UNIVERSITY OF CALIFORNIA, SAN DIEGO

# Modeling Probability Distributions with Predictive State Representations

A dissertation submitted in partial satisfaction of the requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Eric Walter Wiewiora

Committee in charge:

Professor Garrison Cottrell, Chair
Professor Charles Elkan, Co-Chair
Professor Sanjoy Dasgupta
Professor Bhaskar Rao
Professor Emanuel Todorov

2008

## DEDICATION

To my family and friends.

TABLE OF CONTENTS

LIST OF FIGURES

# LIST OF TABLES

## ACKNOWLEDGEMENTS

I would like to thank Professors Rich Sutton and Satinder Singh for many valuable comments and suggestions. I have received much help with formatting this document from Dana Dahlstrom, Kristin Branson and Sameer Agarwal.

Portions of this research has been published at the International Conference on Machine Learning, though most is unpublished. The experiments and initial formulation of the Linearly Dependent Distribution were done in collaboration with Anjum Gupta. During this research, I was funded under a grant from the NSF IGERT program (NSF grant DGE-0333451).

## VITA

| | |
|---|---|
| 1979 | Born, Pittsburgh, Pennsylvania. |
| 2001 | B. S., University of Pittsburgh. |
| 2004 | M. S. University of California, San Diego. |
| 2008 | Ph. D., University of California, San Diego. |

## PUBLICATIONS

A. Rabinovich, A. Vedaldi, C. Galleguillos , E. Wiewiora and S. Belongie, “Objects in Context”, *Proceedings of the International Conference on Computer Vision*, 2007.

A. Strehl, L. Li, E. Wiewiora, J. Langford, and M. Littman, “Efficient Model-free Reinforcement Learning”, *Proceedings of the International Conference on Machine Learning*, 2006.

E. Wiewiora, “Learning Predictive Representations from a History”, *Proceedings of the International Conference on Machine Learning*, 2005.

E. Wiewiora, “Potential-based Shaping and Q Value Initialization are Equivalent”, *Journal of Artificial Intelligence Research*, 2003.

E. Wiewiora, G. Cottrell and C. Elkan, “Principled Methods for Advising Reinforcement Learning Agents”, *Proceedings of the International Conference on Machine Learning*, 2003.

## FIELDS OF STUDY

Major Field: Computer Science and Engineering
Artificial Intelligence.

ABSTRACT OF THE DISSERTATION

**Modeling Probability Distributions with Predictive State Representations**

by

Eric Walter Wiewiora
Doctor of Philosophy in Computer Science

University of California, San Diego, 2008

Professor Garrison Cottrell, Chair
Professor Charles Elkan, Co-Chair

This dissertation presents an in-depth analysis of the Predictive State Representation (PSR), a new model for sequence prediction. The key insight behind PSRs is that predictions of some possible future realizations of the sequence can be used to predict the probability of other possible futures. Previous work has shown PSRs are very flexible, and can be trained from data without many of the drawbacks of similar models.

I present a rigorous theoretical foundation for understanding these models, and resolve several open problems in PSR theory. I also study multivariate prediction, where the model predicts the values of many random variables. The work presented in this dissertation is the first application of PSRs to modeling multivariate probability distributions.

I also perform extensive comparisons of PSR learning algorithms against algorithms for learning other popular prediction models. Surprisingly, the comparisons are not always favorable to PSRs. My empirical results provide an important benchmark for future research on learning PSRs, and my theoretical results may aid development of better learning algorithms.

# Chapter 1

# Introduction

My thesis addresses the problem of prediction. We desire a model of data that is able to extract patterns that the data follows, and use these patterns to anticipate future data that the system has not yet been exposed to. Such a model is often called a generative model, as it can be used to generate data similar to the data it was trained on, as well as estimate the probability of data.

The standard approach to generative modeling consists of three steps:

1. Choose a model class that will be powerful enough to find the relevant patterns in the data.

2. Train the parameters of this model class in order to make the observed data likely to be generated by the model.

3. Develop inference techniques that produce predictions based on the learned model.

These three goals are often in conflict. Models that are powerful are often hard to train and use for predictions. Likewise, simpler models may not be able to model the complexity that is present in the data.

Many popular classes of generative model incorporate latent variables. These are supposed hidden causes that are responsible for the values in the observable data. These models are often quite intuitive, and may offer good explanations of the patterns found in the data. For instance, the hidden Markov model(HMM), uses a latent variable to model a sequence of observations. This hidden variable usually has a readily interpretable meaning. For instance, in speech recognition, the hidden variable capture the word the speaker is trying to express. This hidden variable causes the actual speech (Rabiner, 1989).

Unfortunately, latent variable models are often quite difficult to train from data. By definition, the latent variable is not present in the data, so its properties and influences on the observable data must be observed indirectly. Algorithms that infer these relationships, such as expectation maximization (EM), tend to be slow and imprecise, and susceptible to poor initializations.

This thesis explores Predictive State Representations, a compelling new method for generative modeling. The key insight behind predictive state representations is that predictions of certain aspects of observable data can be used to make other predictions of the data. All parameters in these models refer to the actual data, not supposed latent causes that must be inferred indirectly. A second insight is to use simple operations to predict. All of the predictive models I investigate use simple linear operations to perform inference.

Surprisingly, the predictive representations studied in this work are *more powerful* than their most closely associated latent state model. That is, they can represent any probability distribution the latent state model can, plus many others that cannot be represented by the latent state models. Another surprising property is that predictive representations are often *convergent*: given sufficient data generated from a model in the same class, these algorithms will recover the generating model exactly. This is in stark contrast to EM, which may settle for a locally optimal model, regardless of the amount of training data.

My thesis contributes much to our knowledge of the properties of predictive state representations. I have answered several open problems in PSR theory, and have presented a rigorous theoretical foundation for understanding these models and their capabilities. I have also performed the most extensive comparisons of PSR learning performance against other popular generative models. Surprisingly, the comparisons are not always favorable to PSRs. My empirical results provide an important benchmark for future research on learning PSRs, and my theoretical results may aid progress toward developing better learning algorithms.

We explore two types of prediction problem. Sequence prediction involves predicting a series of observations generated from an unknown source. It is often the case that the only information that is available for predicting the future of the sequence is what has been observed before. We also study multivariate prediction, where we predict the values of many random variables. PSRs were developed for the sequence prediction problem. The work presented in this dissertation is the first exploration of the applicability of predictive representations in modeling multivariate data. We find that the modeling framework can easily accommodate this new type of prediction problem, though there are some caveats in applying previous learning techniques.

## 1.1 My Contribution

This thesis provides the most thorough theoretical analysis of predictive state representations to date. Much of the groundwork of this model class has been developed by Herbert Jaeger (Jaeger, 1998). He was the first to study this model class as a generative model, and developed the theoretical foundations and learning algorithms. Nearly all of the subsequent work on these representations has been heavily influenced by his early and ongoing work. This dissertation is no exception. The work presented here fills in many gaps in the theory of PSRs, and extends the applicability of PSRs to new prediction problems such as multivariate prediction. Note that throughout this document, all stated theorems are original work. Wherever there is a similar theorem presented in a different source, this is explicitly noted.

In chapter 2, I review previous models of sequence prediction, and common metrics for evaluating their performance. In particular, I review previously developed powerful algorithms for learning variable-ordered Markov models. I also explain why these models are the natural competitors to PSRs.

In chapter 3, I present the theory of Predictive State Representations. Much of this theory has already been developed, though my presentation is the most complete to date. I present several new properties that any valid PSR must possess. These include theorem 3, which provides the first set of necessary and sufficient conditions for a PSR to be valid, theorem 4, which shows that an important matrix of parameters has bounded eigenvalues, and theorem 5, which shows that the core predictions used by a PSR to make other predictions do not have to be very complicated. The most important contribution in this chapter is on the *negative probability problem.* Any given PSR may ”predict” a negative probability for some event. I show in theorem 8 that it is *undecidable* whether a given PSR will predict a negative probability for some event.

In chapter 4, I relate the prediction problem for uncontrolled processes to the controlled process case. Here, the observations in the sequence may depend on a sequence of actions. I show the close relationship between these two problems. I introduce a general definition of test that extends beyond any found in the literature. This definition of test allows for the characterization of the sequence to be predicted, as well as the policy: the source of actions. Characterizing policies based on tests has implications for reinforcement learning that I have not yet explored. Finally, theorem 10 shows that a controlled process and the process that generates actions can be jointly modeled as a single process. The theorem also addresses the additional complexity of modeling this coupled processes, relative to the complexity of the two individual processes. This allows model classes that do not handle actions to be used to model controlled processes. An earlier version of these results were presented in Wiewiora (2005).

In chapter 5, I examine the system matrix, a data structure that is important for learning PSRs. I relate this data structure to a Hankel matrix, which forms the basis of many learning algorithms for similar problems. I also address the steady-state system matrix. This structure models the long-term average probabilities of subsequences in a training sequence. I show in theorem 11 that any sequence data generated by a PSR will have well defined average subsequence probabilities. In previous work, this property has only been assumed.

In chapter 6, I review several PSR learning algorithms. Many of these algorithms were only defined for specific data sources or with particular modeling assumptions. My presentation generalizes these algorithms, and presents them in a common framework which aids understanding. I also present several experiments comparing learned PSRs to learned VMMs. This is the first extensive study of these models. Surprisingly, VMMs tend to outperform PSR algorithms, even when the data was produced by a model too complex to represent as a VMM. These results provide a sobering reminder of the immaturity of current PSR learning algorithms.

Finally, in chapter 7, I explore multivariate prediction. I introduce a predictive representation for multivariate prediction, which I call the linearly dependent distribution (LiDD). I show that predicting multiple variables is essentially the same problem as predicting a sequence of observations conditioned on actions. I also show that the LiDD is closely related to naïve Bayes for Probability estimation (NBE), a latent variable model that has been shown to be a simple and powerful predictor (Lowd and Domingos, 2005).

# Chapter 2

# Conventional Models for Sequence Prediction

In this chapter, we formalize the sequence prediction problem as well as introduce several models for this task.

## 2.1 Stochastic Processes

A *stochastic process* (also referred to as a dynamical system) is a source of sequential observations that occur in fixed time steps. We will assume that observations are drawn from a finite set $\mathcal{O}$. Some processes require an action to be provided by a separate agent. This action is from a finite set $\mathcal{A}$. When a stochastic process requires an action, we call it a controlled process; otherwise it is an uncontrolled process. In this chapter we focus on uncontrolled processes. We analyze the incorporation of actions in chapter 4.

Before we formally define a stochastic process, we need to introduce some notation. We define $\mathcal{O}^n$ as the set of all sequences of length $n$. The special sequence $\epsilon$ is the only sequence of length $0$. The set of all sequences is called $\mathcal{O}^*$. We define the concatenation operator on sequences $\mathbf{b}$ and $\mathbf{c}$ as $\mathbf{b} \circ \mathbf{c} \equiv \mathbf{bc}$.

We formally define a stochastic process as a distribution on sequences[1]:

**Definition 1** *A stochastic process is a function $P : \mathcal{O}^* \rightarrow \mathbb{R}$ satisfying*

- ***Boundedness**: $\forall \mathbf{o} \in \mathcal{O}^* : 1 \geq P(\mathbf{o}) \geq 0$*
- ***Consistency**: $\forall n \in \{0, 1, 2, \ldots\}, \forall \mathbf{o} \in \mathcal{O}^n : P(\mathbf{o}) = \sum_{b \in \mathcal{O}} P(\mathbf{o} \circ b)$*
- ***Normalization**: $P(\epsilon) = 1$*

The boundedness property ensures that the stochastic process produces numbers that will always be valid probabilities. The consistency property ensures that the sum probabilities of all continuations of a sequence equals the probability of the original sequence. The fixed measure property, combined with the consistency property, ensures that the sum probabilities of any complete set of sequences of length $n$ always equal 1.

All of the models we use for sequence prediction operate by predicting the next observation, given the previous observations in a sequence. From the predictions of each observation, we predict entire sequences using the *chain rule*

$$P(o_{1:t}) = \prod_{i=1}^{t} P(o_i | \mathbf{o}_{1:i-1}).$$

The string of previous observations up to time $t$, $\mathbf{o}_{1:t}$, is known as the *history* at time $t$: $\mathbf{h}_t$. Usually, the prediction of future observations, given the history, is not a function of the entire history. The aspects of the history that are used for predicting future observations are called the *Markov state* of the process. In other words, given the Markov state, the probabilities of a process' future observations are independent of the history.

## 2.2 Sequence Prediction

Given a sequence $\mathbf{o}_{1:t}$, we desire an accurate prediction of the next observation in the sequence $o_{t+1}$. This prediction is given as a conditional distribution on the next

[1]This definition is streamlined to apply only to discrete time, discrete valued random variables. Also, I define a stochastic process as a function that gives the probability of an assignment of a set of these random variables, rather than the random variables themselves. See Grimmett and Stirzaker (1982) for a more conventional definition.

observation $P(O_{t+1}|\mathbf{h}_t)$. We assume that we use a stochastic process $P$ to predict the next observation. After the prediction is made, the true observation $o_{t+1}$ is revealed, and a loss is incurred.

In this dissertation, we will use two methods for measuring the quality of predictions.

The *log loss* $l_{\log}()$ of a prediction $P$ is defined as

$$l_{\log}(o_{t+1}, P(\cdot|\mathbf{h}_t)) = -\log(P(o_{t+1}|\mathbf{h}_t)).$$

The average log loss, $L()$, for an entire sequence is

$$L_{\log}(\mathbf{o}_{1:t}, P) = \frac{1}{t}\sum_{i=1}^{t} l(o_i, P(\cdot|\mathbf{h}_{i-1})).$$

Throughout this work, we assume the log function is the natural log. The log loss has several advantages for a measure of prediction loss. First, the log loss is calibrated: if there is a true distribution over the probability of the next observation, then choosing $P$ to equal that distribution will yield the lowest expected log loss. Optimizing the log loss is an important objective in parameter estimation, as minimizing the average log loss of a sequence is equivalent to maximizing the sequence's liklihood.

Finally, it can be shown that optimizing the log loss of a model is equivalent to finding a compressed encoding of the sequence (Rissanen and Langdon, 1981). In this context, the average base 2 log loss of a sequence is called its ideal code length. Procedures such as arithmetic coding are able to encode each observation with an average number of bits no greater than 2 plus close to this ideal code length (Willems *et al.*, 1995).

The second loss function we examine is the squared loss $l_2$

$$l_2(o_{t+1}, P) = (1 - P(o_{t+1}))^2.$$

The $l_2$ loss (also called the Brier loss) is often used in sequence prediction tasks where the end goal is not a probabilistic model or an efficient encoding of the sequence (Vovk, 2001). For instance, it is the loss function of choice for the evaluation of weather forecasts. Like $l_{\log}$, if the observations come from a distribution, setting $P$ to that distribution will minimize the expected loss. Unlike $l_{\log}$, this loss function is bounded when $P$ is bounded; if $P$ is always between 0 and 1, the loss will also be between 0 and 1.

## 2.3 Variable-ordered Markov Models

The first class of models we consider are variable-ordered Markov models (VMMs). The primary modeling assumption behind these models is that the next observation in a sequence is a function of the recent history (a suffix of the entire history). We call the suffix of the history that is used to make predictions the *context*. Note that the context of a VMM is the model's *Markov state*. Usually the length of the suffix is bounded by some constant $d$. When $d = 1$, a VMM is also called a *Markov chain*.

There is a long history of learning algorithms for VMMs. Begleiter *et al.* (2004) provide a recent review of some of the more popular methods used for compression. Algorithms for learning VMMs differ in how they choose which contexts to base predictions on, and how they convert the raw counts found in training data into smoothed probability estimates. Specifically, learning algorithms produce probability estimates based on counts of the number of times different observations occur after different contexts.

### 2.3.1 Context Trees

Most VMMs organize their contexts into a data structure called a context tree. We label the contexts that a VMM keeps as $\mathbf{c}_1, \ldots, \mathbf{c}_k$. The root of the tree corresponds to no context. Branching from each node are one labeled edge for each observation. The nodes branching from the root of the tree correspond to looking at the most recent past observation (the length one suffix of the history). Nodes deeper in the tree keep predictions for the context produced by following the labeled edges from node to root.

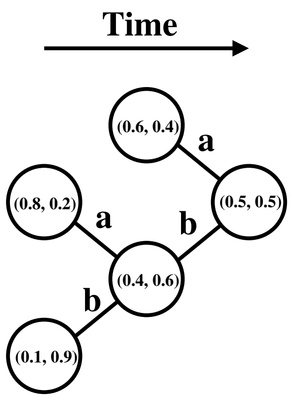


Figure 2.1: An example context tree with $\mathcal{O} = \{a, b\}$. Each node in the tree represents the predictions made, given a certain context. The context is determined by appending the observations on the labeled edges from the node to the root. The probability of observations $a$ and $b$, given a context, are found inside the contexts' node.

We define a function $c()$ that takes a history of observations and returns the deepest context in a context tree that matches this history.

See figure 2.1 for an example of a context tree. In this example, we find $P(a|\mathbf{h} = \ldots ab)$ by traversing the context tree from root to leaf. We follow the edge from the root labeled $b$ because $b$ is the most recent observation in the history. We then follow the edge labeled $a$, which is the second most recent observation. We have now hit a leaf of the tree, and will use the predictions contained within this leaf node $(P(a|\mathbf{h}) = 0.8)$.

### 2.3.2 Context-Tree Weighting

The context-tree weighting (CTW) algorithm is a popular algorithm for learning a VMM (Willems *et al.*, 1995). Recent experiments have shown it to be quite robust on several different sequence prediction tasks (Begleiter *et al.*, 2004) . The CTW algorithm constructs a complete[2] context tree of some height $d$, which is provided as a parameter to the algorithm.

We now address how the context tree is used to make predictions in the CTW

[2]A complete tree has the maximum number of children for each non-leaf node.

algorithm. To ease exposition, we begin with the case of binary observations $O = \{0, 1\}$, before the more general case of an arbitrary number of observations. Assume that the CTW is trained on some sequence $\mathbf{o}_{1:T}$.

First, we consider predictions made at leaf nodes in the tree. For each node $i$, corresponding to context $\mathbf{c}_i$, We count the number of times each observation appears after the context in $\mathbf{o}$. Call these counts $\#0_i$ and $\#1_i$. These counts are turned into predictions using the using the Krichevsky-Trofimov (KT) estimator (Willems *et al.*, 1995). This estimator, which is a modification to Laplace's law of succession, is calculated as

$$P_i^{leaf}(0) = \frac{\#0_i + \frac{1}{2}}{\#0_i + \#1_i + 1}.$$

This estimator is also the Bayesian posterior estimate of the probability of 0, given the prior for the probability is distributed as a Beta distribution with parameters $(\frac{1}{2}, \frac{1}{2})$. The prediction for 1 is calculated by replacing $\#0_i$ in the numerator with $\#1_i$.

When making predictions given some context $\mathbf{c}_{c(\mathbf{h}_t)}$, the CTW algorithm considers all prunings of its context tree. For one specific context, we only have to consider pruning along the path induced by this context. The predictions at each pruning are weighted using a Bayesian posterior that this node is the one used to make the prediction. The prior probability that any node of the context tree is pruned to be a leaf, given that is hasn't been pruned already, is $\frac{1}{2}$ .

We calculate the final prediction from leaf to root.

First, we introduce the context likelihood $\mathcal{L}_j$. This measures the probability of all predictions made by context $\mathbf{c}_j = c(\mathbf{h}_{t-j:t})$ on the training sequence. For leaf nodes, this is calculated as

$$\mathcal{L}_j^{leaf} = \frac{\prod_{x=1}^{\#0_j}(x - \frac{1}{2}) \prod_{y=1}^{\#1_j}(y - \frac{1}{2})}{(\#0_j + \#1_j)!}.$$

It is easy to show that this is the product of all predictions made for the training sequence observations that appear after this context. The context likelihood of an internal node is calculated recursively. For some context $o_k \dots o_d$, the context likelihood is

$$\mathcal{L}_j = \frac{1}{2}\mathcal{L}_j^{leaf} + \frac{1}{2}\Big(\prod_{o \in O} \mathcal{L}_{c(o \circ \mathbf{c_j})}\Big).$$

We can now calculate the final prediction made by the CTW with context $\mathbf{c}_i$. The prediction made at the leaf of the full context tree is simply $P_i^i = P_i^{leaf}$, as calculated above. We index intermediate predictions with a superscript referencing the full context and a subscript to reference the intermediate context. The prediction at an intermediate context is found recursively:

$$P_j^i(0) \propto \frac{1}{2}\mathcal{L}_j^{leaf} P_j^{leaf}(0) + \frac{1}{2}\big(\prod_{o \in O} \mathcal{L}_{c(o \circ \mathbf{c}_j)}\big) P^i_{c(\mathbf{h}_{t-j-1:t})}(0)$$

This calculation can be thought of as taking weighted predictions given the posterior probabilities that the context tree is pruned at this node, or if the node is internal. The final prediction is the one made at the root:

$$P(o|\mathbf{h}_t) = P_{c(\epsilon)}^{c(\mathbf{h}_t)}(o);$$

The CTW algorithm can be extended to handle more than two observations in many possible ways. Instead of using the KT estimator as the leaf predictor, we must use a predictor that can handle multiple observations. The choice of predictor will alter the leaf predictions as well as the leaf likelihoods, but will not alter the recursion. In practice, the best method for predicting multiple observations has been to break the prediction into several binary prediction tasks. See Begleiter *et al.* (2004) for details on methods for doing this.

For the experiments later in the thesis, we use an implementation of the CTW algorithm provided by Begleiter *et al.* (2004), available for download at http://www.cs.technion.ac.il/~ronbeg/vmm/ .

### 2.3.3 Applications and Limitations

Two prominent and successful applications of VMMs have been compression and language modeling (Brown *et al.*, 1992; Bell *et al.*, 1990). For natural language processing, VMMs are often referred to as *n-gram* models. In this situation, $n$ refers to the subsequence length required to learn the model, where the first $n-1$ observations define the context, and the $n$th observation is the one to be predicted.

Despite the success of VMMs, they have been criticized as insufficient for representing stochastic processes of interest to some communities (Littman *et al.*, 2001). The principle limitation is that any information contained in a history before the longest context is lost to the VMM. More elaborate models, which we consider next, are able to keep information arbitrarily long.

## 2.4 Hidden Markov Models

Hidden Markov Models (HMMs), are a popular method for modeling processes that contain some amount of memory. HMMs can be considered a subset of stochastic, nondeterministic finite-state automata (S-NFA), which are studied in the automata literature (Vidal *et al.*, 2005). The main difference between these two approaches is that automata assign probabilities to finite strings, while HMMs find the probability of a prefix to an arbitrarily long sequence [3].

The key modeling assumption in a HMM is that the probability of future observations in the sequence are completely determined by the latent state of the process. The (Markov) state at time $t$, $X_t$, can be one of a finite number ($k$) of possible values. Knowledge of the current state of the process is also sufficient to determine the probability distribution on the next state $X_{t+1}$. The state of the HMM is an unknown random variable whose distribution is determined by the history of the system. We call this distribution the belief state $\mathbf{b}_t$. It is a $1 \times k$ vector, where index $\mathbf{b}_t[i]$ gives the probability that the process is in state $i$ at time $t$. We will also discuss the belief state resulting after observing some history $\mathbf{h}$. We call this belief state $\mathbf{b}(\mathbf{h})$.

A HMM generates observations each time step when the latent state transitions. In state-emission HMMs, the probability of the current observation only depends on the current state. We focus on the more expressive transition HMM. In this type of HMM, the probability of an observation at time $t$ may depend on both the current state $X_t$ as

[3]Some researchers define HMMs as generating finite strings (Gavalda *et al.*, 2006). In this case they are equivalent to S-NFAs

well as the next state $X_{t+1}$. Note that the Markov property still holds, and thus the joint probability of the current observation and next state only depends on the current state.

### 2.4.1 HMM Parameters

The parameters of a HMM are the initial distribution over states and the transition function that determines the distribution over next state and observation, given the current state. Given these parameters, we can calculate the joint probability of a state sequence and an observation sequence. However, except for special case models, we can never be certain of the latent state of the process by examining only the observations. All of the observation probabilities must be integrated over our uncertainty of the latent state. This can be done using linear algebra. We introduce the $k \times k$ matrices $\mathbf{T}_o$, where $o$ is an observation in our finite set $O$. The $[i, j]$th entry in $\mathbf{T}_o$ gives the probability that the next state is $j$, and the current observation is $o$, given that the current state is $i$. We calculate the probability of the observation, given the current belief state $\mathbf{b}_t$ as follows:

$$\begin{aligned} \Pr(O_t = o|\mathbf{b}_t) &= \sum_{i=1}^{k} \Pr(X_t = i) \sum_{j=1}^{k} \Pr(X_{t+1} = j, O_t = o|X_t = i) \\ &= \mathbf{b}_t \mathbf{T}_o \mathbf{1}, \end{aligned}$$

Where $\mathbf{1}$ is the $k \times 1$ column vector of all 1's.

In order to use an HMM for sequence prediction, it is also important to be able to find the belief distribution of the next state, $\mathbf{b}_{t+1}$. We use the same matrices to do this:

$$\begin{aligned} \Pr(X_{t+1} = j|O_t = o, \mathbf{b}_t) &= \frac{\Pr(X_{t+1} = j, O_t = o|\mathbf{b}_t)}{\Pr(O_t = o|\mathbf{b}_t)} \\ &= \frac{\mathbf{b}_t \mathbf{T}_o(:, j)}{\mathbf{b}_t \mathbf{T}_o \mathbf{1}}; \\ \mathbf{b}_{t+1}|O_t = o, \mathbf{b}_t &= \frac{\mathbf{b}_t \mathbf{T}_o}{\mathbf{b}_t \mathbf{T}_o \mathbf{1}}. \end{aligned}$$

Note that this guarantees that at all times $t$, the sum of the entries in $\mathbf{b}_t$ is 1.

Finally, we can also use this model to calculate the probabilities of longer sequences. The probability of sequence $o_1, o_2, \ldots, o_n$, given an initial belief $\mathbf{b}_0$, is given by a series of matrix multiplications:

$$\Pr(o_{1:n}|\mathbf{b}_0) = \mathbf{b}_0 \mathbf{T}_{o_1} \cdots \mathbf{T}_{o_n} \mathbf{1} \tag{2.1}$$

For convenience, we condense the series of $\mathbf{T}$ multiplications into a single $k \times k$ matrix $\mathbf{T}_{o_{1:n}}$.

We define the matrix $\mathbf{T}_* = \sum_{o \in \mathcal{O}} \mathbf{T}_o$. This matrix calculates the transition probabilities of the states without the observations. It is a stochastic matrix, meaning that it is nonnegative, its maximum singular value is 1, and

$$\mathbf{T}_* \mathbf{1} = \mathbf{1}.$$

It is insightful to compare these invariants to those we discuss in chapter 3 for predictive state representations.

### 2.4.2 VMMs and HMMs

HMMs are a proper superset of VMMs. In other words, every process that can be modeled as a VMM can also be modeled as a HMM, but there are some processes that HMMs that can model that cannot be modeled by any VMM. We demonstrate this constructively.

First, we show how to produce a HMM that models the same process as a VMM. Given a VMM with contexts $\mathbf{c}_1, \ldots \mathbf{c}_k$, we produce an HMM with $k$ states: one state for every context. It is clear that the future predictions made by the VMM depend only on the current context. This is precisely the criteria that the HMM state satisfies. The HMM transition matrices $\mathbf{T}_o$ are sparse:

$$\mathbf{T}_o[i,j] = \begin{cases} P(o|\mathbf{c}_i), & \text{if } j = c(\mathbf{c}_i \circ o) \\ 0, & \text{otherwise.} \end{cases}$$

This sparse transition behavior means that given the previous state and the current observation, the current state is completely determined. Such a model is also called a

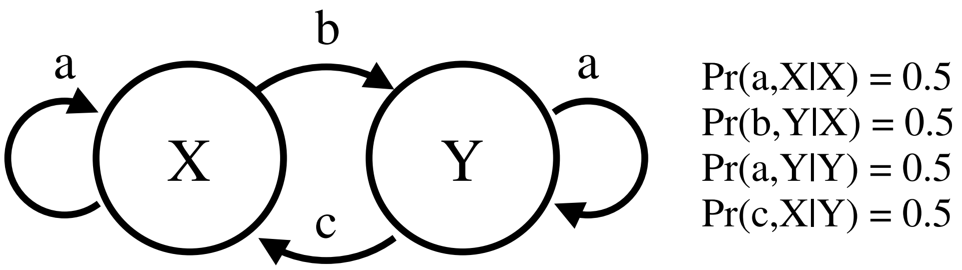


Figure 2.2: An example hidden Markov model (HMM) that cannot be modeled by a finite context variable-order Markov model (VMM).

deterministic probabilistic finite automata (Vidal *et al.*, 2005). These models only have $O(k|\mathcal{O}|)$ parameters, instead of $O(k|\mathcal{O}|^2)$ for the general case HMM.

In figure 2.2, we show a simple example of an HMM that cannot be represented by any finite-context VMM. There are two states, $\{X, Y\}$, and three observations $\{a, b, c\}$. On each step, with equal probability, either the state changes or the state remains the same. When the state is unchanged, an "$a$" is emitted, when there is a transition from $X$ to $Y$, a "$b$" is emitted, and when there is a transition from $Y$ to $X$, a "$c$" is emitted. Note that for all $m = 0, 1, 2, \ldots$, the following probabilities hold:

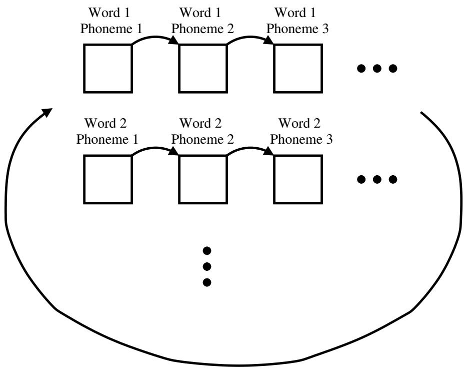
$$\begin{aligned}\Pr(b|\mathbf{h} = ca^m) &= \frac{1}{2}, \\ \Pr(b|\mathbf{h} = ba^m) &= 0.\end{aligned}$$

When $m$ is larger than the longest context in a VMM, the VMM will not be able to distinguish these two histories.

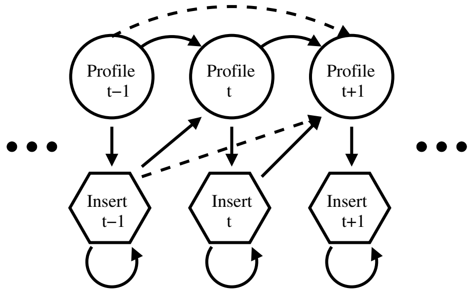
### 2.4.3 Applications

HMMs are primarily used in situations where the latent states have some meaning, and a good deal of knowledge of the transition structure is known. Two such applications are speech recognition (Rabiner, 1989) and biological sequence analysis (Durbin *et al.*, 1999).

For speech recognition, HMMs are used to assess the probability that a speaker



(a) Speech Recognition HMM. Each square represents an HMM subsystem for recognizing a particular phoneme.



(b) Profile HMM. The circle states represent an element of the profile. The hexagonal states represent possible insertions into the profile that result from mutation. The dashed lines represent skips in the profile, also due to mutation.

Figure 2.3: Two examples of HMMs with task-dependent structure and semantics.

intended to say a particular word. The latent state of the HMM contains information on both the current word and the current phoneme within the word. The observations are the (processed) sounds that are vocalized. It is usually the case that phonemes shared across different words will also share the same HMM parameters (Rabiner, 1989).

Biological sequence analysis is a broad discipline with many particular problems. One problem is to determine if a collection of sequences (DNA or amino acid chains) are related through a common ancestor. This has been tackled by modeling the sequences as "noisy" versions of a latent sequence (the profile) that all of them have mutated from. The structure of the profile is usually estimated using some other technique, and refined later using the learning techniques mentioned below.

See figure 2.3 for diagrams of these HMMs. Note that both examples have a highly constrained state transition structure, and that the latent states have specific semantics. The state semantics are vital to how these systems are used in practice.

Another application of HMMs is to track a mobile system and map its position (Shatkay and Kaelbling, 1997). In this application, the latent state is the robot's location, and the observations are "noisy" sensor readings. In this domain, there is a good deal of knowledge about how latent states (map locations) relate to each other. In fact, most systems model state transitions by hand, and only learn the relation between states and observations.

### 2.4.4 Learning

Most HMMs are initially designed by hand, though there are algorithms that can learn a HMM without prior knowledge, using only observation sequences.

The standard objective for HMM learning and model refining is to maximize the likelihood of the training sequence. This is identical to minimizing the log loss of the training string. Unfortunately, finding the $k$ state HMM that generates the string with highest probability is a difficult problem, which is conjectured to be NP-hard (Vidal *et al.*, 2005). As a result, learning algorithms are only likely to find a locally optimal HMM. See Vidal *et al.* (2005) for a review of techniques for learning Hidden Markov Models and related automata.

The most popular procedure for optimizing the parameters of an initial HMM is the Expectation-Maximization (EM) algorithm. For HMM learning, EM is identical to the independently developed Baum-Welch algorithm. We present this algorithm based on the presentations by Bilmes (1997), and Xue and Govindaraju (2002). This algorithm takes an initial HMM, and refines it through a series of local improvements. Each iteration of EM is guaranteed to improve the model's training likelihood, or leave it unchanged. EM alternates between two steps:

1. For each time step $t$ in the training string $\mathbf{o}_{1:n}$, and state $i$, estimate $\mathbf{q}_t[i]$, the conditional probability of being in state $i$ at time $t$, given the entire string and the current HMM parameters.

2. Fix the state visit probabilities $\mathbf{q}_t$. Modify the HMM parameters $(\mathbf{b}_0, \mathbf{T}_o)$ to maximize the expected joint likelihood of the training sequence and the HMM state sequence.

We now go through the details of this algorithm.

In order to find the state visit probabilities, we calculate two helper vectors:

$$\begin{aligned}\boldsymbol{\alpha}_t &= \mathbf{b}_0^\top \mathbf{T}_{\mathbf{o}_{1:t-1}},\\ \boldsymbol{\beta}_t &= \mathbf{T}_{\mathbf{o}_{t:n}} \mathbf{1}.\end{aligned}$$

The $1 \times k$ vector $\boldsymbol{\alpha}_t$ contains the probabilities of arriving in each state at time $t$, given the training sequence up to observation $t-1$. The $k \times 1$ vector $\boldsymbol{\beta}_t$ contains the probabilities of generating the observation sequence $\mathbf{o}_{t:n}$ from each of the $k$ states. We calculate the probability of visiting each state at time $t$ by multiplying these vector entries:

$$\mathbf{q}_t[i] = \frac{\boldsymbol{\alpha}_t[i]\boldsymbol{\beta}_t[i]}{\boldsymbol{\alpha}_t\boldsymbol{\beta}_t}.$$

In order to update the transition probabilities, we will make use of another set of helper variables $\boldsymbol{\xi}_t$. This $k \times k$ matrix calculates the conditional probability that state $i$ transitions to state $j$ on time step $t$. It is calculated as

$$\boldsymbol{\xi}_t[i,j] = \frac{\boldsymbol{\alpha}_t[i]\mathbf{T}_{o_t}[i,j]\boldsymbol{\beta}_{t+1}[j]}{\boldsymbol{\alpha}_t\boldsymbol{\beta}_t}.$$

New HMM parameters are produced using these helper variables:

$$\begin{aligned}\mathbf{b}_0 &\leftarrow \mathbf{q}_0\\ \mathbf{T}_o &\leftarrow \frac{\sum_{t:o_t=o}\boldsymbol{\xi}_t}{\sum_{t=1}^{n}\boldsymbol{\xi}_t}.\end{aligned}$$

These new parameters maximize the likelihood of the expected state sequence and the observation sequence.

After each iteration, the process is repeated. The stopping condition for EM is either when the parameters converge, or when the new HMM parameters begin to lose prediction accuracy on some validation sequence.

# Chapter 3

# Predictive State Representations

We have seen two different approaches to modeling a process that generates a sequence. The variable-order Markov model (VMM) uses a suffix of the history in order to predict the next observation. This suffix is directly observable from the sequence history. Unfortunately, the VMM suffers from limited memory. Any observations that have occurred earlier than the previous context are not used to make future predictions. The hidden Markov model (HMM) uses a distribution on latent states to make predictions. This state distribution is not directly computed from the history. Instead, it is updated recursively based on the most recent observation. Because the latent state is only indirectly related to the observations in the sequence, learning a good state space is difficult without a good deal of prior knowledge.

The predictive state representation (PSR) is a new model that attempts to address the shortcomings of both previous models (Littman *et al.*, 2001; Singh *et al.*, 2004). Like the HMM, a PSR updates its state recursively rather than deriving the state directly from the history. We show that this allows it to keep informations from the history in the state for arbitrarily long. A PSR differs from a HMM in that it does not rely on a latent state representation. Instead, the state is based on beliefs about the probabilities of future observations in the sequence. Specifically, the state of a PSR is a vector of predictions for a selected set of observation sequences, called *tests* (after Rivest and

Schapire (1994)). As more of the sequence is revealed, the accuracy of test probabilities can be evaluated directly.

Predictive state representations are usually presented as a model for controlled stochastic processes, which will be covered in chapter 4. I take an unorthodox approach of first introducing the PSR as a model of a standard stochastic process before discussing the more complex controlled case. Almost all of the mathematical specifics of the PSR were first developed by Herbert Jaeger (2000) under the name Observable Operator Model (OOM). In fact, for uncontrolled stochastic processes, there is a straightforward translation of one model into the other. I present this model under the PSR framework because it is a bit more intuitive than the OOM, and the PSR provides some additional flexibility in the parameterization of the model.

The PSR is closely related to stochastic multiplicity automata (S-MA) (Beimel *et al.*, 2000; Even-Dar *et al.*, 2005; Denis *et al.*, 2006). Many of the concepts behind PSRs and OOMs were first addressed in this context. As is the case for S-NFA, stochastic multiplicity automata are usually defined as assigning probabilities to finite strings, rather than the prefix of an arbitrarily long sequence.

## 3.1 Events in Uncontrolled Systems

Predictive state representations use explicit predictions about future observations to model the state of the system. We specify the target of these predictions using *tests*. Tests measure the possible outputs of the system state that span beyond one time step. This added scope of prediction is able to characterize more of the state of a process, which may not be apparent looking only one step ahead.

For an uncontrolled stochastic process, a test g is a set of observation sequences, where each sequence contains $l_\mathrm{g}$ observations. We call the test for uncontrolled systems *events* [1]. An event is said to have occurred when the next $l$ observations match one of the

[1] The "*event*" name comes from the OOM literature, which has focused on uncontrolled processes. We make the distinction between tests and events to avoid confusion when we discuss controlled processes.

sequences in the test. The probability that a length $l$ event $\mathbf{g}$ succeeds, given a history of observations $\mathbf{h}$ is given by:

$$\Pr(\mathbf{g}|\mathbf{h}) = \sum_{\mathbf{o}\in\mathbf{g}} \Pr(\mathbf{o}|\mathbf{h}) = \sum_{\mathbf{o}\in\mathbf{g}} \prod_{i=1}^{l_{\mathbf{g}}} \Pr(\mathbf{o}_i|\mathbf{h} \circ \mathbf{o}_{1:i-1}).$$

We give special attention to a class of events called s-events. [2] These events consist of a single sequence of observations. Another kind of event we will see are called e-events.[3] One of these events includes all sequences of some length $l$ which end in some observation $o$. We include one special case in our definition of events. The null event is the event consisting of an event of length zero (called $\epsilon$). This event succeeds with probability 1. The null event can be thought of as representing the event that anything happens in the future.

## 3.2 Linear PSRs

A PSR models a stochastic process using a set of events called the *core events*. We say that a set of $k$ events $\mathbf{q}_1, \ldots, \mathbf{q}_k$ are core events of a PSR if given any history $\mathbf{h}$, the success probability of any future s-event $\mathbf{g}$ is a function of the probability that each of the core events will succeed:

$$\Pr(\mathbf{g}|\mathbf{h}) = f_{\mathbf{g}}\Big(\Pr(\mathbf{q}_1|\mathbf{h}), \Pr(\mathbf{q}_2|\mathbf{h}), \ldots, \Pr(\mathbf{q}_k|\mathbf{h})\Big).$$

In this sense, the core events form a basis for predicting every other event. We call the vector of probabilities that the core events succeed, given the history, $\mathbf{q}(\mathbf{h})$. This vector is used much like the belief state vector of a hidden Markov model. The initial event probabilities, $\mathbf{q}(\epsilon)$, are provided as parameters of the PSR.

Linear PSRs assume that all core events are s-events and $f_g$ is a linear function of $\mathbf{q}(\mathbf{h})$:

$$\Pr(\mathbf{g}|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \mathbf{m}_{\mathbf{g}}.$$

[2]The “s” here stands for sequence.

[3]The “e” stands for end.

Call the vector $\mathbf{m_g}$ the predictor vector for $\mathbf{g}$. We say a stochastic process has rank of $k$ if we can predict all s-event probabilities using a PSR with $k$ core events, but no fewer. Such a PSR is said to be a *minimal event* PSR. *For the remainder of this work, we only consider linear PSRs, and simply call them PSRs.*

In order to use a PSR, we need a method for calculating the prediction vectors $\mathbf{m_o}$ for an arbitrary sequence $\mathbf{o}$, and we need to know the success probabilities of the core events $\mathbf{q}(\mathbf{h})$ for different histories $\mathbf{h}$. It is convenient to calculate these probabilities recursively.

First, we need the vector $\mathbf{m}_o$ such that

$$\Pr(o|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \mathbf{m}_o,$$

for each possible observation $o$. These are called the one step predictors.

Also, for each core event $\mathbf{q}_i$ we find the vector $\mathbf{m}_{oi}$ such that

$$\Pr(o \circ \mathbf{q}_i|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \mathbf{m}_{oi}.$$

We call these events the core event extensions, and the vectors $\mathbf{m}_{oi}$ the extension predictors.

The future core event probabilities are recursively updated using $\mathbf{m}_o$ and $\mathbf{m}_{oi}$. When a new observation $o$ received, the new success probability of core event $q_i$ is given by:

$$\Pr(\mathbf{q}_i|\mathbf{h}o) = \frac{\mathbf{q}(\mathbf{h})^\top \mathbf{m}_{oi}}{\mathbf{q}(\mathbf{h})^\top \mathbf{m}_o}.$$

The initial vector of core event probabilities, $\mathbf{q}(\epsilon)$, is a parameter of the PSR.

For convenience, we arrange the core extension predictors for observation $o$ into a matrix $\mathbf{M}_o$, where the $i$th column in this matrix corresponds to the extension event for core event $q_i$:

$$\mathbf{M}_o = \begin{bmatrix} | & | & & | \\ \mathbf{m}_{o1} & \mathbf{m}_{o2} & \dots & \mathbf{m}_{ok} \\ | & | & & | \end{bmatrix}.$$

We call these matrices the core update operators.

Any sequence's probability can be calculated using $\mathbf{M}_o$ and $\mathbf{m}_o$. For sequence $\mathbf{g} = o_1 o_2 \ldots o_l$, and a core event probability vector $\mathbf{q}(\mathbf{h})$, we calculate the success probability of $\mathbf{g}$ as

$$\Pr(\mathbf{g}|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \mathbf{M}_{o_1}\mathbf{M}_{o_2}\cdots\mathbf{M}_{o_{n-1}}\mathbf{m}_{o_n} \tag{3.1}$$

(Littman *et al.*, 2001).

Note that this calculation quite similar to the operations used in equation 2.1 to calculate the probability of a sequence using a HMM. We adopt the same shorthand of calling the result of a series of core update multiplications, $\mathbf{M}_{o_1}\cdots\mathbf{M}_{o_n}$, by the name of the respective sequence $\mathbf{M}_{\mathbf{o}_{1:n}}$. We define the core update operator for the length zero sequence $\mathbf{M}_\epsilon$ as the identity matrix.

The major difference between PSR parameters and HMM parameters is that the PSR parameters have fewer restrictions on their values. Unlike the HMM belief state $\mathbf{b}(\mathbf{h})$, the PSR core event probabilities $\mathbf{q}(\mathbf{h})$ do not have to sum to 1. Unlike the HMM parameters $\mathbf{T}_o$, the PSR parameters $\mathbf{M}_o$ do not need to be probabilities (i.e. the parameters can be greater than one or less than zero). This extra flexibility results in additional modeling power. It was shown by Jaeger (1998) that PSRs can model a strict superset of the stochastic processes that HMMs can model. In other words, every stochastic process that can be modeled by a HMM can also be modeled by a PSR, but there are some processes that can be modeled by a PSR but not by any finite state HMM.

## 3.3 PSRs and OOMs

The Observable Operator Model (Jaeger, 2000; Jaeger *et al.*, 2006b) was developed before the PSR, and uses nearly identical concepts and operations to produce predictions. The main contributions of PSRs over OOMs are a more approachable notation, a more flexible definition of event, and fewer restrictions on the choice of core events. Also, PSRs were initially introduced for modeling controlled processes, while OOMs are used to model processes without actions.

The core events of an OOM are called its *characteristic events*, and their success probabilities after history $\mathbf{h}$ are called $\mathbf{w}(\mathbf{h})$. While the core events of a PSR can be any set of events that can be used to predict all other events, an OOM's characteristic events are constrained to partition the sequences in $\mathcal{O}^l$ for some sufficiently large $l$. This constraint allows the following invariant for all histories $\mathbf{h}$ :

$$\begin{aligned}\mathbf{1}^\top \mathbf{w}(\mathbf{h}) &= \sum_{i=1}^{k} \Pr(\mathbf{q}_i|\mathbf{h}) \\ &= \sum_{\mathbf{o}\in\mathcal{O}^l} \Pr(\mathbf{o}|\mathbf{h}) \\ &= 1.\end{aligned}$$

The core update operators, called $\boldsymbol{\tau}_o$, behave the same as for the PSR, except that they are transposed. This means that when we wish to update the characteristic events after seeing $o_1 o_2 \ldots o_n$, we apply the update operators in *reverse order* (right to left):

$$\Pr(o_1 o_2 \ldots o_n|\mathbf{h}) = \mathbf{1}^\top \boldsymbol{\tau}_{o_n} \cdots \boldsymbol{\tau}_{o_2} \boldsymbol{\tau}_{o_1} \mathbf{w}(\mathbf{h}).$$

Singh *et al.* (2004) proved that OOMs and PSRs model the same class of stochastic processes. James (2005) provides algorithms for converting one representation into the other.

## 3.4 Properties and Invariants of the PSR

PSRs can model a very large class of functions on $\mathcal{O}^* \to \mathbb{R}$. This set is known as the *rational languages* (Denis *et al.*, 2006) . Unfortunately, this set contains many more functions than those that obey the properties of a stochastic process. In this section, we explore some constraints on the PSR parameters when a PSR is modeling a stochastic process, and the question of given a PSR, how can we tell if it is modeling a stochastic process.

At first glance, a rank $k$ PSR appears to have $|\mathcal{O}|k^2$ parameters in the $\mathbf{M}_o$ matrices, $|\mathcal{O}|k$ parameters in the $\mathbf{m}_o$ vectors, and $k$ parameters in $\mathbf{q}(\epsilon)$. However, there exist a series of invariant properties of a PSR that restrict the range of valid parameters for the model. Some of these invariants are easy to check, while checking for some are quite difficult. The main unresolved issue is that unlike earlier models, it is not obvious when a PSR produces outputs that obey the properties of a stochastic process.

We first show a result that will be useful for future proofs:

**Lemma 1** *A PSR with $k$ events is minimal only if for every sequence $\mathbf{o}$, there is a unique predictor vector $\mathbf{m_o}$ such that for all $\mathbf{h}$, $\Pr(\mathbf{o}|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \mathbf{m_o}$.*

PROOF: Assume that there are two different predictor vectors that calculate the probability of sequence $\mathbf{g}$. Call these vectors $\mathbf{m_g}$ and $\mathbf{m_g}'$. Now find some entry $i$, corresponding to event $\mathbf{q}_i$, where the two vectors $\mathbf{m_g}[i]$, $\mathbf{m_g}'[i]$ are different. We can calculate $\Pr(\mathbf{q}_i|\mathbf{h}) = \mathbf{q}(\mathbf{h})[i]$ using the other core events:

$$\begin{aligned}
\mathbf{q}(\mathbf{h})^\top \mathbf{m_g} &= \mathbf{q}(\mathbf{h})^\top \mathbf{m_g}' \\
\mathbf{q}(\mathbf{h})[i](\mathbf{m_g}[i] - \mathbf{m_g}'[i]) &= \mathbf{q}(\mathbf{h})[\{j \neq i\}]^\top (\mathbf{m_g}'[\{j \neq i\}] - \mathbf{m_g}[\{j \neq i\}]) \\
\mathbf{q}(\mathbf{h})[i] &= \frac{\mathbf{q}(\mathbf{h})[\{j \neq i\}]^\top (\mathbf{m_g}'[\{j \neq i\}] - \mathbf{m_g}[\{j \neq i\}])}{(\mathbf{m_g}[i] - \mathbf{m_g}'[i])} \\
&= \mathbf{q}(\mathbf{h})[\{j \neq i\}]^\top \mathbf{m}_{q_i}
\end{aligned}$$

Given this solution for $\mathbf{q}(\mathbf{h})[i]$, we can construct a new prediction vector $\mathbf{m_o^*}$ for any sequence $\mathbf{o}$ that does not use the probability of $\mathbf{q}_i$.

$$\begin{aligned}
\mathbf{q}(\mathbf{h})^\top \mathbf{m_o} &= \sum_{j=1}^{k} \mathbf{q}(\mathbf{h})[j]\mathbf{m_o}[j] \\
&= \mathbf{q}(\mathbf{h})[i]\mathbf{m_o}[i] + \sum_{j \neq i} \mathbf{q}(\mathbf{h})[j]\mathbf{m_o}[j] \\
&= (\mathbf{q}(\mathbf{h})[\{j \neq i\}]^\top \mathbf{m}_{q_i})\mathbf{m_o}[i] + \sum_{j \neq i} \mathbf{q}(\mathbf{h})[j]\mathbf{m_o}[j] \\
&= \mathbf{q}(\mathbf{h})[\{j \neq i\}]^\top \mathbf{m_o^*}
\end{aligned}$$

This contradicts our definition of a minimum event PSR.

□

**Corollary 2** *In a minimum event PSR, the prediction vector $\mathbf{m}_{\mathbf{q_i}}$ for the $i$th core event is all zeros, except for a one in the $i$th entry.*

PROOF: By construction, this vector calculates $\Pr(\mathbf{q}_i|\mathbf{h})$. The previous lemma guarantees that it is unique. $\square$

Here is a list of properties that a PSR must satisfy that are adapted from the OOM literature (Jaeger, 2000). These properties parallel those for a stochastic process presented in definition 1. We make use of two special derived parameters. The vector $\mathbf{m}_\epsilon$ is the predictor for the null event. Given the PSR parameters, we can calculate $\mathbf{m}_\epsilon$ as $\sum_{o\in\mathcal{O}} \mathbf{m}_o$. We will see that this vector functions like the $\mathbf{1}$ vector for both the HMM and OOM. The matrix $\mathbf{M}_* = \sum_{o\in\mathcal{O}} \mathbf{M}_o$ calculates the expected value of the core events on the next time step. By lemma 1, we know that both of these parameters are unique for a given PSR.

**Theorem 3** *[adapted from Jaeger (2000)] A minimum event PSR $\{\mathbf{q}(\epsilon), \mathbf{M_o}, \mathbf{m_o}\}$, where $P(\mathbf{o} \in \mathcal{O}^*)$ is calculated using equation 3.1, and given $\mathbf{m}_\epsilon, \mathbf{M}_*$ as calculated above, models a stochastic process if and only if it has the following properties:*

1. $\mathbf{q}(\epsilon)^\top \mathbf{m}_\epsilon = 1$ ,
2. $\forall \mathbf{o} \in \mathcal{O}, \mathbf{m_o} = \mathbf{M_o}\mathbf{m}_\epsilon$ ,
3. $\mathbf{M}_*\mathbf{m}_\epsilon = \mathbf{m}_\epsilon$ ,
4. $\forall \mathbf{o} \in \mathcal{O}^*, 1 \geq \mathbf{q}(\epsilon)^\top \mathbf{M_o}\mathbf{m}_\epsilon \geq 0$ .

PROOF:

We prove this by showing that a PSR satisfying these properties satisfies the properties of a stochastic process given in definition 1. Also, if the PSR does not satisfy these properties, then it cannot be a model of a stochastic process.

The normalization property of a stochastic process ($P(\epsilon) = 1$) is satisfied by property 1. Likewise, if property 1 is violated, then so is the normalization property.

The consistency property is satisfied by properties 2 and 3:

$$\begin{aligned} P(\mathbf{o}) &= \mathbf{q}(\epsilon)^\top \mathbf{m_o} \\ &= \mathbf{q}(\epsilon)^\top \mathbf{M_o} \mathbf{m}_\epsilon \\ &= \mathbf{q}(\epsilon)^\top \mathbf{M_o} \mathbf{M}_* \mathbf{m}_\epsilon \\ &= \mathbf{q}(\epsilon)^\top \mathbf{M_o} \sum_{b \in \mathcal{O}} \Big( \mathbf{M}_b \Big) \mathbf{m}_\epsilon \\ &= \sum_{b \in \mathcal{O}} \Big( \mathbf{q}(\epsilon)^\top \mathbf{M_o} \mathbf{M}_b \mathbf{m}_\epsilon \Big) \\ &= \sum_{b \in \mathcal{O}} P(\mathbf{o} \circ b). \end{aligned}$$

We now show that if properties 2 or 3 are violated, then so is the consistency property. First, we address property 2. Assume that for some $\mathbf{o} \in \mathcal{O}^*, \mathbf{m_o} \neq \mathbf{M_o} \mathbf{m}_\epsilon$ :

$$\begin{aligned} \sum_{b \in \mathcal{O}} P(ob) &= \sum_{b \in \mathcal{O}} \mathbf{q}(\epsilon)^\top \mathbf{M}_o \mathbf{m}_b \\ &= \mathbf{q}(\epsilon)^\top \mathbf{M_o} \Big( \sum_{b \in \mathcal{O}} \mathbf{m}_b \Big) \\ &= \mathbf{q}(\epsilon)^\top \mathbf{M}_o \mathbf{m}_\epsilon \\ &\neq \mathbf{q}(\epsilon)^\top \mathbf{m}_o = P(o). \end{aligned}$$

A violation of property 3 also results in the consistency property being violated. The proof is nearly identical to that above.

$$\begin{aligned} \sum_{b \in \mathcal{O}} P(ob) &= \sum_{b \in \mathcal{O}} \mathbf{q}(\epsilon)^\top \mathbf{M}_o \mathbf{M}_b \mathbf{m}_\epsilon \\ &= \mathbf{q}(\epsilon)^\top \mathbf{M}_o \mathbf{M}_* \mathbf{m}_\epsilon \\ &\neq \mathbf{q}(\epsilon)^\top \mathbf{M}_o \mathbf{m}_\epsilon = P(o). \end{aligned}$$

Finally, we address property 4. Here we take equation 3.1, but substitute the final one-step predictor with the corresponding core extension matrix and the null event predictor. This new equation must compute the same quantity given property 2 proved above. The boundedness property of a stochastic process is satisfied by property 4, and boundedness is violated if property 4 is violated. □

An immediate consequence of this theorem is that the one-step predictors $\mathbf{m}_o$ are not free parameters if we are given the core extension matrices $\mathbf{M}_o$ and $\mathbf{m}_\epsilon$. Thus, we can use this theorem to reduce the parameters in a rank $k$ PSR by $(|\mathcal{O}| - 1)k$ by replacing the $\mathbf{m}_o$ vectors with $\mathbf{m}_\epsilon$.

Note that the first three properties of theorem 3 are simple to check given a specified model. The last property states that predictions on all sequences are bounded in the range of probabilities. It is not obvious how to check this property. We will address this issue in section 3.5.

We finish this section with one additional restriction on $\mathbf{M}_*$:

**Theorem 4** *For a minimum event PSR, the matrix $\mathbf{M}_*$ has at least one eigenvalue equal to 1. Also, $\mathbf{M}_*$ does not have any eigenvalues with absolute value greater than 1.*

PROOF: Property 2 of theorem 3 shows that $\mathbf{m}_\epsilon$ is a right eigenvector of $\mathbf{M}_*$ with eigenvalue 1. This proves the first part of this theorem.

Assume that there is some eigenvalue of $\mathbf{M}_*$ greater than 1. In this case, there is a nonzero vector $\mathbf{q}'$ such that $\mathbf{q}'^\top \mathbf{M}_* = \alpha \mathbf{q}'^\top$, with $\alpha > 1$. Examine the largest such $\alpha$ where this is the case. Assume that some $\mathbf{q}(\mathbf{h})$ can be decomposed into $\beta\mathbf{q}' + \gamma\mathbf{q}''$, where $\beta > 0$ and $\mathbf{q}'^\top\mathbf{q}'' = 0$. For some sufficiently large $x$, $\mathbf{q}(\mathbf{h}\mathcal{O}^x)^\top = \mathbf{q}(\mathbf{h})^\top {\mathbf{M}_*}^x$ will have some entries outside the range [0 1]. This is because this vector will be dominated by the term $= \alpha^x \beta \mathbf{q}'$, which is unbounded as $x$ increases. This vector, which represents the expected probabilities of the core events $x$ steps in the future, cannot have values outside of $[0, 1]$.

Assume however, that we cannot find a $\mathbf{q}(\mathbf{h})$ that can be broken into components $\beta\mathbf{q}' + \gamma\mathbf{q}''$. In this case, we have added the linear constraint to the core events that $\forall \mathbf{h}, \mathbf{q}(\mathbf{h})^\top \mathbf{q}' = 0$. This constraint reduces the degrees of freedom of our core event probabilities. One of the core events must be redundant. In either case, we have a contradiction.

Finally, consider the case where there is some eigenvalue less than $-1$. The argument for this case is similar: either we can find a history where some expected future core event probability is less than 0, or there is an additional linear constraint to the core event probabilities, meaning they were not minimal. $\square$

A similar theorem appeared in the work of Jaeger (2000). His theorem relies heavily on the OOM representation, however, and is not directly applicable to PSRs.

### 3.4.1 Multiplicity Automata

In their most general form, multiplicity automata are a collection of parameters $\langle \mathcal{O}, r, \boldsymbol{\iota}, \boldsymbol{\mu}_{o\in\mathcal{O}}, \boldsymbol{\gamma}\rangle$, where $\mathcal{O}$ is a finite set, $r$ is a real number, $\boldsymbol{\iota}$ is a $1 \times r$ vector, $\boldsymbol{\mu}_{o\in\mathcal{O}}$ is a $r \times r$ matrix, and $\boldsymbol{\gamma}$ is a $r \times 1$ vector (Beimel *et al.*, 2000; Denis and Esposito, 2004). A multiplicity automaton $A$ defines a function $f_A : \mathcal{O}^* \rightarrow \Re$ such that for sequence $o_1 \ldots o_n$:

$$f_A(o_1 \ldots o_n) = \boldsymbol{\iota}\Big(\prod_{1=1}^{n} \boldsymbol{\mu}_{o_i}\Big)\boldsymbol{\gamma}.$$

Clearly, any PSR is a multiplicity automaton, where $\mathbf{q}(\epsilon)^\top = \boldsymbol{\iota}, \mathrm{M}_o = \boldsymbol{\mu}_o$, and $\mathbf{m}_\epsilon = \boldsymbol{\gamma}$.

It has been shown that for any $f_A$, we can define a multiplicity automata $A'$, where each entry $\boldsymbol{\iota}'[i]$ equals $f_A$'s value on some sequence $\mathbf{o_i}$, and each $\boldsymbol{\mu}_a$ is defined such that:

$$\boldsymbol{\iota}\boldsymbol{\mu}_a[:, i] = f_A(a \circ \mathbf{o}_i)$$

(see Beimel *et al.* (2000) for a presentation of this well-known result). If $f_A$ defines a stochastic process, then $A'$ matches our definition of a PSR.

**Proposition 1** *(see Beimel et al. (2000)) Any stochastic process that can be defined by a multiplicity automaton can also be defined by a PSR.*

### 3.4.2 Regular Form PSRs

Any rank $k$ stochastic process can be modeled by many $k$ event PSRs. Any set of $k$ s-events where each event cannot be predicted by the others will constitute a core event set (Singh *et al.*, 2004). We would like a guideline for choosing core events

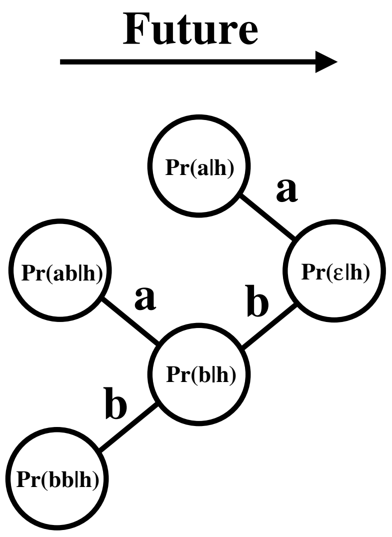


Figure 3.1: A tree of PSR core events that are in regular form. The event sequence is produced by following a node to the root of the tree.

that lead to as simple a representation as possible. Below we present a framework for choosing a particularly advantageous set of core events:

**Definition 2** *A PSR is in regular form if the set of core events is minimal and each core event is either the null event, or an extension of a core event.*

A regular form PSR has core events that can be arranged in a tree structure reminiscent of context tree. The null event makes the root of the tree. All other core events are appended to the tree by adding an edge to the core event that this event extends. See figure 3.1 for a diagram of an event tree.

An algorithm for producing a (nearly) regular form PSR from a HMM has been provided by Littman *et al.* (2001), and later by Even-Dar *et al.* (2005). A similar form for OOMs has been proposed by Jaeger (1998). Jaeger conjectured that all OOMs have such a form, but didn't prove it. I have extended these results to show that all PSRs have an equivalent regular form (Wiewiora, 2005).

**Theorem 5** *Any $k$ event PSR can be converted to a regular form PSR with no more than $k$ events.*

PROOF: Assume PSR $\mathbf{P} = (\mathbf{q}_1 \ldots \mathbf{q}_k, \{\mathbf{m}_o\}, \{\mathbf{M}_o\})$ is not in regular form. We show how to incrementally convert it to a regular form PSR by replacing core events that do not meet the regular form definition.

Consider when the null event is not a core event in $\mathbf{P}$. By our definition of a PSR, there is some $\mathbf{m}_\epsilon$ such that $\mathbf{q}(\mathbf{h})^\top \mathbf{m}_\epsilon = 1$ for all histories $\mathbf{h}$. We find some event $\mathbf{q}_i$ where the $i$th entry in $\mathbf{m}_\epsilon$ is nonzero. Such an event must exist since the probability of the null event is nonzero. We can calculate the probability of $\mathbf{q}_i$ using the other core events and the null event:

$$\begin{aligned}
1 &= \sum_j \Pr(\mathbf{q}_j|\mathbf{h})\mathbf{m}_\epsilon[j] \\
\Pr(\mathbf{q}_i|\mathbf{h})\mathbf{m}_\epsilon[i] &= 1 - \sum_{j\neq i} \Pr(\mathbf{q}_j|\mathbf{h})\mathbf{m}_\epsilon[j] \\
\Pr(\mathbf{q}_i|\mathbf{h}) &= \frac{1}{\mathbf{m}_\epsilon[i]} \Pr(\epsilon|\mathbf{h}) + \sum_{j\neq i} \Pr(\mathbf{q}_j|\mathbf{h}) \frac{-\mathbf{m}_\epsilon[j]}{\mathbf{m}_\epsilon[i]}.
\end{aligned}$$

Because we can calculate $\Pr(\mathbf{q}_i|\mathbf{h})$ for any $\mathbf{h}$, this core event can be replaced with the null event.

Now consider the case where there is some core event $\mathbf{q}_i$ which is not an extension of any other core event. Also, assume there is some extension of a regular form core event $\mathbf{q}_x$, which we call $\mathbf{q}_{ox}$, that is not a core event, and $\mathbf{q}_i$ has a nonzero entry in $\mathbf{m}_{ox}$. We use the same argument made for the null event to show that we can replace $\mathbf{q}_i$ with $\mathbf{q}_{ox}$:

$$\begin{aligned}
\Pr(\mathbf{q}_{ox}|\mathbf{h}) &= \sum_j \Pr(\mathbf{q}_j|\mathbf{h})\mathbf{m}_{ox}[j] \\
\Pr(\mathbf{q}_i|\mathbf{h})\mathbf{m}_{ox}[i] &= \Pr(\mathbf{q}_{ox}|\mathbf{h}) - \sum_{j\neq i} \Pr(\mathbf{q}_j|\mathbf{h})\mathbf{m}_{ox}[j] \\
\Pr(\mathbf{q}_i|\mathbf{h}) &= \frac{\Pr(\mathbf{q}_{ox}|\mathbf{h})}{\mathbf{m}_{ox}[i]} + \sum_{j\neq i} \Pr(\mathbf{q}_j|\mathbf{h}) \frac{-\mathbf{m}_{ox}[j]}{\mathbf{m}_{ox}[i]}.
\end{aligned}$$

Suppose, however, that we have included the null event but cannot find any extensions of the regular form core events that are influenced by the offending core events.

In this case, we show that these offending core events are redundant. By our assumption, we can rearrange the events such that all $\mathbf{M}_o$ have the form:

$$\begin{bmatrix} \mathbf{M}_o^{rr} & \mathbf{M}_o^{rx} \\ \mathbf{0} & \mathbf{M}_o^{xx} \end{bmatrix}.$$

Here, the matrix $\mathbf{M}_o^{rr}$ defines how the regular form events influence the extensions of other regular form events. Likewise, $\mathbf{M}_o^{xx}$ defines how the offending events interact, and $\mathbf{M}_o^{rx}$ define how the regular form events influence the offenders. Note that by our assumption, the offending events cannot influence the regular form event extensions. Also, the offending events cannot influence the one step predictions, as these are extensions of the null event:

$$\mathbf{m}_o = \begin{bmatrix} \mathbf{m}_o^{rr} & \mathbf{0} \end{bmatrix}^\top.$$

Using equation 3.1 above, we can use these PSR parameters to calculate any event probability, including the offending core events. By using simple linear algebra, we also see that the offending core events do not influence these predictions ( i.e. , the entries in the $\mathbf{M}_o^{xx}$ matrices have no influence on the final probability). Thus, these events are redundant. $\square$

In fact, all multiplicity automata can be converted into a regular form. The proof used above can be lightly modified to prove proposition 1.

A regular form PSR has fewer free parameters than a general PSR. Because we have included the null event as a core event, the vector $\mathbf{m}_\epsilon$ is no longer a free parameter. By corollary 2, we know that $\mathbf{m}_\epsilon$ must be a vector of all zeros, except for a 1 in the index for the null event. Also, many core extension events found in the $\mathbf{M}_o$ matrices are also core events. If some event $\mathbf{q}_i$ equals $o \circ \mathbf{q}_j$, then the $j$th column of $\mathbf{M}_o$ is all zeros, except for a 1 in the $i$th row. Given the event tree, a regular form PSR has only $|\mathcal{O}| * (k^2) - k(k-1)$ free entries in the core extension matrices and $k$ entries in $\mathbf{q}(\epsilon)$.

The regular form PSR proof also guarantees that when a set of core events for a process is unknown, we will only have to search "short" events before finding a set of core events.

**Corollary 6** *Every rank $k$ stochastic process has a predictive state representation where each core event has length less than $k$.*

PROOF: Because every core event is an extension of some other core event, the maximal length of any event is the maximum number of times a event can be extended. This is $(k-1)$ times. □

## 3.5 The Negative Prediction Problem

It has been noted that the largest difficulty of using PSRs to model stochastic processes is that they may generate "predictions" that are less than zero. This has been called the negative probability problem by Jaeger (2000). We desire a test for whether a proposed PSR could produce a negative probability for some sequence (*i.e.* the boundedness property of a stochastic process is enforced).

We now present a new result that states that checking the boundedness property is undecidable. Before the main result, we must relate PSRs to the stochastic multiplicity automata (S-MA). As mentioned before, a stochastic multiplicity automaton is similar to a PSR, except that it assigns probabilities to complete strings rather than prefixes of arbitrarily long sequences. For a stochastic multiplicity automaton $A$ , we have:

$$\sum_{\mathbf{o}\in\mathcal{O}^*} P_A(\mathbf{o}) = 1,$$

while for a PSR $S$, we have for every $l = 0, 1, 2, \ldots$ :

$$\sum_{\mathbf{o}\in\mathcal{O}^l} P_S(\mathbf{o}) = 1.$$

Given a minimal $k$ state S-MA $A$ on alphabet $\mathcal{O}$, we can construct a $k+1$ state MA $S$ on alphabet $\mathcal{O} \cup \phi$ such that for $\mathbf{o}, \mathbf{b} \in \mathcal{O}^*, l = 1, 2, \ldots$

$$\begin{aligned} P_S(\mathbf{o} \circ \phi^l) &= P_A(\mathbf{o}), \\ P_S(\mathbf{o} \circ \phi^l \circ \mathbf{b}) &= 0. \end{aligned}$$

We can think of $\phi$ as the emission of a "trap" state in an automaton. Note that once a $\phi$ has been emmitted, the only futue observation with nonzero probability is $\phi$.

Assume that $A$ is defined by the parameters $\langle \boldsymbol{\iota}, \boldsymbol{\mu}_o, \boldsymbol{\gamma} \rangle$. We construct the matrix $\boldsymbol{\mu}^* = \mathbf{I} + \sum_{i=1}^{\infty} \Big( \sum_{o \in \mathcal{O}} \boldsymbol{\mu}_o \Big)^i$, where $\mathbf{I}$ is the identity matrix. This matrix can be used to calculate the sum of $f_A$ on all sequences that begin with a common prefix:

$$\begin{aligned}
\sum_{\mathbf{b} \in \mathcal{O}^*} f_A(\mathbf{o} \circ \mathbf{b}) &= \sum_{\mathbf{b} \in \mathcal{O}^*} \boldsymbol{\iota} \boldsymbol{\mu}_{\mathbf{o}} \cdot \boldsymbol{\mu}_{\mathbf{b}} \boldsymbol{\gamma} \\
&= \boldsymbol{\iota} \boldsymbol{\mu}_{\mathbf{o}} \Big( \sum_{\mathbf{b} \in \mathcal{O}^*} \boldsymbol{\mu}_{\mathbf{b}} \Big) \boldsymbol{\gamma} \\
&= \boldsymbol{\iota} \boldsymbol{\mu}_{\mathbf{o}} \Big( \mathbf{I} + \sum_{\mathbf{b} \in \mathcal{O}} \boldsymbol{\mu}_{\mathbf{b}} + \sum_{\mathbf{b} \in \mathcal{O}^2} \boldsymbol{\mu}_{\mathbf{b}} \dots \Big) \boldsymbol{\gamma} \\
&= \boldsymbol{\iota} \boldsymbol{\mu}_{\mathbf{o}} \Big( \mathbf{I} + \sum_{\mathbf{b} \in \mathcal{O}} \boldsymbol{\mu}_{\mathbf{b}} + (\sum_{\mathbf{b} \in \mathcal{O}} \boldsymbol{\mu}_{\mathbf{b}})^2 \dots \Big) \boldsymbol{\gamma} \\
&= \boldsymbol{\iota} \boldsymbol{\mu}_{\mathbf{o}} \boldsymbol{\mu}^* \boldsymbol{\gamma}.
\end{aligned}$$

If $A$ is a stochastic multiplicity automaton where $f_A$ sums to a finite value, then $\boldsymbol{\mu}^*$ must exist, and will have finite values (Denis and Esposito, 2006). Furthermore, if $A$ is minimal rank, then it is calculated as $(\mathbf{I} - \sum_{b \in \mathcal{O}} \boldsymbol{\mu}_b)^{-1}$. Both of these conditions can be checked efficiently (Denis and Esposito, 2006).

We now define our new MA $S = \langle \boldsymbol{\iota'}, \boldsymbol{\mu'}_o \cup \boldsymbol{\mu'}_\phi, \boldsymbol{\gamma'} \rangle$ which computes a stochastic process if $A$ is an S-MA. The first $k$ columns of vector $\boldsymbol{\iota'}$ equal $\boldsymbol{\iota}$, but has one extra column equal to 0:

$$\boldsymbol{\iota}' = \begin{bmatrix} \boldsymbol{\iota} & 0 \end{bmatrix}.$$

For each $(k+1 \times k+1)$ matrix $\boldsymbol{\mu'}_{o \in \mathcal{O}}$, we set the upper left $k \times k$ block equal to $\boldsymbol{\mu}_o$ The entries in the last row and column all equal zero.

$$\boldsymbol{\mu}'_o = \begin{bmatrix} \boldsymbol{\mu}_o & 0 \\ 0 & 0 \end{bmatrix}.$$

The matrix $\boldsymbol{\mu'}_\phi$ is mostly zero, except for the last column. Here, the first $k$ rows are equal to $\boldsymbol{\gamma}$, and the last row equals 1.

$$\boldsymbol{\mu}'_\phi = \begin{bmatrix} 0 & \boldsymbol{\gamma} \\ 0 & 1 \end{bmatrix}.$$

Finally, the vector $\boldsymbol{\gamma}'$ equals $\boldsymbol{\mu}^*\boldsymbol{\gamma}$ in the first $k$ columns, and the last column equals 1.

$$\boldsymbol{\gamma}' = \begin{bmatrix} \boldsymbol{\mu}^*\boldsymbol{\gamma} \\ 1 \end{bmatrix}.$$

By construction of $S$, we can calculate the following for $\mathbf{o} \in \mathcal{O}^*$ : the total value of $f_A$ for all strings with prefix $\mathbf{o}$

$$f_S(\mathbf{o}) = \boldsymbol{\iota}\boldsymbol{\mu}_{\mathbf{o}}\boldsymbol{\mu}^*\boldsymbol{\gamma} = \sum_{\mathbf{b}\in\mathcal{O}^*} f_A(\mathbf{o}\circ\mathbf{b})$$

the value of $f_A(\mathbf{O})$

$$f_S(\mathbf{o}\circ\phi) = \boldsymbol{\iota}\boldsymbol{\mu}_{\mathbf{o}}\boldsymbol{\gamma} = f_A(\mathbf{o}).$$

We also have the following properties: the symbol $\phi$ will repeat indefinitely after being observed once

$$f_S(\mathbf{o}\circ\phi^n) = \boldsymbol{\iota}\boldsymbol{\mu}_{\mathbf{o}}\boldsymbol{\gamma} * 1^{(n-1)} = f_A(\mathbf{o}),$$

and no other symbol but a $\phi$ can follow a $\phi$

$$f_S(\mathbf{o}\circ\phi\circ b\in\mathcal{O}) = \boldsymbol{\iota}\boldsymbol{\mu}_{\mathbf{o}}\mathbf{O}\boldsymbol{\mu}_b\boldsymbol{\gamma} = 0.$$

**Lemma 7** *Given a multiplicity automaton $A$, and $S$ as constructed above, we have the following:*

- *If $A$ is a S-MA, then $S$ is a stochastic process,*
- *$S$ will produce a negative output if and only if $A$ produces a negative output.*

PROOF: It is immediate from the construction that $S$ will have the boundedness and normalization properties of a stochastic process if $A$ is a S-MA. Consistency can be seen by breaking all sequences into cases. First consider sequences that contain noting but observations from $\mathcal{O}$:

$$\begin{aligned} f_S(\mathbf{o}) &= \sum_{b\in\mathcal{O}^*} f_A(\mathbf{o}\circ b) \\ &= f_A(\mathbf{o}) + \sum_{c\in\mathcal{O}} f_S(\mathbf{o}\circ c) \\ &= f_S(\mathbf{o}\circ\phi) + \sum_{b\in\mathcal{O}} f_S(\mathbf{o}\circ b). \end{aligned}$$

Next, consider sequences that end in $\phi$. By construction, the only symbol that can follow a $\phi$ is another $\phi$. The probability that this will happen is $1$. Thus consistency is preserved. Finally, consisder a sequence that ends with some $o \in \mathcal{O}$, but contained a $\phi$ ealier. In this case, the probability $f_S$ assigns to this sequence is 0, as well as any longer sequences that have this as a prefix.

Now, consider the values that $f_S$ can have. These values are either 0, in the case of an observation following a $\phi$, or a sum of values from $f_A$. A sum can only be negative if one of the elements being added is negative. Also, if there is any $\mathbf{o}$ where $f_A(\mathbf{o}) < 0$, then $f_S(\mathbf{o} \circ \phi) < 0$. □

**Theorem 8** *The problem of whether a PSR outputs a negative value for some sequence* $\mathbf{o} \in \mathcal{O}^*$ *is undecidable.*

PROOF: We use the (rather surprising) result that it is undecidable if some multiplicity automaton is a S-MA (Denis and Esposito, 2004). The proof shows that a solution to the S-MA problem yeilds a solution to the post correspondence problem, a well known undecidable problem. Like the case for the PSR, it is easy to check for every condition of a valid S-MA except that the output is always nonnegative.

If we could decide if a PSR outputs negative probabilities, then we could also decide if an S-MA outputs negative probabilities. Simply convert the MA into PSR form using lemma 7 and proposition 1, and decide if the PSR produces negative probabilities. This contradicts this previously published result. □

### 3.5.1 Practical Solutions

In order to use PSRs that were learned on data, we need some way to work around the contingency that they produce a negative probability. In the experiments we present later, we employ a simple technique:

1. If there is some prediction less than $\iota = 1e - 8$, set this prediction to $\iota$.

2. Renormalize such that $\sum_{o \in \mathcal{O}} P(o) = 1$.

3. If an observation that was originaly predicted to occur with probability zero or less is observed, then set the core test vector on the next time step to $\mathbf{q}(\epsilon)$.

A more complex procedure for dealing with this problem was developed by Jaeger *et al.* (2006b). In practice, this occurance is fairly rare, and any reasonable method for handling this yeild comparable results.

# Chapter 4

# Controlled Processes

In this chapter, we study controlled stochastic processes (or controlled process). Like a a stochastic process, a controlled stochastic process generates an observation $o_t$ from some finite set $\mathcal{O}$, on every time step $t$. The difference is that a controlled process also requires an input on every time step. This input $a_t$ comes from a finite set $\mathcal{A}$. We interleave the action and observation sequences, so a joint sequence will have the form $\mathbf{u}_{1:t} = a_1 o_1 a_2 o_2 \ldots a_t o_t$. We call the joint set of actions and observations $(\mathcal{A} \times \mathcal{O})$ the set of *symbols*.

A controlled process gives the probability of an observation sequence, given that a particular action sequence is input. The formal definition of a controlled process is similar to that of a stochastic process (Definition 1):

**Definition 3** *A controlled stochastic process is a function $Y : (\mathcal{A} \times \mathcal{O})^* \rightarrow \mathbb{R}$ satisfying*

- ***Boundedness**: $\forall \mathbf{u} \in (\mathcal{A} \times \mathcal{O})^* : 1 \geq Y(\mathbf{u}) \geq 0$*
- ***Consistency**: $\forall n \in 0, 1, \ldots \forall \mathbf{u} \in (\mathcal{A} \times \mathcal{O})^n, \forall a \in \mathcal{A} : Y(\mathbf{u}) = \sum_{o \in \mathcal{O}} Y(\mathbf{u} \circ ao)$*
- ***Normalization**: $P(\epsilon) = 1$.*

Note that the controlled process does not give a distribution on the actions. It is assumed that actions are generated by a source distinct from the controlled process. Also, if there is only one action, a controlled process is identical to a stochastic process.

Similar to stochastic processes, we accumulate previous actions and observations into a history, called $\mathbf{h}_t$. In general, the probability of the next observation will depend on the entire history, and the next action. We define the function that gives the conditional probability as we did for stochastic processes

$$Y(\mathbf{ao}_{1:t}) = \prod_{i=1}^{t} Y(a_i o_i | \mathbf{h}_{i-1}).$$

We will sometimes write this in a more conventional notation:

$$Y(a_i o_i | \mathbf{h}) \equiv \Pr(o_i | \mathbf{h}, a_i).$$

Cost functions for measuring prediction accuracy are similar. Given a history and an action, we predict the next observation using a model. We then see the next observation, and incur a cost using one of the loss functions mentioned before.

Controlled processes are vital for the theory of reinforcement learning (Sutton and Barto, 1998) and planning under uncertainty (Cassandra *et al.*, 1994). Given an accurate model of a controlled process, an agent can sample several possible futures, and determine the action which is most likely to yield an advantageous series of observations (Kearns *et al.*, 1999). Sometimes it is only necessary to assume that such a model exists, and that the learner has access to the model's state. This is known as *model-free* reinforcement learning (Sutton and Barto, 1998).

## 4.1 Previous Models

There are two main models of controlled processes: the Markov Decision Process (MDP) and the Partially-observable Markov Decision Process (POMDP). The MDP is related to the VMM, while the POMDP is related to the HMM.

### 4.1.1 Markov Decision Processes

An MDP is the most common controlled process model used in reinforcement learning (Puterman, 1994). The model assumes that the probability of the next observation only depends on the current observation and the next action:

$$\Pr(o_{t+1}|\mathbf{h}_t, a_{t+1}) = \Pr(o_{t+1}|o_t, a_{t+1}).$$

The process begins in a unique start state $o_0$. For every action $a$, we define a matrix $\mathbf{P}_a$, where $\mathbf{P}_a[i,j] = \Pr(O_t = j|O_{t-1} = i, A_t = a)$. The observations are said to have the Markov property: given the observation, the history and the future are independent.

The MDP can be extended such that the state of the system is composed of the recent history, not just the most recent observation. As for the VMM, we define the recent history as the context. Such models are relatively unstudied, though McCallum (1995) has used such models successfully in several reinforcement learning tasks. The model, called U-tree, keeps a context tree as discussed for VMMs.

Learning MDPs from data is usually a secondary goal to the problem of using the model for planning. As such, the algorithms used for learning these systems do not necessarily aim to minimize prediction loss. Given sufficient data, most learning algorithms minimize model error, but when data is sparse, it is often the case that the models are biased towards "optimistic" predictions (Brafman and Tennenholtz, 2002). When uncertain transitions are thought to be rewarding, a planning system will be biased to collect more data, thus improving long-term model accuracy.

When the MDP framework is extended to include contexts, learning is also aimed at improving the performance of planning and reinforcement learning. The criterion used to learn a U-tree is to add more context when there is evidence that longer contexts require different optimal actions.

### 4.1.2 Partially-observable Markov Decision Processes

POMDPs are a straightforward extension of hidden Markov models to controlled systems. Like the HMM, a POMDP has a latent state $x_t$ for every time step $t$. Given knowledge of this state, the history and the future of the process are independent. In general, the state of the system cannot be inferred with complete certainty given only past actions and observations. We use $\mathbf{b}(\mathbf{h})$ to represent the belief state: the distribution over the latent state after observing history $\mathbf{h}$. The initial belief state, $\mathbf{b}(\epsilon)$, is a parameter of the model. Like the HMM, we define transition matrices $\mathbf{T}_{ao}$, where $a \in \mathcal{A}$ and $o \in \mathcal{O}$. The entry $\mathbf{T}_{ao}[i, j]$ equals $\Pr(X_{t+1} = j, O_{t+1} = o | X_t = i, A_{t+1} = a)$. We calculate the probability of a sequence, conditioned on the actions, similar to the HMM:

$$\Pr(\mathbf{ao}_{1:n}) = \mathbf{b}(\epsilon)^\top \Big( \prod_{i=1}^{n} \mathbf{T}_{a_i o_i} \Big) \mathbf{1}.$$

All learning algorithms for the HMM can be adapted to work with POMDPs. The most popular algorithm for learning a POMDP is EM, though in most cases, it is assumed that a fairly accurate model is known a priori.

## 4.2 Policies

In order to use a controlled process to generate a sequence, we need a method for providing actions. We model a source of actions as a *policy*. A policy maps a history to a distribution over the next action that will be supplied to the system. It is identical to a stochastic process, with the roles of actions and observations reversed.

**Definition 4** *A Policy is a function* $R : \mathcal{A}, (\mathcal{A} \times \mathcal{O})^* \rightarrow [0, 1]$ *such that:*

$$\forall \mathbf{h} \in (\mathcal{A} \times \mathcal{O})^*, \sum_{a \in \mathcal{A}} R(a|\mathbf{h}) = 1.$$

When a policy is specified along with a controlled process, the entire system can be modeled as a stochastic process. This coupled system generates symbols from a finite set $\mathcal{U} = \{\mathcal{A} \times \mathcal{O}\}$.

## 4.3 PSRs for Controlled Processes

We now present predictive state representations (PSRs) as a method for modeling a controlled process. The PSR was originally presented as such a model, though the framework is trivially extendable to stochastic processes (Littman *et al.*, 2001).

### 4.3.1 Tests

We must extend our definition of event to include the influence of actions on future observations. Specifically, we would like to know the probability of future observations, given that a specific sequence of actions are taken. In the PSR literature, a test is defined by an action sequence and an observation sequence (Littman *et al.*, 2001). The success probability of a test is the probability that the observation sequence will be observed, given that the action sequence will be followed (with probability 1). These tests become the basic unit of prediction for PSRs modelling controlled processes. In most cases, we deal with the situation where there is a policy interacting with the controlled process. Here, this classical definition of test is not immediately applicable because most, if not all, action sequences occur with probabilities less than 1. We expand the classic definition of tests to handle non-deterministic action selection.

A *generalized test* consists of two events on the joint action-observation space $\mathcal{U}$. The *conditioning event* for test g, called $\mathrm{g}^c$, is assumed to occur over the next $l$ time steps. The *conditioned* event $\mathrm{g}^u$ may or may not occur. The conditioned event must a subset of the conditioning event. At any point in the history of a controlled process, we define the probability of a test g succeeding as the probability that the conditioned event occurs in the next $l$ observations, given that the conditioning event occurs. In general, we must calculate this conditional probability sequentially, as the probability of actions and observations later in an event may depend on the outcomes earlier in an event (Bowling *et al.*, 2006).

If $\mathrm{g}^u$ is a single sequence, we can write its success probability as

$$Y(\mathbf{g}|\mathbf{h}) \equiv \prod_{i=1}^{l} \frac{\Pr(\mathbf{g}_i^u|\mathbf{h} \circ \mathbf{g}_{1:i-1}^u)}{\sum_{e:\, \mathbf{g}_{1:i-1}^u \circ e \in \mathbf{g}_{1:i}^c} \Pr(e|\mathbf{h} \circ \mathbf{g}_{1:i-1}^u)}. \tag{4.1}$$

For each future time step, we calculate the probability of the current symbol in the conditioned event, given the history and the earlier symbols in the conditioned event.

This is the numerator of the above equation. We also calculate the probability that the conditioning event will continue to this point of the conditioned event. We divide by this probability.

The most common generalized test is the s-test. Like the s-event, an s-test contains a single sequence of actions and observations as the conditioned event:

$$\mathbf{s}^u = a_1 o_1 a_2 o_2 \ldots a_l o_l.$$

The conditioning event is all sequences that contain the same actions as the conditioned event:

$$\mathbf{s}^c = \{a_1 \mathcal{O} a_2 \mathcal{O} \ldots a_l \mathcal{O}\}.$$

We rewrite equation 4.1 specifically for s-tests:

$$\begin{aligned} Y(\mathbf{s}|\mathbf{h}) &= \prod_{i=1}^{l} \frac{\Pr(a_i o_i | \mathbf{h} \circ \mathbf{ao}_{1:i-1})}{\sum_{b \in \mathcal{O}} \Pr(a_i b | \mathbf{h} \circ \mathbf{ao}_{1:i-1})} \\ &= \prod_{i=1}^{l} Y(o_i | \mathbf{h} \circ \mathbf{ao}_{1:i-1}, a_i). \end{aligned}$$

The last equality holds whenever the probability of the actions are always nonzero, and only depends on the history and earlier symbols in the test. Note that this is exactly the same as how a controlled process iteratively calculates $Y(\mathbf{s})$ using conditional probabilities. We will sometimes refer to an s-test using only the test's conditioned event $\mathbf{u} = a_1 o_1 \ldots a_k o_k$. The null test is also an s-test, where the length of the test is zero.

We now define the e-test, which is the equivalent of an e-event for a controlled processes. E-tests are the “original” test, appearing in the early works on deterministic environments(Rivest and Schapire, 1994). The conditioned event is the set of all sequences where a particular course of action is followed, and the final observation is some $o_l$:

$$\mathbf{e}^u = \{a_1 \mathcal{O} a_2 \mathcal{O} \ldots a_l o_l\}.$$

The conditioning event is all sequences that contain the same actions as the conditioned event:

$$\mathbf{e}^c = \{a_1 \mathcal{O} a_2 \mathcal{O} \ldots a_l \mathcal{O}\}.$$

Unfortunately, there is no simple method for calculating the probability of an e-test other than aggregating the probabilities of all the s-tests that is is composed of:

$$\begin{aligned} Y(\mathbf{e}|\mathbf{h}) &= \sum_{\mathbf{ao}\in\mathbf{e}^u} Y(\mathbf{ao}|\mathbf{h}) \\ &= \sum_{\mathbf{ao}\in\mathbf{e}^u} \prod_{i=1}^{l} \Pr(o_i|\mathbf{h}\circ\mathbf{ao}_{1:i-1}, a_i). \end{aligned}$$

The reason for the difficulty is that while only the final observation matters for the e-test probability, this observation's probability will depend on the intermediate actions observations.

## 4.3.2 PSRs with tests

In order to model controlled processes, we substitute the s-test for the s-event in our definition of a Predictive State Representation. All of the results presented in chapter 3 carry over quite smoothly to the controlled case.

All one step predictors $\mathbf{m}_{ao}$ and core extension matrices $\mathbf{M}_{ao}$ are indexed by both an action and an observation. To calculate the success probability of s-test $\mathbf{g}$, where $\mathbf{g}^u = a_1 o_1 \dots a_n o_n$ we perform the following operation:

$$Y(\mathbf{g}|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \Big(\prod_{i=1}^{n} \mathbf{M}_{a_i o_i}\Big)\mathbf{m}_\epsilon. \tag{4.2}$$

All of the theorems proven for PSRs in the uncontrolled case transfer to the controlled case without much alteration.

We include one modified theorem. For each action $a$, we define $\mathbf{M}_{a*} = \sum_{o\in\mathcal{O}} \mathbf{M}_{ao}$. This calculates the expected probabilities of the core tests on the next time step, given that action $a$ is taken.

**Theorem 9** *A minimum event PSR* $\{\mathbf{q}(\epsilon), \mathbf{M}_{\mathbf{ao}}, \mathbf{m}_\epsilon\}$*, where for* $\mathbf{ao} \in (\mathcal{A}\times\mathcal{O})^*$, $\Pr(\mathbf{ao})$ *is calculated using equation 4.2, and given* $\mathbf{M}_{a*}$ *defined above, models a controlled process if and only if it has the following properties:*

1. $\mathbf{q}(\epsilon)^\top \mathbf{m}_\epsilon = 1$ ,

2. $\forall a \in \mathcal{A},\ \mathbf{M}_{a*}\mathbf{m}_\epsilon = \mathbf{m}_\epsilon$ ,

3. $\forall \mathbf{ao} \in (\mathcal{A} \times \mathcal{O})^*,\ 1 \geq \mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{ao}}\mathbf{m}_\epsilon \geq 0$ .

PROOF: The proof follows the argument of theorem 3. □

It is important to note that although it may be hard to calculate the success probability of an e-test directly from a joint process on both actions and observations, it can be a simple calculation for a PSR. As we saw, the e-test is a sum of many s-tests. Since each s-test $\mathbf{g}$ can be predicted using the dot product $\mathbf{q}^\top \mathbf{m}_\mathbf{g}$, the sum of many s-test success probabilities can also be predicted using a single dot product:

$$\begin{aligned} Y(\mathbf{e}|\mathbf{h}) &= \sum_{\mathbf{ao} \in \mathbf{e}^u} \mathbf{q}^\top \mathbf{m}_{\mathbf{ao}}, \\ &= \mathbf{q}^\top \Big( \sum_{\mathbf{ao} \in \mathbf{e}^u} \mathbf{m}_{\mathbf{ao}} \Big), \\ &= \mathbf{q}^\top \mathbf{m}_\mathbf{e}. \end{aligned}$$

After a one-time cost of computing $\mathbf{m}_\mathbf{e}$, finding the probability of the e-test is as efficient as an s-test.

## 4.4 Linear Policies

We treat policies much like we have treated the processes that policies control. Like a controlled process, the next action a policy outputs can be an arbitrary function of the entire history of previous observations and actions. This is an infinite set of possible policies. In order to analyze the long-term behavior of a policy, we need a measure of its complexity similar to the number of tests in a PSR.

We do this by introducing the a-test. An a-test is equivalent to an s-test, but with the role of actions and observations reversed. A length $n$ a-test $\mathbf{f}$ composed of conditioned event $\mathbf{f}^u$ and conditioning event $\mathbf{f}^c$ will have the following form:

$$\begin{aligned}\mathbf{f}^u &= o_1 a_1 o_2 a_2 \ldots o_n a_n; \\ \mathbf{f}^c &= \{o_1 \mathcal{A} o_2 \mathcal{A} \ldots o_n \mathcal{A}\}.\end{aligned}$$

The null test is also included in the set of a-tests. Note that these tests begin with an observation instead of an action.

For a given policy $R$, the probability of an a-test $\mathbf{f}$ succeeding is:

$$Y(\mathbf{f}|\mathbf{h} \circ a_0) = \prod_{i=1}^{n} R(a_i|\mathbf{h} \circ a_0 \circ \mathbf{oa}_{1:i-1} \circ o_i).$$

Note that we need histories to end with an action for our a-tests to be defined. It is usually possible to consider a-test probabilities from the last action, except for the first time step. Here, the history is empty. For this special case, the probability of an a-test is undefined.

A policy has rank $k$ if the probability of any a-test is a linear combination of the probability of a set of $k$ a-tests. After any history $\mathbf{h}$, we call the probability vector of the a-tests $\mathbf{r}(\mathbf{h})$.

Like a PSR, we will need to define functions to maintain the probabilities of the success of the a-tests. The one-step tests calculate the probability of the test $\Pr(a|\mathbf{h}, o)$ for all $a$ and $o$. This is a linear function parameterized with a weight vector $\mathbf{v}_{oa}$. Also, we need to calculate the vectors for the a-test extensions. We arrange the weight vectors for the core test extensions of $oa$ into matrix $\mathbf{V}_{oa}$. Because we cannot use a-tests to predict the first action, we must deal with the choice of this first action as a special case: $\Pr(a|\epsilon) = R(a|\epsilon)$. We construct the initialization matrix $\mathbf{V}_a$ and the initialization vector $\mathbf{r}(\epsilon)$, and impose the following relationship:

$$R(a|\epsilon) * \mathbf{r}(a)^\top = \mathbf{r}(\epsilon)^\top \mathbf{V}_a.$$

Like PSRs, a linear policy can be expressed in a regular form where all a-tests are shorter than the policy's rank.

In order to gain intuition on linear policies, consider the class of reactive policies. A reactive policy selects the next action based on only the previous observation:

$$R(a|\mathbf{h}_t) = \Pr(a|o_t).$$

Because all the one step a-tests have constant success probabilities, such a policy has rank 1. Consider the a-test for action $a$ after observation $o$, given the null test as the only core test of the policy:

$$\begin{aligned}\Pr(f_{oa}|\mathbf{h} \circ a') &= \mathbf{r}(\mathbf{h} \circ a')^\top \mathbf{V}_{ao}\mathbf{v}_\epsilon, \\ &= \Pr(\epsilon|\mathbf{h} \circ a')\Big(\Pr(a|o)\Big)1, \\ &= \Pr(a|o).\end{aligned}$$

In general, a $k$th-order reactive policy (one that selects the next action according to the previous $k$ action-observation pairs) will have rank $|\mathcal{A} \times \mathcal{O}|^k$. Linear policies are also capable of representing policies encoded as options (Sutton *et al.*, 1999), finite state automata (Hansen, 1997), and many other forms.

Now we examine the behavior of a linear PSR coupled with a linear policy. It is easy to show that an $n$ state Markov decision process that is controlled by a $m$ state controller will become a $m * n$ state Markov chain. We show a similar result for PSRs.

**Theorem 10** *A rank $k$ controlled process coupled with a rank $f$ linear policy can be modeled as a stochastic process $P : (\mathcal{A} \times \mathcal{O})^* \to [0, 1]$ with rank no greater than $k * f$.*
PROOF: We show this result by constructing the parameters of the coupled process from a parameterization of the controlled process and policy.

The probability that some $ao$ occurs immediately is

$$\begin{aligned}P(ao) &= \Pr(a)\Pr(o|a) \\ &= \mathbf{r}_\epsilon^\top \mathbf{V}_a \mathbf{v}_\epsilon * \mathbf{q}(\epsilon)^\top \mathbf{M}_{ao}\mathbf{m}_\epsilon \\ &= \mathbf{r}_\epsilon^\top \mathbf{v}_a * \mathbf{q}(\epsilon)^\top \mathbf{m}_{ao}.\end{aligned}$$

We calculate the probability of longer sequences inductively. Assume that any sequence $\mathbf{ao}_{1:k}$ of length $k$ can be written as a product of $\mathbf{q}(\epsilon)^\top \mathbf{m}_{\mathbf{ao}_{1:k}}$ and $\mathbf{r}(\epsilon)^\top \mathbf{v}_{\mathbf{ao}_{1:k-1}a_k}$.

We now show how to calculate probability of a length $k+1$ sequence $s_{k+1}$:

$$\begin{aligned}
P(\mathbf{ao}_{1:k} \circ a_{k+1}o_{k+1}) &= P(a_1o_1 \dots a_ko_k) \\
&\quad * \Pr(a_{k+1}|a_1o_1 \dots a_ko_k) \\
&\quad * \Pr(o_{k+1}|a_1o_1 \dots a_ko_ka_{k+1}) \\
&= \mathbf{r}(\epsilon)^\top \mathbf{v}_{\mathbf{ao}_{1:k-1}a_k} * \mathbf{q}(\epsilon)^\top \mathbf{m}_{\mathbf{ao}_{1:k}} \\
&\quad * \frac{\mathbf{r}(\epsilon)^\top \mathbf{V}_{\mathbf{ao}_{1:k}a_{k+1}} \mathbf{v}_\epsilon}{\mathbf{r}(\epsilon)^\top \mathbf{v}_{\mathbf{ao}_{1:k-1}a_k}} \\
&\quad * \frac{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{ao}_{1:k+1}} \mathbf{m}_\epsilon}{\mathbf{q}(\epsilon)^\top \mathbf{m}_{\mathbf{ao}_{1:k}}} \\
&= \mathbf{r}(\epsilon)^\top \mathbf{v}_{\mathbf{ao}_{1:k}a_{k+1}} * \mathbf{q}(\epsilon)^\top \mathbf{m}_{\mathbf{ao}_{1:k+1}}.
\end{aligned}$$

Thus, the probability any sequence of action-observation pairs occurs can be described as the product of two dot products. We rewrite this as the single dot product:

$$\begin{aligned}
\mathbf{q}(\epsilon)^\top \mathbf{m}_s \cdot \mathbf{r}(\epsilon)^\top \mathbf{v}_s &= \mathbf{m}_s^\top \Big(\mathbf{q}(\epsilon) \cdot \mathbf{r}(\epsilon)^\top\Big) \mathbf{v}_s \\
&= \mathbf{m}_s^\top \mathbf{C}(\epsilon) \mathbf{v}_s \\
&= \sum_{i,j} \mathbf{C}(\epsilon)[i,j] \mathbf{m}_s[i] \mathbf{v}_s[j] \\
&= \mathbf{c}(\epsilon)^\top \boldsymbol{\omega}_s,
\end{aligned}$$

where $\mathbf{c}(\epsilon)$ is the vectorized outer product of $\mathbf{q}(\epsilon)$ and $\mathbf{v}(\epsilon)$, and $\boldsymbol{\omega}_s$ is the vectorized outer product of $\mathbf{m}_s$ and $\mathbf{v}_s$. Because any sequence probability is a dot product of a $(k*f)$-dimensional vector, this system cannot have a rank more than $(k*f)$. $\square$

## 4.5 Modeling Controlled Processes as Uncontrolled

Theorem 10 provides a justification for modeling a controlled process using an uncontrolled stochastic process. If we knew that actions were being sampled from some fixed, finite dimension policy, then we could model the joint probabilities of the actions and observations, rather than the conditional probabilities of the observations, given the actions. This can be advantageous, as there is a much larger body of work on models for sequence prediction in the uncontrolled case.

For every history $\mathbf{h}$, our process predicts $\Pr(ao|\mathbf{h})$. This process is trivially capable of making conditioned predictions:

$$\Pr(o|\mathbf{h}, a) = \frac{\Pr(ao|\mathbf{h})}{\sum_{b \in \mathcal{O}} \Pr(ab|\mathbf{h})}.$$

However, this procedure violates a common rule of thumb for machine learning: *never model more than the problem requires*. Basically, our model may waste modeling power on predicting the actions, when the modeling power should be focused on predicting the conditional probabilities of the observations. Theorem 10 suggests that this problem should be worse as the complexity of the policy grows. "Fortunately," it is common in the PSR literature to use a policy that selects actions uniformly at random (Singh *et al.*, 2003; James, 2005; McCracken and Bowling, 2006), though see the work of Bowling *et al.* (2006) for an exception. For such a policy, the rank of the controlled system and the uncontrolled system is identical. We will discuss the merits of these two learning styles in chapter 6.

## 4.6 The System Identification Problem

Given an unknown or partially modeled controlled process, it would be useful to have a policy that will take actions which are useful for learning the structure of the process. We would like this policy gather this information using as few samples as possible. This is known as the system identification problem. For MDPs, the problem has been fairly well studied (Duff, 2003; Kearns and Singh, 1998; Brafman and Tennenholtz, 2002). In particular, Kearns and Singh (1998) show that one can design a policy that can efficiently learn any portion of an MDP that it has a non-negligible chance of visiting in the future.

The case for POMDPs and PSRs is a bit more grim. Even in the case of POMDPs with deterministic observation and state transition functions, building a policy that effi-

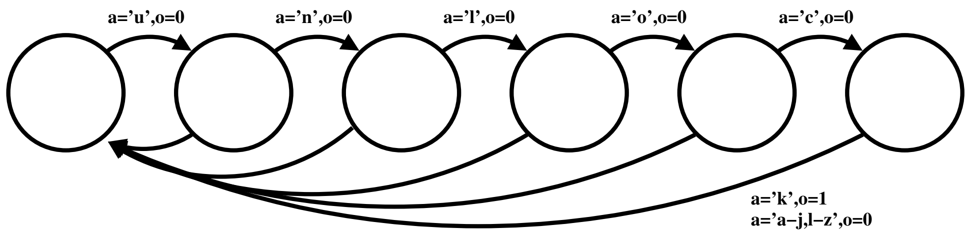


Figure 4.1: An example Partially-observable Markov decision process that is difficult to learn. The observation is always “0” unless the action sequence “unlock” is input.

ciently learns the system is impossible (see Freund *et al.* (1993) for a brief review). The problem can be seen in the POMDP in figure 4.1. This deterministic POMDP has six states, but may require on the order of $26^6$ actions to identify the structure. Here the observations are $\{0, 1\}$, and the actions are lower case letters of the alphabet. The system always produces a $0$ as the observation unless the action sequence $u, n, l, o, c, k$ is input. If any other input is given, the system resets back to an original state without providing any feedback. This model can be thought of as a password protection system. Given such a model, but with an unknown “password” sequence, we will be unable to predict which sequence of observations will yield the “1” observation unless we have stumbled upon it at random. Because this problem is hard for POMDPs, it must also be hard for PSRs. This follows because PSRs can model any system descibed by a POMDP.

# Chapter 5

# The System Matrix

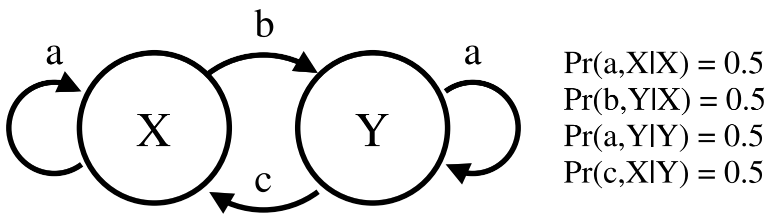
In the previous chapters, we presented the PSR as a representation for stochastic processes, and described their properties and relations to other models of stochastic processes. In this chapter, we present the System Matrix, a data structure that most PSR learning algorithms are built around.

## 5.1 System Matrix Definition

We present another representation of a stochastic process called the *system matrix* $\mathcal{D}$. The rows of $\mathcal{D}$ index the infinite possible histories of a process, and the columns index the s-tests (or s-events for an uncontrolled system). We order the histories and tests from short to long, and alphabetically for sequences of the same length. We abuse notation by indexing the system matrix using both sequence indices and the sequences themselves:

$$\mathcal{D}(i, j) = \mathcal{D}(\mathbf{h}_i, \mathbf{q}_j) = \Pr(\mathbf{q}_j|\mathbf{h}_i).$$

Note that there are a countably infinite number of rows and columns in $\mathcal{D}$. Some histories may be possible to generate using the stochastic process. When some history occurs with probability 0, then that entire row of $\mathcal{D}$ will be zeros. We call such rows unreachable. An example of a System matrix can be seen in figure 5.1.



(a) The stochastic process from 2.2.

| | $\epsilon$ | a | b | c | aa | ab | ac | ba | bb | bc | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\epsilon$ | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0 | $\frac{1}{4}$ | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | ... |
| a | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0 | $\frac{1}{4}$ | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | ... |
| b | 1 | $\frac{1}{2}$ | 0 | $\frac{1}{2}$ | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | 0 | 0 | 0 | ... |
| c | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| aa | 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0 | $\frac{1}{4}$ | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | ... |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ | $\ddots$ |

(b) The System matrix for this process. We assume the process starts in state X.

Figure 5.1: An example stochastic process and its system matrix.

First we examine some of the basic properties of a valid System Matrix as outlined by McCracken and Bowling (2006). These properties are all the result of the system matrix containing probabilities generated by a stochastic process. These must hold for any history $\mathbf{h}_i \in (\mathcal{A} \times \mathcal{O})^*$, and s-test with conditioned event $\mathbf{u}_j$:

- **Bounded Range**: $0 \leq \mathcal{D}(\mathbf{h}_i, \mathbf{u}_j) \leq 1$,
- **Internal Consistency**: $\forall a \in \mathcal{A}, : \mathcal{D}(\mathbf{h}_i, \mathbf{u}_j) = \sum_{b \in \mathcal{O}} \mathcal{D}(\mathbf{h}_i, \mathbf{u}_j \circ a \circ b)$,
- **Unreachable Rows**: If $\mathcal{D}(\epsilon, \mathbf{u}_j) = 0$, then $\mathcal{D}(\mathbf{u}_j, \cdot) = 0$.

If $\mathbf{h}_i$ is reachable, then two additional properties hold:

- **Null Test Identity**: $\mathcal{D}(\mathbf{h}_i, \epsilon) = 1$ ,
- **Conditional Probability**: $\forall \mathbf{o}_j \in \mathcal{O}^*, b \in \mathcal{O} : \mathcal{D}(\mathbf{h}_i \circ b, \mathbf{o}_j) = \frac{\mathcal{D}(\mathbf{h}_i, b \circ \mathbf{o}_j)}{\mathcal{D}(\mathbf{h}_i, b)}$.

The range constraint restricts the entries in $\mathcal{D}$ to be probabilities. The null test constraint enforces the normalization constraint of a stochastic process. The internal consistency constraint ensures that the probabilities within a single row meet the requirements of a stochastic process. The conditional probability constraint is required to maintain consistency between different rows of the matrix.

Note that the system matrix is an alternate definition of a stochastic process, not a model of one. Also, this representation is over-complete: every entry in $\mathcal{D}$ can be inferred from the first row:

$$\mathcal{D}(i,j) = \frac{\mathcal{D}(\epsilon, \mathbf{h_i} \circ \mathbf{q}_j)}{\mathcal{D}(\epsilon, \mathbf{h_i})}.$$

The system matrix for PSRs was first mentioned by Singh *et al.* (2004), though there is a long history of similar structures. Jaeger (1998) described a similar structure called the conditional continuation probabilities of a process. He defines a collection of infinite-dimensional vectors of the probabilities of future events, given differing initial histories. These vectors are precisely the rows of the system dynamics matrix. Jaeger was inspired by earlier work by Ito *et al.* (1992). They used a similar collection of vectors to study the minimal number of states that a HMM requires to model a particular process, and to decide if two HMMs generate the same process.

### 5.1.1 System Matrix Rank

Although a system matrix is infinite, it may still have finite rank. We define the rank of $\mathcal{D}$ as the minimum dimension $r$ such that for every row $i$ and column $j$ in $\mathcal{D}$, we can define two $r$-dimensional vectors $\mathbf{v}_i, \mathbf{w}_j$ such that

$$\mathcal{D}(i,j) = \mathbf{v}_i^\top \mathbf{w}_j.$$

The system matrix provides a particularly appealing interpretation of PSRs (Singh *et al.*, 2004). A PSR with $r$ tests will produce a system matrix with rank no greater than $r$, where $\mathbf{v}_i = \mathbf{q}(\mathbf{h}_i)$ and $\mathbf{w}_j = \mathbf{m}_j$. For the example in figure 5.1, the rank of the system equals 2. This can be seen by using the core events $\{\epsilon, b\}$. Note that every row in the system matrix can be reconstructed as a linear combination of the two rows for core tests $\epsilon$ and $b$. The converse of this statement is also true: every stochastic process whose System matrix has rank $r$ can be modeled by a PSR with $r$ core tests (Singh *et al.*, 2004).

Low rank decompositions have long been used as a regularization and modeling technique for high-dimensional data (Hastie *et al.*, 2001; Landauer *et al.*, 1998; Dailey *et al.*, 2002). In particular, there is a long history of using low rank approximations in order to model systems in the control theory community (Fazel *et al.*, 2004). In the control theory, a popular type of system assumes that there is an underlying linear process which generates real-valued observations with Gaussian noise. These systems can be modeled using a low rank structure similar to the system matrix. A predictive representation of such systems has been studied by Rudary and Singh (2006).

## 5.2 Hankel Matrix Representation

Multiplicity Automata (of which PSRs are a subset) have long been associated with a *Hankel matrix*[1] (see Beimel *et al.* (2000) for an overview). Given a PSR $P$, and its function $f_P : (\mathcal{A} \times \mathcal{O})^* \rightarrow [0, 1]$, we define the Hankel matrix of $P$ as

$$\mathcal{F}(\mathbf{u}_i, \mathbf{u}_j) = f_P(\mathbf{u}_i \circ \mathbf{u}_j) = \mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{u}_i} \mathbf{M}_{\mathbf{u}_j} \mathbf{m}_\epsilon.$$

The Hankel matrix has rows indexed as the prefix of a sequence, and the columns index possible suffixes.

The rank of a stochastic process' Hankel matrix is equal to the dimension of the smallest multiplicity automaton that can model it (Denis and Esposito, 2004). Obviously, the system matrix and the Hankel matrix representations of a process are quite similar. Given a system matrix $\mathcal{D}$ and Hankel matrix $\mathcal{F}$ constructed from the same process, we can easily see their relation. For unreachable rows, both $\mathcal{D}$ and $\mathcal{F}$ are 0. For

[1]In linear algebra, a Hankel matrix is a symmetric matrix where skew-diagonal entries are identical. In our context, this property does not hold, although these matrices have similar properties.

reachable rows of the system matrix, we have:

$$\begin{aligned}
\mathcal{D}(\mathbf{u}_i, \mathbf{u}_j) &= \mathbf{q}(\mathbf{u}_i)^\top \mathbf{M}_{\mathbf{u}_j} \mathbf{m}_\epsilon \\
&= \Big(\frac{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{u}_i}}{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{u}_i} \mathbf{m}_\epsilon}\Big) \mathbf{M}_{\mathbf{u}_j} \mathbf{m}_\epsilon \\
&= \Big(\frac{1}{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{u}_i} \mathbf{m}_\epsilon}\Big) \mathcal{F}(\mathbf{u}_i, \mathbf{u}_j)
\end{aligned}$$

Basically, the system matrix weighs rows of the Hankel matrix inversely proportional to the probability of the row's history occurring. From a learning perspective, this is somewhat counter-intuitive. The more a particular history is seen, the more accurately test probabilities can be estimated. If a history is very uncommon, the entries of $\mathcal{F}$ in this row will be close to zero, while the row of $\mathcal{D}$ will have the same range as higher probability rows. Thus, the entries in $\mathcal{F}$ will be more similar to the frequency of the samples required to learn them. We will address this issue in more depth in chapter 6. For the rest of the chapter, we focus on the Hankel matrix representation, though the results are easily translated to the system matrix.

## 5.3 The Steady-State Hankel Matrix

While the system matrix and the Hankel matrix are a useful conceptual tool for understanding PSRs, this representation is not particularly useful in many learning applications. It is common to receive one contiguous sequence generated by a stochastic process, or a sequence generated by coupling a controlled process and some policy. In this situation, every history is only experienced once. This is clearly not enough information to estimate most entries in $\mathcal{D}$ or $\mathcal{F}$. In order to learn from this type of information, Jaeger (2000) proposed constructing an alternate Hankel matrix where instead indexing rows by a complete history, we index them with a context. We call this new matrix the steady-state Hankel matrix $\bar{\mathcal{F}}$. Such a change in representation requires a rigorous justification, which we now focus on.

We model single sequence learning as observing an increasingly long history $\mathbf{h}_i$, $i = 1, 2, \ldots \infty$. For an uncontrolled stochastic process, we define the steady-state probability of a length $l$ event $\mathbf{u}$ as

$$\bar{P}(\mathbf{u}) = \lim_{t\to\infty} \frac{1}{t} \sum_{i=1}^{t} P(\mathbf{u}|\mathbf{h}_i).$$

The steady-state success probability of some test $\mathbf{g}$, called $\bar{Y}(\mathbf{g})$ is defined as in chapter 4, where we use $\bar{P}$ to define a stochastic process on actions and observations. This definition of $\bar{P}(\mathbf{u})$ immediately raises questions:

- When will $\bar{P}(\mathbf{u})$ be defined? In other words, does this limit as $t$ goes to infinity converge to a single number for any history generated by the process?
- For a given process, will $\bar{P}(\mathbf{u})$ always be the same value, or does it depend on the history being generated?

These questions have been addressed when the history is generated by either a VMM or HMM ( see Grimmett and Stirzaker (1982); Puterman (1994) ). In both of these model classes, the probabilities of future sequences only depend on the current state. Thus, if the frequencies of particular state sequences converge, then so do the probabilities of observation sequences. It is well known that finite state Markov processes (VMMs and HMMs) will settle on some steady-state distribution on the Markov states (Grimmett and Stirzaker, 1982). The distribution that the process settles into may differ, depending on the specific history that has been generated by the process. This behavior is caused by the possibility that the process makes a "one-way" transition to a state where other states cannot be reached. The steady-state distribution will be determined only by the states that are reachable as time goes on.

There are processes where $\bar{P}()$ will not converge, however. For instance, consider the process where $\mathcal{O} = \{x, y\}$ and

$$\Pr(O_t = x) = \begin{cases} 1 & \text{if } \lfloor \log_3(t) \rfloor \text{ is even,} \\ 0 & \text{otherwise.} \end{cases}$$

This process produces a sequence that begins as $x^2y^6x^{18}y^{54}\ldots$, with the length of the strings of repeating observations growing geometrically over time. For large values of $t$, the value of $\bar{P}(x)$ ranges between $\frac{1}{3}$ and $\frac{2}{3}$. This range does not diminish as $t$ approaches infinity.

In OOM theory, it is generally assumed that the process generating the history is a stationary and ergodic OOM (Jaeger *et al.*, 2006b). Being stationary means that the initial core test probabilities are also the expected core test probabilities in the future. Recall that we define the matrix $\mathbf{M}_*$ as $\sum_{o\in\mathcal{O}}\mathbf{M}_o$. This matrix computes the expected probabilities of the core events on the next time step. For a stationary system, we have:

$$\mathbf{q}(\epsilon)^\top\mathbf{M}_* = \mathbf{q}(\epsilon)^\top\mathbf{M}_*^n = \mathbf{q}(\epsilon)^\top.$$

The ergodic assumption is that $\bar{P}()$ exists, and is unique for the process.

### 5.3.1 A Limit Property of PSRs

We take a more careful look at what conditions are sufficient to guarantee that $\bar{\mathcal{F}}$ will converge. This analysis was first done in the work of Wiewiora (2005).

**Theorem 11** *If* $\mathbf{h}$ *has been generated by a valid PSR, for any* $\mathbf{q}_i$ *that constitute a minimal set of core tests for the PSR,* $\bar{P}(\mathbf{q}_i)$ *converges almost surely.*

PROOF: At any point in the evolution of the stochastic process, the system's future is characterized by the (unknown) vector $\mathbf{q}(\mathbf{h}_t)$.

We track the expectation of $\mathbf{q}(h_{t+1})$ using the matrix $\mathbf{M}_*$ as defined in chapter 3. Recall from theorem 4 that $\mathbf{M}_*$ has spectral value one, with at least one eigenvector for eigenvalue one.

Define $\mathbf{M}^*$ as $\lim_{N\to\infty}\sum_{i=1}^N \frac{1}{N}\mathbf{M}_*^i$. This matrix determines the expected value of $\bar{P}(\mathbf{q})$, given the current value of $\mathbf{q}(\mathbf{h})$. Because of the spectral properties of $\mathbf{M}_*$ we have outlined before, we know that this limiting matrix exists (Horn and Johnson, 1986). Since $\mathbf{M}^*$ converges, it must satisfy $\mathbf{M}^* = \mathbf{M}_*\mathbf{M}^*$.

Note that for any possible $\mathbf{q}(\mathbf{h}_i)$,

$$\mathbf{q}(\mathbf{h}_i)^\top\mathbf{M}^* = \mathbf{q}(\mathbf{h}_i)^\top\mathbf{M}_*\mathbf{M}^* = E[\mathbf{q}(\mathbf{h}_{i+1})^\top\mathbf{M}^*].$$

This property is sufficient for $\mathbf{q}(\cdot)^\top \mathbf{M}^*$ to be a martingale (Grimmett and Stirzaker, 1982). Also, note that $\mathbf{q}(\cdot)^\top \mathbf{M}^*$ has bounded variance, due to the fact that the entries in $\mathbf{q}()$ always range between 0 an 1. Using these two results and standard martingale theory, we can show that the random variable $\mathbf{q}(\cdot)^\top \mathbf{M}^* = E[\bar{P}(\mathbf{q})]$ converges to a fixed point almost surely (Grimmett and Stirzaker, 1982). $\square$

We have proven that any PSR will eventually have an average core test probability vector. Call this vector $\bar{\mathbf{q}}$.

**Corollary 12** *If the average success probabilities of all core tests converge to $\bar{\mathbf{q}}$, then the average success probabilities of all events converge almost surely.*

PROOF: At each time step, the probability that an event $\mathbf{g}$ will occur over the next observations is determined by the probabilities of the core tests at that time:

$$P(\mathbf{g}|\mathbf{h}) = \mathbf{q}(\mathbf{h})^\top \mathbf{m}_\mathbf{g}.$$

The long-term average success probability can likewise be written as a linear function of the average success probabilities of the core events:

$$\begin{aligned}
\bar{P}(\mathbf{g}) &= \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{T} P(\mathbf{g}|\mathbf{h}_i) \\
&= \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{T} \mathbf{q}(\mathbf{h}_i)^\top \mathbf{m}_\mathbf{g} \\
&= \Big( \lim_{T\to\infty} \frac{1}{T} \sum_{i=1}^{T} \mathbf{q}(\mathbf{h}_i)^\top \Big) \mathbf{m}_\mathbf{g} \\
&= \bar{\mathbf{q}}^\top \mathbf{m}_\mathbf{g}.
\end{aligned}$$

Thus, if the vector $\bar{\mathbf{q}}$ converges to some fixed values, so does $\bar{P}(\mathbf{g})$. $\square$

## 5.4 Subsampling the Hankel Matrix

For practical applications, it is not possible to deal with the entire Hankel matrix. Instead, we subsample the rows and columns of $\mathcal{F}$. In order to make sure that we have captured the complete behavior of the stochastic process we would like our submatrix $\mathbf{F}$ of the Hankel matrix to have the same rank as the entire matrix.

Fortunately, we can construct some $\mathbf{F}$ with the same rank as the entire Hankel matrix $\mathcal{F}$ , using only "short" tests for the rows and columns. First, recall that a set of core tests will have linearly independent columns in $\mathbf{F}$ . By corollary 6, we know that if a PSR has rank $r$, then there is a set of core tests where each test has length less than $r$. We choose the columns of $\mathbf{F}$ as those that are indexed by these short core tests.

The rows of $\mathcal{F}$ correspond to possible histories in $\mathcal{D}$. As we can select linearly independent columns of $\mathcal{F}$ as core tests, we can select $r$ linearly independent rows of $\mathcal{F}$ to form a basis for predicting all of $\mathcal{F}$ . We call a set of linearly independent rows of $\mathcal{F}$ a set of *core histories*. Likewise, independent rows of $\bar{\mathcal{F}}$ are called *core contexts*.

We define a regular form for core contexts similar to that of the core tests:

**Definition 5** *A set of core histories $\{\mathbf{c}_1 \ldots \mathbf{c}_r\}$ is in regular form if for every $\mathbf{c}_i$, either $\mathbf{c}_i = \epsilon$, or there is some $\mathbf{c}_j$ and $u \in \mathcal{U}$ such that $\mathbf{c}_i = \mathbf{c}_j \circ u$.*

Like regular form core tests, these core histories can be organized into a tree with labeled edges, where each node represents a core history. The s-test for this history is determined by reading the labels from the root to the node. This tree is not a context tree as described for VMMs; whereas nodes in the same branch of a VMM context tree share a common suffix, nodes in the core history tree share a common prefix.

**Lemma 13** *For any minimal set of core tests $\{\mathbf{q}_1 \ldots \mathbf{q}_r\}$, we can find a set of regular form histories $\{\mathbf{c}_1 \ldots \mathbf{c}_r\}$ such that the rank of $\mathbf{F}$ formed with these contexts as rows and these core tests as columns equals $r$.*

PROOF: Proof by construction. First, we choose the row of $\mathbf{F}$ as the row indexed by the null test $\epsilon$. On every step $j = 2 \ldots r$, we add a context $\mathbf{c}_j = \mathbf{c}_i \circ u \in \mathcal{U}$ where $\mathbf{c}_i$ is already in our set of core histories. We assume that there is some $\mathbf{c}_i$ and $u$ such that the vector $\mathcal{F}(\mathbf{c}_i \circ u, \mathbf{q}_1 \ldots \mathbf{q}_r)$ is linearly independent from the earlier rows of $\mathbf{F}$. This process will terminate after $r$ iterations, as we cannot find more than $r$ linearly independent $r$-dimensional vectors.

Assume that for some iteration $j < r$, we cannot find such a $\mathbf{c}_i$ and $u$ such that $\mathcal{F}(\mathbf{c}_i \circ u, \mathbf{q}_1 \dots \mathbf{q}_r)$ is linearly independent. We argue that this contradicts our premise that this set of core tests is minimal. Call the partial matrix of $j-1$ core histories and $r$ core tests $\mathbf{F}$. Because we did not find a linearly independent row to add to $\mathbf{F}$, for each $\mathbf{c}_i$, and $u$ there is some $(j-1 \times 1)$ vector $\mathbf{w}_{iu}$ such that:

$$\mathcal{F}(\mathbf{c}_i \circ u, \{\mathbf{q}_r 1 \dots \mathbf{q}_r\}) = \mathbf{w}_{iu}^\top \mathbf{F}$$

For each $u$, gather the $j-1$ vectors $\mathbf{w}_{iu}^\top$ into a $(j-1 \times j-1)$ matrix $\mathbf{W}_u$ With this we can calculate :

$$\mathbf{F}_u = \mathcal{F}(\{\mathbf{c}_1 \dots \mathbf{c}_{j-1}\} \circ u, \{\mathbf{q}_1 \dots \mathbf{q}_r\}) = \mathbf{W}_u \mathbf{F}.$$

Note that by the definition of a PSR, $\mathbf{F}_u$ also equals $\mathbf{F}\mathbf{M}_u$. We define the row vector $\mathbf{w}_\epsilon$ as a vector of all zeros, except for a 1 in the the row index where the null test is used as a context in $\mathbf{F}$. We have that for any test g, $\mathcal{F}(\text{g}, \{\mathbf{q}_1 \dots \mathbf{q}_r\}) = \mathbf{w}_\epsilon \mathbf{F}_{\mathbf{g}}$.

Now, assume that the test $\mathbf{x} = u_1 \dots u_n$ is one where the vector $\mathcal{F}(\mathbf{x}, \{\mathbf{q}_1 \dots \mathbf{q}_r\})$ is linearly independent of the rows of $\mathbf{F}$. This yields:

$$\begin{aligned}
\mathcal{F}(\mathbf{x}, \{\mathbf{q}_1 \dots \mathbf{q}_r\}) &= \mathbf{F}(\epsilon, :)\mathbf{M}_{\mathbf{x}} \\
&= \mathbf{F}(u_1, :)\mathbf{M}_{\mathbf{x}_{2:n}} \\
&= \mathbf{w}_\epsilon \mathbf{W}_{u_1} \mathbf{F}\mathbf{M}_{\mathbf{x}_{2:n}} \\
&= \mathbf{w}_\epsilon \mathbf{W}_{u_1} \mathbf{W}_{u_2} \mathbf{F}\mathbf{M}_{\mathbf{x}_{3:n}} \\
&\vdots \\
&= \mathbf{w}_\epsilon \mathbf{W}_{\mathbf{x}} \mathbf{F} \\
&= \mathbf{w}_{\mathbf{x}} \mathbf{F}.
\end{aligned}$$

This is a contradiction, as this row can be defined as a linear combination of the rows in $\mathbf{F}$. □

An interesting consequence of this proof is that we can make the same predictions as a PSR using linear combinations of context "probabilities", instead of future test probabilities. In other words, the parameters $(\mathbf{w}_\epsilon, \{\mathbf{W}_u\})$ are just as valid a representation as the standard PSR parameters $(\mathbf{q}_\epsilon, \{\mathbf{M}_u\})$. This avenue of investigation has not been explored, although James (2005) has derived an alternate method for using the context as well as the tests to make PSR predictions.

Although we have shown that all core tests and histories in a full rank submatrix $\hat{\mathcal{F}}$ are short, there are still as many as $\mathcal{U}^{r-1}$ tests that can be rows or columns. It would be desirable to have a method that can automatically search for rows or columns that will be linearly independent. Some research has examined the case where we construct $\hat{\mathcal{F}}$ by requesting particular values of test probabilities $f_p(\mathbf{u} \in \mathcal{U}^*)$ (Beimel *et al.*, 2000; James and Singh, 2004). The value $f_p(\mathbf{u})$ can be used to fill in all entries $\mathcal{F}(\mathbf{u}_i, \mathbf{u}_j)$ such that $\mathbf{u}_i \circ \mathbf{u}_j = \mathbf{u}$. Unfortunately, we cannot use these queries to efficiently recover a full rank set of core tests and histories. Recall the "unlock" POMDP in figure 4.1. Unless the test for $u0n0l0o0c0k1$ or $u0n0l0o0c0k0$ is queried, all observed values of $\mathcal{F}$ will be zero. Finding one of the unlocking tests could take on the order of $\mathcal{A}^{r-1}$ queries. This is a worst-case bound; it is very likely that far fewer queries will be required to find $r$ good core tests and histories.

A heuristic for finding core tests and histories is proposed in the work of James and Singh (2004). They propose an incremental scheme, where core tests and histories are added by examining all extensions of the current core histories and core tests. If no extensions can be found that are linearly independent, then the search terminates. Though this heuristic is not guaranteed to find all core tests, it has worked well in several small experiments (James and Singh, 2004).

Another approach to finding a submatrix of $\mathcal{F}$ is more data driven. Given a training sequence of length $T$, there will only be a finite number of tests that succeed more than some threshold $t$ number of times [2]. All of these above-threshold tests can be identified using the suffix tree data structure in $O(T)$ time. A submatrix $\hat{\mathcal{F}}$ constructed from these tests will have size polynomial in $T$. This approach has been used to identify

[2]The exact number of test successes are $\frac{T^2}{2}$

useful core events in the OOM literature (Kretzschmar, 2003; Jaeger *et al.*, 2006b).

## 5.5 Summary

In this chapter we have presented the system matrix, and the highly related Hankel matrix. These data structures model the probabilities of various tests occuring in a given process. The structure of these matrices allows one to infer quite a bit about a PSR that will be capable of representing it. Most importantly, the rank of this matrix equals the number of core tests required to model the underlying process. Also, we show that although the system matrix is infinite, we can determine if it has low rank by checking only the early rows and columns of the matrix.

We have also examined issues related to estimating the system matrix from training data. We introduce the steady-state Hankel matrix, which captures the average probabilities that tests succeed. If the system generating the sequence we are modeling is a PSR, then this steady-state Hankel matrix will converge to some fixed point.

These results provide a theoretical foundation for the PSR learning algorithms presented in the next chapter.

# Chapter 6

# Learning Predictive State Representations

In this chapter, we present some learning algorithms that have appeared in the PSR and OOM literature, along with an in-depth analysis of their behavior and the choices made in different components of the algorithms.

As opposed to learning an HMM with the EM algorithm, most algorithms for learning PSR with an approximated system matrix are *convergent*: if a PSR is used to generate an infinitely long training sequence, the learned PSR will converge to the same stochastic process (Jaeger, 2000). This is surprising, given that PSRs can model more processes than HMMs.

Also, we perform a series of experiments testing PSRs against more traditional models of stochastic processes. In previous work, PSRs have been compared to HMMs and POMDPs, but there has been little effort at comparing their performance to variable-ordered Markov models. This thesis provides the first in-depth comparison between these two approaches. As we will see, the VMM is surprisingly competitive with the PSR in small training sizes. This is true even when the VMM is not capable of exactly modelling the stochastic process generating the data.

## 6.1 The General Framework

Learning a PSR can be be broken down into two steps, called *Discovery* and *Learning* (Singh *et al.*, 2003). Discovery is the process of choosing the core tests, whose success probabilities will become the state of the learned model. The learning problem is choosing the parameters of the PSR ($\mathbf{q}(\epsilon), \mathbf{M}_u, \mathbf{m}_\epsilon$), in order to best model the training data.

The algorithms we analyze are offline, system matrix based methods. These algorithms all share the property that they take a training sequence, and approximate some submatrix $\bar{\mathbf{D}}$ of the steady state system matrix or $\bar{\mathbf{F}}$ of the steady state Hankel matrix. The PSR parameters are derived from this matrix using the constraint that these matrices should have rank $r$ to produce an $r$-test PSR.

There has been some research on online learning methods for PSRs (Singh *et al.*, 2003; Tanner and Sutton, 2005; McCracken and Bowling, 2006). These algorithms incrementally update the model parameters based on immediate feedback from the process generating the data. With the exception of the method proposed by McCracken and Bowling (2006), these algorithms must be given a set of core tests *a priori*. A systematic investigation of these methods as an alternative to offline approaches is a worthy endeavour, but will not be done in this work.

## 6.2 Approximating Steady-State Probabilities

In order to build an approximate System or Hankel matrix, we need a method for using the data in a training sequence to approximate probabilities that some event or test will succeed. Assume that we are given a training sequence $\mathbf{h}_t = a_1 o_1 \dots a_t o_t$ of alternating actions and observations.

In order to estimate the probability of a length $k$ s-event $\mathbf{g} = a'_1 o'_1 \dots a'_k o'_k$, we can simply count the number of times this sequence appears, and divide by the number of opportunities the sequence had to appear:

$$\hat{P}_t(\mathbf{g}) = \frac{1}{t-k} \sum_{i=1}^{t-k} I(\mathbf{h}_{i+k} = \mathbf{h}_i \circ \mathbf{g}),$$

where $I()$ is the indicator function, which is 1 when the argument is true and 0 otherwise. We call this estimate the empirical frequency of an event. We index $\hat{P}$ by the history length, though when we use the entire history, the subscript is dropped. We extend the estimator to conditional probabilities in the straightforward fashion:

$$\hat{P}(\mathbf{g}|\mathbf{f}) = \frac{\hat{P}(\mathbf{f} \circ \mathbf{g})}{\hat{P}(\mathbf{f})}$$

This estimator was first used to approximate system matrix entries by Jaeger (1998).

Approximating test success probabilities is a bit more involved. In general, the only method for approximating the success probability of an s-test is to use several estimated sequence probabilities. For an s-test s consisting of $k$ actions and observations $a'_1 o'_1 \ldots a'_k o'_k$, we estimate the success probability as:

$$\hat{Y}(\mathbf{s}) = \prod_{i=1}^{k} \hat{P}(o'_i | a'_1 o'_1 \ldots a'_i) = \prod_{i=1}^{k} \frac{\hat{P}(a'_1 o'_1 \ldots a'_i o'_i)}{\hat{P}(a'_1 o'_1 \ldots a'_i)}.$$

It may be possible to simplify this estimation procedure if we have knowledge of the policy. An especially simple case is that of blind policies: those that do not depend on the observation histories (Bowling *et al.*, 2006). Given the blind policy $R$, we can estimate the success probability of an s-test as:

$$\hat{Y}(\mathbf{s}) \approx \prod_{i=1}^{k} \frac{\hat{P}(a'_i o'_i | a'_1 o'_1 \ldots a'_i)}{R(a'_i)} = \frac{\hat{P}(\mathbf{s})}{\prod_{i=1}^{k} R(a'_i)} \approx \frac{\hat{P}(\mathbf{s})}{\hat{P}(a_1 \mathcal{O} \ldots a_k \mathcal{O})}.$$

This estimation procedure is similar to the one used in James and Singh (2004) and Wolfe *et al.* (2005). Note that in order to use this estimate, the policy must choose each action with probability at least $\epsilon > 0$.

Note that when a particular sequence of actions has not been frequently observed in the training sequence, the $\hat{Y}()$ estimates of tests that condition on these actions may be very unstable. This problem could be mitigated with smoothing techniques, such as the KT estimator discussed in chapter 2. Another alternative, which we explore in this chapter, is to model the joint action and observation probabilities $\hat{P}()$, instead of test success probabilities $\hat{Y}()$.

## 6.3 The Suffix History Algorithm

The Suffix History Algorithm is the first offline algorithm for learning PSRs from a single sequence of data (Wolfe *et al.*, 2005). It is an adaptation of an earlier algorithm that learned a PSR from many separate sequences generated from the same controlled process (James and Singh, 2004). Both algorithms were heavily influenced by early work on learning OOMs (Jaeger, 2000). The description of the algorithm given here is based on the presentation by James (2005).

The algorithm proceeds in two steps. First, an approximate submatrix of the system matrix $\hat{\mathbf{D}}$ is built from data. This matrix is square, with every column being a core test and every row a core context. The core histories are used to make core extension submatrices $\hat{\mathbf{D}}_{ao}$, where

$$\hat{\mathbf{D}}_{ao}(\mathbf{c}_i, \mathbf{q}_j) = \hat{Y}(ao \circ \mathbf{q}_j | \mathbf{c}_i),$$

for each core context $\mathbf{c}_i$, and core test $\mathbf{q}_j$

The model parameters $\mathbf{M}_{ao}$ are approximated using matrix inversion:

$$\begin{aligned} \hat{\mathbf{D}}\mathbf{M}_{ao} &= \hat{\mathbf{D}}_{ao}, \\ \mathbf{M}_{ao} &= \hat{\mathbf{D}}^{-1}\hat{\mathbf{D}}_{ao}. \end{aligned}$$

The first equation is given by our derivation of the PSR, and the second equation is possible because by construction, $\hat{\mathbf{D}}$ is a square, full-rank matrix. The remaining parameters are estimated from the data and the one-step prediction vectors $\mathbf{m}_{ao}$:

$$\begin{aligned} \mathbf{q}(\epsilon) &= \hat{Y}(\mathbf{q}), \\ \mathbf{m}(\epsilon) &= \frac{1}{|\mathcal{A}|}\sum_{ao \in \mathcal{U}} \mathbf{m}_{ao}. \end{aligned}$$

Note that if the null test is one of the core tests, the $\mathbf{m}_{ao}$ vectors are already components in $\mathbf{M}_{ao}$.

### 6.3.1 Discovery of Core Tests

At the first iteration the algorithm approximates a submatrix of $\mathcal{D}$, denoted $\hat{\mathbf{D}}_1$, containing all contexts of length one and all tests of length one. Then it approximates the rank of this submatrix, $r_1$. The first set of core tests, $Q_1$, are chosen as any $r_1$ linearly independent columns in $\mathbf{D}_1$. The set of core contexts, $C_1$, are any $r_1$ linearly independent rows in $\mathbf{D}_1$.

For the next iteration, we compute the submatrix of $\mathcal{D}$ with columns corresponding to the union of the tests in $Q_1$ and all one-step extensions of these test. The rows correspond to the union of the core contexts in $C_1$ and all one-step extensions of them. The rank $r_2$, core-tests $Q_2$ , and core-histories $C_2$ are then found. This process is repeated until the rank remains the same for two consecutive iterations (this is The Suffix History Algorithm's stopping condition). If the algorithm stops after iteration $i$, then it returns $Q_i$ and $C_i$ as the discovered core-tests and core-histories respectively.

### 6.3.2 Robust Estimation

We have glossed over the issue of estimating the rank of an approximate system matrix $\hat{\mathbf{D}}$. If all the entries were perfect, we could use standard linear algebra techniques such a Gaussian elimination to find the rank of $\hat{\mathbf{D}}$. However, since the entries are "noisy" due to estimation error, we will not be able to do this. Instead, the Suffix History algorithm uses a technique proposed by Jaeger (1998) to estimate the rank of the matrix. The rank approximation algorithm relies on the number of samples that have gone in to the estimates of entries in $\hat{\mathbf{D}}$; if the entries are estimated from a lot of data, it is more likely that the rank of the matrix is due to actual structure rather than estimation errors. We keep track of the attempt probabilities of various entries in $\hat{\mathbf{D}}$ in the matrix $\mathbf{N}$. When $\hat{\mathbf{D}}$ is composed of event probabilities, the entries in $\mathbf{N}$ are simply the number of times that the event could have been observed in the indexed row of $\mathcal{D}$. For context $\mathbf{c}_i$ of size $x$ and s-event $\mathbf{q}_j$ of size $y$, we have $\mathbf{N}(\mathbf{c}_i, \mathbf{q}_j) = (t - x - y)\hat{P}_{t-y}(\mathbf{c}_i)$.

As usual, finding the values of $\mathbf{N}$ for learning a controlled process is a bit more involved. For a blind policy, we measure $\mathbf{N}(\mathbf{c}_i, \mathbf{q}_j)$ as the number of times the context $\mathbf{c}_i$ is observed in the history, followed by the action sequence given in the s-test $\mathbf{q}_j = a_1 o_1 \ldots a_y o_y$ :

$$\mathbf{N}(\mathbf{c}_i, \mathbf{q}_j) = (t - x - y)\hat{P}_{t-y}(\mathbf{c}_i \circ a_1 \mathcal{O} \ldots a_y \mathcal{O})$$

This rank approximation scheme has not been extended to non-blind policies. Under more general policies, the simple estimator given above will not give a correct count of the number of times $\mathbf{q}_j$ in context $\mathbf{c}_i$ could have succeeded.

The pseudo-code for estimating the rank of some $\hat{\mathbf{D}}$ is given as algorithm 1. The algorithm is based on a rough approximation given by Jaeger (1998), for finding the smallest singular value not caused by sampling noise. The suffix history algorithm uses the estimated rank in two ways. First, the rank is used to decide how many core tests and histories to keep. Second, the rank provides a heuristic criterion for removing redundant tests and histories. Tests and histories are iteratively removed by choosing the test that minimizes the ratio of the largest singular value and the $r$-th singular value. The intuition is that the inverse of a matrix can have highly skewed values if there is a large discrepancy in their singular values. For a full rank, square matrix $\mathbf{D}$, we can calculate its inverse via the SVD:

$$\begin{aligned} \mathbf{D} &=_{(SVD)} \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}, \\ \mathbf{D}^{-1} &= \boldsymbol{V}^{\top}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^{\top}, \end{aligned}$$

where $\boldsymbol{\Sigma}$ is a diagonal matrix of the singular values of $\mathbf{D}$. If some singular values are small, their inverse may be arbitrarily large. The pseudocode for robust suffix history discovery is given in algorithm 2.

Below we discuss some details and implications of the suffix-history algorithm.

- The algorithm requires the specification of a parameter $\kappa$, which effectively controls the number of core tests the algorithm will discover. Experiments have shown that for very large training sequences, a value of $\kappa = 0.1$ will usually result in the correct number of core tests (Wolfe *et al.*, 2005). If we knew how many core tests to expect, we can modify the algorithm to control for the number of core tests explicitly.

- The algorithm does not use much of the system matrix to estimate PSR parameters. For an $r$-test PSR, at most $(1 + |\mathcal{A}|\,|\mathcal{O}|)r^2$ entries are used: $r^2$ in the matrix $\hat{\mathbf{D}}$, plus an additional $r^2$ for each $\hat{\mathbf{D}}_{ao}$.

- As discussed before, using the system matrix may aggravate estimation error. No matter how many times a context is observed, each row of $\mathbf{D}$ will still range between 0 and 1. It may be more reasonable to "weight" the rows by how often the contexts are observed, and thus, how reliable the row's estimates are. This is done implicitly in the Hankel matrix representation.

```
input  : A system submatrix $\mathbf{D}$ of size $n \times m$,
         test attempt counts $\mathbf{N}$,
         confidence parameter $\kappa$
output : Estimated rank $r$

1 if $n > m$ then
2     Set $\mathbf{D} \leftarrow \mathbf{D}^\top$
3 end
4 Calculate $\epsilon = \frac{1}{m*n} \sum_i \sum_j \sqrt{\frac{\mathbf{D}(i,j)(1-\mathbf{D}(i,j))}{\mathbf{N}(i,j)(1-\kappa)}}$ ;
5 Calculate $\sigma_{cutoff} = \epsilon ||\mathbf{D}||_\infty$ ;
6 Calculate the singular values of $\mathbf{D}$: $\sigma_1 \ldots \sigma_m$ ;
7 return $r = |\{\sigma_i > \sigma_{cutoff}\}|$
```

**Algorithm 1**: Suffix History Algorithm: EstimateRank()

```
input  : Training sequence $\mathbf{h}_t$,
         confidence parameter $\kappa$
output:  Estimated system matrix $\hat{\mathbf{D}}$,
         Core extension system matrices $\{\hat{\mathbf{D}}_{ao \in \mathcal{U}}\}$
1  $r_1 = 0; C_1 = \{\epsilon\}$ ; $Q_1 = \{\epsilon\}$ ;
2  for $i = 1$ to $\infty$ do
3      $C' = \{C_i \cup C_i \circ ao \in \mathcal{U}\}$;
4      $Q' = \{Q_i \cup ao \circ Q_i\}$;
5      $r_{i+1}$ = EstimateRank($\hat{\mathbf{D}}(C', Q')$);
6      Remove redundant contexts;
7      while $|C'| > r$ do
8          $C' \leftarrow C' /$ argmin$_{\mathbf{c} \in C'} \frac{\sigma_1(\hat{\mathbf{D}}(C'/\mathbf{c}, Q'))}{\sigma_r(\hat{\mathbf{D}}(C'/\mathbf{c}, Q'))}$
9      end
10     Remove redundant tests;
11     while $|Q'| > r$ do
12         $Q' \leftarrow Q' /$ argmin$_{\mathbf{q} \in Q'} \frac{\sigma_1(\hat{\mathbf{D}}(C', Q'/\mathbf{q}))}{\sigma_r(\hat{\mathbf{D}}(C', Q'/\mathbf{q}))}$
13     end
14     $C_{i+1} = C'$ ; $Q_{i+1} = Q'$ ;
15     if $r_{i+1} \leq r_i$ then
16         return
           $\hat{\mathbf{D}}(C_{i+1}, Q_{i+1}); \{\hat{\mathbf{D}}_{ao}(C_{i+1}, Q_{i+1}) = \hat{\mathbf{D}}(C_{i+1}, ao \circ Q_{i+1})\}$
17     end
18 end
```

**Algorithm 2**: Suffix History Algorithm: Discovery()

## 6.4 Extending the Definition of Core Tests

The suffix history algorithm keeps a system matrix of size $r \times r$ to learn an $r$ test PSR. In the process of doing this, many potentially useful rows and columns of $\mathbf{D}$ are thrown away. It would be quite useful to find a use for this extra information.

We do this by extending our definition of the core tests that compose the process state. Instead of them being the success probabilities of a single s-test, we will allow the core tests to be a linear combination of a collection of s-test success probabilities. We define *generalized core tests* by a *test characterizer* matrix $\mathbf{R}$ (Rosencrantz *et al.*, 2003; Jaeger *et al.*, 2006b). $\mathbf{R}$ is an $m \times r$ matrix where the rows index s-tests in some submatrix $\mathbf{D}$ or $\mathbf{F}$, and the columns index the generalized core tests. We can write the value of the generalized core tests for different contexts in a matrix:

$$\mathbf{Q} = \mathbf{F}\mathbf{R}.$$

We can also address the issue of estimating PSR parameters using additional contexts. If the approximated Hankel matrix $\hat{\mathbf{F}}$ has additional rows, we can find $\mathbf{M}_{ao}$ that approximately solve the following equation:

$$\hat{\mathbf{F}}\mathbf{R}\mathbf{M}_{ao} \approx \hat{\mathbf{F}}_{ao}\mathbf{R}. \tag{6.1}$$

There is no obvious best method for approximating equation 6.1. It is common to approximate $\mathbf{M}_a o$ using the least squares error criterion:

$$\begin{aligned}\mathbf{M}_{ao} &= \text{argmin}_{\mathbf{M}} ||\mathbf{Q}\mathbf{M} - \mathbf{Q}_{ao}||_F^2 \\ &= (\mathbf{Q}^\top\mathbf{Q})^{-1}\mathbf{Q}^\top\mathbf{Q}_{ao}.\end{aligned}$$

This generalized notion of test has the downside that the values in $\mathbf{q}(\mathbf{h})$ are no longer interpretable as individual test success probabilities. This disconnection between the state of the process and the observed future may make it harder to verify the correctness of the state. This problem is often mentioned as a criticism of latent state models (HMMs and POMDPs), and addressing it is one of the primary motivations behind PSRs

(Littman *et al.*, 2001). If this is a concern, it is possible to project a learned generalized-test PSR back into a standard PSR by constructing the learned PSR's system matrix and then search for core tests using standard techniques (Jaeger *et al.*, 2006b).

## 6.5 The T-PSR Algorithm

The first algorithm to use generalized tests was developed by Rosencrantz *et al.* (2003). It is called the transformed PSR algorithm, or TPSR. The algorithm was originally designed to work with a large, sparse system matrix, where each row is a unique history $\mathbf{h}_i$, and each column is an e-event. Because each history is observed only once in our learning setting, the entries in this system matrix are either 0 or 1. It was initially designed for modelling an uncontrolled stochastic process, and it is difficult to extend this algorithm to the controlled case. Although this type of matrix performed well on the authors' experiments (Rosencrantz *et al.*, 2003), these matrices perform poorly on the experiments I have conducted.

Here, we present a modified version of the TPSR algorithm that can work with any system matrix. The key insight into the TPSR algorithm is that the SVD basis for $\mathbf{D}$ yields a good state space for predicting the tests in $\mathbf{D}$. Also, there is a straightforward progression from simpler to more complex models by keeping additional SVD bases as generalized core tests.

The algorithm takes as input a sufficiently large subsample of the system matrix $\mathbf{D}$, the extensions of this system matrix $\mathbf{D}_{ao}$, and the desired rank of the learned PSR. The authors suggest that the rank be chosen using some form of validation on test data. The size of $\mathbf{D}$ should be as large as computational resources and training sample size permit. One technique is to use as rows and columns every event or test that has succeeded some threshold number of times (Jaeger *et al.*, 2006b).

```
input  : System Matrix $\mathbf{D}$, $\{\mathbf{D}_{ao}\}$ ,
         Steady-state test probabilities $\mathbf{d}$,
         PSR Rank $r$
output: PSR parameters $\mathbf{q}(\epsilon), \{\mathbf{M}_{ao}\}, \mathbf{m}_\epsilon$
1  Make $\mathbf{X} = \begin{bmatrix} \mathbf{D} \\ \mathbf{D}_{a_1 o_1} \\ \vdots \end{bmatrix}$;
2  Use SVD to get $\mathbf{X} =_{(SVD)} \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}$;
3  Set test characterizer $\mathbf{R} = \boldsymbol{V}[1:r,:]^\top$;
4  Set $\mathbf{Q} = \mathbf{DR}$;
5  Set $\mathbf{Q}^\dagger = (\mathbf{Q}^\top\mathbf{Q})^{-1}\mathbf{Q}^\top$ ;
6  for $a \in \mathcal{A}\ o \in \mathcal{O}$ do
7      Set $\mathbf{M}_{ao} = \mathbf{Q}^\dagger\mathbf{D}_{ao}\mathbf{R}$
8  end
9  Set $\mathbf{q}(\epsilon) = \mathbf{dR}$;
10 Set $\mathbf{m}_\epsilon = \mathbf{Q}^\dagger\mathbf{1}$;
```

**Algorithm 3**: Transformed-PSR Learning Algorithm

## 6.6 Efficiency Sharpening Procedure

The last learning we discuss is the efficiency sharpening procedure of Jaeger *et al.* (2006b). This method produces a series of learned PSRs that are intended to have improved accuracy and robustness. It was originally presented as a method for learning OOMs, but we present it here in the PSR learning framework. The presentation is somewhat terse, and the reader is referred to the original work for a more thorough treatment and theoretical justification.

Previous approaches to learning OOMs and PSRs, including the two mentioned above, only make use of short-term statistics of the training sequence. No information is kept that is longer than the longest test and context in $\mathbf{D}$. This is in contrast to the E-M procedure for learning HMMs. In the EM algorithm, we explicitly calculate the expected state distribution for every time step, given the entire sequence. The efficiency sharpening procedure keeps some information from the entire training sequence in order to build an improved model.

The ES algorithm takes as input a PSR learned using some other method, and a training sequence $\mathbf{h}_t$. First, the algorithm directly estimates a new core test basis $\mathbf{Q}$ using statistics from the entire sequence. For every position $i$ in the training sequence $\mathbf{h}_t$ we calculate the vector

$$\mathbf{m}_i^r = \frac{1}{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{h}_{i:t}} \mathbf{m}_\epsilon} \left( \mathbf{M}_{\mathbf{h}_{i:t}} \mathbf{m}_\epsilon \right)^\top .$$

These vectors can be thought of as the state of a reversed process (Jaeger *et al.*, 2006b), where the roles of the $\mathbf{q}$ and $\mathbf{m}$ vectors are exchanged. This reversed process calculates the probability of a sequence from the last element to the first. The $\mathbf{m}^r$ vectors are used to build the core test basis for the next estimated PSR. For every context $\mathbf{c}_j$ of length $l$, we calculate

$$\begin{aligned} \mathbf{q}_j &= \sum_{i=l}^{t} I(\mathbf{h}_{i-l} \circ \mathbf{c}_j = \mathbf{h}_i) \mathbf{m}_i^r \\ \mathbf{q}_{aoj} &= \sum_{i=l+1}^{t} I(\mathbf{h}_{i-l-1} \circ ao \circ \mathbf{c}_j = \mathbf{h}_i) \mathbf{m}_i^r \end{aligned}$$

We gather these $M$ contexts into a new basis:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_M \end{bmatrix}; \qquad \mathbf{Q}_{ao} = \begin{bmatrix} \mathbf{q}_{ao1} \\ \vdots \\ \mathbf{q}_{aoM} \end{bmatrix}.$$

Jaeger *et al.* (2006b) proved that if the original PSR was the process generating the data, then then there is some $R$ such that $\mathbf{Q} = \mathbf{DR}$, and that of all $\mathbf{Q}$ of this form, this one will have lowest variance over samples of $\mathbf{h}_t$.

Although this procedure tends to produce more accurate models after every iteration, the new models do not necessarily improve any measure of performance, such as log likelihood of the training sequence. In practice, it is suggested that the ES procedure be stopped when some form of validation error begins to increase, or when it appears that the ES procedure is no longer improving the training sequence likelihood (Jaeger *et al.*, 2006b).

Code for the Efficiency-sharpening procedure is available for download at:

`http://www.faculty.iu-bremen.de/hjaeger/OOM/OOMTool.zip`

All experiments were performed using this code.

## 6.7 Shortcomings of System Matrix Learning Procedures

All system matrix learning algorithms for PSRs suffer from some setbacks. The algorithms we discussed are convergent in the sense that if there is a true model, with an infinitely long training sequence, they will find it (Jaeger, 2000). However, none of the algorithms are particularly efficient. This may be due to a number of factors. One important factor is how the system matrix $\mathbf{D}$ relates to the training sequence $\mathbf{h}$.

The suffix history and TPSR algorithms are concerned with deriving a basis that will approximate $\mathbf{D}$ well. This may be in conflict with predicting sequences. The major cause is that the learned basis$\mathbf{Q}$ may not represent the probabilities of the core tests in an actual sequence. They are, in essence, overfitting the matrix representation of the stochastic process.

## 6.8 Experiments

We now present some experiments testing the modeling capabilities of both PSRs and more traditional models. We choose two domains, reflecting the range of sequence prediction problems. The first task is predicting the future output of several simple POMDPs that have appeared in the literature. These tasks have been the primary benchmark for evaluating the performance of POMDP and PSR algorithms. The second task is a file prediction task. VMMs have long been used for modeling file content for the purpose of compression. This task has recently been studied as a prediction task (Begleiter *et al.*, 2004).

We choose to evaluate the performance of VMMs and PSRs on these tasks. Notably, we do not analyze the performance of HMM learning algorithms such as E-M. There are several reasons for this. The most important is that most applications that involve HMMs do not explicitly address the sequence prediction problem. Most often, the task HMMs are used for is labeling a sequence with the most likely latent HMM state. VMMs and PSRs cannot be used for this task: their states are either trivially derived from the sequence, or not interpretable as a single random variable. Also, there has been extensive study of PSRs and OOMs as compared to HMMs and POMDPs learned via E-M (Jaeger, 1998; Singh *et al.*, 2003; Kretzschmar, 2003; Rosencrantz *et al.*, 2003; James and Singh, 2004; Wolfe *et al.*, 2005; McCracken and Bowling, 2006; Jaeger *et al.*, 2006b). In general, the results show that PSRs tend to learn better models, especially with a large amount of training data. Finally, the EM algorithm commonly used to train HMMs requires a random initialization. This is in contrast to all the PSR and VMM learning algorithms, which are deterministic.

VMMs have been extensively used for sequence prediction. Despite their popularity, little work has been done comparing their performance to PSRs (see Singh *et al.* (2003) for the one exception). The experiments presented here are the first comprehensive attempt at comparing these two algorithms.

### 6.8.1 Methods and Algorithms

For both experiments, we present the learning algorithms with training sequences taken from the first part of a longer sequence. After building their models, they are evaluated on the remaining portion of the sequence. Performance is evaluated using both the log loss and squared loss.

Many models we consider do not explicitly predict observations, given actions. These include the CTW algorithm for training a VMM, and the efficiency sharpening procedure. When training these models on controlled processes, we use the following procedure:

- Create new alphabet $\mathcal{U} = \mathcal{A} \times \mathcal{O}$
- Build model that predicts $u_i = a_i \circ o_i$
- Factor out the probability of the action:

$$\hat{P}(o|a) = \frac{\hat{P}(a \cdot o)}{\sum_{o' \in \mathcal{O}} \hat{P}(a \circ o')}$$

### 6.8.2 Learning POMDPs

We take several example domains used in previous PSR and POMDP learning papers. Most of these are available for download from a web page maintained by Anthony Cassandra:

http://www.cs.brown.edu/research/ai/pomdp/examples/index.html .

The specific POMDPs and some details are given in table 6.1. For the experiments, we sampled $10,000$ or $15,000$ actions and observations from each POMDP. Of these samples, all but the last $5,000$ were used for training, with the last samples used

Table 6.1: Example POMDPs used in experiments

| NAME | POMDP STATES | SYSTEM RANK | ACTIONS | OBSERVATIONS |
|---|---|---|---|---|
| 1D MAZE | 4 | 4 | 2 | 2 |
| 4X3 MAZE | 11 | 10 | 4 | 6 |
| 4X4 MAZE | 16 | 16 | 4 | 2 |
| CHEESE MAZE | 11 | 11 | 4 | 7 |
| FLOAT-RESET | 5 | 5 | 2 | 2 |
| PAINT | 4 | 2 | 3 | 2 |
| SHUTTLE | 8 | 7 | 3 | 5 |
| TIGER | 2 | 2 | 3 | 2 |

for testing. Throughout the sampling, we select actions uniformly at random. For every point in the test sequence, we have the learned models predict the next observation, given the next action. This is then compared against the actual observation, and either log loss or squared loss is assigned to the prediction. All results are averaged over 20 trials.

We try four algorithms for learning stochastic processes. The context tree weighting algorithm learns a variable-order Markov model (VMM). The algorithm only has one parameter: the maximum length of the context. We set this value to 6, though the algorithm performs nearly identically for the depth set higher than 4. We also try three PSR learning algorithms: Efficiency sharpening, transformed PSR and suffix-history. We modified the code of suffix history to accept the model dimension as an input. All of these algorithms were told the correct number of core tests for the domain, but not an exact set of core tests.

The results in tables 6.2 and 6.3 clearly show the CTW algorithm and the efficiency sharpening procedure to be the best performers. It is surprising to note that the VMM learned by the CTW algorithm is not powerful enough to precisely model many of these POMDPs. Thus these models must incur some error over the Bayes optimal error. Both the suffix history algorithm and the TPSR algorithm perform poorly on many

Table 6.2: POMDP results with training size =5,000. We test Context Tree Weighting (CTW), Efficiency Sharpening (ES), Suffix History, and Transformed PSR (TPSR). The parentheses contain: (square loss, log loss). The best score for any domain is underlined.

| DOMAIN | CTW | ES | SUFFIX-HIST | TPSR |
|---|---|---|---|---|
| 1D MAZE | (0.0655, 0.1825) | (0.0619, 0.1786) | (0.2228, 1.5748) | (0.3302, 2.6104) |
| 4X3 MAZE | (0.2389, 0.7432) | (0.2299, 0.9067) | (0.6942, 1.7937) | (0.4898, 3.5040) |
| 4X4 MAZE | (0.0177, 0.0644) | (0.0362, 0.2376) | (0.0268, 0.2091) | (0.1864, 1.2719) |
| CHEESE | (0.0343, 0.1060) | (0.4858, 1.3084) | (0.7345, 1.9482) | (0.6456, 2.4951) |
| FLOATRESET | (0.0529, 0.1530) | (0.0528, 0.1527) | (0.0552, 0.1698) | (0.1812, 0.7202) |
| PAINT | (0.0592, 0.1669) | (0.0607, 0.1699) | (0.0622, 0.1731) | (0.0962, 0.2980) |
| SHUTTLE | (0.1519, 0.4415) | (0.1268, 0.4033) | (0.6398, 1.6098) | (0.5010, 3.7746) |
| TIGER | (0.2519, 0.6978) | (0.2434, 0.6792) | (0.3567, 2.2829) | (0.3213, 2.1523) |

domains. It is worth noting that both algorithms tend to be tested after learning on much larger training sequences.

### 6.8.3 Calgary Corpus

In the second experiment, we learn models of a collection of files. This collection, called the Calgary Corpus, is a dataset of representative academic text files, used to measure compression performance. It is available for download at

ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/ .

We treat each file as an uncontrolled stochastic process, where each byte of the file is an observation. Our task is to predict the last half of each file, given the first half as training. We evaluate model performance using the summed log loss of predicting the remainder of the file. Recall that the log loss is proportional to the number of bits it takes to represent the sequence, given the model. Thus, it is the natural error measurement for this task. This prediction task was investigated by Begleiter *et al.* (2004), where various VMM algorithms were evaluated. In these experiments, the CTW algorithm reliably performed among the best predictors.

Note that files do not behave as a stationary process. Many have headers and tails

Table 6.3: POMDP results for training size: 10,000 (square loss, log loss)

| Domain | CTW | ES | SUFFIX-HIST | TPSR |
|---|---|---|---|---|
| 1D Maze | (0.0623, 0.1764) | (0.0616, 0.1769) | (0.1980, 0.5957) | (0.2422, 1.5714) |
| 4x3 Maze | (0.2336, 0.7152) | (0.2128, 0.7651) | (0.6940, 1.7924) | (0.4432, 2.4819) |
| 4x4 Maze | (0.0172, 0.0614) | (0.0225, 0.1196) | (0.0193, 0.0940) | (0.1131, 0.7247) |
| Cheese | (0.0290, 0.0862) | (0.1121, 0.3234) | (0.7344, 1.9465) | (0.4753, 1.9439) |
| FloatReset | (0.0528, 0.1526) | (0.0527, 0.1529) | (0.0559, 0.1829) | (0.1493, 0.6587) |
| Paint | (0.0602, 0.1689) | (0.0606, 0.1697) | (0.0605, 0.1745) | (0.1114, 0.3932) |
| Shuttle | (0.1438, 0.4193) | (0.1232, 0.3716) | (0.6399, 1.6099) | (0.4686, 2.3325) |
| Tiger | (0.2490, 0.6190) | (0.2446, 0.6818) | (0.2943, 1.5116) | (0.3357, .1518) |

Table 6.4: Calgary Corpus results. Total log loss of predicting last half of the file. The best cost for each file is in bold

| Data set | CTW | ES |
|---|---|---|
| Bib | **86233.5** | 168979.6 |
| Book1 | **673797.6** | 1069073.5 |
| Book2 | **581571.8** | 955277.2 |
| Geo | 164959.4 | **158822.6** |
| News | **418159.7** | 646138.6 |
| Obj1 | 54812.4 | **51124.2** |
| Obj2 | **328147.7** | 455035.3 |
| Paper1 | **61347.6** | 90597.9 |
| Paper2 | **77266.7** | 123020.9 |
| Paper3 | **47404.2** | 68235.3 |
| Paper4 | **16007.7** | 21175.9 |
| Paper5 | **17148.5** | 21207.1 |
| Paper6 | **47245.9** | 68352.3 |
| Pic | 130955.4 | **127174.9** |
| ProgC | **43596.5** | 68824.6 |
| ProgL | **77270.9** | 109812.9 |
| ProgP | **45235.5** | 76238.0 |
| Trans | **84397.4** | 166206.0 |

that are formatted quite differently than the main body. We investigate how well a PSR can perform on such a task, even though some of the modeling assumptions the learning algorithms rely on may be violated. We choose to use the efficiency sharpening procedure, given its good performance on the POMDP corpus. We fix the model dimension to 20, which has been shown to be a good tradeoff between performance and computational resources on text data (Jaeger *et al.*, 2006b). As in the POMDP experiments, we keep the CTW depth fixed to 6.

The results of this experiment are shown in table 6.4. The CTW algorithm usually, but not always, outperforms the ES algorithm. More importantly, when the ES algorithm is the winner, the margin of victory is small. However, it is often the case that when CTW outperforms the ES algorithms, the margin is much larger. On some files, such as BIB and TRANS, the CTW algorithm performs twice as well. These results suggest that the ES algorithm is overly reliant on its modeling assumptions.

## 6.9 Conclusion and Open Problems

The results presented here are not particularly flattering for PSR learning algorithms. In previous chapters, we have presented theory suggesting that PSRs naturally lend themselves to machine learning:

- the statistics needed for learning the models are readily available in training data,
- data structures such as the system matrix have a low rank, a common modeling assumption in machine learning,
- the linear nature of the model prediction and update functions lead to closed-form linear algebra solutions.

In addition, PSRs are an extremely expressive class of model: any VMM or HMM can be converted into a PSR, but not vice versa. Despite these advantages, the PSR learning algorithms we have investigated do not produce particularly good models. This is the case at least when training data is low, or when some of the modeling assumptions are violated.

Variable-order Markov models have shown to be better predictors in these situations. This is the case even when VMMs cannot exactly model the stochastic process to be learned. There are several reason why this may be the case:

- VMM algorithms are more mature. Most of the bad modeling frameworks have been weeded out.
- The VMM directly maximizes the likelihood of the training data, while the PSR indirectly models the training data via the system matrix $\mathbf{D}$.
- The VMM algorithm is more stable and robust. Instead of making future predictions based on (likely faulty) past predictions, the VMM always grounds its predictions in the immediate history of the process.

There is obviously much to be done in order to close this gap. Since PSRs are at least as expressive as VMMs, PSR learning algorithms should be able to fall back to a VMM-style model when it is likely that the PSR parameters may be poorly estimated. This can be accomplished via some sort of regularization in the learning, or by incorporating a Bayesian prior that favor models parameters that are “close to” the model class of VMMs.

It is also necessary to explore the question of how we can train a PSR to maximize training likelihood. An E-M algorithm that guarantees each iteration improves training likelihood would be a huge advancement over current methods.

Unfortunately, these issues must be addressed at a later time.

# Chapter 7

# Multivariate Prediction

So far, we have presented theory and algorithms for modeling stochastic processes. These describe one random variable – the current observation – as it changes over time. In this chapter, we present a framework for using PSRs to model other distributions. In particular, we examine a predictive representation for modeling multivariate random variables, but with no explicit consideration of time. We require that the distribution be over a finite set of $n$ random variables $\mathcal{X} = \{X_1, \ldots X_n\}$, each of which take on no more than $m$ distinct values $\mathcal{O} = \{o_1, \ldots o_m\}$.

We model these distributions as latent state processes. These processes take as input a query, and output the probability of the query being true. The probability of any query is conditionally independent, given information of the latent state. The latent state cannot be directly observed, but only inferred through the results of queries. Latent state processes are expressive enough to encompass many types of distribution, such as the "bag of words" model of documents, and joint probability distributions on a collection of discrete random variables.

We introduce the Linearly Dependent Distribution (LiDD), a new latent state model of discrete multivariate data. LiDDs can be thought of as an extension of Predictive State Representations, which model stochastic processes (Singh *et al.*, 2004). The key modeling assumption for LiDDs is that regardless of prior evidence, the prob-

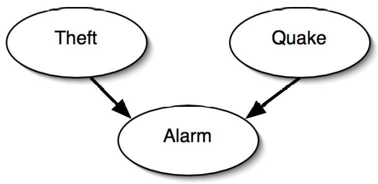


Figure 7.1: A simple graphical model of a burglar alarm and possible causes (Pearl, 1988). This model assumes that whether or not there is a burglary is independent of whether there is an earthquake. The burglar alarm state depends on both of these variables.

abilities of all possible inference queries lie in a low dimensional linear space. We model this subspace directly. This approach obviates the need for indirectly modeling the latent state as a random variable. Unlike previous approaches, a LiDD treats joint unconditioned probabilities on sets of variables as the fundamental unit of the model. This allows for efficient answers to any inference question.

## 7.1 Previous Methods

We briefly discuss graphical models, the most popular approach to modeling joint distributions on discrete variables. Graphical models make conditional independence assumptions between variables (see Murphy (2001) for an introduction). The dependency relations can be arranged in an undirected or directed graph. In a directed graph, the distribution on some random variable $X_i$ is completely determined by the random variables with edges directed at $X_i$. In undirected graphs, the relationship is similar. An edge between two variables indicates that their joint distribution differs from their marginal distributions. See figure 7.1 for an example of a small graphical model.

Although graphical models are a good method for compactly defining joint distributions, using them to infer the marginal distributions over some of the random variables may take time exponential in the size of the model (see *e.g.*, Murphy (2001) for

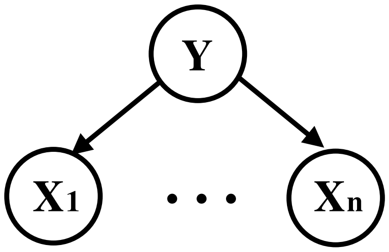


Figure 7.2: A graphical model of the Naïve Bayes Estimator. The random variable $Y$ is latent, and is never observed. The distribution of each $X_1 \ldots X_n$ are independent, given the value of $Y$.

a review of this well known result). Instead, approximate inference techniques, including belief propagation, Monte-Carlo sampling, and variational methods are used to find an approximation of the probability induced by the model. Interestingly, in directed graphical models, it is always efficient to find the probability of a particular complete assignment of values to all random variables. As the number of "free variables" (*i.e.* those variables that can take on any value) increases, the harder the inference problem becomes [1]. This property of graphical models seems a bit perplexing, when we consider graphical models that are learned from training data. Given raw data, the marginal distribution on a few random variables can be estimated by simply finding the frequency of various outcomes in the data. Estimating the probability of a complete assignment of random variables will be much more error prone: as most complete assignments will never be seen at all in any reasonably sized training set.

It is possible to build graphical models designed to make inference efficient (see Meila and Jordan (2000); Bach and Jordan (2001) for recently developed models with this property). Recently, Lowd and Domingos (2005) have shown that a simple graphical model based on the naïve Bayes classifier can achieve results that are comparable to graphical models with much more complex structures. These models, called naive Bayes probability estimators (NBE), are closely related to some "bag of words" mod-

[1]The longer time is due to the fact that graphical models must take the expected value of the observed variable, summed over all the possible configurations of the unobserved variables. The more variables that are unobserved, the larger this sum will be.

els used for document analysis. They have one discrete latent variable $Y$, which takes on values $y_1 \dots y_r$. The primary modeling assumption is that all variables $X_i$ are conditionally independent of each other, given the value of $Y$. A key advantage of NBE is that marginal probability distributions can be calculated quickly without resorting to approximate techniques. NBEs are also highly scalable, and can, in principle, represent any joint distribution on $\mathcal{X}$ (Lowd and Domingos, 2005). See figure 7.2 for a representation of NBE as a graphical model. We will discuss the NBE in detail in section 7.3.

## 7.2 Generalized Stochastic Processes

In order to model joint probabilities on multiple variables and other types of probability distributions, we need to design a flexible protocol. We do this by adapting the stochastic process framework to model other types of data. We call these models generalized stochastic processes (GP). A GP takes as input a *query*, and outputs the probability that this query is true. A query is a composition of elements from a set of primitive queries $u \in \mathcal{U}$.

The semantics of the query depend on the data that is being modeled. For instance, in a controled process, the primitive queries mean "if action $a$ is input, the next observation is $o$". Composing these primitive queries produces a query that refers to several observations into the future.

The bag of words model for document analysis assumes that each document consists of a distribution of words, where the distribution is given by a latent document class (Blei *et al.*, 2003). In this case, a primitive query is "if a random word is drawn from this document, it will be word $o$". A longer query will represent the sampling of words from a single document (from one of the document classes). When the GP is given this query, it will provide the probability that some document will generate this sample of words.

**Multivariate Processes**

We model multivariate prediction as a GP as well. Here the primitive query is "if variable $X_i$ is sampled, the result will be value $o_j$". We call this query $x_i o_j$. Composing multiple primitive queries asks the joint probability that a sequence of random variables, when samples will take on a sequence of values. Our models differ from most multivariate models in one important aspect: if the same random variable is sampled multiple times, it need not have the same value. Instead, the distribution on the random variable is determined by the latent state of the process.

We new formally define a multivariate process:

**Definition 6** *A function $H : (\mathcal{X}\mathcal{O})^* \to [0,1]$ is a multivariate process if:*

- *$H$ is a controlled stochastic process (see definition 3), where $\mathcal{X}$ is treated as the action set, and $\mathcal{O}$ is the observation set,*
- *$H$ satisfies* **commutativity***: $\forall \mathbf{a}, \mathbf{b} \in (\mathcal{X}\mathcal{O})^*,\ H(\mathbf{a} \circ \mathbf{b}) = H(\mathbf{b} \circ \mathbf{a})$.*

Because a multivariate process has all the conditions of a controlled process, we know the output of $H$ can be treated as probabilities. Also, there is a null query, which is always true with probability one. The only new property is commutativity. This reasonable constraint states that the order that the random variables are sampled should not change the resulting probability. As before, we usually use a multivariate process to produce conditional probabilities. Given two queries $\mathbf{a}, \mathbf{b}$, we define the conditional probability under $\mathbf{H}$:

$$H(\mathbf{a}|\mathbf{b}) = \frac{H(\mathbf{b} \circ \mathbf{a})}{H(\mathbf{b})} .$$

We call the information that $H()$ maintains about the query $\mathbf{b}$ in order to make conditional predictions the *latent state* of the process.

As we see, there is a remarkable fit between the domain of the PSR (one observation, changing through time), and the domain of the multivariate process (many observable variables without a concept of time). Later, we exploit this relationship to transfer PSR learning algorithms to this domain.

## 7.3 Naïve Bayes for Probability Estimation

First, we give an example model for multivariate processes. Naïve Bayes is a simple graphical model used for classification, where there is one causal variable $Y$, taking on $r$ values, and several observation variables $X_1 \dots X_n$. The primary modeling assumption is that given the value of the causal variable, the probability distributions on the observations are independent. An *example* is a sample of one or more of the observation variables where $Y$ is fixed to some unknown value. It is assumed that $Y$ is the class label, and the naïve Bayes model is used to infer the most probable class label, given the current example.

Naïve Bayes can also model multivariate data without class labels. In this setting, an example consists of a hidden sample of $Y$, followed by samples from some of the $X_i$. Lowd and Domingos (2005) recently published on this use of the naïve Bayes model, calling it naïve Bayes for probability estimation (NBE).

The NBE model allows us to calculate the probability of any query with a simple equation. Assume we want to know the probability of some query where observable variables $X_1 \dots X_m$ have values $o_1 \dots o_m$. Due to the graphical model structure, we can calculate the joint probability as the product of the probability of each primitive query, given the possible latent state values:

$$H(x_1 o_1 \dots x_m o_m) \;=\; \sum_{i=1}^{r} \Pr(y_i) \prod_{j=1}^{m} \Pr(X_j = o_j | y_i).$$

We simplify this equation using linear algebra:

$$H(x_1 o_1 \dots x_m o_m) = \mathbf{b}_\epsilon^\top \Big( \prod_{j=1}^{m} \mathbf{D}_{x_j o_j} \Big) \mathbf{1},$$

where $\mathbf{b}_\epsilon$ is a vector containing our initial distribution on $Y$, $\mathbf{D}_{xo}$ are $r$ by $r$ diagonal matrices where $\mathbf{D}_{xo}[i,i] = Pr(X = o|y_i)$, and $\mathbf{1}$ is a vector of ones. There is an obvious relationship between this equation and those of equation 2.1 for calculating the probability of a sequence according to a HMM, or equation 3.1, for a PSR. In all cases, the calculation is performed using a series of matrix multiplications, with an initial and final vector multiplication. In fact, the NBE can be interpreted as a class of POMDPs, the controlled version of the HMM.

We define a few other terms. For any query $\mathbf{s}$ composed of a set of $X_i = o_j$ observations, we define a matrix $\mathbf{D_s}$ and vector $\mathbf{d_s}$, where

$$\begin{aligned}\mathbf{D_s} &= \prod_{x_i o_j \in \mathbf{s}} \mathbf{D}_{x_i o_j} \\ \mathbf{d_s} &= \mathbf{D_s 1}.\end{aligned}$$

With these definitions, we can calculate other queries.

A conditional query asks for the probability of query $\mathbf{a}$, possibly conditioned on a query $\mathbf{c}$ having already been observed. To calculate this, we use the following equation:

$$H(\mathbf{a}|\mathbf{c}) = \frac{\mathbf{b}_\epsilon^\top \mathbf{d_{ca}}}{\mathbf{b}_\epsilon^\top \mathbf{d_c}}.$$

If $\mathbf{d_{ca}}$ and $\mathbf{d_c}$ are already computed, this query takes time linear in $r$, the number of values for the causal variable. Calculating $\mathbf{d_a}$ takes time $r \times l$, where $l$ is the number of primitive queries in $\mathbf{a}$. As long as $r$ is small, this run time is extremely efficient compared to standard techniques for performing inference on graphical models. Also, using NBE, it is more efficient to answer shorter queries, while for general graphical models, shorter queries may take more time than longer ones.

If we need the probability of several queries conditioned on the same $\mathbf{s}$, we will want to compute $\mathbf{b}(\mathbf{s})$:

$$\mathbf{b}(\mathbf{s})^\top = \frac{\mathbf{b}_\epsilon^\top \mathbf{D_s}}{\mathbf{b}_\epsilon^\top \mathbf{d_s}}.$$

This gives the probabilities of each possible value of $X$, given that query $\mathbf{s}$ has been observed.

It is also easy to see that NBE is a multivariate process. Because the NBE can be described as a POMDP, the NBE satisfies the first condition of being a controlled process. The NBE is also commutative:

$$\begin{aligned} H(\mathbf{a} \circ \mathbf{c}) &= \mathbf{b}_\epsilon^\top \mathbf{D}_\mathbf{a} \mathbf{D}_\mathbf{c} \mathbf{1} \\ &= \mathbf{b}_\epsilon^\top \mathbf{D}_\mathbf{c} \mathbf{D}_\mathbf{a} \mathbf{1} \\ &= H(\mathbf{c} \circ \mathbf{a}). \end{aligned}$$

This is due to the fact that all the NBE matrices $\mathbf{D}_\mathbf{s}$ are diagonal, and thus commute.

### 7.3.1 Learning NBE Model

The NBE model can be learned using the expectation maximization algorithm (Lowd and Domingos, 2005). It is assumed we are given a training set, consisting of examples $\mathbf{e}_1 \ldots \mathbf{e}_t$ Learning begins with random parameters, and then goes through alternating phases:

1. For each example $\mathbf{e}_i$, estimate the probability of the latent variables given the example:
$$\mathbf{b}(\mathbf{e}_i)^\top = \mathbf{b}(\epsilon)^\top \mathbf{D}_{\mathbf{e}_i}.$$

2. Re-estimate the parameters of the model, given the new beliefs:
$$\begin{aligned} \mathbf{b}(\epsilon)^\top &= \frac{1}{t} \sum_{i=1}^{t} \mathbf{b}(\mathbf{e}_i)^\top, \\ \mathbf{D}_{xo}(j,j)^\top &= \frac{\sum_{i=1}^{t} \mathbf{b}(\mathbf{e}_i)[j] I(xo \in \mathbf{e}_i)}{\sum_{i=1}^{t} \mathbf{b}(\mathbf{e}_i)[j]} \end{aligned}$$

Choosing the number of latent states (the dimension of the $\mathbf{b}$ vectors) is done incrementally based on the performance of the learned NBE model on validation data (see Lowd and Domingos (2005) for details). Code for E-M learning of NBE is available for download at:

http://www.cs.washington.edu/ai/nbe/ .

## 7.4 Linearly Dependent Distributions

We now introduce the Linearly Dependent Distribution (LiDD). These models generalize the Naive Bayes Probability Estimator (NBE), which has been shown to be both efficient and competitive in modeling real-world data (Lowd and Domingos, 2005). LiDDs keep many desirable properties of NBE, such as fast exact inference, but are capable of representing a larger class of distributions. Just as PSRs are a generalization of the HMM and POMDP, we will see that LiDDs are a generalization of the NBE.

According to the NBE model, every example comes from some fixed but unknown value for $Y$. In practice, there will be residual uncertainty about the value for $Y$ after processing evidence in the form of the outcomes of queries. This uncertainty after observing query $\mathbf{s}$ is captured in the vector $\mathbf{b}(\mathbf{s})$. Regardless of the evidence processed, the probability of any query on new samples is completely specified by the current belief vector on $Y$. In fact, the probability of the query $\mathbf{a}$ is a linear function of the belief vector $\mathbf{b}(s)$ and the coefficients for the queries $\mathbf{d}_{\mathbf{a}}$.

This insight provides us with a new interpretation of NBE models. Consider the infinite dimensional space where each dimension is associated with a query. Any multinomial process defines a single point in this space, where the coordinate in dimension $\mathbf{a}$ is the probability this distribution assigns to query $\mathbf{a}$. Examine all processes induced by a NBE model with different values for $\mathbf{b}(\mathbf{s})$. All these distributions lie on subspace of dimension no more than $r-1$, where $r$ is the number of choices for $Y$.

A linearly dependent distribution (LiDD) is a process on queries that can be modeled by such a linear subspace. The current coordinates on this subspace (a function of $\mathbf{b}(s)$) are uniquely defined by the point's value along $r$ linearly independent dimensions. The probability of any of the infinite possible queries is linearly dependent on the probability of these $r$ queries.

### 7.4.1 Multivariate Hankel Matrix

We have assumed there is a linear structure to the probabilities of queries. In order to access this structure, we enumerate every possible query. Assume, w.l.o.g. that in this enumeration, the null query is first, followed by queries on one variable, then two variables, etc.

From this enumeration we construct the *multivariate Hankel matrix* $\mathcal{S}$, where for queries $\mathbf{a}_i$ and $\mathbf{a}_j$, $\mathcal{S}(i, j) = H(\mathbf{a}_i \circ \mathbf{a}_j)$. The multivariate Hankel matrix inherits all the properties of a Hankel matrix for a controlled process, though there is additional structure in the multivariate version. One immediately notices that due to commutativity, $\mathcal{S}$ is symmetric. Commutativity also induces symmetry within a row or column of the $\mathcal{S}$. For any row $i$, and queries $\mathbf{a}_j$ , $\mathbf{a}_k$, we have

$$\begin{aligned}
\mathcal{S}(i, j) &= \mathcal{S}(j, i), \\
\mathcal{S}(i, \mathbf{a}_j \circ \mathbf{a}_k) &= \mathcal{S}(i, \mathbf{a}_k \circ \mathbf{a}_j).
\end{aligned}$$

The modeling assumption behind LiDDs is that although $\mathcal{S}$ is a countably infinite matrix, it has a (small) rank of $r$. A LiDD is characterized by a choice of $r$ linearly independent columns in $\mathcal{S}$. Because $\mathcal{S}$ has rank $r$, we know that the values in every other column of $\mathcal{S}$ must be some linear combination of the values in these $r$ columns. The queries that correspond to these linearly independent columns are the *core queries* of the distribution. Following the notation of the PSR, we vectorize these columns' entries for row $\mathbf{a}_i$ as $\mathbf{q}(\mathbf{a}_i)$.

### 7.4.2 LiDD Parameters

In addition to the choice of core queries, a LiDD also requires the specification of some of the linear coefficients that produce other columns of $\mathcal{S}$. Specifically, we require the coefficients for the column corresponding to the null query, and columns of queries where a core query is conjoined with a primitive query. We refer to the vector of coefficients for the null query as $\mathbf{m}_\epsilon$. Given $\mathbf{q}(\mathbf{a}_i)$ and $\mathbf{m}_\epsilon$, we can calculate $H(\mathbf{a}_i)$ as a simple dot product:

$$H(\mathbf{a}_i) = \mathbf{q}(\mathbf{a}_i)^\top \mathbf{m}_\epsilon.$$

The vector for primitive query $xo$ conjoined with core query $j$ is defined as $\mathbf{m}_{xoj}$. We call these the core extension queries. Using these vectors, and the core queries for row $\mathbf{s}$, we can calculate the probability of a core query on row $\mathbf{s} \circ xo$. For mathematical convenience, we arrange the $k$ core extension queries for primitive query $xo$ into a square matrix $\mathbf{M}_{xo}$. With this, we have

$$\mathbf{q}(\mathbf{a}_i \circ xo)^\top = \mathbf{q}(\mathbf{a}_i)^\top \mathbf{M}_{xo}.$$

By composing a sequence of these matrix operations, we can begin with $\mathbf{q}(\epsilon)$, and derive $H(\mathbf{s})$ for arbitrary event $\mathbf{s} = x_1 o_1 \ldots x_n o_n$:

$$H(\mathbf{s}) = \mathbf{q}(\epsilon)^\top \mathbf{M}_{x_1 o_1} \times \ldots \times \mathbf{M}_{x_n o_n} \mathbf{m}_\epsilon. \tag{7.1}$$

As opposed to general graphical models, calculating the probability of a query using a LiDD will always take time polynomial in the size of the model and the query. Queries that involve only a few variables can be done very quickly. Larger queries can be done through a series of matrix multiplies, or can be done immediately if we have computed $\mathbf{m}_\mathbf{s} = \mathbf{M}_{s_1} \mathbf{M}_{s_2} \ldots \mathbf{m}_\epsilon$ beforehand.

Note that unlike NBE, we put no restrictions on the parameters $\mathbf{q}(\epsilon), \{\mathbf{M}_{xo}\}, \mathbf{m}_\epsilon$ other than they produce a valid multinomial process when equation 7.1 is used. We will discuss checking the constraints for a multinomial process can be checked later.

### 7.4.3 From NBE to LiDD

By examining NBE as a family of distributions confined on a subspace, we are lead to a reparameterization of the model as a LiDD. In order to gain some intuition, we now work through the details of the transformation from a NBE to a LiDD. Choose $r$ queries $\mathbf{q}_1 \ldots \mathbf{q}_r$ such that the vectors $\mathbf{d}_{\mathbf{q}_i} = \mathbf{D}_{\mathbf{q}_i} \mathbf{1}$ are all linearly independent [2]. These

[2] We assume that we can find $r$ such queries. If we cannot, a similar, but more complex construction is possible. See Littman *et al.* (2001) for the full construction from POMDP to PSR.

queries are a set of core queries for this LiDD. Because the NBE is also a stochastic process, by theorem 3.1 we know that if the NBE produces a rank $r$ Hankel matrix $\mathcal{S}$, then we will be able to find a set of $\mathbf{q}_1 \ldots \mathbf{q}_r$ where each query is composed of fewer than $r$ primitive queries.

Arrange these $r$ column vectors into a square matrix $\mathbf{D}$, such that

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_{q_1} \ldots \mathbf{d}_{q_k} \end{bmatrix}.$$

We use $\mathbf{D}$ to redefine the model. The vector $\mathbf{q}(\mathbf{s})^\top = \mathbf{b}(\mathbf{s})^\top \mathbf{D}$ are the probabilities of the core queries, given query $s$ has occurred. The matrices $\mathbf{M}_{xo} = \mathbf{D}^{-1}\mathbf{D}_{xo}\mathbf{D}$ are used for calculating the probabilities for each of the core queries, that this core query and query $xo$ both succeed. The vector $\mathbf{m}_\epsilon = \mathbf{D}^{-1}\mathbf{1}$ calculates the probability of the null query in our new representation.

It is easy to see that these new parameters maintain all the functionality of the original NBE model. The probability of event $\mathbf{s} = s_1 \ldots s_l$ is calculated as

$$\begin{aligned} H(\mathbf{s}) &= \mathbf{q}(\epsilon)^\top \Big( \prod_i \mathbf{M}_{s_i} \Big) \mathbf{m}_\epsilon, \\ &= \mathbf{b}(\mathbf{s})^\top \mathbf{D} \Big( \prod_i \mathbf{D}^{-1} \mathbf{D}_{s_i} \mathbf{D} \Big) \mathbf{D}^{-1} \mathbf{m}_\epsilon, \\ &= \mathbf{b}(\mathbf{s})^\top \Big( \prod_i \mathbf{D}_{s_i} \Big) \mathbf{m}_\epsilon. \end{aligned}$$

In the final calculation, each $\mathbf{D}$ is canceled by being multiplied by its inverse.

It is reasonable to ask why such a parametrization is desirable. The key here is that in the new model, the NBE parameters that define the influence of latent variable $Y$ are replaced with parameters that describe how the probability of the core queries relate to the probability of other queries. The probabilities of various queries can be directly approximated from data, while the probabilities related to a latent variable must be indirectly deduced using techniques such as expectation-maximization.

### 7.4.4 Modeling Power

It has been shown that PSRs and OOMs model a larger family of processes than POMDPs or HMMs (Jaeger, 2000). Specifically, Jaeger demonstrated that there is a stochastic process called the probability clock, which can be modeled with a three-event PSR, but could not be modeled with a HMM with a finite number of latent states. It is reasonable to ask if the same type of argument can be made for LiDD versus NBE.

The first observation is that if each variable were sampled only once, then a NBE could model any joint distribution on these variables. The model keeps a latent variable $Y$ with $r = |\mathcal{X}|^{|\mathcal{O}|}$ latent states, where each state keeps the probability of one particular configuration of all the variables of $\mathcal{X}$. The matrices $\mathbf{D}_{xo}$ only have entries of 0 or 1, where $\mathbf{D}_{xo}[i,i] = 1$ if the configuration corresponding to state $y_i$ has variable $x$ taking on value $o$. Although this NBE model is terribly large, it is capable of modeling any process where variables are sampled once. This means that the only fundamental difference in modeling power between LiDDs and NBE will stem from sampling at least one variable multiple times.

We can modify the probability clock used in Jaeger's proof such that it becomes a multivariate process. We define a multivariate process $H$ with one variable $x$, taking two values $a, b$. Define $\mathbf{q}(\epsilon) = [.75, 0, 0.25]$, $\mathbf{m}_\epsilon = [1, 1, 1]$, and the two operators as follows:

$$\mathbf{M}_{xa} = \frac{1}{2} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix},$$

$$\mathbf{M}_{xb} = \mathbf{I} - \mathbf{M}_{xa}.$$

Here $c$ is the cosine of 1 and $s$ is the sine of 1. The operator $\mathbf{M}_{xa}$ can be thought of as rotating the conditional probability of observing an $a$ around a sine wave. The amount that is rotated is 1 radian, which does not produce a regular, periodic cycle of values. See figure 7.3 for a visualization. Jaeger (2000) has shown that this process cannot be represented by any HMM or POMDP.

Some simple algebra shows that $H$ meets all the requirements of a controlled process. It also satisfies the commutativity constraint. Note that $\mathrm{M}_{xb}$ is a polynomial[3] of $\mathrm{M}_{xa}$ . Matrices that are polynomials of some base matrix will commute (Horn and Johnson, 1986). Thus, because these matrices commute, the multivariate process built from these matrices also commutes.

It appears that LiDDs can model more distributions on multivariate processes, but it is still uncertain if this extra flexibility corresponds to being better able to model any processes that appear in the real world. On the face of it, our example process is rather odd. Basically, this process keeps the count of both $a$'s and $b$'s already observered, and then assigns a probability to the nextobservationn that is a highly nonlinear function of these counts.

### 7.4.5 Enforcing Commutativity

We now briefly discuss the commutativity constraint omultivariatete processes, and how this constraint relates to LiDDs. It is clear that the commutativity constraint holds if the set of matrices $\{\mathrm{M}_{xo}\}$ commute. There has been a good deal of interest in studying the conditions when a group of matrices commute (Horn and Johnson, 1986; Carrette, 1998). To my knowledge, there are no properties that easily characterize necessary and sufficient conditions for a collection of matrices to commute. In other words, it is difficult to find structural constraints on $\mathrm{M}$ that will be easy to enforce by a learning algorithm.

There is one sufficient condition that we have used in our conversion of NBE to LiDD:

**Definition 7** *A collection of $r \times r$ matrices $\mathrm{M}_1 \ldots \mathrm{M}_n$ is* **simultaneously diagonalizable** *if we can find some $k \geq r$, a $k \times r$ matrix $\mathbf{T}$ and a $r \times k$ matrix $\mathbf{T}^\dagger$ such that*

- $\mathbf{T}^\dagger \mathbf{T} = \mathbf{I}$

[3] A polynomial of a matrix is a sum of powers of a matrix, multiplied by a scalar: $\sum_k \alpha_k \mathbf{M}^{\beta_k}$.

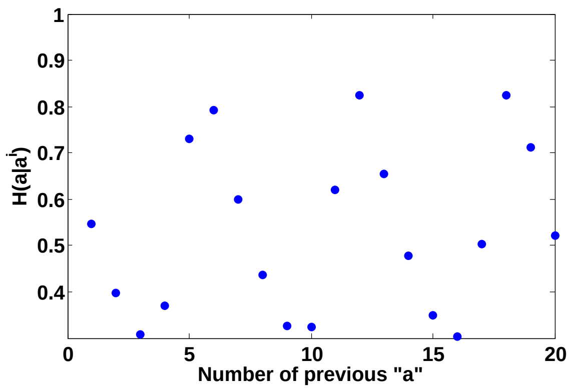


Figure 7.3: The conditional probability that the next query of $x$ will be $a$, given sequences of $a$ of varying length. Note that the point values oscillate, but are not periodic.

- $\forall i, \mathbf{K}_i = \mathbf{T}\mathbf{M}_i\mathbf{T}^\dagger$ *is a diagonal matrix.*

Our definition differs slightly from more conventional ones (see, *e.g.* Horn and Johnson (1986)), in that we allow the diagonalizing matrices $\mathbf{T}$ and $\mathbf{T}^\dagger$ to be non-square. In our conversion from a NBE to a LiDD, we used matrices $\mathbf{D}$ and $\mathbf{D}^{-1}$ to convert a diagonalized collection of NBE matrices to a LiDD representation on core queries. We can reverse the process by assigning $\mathbf{T} = \mathbf{D}^{-1}$ and $\mathbf{T}^\dagger = \mathbf{D}$.

If $\mathbf{T}$ and $\mathbf{T}^\dagger$ are fixed and known, then we can directly learn $\mathbf{K}_i$, and we'll be sure that the resulting LiDD will satisfy commutativity. It is unclear what, if any, flexibility is lost by constraining the LiDD to have this form. Also, there is no obvious guideline for choosing $\mathbf{T}$ and $\mathbf{T}^\dagger$ .

In practice, we do not explicitly enforce commutativity on our learned LiDDs. Even so, the resulting parameters will usually satisfy commutativity up to a relatively small margin of error. Instead, we enforce that all queries are handled in order, where queries on variable $x_i$ are handled before $x_j$ for $i < j$, and queries $(x_i = o_k)$ are handled before $(x_i = o_l)$ for $k < l$.

## 7.5 Learning LiDDs

One of the primary advantages of LiDDs is that all parameters are in terms of the relation between the probability of queries that involve only observable variables. The task of learning a LiDD is to find a basis that describes a LiDD's subspace through query probabilities, and then the parameters necessary to define it. We do this by applying variants of the system matrix-based PSR algorithms to a version of $\mathcal{S}$.

We assume that the training data consists of a collection of examples $\mathbf{E} = \mathbf{e}_1, \ldots \mathbf{e}_t$. Each example is a collection of one or more samples from the variables $x_1, \ldots, x_m$. We assume that the sampling process is not influenced by the outcome of the samples. In fact, we only consider training sets where each variable is sampled exactly once, though the learning algorithms we present can be adapted to other sampling processes.

### 7.5.1 Approximate Hankel Matrix

As for learning a PSR, we must build an approximate Hankel matrix $\mathcal{S}$ that captures the structure present in query probabilities. This involves the following steps:

1. Construct a submatrix $\mathbf{S}$ by choosing particular rows and columns of $\mathcal{S}$.

2. For each entry in $\mathbf{S}$, corresponding to query $\mathbf{g}$, we must approximate $H(\mathbf{g})$ from training data.

First, we need a method for estimating the probability that some query occurs in a single $\mathbf{e}_i$ . Assume $\mathbf{e}_i = x_{i_1}o_{i_1}\ldots x_{i_k}o_{i_k}$. The probability of some query $\mathbf{g} = x_1o_1\ldots x_ho_h$ is

$$\hat{H}(\mathbf{g}|\mathbf{e}_i) = \prod_{j=1}^{h}\hat{P}(x_jo_j|\mathbf{e}_i) = \prod_{j=1}^{h}\frac{\sum_{x_{i_l}=x_j} I(o_{i_l}=o_j)}{\sum_{x_{i_l}=x_j} 1}\ .$$

In words, we calculate the empirical probability of each of the primitive queries being true in our example, and then multiply them together. Note that if each variable $x_i$ is sampled only once, then the probability of a query given an example will always be 0 or 1. Also, if our query asks us to sample one variable multiple times, then the estimated probability will be biased. This is because we are using the one sample in $\mathbf{e}_i$ to estimate multiple samples that are supposed to be independent. This bias is evident when multiple instances of the same primitive query are composed. If the primitive query succeeds, then the estimated probability of the query succeeding an arbitrary number of times will always be 1. This could only be true if the true success probability or the primitive query is 1. This is a serious issue, which needs to be addressed in future research. Note that this problem is present in our estimation procedure for steady-state event and test probabilities, though if the history is long relative to the size of the tests, this bias is not particularly large.

In order to estimate the unconditioned query probability, we take the expectation of the query probability over all examples:

$$\hat{H}(\mathbf{g}) = \sum_{\mathbf{e}_i\in\mathcal{E}}\Pr(\mathbf{e}_i)\hat{H}(\mathbf{g}|\mathbf{e}_i).$$

Assuming that the examples are sampled from the distribution we are interested in modeling, we use the following approximation (recall that our set of examples $\mathbf{E}$ has $t$ examples in it):

$$\hat{H}(\mathbf{g}) = \frac{1}{t}\sum_{\mathbf{e}_i\in\mathbf{E}}\hat{H}(\mathbf{g}|\mathbf{e}_i).$$

Note that our estimates of $\hat{H}$ do not converge to the true values $H()$ as the number of examples increases. This is because the biases present in each of our conditional estimates does not decrease as the number of examples increases. For example, consider these two queries that are estimated from a data set where each variable is sampled once:

$$\begin{aligned}\hat{H}(xo) &= \frac{1}{t}\sum_{\mathbf{e}_i \in \mathbf{E}} I(xo \in \mathbf{e}_i),\\ \hat{H}(xoxo) &= \frac{1}{t}\sum_{\mathbf{e}_i \in \mathbf{E}} I(xo \in \mathbf{e}_i) I(xo \in \mathbf{e}_i)\\ &= \hat{H}(xo).\end{aligned}$$

Again, this estimate can only be correct if $\hat{H}(xo|xo) = 1$. We partially address this issue in our algorithms.

**The Training Matrix**

We can arrange the estimated query probabilities $\hat{H}()$ into an approximate Hankel matrix $\hat{\mathbf{S}}$. Note that we can do this efficiently using linear algebra. We define the *training matrix* $\mathbf{T}$, where the rows index examples in our training set, and the columns index the queries in our Hankel matrix. The entry $\mathbf{T}(i, j)$ equals 1 if query $\mathbf{g}_j$ is present in example $\mathbf{e}_i$. We can produce a symmetric sub-sample of the Hankel matrix where rows and columns are indexed by the columns of $\mathbf{T}$ using a simple matrix multiply:

$$\hat{\mathbf{S}} = \frac{1}{t}\mathbf{T}^\top \mathbf{T}.$$

We can work with this matrix $\hat{\mathbf{S}}$ as if it were a Hankel matrix derived from a stochastic process, and then learn a PSR based on this matrix.

Alternately, we could learn using the matrix $\mathbf{T}$ directly. Note that the rank of $\mathbf{S}$ will be the same as for $\mathbf{T}$, and the right eigenvectors for $\mathbf{T}$ will be the same as those of $\mathbf{S}$. The main difference is that the rows of $\mathbf{S}$ average information across examples, while in $\mathbf{T}$, the probabilities for each example are approximated separately.

In this setting, we assume that all examples have some fixed but unknown value for the core queries $\mathbf{q}(\mathbf{e}_i)$. The values in $\mathbf{T}$ represent one sample from the distribution $\mathbf{q}(\mathbf{e}_i)^\top \mathbf{m}_j$ . Note that when the query $\mathbf{g}_j$ does not sample a variable more than once, this is a draw from the actual distribution, but when query calls for a variable to be sampled multiple times, the sample in $\mathbf{T}$ is biased.

### 7.5.2 Basic LiDD Learning Framework

The algorithm we use for learning a LiDD is based on the TPSR algorithm (Rosencrantz *et al.*, 2003). We find some training matrix $\mathbf{T}$, and a characterizer $V$ such that the core event probabilities for each row are calculated as

$$\mathbf{Q} = \mathbf{TV}.$$

We use two types of characterizer. The simpler method chooses $\mathbf{V}$ as the set of columns in $\mathbf{T}$ least well approximated by the other columns selected in $\mathbf{V}$. We also use the SVD to find a low rank basis for $\mathbf{T}$, and use this basis as our characterizer.

From this basis, we can estimate the LiDD parameters. We create a diagonal matrix $\mathbf{D}_{xo}$ where the $(i,i)$th entry is 1 if $xo$ was observed in $\mathbf{e}_i$. Using this matrix, we can estimate the update operators $\mathbf{M}_{xo}$:

$$\begin{aligned} \mathbf{QM}_{xo} &\approx \mathbf{T}_{xo} \\ &= \mathbf{D}_{xo}\mathbf{Q} \end{aligned}$$

We solve this approximation using the pseudo-inverse :

$$\mathbf{M}_{xo} = \mathbf{Q}^{\dagger}\mathbf{D}_{xo}\mathbf{Q}.$$

The remaining parameters are calculated as follows:

$$\begin{aligned} \mathbf{m}_{\epsilon} &= \mathbf{Q}^{\dagger}\mathbf{1}, \\ \mathbf{q}(\epsilon)^{\top} = &= \frac{1}{t}\sum_{i=1}^{t}\mathbf{Q}[i,:]. \end{aligned}$$

## 7.6 Experiments

We perform two set of experiments on learning LiDDs. The first experiment examines the representation that can be learned using LiDDs, and the second evaluates the quality of the LiDD as a generative model of handwritten digits. The results here are exploratory and qualitative in nature. Further work has shown that on more quantitative evaluations of model performance, the NBE consistently outperforms LiDDs. This is undoubtedly related to the fact that NBE is trained to directly maximize training data likelihood, and that the NBE is a simpler model, less prone to overfitting.

### 7.6.1 MNIST Handwritten Digit Corpus

In this experiment, we learn LiDD models using MNIST digit dataset. We replicate some of the experiments done in the work of Bach and Jordan (2001). The original training dataset consists of 60,000 images of size 28 X 28. We crop the white space around each image and then resize them to 16 X 16 pixels. We then discretize the images such that each pixel value is thresholded to 0 or 1.

#### LiDD Representation

In the first experiment, we investigate the representation that is learned by a LiDD. We use a core query discovery algorithm that incrementally learns core queries with procedure similar to the suffix history discovery algorithm presented in algorithm 2:

**Greedy LiDD Discovery ($\delta$)**

1. Build $\mathbf{T}$ as described above, with the null query as the only column.

2. Calculate $\mathbf{q}(i)$ for every row of $\mathbf{T}$

3. For each primitive query $xo$, calculate $\mathbf{T}_{xo}$, and approximate $\mathrm{M}_{xo}$ using least squares regression using the current core queries as a basis.

4. From all $\mathbf{T}_{xo}$, choose the column $j$ with highest squared error $(\mathbf{T}_{xo} - \mathbf{T}\mathbf{M}_{xo})^2$. Add this column to $\mathbf{T}$ as the new core event.

5. Repeat steps 2 through 4 until no column of $\mathbf{T}_{xo}$ has error greater than $\delta$.

We iterate this procedure 150 times on the handwritten "two" corpus. In order to understand what information the core queries are capturing, we calculate images where the probability of each pixel $p$ being on is determined, assuming that some core query $\mathbf{q}_i$ has occured:

$$\hat{H}(p1|\mathbf{q}_i) = \frac{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{q}_i} \mathbf{M}_{p1} \mathbf{m}_\epsilon}{\mathbf{q}(\epsilon)^\top \mathbf{M}_{\mathbf{q}_i} \mathbf{m}_\epsilon} .$$

The results of this can be seen in figure 7.4. Note that most core tests predict a small area of the image will contain dark pixels, while the pixel values in the rest of the image are less certain. Despite this localized certainty, each core query seems specialized for a particular type of two. Note that some core queries assign higher probabilities to curly twos, while others specialize in straight-bottomed twos.

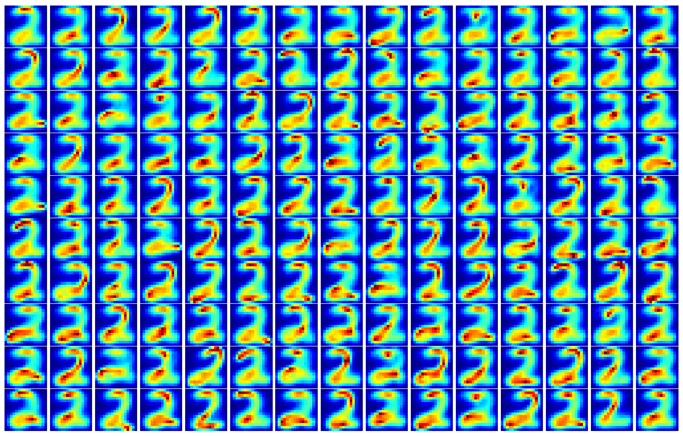


Figure 7.4: The conditional probabilities of each pixel, given one of 150 core queries. Red represents a high probability, and blue represents a low probability. Note this diversity of representation of the different types of twos.

## Digit Reconstruction

We learn one LiDD for each of the ten digits. To train our model for the 256 dimensional data, we use a heuristic to decide which possible queries should be used as the core queries. We want core events that show high variability and are relatively uncorrelated. Our heuristic starts by choosing primitive queries as core queries. We choose the primitive query with the highest variance on the training set, and then choose primitive queries that are least correlated with previous core queries, but still have a high variance. We iterate this procedure until 80 primitive queries are picked. Then we look at all the events that are formed by conjunctions of the 80 chosen events, generating approximately, 6400 events. We then choose 60 two-variable queries by using the same method as that of primitive queries. We get total of 141 core queries including the $\epsilon$ event. We then run PCA on these events and keep the top 80 principal components as our basis.

We use our models to regenerate a digit when less than half of the original image is given. We give the model a query containing the value of some of the pixels $\mathbf{x}$. To reconstruct the image, we calculate for each pixel $p$,

$$\hat{H}(p1|\mathbf{x}) = \frac{\mathbf{q}(\epsilon)^\top \mathbf{M_x M}_{p1}\mathbf{m}_\epsilon}{\mathbf{q}(\epsilon)^\top \mathbf{M_x m}_\epsilon}\ .$$

If the conditional probability of a pixel is greater than $\frac{1}{2}$, then we mark the pixel as ”on”. Note that we calculate this value even on the pixels that were given in $\mathbf{X}$.

Example results of the experiments are tallied in figures 7.5 and 7.6. The sample image given to the model was taken from a separate test set that consisted of 10,000 images. The pixels from the image were deleted in two ways. The first method, erased randomly 80% of both “on” and “off” pixels in the image. The second method had the right 10 columns erased from the image. The correct model was then used to regenerate the image, that being the model learned for the digit that test sample belonged to. We also used a model that was learned on a similar digit but not the test sample digit. We determine that a pixel is “on” if it has a greater than 50% chance of being on, given the pixels that were not deleted. Notice that there is a great deal of variation in the "2" reconstructions when the model is given different initial pixel values.

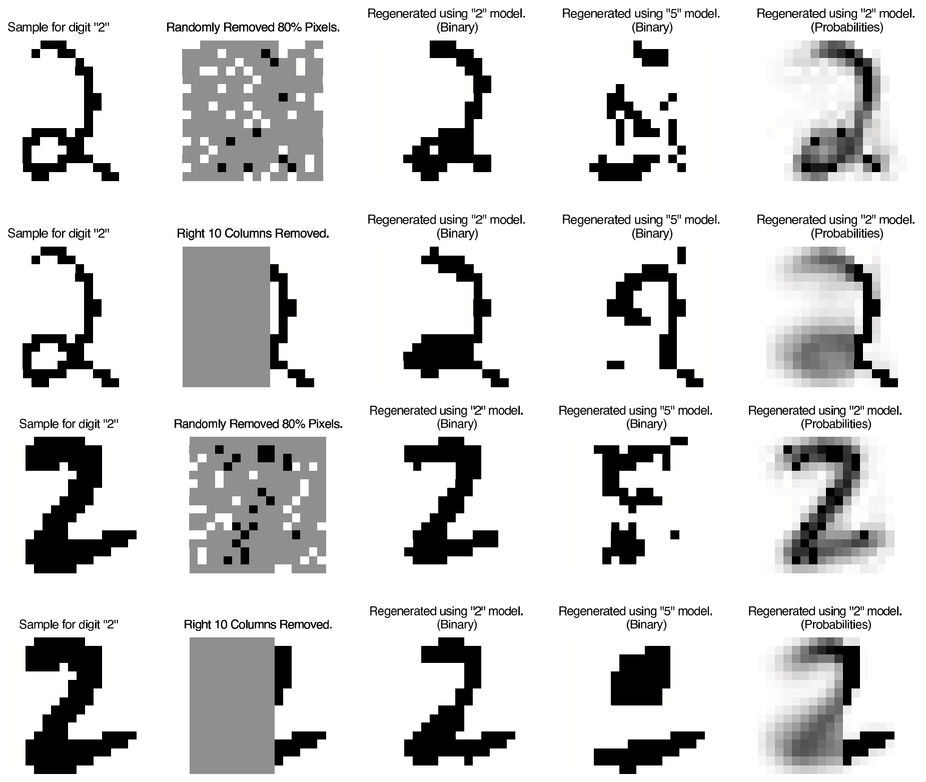

Sample for digit "2"
Randomly Removed 80% Pixels.
Regenerated using "2" model. (Binary)
Regenerated using "5" model. (Binary)
Regenerated using "2" model. (Probabilities)
Sample for digit "2"
Right 10 Columns Removed.
Regenerated using "2" model. (Binary)
Regenerated using "5" model. (Binary)
Regenerated using "2" model. (Probabilities)
Sample for digit "2"
Randomly Removed 80% Pixels.
Regenerated using "2" model. (Binary)
Regenerated using "5" model. (Binary)
Regenerated using "2" model. (Probabilities)
Sample for digit "2"
Right 10 Columns Removed.
Regenerated using "2" model. (Binary)
Regenerated using "5" model. (Binary)
Regenerated using "2" model. (Probabilities)

Figure 7.5: Handwritten twos reconstructed from a small amount of data

## 7.7 Discussion and Future Work

In this chapter, we presented a new method for modeling probability distributions on multiple variables. The Linearly Dependent Distribution (LiDD) is a model capable of modeling a huge class of distributions, while remaining tractable to learn and predict with. We have shown that the LiDD is a special case of a PSR, where we enforce additional properties that the model must obey.

Learning LiDDs is possible with an adaptation of methods used for learning PSRs. We have shown that the learning algorithm seems to learn reasonable models,

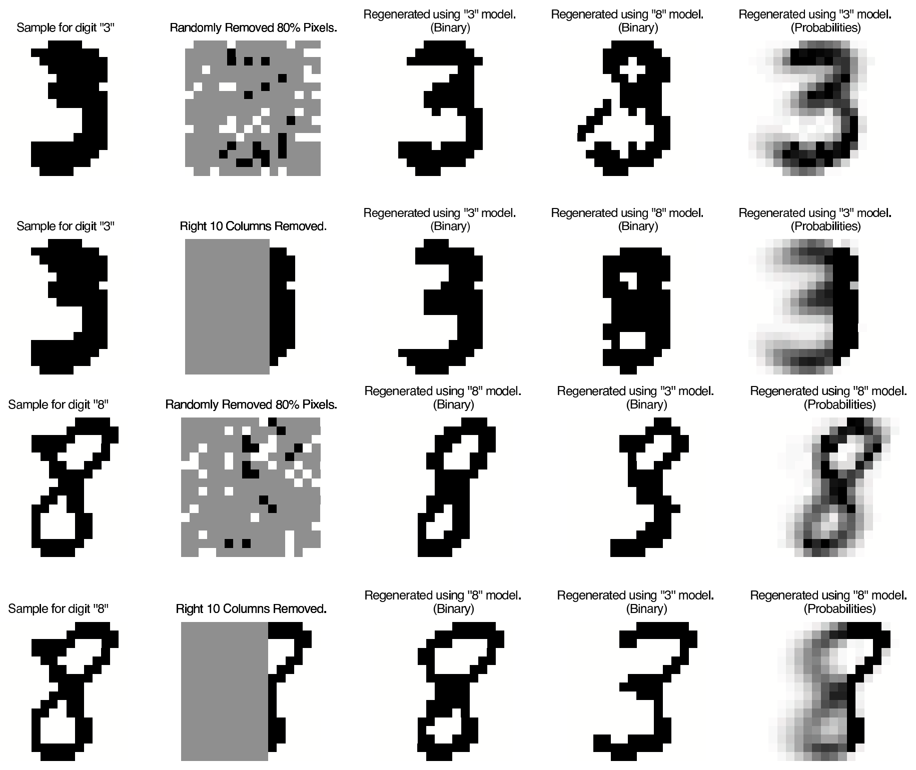

Sample for digit "3"
Randomly Removed 80% Pixels.
Regenerated using "3" model. (Binary)
Regenerated using "8" model. (Binary)
Regenerated using "3" model. (Probabilities)
Sample for digit "3"
Right 10 Columns Removed.
Regenerated using "3" model. (Binary)
Regenerated using "8" model. (Binary)
Regenerated using "3" model. (Probabilities)
Sample for digit "8"
Randomly Removed 80% Pixels.
Regenerated using "8" model. (Binary)
Regenerated using "3" model. (Binary)
Regenerated using "8" model. (Probabilities)
Sample for digit "8"
Right 10 Columns Removed.
Regenerated using "8" model. (Binary)
Regenerated using "3" model. (Binary)
Regenerated using "8" model. (Probabilities)

Figure 7.6: Handwritten threes and eights reconstructed from a small amount of data

though these results are quite preliminary. Further experiments have shown that in general, learned model performance is worse than the Naive Bayes Estimator (NBE), learned through E-M. It is important to address the issues that lead to worse performance. It is likely that the problems with learning LiDDs closely track the problems with learning PSRs. Due to the simpler structure of the LiDD, this is a good starting point for producing more robust and accurate learning methods for all PSRs.

# Chapter 8

# Conclusion

In this dissertation, I have developed the theoretical framework for analyzing Predictive State Representations. In chapter 3, I present necesarry and sufficient conditions for a PSR to produce valid probability estimates. I also show that checking one of these conditions – nonegative predicitons – is undecidable.

In chapter 4, I develop the theory of PSR for modeling controlled processes. I show that we can extend the definition of test to model both the process generating observations, and the policy generating actions. I also show that we can model a controlled process interacting with a policy as a single uncontrolled process.

In Chapter 5, I discuss the theory behind the system matrix, a data structure that is used to learn PSRs. I show that we do not have to search the entire system matrix in order to discover its low rank structure. I also show that if the process to be modeled can be described as a PSR, then we can build a system matrix from the process' avarage event probabilities.

In chapter 6, I compare the prediction performance of learned PSRs to other popular generative models. Often, the comparison is not flattering for the PSR algorithms.

Finally, in chapter 7 I present a framework for applying PSR algorithms to other prediction problems. The Linearly Dependent Distribution (LiDD) is the result of this exploration. Although the results on this model are quite preliminary, this framework offers an interesting interpretation of multivariate prediction.

## 8.1 Future Directions

The framework and analysis presented in this work suggest many directions for future research.

An important contribution of this work is the introduction of the a-test for representing policies. This formalism has the promise of unifying the analysis of many classes of policies. For example, the PSR framework may be provide the proper analytic tools for justifying the benefit of temporal abstraction through options (Sutton *et al.*, 1999). Also, it is possible to use this framework to directly learn a PSR-based policy through policy gradient or actor-critic techniques (Sutton and Barto, 1998).

The results on PSR learning performance show that much more work needs to be put into learning PSRs with small amounts of data. There is no inherent reason why VMM predictors should be more accurate than PSRs, given that PSRs can represent any process that a VMM can. The performance discrepancy may be attributed to many causes:

- **Regularization**: VMMs model a smaller class of distributions than PSRs. This allows for a faster convergence on the one model that best predicts the data. In addition to this, VMM learning algorithms employ advanced probability smoothing techniques that provide robustness against poorly estimated distributions. In order to transfer these advantages to PSRs, two things must be done. One is to regularize the estimates in the system or Hankel matrix using probability smoothing techniques. This has been done, in a fashion, in the work of McCracken and Bowling (2006).

  Second, we should regularize the gross structure of the transition operators $\mathbf{M}_u$ learned by the PSR. Lacking evidence in the training data to suggest otherwise, a PSR learning algorithm should prefer operators that are ”close to” those produced by converting a VMM into a PSR. In order to do this, we need to better characterize the set of PSRs that represent the same distribution as a VMM.

- **Temporal Robustness**: One of the foundational principles of PSRs is that we can use predictions about some possible future observations to predict other observations. In theory, such a representation is incredibly powerful, but in practice, learned PSRs are often unstable. This is because errors in predictions accumulate over time as one set of predictions leads to the next. VMMs, in contrast, base their predictions on information outside of the model: the recent history. Thus, any errors in previous predictions do not propagate into future predictions. While the auto-regressive nature of PSRs is essential for their modeling power, it may be possible to ”reset” the predictions under some circumstances. This issue has been explored by James (2005), though the current results are still preliminary.

- **Learning Objective**: In my opinion, the largest problem with the current state of PSR learning is the learning objective. The algorithms I've described in this dissertation attempt to minimize the error of the tests in the system or Hankel matrix. This objective only loosely corresponds to minimizing prediction error on the training string. VMM and HMM learning is much more oriented toward reducing prediction error on the training string. A learning algorithm such as E-M, which is proven to reduce training error on every iteration would be a huge advance. There has been some work on this topic for a restricted class of PSRs, but nothing for unconstrained models (Jaeger *et al.*, 2006a). I have worked on this problem, and have found it to be more difficult than I thought.

# Appendix A

# List of Notation

| | |
|---|---|
| $\circ$ | Sequence concatenation operator |
| $\mathcal{A}$ | set of actions |
| $a_t$ | action at time $t$ |
| $\mathcal{D}$ | system dynamics matrix |
| $\mathbf{D}$ | a finite submatrix of $\mathcal{D}$ |
| $\mathbf{h}_t$ | observation history at time $t$ |
| $\mathbf{M}_o$ | PSR update operator for observation $o$ |
| $\mathbf{m}_o$ | PSR projection vector for observation $o$ |
| $\mathcal{O}$ | set of observations |
| $O_t$ | random variable for the observation at time $t$ |
| $o_t$ | observation at time $t$ |
| $q_i$ | a core test, event or query |
| $\mathbf{q}(\mathbf{h}_t)$ | probability of core tests at time $t$ |
| $\mathbf{s}_t$ | suffix at time $t$ |

# References

Bach, F. R., and Jordan, M. I., 2001: Thin junction trees. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

Begleiter, R., El-Yaniv, R., and Yona, G., 2004: On prediction using variable order Markov models. *Journal of Artificial Intelligence Research (JAIR)*, **22**, 385–421.

Beimel, A., Bergadano, F., Bshouty, N. H., Kushilevitz, E., and Varricchio, S., 2000: Learning functions represented as multiplicity automata. *Journal of the ACM*, **47**(3), 506–530.

Bell, T. C., Cleary, J. G., and Witten, I. H., 1990: *Text Compression*. Prentice-Hall, Englewood Cliffs, NJ.

Bilmes, J., 1997: A gentle tutorial on the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. Technical Report ICSI-TR-97-021, International Computer Science Institute, Berkeley CA.

Blei, D., Ng, A., and Jordan, M., 2003: Latent Dirichlet allocation. *Journal of Machine Learning Research (JMLR)*, **3**, 993–1022.

Bowling, M., McCracken, P., James, M., Neufeld, J., and Wilkinson, D., 2006: Learning predictive state representations using non-blind policies. In *Proceedings of the Twenty-Third International Conference on Machine learning (ICML)*, 129–136. ACM Press.

Brafman, R. I., and Tennenholtz, M., 2002: R-MAX - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, **3**, 213–231.

Brown, P. F., Pietra, V. J. D., deSouza, P. V., Lai, J. C., and Mercer, R. L., 1992: Class-based n-gram models of natural language. *Computational Linguistics*, **18**(4), 467–479.

Carrette, P., 1998: Reliable method for the evaluation of commuting matrices. Technical report, Linkoping University.

Cassandra, A. R., Kaelbling, L. P., and Littman, M. L., 1994: Acting optimally in partially observable stochastic domains. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI)*. AAAI Press/MIT Press.

Dailey, M. N., Cottrell, G. W., Padgett, C., and Adolphs, R., 2002: EMPATH: A neural network that categorizes facial expressions. *Journal of Cognitive Neuroscience*, **14**(8), 1158–1173.

Denis, F., and Esposito, Y., 2004: Learning classes of probabilistic automata. In *Seventeenth Annual Conference on Learning Theory (COLT)*.

Denis, F., and Esposito, Y., 2006: Rational stochastic languages. arXiv cs.LG/0602093.

Denis, F., Esposito, Y., and Habrard, A., 2006: Learning rational stochastic languages. In *Nineteenth Annual Conference on Learning Theory (COLT)*.

Duff, M., 2003: Design for an optimal probe. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G., 1999: *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge.

Even-Dar, E., Kakade, S., and Mansour, Y., 2005: Planning in POMDPs using multiplicity automata. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Fazel, M., Hindi, H., and Boyd, S., 2004: Rank minimization and applications in system theory. In *Proceedings of the American Control Conference*.

Freund, Y., Kearns, M., Ron, D., Rubinfeld, R., Schapire, R. E., and Sellie, L., 1993: Efficient learning of typical finite automata from random walks. In *Proceedings of the twentyfourth Annual ACM Symposium on Theory of Computing*.

Gavalda, R., Keller, P. W., Pineau, J., and Precup, D., 2006: PAC-learning of Markov models with hidden state. In *Proceedings of the 11th European Conference on Machine Learning (ECML)*. Springer-Verlag Lecture Notes in Artificial Intelligence.

Grimmett, G., and Stirzaker, D., 1982: *Probability and Random Processes*. Claredon Press, Oxford.

Hansen, E., 1997: An improved policy iteration algorithm for partially observable MDPs. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

Hastie, T., Tibshirani, R., and Friedman, J. H., 2001: *The Elements of Statistical Learning*. Springer-Verlag, New York.

Horn, R. A., and Johnson, C. R., 1986: *Matrix analysis*. Cambridge University Press, Cambridge.

Ito, H., Amari, S., and Kobayashi, K., 1992: Identifiability of hidden markov information sources and their minimum degrees of freedom. *IEEE Transactions on Information Theory*, **38**(2).

Jaeger, H., 1998: Discrete-time, discrete-valued observable operator models: a tutorial (updated through 2004). Technical Report 42, GMD - German National Research Center for Information Technology.

Jaeger, H., 2000: Observable operator models for discrete stochastic time series. *Neural Computation*, **12**(6), 1371–1398.

Jaeger, H., Zhao, M., and Kolling, A., 2006a: Efficient estimation of OOMs. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

Jaeger, H., Zhao, M., Kretzschmar, K., Oberstein, T., Popovici, D., and Kolling, A., 2006b: Learning observable operator models via the ES algorithm. In *New Directions in Statistical Signal Processing: from Systems to Brain*, editors S. Haykin, J. Principe, T. Sejnowski, and J. McWhirter, 417–464. MIT Press.

James, M. R., 2005: *Using Predictions for Planning and Modeling in Stochastic Environments*. Ph.D. thesis, University of Michigan.

James, M. R., and Singh, S., 2004: Learning and discovery of predictive state representations in dynamical systems with reset. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML)*.

Kearns, M., and Singh, S., 1998: Near-optimal reinforcement learning in polynomial time. In *Proceedings of the fifteenth International Conference on Machine Learning (ICML)*. Morgan Kaufmann, San Francisco, CA.

Kearns, M. J., Mansour, Y., and Ng, A. Y., 1999: A sparse sampling algorithm for near-optimal planning in large markov decision processes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Kretzschmar, K., 2003: Learning symbol sequences with observable operator models. Technical report, GMD - German National Research Center for Information Technology.

Landauer, T., Foltz, P., and Laham, D., 1998: Introduction to latent semantic analysis. *Discourse Processes*, **25**, 259–284.

Littman, M., Sutton, R., and Singh, S., 2001: Predictive representations of state. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

Lowd, D., and Domingos, P., 2005: Naive Bayes models for probability estimation. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*.

McCallum, A., 1995: Instance-based utile distinctions for reinforcement learning with hidden state. In *Proceedings of the Twelth International Conference on Machine Learning (ICML)*.

McCracken, P., and Bowling, M., 2006: Online discovery and learning of predictive state representations. In *Advances in Neural Information Processing Systems (NIPS)*. MIT Press.

Meila, M., and Jordan, M. I., 2000: Learning with mixtures of trees. *Journal of Machine Learning Research (JMLR)*, **1**, 1–48.

Murphy, K. P., 2001: An introduction to graphical models. Technical report.

Pearl, J., 1988: *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Puterman, M., 1994: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, New York.

Rabiner, L., 1989: A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, **77**.

Rissanen, J., and Langdon, G., 1981: Universal modeling and coding. *IEEE Transactions on Information Theory*, **27**(1), 12–23.

Rivest, R. L., and Schapire, R. E., 1994: Diversity-based inference of finite automata. *Journal of the ACM*, **41**(3), 555–589.

Rosencrantz, M., Gordon, G., and Thrun, S., 2003: Learning low dimensional predictive representations. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML)*.

Rudary, M., and Singh, S., 2006: Predictive linear-gaussian models of controlled stochastic dynamical systems. In *Proceedings of the twenty-third international conference on Machine learning (ICML)*. ACM Press.

Shatkay, H., and Kaelbling, L. P., 1997: Learning topological maps with weak local odometric information. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*.

Singh, S., James, M., and Rudary, M., 2004: Predictive state representations: A new theory for modelling dynamical systems. In *Conference on Uncertainty in Artificial Intelligence (UAI)*.

Singh, S., Littman, M., Jong, N., Pardoe, D., and Stone, P., 2003: Learning predictive state representations. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML)*.

Sutton, R., Precup, D., and Singh, S., 1999: Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, **112**(1-2), 181–211.

Sutton, R. S., and Barto, A. G., 1998: *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA.

Tanner, B., and Sutton, R. S., 2005: TD($\lambda$) networks: temporal-difference networks with eligibility traces. In *Proceedings of the Twenty-Second International Conference on Machine learning (ICML)*. ACM.

Vidal, E., Thollard, F., de la Higuera, C., Casacuberta, F., and Carrasco, R., 2005: Probabilistic finite-state machines-part II. *PAMI*, **27**(7), 1026–1039.

Vovk, V., 2001: Probability theory for the brier game. *Theoretical Computer Science*, **261**(1), 57–79.

Wiewiora, E., 2005: Learning predictive representations from a history. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*. ACM Press.

Willems, F. M. J., Shtarkov, Y. M., and Tjalkens, T. J., 1995: The context-tree weighting method: basic properties. *IEEE Transactions on Information Theory*, **41**(3), 653–664.

Wolfe, B., James, M. R., and Singh, S., 2005: Learning predictive state representations in dynamical systems without reset. In *Proceedings of the Twenty-second International Conference on Machine learning (ICML)*. ACM.

Xue, H., and Govindaraju, V., 2002: A stochastic model combining discrete symbols and continuous attributes and its application to handwriting recognition. In *Proceedings of the Fifth International Workshop on Document Analysis Systems*.