RBRL is non-iterative. Regular BRL training uses Markov Chain Monte Carlo (MCMC) sampling, which requires iterating a Markov chain many times until it converges to its stationary distribution. Anywhere from 10^4 to 10^6 or more iterations might be needed. In contrast, RBRL adaptation occurs in a single pass.

Third, most of the RBRL calculations are highly parallelizable. The normalization and resampling operations can be more complicated to parallelize, but they can occur in the background while a user is examining RBRL's results. For many applications, they can likely be performed much faster than new data and user feedback are acquired.

7.1.3 Dataset Shift

RBRL can include a model for dataset shift, which is analogous to the state dynamics model in RBT. Antecedent feedback acts like a control input that changes the rule list. Along with process noise, it affects the rule list prior through the transition distribution. It does not have a pseudocount interpretation.

7.1.4 Multiple Noisy Truth Labels

RBRL can also handle noisy truth labels from multiple truthers, provided that the conditional distributions of the truthers' labels given the correct label can be estimated (e.g., with the EM algorithm). This property results from the analogy between the correct labels and the pristine system output in RBT and from the analogy between the noisy truth labels and the noisy observations in RBT.

Rule feedback and new features act like direct-feed control inputs; they affect the correct labels and affect the rule list posterior through the likelihood function. The Dirichlet-multinomial consequent gives the rule feedback a pseudocount interpretation.

7.1.5 Interactive Machine Learning

The classifiers generated by RBRL can be understood by users, the users can provide feedback on features and labels, and adaptation can be computed in an efficient, highly parallel manner. Although RBRL algorithms remain to be implemented in FY18, the combination of the above properties suggest that RBRL should be well-suited for interactive machine learning applications.

7.2 RBRL ALGORITHMS

The RBT analogy also means that several algorithms for RBRL are readily available. We have described two main types of algorithms: grid-based RBRL (GB-RBRL) and particle-filter RBRL (PF-RBRL).

GB-RBRL is theoretically optimal, but the state space of rule lists is so large that practical implementations will be suboptimal. GB-RBRL includes the special static-case RBRL (SC-RBRL)

algorithm, which does not support antecedent feedback or dataset shift. However, SC-RBRL corresponds to the common situation where one assumes that there is no dataset shift but one merely acquires more and more data over time. SC-RBRL can also apply an exponential-decay factor to offer a crude way to adjust to a changing environment.

PF-RBRL includes a generic algorithm that requires an importance distribution. For RBRL, the optimal importance distribution can theoretically be computed, but doing so is likely to be impractical. The sampling importance resampling RBRL (SIR-RBRL) algorithm offers a straightforward, efficient, suboptimal algorithm. It can incorporate the dataset-shift model, and the resampling step could allow new frequent patterns and candidate rule lists to be introduced over time.

7.3 PLANS AND FUTURE WORK

Our FY17 work on AIM has laid the foundation for implementing RBRL algorithms. In FY18, we plan to complete implementations of SC-RBRL and SIR-RBRL. We will conduct experiments on several real data sets to demonstrate the various RBRL capabilities and measure performance.

Some other potential areas for future work follow. First, RBRL is well-suited to support interactive machine learning. It also offers offer new visualization and human-machine interface (HMI) opportunities. In fact, RBRL can potentially go beyond the "interface" in HMI because the classifiers—not just the interfaces—are designed with the user in mind. These possibilities might also be explored in FY18 or in future work.

Second, RBRL can provide users with a best counter-argument against the current prediction.[16] This capability is enabled by the collection of candidate rule lists that RBRL already maintains. Likewise, separate instances of RBRL—one with and one without user feedback—could be used to illustrate the separate influence of user feedback.

Finally, it would be worthwhile to look for importance distributions that might improve upon SIR-RBRL. It would also be interesting to study some of the particle-filter convergence results and investigate if they can be modified for RBRL.

[16] Credit for this idea goes to Cynthia Rudin.

This page intentionally left blank.

APPENDIX A: REVIEW OF RELEVANT RANDOM VARIABLES

A.1 CATEGORICAL AND MULTINOMIAL

A.1.1 Categorical

Without loss of generality, a *categorical* RV over L categories is a scalar RV that takes values in $\{1, 2, \ldots, L\}$. Its distribution is parameterized by a vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_L)$ with $\theta_\ell \geq 0$, $\forall \ell$, and $\sum_{\ell=1}^{L} \theta_\ell = 1$. For $y \sim \text{Categorical}(\boldsymbol{\theta})$, the PMF is

$$p(y|\boldsymbol{\theta}) = \prod_{\ell=1}^{L} \theta_\ell^{I(\ell=y)} = \theta_y, \quad y = 1, 2, \ldots, L, \tag{A.1}$$

and $p(y|\boldsymbol{\theta}) = 0$ for $y \notin \{1, 2, \ldots, L\}$. When $L = 2$, the categorical RV reduces to the Bernoulli RV.

The result of M independent Categorical($\boldsymbol{\theta}$) trials is $\mathbf{y} = (y_1, y_2, \ldots, y_M)$, a vector of length M, where each element y_m is an integer giving the category chosen on the mth categorical trial. A graphical model is shown in Figure A.1. The PMF of $\mathbf{y}$ is

$$\begin{aligned} p(\mathbf{y}|\boldsymbol{\theta}) &= \prod_{\ell=1}^{L} \theta_\ell^{N_\ell(\mathbf{y})} \\ &= \prod_{m=1}^{M} \prod_{\ell=1}^{L} \theta_\ell^{I(y_m=\ell)} \\ &= \prod_{\ell=1}^{L} \prod_{m=1}^{M} \theta_\ell^{I(y_m=\ell)} \\ &\overset{(a)}{=} \prod_{\ell=1}^{L} \theta_\ell^{N_\ell(\mathbf{y})}, \end{aligned} \tag{A.2}$$

where in (a), $N_\ell(\mathbf{y})$ is the number of times category ℓ occurs in $\mathbf{y}$, and

$$\sum_{\ell=1}^{L} N_\ell(\mathbf{y}) = M. \tag{A.3}$$

A.1.2 Multinomial

A *multinomial* RV $\vec{y} \sim \text{Multinomial}(M, \boldsymbol{\theta})$ is a length-L random vector $\vec{y} = (\vec{y}_1, \vec{y}_2, \ldots, \vec{y}_L)$, where $\vec{y}_\ell$ is an integer with the number of occurrences of the ℓth category in M independent Categorical($\boldsymbol{\theta}$) trials. Its distribution is parameterized by M, the number of categorical trials, and $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_L)$, the parameter vector for the Categorical($\boldsymbol{\theta}$) RVs. Figure A.2 displays a graphical model. The multinomial PMF is

$$p(\vec{y}|M, \boldsymbol{\theta}) = \binom{M}{\vec{y}_1 \quad \vec{y}_2 \quad \cdots \quad \vec{y}_L} \theta_1^{\vec{y}_1} \theta_2^{\vec{y}_2} \cdots \theta_L^{\vec{y}_L} = \frac{M!}{\vec{y}_1! \vec{y}_2! \cdots \vec{y}_L!} \prod_{\ell=1}^{L} \theta_\ell^{\vec{y}_\ell}, \tag{A.4}$$

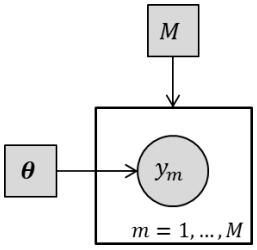

Figure A.1. Independent categorical RVs graphical model: M independent Categorical($\boldsymbol{\theta}$) trials

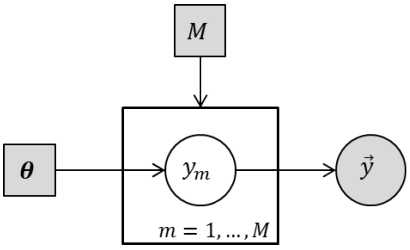

Figure A.2. Multinomial graphical model: Single Multinomial($M, \boldsymbol{\theta}$) trial

where $\vec{y}_\ell \in \{0, 1, \ldots, M\}$, $\forall \ell$, and $\sum_{\ell=1}^{L} \vec{y}_\ell = M$. When $L = 2$, the multinomial RV reduces to the binomial RV.

A.1.3 Relationship

The categorical and multinomial RVs are closely related, as evident from Figures A.1 and A.2 and the PMFs in (A.2) and (A.4). The difference is whether one observes M individual categorical RVs ($\mathbf{y}$) or the number of occurrences ($\vec{y}$) of each category.

Since the multinomial RV summarizes the individual categorical RVs, its distribution must account for their possible combinations, which leads to the multinomial coefficient in (A.4) that is absent from (A.2). In addition, the length of $\mathbf{y}$ varies with M while the length of $\vec{y}$ is always L.

When $M = 1$, the Categorical($\boldsymbol{\theta}$) RV $\mathbf{y}$ is a scalar $\mathbf{y} = y \in \{1, \ldots, L\}$. The corresponding Multinomial($1, \boldsymbol{\theta}$) RV $\vec{y}$ is a length-L vector of the form $(0, \ldots, 0, 1, 0, \ldots, 0)$, which has a one at the yth element (i.e., $\vec{y}_y = 1$) and zeros at all other elements ($\vec{y}_\ell = 0$, $\ell \neq y$); this form of $\vec{y}$ is called *one-hot encoding*. In this case, both PMFs evaluate to θ_y, so y and $\vec{y}$ are essentially equivalent except that one is a scalar, and the other is a one-hot encoded vector. For these reasons, the terms "categorical" and "multinomial" are sometimes used interchangeably, and "M independent Multinomial($\boldsymbol{\theta}$)" RVs actually refers to M independent Categorical($\boldsymbol{\theta}$) or Multinomial($1, \boldsymbol{\theta}$) RVs.

A.2 DIRICHLET

A *Dirichlet* RV over the open $(L-1)$-dimensional simplex is a length-L random vector $\boldsymbol{\theta} = (\theta_1, \theta_2, \ldots, \theta_L)$, where $\theta_\ell \geq 0$, $\forall \ell$, and $\sum_{\ell=1}^{L} \theta_\ell = 1$. Its distribution is parameterized by a parameter vector $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_L)$ with $\alpha_\ell > 0$, $\ell = 1, \ldots, L$. The elements of $\boldsymbol{\alpha}$ are known as "concentration parameters."

The PDF of $\boldsymbol{\theta} \sim \text{Dirichlet}(\boldsymbol{\alpha})$ is [33] [32]

$$p(\boldsymbol{\theta}|\boldsymbol{\alpha}) = \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \theta_1^{\alpha_1 - 1} \theta_2^{\alpha_2 - 1} \cdots \theta_L^{\alpha_L - 1} = \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)}{\prod_{\ell=1}^{L} \Gamma(\alpha_\ell)} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell - 1}, \quad \theta_\ell \geq 0, \forall \ell; \sum_{\ell=1}^{L} \theta_\ell = 1; \qquad \text{(A.5)}$$

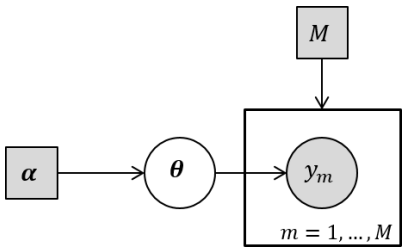

Figure A.3. Dirichlet-categorical graphical model: M independent Categorical($\boldsymbol{\theta}$) trials with $\boldsymbol{\theta} \sim$ Dirichlet($\boldsymbol{\alpha}$)

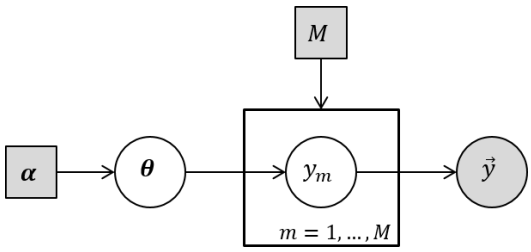

Figure A.4. Dirichlet-multinomial graphical model: Single Multinomial(M, $\boldsymbol{\theta}$) trial with $\boldsymbol{\theta} \sim$ Dirichlet($\boldsymbol{\alpha}$)

where $\mathrm{B}(\boldsymbol{\alpha}) = \big(\prod_{\ell=1}^{L} \Gamma(\alpha_\ell)\big)/\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)$ is the multivariate beta function. When $\boldsymbol{\alpha} = (1, \ldots, 1)$, the PDF becomes $p(\boldsymbol{\theta}|\boldsymbol{\alpha} = (1, \ldots, 1)) = \Gamma(L) = (L-1)!$, which is the uniform distribution over the $(L-1)$-dimensional simplex. When $L = 2$, the Dirichlet RV reduces to the beta RV.

A useful relationship is derived next. Since every PDF integrates to one, for $\boldsymbol{\theta} \sim \mathrm{Dirichlet}(\boldsymbol{\xi})$ with $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_L)$,

$$\begin{aligned} 1 &= \int p(\boldsymbol{\theta}|\boldsymbol{\xi})\, d\boldsymbol{\theta} \\ &= \frac{\Gamma(\sum_{\ell=1}^{L} \xi_\ell)}{\prod_{\ell=1}^{L} \Gamma(\xi_\ell)} \int \prod_{\ell=1}^{L} \theta_\ell^{\xi_\ell - 1}\, d\boldsymbol{\theta}, \end{aligned}$$

which gives

$$\int \prod_{\ell=1}^{L} \theta_\ell^{\xi_\ell - 1}\, d\boldsymbol{\theta} = \frac{\prod_{\ell=1}^{L} \Gamma(\xi_\ell)}{\Gamma(\sum_{\ell=1}^{L} \xi_\ell)}. \tag{A.6}$$

A.3 DIRICHLET-CATEGORICAL

The *Dirichlet-categorical* RV is a length-M vector $\mathbf{y}$ of M independent Categorical($\boldsymbol{\theta}$) trials with $\boldsymbol{\theta}|\boldsymbol{\alpha} \sim \mathrm{Dirichlet}(\boldsymbol{\alpha})$. It is a compound distribution since $\boldsymbol{\theta}$ is itself a Dirichlet($\boldsymbol{\alpha}$) RV, and $\boldsymbol{\alpha}$ is sometimes called a "hyperparameter" or "hyperparameters." Figure A.3 shows a graphical model for the Dirichlet-categorical RV. When $L = 2$, the Dirichlet-categorical distribution reduces to the beta-Bernoulli distribution.

Because of the mixing of the terms "categorical" and "multinomial," the Dirichlet-categorical RV is sometimes referred to as a "Dirichlet-multinomial RV." This report uses the Dirichlet-categorical RV, but in keeping with the terminology of Letham *et al.* [12], the majority of the report refers to it as a "Dirichlet-multinomial RV." The Dirichlet-multinomial RV with its stricter meaning is covered in App. A.4.

A.3.1 Probability Mass Function

The Dirichlet-categorical PMF is obtained by marginalizing over $\boldsymbol{\theta}$:

$$\begin{aligned}
p(\mathbf{y}|\boldsymbol{\alpha}) &= \int p(\mathbf{y}|\boldsymbol{\theta}, \boldsymbol{\alpha}) p(\boldsymbol{\theta}|\boldsymbol{\alpha})\, d\boldsymbol{\theta} \\
&= \int p(\mathbf{y}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha})\, d\boldsymbol{\theta} \\
&\overset{(a)}{=} \int \left(\prod_{\ell=1}^{L} \theta_\ell^{N_\ell(\mathbf{y})} \right) \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)}{\prod_{\ell=1}^{L} \Gamma(\alpha_\ell)} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell - 1}\, d\boldsymbol{\theta} \\
&= \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)}{\prod_{\ell=1}^{L} \Gamma(\alpha_\ell)} \int \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell + N_\ell(\mathbf{y}) - 1}\, d\boldsymbol{\theta} \\
&\overset{(b)}{=} \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)}{\prod_{\ell=1}^{L} \Gamma(\alpha_\ell)} \frac{\prod_{\ell=1}^{L} \Gamma(\alpha_\ell + N_\ell(\mathbf{y}))}{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell + N_\ell(\mathbf{y}))} \\
&= \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)}{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell + N_\ell(\mathbf{y}))} \prod_{\ell=1}^{L} \frac{\Gamma(\alpha_\ell + N_\ell(\mathbf{y}))}{\Gamma(\alpha_\ell)} \qquad \text{(A.7)} \\
&\overset{(c)}{=} \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell)}{\Gamma(M + \sum_{\ell=1}^{L} \alpha_\ell)} \prod_{\ell=1}^{L} \frac{\Gamma(\alpha_\ell + N_\ell(\mathbf{y}))}{\Gamma(\alpha_\ell)}, \qquad \text{(A.8)}
\end{aligned}$$

where (a) is from (A.2) and (A.5), (b) is from (A.6) with $\xi_\ell = \alpha_\ell + N_\ell(\mathbf{y})$, and (c) is from (A.3).

If $M = 1$ so that $\mathbf{y} = y = \ell$, this expression becomes

$$\begin{aligned}
p(y = \ell|\boldsymbol{\alpha}) &= \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'})}{\Gamma(1 + \sum_{\ell'=1}^{L} \alpha_{\ell'})} \frac{\Gamma(\alpha_\ell + 1) \prod_{\ell'=1, \ell' \neq \ell}^{L} \Gamma(\alpha_{\ell'})}{\prod_{\ell'=1}^{L} \Gamma(\alpha_{\ell'})} \\
&\overset{(a)}{=} \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'})}{(\sum_{\ell'=1}^{L} \alpha_{\ell'}) \cdot \Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'})} \frac{\alpha_\ell \cdot \Gamma(\alpha_\ell)}{\Gamma(\alpha_\ell)} \\
&= \frac{\alpha_\ell}{\sum_{\ell'=1}^{L} \alpha_{\ell'}}, \qquad \text{(A.9)}
\end{aligned}$$

where (a) is because $\Gamma(x+1) = x \cdot \Gamma(x)$, $x > 0$.

A.3.2 Posterior for Categorical Parameters

By definition, the prior distribution of $\boldsymbol{\theta}|\boldsymbol{\alpha}$ is Dirichlet($\boldsymbol{\alpha}$). Given $\mathbf{y}$, the posterior distribution of $\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\alpha}$ is

$$\begin{aligned} p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\alpha}) &\overset{(a)}{\propto} p(\mathbf{y}|\boldsymbol{\theta}, \boldsymbol{\alpha}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}) \\ &= p(\mathbf{y}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\boldsymbol{\alpha}) \\ &\overset{(b)}{=} \left(\prod_{\ell=1}^{L} \theta_\ell^{N_\ell(\mathbf{y})} \right) \frac{1}{\mathrm{B}(\boldsymbol{\alpha})} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell - 1} \\ &\propto \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell + N_\ell(\mathbf{y}) - 1}, \end{aligned}$$

where (a) is Bayes' rule, and (b) is because the likelihood function for $\boldsymbol{\theta}$ is (A.2), viewed as a function of $\boldsymbol{\theta}$ rather than $\mathbf{y}$, and because of (A.5).

The posterior distribution must integrate to one, and from (A.6), the necessary normalization factor is $\Gamma(\sum_{\ell=1}^{L} \alpha_\ell + N_\ell(\mathbf{y})) / \big(\prod_{\ell=1}^{L} \Gamma(\alpha_\ell + N_\ell(\mathbf{y})) \big)$. Hence,

$$p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\alpha}) = \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_\ell + N_\ell(\mathbf{y}))}{\prod_{\ell=1}^{L} \Gamma(\alpha_\ell + N_\ell(\mathbf{y}))} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell + N_\ell(\mathbf{y}) - 1} \tag{A.10}$$

$$= \frac{1}{\mathrm{B}(\boldsymbol{\alpha} + \mathbf{N}(\mathbf{y}))} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell + N_\ell(\mathbf{y}) - 1}, \tag{A.11}$$

where $\mathbf{N}(\mathbf{y}) = (N_1(\mathbf{y}), \ldots, N_L(\mathbf{y}))$.

As a result, the posterior is another Dirichlet distribution: $\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha} + \mathbf{N}(\mathbf{y}))$. The posterior Dirichlet parameters are just the prior Dirichlet parameters $\boldsymbol{\alpha}$ plus the occurrence counts $\mathbf{N}(\mathbf{y})$ for the observed data $\mathbf{y}$. For this reason, the prior Dirichlet parameters $\boldsymbol{\alpha}$ can themselves be interpreted as "pseudocounts" [32], or counts of postulated pseudo-observations made prior to the observations $\mathbf{y}$.

Finally, since the prior and posterior distributions of $\boldsymbol{\theta}$ are both Dirichlet distributions, they are said to be "conjugate distributions" [32]. The Dirichlet prior is called the "conjugate prior" of the likelihood function $p(\mathbf{y}|\boldsymbol{\theta})$.

A.3.3 Posterior Predictive

Given $\mathbf{y}$, the *posterior predictive* describes the distribution of $\tilde{\mathbf{y}}|\mathbf{y}, \boldsymbol{\alpha}$ for a vector $\tilde{\mathbf{y}} = (\tilde{y}_1, \ldots, \tilde{y}_{\tilde{M}})$ of $\tilde{M}$ independent Categorical($\boldsymbol{\theta}$) RVs, with $\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha} + \mathbf{N}(\mathbf{y}))$. It is obtained by

marginalizing over $\boldsymbol{\theta}$:

$$
\begin{aligned}
p(\tilde{\mathbf{y}}|\mathbf{y},\boldsymbol{\alpha}) &= \int p(\tilde{\mathbf{y}}|\mathbf{y},\boldsymbol{\theta},\boldsymbol{\alpha}) p(\boldsymbol{\theta}|\mathbf{y},\boldsymbol{\alpha})\, d\boldsymbol{\theta} \\
&= \int p(\tilde{\mathbf{y}}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y},\boldsymbol{\alpha})\, d\boldsymbol{\theta} \\
&\overset{(a)}{=} \int \left(\prod_{\ell'=1}^{L} \theta_{\ell'}^{N_{\ell'}(\tilde{\mathbf{y}})} \right) \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\prod_{\ell'=1}^{L} \Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \prod_{\ell'=1}^{L} \theta_{\ell'}^{\alpha_{\ell'}+N_{\ell'}(\mathbf{y})-1}\, d\boldsymbol{\theta} \\
&= \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\prod_{\ell'=1}^{L} \Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \int \prod_{\ell'=1}^{L} \theta_{\ell'}^{\alpha_{\ell'}+N_{\ell'}(\mathbf{y})+N_{\ell'}(\tilde{\mathbf{y}})-1}\, d\boldsymbol{\theta} \\
&\overset{(b)}{=} \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\prod_{\ell'=1}^{L} \Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \frac{\prod_{\ell'=1}^{L} \Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{\mathbf{y}}))}{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{\mathbf{y}}))} \\
&= \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{\mathbf{y}}))} \prod_{\ell'=1}^{L} \frac{\Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{\mathbf{y}}))}{\Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \qquad \text{(A.12)} \\
&\overset{(c)}{=} \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\Gamma(\tilde{M} + M + \sum_{\ell'=1}^{L} \alpha_{\ell'})} \prod_{\ell'=1}^{L} \frac{\Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{\mathbf{y}}))}{\Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}, \qquad \text{(A.13)}
\end{aligned}
$$

where (a) is from (A.2) and (A.10), and (b) is from (A.6) with $\xi_{\ell'} = \alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{\mathbf{y}})$, and (c) is from (A.3). These expressions resemble the PMF forms (A.7) and (A.8).

For the special case $\tilde{M} = 1$, $\tilde{\mathbf{y}} = \tilde{y} = \ell \in \{1, \ldots, L\}$, and (A.12) becomes

$$
\begin{aligned}
p(\tilde{y} = \ell|\mathbf{y},\boldsymbol{\alpha}) &= \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{y}))} \prod_{\ell'=1}^{L} \frac{\Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}) + N_{\ell'}(\tilde{y}))}{\Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \\
&\overset{(a)}{=} \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\Gamma(1 + \sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \frac{\Gamma(\alpha_{\ell} + N_{\ell}(\mathbf{y}) + 1) \prod_{\ell'=1, \ell' \neq \ell}^{L} \Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\prod_{\ell'=1}^{L} \Gamma(\alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \\
&= \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{\Gamma(1 + \sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \frac{\Gamma(\alpha_{\ell} + N_{\ell}(\mathbf{y}) + 1)}{\Gamma(\alpha_{\ell} + N_{\ell}(\mathbf{y}))} \\
&\overset{(b)}{=} \frac{\Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))}{(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y})) \cdot \Gamma(\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y}))} \frac{(\alpha_{\ell} + N_{\ell}(\mathbf{y})) \cdot \Gamma(\alpha_{\ell} + N_{\ell}(\mathbf{y}))}{\Gamma(\alpha_{\ell} + N_{\ell}(\mathbf{y}))} \\
&= \frac{\alpha_{\ell} + N_{\ell}(\mathbf{y})}{\sum_{\ell'=1}^{L} \alpha_{\ell'} + N_{\ell'}(\mathbf{y})} \qquad \text{(A.14)} \\
&\overset{(c)}{=} \frac{\alpha_{\ell} + N_{\ell}(\mathbf{y})}{M + \sum_{\ell'=1}^{L} \alpha_{\ell'}}, \qquad \text{(A.15)}
\end{aligned}
$$

where (a) is because $N_{\ell}(\tilde{y}) = 1$, and $N_{\ell'}(\tilde{y}) = 0$, $\ell' \neq \ell$; (b) applies the relation $\Gamma(x+1) = x \cdot \Gamma(x)$, $x > 0$; and (c) is from (A.3).

Similarly, if no observations $\mathbf{y}$ are available, then $p(\tilde{\mathbf{y}}|\boldsymbol{\alpha})$ is the *prior predictive*; it is just the PMF (A.7), (A.8), or (A.9) evaluated at $\mathbf{y} = \tilde{\mathbf{y}}$.

A.4 DIRICHLET-MULTINOMIAL

The *Dirichlet-multinomial* RV is a length-L vector $\vec{y} \sim \text{Multinomial}(M, \boldsymbol{\theta})$ with $\boldsymbol{\theta}|\boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha})$. It is a compound distribution, and a graphical model appears in Figure A.4. When $L = 2$, it reduces to the beta-binomial distribution. For completeness, expressions for the Dirichlet-multinomial RV $\vec{y}$ are given here, without derivation.

The PMF is

$$p(\vec{y}|\boldsymbol{\alpha}) = \frac{(M!)\,\Gamma(\sum_{\ell=1}^{L}\alpha_\ell)}{\Gamma(\sum_{\ell=1}^{L}\alpha_\ell + \vec{y}_\ell)} \prod_{\ell=1}^{L} \frac{\Gamma(\alpha_\ell + \vec{y}_\ell)}{(\vec{y}_\ell!)\,\Gamma(\alpha_\ell)} \tag{A.16}$$

$$= \frac{(M!)\,\Gamma(\sum_{\ell=1}^{L}\alpha_\ell)}{\Gamma(M + \sum_{\ell=1}^{L}\alpha_\ell)} \prod_{\ell=1}^{L} \frac{\Gamma(\alpha_\ell + \vec{y}_\ell)}{(\vec{y}_\ell!)\,\Gamma(\alpha_\ell)}. \tag{A.17}$$

When $M = 1$, $\vec{y}$ one-hot encodes the ℓth category, and the PMF becomes

$$p(\vec{y} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{=1 \text{ for } \ell\text{th element only}} |\boldsymbol{\alpha}) = \frac{\alpha_\ell}{\sum_{\ell'=1}^{L}\alpha_{\ell'}}. \tag{A.18}$$

The posterior distribution $\boldsymbol{\theta}|\vec{y}, \boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha} + \vec{y})$:

$$p(\boldsymbol{\theta}|\mathbf{y}, \boldsymbol{\alpha}) = \frac{\Gamma(\sum_{\ell=1}^{L}\alpha_\ell + \vec{y}_\ell)}{\prod_{\ell=1}^{L}\Gamma(\alpha_\ell + \vec{y}_\ell)} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell + \vec{y}_\ell - 1} \tag{A.19}$$

$$= \frac{1}{\text{B}(\boldsymbol{\alpha} + \vec{y})} \prod_{\ell=1}^{L} \theta_\ell^{\alpha_\ell + \vec{y}_\ell - 1}. \tag{A.20}$$

The prior and posterior are again conjugate, and the Dirichlet prior is also the conjugate prior of the likelihood function $p(\vec{y}|\boldsymbol{\theta})$ [cf. (A.4)].

The posterior predictive for $\tilde{\vec{y}} \sim \text{Multinomial}(\tilde{M}, \boldsymbol{\theta})$ and $\boldsymbol{\theta}|\vec{y}, \boldsymbol{\alpha} \sim \text{Dirichlet}(\boldsymbol{\alpha} + \vec{y})$ closely resembles (A.16):

$$p(\tilde{\vec{y}}|\vec{y}, \boldsymbol{\alpha}) = \frac{(\tilde{M}!)\,\Gamma(\sum_{\ell=1}^{L}\alpha_\ell + \vec{y}_\ell)}{\Gamma\big(\sum_{\ell=1}^{L}\alpha_\ell + \vec{y}_\ell + \tilde{\vec{y}}_\ell\big)} \prod_{\ell=1}^{L} \frac{\Gamma(\alpha_\ell + \vec{y}_\ell + \tilde{\vec{y}}_\ell)}{\big(\tilde{\vec{y}}_\ell!\big)\,\Gamma(\alpha_\ell + \vec{y}_\ell)} \tag{A.21}$$

$$= \frac{(\tilde{M}!)\,\Gamma(\sum_{\ell=1}^{L}\alpha_\ell + \vec{y}_\ell)}{\Gamma(\tilde{M} + M + \sum_{\ell=1}^{L}\alpha_\ell)} \prod_{\ell=1}^{L} \frac{\Gamma(\alpha_\ell + \vec{y}_\ell + \tilde{\vec{y}}_\ell)}{\big(\tilde{\vec{y}}_\ell!\big)\,\Gamma(\alpha_\ell + \vec{y}_\ell)}. \tag{A.22}$$

For the special case $\tilde{M} = 1$, this expression becomes

$$p(\tilde{\vec{y}} = \underbrace{(0, \ldots, 0, 1, 0, \ldots, 0)}_{=1 \text{ for } \ell\text{th element only}} |\vec{y}, \boldsymbol{\alpha}) = \frac{\alpha_\ell + \vec{y}_\ell}{\sum_{\ell'=1}^{L}\alpha_{\ell'} + \vec{y}_{\ell'}} \tag{A.23}$$

$$= \frac{\alpha_\ell + \vec{y}_\ell}{M + \sum_{\ell'=1}^{L}\alpha_{\ell'}}. \tag{A.24}$$

The prior predictive is the PMF (A.16) evaluated at $\vec{y} = \tilde{\vec{y}}$.

This page intentionally left blank.

APPENDIX B: NOTES ON BAYESIAN RULE LISTS

This appendix contains some mathematical details from Letham, Rudin, McCormick, and Madigan [12]. Let $\mathcal{A} = (a_1, a_2, \ldots, a_{|\mathcal{A}|})$ be the antecedent set, and let $a_0 \equiv \emptyset$ be the antecedent for the default rule. Also let $\boldsymbol{\alpha}_k = \boldsymbol{\alpha}$, $k = 0, 1, \ldots, |\mathcal{A}|$, be copies of the Dirichlet hyperparameter vector $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_L)$, and let $\vec{\boldsymbol{\alpha}} = (\boldsymbol{\alpha}_0, \boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_{|\mathcal{A}|})$. Next, let $d = (a_{k(1)}, a_{k(2)}, \ldots, a_{k(|d|)})$ be the rule list. For $j = 1, \ldots, |d|$, let $\boldsymbol{\theta}_{k(j)} = (\theta_{k(j),1}, \ldots, \theta_{k(j),L})$, be the multinomial/categorical parameter vector for antecedent $a_{k(j)}$, and let $k(0) = 0$ and $\boldsymbol{\theta}_{k(0)} = \boldsymbol{\theta}_0$ be the vector for the default rule. Finally, let $N_{j,\ell}(d, \mathbf{x}, \mathbf{y})$ be the number of samples in $(\mathbf{x}, \mathbf{y})$ that are captured by antecedent $a_{k(j)}$ and have label ℓ, and let $\mathbf{N}_j(d, \mathbf{x}, \mathbf{y}) = (N_{j,1}(d, \mathbf{x}, \mathbf{y}), \ldots, \mathbf{N}_{j,L}(d, \mathbf{x}, \mathbf{y}))$.

To convert our notation to that of Letham *et al.*, one can make the following replacements: $k(j) \leftarrow j$, $\alpha_{k(j),\ell} \leftarrow \alpha_\ell$, $\vec{\boldsymbol{\alpha}} \leftarrow \boldsymbol{\alpha}$, $N_{j,\ell} \leftarrow N_{j,\ell}(d, \mathbf{x}, \mathbf{y})$ and $\mathbf{N}_j(d, \mathbf{x}, \mathbf{y}) \leftarrow \mathbf{N}_j$.

B.1 RULE LIST LIKELIHOOD FUNCTION

The likelihood function is obtained by marginalizing over $\boldsymbol{\theta}_{k(0)}, \ldots, \boldsymbol{\theta}_{k(|d|)}$:

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{x}, d, \vec{\boldsymbol{\alpha}}) &= \int \cdots \int p(\mathbf{y}|\mathbf{x}, d, \boldsymbol{\theta}_{k(0)}, \ldots, \boldsymbol{\theta}_{k(|d|)}) p(\boldsymbol{\theta}_{k(0)}, \ldots, \boldsymbol{\theta}_{k(|d|)}|\vec{\boldsymbol{\alpha}})\, d\boldsymbol{\theta}_{k(0)} \cdots d\boldsymbol{\theta}_{k(|d|)} \\
&\overset{(a)}{=} \int \cdots \int p(\mathbf{y}|\mathbf{x}, d, \boldsymbol{\theta}_{k(0)}, \ldots, \boldsymbol{\theta}_{k(|d|)}) \prod_{j=0}^{|d|} p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)})\, d\boldsymbol{\theta}_{k(0)} \cdots d\boldsymbol{\theta}_{k(|d|)},
\end{aligned}
$$

where (a) is because the Dirichlet RVs $\boldsymbol{\theta}_{k(0)}, \ldots, \boldsymbol{\theta}_{k(|d|)}$ are independent, and each one only depends on its corresponding hyperparameter $\boldsymbol{\alpha}_{k(j)}$.

The samples captured by antecedent $a_{k(j)}$ are independent of those captured by the other antecedents, and they are independently distributed Multinomial($\boldsymbol{\theta}_{k(j)}$) [or Categorical($\boldsymbol{\theta}_{k(j)}$)], so

$$
p(\mathbf{y}|\mathbf{x}, d, \boldsymbol{\theta}_{k(0)}, \ldots, \boldsymbol{\theta}_{k(|d|)}) = \prod_{j:\sum_{\ell=1}^{L} N_{j,\ell} > 0} p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\theta}_{k(j)}),
$$

where the indexing means that the product only needs to include antecedents that capture at least one sample. Then

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{x}, d, \vec{\boldsymbol{\alpha}}) &= \int \cdots \int \left[\prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}>0} p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\theta}_{k(j)}) \right] \prod_{j=0}^{|d|} p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)})\, d\boldsymbol{\theta}_{k(0)} \cdots d\boldsymbol{\theta}_{k(|d|)} \\
&\overset{(a)}{=} \int \cdots \int \left[\prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}>0} p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\theta}_{k(j)}) p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)}) \right] \\
&\qquad \times \prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}=0} p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)})\, d\boldsymbol{\theta}_{k(0)} \cdots d\boldsymbol{\theta}_{k(|d|)} \\
&= \left[\prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}>0} \underbrace{\int p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\theta}_{k(j)}) p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)})\, d\boldsymbol{\theta}_{k(j)}}_{=p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\alpha}_{k(j)})} \right] \\
&\qquad \times \left[\prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}=0} \underbrace{\int p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)})\, d\boldsymbol{\theta}_{k(j)}}_{=1} \right] \\
&= \prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}>0} p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\alpha}_{k(j)}),
\end{aligned}
\tag{B.1}
$$

where (a) splits the product over $p(\boldsymbol{\theta}_{k(j)}|\boldsymbol{\alpha}_{k(j)})$ into antecedents that captured at least one sample and antecedents that did not capture any samples.

Each $p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\alpha}_{k(j)})$ in (B.1) is the PMF of a Dirichlet-multinomial RV for the samples captured by antecedent $a_{k(j)}$. This PMF is available from (A.7) by replacing $N_\ell(\mathbf{y})$ with $N_{j,\ell}(d, \mathbf{x}, \mathbf{y})$ and α_ℓ with $\alpha_{k(j),\ell}$:

$$
p(\mathbf{y}|\mathbf{x}, a_{k(j)}, \boldsymbol{\alpha}_{k(j)}) = \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) + \alpha_{k(j),\ell})} \prod_{\ell=1}^{L} \frac{\Gamma(N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) + \alpha_{k(j),\ell})}{\Gamma(\alpha_{k(j),\ell})}. \tag{B.2}
$$

Substituting (B.2) into (B.1) gives

$$
\begin{aligned}
p(\mathbf{y}|\mathbf{x}, d, \vec{\boldsymbol{\alpha}}) &= \prod_{j:\sum_{\ell=1}^{L} N_{j,\ell}(d,\mathbf{x},\mathbf{y})>0} \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) + \alpha_{k(j),\ell})} \prod_{\ell=1}^{L} \frac{\Gamma(N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) + \alpha_{k(j),\ell})}{\Gamma(\alpha_{k(j),\ell})} \\
&\overset{(a)}{=} \prod_{j=0}^{|d|} \frac{\Gamma(\sum_{\ell=1}^{L} \alpha_{k(j),\ell})}{\Gamma(\sum_{\ell=1}^{L} N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) + \alpha_{k(j),\ell})} \prod_{\ell=1}^{L} \frac{\Gamma(N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) + \alpha_{k(j),\ell})}{\Gamma(\alpha_{k(j),\ell})},
\end{aligned}
$$

where (a) is because, if there are no data samples for antecedent $a_{k(j)}$ (meaning $N_{j,\ell}(d, \mathbf{x}, \mathbf{y}) = 0$, $\forall \ell$), then (B.2) equals unity, and so the product can be safely calculated over all j. The last equation is the rule list likelihood function (6).

B.2 POSTERIOR PREDICTIVE FOR POINT ESTIMATE

Given a rule list point estimate d and an unlabeled sample $\tilde{x}$, we want to use d to predict the label $\tilde{y}$. Let $j(d, \tilde{x})$ be the index of the first antecedent in d that captures $\tilde{x}$. Then $\tilde{y} \sim \text{Multinomial}(\boldsymbol{\theta}_{k(j(d,\tilde{x}))})$ and $\boldsymbol{\theta}_{k(j(d,\tilde{x}))} \sim \text{Dirichlet}(\boldsymbol{\alpha} + \mathbf{N}_{j(d,\tilde{x})})$, so the posterior predictive is given by (A.14) with $\alpha_\ell \leftarrow \alpha_{k(j(d,\tilde{x})),\ell}$ and $N_\ell(\mathbf{y}) \leftarrow N_{k(j(d,\tilde{x}))}(d, \mathbf{x}, \mathbf{y})$:

$$p(\tilde{y} = \ell | \tilde{x}, d, \mathbf{x}, \mathbf{y}, \vec{\boldsymbol{\alpha}}) = \frac{\alpha_{k(j(d,\tilde{x})),\ell} + N_{j(d,\tilde{x}),\ell}(d, \mathbf{x}, \mathbf{y})}{\sum_{\ell'=1}^{L} \alpha_{k(j(d,\tilde{x})),\ell'} + N_{j(d,\tilde{x}),\ell'}(d, \mathbf{x}, \mathbf{y})}, \quad \ell = 1, \ldots, L,$$

which is (10).

This page intentionally left blank.

APPENDIX C: REVIEW OF RECURSIVE BAYESIAN TRACKING

This section quickly reviews *recursive Bayesian tracking* (RBT), which is also known as state estimation and includes Kalman filtering [20] [21] and particle filtering [42]. An excellent overview of RBT appears in [22].

C.1 STATE-SPACE MODEL

RBT is based on the state-space approach to dynamical systems [49] [50]. We use a prime to indicate RBT entities. The *state* at discrete time index n is designated by s'_n, a vector in a D'-dimensional space $\mathbf{S}'$, which can be continuous or discrete. The state-space model is

$$s'_n = f'(s'_{n-1}, u'_n, v'_{n-1}) \tag{C.1}$$
$$z'_n = h'(s'_n, u'_n, w'_n), \tag{C.2}$$

where the *control and dynamics model* (C.1) describes how the known *control input* u'_n and random *process noise* v'_{n-1} cause the state to change over time, and the *measurement model* (C.2) describes how measurement noise w'_n creates of noisy observations z'_n. From (C.1), the state sequence s'_1, $\ldots$, s'_{n-1}, s'_n, $\ldots$ is a first-order Markov process, and from (C.2), the measurement process is memoryless. The functions f' and h' can be made time-varying if desired.

The model presented here differs from the standard model in two minor ways. First, it includes a known control input u'_n, which is usually omitted. Second, the control and dynamics model (C.1) includes a direct-feed control input u'_n; control systems often use $s'_n = f'(s'_{n-1}, u'_{n-1}, v'_{n-1})$, which delays the effect of the control input by one time step.

The state dynamics model f' and assumptions about the process noise v'_{n-1} determine the *state transition distribution* $p(s'_n|s'_{n-1}, u'_n)$. The measurement model h' and assumptions about the measurement noise w'_n determine the *state likelihood function* $p(z'_n|s'_n, u'_n)$. The derivation of RBRL relies on finding suitable forms for these distributions.

C.2 USEFUL FORMS

We make some assumptions about the forms of f' and h' that are useful for making the analogy with RBRL. They correspond to the detailed state-space model in Figure C.1.

First, we assume the control input has three components, so let $u'_n = (x'_n, a'_n, r'_n)$, so that the components correspond to $\mathbf{x}_n$, $\mathbf{a}_n$, and $\mathbf{r}_n$ in RBRL. We also restrict f' to functions that can be expressed as a composition of functions:

$$\tilde{s}'_n = f'_{a'}(s'_{n-1}, a'_n) \tag{C.3}$$
$$s'_n = f'_{v'}(\tilde{s}'_n, v'_{n-1}), \tag{C.4}$$

so that

$$s'_n = f'_{v'}(f'_{a'}(s'_{n-1}, a'_n), v'_{n-1}). \tag{C.5}$$

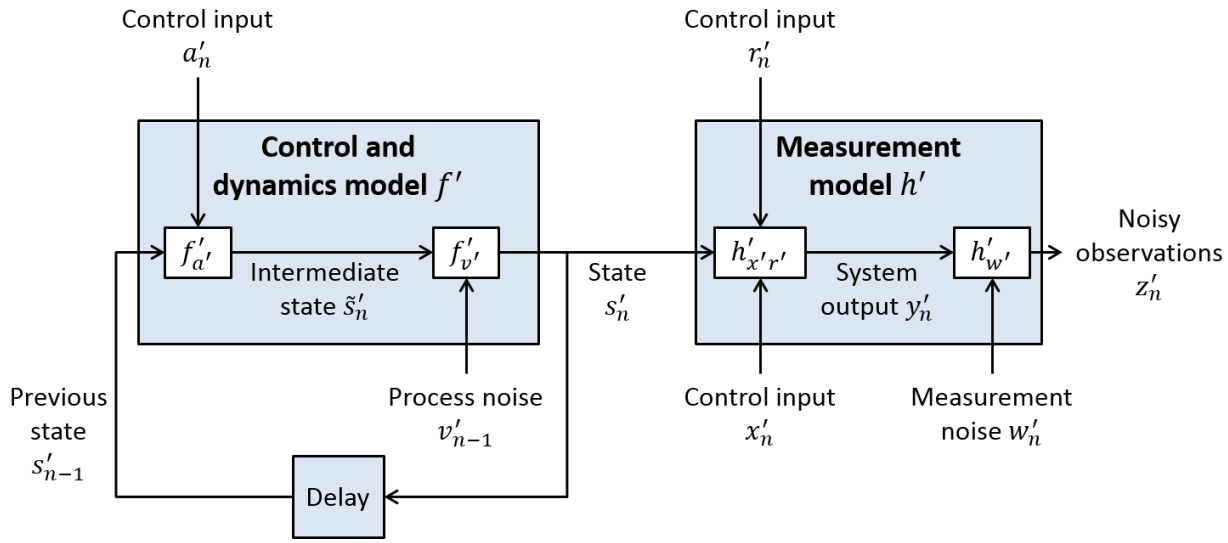

Figure C.1. Detailed state-space model in recursive Bayesian tracking

The state s'_{n-1} first transitions to an intermediate state $\tilde{s}'_n$ in a non-random way due to the known control input a'_n. Then $\tilde{s}'_n$ transitions to the state s'_n in a stochastic fashion due to the unknown process noise v'_{n-1}.

Second, we restrict h' to satisfy

$$y'_n = h'_{x'r'}(s'_n, x'_n, r'_n), \tag{C.6}$$
$$z'_n = h'_{w'}(y'_n, w'_n), \tag{C.7}$$

so

$$z'_n = h'_{w'}(h'_{x'r'}(s'_n, x'_n, r'_n), w'_n). \tag{C.8}$$

Here, y'_n is the pristine *system output* at time n. The direct-feed control inputs x'_n and r'_n are so-named because they directly affect the system output; they only have an indirect effect on the state. The system output is corrupted by measurement noise to produce the noisy observations z'_n.

Finally, the above assumptions yield the detailed Hidden Markov model in Figure C.2, Removing the control inputs and noises gives the simplified model in Figure C.3.

C.3 RECURSION

At each time n, RBT estimates the state s'_n given all available measurements $z'_{1:n}$. We quickly derive the recursive estimation equations for a continuous state space. Figure C.4 displays a block diagram of the resulting procedure. Let $p(s'_{n-1}|z'_{1:n-1}, x'_{1:n-1}, a'_{1:n-1}, r'_{1:n-1})$ be available; at time $n = 1$, it is just the state prior $p(s'_0)$.

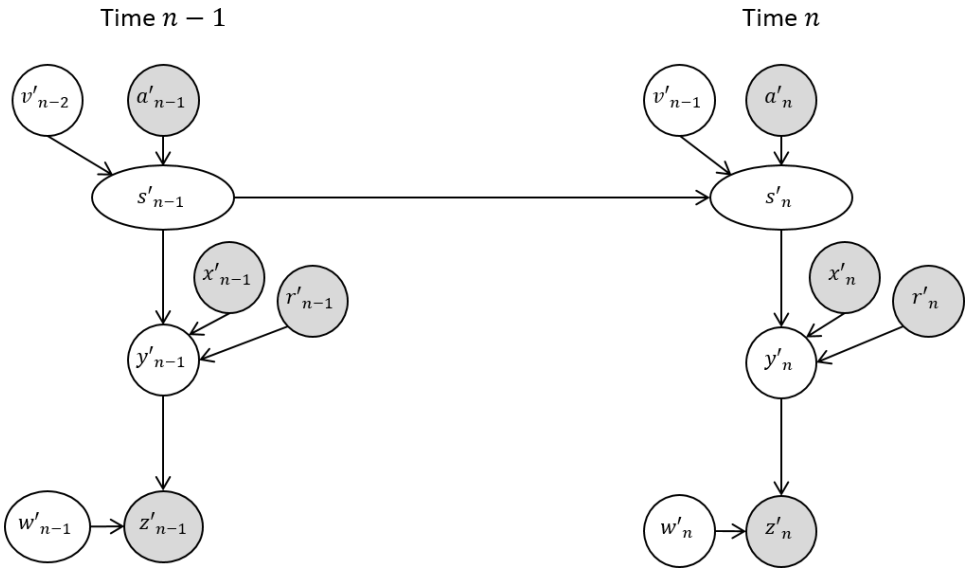

Figure C.2. Detailed hidden Markov model for recursive Bayesian tracking

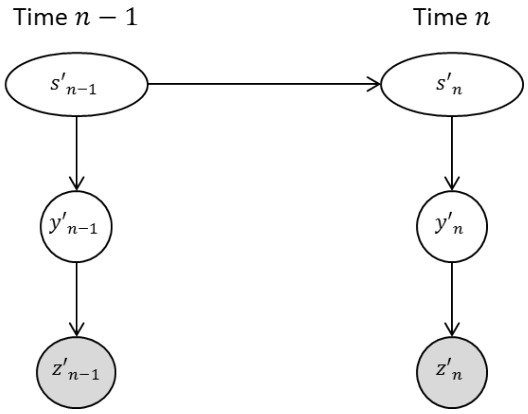

Figure C.3. Simplified hidden Markov model for recursive Bayesian tracking

At time n, the *control and dynamics update*[17] uses the known control input a'_n to calculate the *state prior at time n* according to

$$\begin{aligned}
&\underbrace{p(s'_n|z'_{1:n-1}, x'_{1:n-1}, a'_{1:n}, r'_{1:n-1})}_{\text{state prior at } n} \\
&\quad = \int p(s'_n|s'_{n-1}, z'_{1:n-1}, x'_{1:n-1}, a'_{1:n}, r'_{1:n-1}) p(s'_{n-1}|z'_{1:n-1}, x'_{1:n-1}, a'_{1:n}, r'_{1:n-1})\, ds'_{n-1} \\
&\quad \overset{(a)}{=} \int \underbrace{p(s'_n|s'_{n-1}, a'_n)}_{\substack{\text{state transition} \\ \text{distribution}}} \underbrace{p(s'_{n-1}|z'_{1:n-1}, x'_{1:n-1}, a'_{1:n-1}, r'_{1:n-1})}_{\text{state posterior at } n-1}\, ds'_{n-1},
\end{aligned} \tag{C.9}$$

where the first term in (a) only depends on s'_{n-1} and a'_n from (C.5), and the second term is because s'_{n-1} does not depend on quantities after time $n-1$.

The control inputs x'_n and r'_n and the next observation z'_n are then exploited by the *measurement update*, which calculates the *state posterior at time n*:

$$\begin{aligned}
&\underbrace{p(s'_n|z'_{1:n}, x'_{1:n}, a'_{1:n}, r'_{1:n})}_{\text{state posterior at } n} \\
&\quad \overset{(a)}{\propto} p(z'_n|s'_n, z'_{1:n-1}, x'_{1:n}, a'_{1:n}, r'_{1:n}) p(s'_n|z'_{1:n-1}, x'_{1:n}, a'_{1:n}, r'_{1:n}) \\
&\quad \overset{(b)}{=} \underbrace{p(z'_n|s'_n, x'_n, r'_n)}_{\substack{\text{state likelihood} \\ \text{function}}} \underbrace{p(s'_n|z'_{1:n-1}, x'_{1:n-1}, a'_{1:n}, r'_{1:n-1})}_{\text{state prior at } n},
\end{aligned} \tag{C.10}$$

where (a) is Bayes' rule; the first term in (b) is from (C.8); and the second term in (b) does not depend on x'_n or r'_n because of (C.5). A *filtered state estimate* $\hat{s}'_n$ can be computed from the state posterior at time n.[18] An optimal estimate like the conditional mean or most probable state is normally chosen.

C.4 ALGORITHMIC FORMS

The preceding section derived the RBT recursion; this section briefly covers some algorithms that can be used to implement it and are applicable to RBRL. First, when the state space is finite, a *grid-based method* (GBM) provides the theoretically optimal solution [22, Sec. III-B]. The state distribution becomes a PMF, and the GBM operates by storing every possible state and its associated probability mass. The control and dynamics update (C.9) becomes a summation rather than an integral. A GBM is optimal in theory and does not encounter issues of degeneracy, sample impoverishment, and resampling that arise in other RBT algorithms. However, a GBM can only be implemented when the state space is static and fairly small.

The second applicable RBT algorithm is *particle filtering* (PF), a sequential Monte Carlo (SMC) method. PF maintains N' *particles* or pairs $(s'^{[i]}_n, w^{[i]}_n)$, $i = 1, \ldots, N'$, where $s'^{[i]}_n$ is a sample

[17] In RBT, this update is usually called the "prediction update," but we avoid this term to prevent confusion.

[18] In RBT, it is also common to compute a *predicted state estimate* from the state prior at time n before z'_n has been observed. This estimate is of less interest for our purposes than the filtered state estimate.

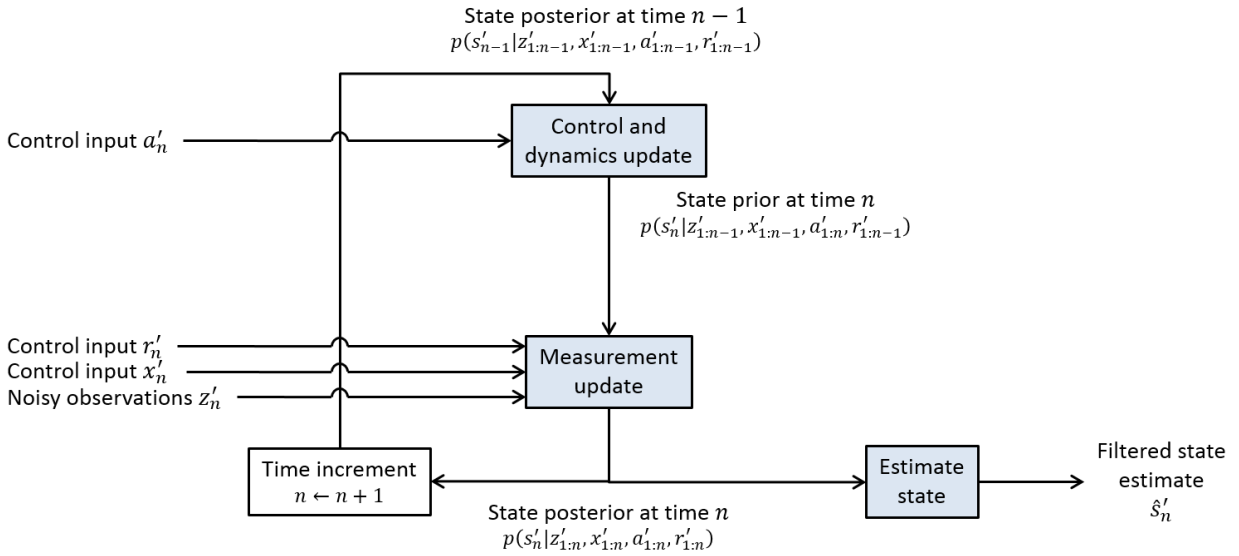

Figure C.4. Estimation procedure in recursive Bayesian tracking

state, and $w_n^{[i]} \geq 0$ is its weight. It is possible for $s_n'^{[i]}$ and $s_n'^{[j]}$ to be equal, so a superscript bracketed index is used instead of a parenthesized index. The sample states and weights change over time. For a continuous state space, PF approximates the posterior distribution as

$$p(s'_n|z'_{1:n}, u'_{1:n}) \approx \sum_{i=1}^{N'} w_n^{[i]} \delta\big(s'_n - s_n'^{[i]}\big), \tag{C.11}$$

where $\delta(x)$ is the Dirac delta function: $\delta(x) = 0$ for $x \neq 0$, and $\int_{-\infty}^{\infty} f(x)\delta(x)\,dx = f(0)$. PF implementations use different resampling tactics to counter the problems of degeneracy and sample impoverishment.

This page intentionally left blank.

GLOSSARY

AIM	Adaptable Interpretable Machine Learning
BRL	Bayesian Rule Lists
CART	Classification and Regression Trees
EM	expectation-maximization
FPM	frequent pattern mining
FRL	Falling Rule Lists
FY	fiscal year
GB-RBRL	grid-based Recursive Bayesian Rule Lists
GBM	grid-based method
HMI	human-machine interface
HMM	hidden Markov model
MAP	*maximum a posteriori*
MCMC	Markov Chain Monte Carlo
MIT	Massachusetts Institute of Technology
ML	machine learning
NP	non-deterministic polynomial time
PDF	probability density function
PF	particle filter; particle filtering
PF-RBRL	particle-filter Recursive Bayesian Rule Lists
PMF	probability mass function
RBRL	Recursive Bayesian Rule Lists
RBT	recursive Bayesian tracking
ROC	receiver operating characteristic
RV	random variable
SC-RBRL	static-case Recursive Bayesian Rule Lists
SIR	sampling importance resampling
SIR-RBRL	sampling importance resampling Recursive Bayesian Rule Lists
SMC	sequential Monte Carlo

This page intentionally left blank.

REFERENCES

[1] D.J. Berkeley, J.P. Simmons, and C. Massey, "Algorithm aversion: People erroneously avoid algorithms after seeing them err," *J. Experimental Psychology: General* 144(1), 114–126 (2014).

[2] K.E. Schaefer, J.Y.C. Chen, J.L. Szalma, and P.A. Hancock, "A meta-analysis of factors influencing the development of trust in automation: Implications for understanding autonomy in future systems," *Human Factors* 58(3), 377–400 (2016).

[3] H.J.D. Reynolds, *Decision Making under Uncertainty: Theory and Application*, MIT Press, chap. Integrating Automation with Humans, pp. 291–316 (2015), mykel J. Kochenderfer, ed.

[4] A.A. Freitas, "Comprehensible classification models– a position paper," *ACM SIGKDD Explorations Newsletter* 15(1), 1–10 (2013).

[5] L. Breiman, "Random forests," *Machine Learning* 45 (2001).

[6] R.C. Arkin, P. Ulam, and B. Duncan, "An ethical governor for constraining lethal action in an autonomous system," Mobile Robot Laboratory, College of Computing, Georgia Inst. of Technology, Georgia Inst. of Technology, Atlanta, GA 30332, Technical report GIT-GVU-09-01 (2009).

[7] E. Guizzo and E. Ackerman, "When robots decide to kill," *IEEE Spectrum* pp. 38–43 (2016).

[8] R.C. Holte, "Very simple classification rules perform well on most commonly used datasets," *Machine Learning* 11, 63–91 (1993).

[9] L. Breiman, "Statistical modeling: The two cultures," *Statistical Science* 16(3), 199–231 (2001).

[10] B.F. Gage, A.D. Waterman, W. Shannon, M. Boechler, M.W. Rich, and M.J. Radford, "Validation of clinical classification schemes for predicting stroke," *J. Amer. Med. Assoc.* 285, 2864–2870 (2001).

[11] U.S. Special Operations Command, U.S. Dept. of Defense (ed.), *Special Operations Forces Medical Handbook*, U.S. Special Operations Command, U.S. Dept. of Defense, 2nd ed. (2008).

[12] B. Letham, C. Rudin, T.H. McCormick, and D. Madigan, "Interpretable classifiers using rules and Bayesian analysis: Building a better stroke prediction model," *Annals Applied Stat.* 9(3), 1350–1371 (2015).

[13] B. Ustun and C. Rudin, "Supersparse linear integer models for optimized medical scoring systems," *Machine Learning* 102(3), 349–391 (2016), URL https://doi.org/10.1007/s10994-015-5528-6.

[14] F. Wang and C. Rudin, "Falling rule lists," in *Artificial Intelligence and Statistics (AISTATS)* (2015), pp. 1013–1022.

[15] J.G. Moreno-Torres, T. Raeder, R. Alaiz-Rodriguez, N.V. Chawla, and F. Herrera, "A unifying view on dataset shift in classification," *Pattern Recognition* 45, 521–530 (2012).

[16] J. Quinonero-Candela, M. Sugiyama, A. Schwaighofer, and N.D. Lawrence (eds.), *Dataset Shift in Machine Learning*, MIT Press (2009).

[17] J. Thornton, M. DeAngelus, and B. Miller, "Feedback-driven multimedia retrieval using feature-rich representations," MIT Lincoln Laboratory, Lexington, Massachusetts (2014).

[18] S. Dasgupta and J. Langford, "Active learning tutorial," Univ. of California, San Diego, Computer Science and Engineering, Artificial Intelligence Group, URL http://hunch.net/~active_learning/ (2009), Intl. Conf. Machine Learning (ICML), URL visited Oct. 10, 2017.

[19] S. Dasgupta, "Two faces of active learning," *Theoretical Computer Science* 412(19), 1767–1781 (2011).

[20] E.W. Kamen and J.K. Su, *Introduction to Optimal Estimation*, London, United Kingdom: Springer-Verlag (1999).

[21] F.L. Lewis, *Optimal Estimation: With an Introduction to Stochastic Control Theory*, Wiley (1986).

[22] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Trans. Sig. Proc.* 50(2), 174–188 (2002).

[23] A. Doucet and A.M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of Nonlinear Filtering* 12(3), 656–704 (2009).

[24] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. 1993 ACM SIGMOD Intl. Conf. Management of Data*, New York, NY, USA: ACM (1993), SIGMOD '93, pp. 207–216, URL http://doi.acm.org/10.1145/170035.170072.

[25] R. Agrawal, R. Srikant, et al., "Fast algorithms for mining association rules," in *Proc. 20th Intl. Conf. Very Large Data Bases, VLDB* (1994), vol. 1215, pp. 487–499.

[26] C. Borgelt, "An implementation of the FP-growth algorithm," in *Proc. 1st Intl. Workshop Open Source Data Mining: Frequent Pattern Mining Implementations*, New York, NY, USA: ACM (2005), OSDM '05, pp. 1–5, URL http://doi.acm.org/10.1145/1133905.1133907.

[27] M.J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.* 12(3), 372–390 (2000), URL http://dx.doi.org/10.1109/69.846291.

[28] L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen, *Classification and Regression Trees*, CRC press (1984).

[29] J.R. Quinlan, *C4.5: Programs for Machine Learning*, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. (1993).

[30] L. Hyafil and R.L. Rivest, "Constructing optimal binary decision trees is NP-complete." *Inf. Process. Lett.* 5(1), 15–17 (1976), URL http://dblp.uni-trier.de/db/journals/ipl/ipl5.html#HyafilR76.

[31] R.L. Rivest, "Learning decision lists," *Mach. Learn.* 2(3), 229–246 (1987), URL http://dx.doi.org/10.1023/A:1022607331053.

[32] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press (2009).

[33] D.M. Blei, A.Y. Ng, and M.I. Jordan, "Latent Dirichlet allocation," *J. Machine Learning Res.* 3, 993–1022 (2003).

[34] S. Chib and E. Greenberg, "Understanding the Metropolis-Hastings algorithm," *The American Statistician* 49(4), 327–335 (1995), URL http://www.jstor.org/stable/2684568.

[35] C. Chen and C. Rudin, "An optimization approach to learning falling rule lists," Duke University, Dept. of Computer Science, Durham, North Carolina, URL https://arxiv.org/abs/1710.02572 (2017), URL visited Oct. 10, 2017.

[36] A.P. Dawid and A.M. Skene, "Maximum likelihood estimation of observer error-rates using the EM algorithm," *Appl. Statist.* 28(1), 20–28 (1979).

[37] A.P. Dempster, N.M. Laird, and D.B. Rubin, "Maximum likelihood estimation from incomplete data via the EM algorithm," *J. Royal Stat. Soc. B* 39(1), 1–38 (1977).

[38] V.C. Raykar, S. Yu, L.H. Zhao, G.H. Valadez, C. Florin, L. Bogoni, and L. Moy, "Learning from crowds," *J. Mach. Learn. Res.* 11, 1297–1322 (2010).

[39] P. Smyth, U. Fayyad, M. Burl, P. Perona, and P. Baldi, "Inferring ground truth from subjective labelling of Venus images," in *Advances Neural Info. Proc. Sys. (NIPS)* (1995), vol. 7, pp. 1085–1092.

[40] P. Welinder, S. Branson, S. Belongie, and P. Perona, "The multidimensional wisdom of crowds," in *Advances Neural Info. Proc. Sys. (NIPS)* (2010), vol. 23, pp. 2424–2432.

[41] J. Whitehill, T. Wu, J. Bergsma, J.R. Movellan, and P.L. Ruvolo, "Whose vote should count more: Optimal integration of labels from labelers of unknown expertise," in *Advances Neural Info. Proc. Sys. (NIPS)* (2009), vol. 22, pp. 2035–2043.

[42] N.J. Gordon, D.J. Salmond, and A.F.M. Smith, "Novel approach to nonlinear/non-Gaussian Bayesian state estimation," *IEE Proc. F - Radar and Sig. Proc.* 140(2), 107–113 (1993).

[43] T. Li, M. Bolić, and P.M. Djurić, "Resampling methods for particle filtering: Classification, implementation, and strategies," *IEEE Sig. Proc. Magazine* pp. 70–86 (2015).

[44] W.R. Gilks and C. Berzuini, "Following a moving target—Monte Carlo inference for dynamic Bayesian models," *J. R. Statist. Soc. B* 63, 127–146 (2001).

[45] M. Pitt and N. Shephard, "Filtering via simulation: Auxiliary particle filters," *J. Amer. Statist. Assoc.* 94(446), 590–599 (1999).

[46] C. Musso, N. Oudjane, and F. LeGland, "Improving regularised particles filters," in A. Doucet, J.F.G. de Freitas, and N.J. Gordon (eds.), *Sequential Monte Carlo Methods in Practice*, New York: Springer-Verlag (2001).

[47] D. Crisan and A. Doucet, "A survey of convergence results on particle filtering methods for practitioners," *IEEE Trans. Sig. Proc.* 50(3) (2002).

[48] X.L. Hu, T. Schön, and L. Ljung, "A basic convergence result for particle filtering," *IEEE Trans. Sig. Proc.* 56(4), 1337–1348 (2008).

[49] W.L. Brogan, *Modern Control Theory*, Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 3rd ed. (1991).

[50] T. Kailath, *Linear Systems*, Prentice-Hall (1980).