# IMPROVED EMPIRICAL METHODS IN REINFORCEMENT-LEARNING EVALUATION

**BY VUKOSI N. MARIVATE**

**A dissertation submitted to the**

**Graduate School—New Brunswick**

**Rutgers, The State University of New Jersey**

**in partial fulfillment of the requirements**

**for the degree of**

**Doctor of Philosophy**

**Graduate Program in Computer Science**

**Written under the direction of**

**Michael L. Littman**

**and approved by**

_____

_____

_____

_____

**New Brunswick, New Jersey**

**January, 2015**

## ABSTRACT OF THE DISSERTATION

# IMPROVED EMPIRICAL METHODS IN REINFORCEMENT-LEARNING EVALUATION

**by VUKOSI N. MARIVATE**

**Dissertation Director: Michael L. Littman**

The central question addressed in this research is "can we define evaluation methodologies that encourage reinforcement-learning (RL) algorithms to work effectively with real-life data?" First, we address the problem of overfitting. RL algorithms are often tweaked and tuned to specific environments when applied, calling into question whether learning algorithms that work for one environment will work for others. We propose a methodology to evaluate algorithms on distributions of environments, as opposed to a single environment. We also develop a formal framework for characterizing the "capacity" of a space of parameterized RL algorithms and bound the generalization error of a set of algorithms on a distribution of RL environments given a sample of environments. Second, we develop a method for evaluating RL algorithms offline using a static collection of data. Our motivation is that real-life applications of

RL often have properties that make online evaluation expensive (such as driving a robot car), ethically questionable (such as treating a disease), or simply impractical (such as challenging a human chess master). We compared several offline evaluation metrics and found our new metric (relative Bellman update error") addresses shortcomings in more standard approaches. Third, we examine the problem of evaluating behavior policies for individuals using observational data. Our focus is on quantifying the uncertainty that arises from multiple sources: population mismatch, data sparsity, and intrinsic stochasticity. We have applied our method to a collection of HIV treatment and non-profit fund-raising appeals data.

# Preface

Portions of this dissertation are based on work previously published or submitted for publication by the author (Marivate et al., 2014; Marivate and Littman, 2013).

# Acknowledgements

*Ku Khensa* - To Thank. The path I have taken over the last few years has so many wonderful people who have made it possible.

Without the support and encouragement of my wife, Thembekile, I have no possible idea of how I would have been able to survive the long road: From moving to a new land, settling in and trying to thrive, it was all possible because of her. Her support has been unmatched and I am forever thankful for her patience in letting me be a graduate student for these past few years.

I would like to thank my parents and siblings for their wonderful support throughout all these years. I would also like to acknowledge the support of my wider family including my in-laws. At Rutgers and back home in South Africa, I would like to acknowledge close friends who have also encouraged my journey.

On this scientific journey, there is so much I would like to thank my advisor Prof. Michael L. Littman for. His support and guidance in my pursuit will forever leave a lasting mark on me. Through the hard times and the good times he has instilled in me a sense of critical inquisitiveness and desire to do better.

I wish to acknowledge my dissertation committee Tina Eliassi-Rad, Amelie Marian and Susan Murphy (University of Michigan). Their feedback during my writing process made this dissertation better.

I would like to acknowledge my lab mates in the RL3 lab at Rutgers. The interactions we had were always an opportunity for me to learn from you as well

establish a better understanding of the broader field of Reinforcement Learning. Thank you and I wish you all the best in your endeavors as part of the extended RL3 alumni all over the world.

The work presented in this dissertation would not be possible without a wonderful set of collaborators. I would like to thank Emma Brunskill (CMU) and Jessica Chemali (CMU) for working with me and my advisor over the years. Our weekly meetings truly became a rallying point for the latter part of my PhD journey. I extend this same gratitude to James Macglashan (Brown University), Carl Trimbach (Brown University), Matthew Taylor (Washington State) and Eli Upfal (Brown University).

To work with real-world data needs the interaction between the scientist and domain experts. As such, I am very thankful for Benjamin Muthambi (Pennsylvania Department of Health), Tonya Crook (Penn State Hershey), Rami Kantor (Brown University) and Joseph Hogan (Brown University) for their insightful interactions and patience with me as I navigated modeling medical data.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

# Dedication

I wish to dedicate this dissertation to my parents and grandparents. For their support I will be eternally grateful.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1
# Introduction

> Measurement is the first step that leads to control and eventually to improvement. If you can't measure something, you can't understand it. If you can't understand it, you can't control it. If you can't control it, you can't improve it. *H. James Harrington*

## 1.1 Evaluation in reinforcement learning

In machine learning (ML), there are numerous evaluation approaches and metrics used for different types of problems or to provide insight into the behavior of learning algorithms. In supervised learning, the goal of evaluation may be to characterize certain properties of an algorithm, for example, that could be: how fast the algorithm learns, its computational complexity, the generalization power over multiple application domains etc. Another goal might be to characterize the outcomes of a learning algorithm. For example, an outcome could be the performance of the classifier or regression function produced by a supervised-learning algorithm. There are many metrics associated with evaluating this second goal. In classification, just to name a few, the metrics used could be: Accuracy, Precision, Recall, Area Under Curve and Receiver Operating Characteristics (Caruana and Niculescu-Mizil, 2004).

Reinforcement learning (Sutton, 1988) has similar challenges as machine learning when it comes to evaluation. This dissertation argues that there is still

more to do in terms of investigating and expanding evaluation methods that can better give insight into the power of reinforcement-learning algorithms. First, I will create experiments and metrics that better characterize the *capacity of reinforcement-learning algorithms to learn*. That is, the experiments and metrics should characterize how well reinforcement-learning algorithms can learn over different types of problems. Second, I would like to better predict the performance of the outcomes of such algorithms, that is, the performance of the policies produced by reinforcement-learning algorithms, even when we do not have access to the original environments but only pre-collected (batch) data.

This dissertation tackles these two themes in three sections. First, I look at how we can create evaluation approaches that can better characterize the learning capability of algorithms when we have direct access to environments. Secondly, I look at evaluating algorithms where we do not have direct access to environments but have indirect access through batch data for evaluation (maybe even learning). Finally, I look at generally evaluating policies arising from reinforcement-learning algorithms and quantifying the uncertainty in predicting the outcomes. I develop policy evaluation models that take into account uncertainty arising from using batch data for learning in sequential decision making.

The rest of this section first presents some basic background to reinforcement learning. Then I discuss the need for improved methods of online evaluation of reinforcement-learning algorithms. In the latter part of the section, I discuss the motivation for the use of reinforcement-learning algorithms on real-world data (often batch data) and the need to create better evaluation algorithms to support the development of algorithms for this setting. Putting it all together, I present my thesis statement.

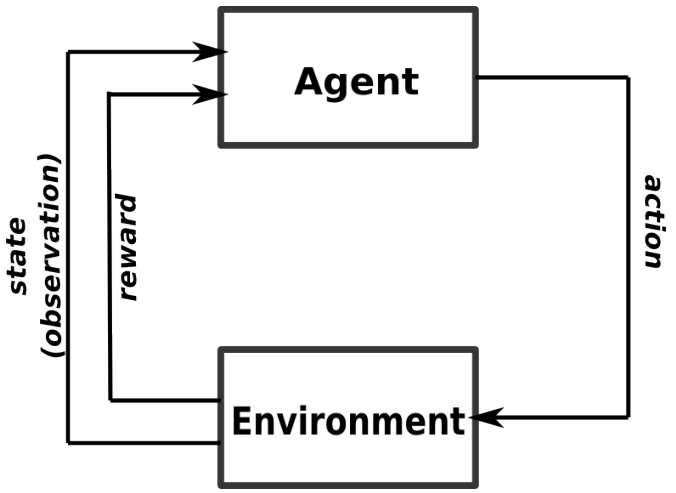


Figure 1.1: Reinforcement-learning framework

### 1.1.1 Brief Introduction to reinforcement learning

Reinforcement learning (RL) is a subset of machine learning that deals with making sequential decisions under uncertainty. Reinforcement learning involves an agent that interacts with its environment. The agent can perceive a state through receiving observations from the environment, perform actions and receive some reward (Figure 1.1).

The goal of reinforcement learning is for the agent to learn to maximize this reward. For our purposes, the final outcome resulting from a reinforcement-learning algorithm is a policy. A policy is a mapping from states to actions, indicating which action(s) to take in a given state of the environment, thus defining an agent's behavior. Reinforcement learning thus aims to solve the complete Artificial Intelligence (AI) (Russell and Norvig, 2010) problem, *an agent acting in a world, trying to maximize its utility*.

Learning in reinforcement learning can take place online, that is, with the agent having direct access to the environment or a simulator of the environment. Alternatively, learning can take place offline, that is, the agent has access

to batch data that has in it an encoding of interactions with that environment. This batch data may have been collected from another agent or from observing some particular process. Similarly, *evaluation* in reinforcement learning can take place either online or offline. Evaluation, normally in the form of measuring the performance of the resulting policy, can be tested directly on the environment in question. Conventionally, the performance of a reinforcement learning algorithm is measured as the expected return of the agent executing the policy resulting from the reinforcement-learning algorithm. Offline evaluation takes the form of using the batch data and a policy or encoding of a policy, and predicting the performance that would result in the real online environment.

### 1.1.2 The quest for better online evaluation

Reinforcement learning, like machine learning, is an experimental science (Langley, 1988). In analyzing reinforcement-learning algorithms, researchers and discuss theoretical aspects of algorithms and/or empirical performance properties. The latter is normally achieved by computing and showing the learning performance of an algorithm or algorithms in an experiment. In reinforcement learning, researchers control a large part of the experiment as well as the manner in which we evaluate the algorithms. In these experiments, the researcher chooses an environment or environments in which the learning will be carried out. The amount of learning time, typically measured as the number of learning episodes in which the algorithm is allowed to learn is varied and the online performance of the resulting policy is collected. This manner of evaluation results in *learning curves*, which are compared between algorithms. An example of such a comparison is shown in Figure 1.2

When discussing the role of experiments in machine learning, Langley (1988)

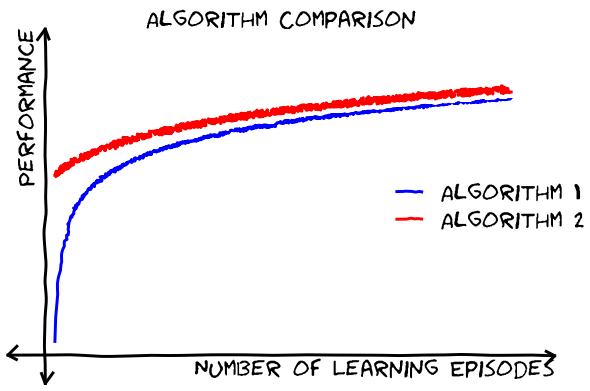


Figure 1.2: Learning curve illustration of two algorithms in an online setting.

covers a number of scenarios within which experiments can be set up to understand the behavior of algorithms. First, a researcher can choose an environment, vary the learning method and then analyze the resulting performances. In reinforcement learning, this is done by comparing one algorithm to another, producing graphs similar to the one shown earlier in Figure 1.2. Researchers may use such comparisons to show what algorithm is best for a given problem. Researchers may also compare their algorithm with different sets of underlying settings or handicaps, further illustrating algorithm behavior.

Secondly, when conducting machine-learning experiments, the researcher can vary environment characteristics. In reinforcement learning, this has not been a major factor in experimentation. Most researchers vary the type of environments in which their algorithm is applied but rarely do they vary properties of single environments. The most faithful attempt at investigating the impact of varying environment characteristics in reinforcement learning was developed in learning and evaluation criteria of the Reinforcement Learning Competitions (2007-2009) (Whiteson et al., 2010). These competitions included multiple types

of of environments. For training and evaluation within each environment, competitors were given a sample of environments drawn from a generalized environment. For example, in the classic Mountain Car type environment (Sutton, 1988), where the goal is to get an underpowered car to reach the top of a hill, the sample environments differed by having different levels of noisy observations, as well as the varying levels of stochasticity in the outcomes of actions.

More recently, the Atari Arcade environment (Bellemare et al., 2013) has also introduced an aspect of varying the environment characteristics, though arguably this is done by varying the types of environments and not properties within each environment. Concretely, the environments in the Atari Arcade environment differ at the level of varying video-games themselves (Pac-Man vs. Frogger), instead of an approach that monitors the effect of varying the configurations of a game such as Pac-Man.

The Reinforcement Learning Competitions also had such a challenge in their Polyathlon competition, where competitors could expect different types of environments to be used for learning. As such, in both the Arcade and the Polyathlon domains, the researchers wanted to spur the creation of algorithms that could better generalize across multiple problems. Such an approach to experimentation may still be undone by algorithms that simply aim to identify what type of environment they are in and then deploy the best algorithm for such a situation. Some may argue that this approach is still fine in as much as it still results in good performance. I do not believe that this is what ultimately reinforcement-learning researchers are trying to do when creating new algorithms. I believe that reinforcement-learning researchers are trying to create algorithms that have the capacity to learn and as such we would like to evaluate the capacity of algorithms to learn and not to over-fit to specific problems.

Other researchers argue against benchmark or "bake-off" domains that result from such competitions (Drummond, 2006; Langley, 2000), as they may preclude researchers from investigating other less popular problems. I wish reinforcement learning had this type of problem, but alas we do not have a heavily used testbed for our algorithms. I believe that having benchmark domains is still a good goal, encouraging researchers to share algorithms and improving cross comparability. Such benchmarks can be used to reveal interesting algorithm behavior as long as researchers can be mindful of the manner they perform experiments and sensible evaluation methodology.

I believe there is still work to do in creating evaluation methods that better capture the behavior of reinforcement-learning algorithms. I believe a mixture of both being able to vary the algorithm parameters as well as the characteristics of the environment is needed. I believe that evaluating algorithms in such experiments will give researchers a better understanding of the strengths and pitfalls of their algorithms. More importantly, I believe this type of approach will spur research into creating self contained reinforcement-learning algorithms that learn to adapt given different environments, instead of researchers instantiating algorithms such that they perform well on single environments. As such, instead of simply saying that algorithm A is better than algorithm B in environment X, we would like to be able to capture the capacity of a reinforcement-learning algorithm to learn in environments like environment X as well as intuition about how well the algorithm generalizes across environments.

### 1.1.3 Creating reinforcement-learning algorithms that deal with real world data

Treating machine learning as an experimental science, as in any science, we would like to be able to gain insight into real world phenomena that we can observe. As such, we would like to be able to have reinforcement-learning algorithms solve real world problems. To do so, we need algorithms deployed in real world applications. In most cases this is easier said than done. There are cases where environments are compact and easy to model. Take for example the breakthrough of having a reinforcement-learning algorithm learning to play the game of Backgammon at master level (Tesauro, 1994). Unfortunately, most real world problems are not easy to model. Another example are environments set in domains such as healthcare. Let's say we would like to create a reinforcement-learning algorithm that can find sequential treatments for patients suffering from diabetes. We may not be able to have direct access to the subjects for multiple reasons, least being the ethical considerations of exploring different treatments. On the other hand, environments such as flying a hobby helicopter (Ng et al., 2006) are hard problems to solve but are made even harder by the complexity of direct access to the helicopter in the learning process. Such challenges necessitate the use of high fidelity simulators. Even so, these simulators might be hard to deploy, needing specialized computation hardware, and as such, the simulators become hard to share between researchers. Complex and hard to access environments make it harder for researchers to compare methods to each other, further necessitating an approach that might side step such restrictions.

In supervised learning, a lot of progress has been made in applying methods

in the real world, especially after the introduction of the UCI database (Newman et al., 1998). The UCI database provides what Langley and Kibler (1991) term "Natural Domains" of classification and regression problems. The development and evaluation of supervised learning algorithms on the UCI database datasets has shown (and ushered) the real world applicability of such methods. An analogous database of datasets of sequential decision problems might spur on new research and discoveries in reinforcement learning. The explosion of data collected from healthcare, education and marketing, as well as the reduction in computational cost, has resulted in an opportunity for the application of reinforcement-learning algorithms. The natural question that arises is how do we now evaluate and compare different reinforcement-learning algorithms given that we only have access to the batch data?

There are few past approaches to evaluate reinforcement-learning algorithms with batch data. Later in the dissertation, we explore the approach of directly evaluating algorithms, comparing metrics, using batch data, and characterize shortcomings. Afterwards, given that the evaluation of the RL algorithms relies heavily on the data collection process, we develop methods to evaluate policies resulting from reinforcement-learning algorithms, taking into account multiple sources of uncertainty given the batch data.

### 1.1.4 Thesis Statement

The applicability of reinforcement-learning methods to real-world challenges can be improved by novel evaluation methodologies, including online procedures for characterizing the capacity of reinforcement-learning algorithms and offline evaluation procedures that account for the uncertainties resulting from the use of noisy batch data.

## 1.2 A Road-map for this document

This section gives an overview of each of the chapters of this dissertation, briefly describing the evaluation challenges tackled in each chapter.

### 1.2.1 Improved Online evaluation for Reinforcement Learning

I first tackle the problem of creating an evaluation approach that will lead to less overfitting in RL methods. I propose approaching reinforcement learning from two different levels. First, defining reinforcement learning as taking in a single environment and producing a policy for that environment. Secondly, defining meta-reinforcement learning as an algorithm that takes in a sample of environments and produces a reinforcement-learning algorithm. I explore the use of Rademacher complexity in estimating the generalization error of meta-reinforcement learning algorithms. Then utilizing a sample-optimized generalization bound to quantify generalization given sets of algorithms. Further, I show how cross-validation may be sufficient to make decisions on which algorithms to use given different situations. I present ensemble reinforcement learning as a possible meta-reinforcement learning algorithm.

### 1.2.2 Offline evaluation of reinforcement-learning algorithms

I compare several offline reinforcement-learning evaluation metrics, pointing out significant shortcomings that limit their utility. Proposing a new metric, the Relative Bellman Update Error, that scores pairs of value functions using offline data. I provide formal analysis and empirical results that suggest the Relative Bellman Update Error metric is a viable way of comparing value functions offline.

### 1.2.3 Uncertainty in offline evaluation of RL algorithms

I advocate applying techniques from batch reinforcement learning to predict the range of effectiveness that policies resulting from RL algorithms might have for individualization. Identifying three sources of uncertainty and present a method that addresses all of them. It handles the uncertainty caused by *population mismatch* by modeling the data as a latent mixture of different subpopulations of individuals, it explicitly quantifies *data sparsity* by accounting for the limited data available about the underlying models, and incorporates *intrinsic stochasticity* to yield estimated percentile ranges of the effectiveness of a policy for a particular new individual. Using this approach, I highlight some interesting variability in policy effectiveness amongst individuals in two real-world datasets. The approach highlights the potential benefit of taking into account individual variability and data limitations when performing batch policy evaluation for new individuals.

# Chapter 2
# Background

## 2.1 Introduction

In this chapter, I survey a number of important reinforcement-learning concepts that form the basis of the studies explored in later chapters. I first give a background of the reinforcement-learning framework. I then discuss two approaches to developing learning algorithms in this framework that are used in later chapters.

## 2.2 Reinforcement-learning framework

This section introduces the classical reinforcement-learning (RL) framework (Sutton, 1988). At its most basic, reinforcement learning is concerned with an agent that acts in some environment and receives some form of a reward. The agent perceives some state through receiving observations from the environment. Thus, the agent is in some state $s_k$ in the environment at time step $k$ and chooses an action $a_k$ to execute. After execution of the action in the environment, the agent receives a reward $r_k$ and moves to a next state $s_{k+1}$. This process proceeds indefinitely or to some predetermined timestep. The goal of reinforcement learning is to find a good policy in the environment that leads to a large cumulative reward.

Formally, in RL we describe the environment as a *Markov decision processes* (MDP). An MDP is a 5-tuple $\langle S, A, T, R, \gamma \rangle$ where:

- $S$ is the state space.
- $A$ is the action space.
- $T : S \times A \times S' \to [0, 1]$ is the transition function, the probabilities of transitions between states as actions are taken.
- $R : S \times A \to \Re$ is the reward function, which returns reward values as a function of states and actions.
- $\gamma \in [0, 1)$ is the discount factor, which weighs the rewards obtained from future actions versus present actions.

Over a sequence of state transitions, the *return* is the sum of discounted rewards received. A *policy* $\pi : S \to A$ is a mapping from state to action that defines an agent's behavior. That is, it gives either a deterministic or stochastic mapping of which actions to take in a given state. Given a policy $\pi$ and an MDP, I define the *value function*

$$V^{\pi}(s) = \sum_{a \in A} \pi(s, a) R(s, a) + \gamma \sum_{s' \in S} T(s'|s, \pi(s)) V^{\pi}(s'), \tag{2.1}$$

as the prediction the discounted return obtained from starting in state $s$ and following policy $\pi$. In this dissertation, this quantity is important as it is equivalent to the expected sum of discounted rewards of starting in a state $s_0$ and executing policy $\pi$, $E[\sum_{k=0}^{K} \gamma^k R(s_k, \pi(s_k))]$. Similarly, I define the *state-action value function* as

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, \pi(s)) V^{\pi}(s'), \tag{2.2}$$

that is, the prediction of the discounted return obtained from state $s$ if action $a$ is taken for one step and then actions are chosen according to policy $\pi$. Equations (2.1) and (2.2) are known as *Bellman Equations* (Sutton, 1988). Further interrogation of the state-action value function will be made later when I cover multiple approaches to evaluating reinforcement-learning algorithms with batch data.

In general, the goal of reinforcement-learning algorithms, given an MDP, is to find the optimal policy $\pi^\star$, which results in the optimal state-action value function $Q^\star$—$Q^\star(s,a) \geq Q^\pi(s,a)\ \forall \pi, s, a$. It follows that

$$V^\star(s) = \max_a Q^\star(s,a)$$

and

$$\pi^\star(s) = \operatorname*{argmax}_a Q^\star(s,a).$$

There are different approaches for learning the policy in reinforcement learning. In the following subsections, we will discuss two approaches that will be revisited in later chapters.

Given a full MDP model, we can find the optimal policy by finding $V^\star$. We can estimate $V^\star$ using the value-iteration algorithm (Bellman, 1957) shown in Algorithm 1. There are a number of termination criteria used for the algorithm. One can run the algorithm until it approximately converges. A somewhat natural way to do so is to monitor the largest change in the estimate at each iteration, $\delta$, and when it is smaller than some small value $\epsilon$, terminate. At termination, we can then extract the policy $\pi$.

There are other approaches of finding the optimal policy that do not need the direct estimation of the optimal value function, such as the policy-iteration algorithm (Sutton, 1988). The focus of this dissertation is on estimating value functions, which is a key step in policy-iteration algorithms.

**Algorithm 1** Value Iteration

**Input:** MDP $\{T, S, A, R, \gamma\}, \epsilon$
Initialize $V(s) = 0 \; \forall s \in S$
$\delta = \inf$
**while** $\delta > \epsilon$ **do**
  $\delta = 0$
  **for** $s \in S$ **do**
    $V_{prev} = V(s)$
    **for** $a \in A$ **do**
      $Q(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) V(s')$
    **end for**
    $V(s) = \max_a Q(s, a)$
    $\pi(s) = \text{argmax}_a Q(s, a)$
    **if** $||V_{prev} - V(s)|| > \delta$ **then**
      $\delta = ||V_{prev} - V(s)||$
    **end if**
  **end for**
**end while**
**return** $V$, $\pi$

## 2.3 Reinforcement-learning algorithms

In this section, I give an overview of two approaches to reinforcement learning. First, I discuss model-based reinforcement learning. Then, we discuss algorithms that do not learn the model but instead aim to estimate the value function, model-free reinforcement-learning algorithms.

### 2.3.1 Model-based reinforcement learning

Model-based approaches are a class of reinforcement-learning algorithms that aim to learn the MDP model dynamics and use this model to derive the optimal policy. That is, they aim to learn the transition function $T$ and reward function $R$ directly through interacting with the environment. The encoding of this model $M$ can vary among approaches. Later in the dissertation, I will use a tabular representation for the model.

The simplest approach to learn the model is to explore the state-space and keep statistics of states visited, actions taken and reward received. Given these tallies, we can then contract the maximum likelihood estimate (MLE) transition model and reward function. After constructing the model, we can then use algorithms such as Algorithm 1 to find the optimal policy or estimate of the optimal value function for this approximate model.

Collecting these tallies requires the availability of an algorithm to choose actions while interacting with the MDP. After arriving in a state, most model-based algorithms carry out two important steps. First, the model-based algorithm chooses an action given its estimate of the best action given value iteration. Afterwards, in the second step, the algorithm experiences a state transition and receives some reward, updating its tallies. Given this information, the algorithm updates its model and goes back to the first stage. In the data collection phase, a balance has to be struck between exploring different parts of the state-action space and exploiting the value estimates gained during the learning process; this challenge is known as the exploration-exploitation trade-off (Sutton, 1988). As such, the algorithms usually consist of an action-selection policy, e.g. Epsilon Greedy, that has an accompanying exploration parameter.

In a second setting, data is collected before any planning takes place. The RL algorithm is given a pre-collected set of experience data. In this batch setting, the success of the algorithm relies on data being collected well. In the latter part of this dissertation, I will show how to evaluate policies using this model-based approach. Model-based approaches are easy to use when the state space is discrete, small and tractable, since the tallying operation is sufficient for learning an accurate model. When the state space is continuous, one can either discretize the continuous input from the MDP or estimate models using

better underlying function approximators (Atkeson et al., 1997).

### 2.3.2 Model-free algorithms

Earlier, I discussed algorithms that first learn the model of the MDP and then learn the optimal policy through algorithms such as value iteration. In this subsection, I discuss approaches that require no models. A classic approach is to use temporal difference algorithms (Sutton, 1988). Temporal difference (TD) methods use experience tuples $(s, a, r, s')$ representing transitions in the environment. The simplest implementation, TD(0), updates the estimate of the state-action value function using the update rule

$$Q_{l+1}(s, a) = Q_l(s, a) + \alpha(r + \gamma \max_{a'} Q_l(s', a') - Q_l(s, a)), \tag{2.3}$$

where $\alpha$ is the learning rate. Given the experience tuples, each time we experience a state action pair, $(s, a)$, the estimate of $Q(s, a)$ is updated to be closer to $r + \gamma \max_{a'} Q(s', a')$. With infinite experience of each action in each state, this algorithm is guaranteed to converge to the optimal state-action value function $Q^\star$ (Watkins and Dayan, 1992).

There are different TD-based algorithms. There are relatively simple algorithms such as Q-learning (Watkins and Dayan, 1992), which uses a simple update rule and is policy independent. An example of a policy-dependent algorithm is SARSA (Sutton, 1988), which is similar to Q-learning but incorporates action choices from the policy that would result from its estimate of the value function as part of the update. That is, the update is of the form

$$Q_{l+1}(s, a) = Q_l(s, a) + \alpha(r + \gamma Q_l(s', \hat{a}) - Q_l(s, a)), \tag{2.4}$$

where $\hat{a}$ is the action chosen by the policy. Thus, for SARSA, a learning step

is comprised of tuple $(s, a, r, s', \hat{a})$ . For both model-free and model-based algorithms, representation of the learned information can use different forms of function approximation. In small, discrete environments, the tendency is to use tabular representations of state and actions. In continuous states and/or action environments, other function approximation schemes are used. As such, the algorithms discussed above may incorporate different function approximation schemes. Convergence when using function approximation may not be guaranteed (Baird, 1995), but using function approximation has allowed reinforcement-learning algorithms to scale to some large problems (Busoniu et al., 2010; Kober et al., 2013).

The model-free algorithms described above tend to be used online, updating their estimates during data collection. The policies resulting from their estimates are then used to collect more data. There is another set of algorithms that are more suited to learning from batch data. Algorithms such as Least-Squared Policy Iteration (Lagoudakis and Parr, 2003) and Kernel Based RL Ormoneit and Śaunak Sen (2002) estimate the optimal Q-value using only pre-collected batch data. I will use such algorithms later when discussing metrics for evaluate RL algorithms using only batch data.

# Chapter 3
# Improved Online Evaluation of Algorithms

Part of the work done in this chapter was accomplished with the collaboration of Michael Littman, James MacGlashan, Matthew E. Taylor, Carl Trimbach and Eli Upfal

## 3.1 Introduction

Classically, the input in the training phase in reinforcement learning is an MDP and an algorithm. The output of this first phase is a policy. In the evaluation phase, we take the resulting policy and deploy it in the same MDP used for training. We record the cumulative discounted return during this evaluation. We compare this metric, the cumulative discounted return, across different policies created by different algorithms. This type of evaluation is classed as being *online*, that is, it is done with access to the real MDP.

Typically, we plot the performance of the policies resulting from algorithms versus the amount of training/learning experience used to create the policy. An example of such a plot is illustrated in Figure 3.1 with policies from 2 algorithms, Algorithm 1 and Algorithm 2.

This type of evaluation highlights how fast or slow algorithms converge to policies with (hopefully) a high level of performance. In the illustration, we see that with very low training episodes, we would likely make the choice of using Algorithm 2. But, as the amount of learning experience increases, we see

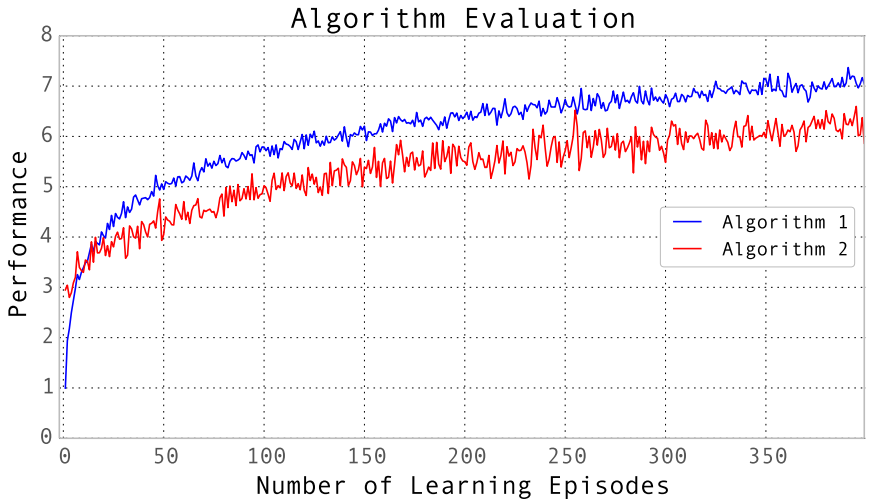


Figure 3.1: Illustration of a typical learning curve comparing the performance of policies resulting from two algorithms

that Algorithm 1 produces the best policies, leading to the highest accumulated reward. This has, for a long period of time, been the standard practice in comparing algorithms in the community. When comparing algorithms in different environments, researchers would likely only report the performance at some high number of learning episodes. Thus, in making choices about which algorithms to use, we would use the performance of an algorithm in a single MDP as a data point.

A question that needs to be asked then is how well does an algorithm's performance generalize across different problems. Using a single MDP is akin to fitting a curve to a single data point. That is, training and evaluating an algorithm on a single MDP is unlikely to be very instructive in predicting performance on other MDPs.

As such, the logical—by supervised learning standards—next step is to find

way of evaluating algorithms that allows effective generalization to other problems. In particular, evaluation should discourage overfitting. In the MDP setting, problems are MDPs.

In this chapter, I explore the approach of using *distributions* of MDPs to evaluate classes of algorithms. We can think of a class of algorithms as algorithms mixed with its parameter space.

### 3.1.1 Related work

In supervised machine learning, there is a lot of work that has gone into understanding the behavior of algorithms given different amounts of data and difficulty in generalization. Generalization is how well we can predict the performance of an algorithm on an unseen test set given its performance on the training set. Given that we choose an algorithm from its class to maximize performance on the training set, a principal challenge is how to perform an evaluation in such a way that discourages *overfitting*—good training set performance leading to poor test set performance (Dietterich, 1995).

In reinforcement learning, some other work has looked at approaches to training and testing learning algorithms to encourage better generalization. One of the largest efforts was the Reinforcement Learning Competitions (2007–2009) (Whiteson et al., 2010) and their associated evaluation approach (Whiteson et al., 2011). Their perspective is to imagine that MDPs are drawn from a *generalized environment* $\mathcal{U} : \Theta \to [0, 1]$ from which individual MDPs can be sampled. Given this framework, first a training sample of MDPs is used to choose a reinforcement-learning algorithm and its parameters, after which this algorithm is evaluated on a testing set drawn from the same distribution. While important for insuring the validity of the results in the competition, use of this

approach is still not widespread. This chapter first provides support for this type of approach and then looks at evaluation procedures we can use to gain better understanding of known algorithms and how their performance varies with different MDP characteristics.

A more recent approach that is related is the use of the Atari Arcade video game MDP (Bellemare et al., 2013). For clarity, I will refer to it as the Atari Arcade platform. The Atari Arcade platform introduced the concept of varying the environment 'types' by varying specific video games (Pac-Man vs. Frogger) that an algorithm has access to for training and testing. The video games could be sports games, action games, adventure games etc. The focus is less on what happens when one takes a game such as Pac-Man and monitors the effects of changing game properties such as the number of enemies (ghosts). The Reinforcement Learning Competitions themselves had a challenge similar to the Atari Arcade platform, albeit at a smaller scale and ambition. The challenge was known as the *Polyathlon*. In the Polyathlon, both the types of environments and individual MDP parameters were varied (Whiteson et al., 2010). So a type of environment could be a gridworld, a physics based environment etc. Within the training and tests sets there could be different variations of the same gridworld or physics environment.

In both the Arcade and the Polyathlon domains, the researchers wanted to spur the creation of algorithms that can better generalize across multiple problems. (Dabney et al., 2013) introduced a performance metric that dealt with the biases that might be introduced when tuning parameters for algorithms and/or when researchers try to compare their tuned algorithms in MDPs to those tuned by other researchers.

### 3.1.2 Outline

This chapter is split into 3 sections. I discuss current reinforcement-learning evaluation methods and show why the evaluation methods may not align with the goals of researchers. I then explore online evaluation approaches that better measure the learning power of RL algorithms. That is, I explore evaluation approaches that quantify the generalization capability of algorithms. To do so, I introduce the problem of meta-reinforcement learning and methods for evaluating meta-reinforcement-learning algorithms. Lastly, I discuss potential meta-reinforcement learning algorithms, discussing ensemble reinforcement learning and connecting this approach to ensemble supervised learning.

## 3.2 Generalization in reinforcement learning

In this section, I will explore the concept of generalization in RL by using a small but interesting MDP to shed light on the pitfalls of making algorithmic decisions using a single MDP. I also introduce the problem of the meta-reinforcement learning, an expansion of the reinforcement-learning problem that incorporates learning and evaluation on multiple MDPs.

### 3.2.1 The 5-state chain

What if we would like a learning algorithm to generalize across different MDPs? To explore this question, I describe a running example that will be used in this chapter. I introduce the 5-state chain MDP.

The 5-state chain environment consists of 5 states and 2 actions, **a** and **b** (Figure 3.2). In the classic version of the environment, the effect of the actions in this system are stochastic. With $p = 0.2$ as the *slip probability*, there is a

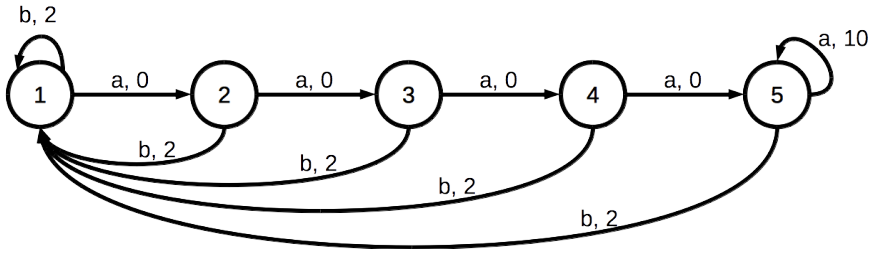


Figure 3.2: The 5-state chain (Strens, 2000)

0.8 $(1 - p)$ probability that the agent will move in the direction of the chosen action and a 0.2 $(p)$ probability that the agent will move in the direction of the other action. A reward of 0 is given for choosing **a** in any state. The exception is that in State 5 it results in a reward of 10. In all the states, action **b** results in a reward of 2. The discount factor used is $\gamma = 0.95$. For this environment, performance of a learning algorithm is measured as the probability of finding the optimal policy for the MDP after 1000 steps of learning experience.

First, for learning in the classic 5-state chain, I consider two variations of the widely used Q-learning learning (Watkins and Dayan, 1992). For both variants, the learning rate $(\alpha)$ is varied between 0.0 and 0.5 and the exploration rate $(\epsilon)$ varied between 0.0 and 0.4. In the *constant initialization* Q-learning algorithm, all initial Q values are set to 45 (a value near the optimal state-action value function). In the *variable initialization* algorithm, each state-action pair has an initial Q-value set independently to a value between 0 and 200. We observe that the parameters of the first algorithm are a strict subset of those of the second algorithm.

We would like to try and answer the question: Which algorithm results in the best performance? To do so, we can generate multiple Q-learning algorithms by sampling different parameters and then testing them in the MDP. What we find is that the best parameter values for the constant initialization

Table 3.1: Initial Q-Values of best variable initialization algorithm in Five-State chain

| | $Q(s, a = \text{a})$ | $Q(s, a = \text{b})$ |
|---|---|---|
| $s = 1$ | 93.86 | 78.53 |
| $s = 2$ | 85.27 | 104.07 |
| $s = 3$ | 93.52 | 19.53 |
| $s = 4$ | 136.77 | 0.42 |
| $s = 5$ | 194.82 | 80.62 |

algorithm results in a performance (probability of finding the optimal policy) of 0.945 and, with the variable initialization algorithm, we find the best parameters result in a performance of 1.000. The parameters of the best performing constant initialization algorithm are $\alpha = 0.16, \epsilon = 0.40$. For the best performing variable initialization algorithm, the parameters are $\alpha = 0.05, \epsilon = 0.16$ and the initial Q-values shown in Table 3.1.

Is it sensible then to recommend preferring the variable initialization algorithm over the constant initialization algorithm at all times? With only a single MDP, we might be overconfident in the general performance of one algorithm over another. The better thing to do is to compare the algorithms across the distributions of MDPs we would like our algorithms to be deployed in. Consider two different distributions of 5-state chain MDPs:

- *Static Order* 5-state chain distribution: The MDPs are 5-state chains with varying slip probabilities between 0.19 and 0.21 and the order of the states is consistent with the original 5-state chain. This distribution only contains chains very similar to the original 5-state.

- *Dynamic Order* 5-state chain distribution: The distribution consists of 5-state chains with a variable order of the states and slip probabilities varying between 0.0 and 0.5. An example with states reordered is shown in

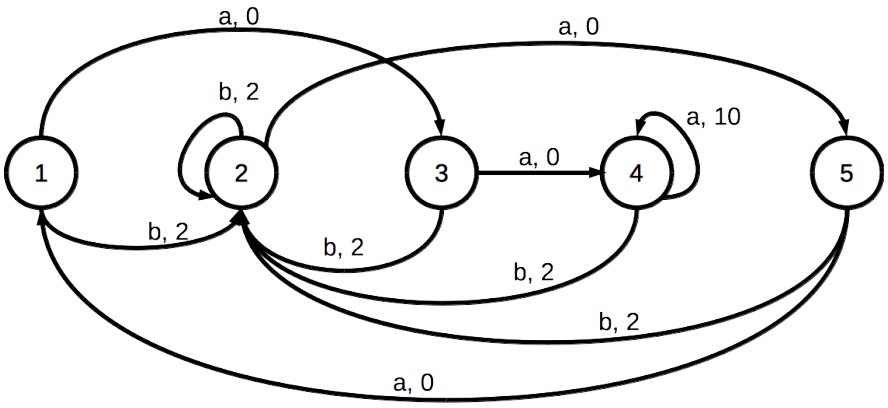


Figure 3.3: A sample of the Dynamic Order 5-state chain

Figure 3.3. The Static Order distribution is a subset of MDPs drawn from the Dynamic Order distribution, while most MDPs drawn from the Dynamic Order distribution would not be drawn from the Static Order distribution.

The best algorithm from the variable initialization for the original 5-state chain performs poorly on average in the Dynamic Order 5-state chain MDP distribution. The variable initialization algorithm achieves an expected performance of 0.293 (that is, it has 0.293 probability of finding the optimal policy in the distribution). What this result indicates is that by tuning the parameters, specifically the initial state-action values, of the algorithm to the original, 5-state chain single MDP, the algorithm ends up encoding part of the MDP itself within its parameters. The setting results in poor performance if the MDP distribution that it is then deployed on is sufficiently dissimilar. On the other hand, the constant initialization algorithm performs better than the variable initialization algorithm in this space, achieving an expected performance of 0.632 in MDPs drawn from the distribution. As such, the variable initialization algorithm *overfit* the original 5-state chain.

However, when comparing the relative performance of the two algorithms

in the Static Order MDP Distribution, the variable initialization performs as well as in the original 5-state chain (achieving a performance level of 0.999) and the constant initialization algorithm does slightly worse (achieving a performance level of 0.901). What is observed in the Static Order MDP distribution is *underfitting* by the constant initialization algorithm. Had we chosen are more flexible algorithm class, we would have gotten better performance. This simple example illustrates why we need to change the way we evaluate RL problems to take into account variations in the space of possible test MDPs. To that end, I introduce the meta-reinforcement learning problem next.

### 3.2.2 Meta-reinforcement learning

When we choose a reinforcement-learning algorithm, we want it to work well in the environments in which it will be deployed. The classic reinforcement-learning problem is deriving a good policy for an MDP given the ability to interact with it. This definition is generalized to capture learning in the presence of MDP distributions. Let $X$ be the space of possible environments and $D$ be a distribution over these environments. Let $G$ be a set of available learning algorithms. The set $G$ could be made up of individual algorithms or sets of different combinations of algorithms (an ensemble). The *meta-reinforcement learning* problem is, given $G$ and a sample $\mathcal{S}$ of environments drawn from distribution $D$, find a learning algorithm $\hat{g}$ that results in good expected performance on MDPs drawn from $D$.

Formally, let $F$ be a finite set of functions on a domain $X$ such that for all $f \in F, f : X \to [0, 1]$. Concretely, we have $F = f_g | g \in G$, where $f_g(x)$ is the outcome of running learning algorithm $g$ on MDP $x$. We call $F$ the set of *evaluation functions* for algorithm set $G$.

Ideally, a meta-reinforcement learning algorithm outputs the best algorithm,

$$g^\star = \underset{g \in G}{\operatorname{argsup}}\, E_D[f_g(x)] = \underset{g \in G}{\operatorname{argsup}} \int_X f_g(x) P(x) dx. \tag{3.1}$$

But, as the algorithm will only have access to a sample $\mathcal{S}$ made up of $x_1, \ldots, x_m$ drawn from distribution $D$ on $X$, then choosing the best algorithm given the sample results in

$$\hat{g} = \underset{g \in G}{\operatorname{argsup}}\, \hat{E}_{\mathcal{S}}[f_g(x)] = \underset{g \in G}{\operatorname{argsup}} \frac{1}{m} \sum_{x \in \mathcal{S}} f_g(x). \tag{3.2}$$

For this problem formulation, if a single MDP is representative of the class of environments that will be encountered by the learning algorithm (an example being the Static Order 5-state chain distribution), then there is no need to learn. A fixed policy will suffice for any MDPs within that class of environments. If the single MDP is not representative of the class of environments, we cannot choose an algorithm that does well in a single MDP as that algorithm is likely to overfit.

In reality, we do not have access to the full distribution $D$ but will have some sample $\mathcal{S}$. We use the samples $\mathcal{S}$ as a training set available to us. But, how large should the training set be so that we can make good decisions in selecting the algorithm to be deployed in the full distribution? That is, how do we reduce the generalization error, $|\hat{E}_{\mathcal{S}}[f_{\hat{g}}] - E_D[f_{\hat{g}}]|$, the difference between the performance we optimize for in the sample (training) set and the actual expected performance of that algorithm in the full distribution?

To illustrate this issue, I return to the two 5-state chain distributions explored earlier. Static Order distribution results are shown in Figure 3.4 and Dynamic Order distribution results are shown in Figure 3.5. I plot the expected performance of the best algorithms—$\hat{g}$ for both the constant and variable initialization Q-learning using the training set $\mathcal{S}$—as solid lines. I also include

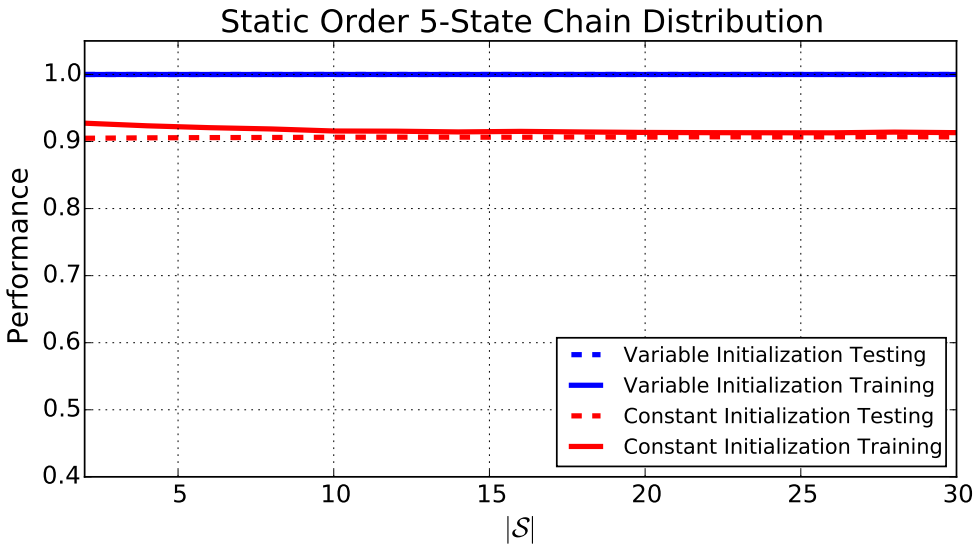


Figure 3.4: Performance in the Static Order 5-state chain environment

the performance, illustrated as dotted lines, of those best algorithms in the testing set – the full distribution approximated by sampling 1000 MDPs from each distribution.

For the Static Order distribution (Figure 3.4) , the constant initialization algorithms consistently underfit—they are not able to reach as good performance as the variable initialization (a more powerful algorithm space). In the Dynamic Order distribution (Figure 3.5), though, we observe that with very few training samples, $|\mathcal{S}|$ less than 4, the variable initialization algorithm has the largest training performance but the worst testing performance—it has large generalization error. Variable initialization is clearly overfitting with few training samples. On the other hand, the constant initialization has a lower training performance, but a lower generalization error. As the amount of data is increased, the training performance for both algorithms is reduced, indicating the richness of the set of MDPs, while the testing performance increases. This

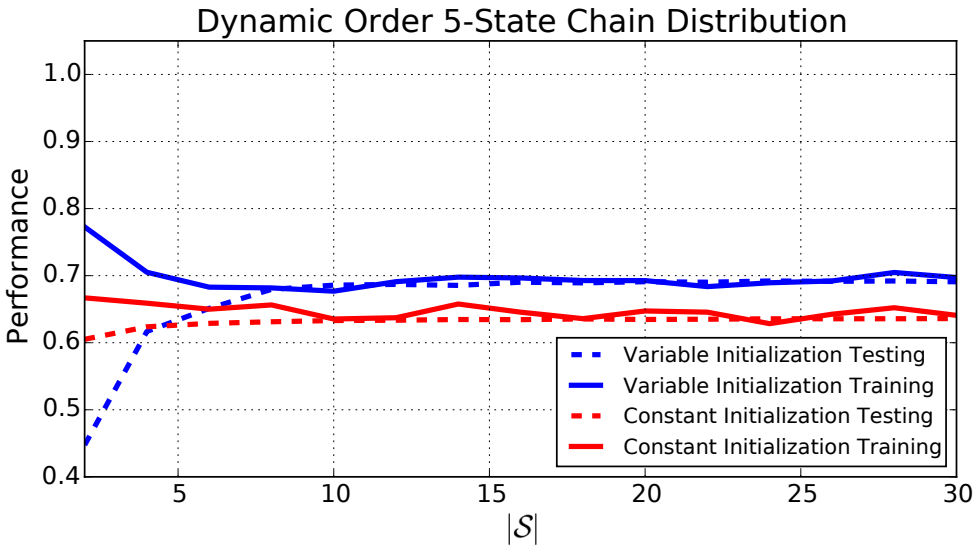


Figure 3.5: Performance in the Dynamic Order 5-state chain environment

pattern is familiar, as it mimics the learning curves regularly observed in supervised learning. With enough training data, the variable initialization algorithm has better performance on the full distribution than the constant initialization algorithm. Had we not had the testing performance on the full distribution available and only a few training samples, choosing the variable initialization algorithm may have led to bad choices for the full distribution. In the absence of enough training samples, the better choice would have been to choose the algorithms from the constant initialization set instead. In practice, we would not have access to the testing performance, and as such there is a need to look at methods that allow us recognize overfitting based on training performance alone.

### 3.2.3 Unresolved questions

How can we better create reinforcement-learning algorithms that generalize well? To answer that question, we need a mechanism by which we can estimate

the generalization error of algorithms given a small sample from the environment distribution. I cannot reasonably expect researchers to run their learning algorithms on an infinite set of environments. Hidden in this approach is the question of quantifying how hard it is to find good parameters for an algorithm given only a sample of environments. This question is not trivial as it also contains within it problems that might face researchers who are not familiar with unwritten specifics of certain reinforcement-learning algorithms. Further, we would like to know how many samples are necessary for us to be able to make meaningful conclusions about the performance of algorithms. In the next section, I introduce a procedure for evaluating sets of reinforcement-learning algorithms online built on top of the theoretical underpinning of function complexity.

## 3.3 Evaluation for meta-reinforcement learning

Given a set $\mathcal{S}$ of MDP samples, we would like to be able to predict how well one set of algorithms performs compared to another set of algorithms. Concretely, we would like to predict the generalization error of different sets of algorithms. One way to do so is to use the concept of the complexity of the set of evaluation functions $F$. The complexity defines how expressive the set of algorithms $G$ is relative to a set of environments $X$. Intuitively, if the complexity of an algorithm is small, then the full distribution performance of an algorithm—even with a small set of training samples $\mathcal{S}$—will be very narrow. If the complexity of an algorithm is large, the performance of the best algorithm chosen in the training set in the full distribution will vary. As such, a larger set of samples will be needed to better find the best algorithm.

### 3.3.1 Rademacher complexity

In supervised learning, specifically, Boolean classification, hypothesis class complexity is often expressed in terms of *VC dimension* (Mohri et al., 2012). Given a hypothesis class of classifiers $C$ defined over inputs $Z$, the VC dimension quantifies the complexity of $C$. The VC dimension of $C$, $d(C)$, is the size of the largest set of inputs that, for all possible assigned labels of those inputs, a function can be produced that correctly labels all the points. We are then able to produce a *uniform convergence bound*, a probabilistic bound that holds uniformly for all functions in $C$, in terms of $d(C)$. These bounds are then used to bound the generalization errors of learning algorithms. The VC dimension is known for a number of hypothesis classes.

Unfortunately, VC dimension does not apply directly to the hypothesis classes that appear in reinforcement learning and as such we cannot use it to produce uniform convergence bounds. A related complexity measure, which applies more broadly not just Boolean classifiers, is the Rademacher complexity. Rademacher complexity (Bartlett and Mendelson, 2003), $\mathcal{R}$, quantifies the richness/expressiveness of a function set with respect to a target distribution. In the RL case, we would be measuring the richness of a set of reinforcement-learning algorithms relative to a target environment distribution. The Rademacher complexity can be used to produce a uniform convergence bound. Rademacher complexity has been used in the past in the RL setting to analyze the Generalized Classification-based Approximate Policy Iteration framework (Farahmand et al., 2012).

The definition of the Rademacher complexity is: Given a finite set of functions $F$, the Rademacher complexity is

$$\mathcal{R}_m(F) = E_{\mathcal{S}}[E_\sigma[\sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i f(x_i)]], \tag{3.3}$$

where $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$ is a vector of $m$ independent random variables such that $\sigma_i \in \{-1, +1\}$, $P(\sigma_i = 1) = (\sigma_i = -1) = 1/2$ and $\mathcal{S}$ is a set of MDPs sampled from $D$, $\mathcal{S} = \{x_1, x_2, \ldots, x_m\}$. The empirical Rademacher complexity for a fixed sample $\mathcal{S}$ is

$$\mathcal{R}_{\mathcal{S}}(F) = E_\sigma[\sup_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} \sigma_i f(x_i)]. \tag{3.4}$$

The $\sigma$ variables intuitively create different configurations of the dataset, producing different noisy variations of the input dataset. The supremum thus carries out an optimization over all functions in $F$ for these different variations of the dataset. If the function class is rich, then it will, on average, be able to fit the noisy input sample set. The Rademacher complexity is high if the function class $F$ can fit these noisy variations. If so, larger input sets would be needed to avoid overfitting.

An important uniform convergence bound using Rademacher complexity (Mohri et al., 2012, Theorem 3.1) says, with probability $1 - \delta$ for samples $\mathcal{S}$, for any function $f \in F$, we have that

$$E_D[f(x)] \leq \hat{E}_{\mathcal{S}}[[f(x)] + 2\mathcal{R}_m(F) + \sqrt{\frac{\ln(1/\delta)}{2m}}, \tag{3.5}$$

and

$$E_D[[f(x)] \leq \hat{E}_{\mathcal{S}}[f(x)] + 2\mathcal{R}_{\mathcal{S}}(F) + 3\sqrt{\frac{\ln(2/\delta)}{2m}}, \tag{3.6}$$

for the empirical Rademacher complexity. The Rademacher complexity thus gives us an approach to quantify the expressiveness of function class $F$, given

only samples of environments. One difference from VC dimension is that it depends on the inputs and not just the function class. As such, we are able to make decisions on the potential generalization performance of a set of algorithms given a smaller set of environments. It is a very good fit for understanding generalization error in meta-reinforcement learning. It tells us more about a set of learning algorithms than we can learn from just using single environments for evaluations.

One challenge is that, in practice, it is hard to compute the supremum in the definition of $\mathcal{R}_{\mathcal{S}}(F)$. An alternative is to adapt the bound such that we only need to calculate the maximum approximately. In the next subsection, the above bound is adapted for a specific kind of approximation to the supremum—the maximum computed over a size $k$ set of functions sampled independently from $F$, $F_k$.

### Sample-optimized generalization bound

I now handle the case in which we have a smaller finite set of $k$ functions sampled from $F$, $F_k$, with $H$ as a probability distribution over $F$. With only $k$ functions, Equation 3.4 now changes to $\mathcal{R}_{\mathcal{S}}(F_k)$, which is easier to calculate as we now use a maximum over the smaller set of functions, $F_k$, instead of a supremum. With the set $F_k$, the function with the best expected performance over the the samples $\mathcal{S}$ is

$$f_{\mathcal{S}}^k = \underset{f \in F_k}{\operatorname{argmax}} \frac{1}{m} \sum_{i=1}^{m} f(x_i).$$

The accompanying generalization error with $F_k$ is $|\hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k] - E_D[f_{\mathcal{S}}^k]|$.

**Theorem 3.3.1.** *The expected generalization error $E_{H,\mathcal{S}}[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k]]$ is bounded*

*by* $2R_m(F)$*, and with probability* $1-\delta$ *over choices of* $\mathcal{S}$*,*

$$E_H[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k]] \leq 2E_H[\mathcal{R}_m(F_k)] + \sqrt{\frac{\ln(1/\delta)}{2m}}.$$

*Proof.* For a fixed $F_k$, applying the Rademacher bound (Equation 3.5), we have simultaneously for every $f \in F_k$,

$$E_{\mathcal{S}}[E_D[f] - \hat{E}_{\mathcal{S}}[f]] \leq 2\mathcal{R}_m(F_k).$$

In particular, this statement holds for the function $f_{\mathcal{S}}^k$. Thus, for a fixed $F_k$,

$$E_{\mathcal{S}}[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k]] \leq 2\mathcal{R}_m(F_k).$$

Taking expectation over the choices of $F_k \subseteq F$, we have

$$E_{H,\mathcal{S}}[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k] \mid F_k] \leq 2E_H[\mathcal{R}_m(F_k) \mid F_k].$$

We now observe that since $0 \leq f(x) \leq 1$, a change in one sample $x_i$ can only change the value of $E_{H,\mathcal{S}}[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k] \mid F_k]$ by $1/m$. Applying the Azuma-Hoeffding martingale inequality (Mitzenmacher and Upfal, 2005, Theorem 12.4), as in the argument for Rademacher bounds, we prove with probability $1-\delta$ over the distribution of $\mathcal{S}$,

$$E_H[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k] \mid F_k] \leq 2E_H[\mathcal{R}_m(F_k) \mid F_k] + \sqrt{\frac{\ln(1/\delta)}{2m}}.$$

□

Changing the sup to a max over a finite samples of functions in the empirical Rademacher bounds results in bounds that are similar and hold over the average finite sample. There is a need for a practical method that accurately estimates $E_H[\mathcal{R}_m(F_k)]$ from the samples as it is not possible to compute the exact expectation.

**Corollary 3.3.1.** *Assume that we choose $\ell$ random sets $F_k^1, \ldots, F_k^\ell$ from $F$ according to the distribution $H$, and a set $\mathcal{S}$ of $m$ random environments according to the distribution $D$. With probability $1 - 2\delta/3$ over the choice of $\mathcal{S}$ and the $F_k^j$s,*

$$E_H[\mathcal{R}_m(F_k)] \leq \frac{1}{\ell} \sum_{j=1}^{\ell} \mathcal{R}_{\mathcal{S}}(F_k^j) + \sqrt{\frac{\ln(3/\delta)}{2m}}.$$

*Proof.* For a fixed $F_k$, the bound on the empirical Rademacher complexity (Mohri et al., 2012, Equation 3.14) gives, with probability $1 - \frac{\delta}{3}$,

$$\mathcal{R}_m(F_k) \leq \mathcal{R}_{\mathcal{S}}(F_k) + \sqrt{\frac{\ln(3/\delta)}{2m}}.$$

Applying the same Azuma-Hoeffding martingale inequality to bound

$$\left| \frac{1}{\ell} \sum_{j=1}^{\ell} \mathcal{R}_{\mathcal{S}}(F_k^j) - E_H[\mathcal{R}_{\mathcal{S}}(F_k)] \right|.$$

Given that $f \in [0, 1]$ for all $F_k$, changing a single $x_i$ may change each function by at most 1. Therefore, $\mathcal{R}_{\mathcal{S}}(F_k^j)$ changes by at most $1/m$ since it is an empirical average over $m$ such functions. The total change in the empirical average over the Rademacher values that results is $\frac{\ell}{m} \cdot \frac{1}{\ell} = \frac{1}{m}$. Applying the inequality, we get

$$\Pr\left( \left| \frac{1}{\ell} \sum_{j=1}^{\ell} \mathcal{R}_{\mathcal{S}}(F_k^j) - E_H[\mathcal{R}_{\mathcal{S}}(F_k)] \right| \geq \sqrt{\frac{\ln(3/\delta)}{2m}} \right) \leq \delta/3.$$

□

Using Corollary 3.3.1, we can remove the exact expectations over $H$ and $\mathcal{S}$ in Theorem 3.3.1.

**Corollary 3.3.2.** *With probability $1 - \delta$*

$$E_H[E_D[f_{\mathcal{S}}^k] - \hat{E}_{\mathcal{S}}[f_{\mathcal{S}}^k]] \leq \frac{2}{\ell} \sum_{j=1}^{\ell} \mathcal{R}_{\mathcal{S}}(F_k^j) + 5\sqrt{\frac{\ln(3/\delta)}{2m}}.$$

These results justify an approach to meta-reinforcement learning using sets of algorithms. Given the sample of environments $\mathcal{S}$ and set of algorithms $G$, create a *mixture ensemble* over $G$ by repeating, $\ell$ times, the process of selecting $k$ algorithms from $G$ and then selecting the one with the best training performance. Now, assign each of these winning algorithms a probability of $1/\ell$ in the mixture ensemble. That is, when deploying the meta-reinforcement learning algorithm for testing, we deploy the mixture ensemble and sample uniformly from the $\ell$ winning algorithms. These bounds provide a way to measure the generalization error of this mixture ensemble.

### 3.3.2 Sample-optimized generalization bound experiments

Here, I present the results of applying the sample-optimized generalized bounds to the 5-state chain and a larger a second environment.

#### 5-state chain experiments

I apply the sample-optimized Rademacher bounds derived to the two 5-state chain distribution environments. Figure 3.6 illustrates the sample-optimized bounds for the earlier 5-state chain examples.

For these plots, I show the performance of using a mixture ensemble with $k$ functions. That is, I sample $k$ Q-learning algorithms for both constant and variable initialization algorithms. In contrast with out earlier plots, in Figures 3.4 and 3.5, I plot the expected training and testing performance of using the best algorithm found by taking the max over $k$ sampled algorithms. As such, in the plots, $k$ indicates the number of evaluations done, or the "power" of the optimization. The training and testing performance is averaged over 50 runs, and, for the sample-optimized bound, $\ell = 50$.

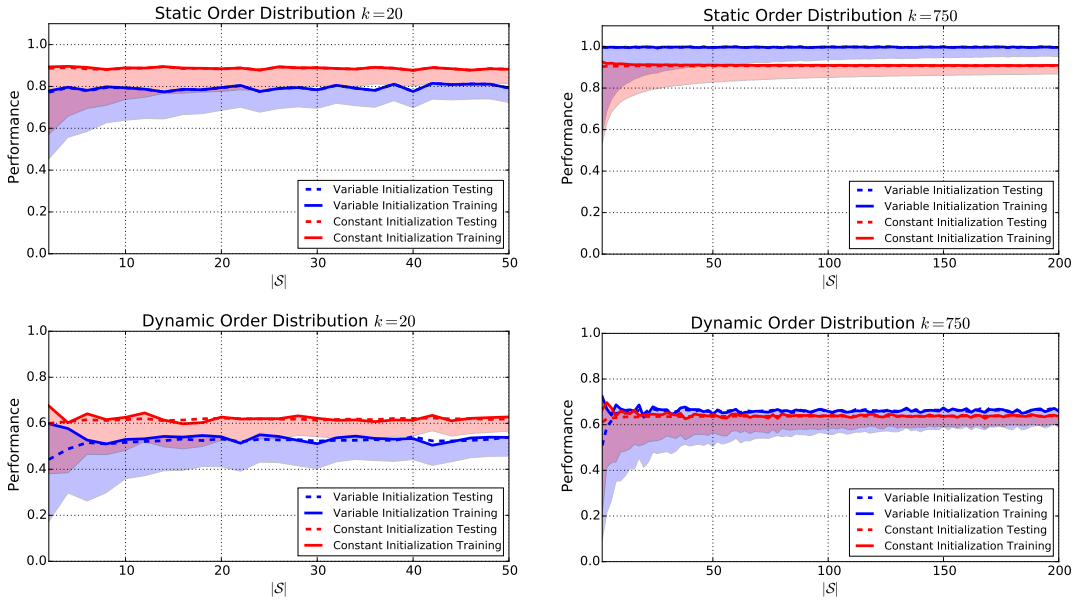


Figure 3.6: Sample-optimized Rademacher bounds for the Static and Dynamic Order 5-state chain MDP distributions at different values of $k$

For the Static Order distribution (top plots of Figure 3.6), note that, when using $k = 20$, the constant initialization algorithm has a higher training and testing performance as well as a lower sample-optimized Rademacher bound compared to the variable initialization algorithm. The variable initialization algorithm has a lower training and testing performance and the sample-optimized Rademacher complexity is larger, leading to an even lower lower bound. What is important to note is that, for all the plots, with a small MDP sample size, the sample-optimized Rademacher bound is wider but gets narrower as more training data is available. For both the constant and variable initialization, with a larger algorithm function set, $k = 750$, the training and testing performance increase and the generalization bounds also increase in width. The variable initialization now results in the best training performance and the bound also shows that it is better to choose the variable initialization algorithm in tackling problems from the Static Order distribution.

For the Dynamic Order distribution (lower plots of Figure 3.6), we have a similar situation for both optimization strengths as in the Static Order distribution experiments. With $k = 20$, the constant initialization algorithm has higher training performance and the lower bound on the performance is higher than the variable initialization. With $k = 750$, the variable initialization algorithm has a higher training performance but the lower bound is much lower than that of the constant initialization for $|\mathcal{S}| < 200$. As the size of the MDP training set increases, both algorithm bounds decrease but it takes a large sample size to make the decision to use variable initialization over constant initialization given the generalized bound. This is in contrast to observing that, for the testing performance, we would make the decision at $|\mathcal{S}| = 20$ for $k = 750$. As such the sample-optimized Rademacher bound is pessimistic.

**Mountain car**

The second environment I use in this chapter is the classic Mountain Car (Figure 3.7). Here, the goal is to get an under-powered car to reach the top of a hill. The MDP's state space has two dimensions (position and velocity) and three actions (forward throttle, backward throttle and no throttle). The system has a reward of 0.0 when the car reaches the top of the hill and a reward of $-1.0$ for every time point when the car is not at the top of the hill. The discount factor for this MDP is $\gamma = 0.95$.

In the classic configuration, the MDP is deterministic and the car always starts at the bottom of the hill. For our purposes, I make two changes. First, I make the MDP stochastic by adding varying levels (per MDP) of noise to the force applied by the throttle. Second, I vary the gravity variable for each MDP.

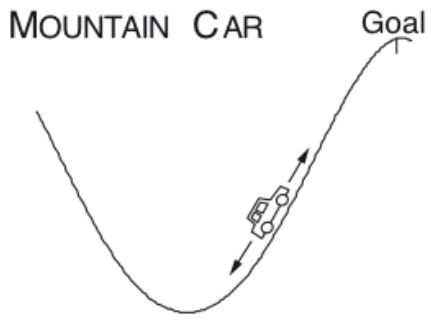


Figure 3.7: Mountain car environment from Sutton (1988)

For simplicity, I measure the performance of a reinforcement-learning algorithm as the negated expected number of steps to goal of the policy it identifies after 50 learning episodes (each episode can last up to 1000 steps). In experiment plots, I present this value as normalized between 0 and 1.0, so higher is better.

The learning algorithms in the Mountain Car experiments come from both classic Q-learning and Q-Lambda (Q-learning with eligibility traces (Sutton, 1988)). Both algorithm sets have a discretization of 10 blocks for each continuous dimension, and I vary the learning rate ($\alpha$) between 0.0 and 0.5. I also vary the exploration rate ($\epsilon$) between 0.0 and 0.5. The default initialization value for state–action values is fixed at 0.0 for both algorithm sets. The difference in the two algorithm sets is that *Algorithm 1* effectively has a fixed $\lambda$ of 0.0, while the second, *Algorithm 2*, has a $\lambda$ that takes on a value between 0.0 and 0.6. I show the performance of meta-reinforcement-learning algorithms from both algorithm sets in Figure 3.8. For the plot, I used 200 MDPs to approximate the full distribution and 50 learning episodes with a maximum of 1000 steps.

Algorithm 2 has higher performance in the meta-reinforcement-learning setting, but how do the two sets compare in the context of the sample-optimized

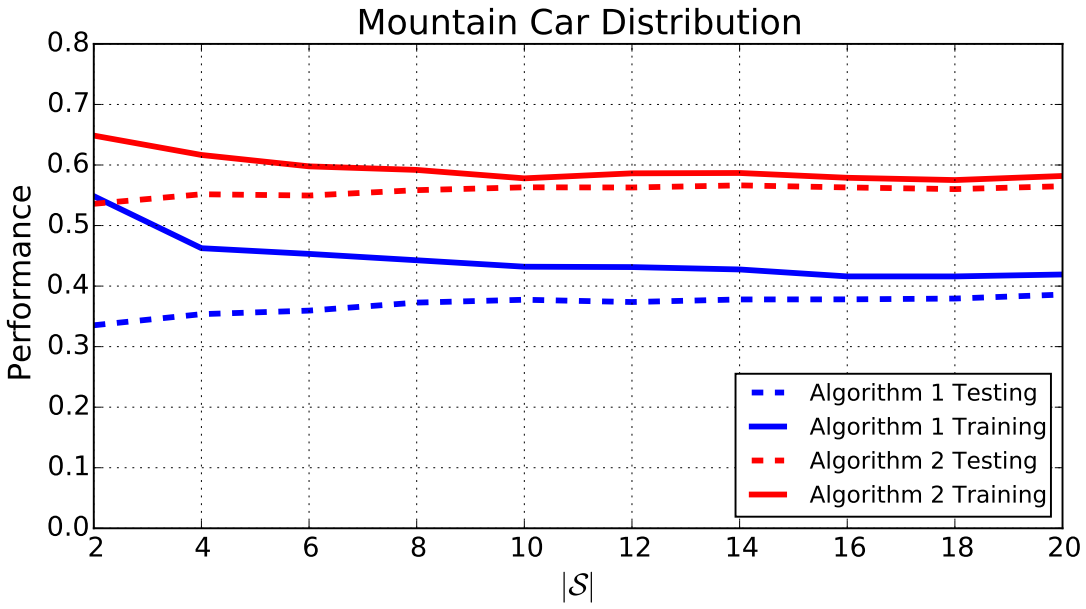


Figure 3.8: Meta-reinforcement learning performance in the Mountain Car Distribution

Rademacher bound? Figure 3.9 shows that, with lower optimization power, $k = 10$, the Algorithm 1 and Algorithm 2 sets have similar training performance, but Algorithm 1 has lower testing performance. Initially, the sample-optimized bounds indicate that Algorithm 2 has a lower performance on the full distribution, but, very quickly, at $|\mathcal{S}| = 8$, we see that Algorithm 1 switches to have the worse lower bound. Here, the complexity of the Algorithm 2 set is higher and as such the bound is more pessimistic. With an increasing number of MDPs, the graph communicates that the Algorithm 2 set is better algorithm to deploy. With $k = 300$, the Algorithm 1 set clearly has a higher training performance but at the same time its sample-optimized Rademacher complexity is high, giving rise to a worse lower bound at $|\mathcal{S}| = 2$. Again, with increasing MDP samples ($|\mathcal{S}| \geq 4$), it becomes clear that the Algorithm 1 set has the better lower bound.

In both the 5-state chain and Mountain Car domain, the sample-optimized

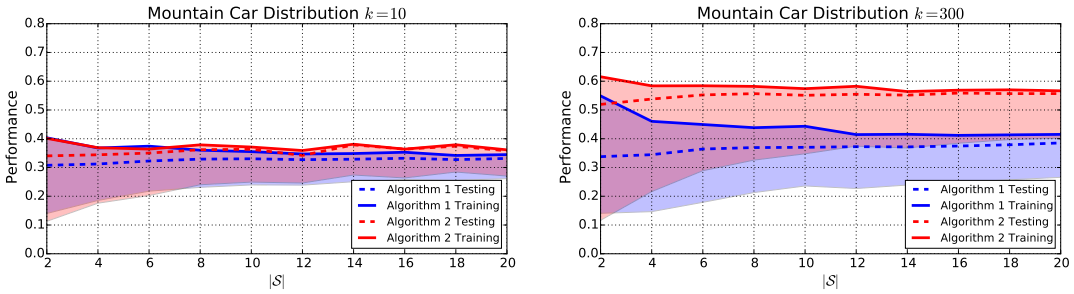


Figure 3.9: Sample-optimized Rademacher bounds for the Mountain Car MDP distribution at different values of $k$

Rademacher bounds do help us identity overfitting of algorithms and give us a method to choose which algorithm set to use given the environment sample $\mathcal{S}$. One practical problem is that the bounds are very conservative (pessimistic), producing very wide bounds for small samples. In the next subsection, I look at using cross-validation as a way to produce tighter bounds with smaller sets of MDP samples.

### 3.3.3 Cross-validation

Borrowing another technique from supervised learning, I next look at cross-validation. Cross-validation is an often-used technique to assess how well an algorithm or function generalizes (Kohavi, 1995). In cross-validation, given a set of samples $\mathcal{S}$ of size $m$, I partition the data into a training set and a held out set. This procedure should not be confused with the earlier setup of using all $m$ samples for training and then attempting to predict the performance in the full distribution $D$. I use the training sets to find the best algorithm given $F_k$ functions and then test this algorithm on the held out set. For the experiments, I use repeated random partitioning (random sub-sampling cross-validation (Kohavi, 1995) with random partition sizes) of the $m$ samples to get a sense of how well an algorithm generalizes. For all of the experiments below, the cross-validation

sampling was repeated 50 times while the partitioning was repeated 100 times. I also indicate the size ($k$) of the mixture ensemble in any of the experiments I highlight.

**Cross-validation experiments**

Returning to the 5-state chain, I now plot the results of the cross-validation performance of both the Static and Dynamic Order 5-state chain vs. the size of the environment samples $|\mathcal{S}|$ (Figure 3.10). The bottom of the shaded area in the graphs is the expected test performance given the random partitioning of the environment samples. We observe that the shape of both the graphs are similar to those of the sample-optimized Rademacher bounds. For the Dynamic Order 5-state chain, the variable initialization algorithm has a large generalization gap at small training set size but this gap is reduced as more samples are available for training. An important difference between the sample-optimized Rademacher bound and the cross-validation bound is that, with a small training set size, $|\mathcal{S}|$, we can already see where we can make decisions about which algorithm set to pick. This property is very evident in the Dynamic Order distribution with $k = 750$ (bottom right of Figure 3.10). In contrast to the sample-optimized Rademacher bounds I showed earlier in Figure 3.6, where we needed $|\mathcal{S}|$ to be close to 200 before we chose the variable initialization over constant initialization at $k = 750$, we need fewer samples using cross-validation ($|\mathcal{S}| = 20$ is sufficient to choose one algorithm set over the other). Secondly, we track the real full distribution testing performance well.

I return to the Mountain Car distribution and illustrate cross-validation in action. The results of Algorithm 1 and Algorithm 2 sets are shown in Figure 3.11. With $k = 10$, we quickly notice that Algorithm 1 tends to overfit

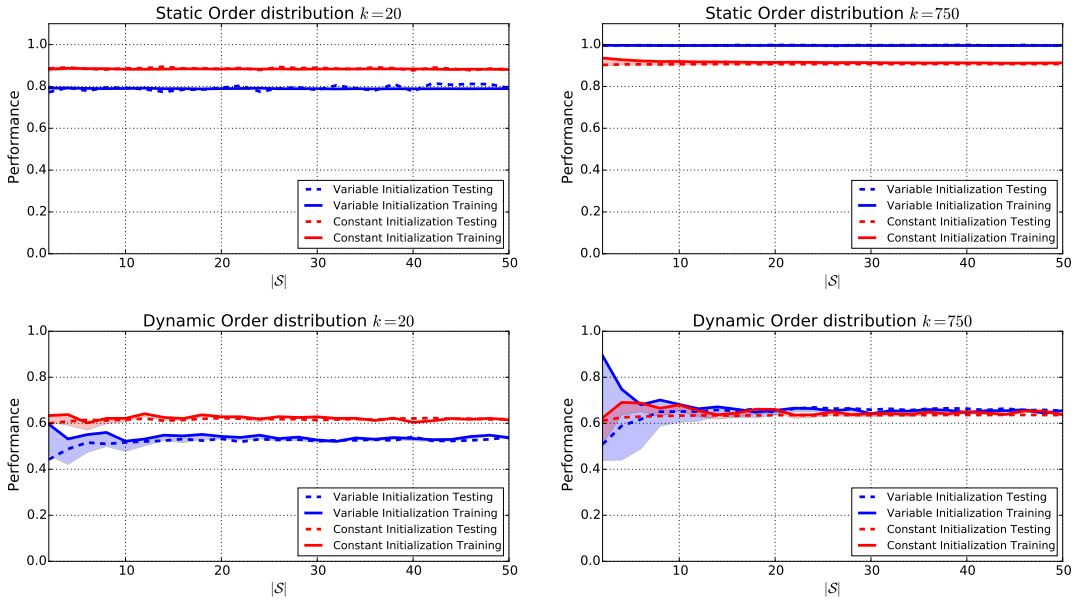


Figure 3.10: 5-State Chain distribution Cross-Validation

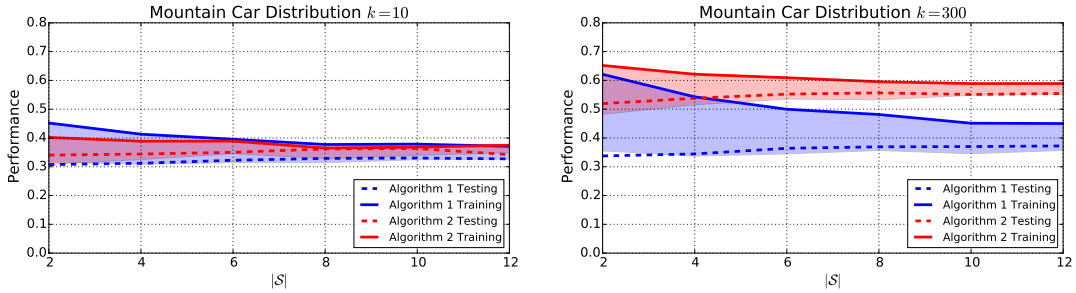


Figure 3.11: Mountain Car distribution Cross-Validation

and Algorithm 2 is the better choice in the worst case. With $k = 300$, it is even clearer that Algorithm 2 is the better choice. The generalization gap in cross-validation is much narrower than in the sample-optimized Rademacher bounds discussed earlier. This applies both to the Mountain Car and 5-state chain domains. We see that cross-validation is a viable approach for evaluating meta-RL algorithms.

So far, I have discussed evaluation of meta-RL algorithms and a mixture ensemble as a potential meta-RL approach. In the next section, I discuss other

approaches to meta-RL and explore a second ensemble-based approach to creating meta-RL algorithms.

## 3.4 Approaches to meta-reinforcement learning

So far in this chapter, I have introduced two ways of creating meta-reinforcement learning algorithms, optimizing over a whole algorithms space or using a mixture ensemble. On a high level, these are potential the ways of creating meta-RL algorithms:

- Searching over the complete parameter spaces of known algorithms. This approach could be prohibitive as it becomes computationally expensive. For algorithm sets that have large search spaces, finding the best algorithm becomes nearly impossible.
- Optimizing over the parameter space of known algorithms (stochastic optimization, multi-armed bandit based approaches). We have a large literature in optimization and multi-armed bandits (Kleinberg et al., 2008) that could be used to create meta-reinforcement learning algorithms.
- Mixture based ensembles with random selection of algorithms.
- Ensembles that learn to select or fuse algorithms.

In this final section, I return to the idea of ensembles, and discuss the use of ensembles in supervised learning and possible approaches to meta-RL algorithms. I introduce an approach to ensemble reinforcement learning using a linear modular learner and show some preliminary results of an application of such a learner.

### 3.4.1 Meta-reinforcement learning using ensembles

Across a wide range of computational domains, ensemble learning methods have proven extremely valuable for reliably tackling complex problems. Ensemble (or sometimes modular or portfolio) methods harness multiple, perhaps quite disparate, algorithms for a problem class to greatly expand the range of specific instances that can be addressed. They have emerged as state-of-the-art approaches for word sense disambiguation (Florian and Yarowsky, 2002), crossword solving (Littman et al., 2002), satisfiability testing (Xu et al., 2010), movie recommendation (Bell et al., 2010) and question answering (Ferrucci et al., 2010).

In the context of ensemble-based learning, there are two main approaches for combining algorithms to make decisions: algorithm selection and algorithm fusion (Polikar, 2006). Both approaches begin with a set of independent *base* algorithms that can be used (with varying degrees of success) to attack the problem at hand. The task of the ensemble learner in algorithm selection, then, is to map inputs to base algorithms. In principle, the selection problem (input to algorithm) could be as hard as the learning problem itself (input to output), limiting its utility. In algorithm fusion, on the other hand, the ensemble learner is responsible for finding a rule for combining the outputs of the base algorithms (in an input-independent way).

The successful applications of ensemble methods listed earlier are all of the fusion type. In these cases, supervised examples are available for directly training the ensemble learner to find an accurate fusion rule. There are multiple frameworks that constitute ensemble classifiers. The frameworks differ by the way low-level classifiers interpret input data, their aggregation schemes and whether the base classifiers are dependent or independent of each other (Rokach,

2010). Training examples are not available to learners in an RL setting, however, necessitating a novel approach if one wants to apply algorithm fusion in RL. In the RL setting, the ensemble learner only has access to the MDP.

Work by Wiering and van Hasselt (2008) and van Hasselt (2011) investigates different methods of fusing multiple RL algorithms. Their base algorithms output action distributions, which are combined using either voting, policy addition or policy multiplication. The authors handle the lack of a supervision signal in the RL setting by adopting a fixed merging method—the ensemble learner does not learn at all. As a result, these methods cannot adapt to focus on the appropriate base algorithms for the task at hand. The authors found that, in most of their test cases, ensemble learners performed second best to the best tuned algorithm. Work by Sun and Peterson (1999) investigated propagating the ensemble learners' Bellman error to lower level function approximators (Neural Networks). The work I describe here does not do that.

In the following subsection, I explore the approach of using temporal difference learning to solve the temporal credit assignment problem for a modular ensemble learner and thus to appropriately weight the contributions from each individual base algorithm. In this scheme, the base algorithms must output action-value estimates (Q values) instead of simply giving action recommendations. Ultimately, in this approach, the modular learner is an RL method that uses the base-level value estimates as its *feature* representation.

### 3.4.2 Temporal difference combination of RL algorithms

In this subsection, I present an approach to ensemble-based RL using a linear Temporal Difference (TD) learning algorithm as a modular learner to combine the value estimates from multiple base RL algorithm algorithms. The approach goes beyond earlier efforts in ensemble RL (Wiering and van Hasselt, 2008) in that I develop a fusion method that is adjusted given the performance of the base algorithms in the ensemble instead of combining low-level algorithms according to a fixed rule. In the ensemble classifier approach, given $n$ classifiers, each classifier $i$ has a prediction $d_{i,j}(\mathbf{x})$ as to whether the data point $\mathbf{x}$ belongs to class $\omega_j$. The final prediction of the modular learner, $\mu_j(\mathbf{x})$, for class $\omega_j$ is $\mu_j(\mathbf{x}) = \sum_{i=1}^{n} w_{i,j} d_{i,j}(\mathbf{x})$. The supervised modular learner uses held out labeled training data to learn the combination weights $w_i$ for each base classifier. Whereas supervised classifiers map instances to labels, in a value-function-based setting, RL algorithms map states and actions to action values (the Q-function).

Using the supervised ensemble weighted learning as a guide, we can develop a parallel approach in which separate RL algorithms (base algorithms) create their own Q-functions. The RL modular learner then estimates the environment's Q-function via a weighted linear combination of the Q-values learned by the base algorithms. The final Q-value given $k$ RL base algorithms is

$$Q_{\mathbf{W}}(s,a) = \sum_{i=1}^{k} w_i Q_i(s,a),$$

where $w_i$ are the weights and $Q_i(s,a)$ is the estimated Q-value of state $s$ and action $a$ for RL base algorithm $i$. The RL modular learner learns the weights $w_i$ for each base algorithm. Given that labeled examples are not available in the RL setting, another error metric needs to be used. In TD-based algorithms (Sutton,

1988), the natural error metric is the Bellman error (discussed further in Chapter 4),

$$E_{RL,\mathbf{w}} = \sum_t [R(s,a) + \gamma Q_{\mathbf{W}}(s',a') - Q_{\mathbf{W}}(s,a)]^2. \tag{3.7}$$

An advantage of this error metric is that it does not require labeled examples. This formulation, arrived at by translating standard linear ensemble methods to the RL setting, is an exact match for the problem solved by linear TD methods. The twist is that the role of "state" in this formulation is the Q-value estimates produced by the base algorithms. With that substitution in place, any existing TD method can be applied to learn weights for the modular learner.

Given that both the base algorithms and the modular learner need to adapt, I run learning in two stages. First, the base algorithms are trained on the environment in question by themselves, then they are frozen and then the modular learner adapts its weights to combine the Q-values of the base algorithms. I have experimented with adapting the modular learner and base algorithms simultaneously, but the results were less than stable.

As the modular learner searches for the best linear combination of the base algorithm Q-values, using them as features, convergence guarantees are similar to those of other linear TD learning algorithms (Tsitsiklis and Van Roy, 1997). With the above description of the combination of base algorithms, we can view each base algorithm's Q-value as a feature for the modular learner that is dependent on a state and action pair, $(s,a)$. The two-stage modular learner is a least squares algorithm (Boyan, 2002) that minimizes Equation 3.7 and thus converges to the weights that results in the smallest error between the estimates and the real returns.

**Ensemble approach to solving MDPs**

To assess the fusion ensemble approach in an RL setting, I carried out the evaluation on the generalized MDPs framework (Whiteson et al., 2009) discussed earlier in Section 3.1.1. In this case I use a set of of $|\mathcal{S}|$ training MDPs to find a set of base algorithms to include in the modular learner. I then evaluate the performance of this modular learner on an unseen test set.

The environment in the modular-learner experiments is the generalized Mountain-Car environment taken from the 2008 RL competition (Whiteson et al., 2010). The mountain-car environment was generalized by adjusting observations from the environment (noisy observations), as well the outcomes of the actions taken (stochastic actions). This version of the Mountain Car environment is slightly different to the one earlier used in the chapter in that the observations (state variables) are noisy.

In this fusion ensemble approach, I create a diverse set of algorithms by taking a set of $|\mathcal{S}|$ training MDPs to identify a promising set of base algorithms (algorithms that perform well in some of the training MDPs). This set of algorithms would then be used in the modular learner to tackle the test MDPs. Thus as input the ensemble algorithm receives training MDPs and as output is a modular RL learner comprised of a set of $k$ base algorithms that will be combined.

Using this approach, I had access to 10 training MDPs. For each of the training MDPs, 10 Q-learning algorithms with different parameters were trained and evaluated. The parameters of the top learner from each of the MDPs were

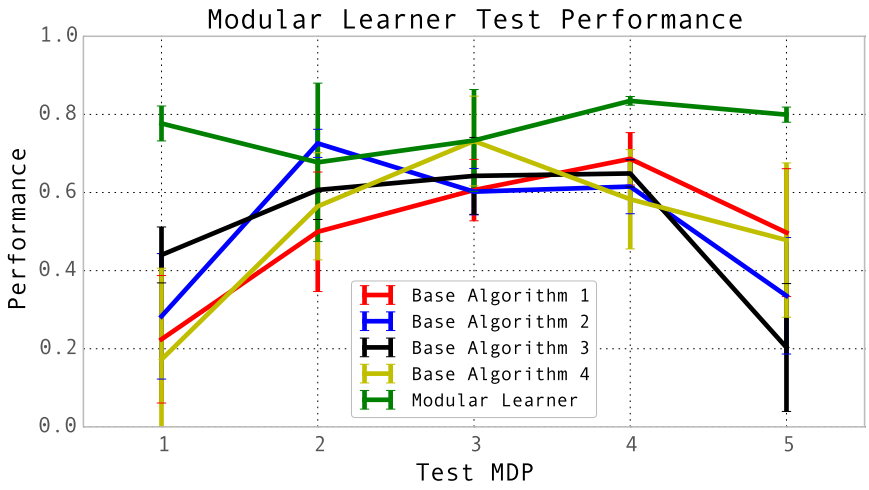


Figure 3.12: A modular-learner that combines the results of multiple RL algorithms achieves higher reward across a collection of environments than individual RL algorithms.

compared against each other for diversity. Some of the parameters were equivalent and thus only 4 parameters were used for the base algorithms of the modular learner. The candidate parameter comparison for diversity can be accomplished by a clustering algorithm but in this case was done manually. The RL base algorithms created using the 4 parameters were also evaluated individually on the test MDPs.

For evaluation, the algorithms (4 individual Q-Learning base algorithms) and the modular learner (Least Squares Temporal Difference Learning) had 1000 training episodes on each test MDP. The modular learner equally allocated episodes of interaction between itself and each of its base algorithms. Evaluation was on the last 50 learning episodes. The modular learner's exploration rate ($\varepsilon$) was fixed at 0.05 for all tests. The accumulated rewards for each test MDP configuration were averaged over 5 runs and are shown in Figure 3.12.

In the 5 test MDPs, base algorithm 1 had the highest score among the base algorithms 2 times, base algorithm 2 was highest 1 time, base algorithm 3 was

highest 1 time and base algorithm 4 was highest 1 time, The modular learner is statistically indistinguishable from the best in all 5 test MDPs. In addition, in Test MDP 1, the modular learner's policy is significantly better than any of the base algorithms. If we had used only one set of parameters to learn on the generalized test MDPs, we would not have gotten as good performance as the ensemble approach.

To sum up, using a TD algorithm, the high-level modular learner is able to identify how to weigh and combine low-level (base) algorithms so as to obtain high returns. Since the modular learner weighs and combines Q values, other algorithms for the base algorithms can be added/substituted as long as they estimate Q values. Model-based RL algorithms such as RMax (Brafman and Tennenholtz, 2003) could be substituted for the base algorithms as they compute Q-functions indirectly. The modular learner's TD algorithm can also be substituted with other more efficient algorithms. For example, we could have the modular learner be a selective learner by using LARS-TD (Kolter and Ng, 2009). This ensemble approach still needs more study. The effects of varying parameters, such as the discount factor, are still not clear and could provide better insight in diversity creation for the ensemble RL setting. Furthermore, investigating how the size of ensemble can be dynamically changed—removing or adding base algorithms—is another avenue worth investigating. These promising initial results indicate that we can create meta-reinforcement-learning algorithms that can adapt over different MDPs.

## 3.5 Summary

In this chapter, I presented improved ways to evaluate reinforcement-learning algorithms online. By extending the reinforcement-learning problem to deal

with learning and evaluation in a distribution of MDPs, we gain better insight into the behavior of algorithms in relation to the distributions of MDPs they would be deployed in. Importantly, we also reduce the chances that the algorithms we create will overfit to single MDPs.

By using the sample-optimized Rademacher bound, we can take a number of sample MDPs and better make decisions about how well an algorithm set would perform in the full distribution where the sample was drawn from. I illustrated the application of cross-validation to the meta-reinforcement learning problem. Multiple MDP-based evaluation is important, not just for seasoned reinforcement-learning researchers, but also for novices who may not understand the power of the numerous reinforcement-learning algorithms available. Lastly, I discussed possible ways in which meta-reinforcement-learning algorithms could be created. I also presented an example of the use of a fusion ensemble reinforcement-learning algorithm in tackling MDPs that are drawn from a distribution.

# Chapter 4

# Offline Evaluation of Value-Based Reinforcement-Learning Algorithms

## 4.1 Introduction

To assess the effectiveness of reinforcement-learning algorithms, it is important to have a way to compare pairs of algorithms head to head. The gold standard evaluation is to measure the return of each algorithm online (Kaelbling et al., 1996), declaring the algorithm that produced the highest return the winner.

While online evaluation is, in some sense, the only true measure of an algorithm's performance, there are many reasons to desire an offline evaluation metric using a fixed set of pre-collected data:

- If it is not possible to interact with the target environment and no veridical simulator is available, offline evaluation is the only option.
- Even if a good simulator is available, if it is complex, time consuming, or expensive to use, offline evaluation might be preferred.
- Even if a simulator is available and easy to share, offline evaluation can be important for making the results more comparable. (Many research groups have rewritten classic RL environments like "mountain car" due to language incompatibilities or other constraints.)
- When a target environment is a human being (a patient with epilepsy or a

> grandmaster chess opponent, say) there are practical, and sometimes even ethical, concerns about testing learning results online. Offline evaluation can make it possible to collect the data once under controlled conditions and then share it with researchers throughout the community.

Evaluating learned classifiers offline using labeled batch data is standard practice in the supervised-learning community. The existence of objective evaluation metrics that can be applied directly using batch data has enabled the creation and comparison of multitudes of algorithms and dramatic improvements in performance over the years. The introduction of the UCI Machine Learning data repository (Newman et al., 1998) transformed the way machine-learning (ML) research is conducted and led to the development of multiple evaluation metrics as well as inspired other ML sub-communities to develop standards to collect data and evaluate performance (Bay et al., 2000).

An even more significant use of the UCI database is its availability of "Natural Domains" (Langley and Kibler, 1991) of classification and regression problems. The UCI database allows researchers to compare their methods using different metrics on real world problems. The success of algorithms on these Natural Domains provides some evidence on the general applicability of those algorithms. The explosion of data collected from domains such as education (Siemens and Long, 2011), healthcare (Murdoch and Detsky, 2013) and marketing (Mayer-Schönberger and Cukier, 2013) has resulted in large datasets that are sequential in nature. These datasets may not necessarily be used in a sequential manner currently, but researchers might gain more insight and knowledge were they to have tools to better apply and use RL approaches with such data. As such, a question I would like to assist in answering is how do we evaluate and compare reinforcement-learning algorithms when we only have access to batch data?

### 4.1.1 Related Work

Existing approaches for offline evaluation for RL have taken a number of forms. Nouri et al. (2009) presented an approach for evaluating value functions for a fixed policy. Although they provided preliminary results suggesting that algorithms that are effective for learning the value function for a fixed policy can also be used to learn to maximize reward, the emphasis on fixed policies instead of return-maximizing policies limits the attractiveness of this approach.

An ideal offline evaluation metric evaluates policies, but such metrics appear to require strong assumptions. Fonteneau et al. (2010) presented an evaluation method that averages the return on a trajectory constructed by selecting actions using the policy and then finding the closest one-step transitions from batch-collected transition data. It requires an assumption that "nearest neighbor" modeling produces accurate trajectories, which it will not in general. Li et al. (2011) and Dudik et al. (2011) provided evaluation algorithms that apply only in bandit domains (roughly MDPs where state transitions do not depend on the current action or state), not general environments. However, in the bandit setting, they are quite robust, working directly from batch data without the need for a simulator or a model of the reward function.

The work I present in this chapter differs from that of Fonteneau et al. (2010) in that I develop an evaluation metric that uses the collected batch data directly and not as a proxy to reconstruct the likely trajectories that would have been produced by the resultant policy. This work also differs from that of Nouri et al. (2009) in that I evaluate value functions with respect to the optimal value function, not an arbitrary fixed policy.

### 4.1.2 Outline

This chapter presents a first approach to evaluating reinforcement-learning algorithms offline, that is, using sequential batch data. I first survey different possible metrics that could be used for comparing value-based RL algorithms against each other. I then present a novel metric, the *Relative Bellman Update Error*, and discuss its properties. I then demonstrate its use in comparing the performance of value functions learned in several benchmark environments. Finally, I discuss the merits of the different metrics tested and then summarize the chapter.

## 4.2 Survey of evaluating state-action value functions

In this chapter I use the definitions from Chapter 2: $S$ as the state space, $A$ as the action space, $T : S \times A \times S' \to [0, 1]$ as the transition probabilities between states as actions are taken, and the reward function $R : S \times A \to \Re$. To assist in navigating different approaches to metrics for reinforcement-learning algorithms, I examine some additional tools. Given an experience tuple $\langle s, a, r, s' \rangle$, I define the *sample Bellman backup operator* $B^{s,a}_{r,s'}$ as a mapping from state-action value function $Q$ to a new state-action value function that is identical to $Q$ except

$$(B^{s,a}_{r,s'}Q)(s, a) = r + \gamma \max_{a'} Q(s', a'). \tag{4.1}$$

Using this notation, the standard Bellman backup operator $B$ can be written as $(BQ)(s, a) = \{(B^{s,a}_{r,s'}Q)(s, a)\}^{s',r}_{T,R}$. The notation $\{\cdot\}^{s',r}_{T,R}$ denotes that we are taking an expected value of the expression $\cdot$ with $s' \sim T(s, a, s')$ and $r = R(s, a)$. The Q-learning algorithm (Watkins and Dayan, 1992) can be written $Q \leftarrow_\alpha B^{s,a}_{r,s'}Q$, where $\leftarrow_\alpha$ is the learning update as discussed earlier in equation 2.3. We seek an

evaluation metric that (1) is easy to calculate given batch data and (2) enables us to compare the performance of one state-action value function to another. That is, given a set of experience tuples of the form $\langle s, a, r, s' \rangle$ and a pair of state-action value functions $Q_1$ and $Q_2$, we want an evaluation of which value function is superior.

In this section, I survey some ways of assessing state-action value functions. For consistency, I define each metric $M(Q_1, Q_2)$ so that it returns a positive value if $Q_1$ is believed to be superior, a negative value if $Q_2$ is believed to be superior, and zero if they are judged as being equally good.

### 4.2.1 Expected Return (online)

The most direct score for comparing state-action value functions is the expected return, the discounted sum of rewards along an episode following the greedy policy $\pi_Q(s) = \text{argmax}_a Q(s, a)$. Even though it is an online metric, I use it as the gold standard to compare other metrics. The greedy policy is applied on the real environment or simulator and the collected rewards are summed. That is,

$$\text{return}(Q) = E\left[\sum_{n=0}^{N-1} \gamma^n R(s^n, a^n)\right],$$

where $R(s^n, a^n)$ is the reward at timestep $n$ for being in state $s^n$ and performing action $a^n$ where $a_n = \pi_Q(s_n)$ and $s_{n+1} \sim T(s_n, a_n, s_{n+1})$. Here, $N$ is the maximum number of timesteps or the trajectory length. The return metric is

$$M_{\text{return}}(Q_1, Q_2) = \text{return}(Q_1) - \text{return}(Q_2).$$

### 4.2.2 Model Free Monte Carlo policy evaluation

Fonteneau et al. (2010) developed *Model Free Monte Carlo-like policy evaluation* (MFMC). It is designed to evaluate a policy given a set of experience tuples, but we can use it to evaluate a value function $Q$ by considering its greedy policy $\pi_Q(s) = \text{argmax}_a Q(s,a)$. The output is the return based on trajectories sampled using the policy and an approximate model of the transition dynamics. In particular, next states are chosen by finding the closest one-step transitions in the set of experience tuples. Given start state $s_0$ and policy $\pi$, the expected return calculated by MFMC is

$$\text{mfmc(Q)} = \mathbb{R}_p(s_0, Q) = \frac{1}{p}\sum_{i=1}^{p}\sum_{n=0}^{N-1}\gamma^n r^{l_n^i},$$

where $p$ is the number of trajectories used in the simulation, $N$ is the trajectory length, and $r^{l_t^i}$ is the reward "experienced" from the batch transition sample $l_n^i$ (defined next). The batch transition sample is selected by finding the closest unused sample using a distance function $l_n^i = \text{argmin}_{(\tilde{s},\tilde{a},r,\tilde{s}')\in U}\Delta((s_n^i, a_n^i),(\tilde{s},\tilde{a}))$, where $a_n^i = \text{argmax}_a Q(s_n^i, a)$, $s_0^i = s_0$, $s_{n+1}^i = \tilde{s}'^{l_n^i}$ and $U$ is the set of unused samples. This evaluation metric does not require a simulation model but, in essence, empirically estimates a model using the batch of experience tuples and its predefined similarity metric.

A positive attribute of this metric is that it produces values that are intended to match the return of a given policy or value function. In the limit of infinite data, the values produced are actually equivalent to the gold standard of expected return from a simulator. In practice, however, a negative attribute of this metric is that it is itself a learning algorithm. In fact, it is quite close to kernel-based learning methods (Ormoneit and Śaunak Sen, 2002; Jong and

Stone, 2007). As such, the evaluation metric is apt to produce wildly skewed results in favor of learning algorithms that make similar assumptions about state similarities.

I define the metric based on MFMC as

$$M_{\text{MFMC}}(Q_1, Q_2) = \text{mfmc}(Q_1) - \text{mfmc}(Q_2).$$

### 4.2.3 Distance from optimal values

The state-action value function $Q^*$ is the solution to the equation $Q^* = BQ^*$. The greedy policy with respect to $Q^*$ maximizes expected return. As such, it is natural to assess a state-action value function $Q$ by its distance from $Q^*$. Using $||Q|| = \max_{s,a} |Q(s,a)|$ as the max norm of $Q$, we can evaluate $Q$ via $||Q^* - Q||$.

Singh and Yee (1994) relate the quantity $||Q^* - Q||$ to the difference in return between following $Q$'s greedy policy and following $Q^*$'s. Informally, if $||Q^* - Q|| = \delta$, that means that the action selected greedily at each time step has a value that is no more than $2\delta$ suboptimal. If this loss is repeated at every step, the total loss will be $2\delta/(1-\gamma)$.

Note that if $Q = Q^* + K$ for a large constant $K$, $||Q^* - Q||$ will be large ($K$, in fact), but $Q$'s greedy policy is identical to $Q^*$'s. A bad score by this metric does not imply a bad policy. However, it does imply a worse *bound* on how bad the policy might be.

I define the metric based on the distance from $Q^*$ to be

$$M_{\text{dist}}(Q_1, Q_2) = ||Q^* - Q_2|| - ||Q^* - Q_1||.$$

A positive property of this metric is that $M_{\text{dist}}(Q^*, Q)$ is non-negative for all $Q$. That is, no state-action value function is judged superior to $Q^*$. Its largest

drawback is that it cannot be used unless $Q^*$ is known, which will only be true for the most basic benchmark problems.

### 4.2.4 Bellman Residual

The Bellman backup of $Q$ is $Q' = BQ$. As mentioned above, the optimal value function is obtained when $Q = Q'$, suggesting that the distance between these quantities $||Q' - Q||$, sometimes called the *Bellman residual*, is another useful way of evaluating $Q$.

Porteus (1982) provides an analysis that can be used to relate $||Q' - Q||$ to $||Q^* - Q||$ and therefore to the difference in expected return between following $Q$'s greedy policy and following $Q^*$'s. Informally, if $||Q' - Q|| = \kappa$,

$$\begin{aligned} \delta &= ||Q^* - Q|| \\ &\leq ||Q^* - Q'|| + ||Q' - Q|| \\ &\leq \gamma||Q^* - Q|| + ||Q' - Q|| \\ &\leq \gamma\delta + \kappa, \end{aligned}$$

from which we derive that $\delta \leq \kappa/(1-\gamma)$. A high value of the Bellman residual does not imply a poor value function. However, a value function with a lower Bellman residual has a tighter bound on its suboptimality.

I define the metric based on the Bellman residual to be

$$M_{\text{residual}}(Q_1, Q_2) = \|BQ_2 - Q_2\| - \|BQ_1 - Q_1\|.$$

Once again, this metric also has the property that $M_{\text{residual}}(Q^*, Q)$ is non-negative for all $Q$. No state-action value function is judged superior to $Q^*$. Its largest drawback is that it cannot be used unless the transition function is known (or is very densely sampled), because the transition function is needed for computing the Bellman backup.

### 4.2.5 Bellman Update Error

Consider again the Bellman residual and why it cannot be used without a model. For a value function $Q$, we would like to evaluate $||Q - BQ||$ where $BQ$ is the Bellman backup applied to $Q$. That is, we want $Q(s,a)$ to be as close as possible to the average (over next states) value of $(B^{s,a}_{r,s'}Q)(s,a)$.

There is a closely related problem that we can use as an analogy. Imagine we have some distribution $D$ and we are sampling values $x$ from $D$. If we want to encourage the selection of a value of $m$ that is close to the expectation of $x$, we can employ the squared error measure $\{(m-x)^2\}^x_D$ because setting $m$ to be the mean of $D$ minimizes this quantity.

Moving this idea to the value-function context suggests the following error measure I call *Bellman update error* (BUE):

$$\text{BUE}(Q) = \{\{(Q(s,a) - (B^{s,a}_{r,s'}Q)(s,a))^2\}^{s',r}_{T,R}\}^{s,a}_{\Pi}. \tag{4.2}$$

Here, state-action pairs are sampled from some probability distribution $\Pi$ and next states are sampled from the transition function.

The idea of Bellman update error is natural—it can be thought of as the Q-learning rule reconceptualized as an error measure—and has been used (explicitly or implicitly) repeatedly in the RL literature. For example, it justifies the use of a mean squared best fit when function approximation is used (Boyan and Moore, 1995). However, the Bellman update error is known to have some serious problems (Baird, 1995; Sutton and Barto, 1998).

To better understand the error measure and its problems, it can be rewritten it as follows. First, define $Q' = BQ$. Using the fact that

$$\{(B^{s,a}_{r,s'}Q)(s,a)\}^{s',r}_{T,R} = Q'(s,a)$$

and

$$\mathrm{Var}[(B^{s,a}_{r,s'}Q)(s,a)] = \{((B^{s,a}_{r,s'}Q)(s,a))^2\}^{s',r}_{T,R} - Q'(s,a)^2,$$

it is straightforward to rewrite $\mathrm{BUE}(Q)$ as

$$\begin{aligned}
&\{\{(Q(s,a) - (B^{s,a}_{r,s'}Q)(s,a))^2\}^{s',r}_{T,R}\}^{s,a}_{\Pi} \\
&\quad = \{(Q(s,a)^2 - 2Q(s,a)Q'(s,a) + \{(B^{s,a}_{r,s'}Q)(s,a)^2\}^{s',r}_{T,R})\}^{s,a}_{\Pi} \\
&\quad = \{(Q(s,a)^2 - 2Q(s,a)Q'(s,a) + Q'(s,a)^2 + \mathrm{Var}[(B^{s,a}_{r,s'}Q)(s,a)])\}^{s,a}_{\Pi} \\
&\quad = \{(Q(s,a) - Q'(s,a))^2\}^{s,a}_{\Pi} + \{\mathrm{Var}[(B^{s,a}_{r,s'}Q)(s,a)]\}^{s,a}_{\Pi}.
\end{aligned}$$

That is, the Bellman update error is the squared Bellman residual *plus* the variance of the Bellman backup. It is worth noting that the variance term disappears if the environment is deterministic, in which case BUE becomes the mean squared Bellman residual.

In stochastic environments, however, the variance term can have a significant impact on the metric's value. When choosing a value function to minimize BUE, the appearance of $Q$ in two places in the expression creates a tension between selecting $Q$ to match the Bellman residual and selecting $Q$ to minimize the variance term. As a result, it can be the case that the optimal value function $Q^*$ is *not* the value function that minimizes BUE. I show an example of this situation in Section 4.5.

I define the metric based on BUE as

$$M_{\mathrm{BUE}}(Q_1, Q_2) = \mathrm{BUE}(Q_2) - \mathrm{BUE}(Q_1).$$

To summarize, the BUE metric has the positive attribute that it can be estimated without knowing $Q^*$ or $T$. Further, it can be estimated from sampled data as values are combined across state-action pairs sampled from a distribution $\Pi$. If test data is drawn from this distribution, approximating this average is

straightforward. Its negative attribute is that, in stochastic environments, poor value functions can be rated as superior to the optimal value function because of the errant variance term in its formulation.

## 4.3 Relative Bellman Update Error

I propose the following novel metric for comparing two state-action value functions $Q_1$ and $Q_2$. Like BUE, it can be estimated from a sample of experience tuples. But, it has some important properties that set it apart from BUE. For example, it can produce a bound on the suboptimality of return, and it correctly selects $Q^*$ as the optimal value function given the right sampling distribution $\Pi$.

The basic idea is to consider a Bellman update on the *average* of $Q_1$ and $Q_2$ instead of one or the other. For state-action value function $Q_i$ ($i$ is 1 or 2), we have

$$\text{RBUE}_i^{s,a}(Q_1, Q_2) = \{(Q_i(s,a) - (B_{r,s'}^{s,a}(Q_1 + Q_2))(s,a)/2)^2\}_{T,R}^{s',r}.$$

These quantities are then combined into a metric via

$$M_{\text{RBUE}}(Q_1, Q_2) = \{\text{RBUE}_2^{s,a}(Q_1, Q_2) - \text{RBUE}_1^{s,a}(Q_1, Q_2)\}_{\Pi}^{s,a}.$$

I provide proofs of RBUE's properties next.

## 4.4 Formal evaluation of Relative Bellman Update Error

In this section, I analyze the formal properties of the Relative Bellman Update Error metric.

### 4.4.1 No variance term

Using $\bar{Q} = (Q_1 + Q_2)/2$, $\bar{Q}' = B\bar{Q}$, and following a similar derivation to the one used in Section 4.2.5,

$$\begin{aligned} M_{\text{RBUE}}(Q_1, Q_2) &= \{((Q_2(s,a) - \bar{Q}'(s,a))^2 + \text{Var}[(B^{s,a}_{r,s'}\bar{Q})(s,a)]\}^{s,a}_{\Pi} \\ &\quad -\{((Q_1(s,a) - \bar{Q}'(s,a))^2 + \text{Var}[(B^{s,a}_{r,s'}\bar{Q})(s,a)]\}^{s,a}_{\Pi} \\ &= \{(Q_2(s,a) - \bar{Q}'(s,a))^2 - (Q_1(s,a) - \bar{Q}'(s,a))^2\}^{s,a}_{\Pi}. \end{aligned}$$

The important thing to note is that the variance terms cancel, leaving only the difference of the squares of the distance from $Q_2$ and $Q_1$ to the Bellman backup of $\bar{Q}$.

This approach to getting rid of the variance term is similar to the policy-evaluation loss function introduced by Antos et al. (2008). Their empirical policy-evaluation procedure takes in a policy $\pi$ and a sample trajectory and returns a state-action value function that approximates the value of the given policy. The policy-specific state-action value function is constructed by minimizing a metric similar to $M_{\text{RBUE}}$.

### 4.4.2 Optimal values

A significant problem with the more direct metric $M_{\text{BUE}}$ is that it can judge the optimal value function $Q^*$ as being inferior to a very poor value function. In this section, I analyze a variant of $M_{\text{RBUE}}$ that uses the max-norm instead of $\{\cdot\}^{s,a}_{\Pi}$. The same result does not hold for $\{\cdot\}^{s,a}_{\Pi}$ with general distributions $\Pi$, but I conjecture that it holds when $\Pi$ is the stationary distribution of the optimal policy.

The max-norm is defined as $\|Q\| = \max_{s,a} |Q(s,a)|$. In max-norm form,

$$M^{\infty}_{\text{RBUE}}(Q_1, Q_2) = \|Q_2 - \bar{Q}'\| - \|Q_1 - \bar{Q}'\|,$$

where $\bar{Q}' = B(Q_1 + Q_2)/2$ for Bellman update $B$.

**Theorem 4.4.1.** *The metric $M^{\infty}_{RBUE}(Q^*, Q)$ is non-negative for all $Q$.*

*Proof.* Let $\delta = \|Q^* - Q\|$. I claim that it follows that $\|Q^* - \bar{Q}\| = \delta/2$. To see why, consider any $(s, a)$ and observe that $Q^*(s,a) - \bar{Q}(s,a) = Q^*(s,a) - (Q^*(s,a) + Q(s,a))/2 = (Q^*(s,a) - Q(s,a))/2$. Since this equality holds for every $(s, a)$, it is true for the $(s, a)$ for which $|Q^*(s,a) - Q(s,a)|$ is maximized.

It follows from standard contraction properties of $B$ (Williams and Baird, 1993) that $\|Q^* - \bar{Q}\| = \delta/2$ implies that $\|Q^* - B\bar{Q}\| \leq \gamma\delta/2$. Given that $\delta = \|Q^* - Q\| \leq \|Q^* - B\bar{Q}\| + \|B\bar{Q} - Q\|$ by the triangle inequality, we have

$$\begin{aligned}
\|Q^* - B\bar{Q}\| &\leq \gamma\delta/2 \\
\|Q^* - B\bar{Q}\| &< \delta/2 \\
2\|Q^* - B\bar{Q}\| &< \delta \\
\|Q^* - B\bar{Q}\| &< \delta - \|Q^* - B\bar{Q}\| \\
\|Q^* - B\bar{Q}\| &< \|Q^* - B\bar{Q}\| + \|B\bar{Q} - Q\| \\
&\quad - \|Q^* - B\bar{Q}\| \\
\|Q^* - B\bar{Q}\| &< \|B\bar{Q} - Q\|.
\end{aligned}$$

□

### 4.4.3 Loss bounds for RBUE

For the max-norm variant $M^{\infty}_{\text{RBUE}}$, I prove a relation between the value it produces and the suboptimality of the value function.

Consider two state-action value functions, $Q_1$ and $Q_2$. Let $\Delta$ be a bound on how close they are to each other: $\|Q_1 - Q_2\| \leq \Delta$. The closer they are, the more reliable the relative bounds will be. Let $\delta_1 = \|Q_1 - Q^*\|$ and $\delta_2 = \|Q_2 - Q^*\|$.

These are the quantities we are interested in studying. Let $\bar{Q} = (Q_1 + Q_2)/2$ and $\bar{Q}' = B\bar{Q}$ for Bellman operator $B$. By the triangle inequality, we have $\|Q_1 - Q^*\| \leq \|Q_1 - \bar{Q}'\| + \|\bar{Q}' - Q^*\|$.

Let $\|Q_1 - \bar{Q}'\| \leq \epsilon_1$ and $\|Q_2 - \bar{Q}'\| \leq \epsilon_2$ bound the relative Bellman residuals. We can also bound

$$\begin{aligned}
\|\bar{Q}' - Q^*\| &= \|B\bar{Q} - Q^*\| \\
&\leq \gamma\|\bar{Q} - Q^*\| \\
&\leq \gamma\|(Q_1 + Q_2)/2 - Q^*\| \\
&\leq \gamma\|(Q_1 + (Q_1 + \Delta))/2 - Q^*\| \\
&\leq \gamma\|Q_1 - Q^*\| + \gamma\Delta/2 \\
&\leq \gamma\delta_1 + \gamma\Delta/2.
\end{aligned}$$

Putting these pieces together, we have that

$$\begin{aligned}
\|Q_1 - Q^*\| &\leq \|Q_1 - \bar{Q}'\| + \|\bar{Q}' - Q^*\| \\
\delta_1 &\leq \epsilon_1 + \gamma\delta_1 + \gamma\frac{\Delta}{2} \\
\delta_1 &\leq \frac{\epsilon_1}{(1-\gamma)} + \gamma\frac{\Delta}{2(1-\gamma)}.
\end{aligned}$$

That is, the distance between $Q_1$ and optimal can be bounded as a function of the values computed as part of the calculation of $M^{\infty}_{\text{RBUE}}$. That is, we can keep track of the largest difference between two functions $\Delta$ and the largest difference between one function $Q_1$ and $\bar{Q}'$.

### 4.4.4 Estimating RBUE via samples

As the main goal is to develop a metric that allows for offline evaluation on a wide range of environments, I focus on the use of samples instead of exhaustive

comparisons of value functions. In particular, although $M^{\infty}_{\text{RBUE}}$ is better understood at present, it cannot be computed using a sample. In contrast, $M_{\text{RBUE}}$ can be estimated well by sampling experience from distribution $\Pi$. The expectation of such an estimate matches that of the $M_{\text{RBUE}}$ metric—future work should quantify the variance introduced by finite sample effects.

There are numerous methods by which samples could be collected. It is important to adequately explore the parts of the state-action space relevant to optimal or near-optimal policies. Of course, there are environments in which undirected data collection is not possible as it may lead to undesirable effects. For example, in a flying helicopter environment, thorough exploration could lead to pricey crashes. Similarly, in a environment for learning to play chess against a grandmaster, the grandmaster would be unable to play millions of games to attempt to provide coverage of likely outcomes. Instead, a reasonable scheme is to collect trajectories using policies that might be suboptimal in practice but provide samples from important parts of the state-action space.

## 4.5 Empirical evaluation of offline evaluation metrics

To assess the value of the different offline evaluation metrics, I carried out comparison experiments using a number of different reinforcement-learning environments. The metrics used for evaluations were $M_{\text{return}}$, $M_{\text{dist}}$, $M_{\text{BUE}}$, $M_{\text{RBUE}}$ and $M_{\text{MFMC}}$. The evaluations I report are on the environments Mountain Car (Sutton and Barto, 1998), Five-State Chain (Strens, 2000) and Marble Maze (Leffler et al., 2007). The state-action value functions evaluated were trained using batch reinforcement-learning algorithms, namely: Least-Squares Policy Iteration (LSPI) (Lagoudakis and Parr, 2003), with either a grid-like or Fourier basis (Konidaris et al., 2008), and Q-learning with experience replay (Lin, 1992). I

provide this information to provide context for the reader, but it actually does not matter how the value functions were obtained/created as I will only be using them for comparing the different evaluation metrics. Furthermore, I will also be evaluating the learned state-action value functions against the optimal state-action function and the average reward state-action function, denoted $Q_{\text{AV}}$ in the figures.

### 4.5.1 Benchmark environments

In this subsection, I describe and provide metric results for three environments. For all of the environments and experiments, $\gamma = 0.95$. For $M_{\text{MFMC}}$, I use a Euclidean distance function.

#### Mountain Car

The first environment in the exploration of evaluation metrics was the classic Mountain Car that has been discussed earlier in the dissertation (Figure 3.7). The environment's state space has two dimensions (position and velocity) and three actions (forward throttle, backward throttle and no throttle). The environment has a reward of 0.0 when the car reaches the top of the hill and a reward of $-1.0$ for every time point when the car is not at the top of the hill. In the configuration I use for the experiments, the environment is deterministic and the car always starts at the bottom of the hill.

The setup of the experiment was as follows. To calculate $M_{\text{BUE}}$, $M_{\text{RBUE}}$ and $M_{\text{MFMC}}$, the batch data consisted of 5 sets of 30,000 transitions sampled from data collected from a suboptimal policy. The suboptimal, non-stationary policy took an average of 677 steps to reach the goal. For $M_{\text{MFMC}}$, I had $p = 30$ and $N = 1000$. The metric $M_{\text{return}}$ also used $N = 1000$. I evaluated 16

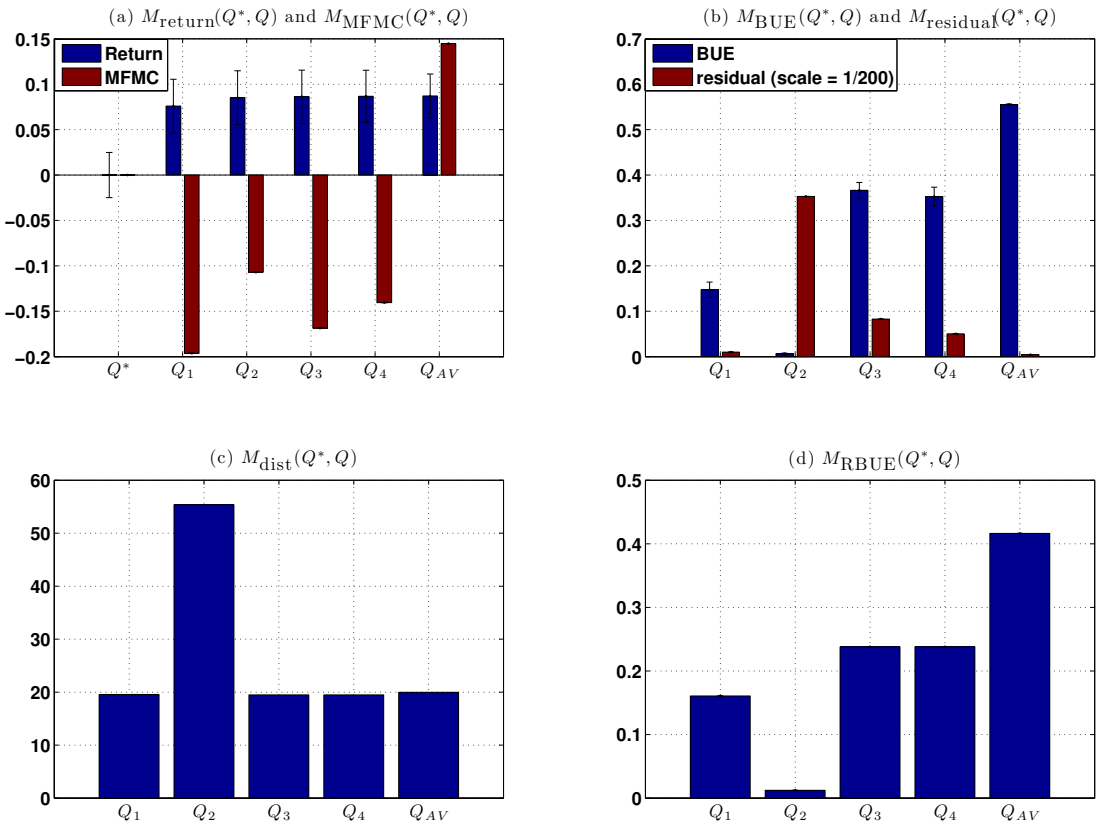


Figure 4.1: Metrics for Mountain Car

learned state-action value functions (8 from Q-learning, 8 from LSPI), the optimal state-action value function $Q^*$ and the constant function $Q_{\text{AV}}$. Given a distribution $\Pi$, I calculate the constant state-action value function that minimizes BUE (Equation 4.2) using the average reward appearing in the distribution, $\tilde{r}_\Pi = \{R(s,a)\}_\Pi^{s,a}$. That is, $Q_{\text{AV}}(s,a) = \frac{\tilde{r}_\Pi}{1-\gamma}$.

The metrics for a selected subset of state-action value functions are shown in Figure 4.1. For this environment, I also include $M_{\text{residual}}$ to aid in better understanding of the results of the other metrics.

The optimal $Q^*$ has the best performance in terms of return and the other value functions are all roughly equally suboptimal. Interestingly, $M_{\text{MFMC}}$ does

a poor job of evaluating the value functions for this problem. Although it correctly identifies $Q_{\text{AV}}$ as suboptimal, it estimates $Q_1$ through $Q_4$ as superior to $Q^*$. I analyze and discuss the poor performance of $M_{\text{MFMC}}$ later in the Marble Maze subsection (4.5.1). The metrics $M_{\text{dist}}$, $M_{\text{BUE}}$ and $M_{\text{RBUE}}$ all score $Q^*$ as better than the other value functions, although $M_{\text{BUE}}$ and $M_{\text{RBUE}}$ only barely prefer $Q^*$ to $Q_2$. On the other hand, $M_{\text{dist}}$ identifies $Q_2$ as quite a bit worse than the others, suggesting that the learner $Q_2$ fits the sampled test data well but that there are certain state-action values in the state-action space that it approximates poorly. Examining $M_{\text{residual}}$ gives us insight into this issue. As $M_{\text{residual}}$ covers the whole state-action space, it highlights which of the state-action value functions has the highest Bellman residual error over the whole state-action space. The $M_{\text{residual}}$ metric prefers $Q_2$ least when compared to $Q^*$. This observation confirms that among the state-action pairs that were not in the sampled data, $Q_2$ has a poor approximation. The rest of the learners have relatively lower $M_{\text{residual}}$.

**Five-State Chain**

The second environment I use to analyze the different offline metrics is the earlier discussed classic Five-State chain (Figure 3.2). In the classic Five-State chain, the effect of the actions are stochastic. There is an 80% chance that the agent will move in the direction of the chosen action and a 20% chance that the agent will move in the direction of the other action. A reward of 0 is given for choosing **a** in any state. The exception is that in state 5 action it results in a reward of 10. In all the states, action **b** results in a reward of 2. For the metrics that need a start state, the starting position was state 1. To compute the metrics, I used 5 sets of 10,000 transitions sampled from the optimal policy with $\varepsilon$-greedy exploration,

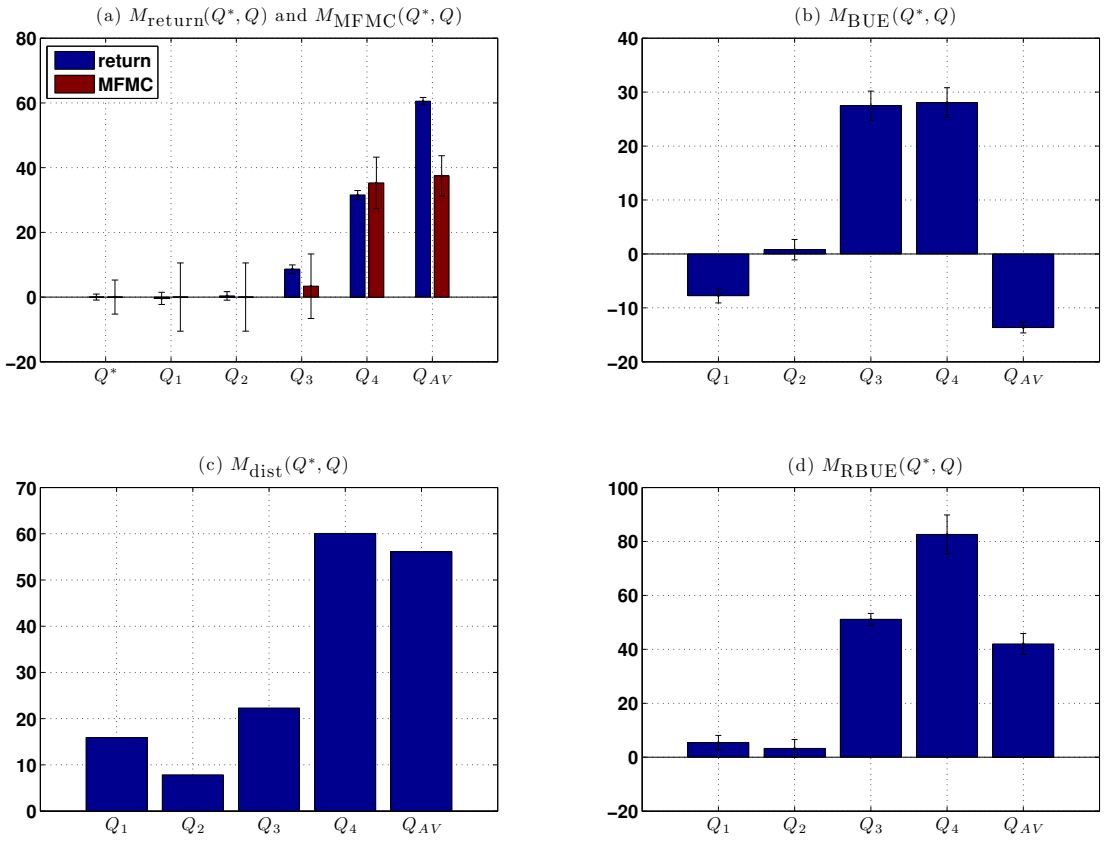


Figure 4.2: Metrics for Five-state chain

$\varepsilon$=0.2, and uniform sampling of states. For $M_{\text{MFMC}}$, $p = 50$ and $N = 100$ and $M_{\text{return}}$ used the same $N$. I compared 20 learned state-action value functions and $Q^*$ and $Q_{\text{AV}}$.

The results for these five value functions in this environment are shown in Figure 4.2. In this smaller, more thoroughly sampled example, $M_{\text{MFMC}}$ tracks $M_{\text{return}}$ well.

The metrics $M_{\text{dist}}$ and $M_{\text{RBUE}}$ make similar assessments to each other, incorrectly flagging $Q_3$ as nearly as good as $Q^*$. As this environment is stochastic, $M_{\text{BUE}}$ is able to judge two value functions as superior to $Q^*$. The most interesting comparison is with $Q_{\text{AV}}$, the worst value function by nearly every other metric. Because it is a constant function, the variance term of $M_{\text{BUE}}$ for this

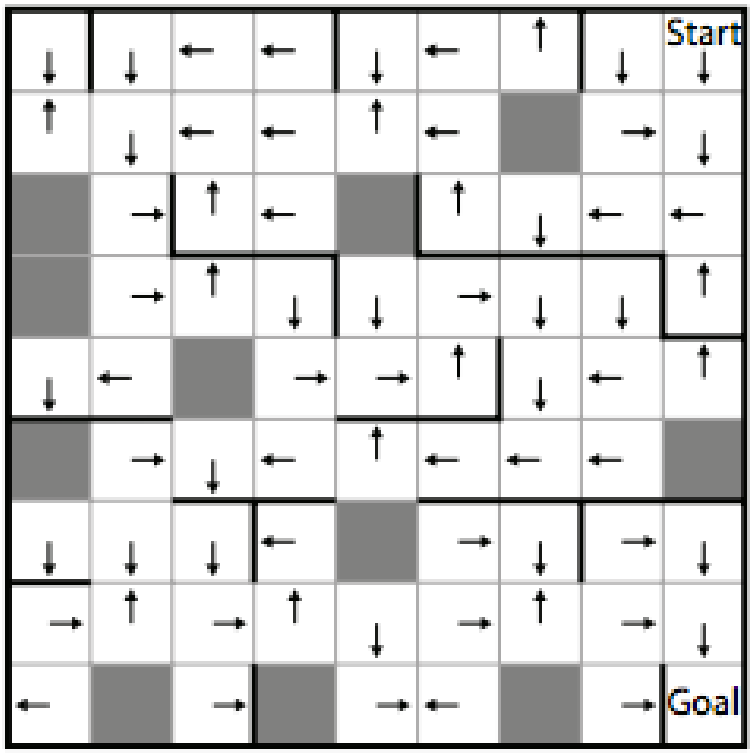


Figure 4.3: Marble Maze with optimal policy (Leffler et al., 2007)

function is zero, which makes the value function appear much better than it is.

**Marble Maze**

The Marble Maze is a 2-dimensional discrete grid world with 81 states that includes a start state, pits, walls, and a goal (Figure 4.3). Its actions are *up*, *down*, *left* and *right*. Action effects are stochastic—there is 0.8 probability that the agent will move in the direction of the action, 0.1 probability that it will move left of the action chosen and 0.1 probability it will move right of the action chosen. A reward of $-0.0001$ is given for every timestep until the goal or a pit is reached. A reward of $-1.0$ was given for falling into a pit and a reward of 1.0 was given for reaching the goal.

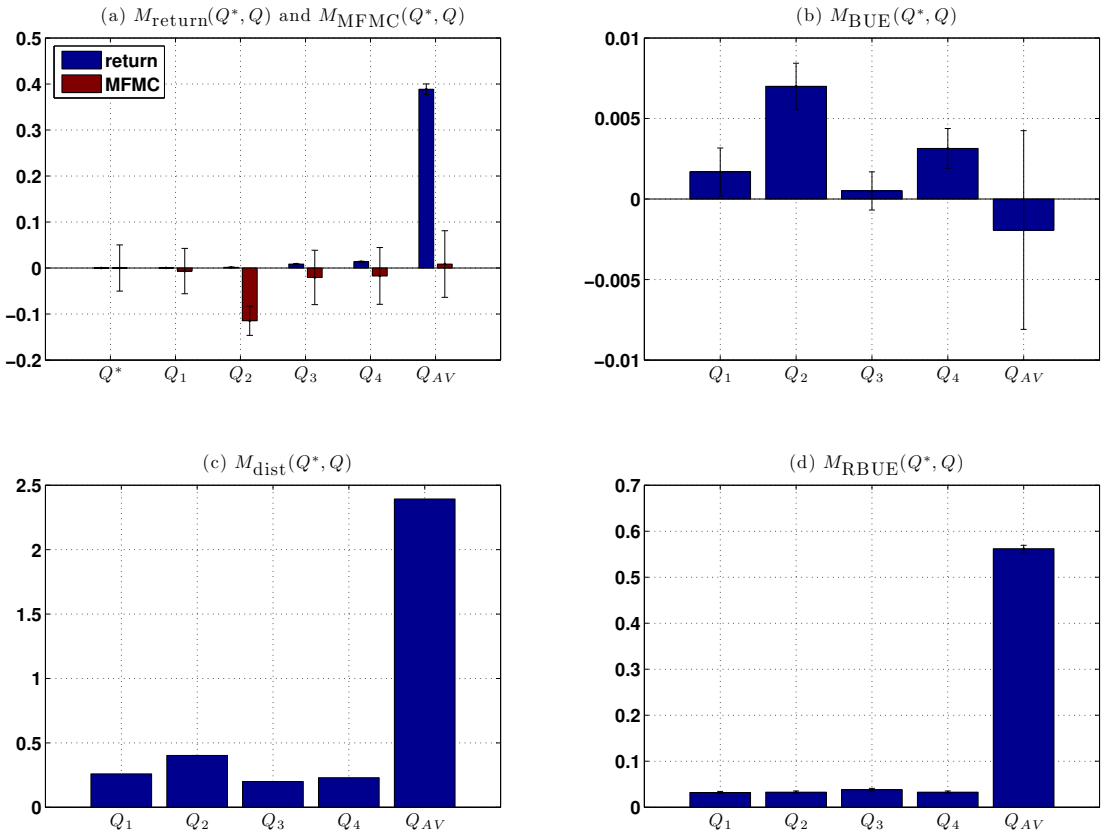


Figure 4.4: Metrics for Marble Maze

For $M_{\text{BUE}}$, $M_{\text{RBUE}}$ and $M_{\text{MFMC}}$, the data consisted of 5 sets of 30,000 transitions sampled from data collected from a random policy. In the data collection, the random policy also had random start states, instead of the one specified in Figure 4.3. The average trajectory length was 13.5 steps. For $M_{\text{MFMC}}$, the evaluation used $p = 50$ and $N = 1000$. The metric $M_{\text{return}}$ also used $N = 1000$. The return was calculated from a single start state as shown in Figure 4.3 . I create the same number and types of state-action value functions as described in Section 4.5.1. Comparisons for a subset of the state-action value functions is shown in Figure 4.4.

In this environment, all the value functions performed nearly optimally, with the exception of $Q_{\text{AV}}$, which was quite poor. The metric $M_{\text{MFMC}}$ does not

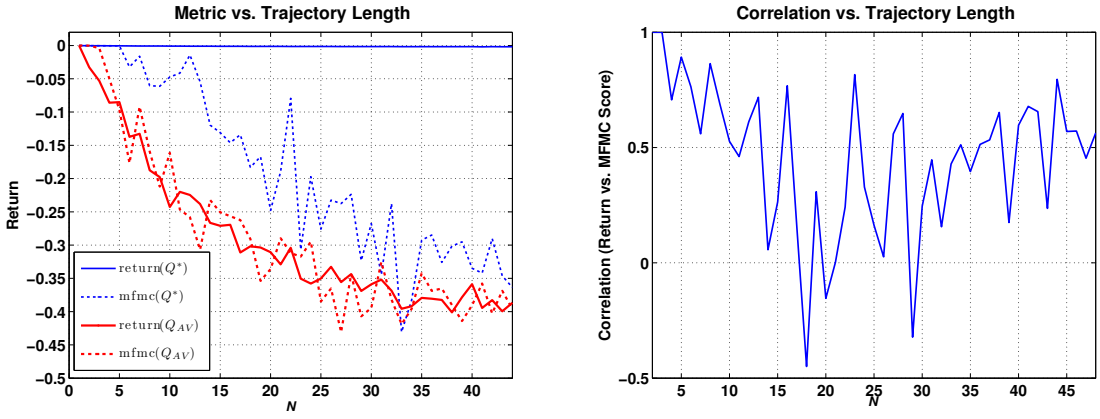


Figure 4.5: Analysis of Return and MFMC scores in Marble Maze

track $M_{\text{return}}$ well, rating $Q_{\text{AV}}$ as good compared to $Q^*$ and $Q_2$ as better than $Q^*$. Here, $M_{\text{RBUE}}$ and $M_{\text{dist}}$ correctly assess the learned value functions as being near optimal and $Q_{\text{AV}}$ as being poor. Again, as suggested by the analysis, the stochastic nature of this environment causes trouble for $M_{\text{BUE}}$, which rates $Q_{\text{AV}}$ as superior to $Q^*$ due to its lower variance.

The $M_{\text{MFMC}}$ metric floundered in this environment just as it had in Mountain Car. We find that the utility of $M_{\text{MFMC}}$ is reduced when trajectories and action chains are long. From Figure 4.5, we see that, as $N$ increases, $\text{mfmc}(Q^*)$ diverges from $\text{return}(Q^*)$. In contrast, the performance of $Q_{\text{AV}}$—essentially a random policy—is predicted accurately. When we look at the correlation over all the action-value functions' policies evaluated for this environment, we find that, after $N > 3$ or so, the correlation between the two scores is erratic.

### 4.5.2 Comparing the metrics

One way of comparing the evaluation metrics is to examine the correlations between each of the metrics and the gold standard, $M_{\text{return}}$. Table 4.1 reports the correlations for Mountain Car (MC), Five-state Chain (5S), and Marble Maze

Table 4.1: Correlation of metrics vs. $M_{\text{return}}$

| Metric | MC | 5S | MM |
|---|---|---|---|
| $M_{\text{DIST}}$ | 0.5718 | 0.5346 | 0.9192 |
| $M_{\text{BUE}}$ | 0.1194 | 0.2990 | −0.3492 |
| $M_{\text{RBUE}}$ | 0.1197 | 0.3029 | 0.9385 |
| $M_{\text{MFMC}}$ | −0.0294 | 0.9737 | 0.2016 |

(MM). The correlation calculation is over the full set of action-value functions for each environment.

Only two of the metrics evaluated consistently correlated positively with $M_{\text{return}}$, $M_{\text{dist}}$ (which requires knowledge of $Q^*$) and the newly proposed $M_{\text{RBUE}}$.

In Mountain Car, $M_{\text{MFMC}}$ produced inaccurate predictions. In the Five-state chain (5S), all the metrics correlated positively, but $M_{\text{BUE}}$ was the weakest because of its difficulty in dealing with stochastic outcomes. In Marble Maze (MM), $M_{\text{dist}}$ and $M_{\text{RBUE}}$ correlated very well with $M_{\text{return}}$ while the stochastic nature of this domain resulted in a negative correlation for $M_{\text{BUE}}$.

Summarizing the findings across all metrics:

- $M_{\text{return}}$ is the gold standard online metric. Its major drawback is that it cannot be used offline—access to the real environment or an accurate simulator is required.

- $M_{\text{dist}}$ correlates reasonably well with $M_{\text{return}}$ and can be applied offline. Its use, however, requires knowledge of $Q^*$, which can be extremely difficult to obtain.

- $M_{\text{MFMC}}$ is an offline metric. It is easy to calculate and, in many cases, it can produce very accurate estimates of $M_{\text{return}}$. In general, it requires a good similarity function and sufficient testing data to adequately represent the state-action space. It also seems to have difficulties accurately simulating

long trajectories, limiting its utility.

- $M_{\text{BUE}}$ is an offline metric and is also very easy to compute. Its drawback is that it produces inaccurate evaluations in non-deterministic environments because of a sensitivity to variance.

- The proposed metric, $M_{\text{RBUE}}$, is also easily computable offline. It is robust to non-deterministic settings and has the potential to be developed further. A shortcoming is that it can only make relative judgments between pairs of state-action value functions, making it impossible to produce a ranking of a collection of functions directly. Like all offline methods, it has a dependency on how testing data is collected—a biased sample can produce misleading evaluations.

## 4.6 Summary and future work

In this chapter, I presented a first approach to evaluating reinforcement-learning algorithms offline. The relative Bellman update error metric ($M_{\text{RBUE}}$) introduced in this chapter can be used to build an online reinforcement-learning evaluation repository. I believe that, if sampled data sufficiently explores the state and action space, $M_{\text{RBUE}}$ provides a foundation on which batch sampled data can be made available to researchers via the Internet and algorithms can be evaluated on common datasets without the complexity of creating and sharing simulators. The metric can be used completely offline to evaluate the algorithms or it can allow for a centralized service that could evaluate algorithms by comparing the state-action value functions they produce to other collected state-action value functions. Note that mechanisms for producing rankings

from pairwise comparisons are well studied in the board game and sports communities; future work will examine adapting these schemes to the algorithm evaluation setting.

Another potential use of batch metrics is in creating selection-based ensemble reinforcement-learning algorithms. Earlier (Section 3.4.1) I discussed an approach to a fusion based ensemble based reinforcement learning algorithm, but we could use the batch data collected during the testing phase of an ensemble algorithm to make choices about which algorithms to use. That is, we could select algorithms in an ensemble using a batch metric.

In the next chapters, I continue the exploration of offline evaluation of RL algorithms by focusing on evaluating the policies produced by the algorithms using the batch data. Given that metrics which use batch data are affected by the manner data collection is done and whether important parts of the state space are covered, we need to account for the uncertainty that might result when doing evaluation using batch data.

# Chapter 5
# Uncertainty in Offline Evaluation of Policies

## 5.1 Introduction

In Chapter 4, we explored the use of multiple metrics to do offline evaluation for reinforcement learning. The focus was on using comparisons of the state-action value functions from different algorithms. A limitation of the developed metric, Relative Bellman Update Error, was that it could produce misleading evaluations of algorithms given biased samples. In collecting the data samples, there is no guarantee there is sufficient data for all state-action pairs and no dependable relationship between the policy being evaluated and the policy used for collection. With this limitation in mind, I focus on how we can quantify uncertainty in our evaluation of algorithms given batch data.In Chapter 3, when looking at doing online evaluation, policies resulting from algorithms on the real MDPs (online) were evaluated. We now return to evaluating the policies, but using batch data (offline). To evaluate policies using batch data, we need to deal with limitations that batch data introduces. Specifically, the policy used to collect samples may not have adequately explored the state space relative to the policy being evaluated. The policy we are trying to evaluate may result in a very different state–action frequency than the one that generated the batch data.

Domains like medicine, education, and marketing involve sequential interventions to *individuals*: treating patients, teaching students and tempting customers. There exist many possible sequential interventions or policies and there is a huge need and opportunity to create personalized predictions of their effectiveness for unique individuals. In one way of thinking, each task is run on set of distinctive MDPs. Recognizing that different individuals are slightly different environments means that information that helps identify the behavior of the current individual could be very useful to inform which strategy to employ to maximize reward. To create individualized predictions, we seek to leverage the increasing use of electronic medical record systems Yoo et al. (2012); Stewart et al. (2014) Massive Open Online Classes (Siemens and Long, 2011), and online stores (Chen et al., 2012), which are producing a wealth of information about existing deployed policies and their outcomes. It offers a new opportunity to inform the selection of personalized sequential policies that are empirically evaluated on prior data.

The goal, in this chapter, is to develop algorithms that can use batch information to make accurate predictions about the range of effectiveness a particular strategy might have for a given individual, such as the range of quality-adjusted life years a particular diabetes patient might experience if she follows a certain treatment strategy. Such ranges are important since both individuals and organizations are frequently not risk-neutral and wish to understand the distribution of potential returns given the many sources of uncertainty.

Significant sources of uncertainty are:

1. *Data sparsity*. If there is a limited amount of data that is relevant to the policy or individual of interest, it will yield significant uncertainty in the resulting policy-return estimates.

2. *Latent feature uncertainty*. Though a new individual is often similar to some of the individuals about which we have data, exactly which previous individuals a new person is related to is frequently unknown *a priori*, leading to difficulties in predictions.

3. *Intrinsic stochasticity*. Inherent uncertainty in outcomes, even if the true probabilities of each outcome are precisely known, also makes a significant contribution to the uncertainty concerning the result of executing a policy.

In this chapter, I quantify prediction accuracy in a way that can be applied in the context of batch data. I introduce an approach that produces personalized assessments of an input policy's effectiveness given batch data. To my knowledge, this model-based reinforcement-learning algorithm is the first approach that addresses all three important sources of uncertainty over predictive payoff. It handles *latent feature uncertainty* by modeling the data as a latent mixture of subpopulations of individuals, explicitly quantifies *data sparsity* by accounting for the limited data available about the underlying models, and incorporates *intrinsic stochasticity* into its predictions to yield estimated percentile ranges of the effectiveness of a policy for a particular new individual instead of only the policy's average effectiveness.

### 5.1.1 Related work

In this chapter, we will be concentrating on predicting the possible returns resulting from executing a given policy in the context of a specific individual. We assume that the individual–environment interaction can be modeled as a Markov decision process (MDP). The value of a policy starting from a state $s$,

$V^{\pi}(s)$, is the expected sum of discounted rewards obtained from state $s$ when actions are selected according to policy $\pi$. We assume that the transition and reward models are not provided in advance.

In model-based RL (Sutton and Barto, 1998), an explicit estimate of the model parameters is computed and used to extract the value function. The policy-evaluation problem is to estimate the value of a given policy, which is the problem I focus on in this chapter. Batch RL assumes access to a previously collected dataset of state transitions and uses it to perform policy evaluation or identify the best policy.

Though most popular and effective batch RL methods (e.g., Lagoudakis and Parr (2003)) do not explicitly consider the uncertainties I describe, there has been some research on estimating uncertainty over the policy value (the uncertainty over the expected sum of rewards) when estimating with limited data. Strehl and Littman (2005) derived theoretical bounds on the resulting uncertainty in the policy value of discrete state-action MDPs when computed using MDP model parameter values that may be inaccurate (due to limited data). Mannor et al. (2007) estimated the bias and variance of the value of a policy due to estimating with sparse data, also in the context of discrete state–action MDPs. In the context of continuous-state MDPs, Shortreed et al. (2011) introduced an approach to learn the optimal policy and value as well as its uncertainty given prior (sparse) data. The method was developed to find optimal treatment strategies for patients with schizophrenia using clinical trial data. The major assumption of the approach is that the value function can be estimated using a linear combination of features.

Much less attention has been paid to incorporating stochasticity uncertainty. Work by Tamar et al. (2013) extended TD learning to estimate the variance in

policy return using linear function approximation. Other work developed parametric (Morimura et al., 2010b) and nonparametric (Morimura et al., 2010a) algorithms for approximating the density of returns by defining a distributional Bellman equation.

To my knowledge, no work in RL considers the uncertainty that arises when an individual's match to existing individuals in the observed data is unknown (latent feature uncertainty). In high stakes domains, it is important to take all three sources of uncertainty into account to create personalized policy-evaluation estimates.

### 5.1.2 Outline

In this chapter, I first introduce the $\alpha$-interval loss function, a loss function that will allow us to evaluate range predictions. I then describe a gridworld example that will be used throughout the chapter in different configurations. I define the latent class MDP, an approach to taking into account the different sources of uncertainty already described. The need for identifying latent structure is motivated via an extension of the gridworld example. An algorithm to learn the latent class MDP and produce outcome predictions is developed. Finally, I demonstrate how this approach allows us to highlight variability in policy effectiveness amongst HIV patients given a prior patient treatment dataset as well as predicting the outcome of a not-for-profit fund-raising campaign. The results show that taking into account individual variability and data limitations can lead to improved range predictions of evaluating the effectiveness of a policy for new individuals given prior batch data.

## 5.2 Interval loss function

A loss function defines a penalty for a prediction $\hat{x}$ as a function of the actual realization $x$ of the predicted quantity represented by the random variable $X$. For example, squared loss is defined as $J(\hat{x}|x) = (x - \hat{x})^2$ and is well known (Murphy, 2012) to be minimized by setting $\hat{x} = \mathbb{E}[X]$—the mean of the predicted quantity. We introduce an $\alpha \in (0, 1]$ interval loss function, designed to encourage accurate predictions of the *range* of possible realizations of the random variable being predicted. Concretely, if the prediction for random variable $X$ is the range $(\ell, u)$ and the actual value turns out to be $x$, I define the interval loss as

$$J_\alpha(\ell, u|x) = \begin{cases} \frac{\alpha}{2}(u - \ell), & \text{if } \ell \leq x \leq u \\ x - \frac{\alpha}{2}\ell - (1 - \frac{\alpha}{2})u, & \text{if } x > u \\ \frac{\alpha}{2}u + (1 - \frac{\alpha}{2})\ell - x, & \text{if } x < \ell. \end{cases} \tag{5.1}$$

That is, a realization that falls within the predicted interval receives a constant loss of $\alpha/2(u - \ell)$ and a prediction outside the interval receives a loss that is its distance from a specific point inside the interval.

The penalties are defined so that the loss is minimized by setting the bounds so that a fraction of $\alpha/2$ of the observations fall below the interval and $\alpha/2$ of the observations fall above the interval. This loss function extends the absolute loss, which is minimized by predicting the median (Murphy, 2012).

**Theorem 5.2.1.** *$J_\alpha(\ell, u|x)$ is minimized when $P(x > u) = \frac{\alpha}{2}$ and $P(x < l) = \frac{\alpha}{2}$.*

*Proof.* First, note that

$$\begin{aligned}\mathbb{E}[J_\alpha(\ell, u|x)] &= \frac{\alpha}{2}\int_\ell^u (u-\ell)p(x)dx \\ &+ \int_{-\infty}^{\ell}[\frac{\alpha}{2}u + (1-\frac{\alpha}{2})\ell - x]p(x)dx \\ &+ \int_u^{\infty}[-\frac{\alpha}{2}\ell - (1-\frac{\alpha}{2})u + x]p(x)dx. \end{aligned} \tag{5.2}$$

This equation is minimized by setting $u$ and $\ell$ to the solutions of $\frac{\partial \mathbb{E}[J_\alpha]}{\partial u} = 0$ and $\frac{\partial \mathbb{E}[J_\alpha]}{\partial \ell} = 0$, respectively. Using Leibniz's rule to differentiate the integrals, we see

$$\begin{aligned} 0 &= \frac{\partial \mathbb{E}[J_\alpha]}{\partial u} \\ &= \frac{\alpha}{2}\left[\int_\ell^u p(x)dx + \int_{-\infty}^{\ell} p(x)dx\right] - \int_u^{\infty}(1-\frac{\alpha}{2})p(x)dx \\ &= \frac{\alpha}{2}P(\ell \leq x \leq u) + \frac{\alpha}{2}P(x < \ell) - (1-\frac{\alpha}{2})P(x > u). \end{aligned}$$

Solving for $u$, we get $P(x > u) = \frac{\alpha}{2}$. That is, $u$ should be the upper $\alpha/2$ quantile. An analogous argument can be used to show that $P(x < \ell) = \frac{\alpha}{2}$.

□

The next section presents comparisons of approaches for estimating outcome ranges for policies given batch data.

### 5.2.1 An illustrative gridworld example

Next, I explore the notion of interval loss in the context of a concrete example. Consider the problem of learning to predict the return in a version of the $4 \times 3$ gridworld domain from Russell and Norvig (2010) (Figure 5.1). In this domain, the goal of the robot is to reach the top right corner. The robot always starts at the bottom left corner. The state space consists of 11 grid positions, excluding an obstacle near the center. The actions that are available are *up*, *down*, *left* and

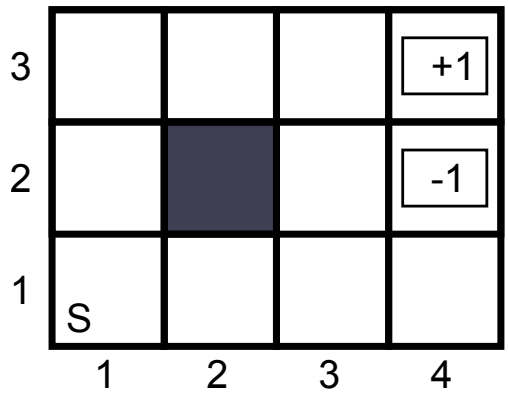


Figure 5.1: $4 \times 3$ Gridworld

*right*. The reward is $-0.04$ for being in any state except the goal state ($+1$) and the location immediately below the goal ($-1$), both of which are terminal states.

The agent has probability $1 - p$ that it will move in the direction of the action chosen, $p/2$ that it will move left of the action chosen, and $p/2$ that it will move right of the action chosen. We term this stochasticity $p$ the *slip probability* of the MDP. The experiment that follows used slip probability $p = 0.3$.

We examine four policy-evaluation prediction algorithms:

1. *Expected Outcome*: Compute the maximum likelihood estimates of the transition function and reward function from the data. From the learned model, compute the expected return $\hat{x}$ of the policy and predict the interval $(\hat{x}, \hat{x})$. This approach ignores uncertainty due to data sparsity and marginalizes out the intrinsic stochasticity.

2. *Certainty Equivalence Ranges*: Again, compute the maximum likelihood model. Use it to estimate the $\alpha$-percentile range of returns by simulating the rollout of the policy in the model. This approach accounts for intrinsic stochasticity in the return but ignores the issue of data sparsity.

3. *Variance Algorithm*: Use the approach of Mannor et al. (2007) to estimate

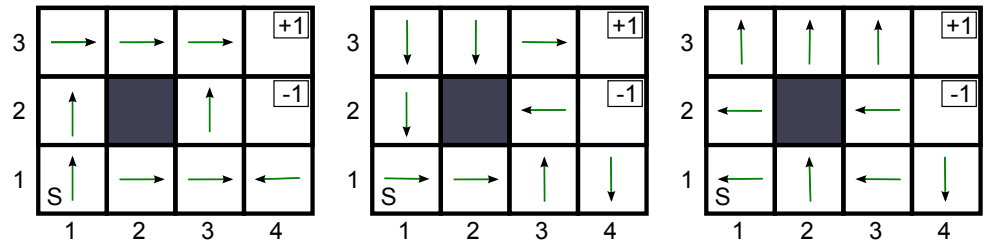


Figure 5.2: Gridworld policies $\pi_1$, $\pi_2$ and $\pi_3$ respectively (left to right)

the variance of the outcome, using it to calculate the $\alpha$-confidence intervals (assuming a Gaussian form). This approach accounts for data sparsity but ignores intrinsic stochasticity.

4. *Mixed Model*: This algorithm creates the Dirichlet posterior distribution over the multinomial transition model probabilities. (For efficiency, the reward function is estimated by its maximum likelihood model.) It then samples transition models from the posterior and carries out rollouts as in the certainty equivalence ranges method. This approach factors in both data sparsity and intrinsic stochasticity when making its predictions.

Data was collected by sampling uniformly one of 3 policies and then acting in the gridworld, beginning from the start state and acting for a maximum of 50 steps. The deterministic polices are: A policy that initially moves upwards and then to the goal trying to avoid the pit; a policy that goes to the goal by moving safely next to the pit; a policy that is unlikely to reach the goal, choosing actions that are perpendicular to the action the first policy would have chosen. The policies are illustrated in Figure 5.2. The amount of data presented to each of the four prediction algorithms listed above is varied.

The interval loss function for each approach for three different policies was computed. Concretely, given an input set of sampled data, a particular policy,

and one of the four prediction algorithms, an interval over the potential discounted returns, $\gamma = 0.95$, of this policy is computed. The interval loss over a held out set of 300 returns (100 for each of the policies) was computed. The experiment was repeated 10 times.

The results, shown in Figure 5.3, show—not surprisingly— that prediction accuracy for all prediction algorithms is lowest (that is, loss is highest) with the least data. Interval loss drops with increasing data for 3 of the prediction algorithms. In all cases, data sparsity uncertainty is resolved with more data, leading the expected outcome and variance algorithms to converge to the same interval loss (that of predicting the mean) and the certainty equivalence ranges and mixed model algorithms to converge to the same interval loss (the minimum possible with this distribution). The variance algorithm follows an interesting pattern. It has lower accuracy as its predicted variance of model parameters decreases. As it gets more data, however, it converges to the same interval loss as the expected outcome algorithm—the level associated with predicting the mean of the true distribution. Similarly, the mixed model and certainty equivalence ranges algorithm converge to the same low interval loss—that of the best possible interval predictions. Throughout the graph, the more explicitly the algorithm treats the two sources of uncertainty, the lower the loss it achieves. The Mixed Model achieves lower interval loss with scarce data (parameter uncertainty) and deals well with intrinsic stochasticity even when there is a lot of data. (At 100 training samples, 72% of the test data points were within the 80% predicted range.)

Figure 5.4 visualizes the predictions made by the four prediction algorithms in the lowest data case considered, four training samples. The gold standard is shown at the left of the plot, followed by the results of ten different runs of the

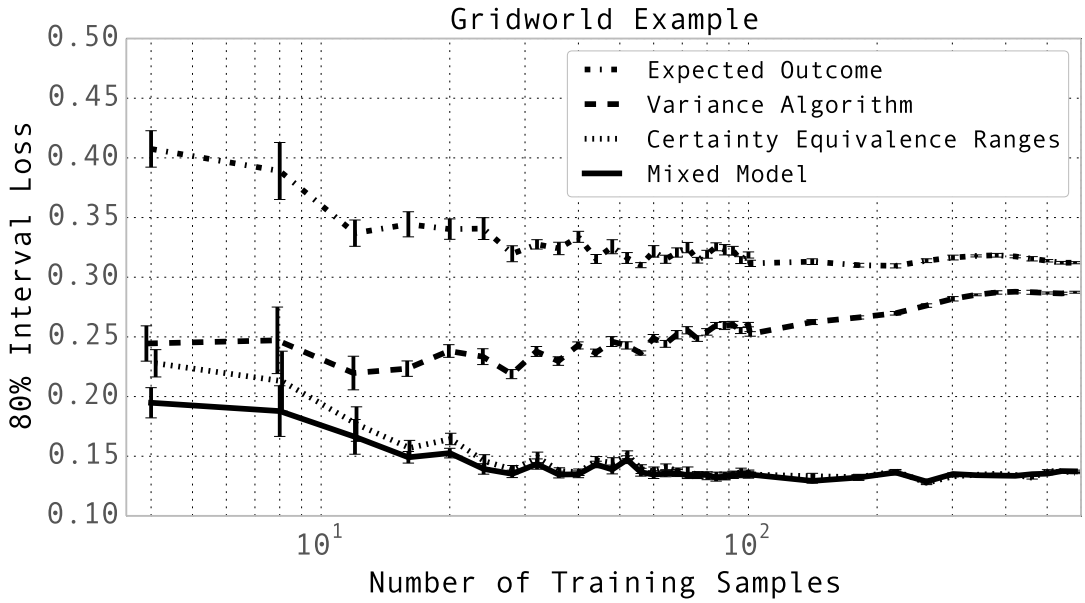


Figure 5.3: Comparison of different policy-evaluation prediction algorithms by interval loss

mixed model, certainty equivalence ranges, variance, and expected outcome algorithms. As is evident, there is significant variability from run to run given only four training samples. Of all the algorithms, the mixed model algorithm is best able to consistently generate good predictions of the true range.

## 5.3 Latent class MDPs

The mixed model algorithm effectively deals with uncertainty due to data sparsity and the intrinsic stochasticity of a policy's return. However, frequently, an additional source of uncertainty arises because it is unclear how the new individual, for which we wish to produce a personalized estimate of the effectiveness of a particular policy, relates to the individuals about which we already have data. It is often the case that different subgroups of individuals are best modeled with different model parameters. For example, Lewis (2005) found

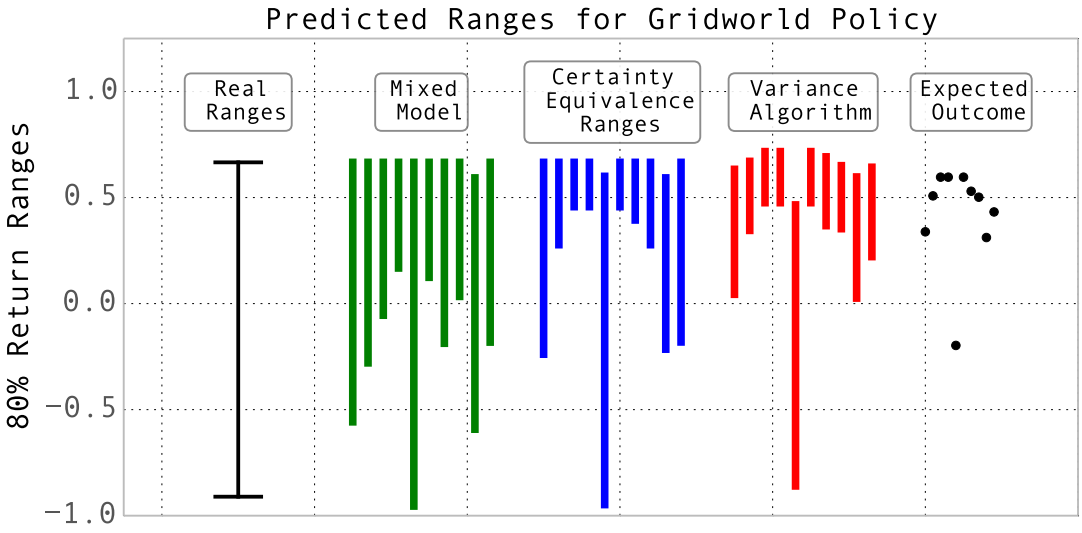


Figure 5.4: Predicted ranges for the four prediction algorithms based on only 4 training samples for Policy 1

that when computing a dynamic pricing policy for news delivery, there were two latent types of customers, each with distinct MDP transition parameters. Such different parameters can lead to quite different returns for the same policy. For instance, the effectiveness of a treatment may depend on an individual's age, sex, ethnic background, illness severity or functions of her underlying complex physiology.

If these subpopulations are known in advance, each subpopulation can be given its own predictor. However, sometimes the right way of breaking up the community of individuals into groups or classes with similar behavior (in response to a sequence of decisions) is revealed only through the data itself (such as in the customer work of Lewis (2005)). In this situation, we can view each individual as belonging to a latent class. The *latent class MDP model* provides a framework for representing and learning these classes, and using them to improve the personalized policy predictions.

To define a latent class MDP, I augment the standard MDP model with a set

of classes $C$, and a static $d$-dimensional feature vector $f$. The transition function (earlier described in Section 2.2) becomes conditioned on the class: $T_c(s' \mid a, s)$ is the probability that state $s$ will transition to state $s'$ given action $a$ for agents in class $c \in C$. The static feature vector is assumed to be generated from a multi-variate normal with class-dependent parameters: mean $\boldsymbol{\mu}_c$ and diagonal covariance matrix $\boldsymbol{\Sigma}_c$. The static feature vector is used to capture the common situation that there is some additional, static features known about an individual (e.g., demographics).

We assume the input batch data is provided in the form of observational samples, $O = \{\zeta_1, \ldots, \zeta_N\}$. Each sample $\zeta_i$ has an associated feature vector and a trajectory of $K_i$ steps. A standard MDP is simply a special case of the proposed formulation with no additional static features. We assume that a state $s$, such as the top right corner of a grid or a patient with high blood pressure, is observed via measured data $o$, for example, GPS coordinate or readings from a blood-pressure meter. The $i$th sample $\zeta_i$ can be written as

$$\zeta_i = \{f_i\}\{(s_i^1, a_i^1, r_i^1), \ldots, (s_i^{K_i}, a_i^{K_i}, r_i^{K_i})\},$$

where $f_i$ is the $d$-dimensional feature vector. The reward $r_i^k$ is assumed to be a function of the observation or observation history. The returns that are received for an observation sample $\zeta$ are defined as $\mathfrak{R}(\zeta) = \sum_{k=1}^{K_i} \gamma^{k-1} r_k$. To illustrate the impact of latent structure on decision making, I present a latent structure example that is an extension of the earlier gridworld.

### 5.3.1 Impact of latent structure on decision making

Let's say we design cleaning robots for one-room apartments and that the customers own apartments that can differ along a number of features such as size,

types of floors, or the presence of a pet. We would like to offer personalized predictions of effectiveness: How well will each cleaning policy work in a particular apartment? We have access to data from the robot's use by other customers. How do we use it to offer accurate predictions for a new customer? To explore this issue, I instantiate the apartment problem in an extension of the grid-world domain discussed earlier.

We introduce 3 types of apartments that the robot can encounter. The apartments have two measurable features (floor smoothness, floor slope) that affect the movement of the robot. The types encode the degree of stochasticity for actions. Specifically, on each type of floor, the robot has probability $1 - p$ that it will move in the direction of the action chosen, $p/2$ that it will move left of the action chosen, and $p/2$ that it will move right of the action chosen. As before, we term this stochasticity $p$ the slip probability of an apartment.

For the experiment, the measurable floor features were drawn from 3 underlying normal distributions corresponding to the three apartment types, with different means and variance. For $\boldsymbol{\mu}_1 = \{0.72, -0.50\}$, $\boldsymbol{\sigma}_1^2 = \{0.31, 0.42\}$, $\boldsymbol{\mu}_2 = \{-0.66, -0.66\}$, $\boldsymbol{\sigma}_2^2 = \{0.43, 0.36\}$ and $\boldsymbol{\mu}_3 = \{-0.02, 0.83\}$, $\boldsymbol{\sigma}_3^2 = \{0.32, 0.31\}$. The slip probabilities $p$ were 0.0, 0.3, and 0.9 for the three different apartment types.

To illustrate the performance of predictions made from using data collected from the apartment gridworld without first attempting to find latent structure, the results of three different cleaning policies are compared (the gridworld policies discussed in Figure 5.2). The percentile ranges derived from mixing this data together are presented and compared to the ranges derived from a single type of apartment in which our new owner would belong if we had used

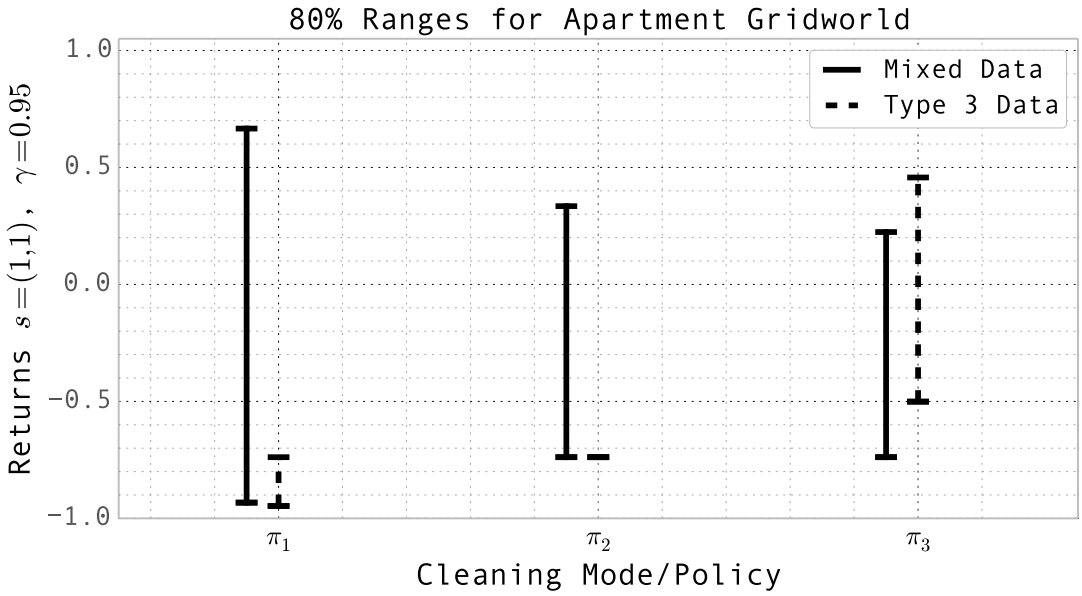


Figure 5.5: Ranges for returns in the gridworld domain using mixed data (solid line) vs. data from apartment Type 3 only(dashed line) for each policy

the feature information in classifying them. The 80% ranges for the two scenarios using 10000 collected trajectories, for each scenario, with $\gamma = 0.95$ are plotted. The resulting ranges for the return starting at block $(1, 1)$ are shown in Figure 5.5.

Unsurprisingly, the figure shows that the 80% return ranges for each of three different policies are quite different when the apartment type is ignored (solid ranges in Figure 5.5) or known (dashed ranges). Ignoring the type results in more uncertainty. The resulting bounds are also less informative about the best policy to choose—$\pi_1$ seems marginally better if we ignore the types and $\pi_3$ seems better if we know them.

This simple example suggests that if latent structure is present, it can be very helpful to model it explicitly since it can lead to improved policy-evaluation predictions.

## 5.4 Modeling uncertainty in latent class MDPs

In this section, I present the model-based Latent Structure and Uncertainty (LSU) algorithm for producing ranges over the effectiveness of a given policy for a new individual (see Algorithm 3). LSU uses observational data to learn a set of proposed latent class MDPs, and uses these models, plus the observable features of the individual, to estimate the set of policy outcomes. I next present a strategy for finding latent classes. Then, I discuss how to quantify the different types of uncertainty. Closing this section, I revisit the apartment-gridworld example.

### 5.4.1 Learning the parameters of a latent class MDP

We present an Expectation-Maximization-based clustering algorithm for fitting the parameters of a latent class MDP from observational data. I assume as input that the number of hidden classes $C$ is given, but in the empirical results I will vary this value and evaluate the resulting loss. Assume we have a set $C = \{1, \ldots, M\}$ of hidden individual classes. The objective is to find the parameters of a $C$-latent-class MDP that maximizes the likelihood of the given $N$ observational samples $O = \{\zeta_1, \ldots, \zeta_N\}$. This likelihood function incorporates both the trajectories and static features. (For simplicity, I omit the reward for the likelihood model.) The sample likelihood, $L_{\mathcal{N}}(\zeta_i|c)$, of features being drawn from model $c$ is $\mathcal{N}(f_i|\mu_c, \Sigma_c)$:

$$\begin{aligned} L_{\mathcal{N}}(\zeta_i|c) &= \mathcal{N}(f_i|\mu_c, \Sigma_c) \\ &= \frac{1}{(2\pi)^{d/2}\,|\Sigma_c|^{1/2}} e^{-\frac{1}{2}(f_i-\mu_c)^T \Sigma_c^{-1}(f_i-\mu_c)}. \end{aligned}$$

For the transitions, I create a set of $M$ transition functions, $T_c(s^{k+1} \mid a^k, s^k)$ for all $s \in S$, $a \in A$ and $c \in C$. We define $\rho_c$ as the prior probability for class $c$.

The likelihood of drawing the transitions $\zeta_i = (s_i^1, a_i^1), \ldots (s_i^K, a_i^K)$, from class $c$ is

$$L_T(\zeta_i|c) = \prod_{k=1}^{K} T_c(s_i^{k+1} \mid a_i^k, s_i^k).$$

We define

$$\tau_{ic} = \frac{L_{\mathcal{N}}(\zeta_i|\mu_c\Sigma_c)L_T(\zeta_i|T_c)\rho_c}{\sum_{c' \in C} L_{\mathcal{N}}(\zeta_i|\mu_{c'}, \Sigma_{c'})L_T(\zeta_i|T_{c'})\rho_{c'}} \tag{5.3}$$

as the probability that observation $i$ belongs to class $c$, and

$$\Psi_M = (\rho_1, \cdots, \rho_M, \mu_1, \cdots, \mu_M, \Sigma_1, \cdots, \Sigma_M, T_1, \cdots, T_M)$$

as the parameter space for the complete model. As such, the log likelihood of the samples is

$$L(\mathbf{O}|\Psi_M) = \log \prod_{i=1}^{N} \sum_{c \in C} \rho_c L_{\mathcal{N}}(\zeta_i|\mu_c, \Sigma_c) L_T(\zeta_i|T_c). \tag{5.4}$$

Equations 5.3 and 5.4 form the basis of an Expectation Maximization (EM) algorithm for a fixed number of models $M$. As we can split the likelihood function into two, the EM algorithm uses the standard EM maximization step for the Gaussian mixture models representing features (Bilmes, 1998) as well as that for the multinomial mixture model representing the transition functions. We focus on the details of updating the maximum likelihood estimations for the transition model. We will use a portion of the quantities estimated within this model to estimate uncertainty later on. In the $h$-th iteration, we have calculated $\tau_{ic}^h$, defined in Equation 5.3, in the E step, and in the M step we will iterate $\Psi_M^h \rightarrow \Psi_M^{h+1}$ by finding the maximum likelihood estimates for the Gaussian and multinomial mixture models. Additionally as part of LCEM, we also calculate the maximum likelihood estimate of the reward function $\hat{R}(s, a, s')$ for all $s, s' \in S$, $a \in A$ given the training data. We do not know the number of models $M$ so run EM with several values of $M$. We monitor the likelihood of

**Algorithm 2** Latent Class Expectation Maximization (LCEM)

**Input:** Observations $O$, number of latent classes $M$, number of iterations $H$

Initialize $\Psi_M^0$ randomly.
Split data into training $O_{Tr}$ and validation $O_V$ sets.
Run EM for $H$ iterations with $O_{Tr}$, return $\Psi_M^{H+1}$
Calculate $L(O_V|\Psi_M^{H+1})$
**return:** EM model $\mathbf{\Psi}^*$, $\hat{\boldsymbol{R}}$

a held out set (cross validation) but later discuss how to make a choice on an appropriate value of $M$ using the interval loss function. This algorithm, Latent Class Expectation Maximization (LCEM), is outlined in Algorithm 2.

### 5.4.2 Computing an Interval of Possible Returns

The output of the LCEM algorithm is a latent class MDP with $M$ classes as well as membership probabilities of each training trajectory to each of the latent classes. I now describe the Latent Structure and Uncertainty (LSU) algorithm (Algorithm 3) that uses these classes to estimate the range of returns of a particular policy for a new individual.

First, I wish to explicitly represent the uncertainty we have over the latent class MDP parameters due to the limited data[1]. I do so by not treating the resulting estimates from the EM procedure as point estimates, but instead by creating a Bayesian posterior probability distribution over the latent class MDP parameter estimates. This approach is an extension of the mixed model described in the gridworld example. To do so, I take the trajectories and use their soft assignments to each class to produce a Dirichlet posterior distribution over the multinomial transition model probabilities associated with each latent class

[1] Model uncertainty due to the local EM search is not modeled, but we can later assess how well the approach does in terms of capturing real individuals' returns.

MDP. In more detail, for each class, for each state–action pair $(s, a)$, I define $Dir(\lambda)$, where $\lambda_{(s,a)}$ is a count vector for each next state $s'$. This vector is set as the number of occurrences of $(s, a, s')$ triples experienced in the training observation data, weighted by the probability that each trajectory was assigned to model $c$.

For a new individual $i$ (for whom we wish to produce a personalized policy prediction), we can calculate the probability of latent class membership[2] using their input static features $f_i$, which are assumed to be known in advance (such as demographic features):

$$w_{ic} = p(c|f_i) = \frac{\mathcal{N}(f_i|\mu_c\Sigma_c)\rho_c}{\sum_{c' \in C} \mathcal{N}(f_i|\mu_{c'}, \Sigma_{c'})\rho_{c'}}. \tag{5.5}$$

We are now ready to describe how a range for the policy returns for a new individual $i$ can be estimated. First compute $w_{ic}$ for each class $c \in C$ and then repeat the following procedure many times: Sample a latent class $c$ given the individual's probability weight vector $\boldsymbol{w}_i$. Then sample a transition model for each state–action pair $\bar{T}_c$ from class $c$'s associated Dirichlet distributions. Perform a trajectory rollout, using the policy of interest to select actions for the states encountered, and the sampled transition model to generate the simulated transitions. Record the resulting return obtained during this rollout, and then repeat this whole process. Finally, use the empirical $\alpha/2$ percentiles as the prediction. Algorithm 3 summarizes the entire prediction algorithm, named Latent Structure and Uncertainty (LSU). The effectiveness of LSU is evaluated

[2] We could also incorporate uncertainty over the feature-class parameters, by computing hyperparameters over the fit Gaussian models, and adding an additional outer loop to the LSU algorithm in which we first sample a model from the feature-class hyperparameters, and then use that model to compute the probability of latent class membership for the individual. However, the experimental results suggest that incorporating additional uncertainty over the model parameters provides a significant benefit only when data is very limited, and therefore suspect that incorporating feature-class model uncertainty would make little difference empirically.

**Algorithm 3** Latent Structure and Uncertainty (LSU)

**Input:** Observations $O$, policy $\pi$, new individual $i$, simulation steps $N$, number of latent classes $M$
$H \leftarrow$ number of EM iterations
$\boldsymbol{\Psi}, \hat{\boldsymbol{R}} \leftarrow$ Run LCEM$(O, M, H)$
$\boldsymbol{w}_i \leftarrow$ Membership probability to classes in $\boldsymbol{\Psi}$ (5.5)
**for** $n = 1$ **to** $N - 1$ **do**
  Sample latent class $c$ from $\boldsymbol{\Psi}$ according to $\boldsymbol{w}_i$
  Sample $\bar{T}_c^n$ from $Dir(\lambda_{S,A}^c)$
  $\mathfrak{R}^n \leftarrow$ Simulate $\pi$ trajectory in $\bar{T}_c^n$ with $\hat{R}$
**end for**
Calculate $\alpha/2$ and $1 - \alpha/2$ ranges for $\mathfrak{R}$

by seeing whether a held-out individual's true return for the policy in question lies within the predicted interval.

To evaluate the accuracy of the predictions, we thus need the interval loss function described earlier in Section 5.2. In this setting, for each individual we can predict an $\alpha/2$ interval, but need a way in which we can say, on average whether the interval predictions are accurate. As such, the interval loss function captures how well the empirically estimated intervals cover the real outcomes.

Next, I briefly discuss the computational complexity of the LSU algorithm before revisiting the apartment gridworld and applying LSU to that problem.

### 5.4.3 Computational cost of the LSU algorithm

Here,I present the computational complexity for the full LSU algorithm on a dataset. First, I will present the analysis of LCEM and then the analysis of the prediction phase of LSU. The presentation is split into first the EM section and then the process of calculating the outcome ranges.

For LCEM, as input, we are given: $N$ trajectories, $C$ latent classes, a maximum of $K$ steps per trajectory and $d$ individual features. The computational cost, per epoch, of the LCEM algorithm is: $\mathcal{O}(NCd)$ to calculate the likelihood due to the individual features, $\mathcal{O}(NCK)$ to calculate the likelihood due to the trajectory transitions and finally $\mathcal{O}(NC)$ to put them together and calculate equation (5.3). As such, the complete E-Step costs

$$\mathcal{O}(NC(d+K)).$$

LCEM can benefit from being distributed. Approaches such as those from Wolfe et al. (2008) can be used to speed up the deployment of the algorithm.

The M-step, on the other hand, consists of: Updating the mean and diagonal covariance matrix of each cluster in $C$ with cost $\mathcal{O}(2NCd)$; Calculating the maximum likelihood estimates of the transition function results with cost $\mathcal{O}(NKC)$ to calculate the scaled $(s, a, s')$ statistics and then $\mathcal{O}(|S||A|C)$ for the ML estimates. The total M-Step cost is

$$\mathcal{O}(2NCd + NCK + |S||A|C).$$

For the LSU algorithm, with a maximum of $K$ steps to roll out the cost of each simulation epoch, consists of: $\mathcal{O}(Cd)$ to calculate the individual's membership probabilities (calculated once); $\mathcal{O}(|S||A|)$ to sample a new transition function; and $K$ simulation steps. The total cost with $m$ simulation epochs and $\mathcal{O}(1)$ to calculate the range is

$$\mathcal{O}(Cd + |S||A|m + Km + 1).$$

In general, for learning the latent model, we would like to have $N \gg |S||A|$ and as such the learning computational cost would be dominated by $\mathcal{O}(NC(d+K))$. This indicates that as we increase the number of latent classes the computational cost increases linearly in $C$.

### 5.4.4 Applying LSU to the apartment gridworld

Returning to the apartment gridworld, Figure 5.6(a) presents the observation likelihood varying the number of classes in LCEM using mixed data from all three MDP types (1500 trajectories). The LCEM algorithm was run with 20 EM iterations and the experiment repeated 10 times with the 300 samples (100 for each policy and uniformly sampled apartment) used to calculate the loss function. First, the likelihood (Figure 5.6(a)) of the validation data increases as the number of latent states increases. After $M = 3$, the gains in likelihood are small. Inspecting interval loss, $J_\alpha$, for the validation set, Figures 5.6(b) and (c), we see that, as we increase the number of latent classes, we decrease the interval loss for the 95% and 80% return ranges up until $M = 3$. Afterwards, the interval loss starts increasing. As such, we see that we can use the interval loss to gauge the accuracy of our predictions and as an indicator that overfitting is occurring.

Figure 5.7 presents the comparison of the loss for a version of LSU that used the maximum likelihood parameters, *LSU Certainty Equivalence*, and another that returns the expected value instead, *LSU Expected*. Just as in Figure 5.3, LSU Expected has a higher loss and in this case LSU Certainty Equivalence has similar loss to full LSU.

Figure 5.8 provides ranges for three different policies (shown in Figure 5.2) under three different conditions. The first two conditions –all data mixed (left range for each policy) and all data from apartment Type 3 (middle range for each policy) —were described earlier. They represent, in a sense, the worst (features not used) and the best (true class known) predictions that can be made for a new apartment of Type 3. The third condition (right range for each policy) comes from applying LSU with $M = 3$ to the same data to find the latent classes and then presenting 1000 randomly generated apartment features from Type 3

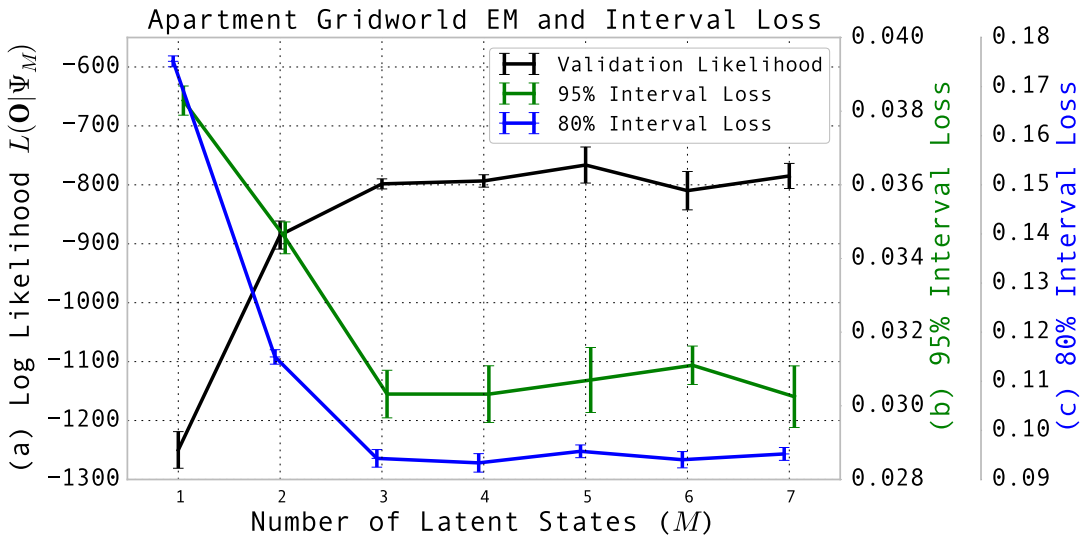


Figure 5.6: Validation Likelihood (solid line) for the LSU algorithm with mixed observational data from the apartment gridworld. Included is the 95% (dashed line) and 80% (dotted dashed line) interval loss.

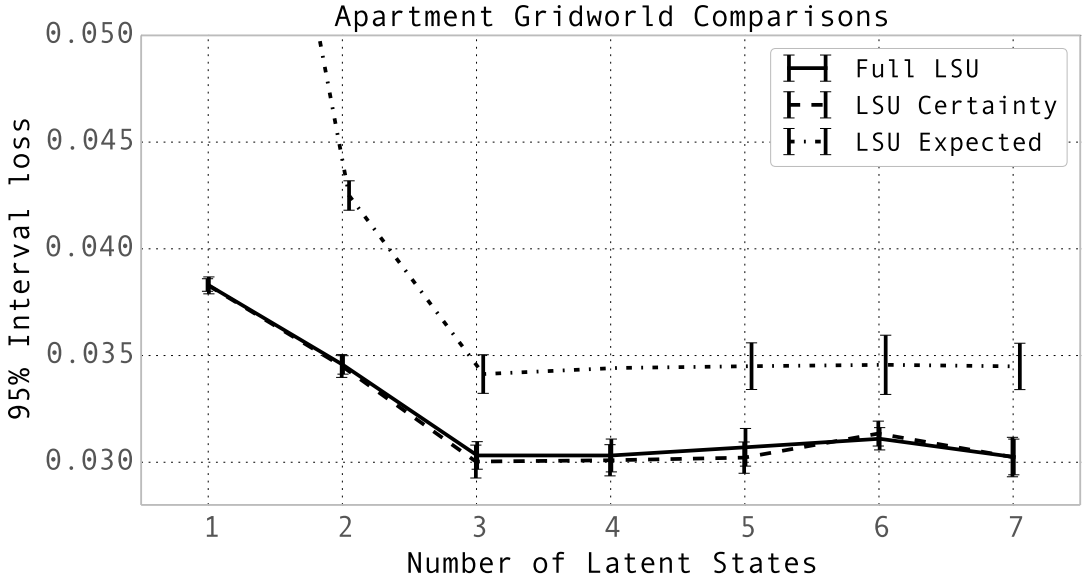


Figure 5.7: 95% interval loss function for differing versions of LSU. The full LSU algorithm is shown as a solid line, the LSU Certainty Equivalence is dashed and the LSU Expected algorithm is dashed and dotted

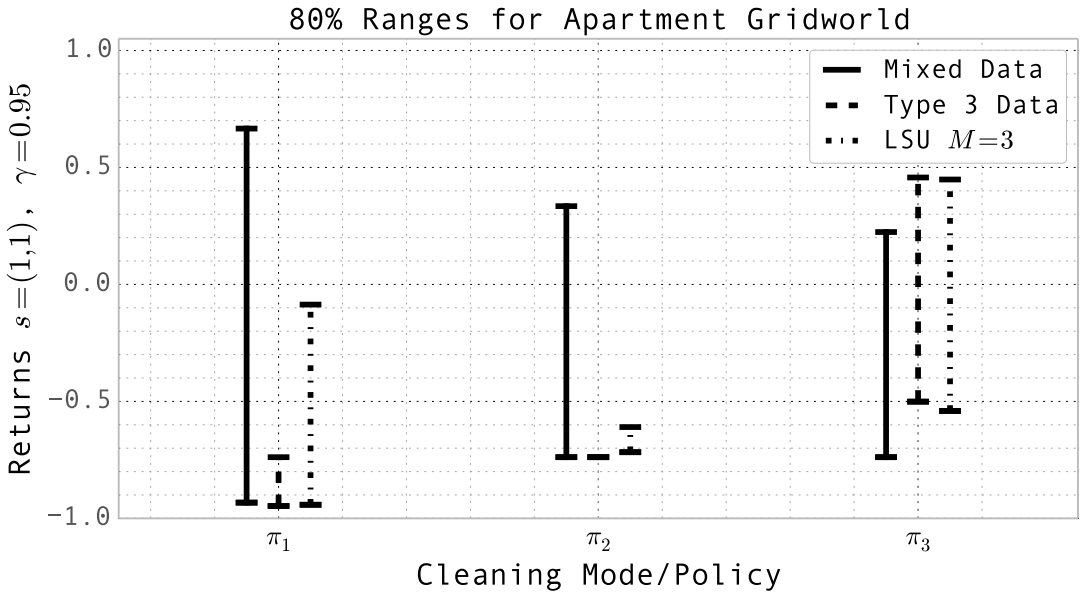


Figure 5.8: Ranges for returns in the gridworld domain for three policies. The left range for each policy is using mixed data, the middle range using data only from apartment Type 3 and the right range is the range calculated by LSU given features from apartment Type 3 owners and $M = 3$.

to LSU to estimate the value of each policy. The 1000 resulting ranges were then averaged together. As can be seen, these ranges are much closer to the best case than the worst case—LSU does a good job of estimating values from features in this case.

If we know which class each individual falls into, we can easily estimate the range of outcomes for each cleaning policy by maintaining separate statistics for each class. In practice, however, it is impossible to know that an individual comes from a particular class. I ran an additional experiment to uncover how the LSU algorithm behaves when presented with individuals from different classes. I sampled features for 100 individuals uniformly from the three classes. I then applied Equation 5.5 to estimate the probability that each individual was Type 3 and used LSU to compute the resulting range of outcomes. I plotted the probability of the individual being in Type 3 against the computed ranges of

the returns from two different policies.

The results, shown in Figure 5.9, have several noteworthy properties. First, for individuals whose probability of being Type 3 was near 1.0, policy $\pi_2$ is clearly significantly better than policy $\pi_1$, despite its wide range of possible outcomes due the stochasticity in the MDP's transitions.

Second, for individuals whose probability of being Type 3 was 0.0, both policies had narrower intervals. But, since there is uncertainty as to which of the other types of gridworlds the individuals are, there is still uncertainty in outcomes. These individuals should be given a different recommendation—$\pi_1$ is much better than $\pi_2$. Finally, in the middle of the graph, ranges are wider, capturing the fact that there is additional uncertainty as to which latent class these individuals match. For these individuals, the choice of policy would depend on their risk attitude: the top of $\pi_1$'s range is better than $\pi_2$'s, but its bottom is also lower.

## 5.5 Application to real world datasets

Having applied LSU to a synthetic dataset, we now move to real world datasets. The first case study shows the application of LSU to an HIV dataset to predict the outcomes of treatment choices available to patients. The second provides predictions of the outcomes of non-profit fund-raising appeals using a historical funding appeal donation dataset from the Paralyzed Veterans of America.

### 5.5.1 Personalized treatment uncertainty

In this subsection, I detail the application of the LSU approach to a real life observational collection. I used an HIV (Human Immunodeficiency Virus) dataset

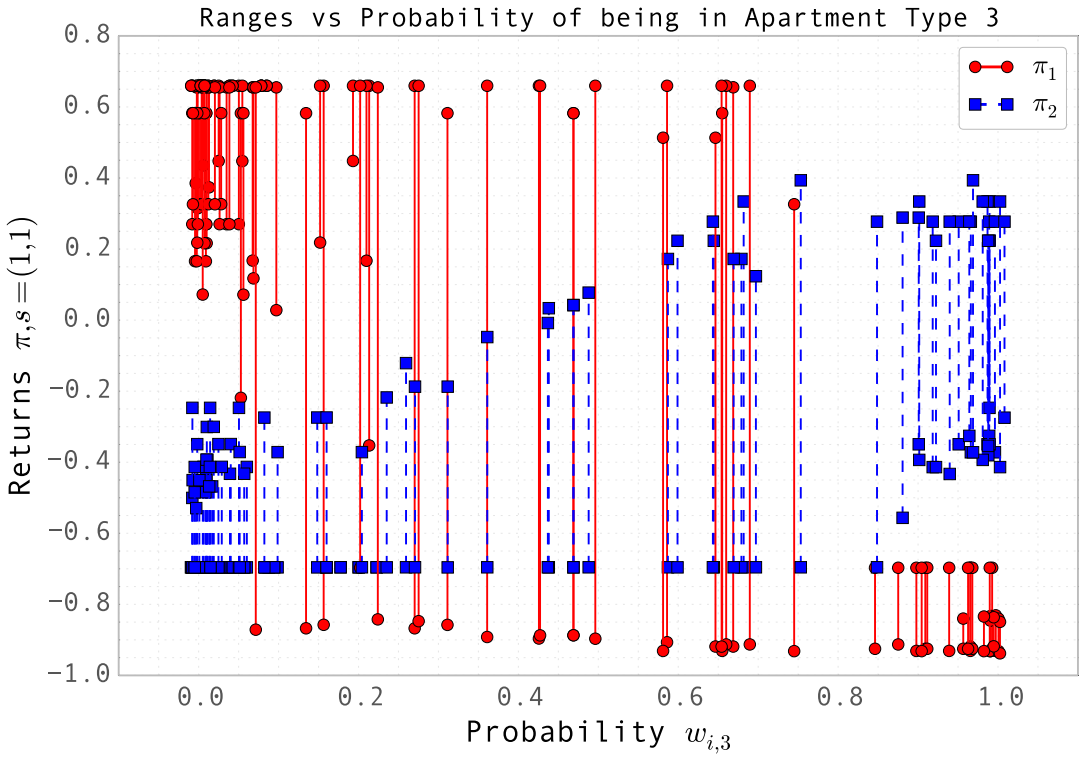


Figure 5.9: LSU 60% return ranges for 2 polices vs. probability of an individual being of Gridworld Type 3

from the EUResist project (Zazzi et al., 2012; Prosperi et al., 2009). This data differs from that of clinical trials in that it is an amalgamation of observational datasets from different patients, hospitals, and European countries. Our experiment is not a rigorous treatment of the efficacy of HIV treatments or EUResist, but serves to illustrate the potential use of the LSU approach with observational data from important areas such as medicine.

**Human Immunodeficiency Virus**

HIV is a lentivirus that slowly leads to the Acquired Immunodeficiency Syndrome (AIDS) (Weiss, 1993). After the initial infection, the virus replicates and infects and destroys CD4 T-cells, leading to the collapse of the immune system. A number of antiretroviral therapies are used in the treatment of HIV. Though these therapies are often successful, in the long term, the virus mutates, making the treatment less effective.

As such, HIV drug-treatment therapies require monitoring as well as periodic changes to the regimen. Ernst et al. (2006) have previously presented an approach of treating the HIV treatment problem as a sequential decision problem, specifically optimizing for structured therapy interruptions. Shechter et al. (2008) and Braithwaite et al. (2008) presented approaches that optimize when to start HIV therapies. Braithwaite et al. (2008), on the other hand, presented an approach to optimizing the CD4 cell threshold to begin therapies. In this work, I assume the patient is always on a treatment and use an HIV dataset (instead of a simulator) to predict the treatment-effectiveness ranges for different treatments.

**EuResist dataset**

The EuResist dataset consists of 18467 patients going through numerous HIV treatment therapies. The dataset was originally envisioned as a tool for researchers to build models that predict whether a drug would be effective or not on a patient (Zazzi et al., 2012; Prosperi et al., 2009). Current literature in the area focuses on predicting the response between 4–12 weeks. For a drug to be classified as effective, it must reduce the viral load 100-fold from baseline or result in the virus being undetectable. I use the dataset to evaluate the effectiveness of sequential therapies, similar to the approach of Shortreed et al. (2011). A significant difference from this prior work is that observational data instead of clinical trial data is being used in this case. This approach potentially makes it possible to bring a larger amount of data to bear on policy evaluation, but the lack of controls can make the interpretation of past treatments challenging. The approach to this HIV problem is different of that of Ernst et al. (2006) in that I am not evaluating policies that use treatment interruptions (I assume the patient is always on a treatment) and further use data directly from HIV patient observations as opposed to a mathematical model of the virus' progression.

I extracted from the dataset the set of patients who underwent at most 2 different treatment therapies over a 24-month period. The periods take place anytime between Jan. 2000 and Dec. 2010. I treat the viral load as a state variable and tracked its changes monthly over 24 months. Since patients did not have their viral loads taken at regular intervals, the viral loads for months in which there was no data were interpolated. A piecewise linear interpolation was used, where values for the viral load were calculated every 30 days (encoding a month). Similar to Shortreed et al. (2011), I further encode the patient's treatment stage. I take their first treatment therapy as Stage 0 and the second

treatment, after a switch, is Stage 1. Thus, $s_{1,0}$ is state 0 in stage 1.

The patients' continuous features in the latent class MDP were: baseline viral load, baseline CD4 count, baseline CD4 percentage, age and number of previous treatments.[3] The features were standardized, a linear re-scaling of the features so that each of the features has zero mean and standard deviation of 1. With assistance from HIV health-care experts, I identified the top 10 therapy/drug cocktail groups occurring in the reduced data set, discarding data that used therapies outside these groups. Each unique therapy was taken as an action. The state space was discretized by binning the values of the viral load. The bins for the viral load, in copies/mL, were [0.0,50,100,1K,100M]. State $s_{0,0}$ and state $s_{0,4}$ are thus viral loads between 0 and 50.0 copies/mL and 1K and 100M copies/mL, respectively. The reward function was the negation of the Area Under Curve (AUC), calculated monthly, of the viral load over the period being studied. This reward function favors a patient having a lower viral load over a long period of time. I calculated the return with $\gamma = 1.0$ but with a maximum of 24 steps (24 months), ensuring that returns are finite.

**Discovering latent classes**

The initial step in the approach is to find if there is indeed latent structure. To find the number of latent classes into which the HIV patient data should be split, I again randomly partitioned the data into a training set (for EM) and validation set. The training set, after standardization and removing outliers, consisted of 6552 samples while the validation set had 250 samples. To calculate the interval loss function for the dataset, I sought out the most common

[3] The dataset includes virus genomic information for only some of the patients, as such was not used it as a feature. Including genomic information is a great opportunity for future work as it is a valuable marker for resistance and mutations.

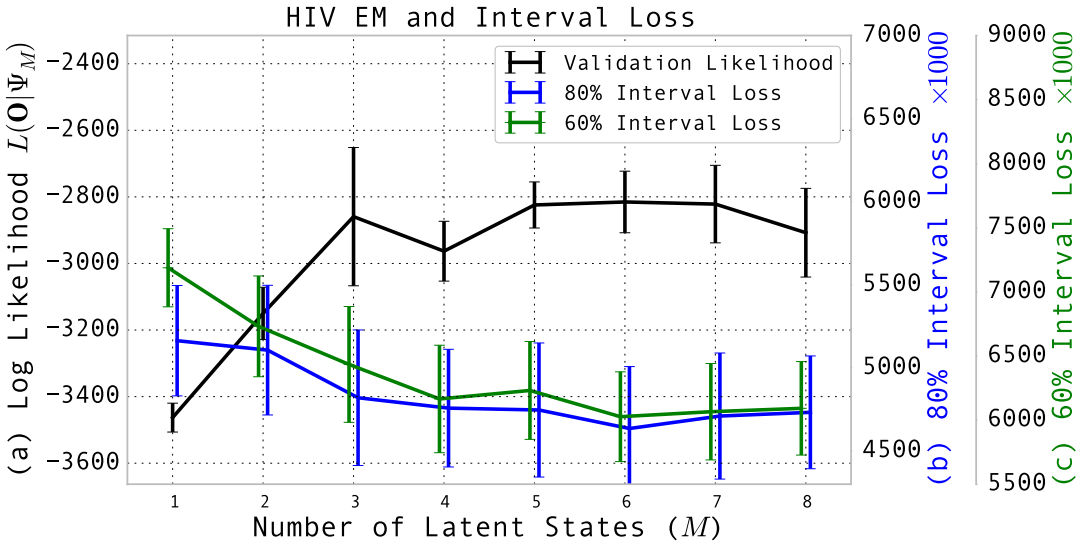


Figure 5.10: Likelihood and loss functions for the LSU algorithm with mixed observational data from the HIV dataset

two-stage treatment policies that were observed in the dataset. From the sub-population that followed these top policies, I sampled 50 patients and made them part of the validation set of 250. These 50 patients are those who would have a the highest likelihood of a switch and as such we can evaluate their full two stage policy. The results of running the LSU algorithm on the observational HIV data is plotted in Figure 5.10. The best performance of multiple EM runs (as per the interval loss function) with the same training samples is shown. The experiment is repeated 8 times.

The likelihood of observing the data increased as we increased the number of latent classes. Similarly, the 80% and 60% (Figure 5.10 (b) and (c)) interval losses drop, indicating that the interval estimates improve as well. The plot indicates that the HIV data indeed has latent structure. The gains after $M > 3$ are small.

Over multiple runs of the algorithm, there is evidence of a split between clusters comprised of patients with lower viral baseline loads (approximately

10K copies/mL) and other clusters that, in comparison, have high baseline viral loads. The low viral-load cluster tends to have a higher prior probability than the rest of the clusters. Having analyzed multiple runs of the LCEM algorithm for this dataset, the algorithms does not settle on a single value for this cluster features, as such I indicate that this is an approximation. This may indicate the need for more data to settle the feature parameters of the cluster. The 60% ranges of returns of two treatment policies from a good run with $M = 4$ (Figure 5.11) are shown to illustrate the impact the different clusters may have on the predicted outcomes. As we do not have the original model distributions, a subsample of 200 patients is drawn from the original dataset and their features used to visualize the outcome ranges vs. the probability of falling into the latent class with the highest estimated prior probability [4].

The interval predictions are shown in Figure 5.11. Patients with a higher probability of belonging in this latent class show an advantage to policy $\pi_1$: higher overall returns and tighter ranges than $\pi_2$. For patients with an almost 0 probability of being in this class, the opposite is true: $\pi_1$ now has extremely large ranges of possible returns, whereas $\pi_2$ seems to do quite well and have fairly tight ranges. There is substantially more uncertainty over the return of $\pi_1$ for patients with a near-zero probability of this class than for other patients or the other policy. This example illustrates exactly the sort of issue that I hoped the LSU approach can capture—adjusting the predicted returns for a policy based on differences in the latent class membership of different individuals, which can capture differences in the underlying models as well as differences

[4] To improve comparability, the graph includes only patients that had a baseline viral load of between 100–1000 copies/mL (start state of $s_{0,2}$). The latent class visualized has an estimated prior probability of 0.50, while the three other classes have probabilities of 0.23, 0.21 and 0.06, respectively.

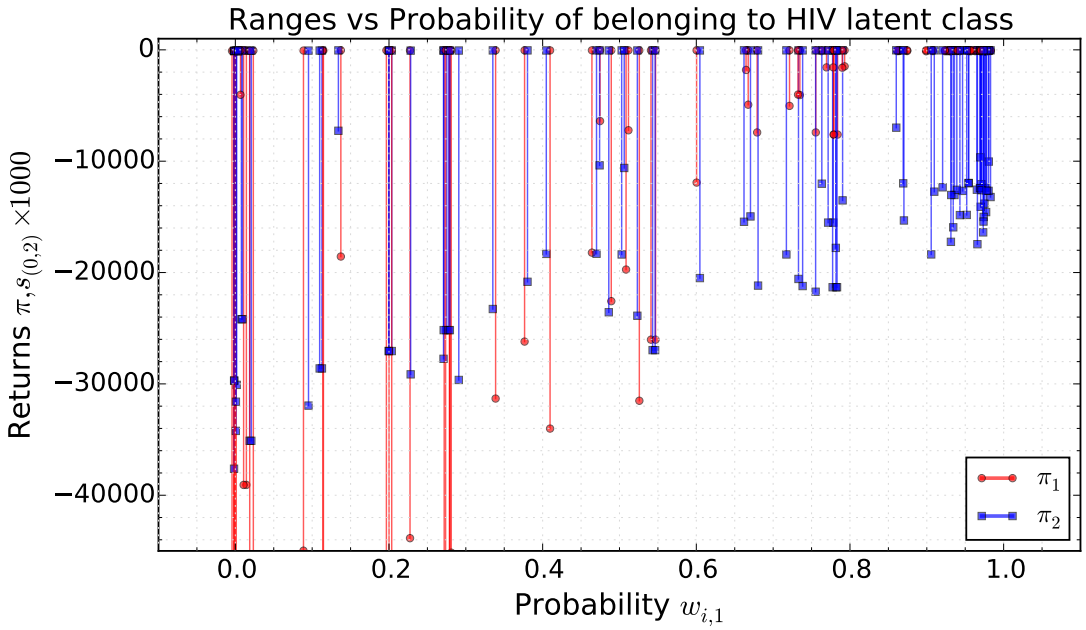


Figure 5.11: LSU intervals for 2 HIV therapy polices vs. probability of an individual being in the dominant latent class

in the amount of data about those classes.

### 5.5.2 Personalized fund-raising appeals

The second dataset the latent class approach was applied to is the donor history database of the Paralyzed Veterans of America (PVA). This dataset was originally used for the Knowledge Discovery and Data Mining Tools Competition in 1998 (Parsa, 1998). The dataset contains information about the PVA's direct mail appeal history with multiple donors. I will first describe the dataset and then present the results of applying the LSU algorithm to extract latent structure to improve policy evaluations.

**PVA donor database**

The PVA is a not-for-profit organization that provides programs and services for US veterans with spinal injuries or disease. They are one of the largest direct mail fund-raisers in the USA. The original goal of the release of this dataset was to predict whether donors would respond to a campaign or not given their donation history and information about the last 22 campaigns sent by PVA, covering the years 1994, 1995 and 1996. This dataset was presented as a classification problem. The dataset included information about campaigns sent prior to the campaign to be classified. From each of the donors, there were multiple demographic variables captured. For each of the campaigns, the date on which the campaign was sent, types of mailer sent, the date and amount of donation (if received) were also recorded.

For my analysis, the data are reformulated into a sequential decision problem. I would like to predict the range of cumulative donations received from donors in a certain period given different mailing policies. I use the Recency, Frequency and Amount (RFA) or Recency, Frequency and Monetary Value (RFM) formulation (Kahan, 1998) to encode the state the donors are in. The RFA summarizes the history of a donor. The original dataset had its own RFA formulation that was only available for the months in which campaigns were sent. To supplement this information, the RFA is recalculated on a monthly basis and adopted a simple RFA formulation. The RFA calculation is started at September 1993, with every donor having an RFA of [0,0,0]. For every month after that, until September 1996, I checked if any campaigns were sent (via the campaign information provided or the date of original gift recorded in the database) and whether donations were received for those donations or not. For simplicity, an action is defined as being a “sent campaign” or “no sent campaign” in each

month. If two campaigns were sent, they were classified as a single event instead of two.

Similar to the work of Mannor et al. (2007), the RFA components are discretized into 4 bins. For the recency (months since last gift), the discretization used is [0,1,4,8,27]. For the frequency (the number of gifts in the last 6 months), the discretization used is [0,0.99,1,2,6]. For the amount (average donations in the last 6 months), [0,1,10.5,16,1000]. There are numerous ways in which the RFA can be calculated and investigators have flexibility. This formulation was chosen for its simplicity. This discretization resulted in 18 accessible states and 2 actions (send a campaign or don't send a campaign). The donors' continuous features in the latent class MDP were: Age, number of donations that had been sent by donor before the start of the campaigns in the dataset, total amount that had been sent by donor before the start of the campaigns in the dataset. There are other features available in the dataset, especially pertaining to the demographics of the neighborhoods that the donors live. For this dissertation, I chose to focus on individual donor features; however, this analysis can be expanded further. For evaluation purposes, a policy is defined in this application area as a linear set of action choices over the 25 months (starting August 1994) in which outcomes will be predicted. For example, a policy could be: send a campaign every two months, or send a campaign to a donor every month. Each mailer costs 68 cents as provided in the dataset description. As such, if a campaign is sent to a donor and no gift is received, a reward of $-0.68$ is assigned. If a gift is received, 0.68 is subtracted from the amount of the donation. The return is calculated with $\gamma = 1.0$ and a maximum of 25 steps (25 months).

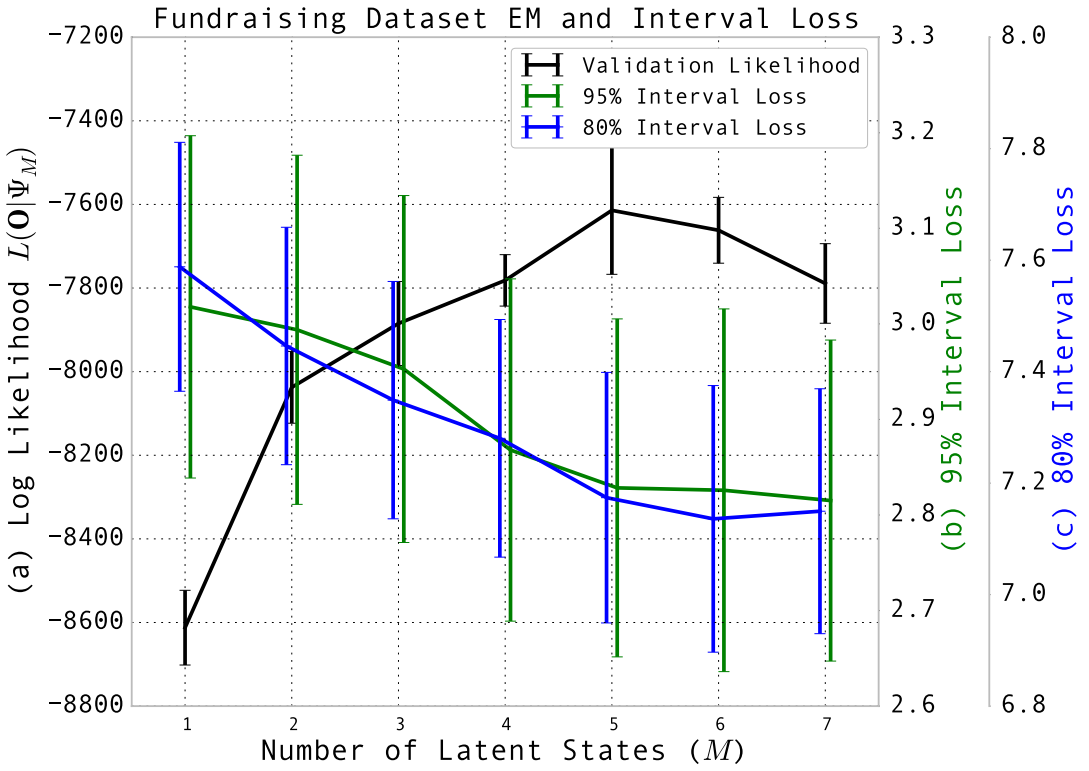


Figure 5.12: Likelihood and loss functions for the LSU algorithm with mixed observational data from the fund-raising dataset

**Latent class analysis**

Here, I present the results of running LSU on the fund-raising dataset. The data is randomly split into training set and validations set. After cleanup and removing any outliers, I use 10000 samples for training. An additional 400 data samples are used for EM validation and to calculate the loss function. The original dataset has ∼90000 donors, but a sample of 10000 is used for training and 400 for the validation and interval loss function calculation. The experiment was repeated 8 times. Figure 5.12 presents the the results of running LSU on the dataset.

The likelihood of the observations increases as we increase the number of

latent classes. Similarly, the 80% and 95% interval loss is reduced as we incorporate more latent classes. The 80% loss starts slightly increasing when we reach $M = 5$ latent classes. The likelihood of the data also slightly dips once we are at 6 underlying classes. So, as with the HIV dataset, modeling the latent structure leads to more accurate predictions.

Next, I display 60% ranges of returns of two different fund-raising appeal policies with $M = 5$. Policy $\pi_1$ sends appeals every four months and policy $\pi_2$ sends appeals every 2 months. I drew a subsample of 200 donors from the original dataset and used their features to visualize the ranges versus the probability of falling into one of the latent classes. Only individuals who had an initial starting state (RFA) of [0,0,0] are considered. The latent class visualized, *Cluster 1*, has an estimated prior probability of 0.24, while the four other classes have probabilities of 0.27,0.13,0.20 and 0.16, respectively. This latent class is composed of donors of 66 years of age, an average of $88 in previous donations and average of 11 previous donations sent. This class was chosen to illustrate the differences between it and other classes in terms of the range of expected returns.

The interval predictions are shown in Figure 5.13. As a donor's probability of belonging in this latent class gets nearer to 1.0, we see that policy $\pi_2$ has an advantage over $\pi_1$ and the ranges for policy $\pi_1$ are narrower. As we move from right to left, donors decreasing in their probability of falling into this latent class, the ranges for $\pi_1$ tend to be wider while the lower limits of $\pi_2$ tend to be higher.

Through analyzing multiple runs of the algorithm, we found a strong cluster, *Cluster 2*, made up of individuals who are around 57 years old and have sent no donations prior. This cluster, visualized in Figure 5.14, is more likely to send

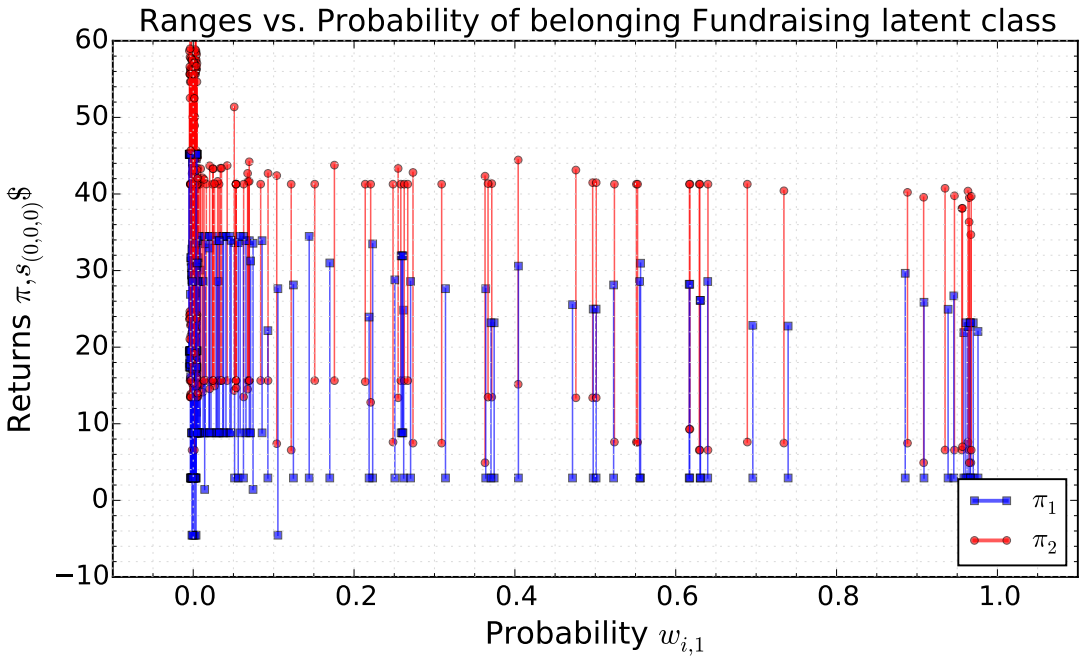


Figure 5.13: 60% returns of two policies versus being in donor Cluster 1

more donations and as such has higher upper and lower limits as compared to Cluster 1.

This section explored the application of LSU to real-world datasets. Through analyzing the results, we see that if there is latent structure in the data, the model can discover and exploit it given features from the individual. The latent clusters discovered allow us to better predict the outcomes of policies as evidenced by the reduction in the interval loss.

One natural next question is whether different classes have different best policies; intuitively, do different groups have better outcomes under different policies? The current loss function is not designed to perform partitions of the population into those with different best policies. Rather, it seeks to provide accurate predicted ranges of the effectiveness of a policy and capture differences among groups in this effectiveness. This second issue, which is tackled here, is still important to both decision makers and the individuals in question—two

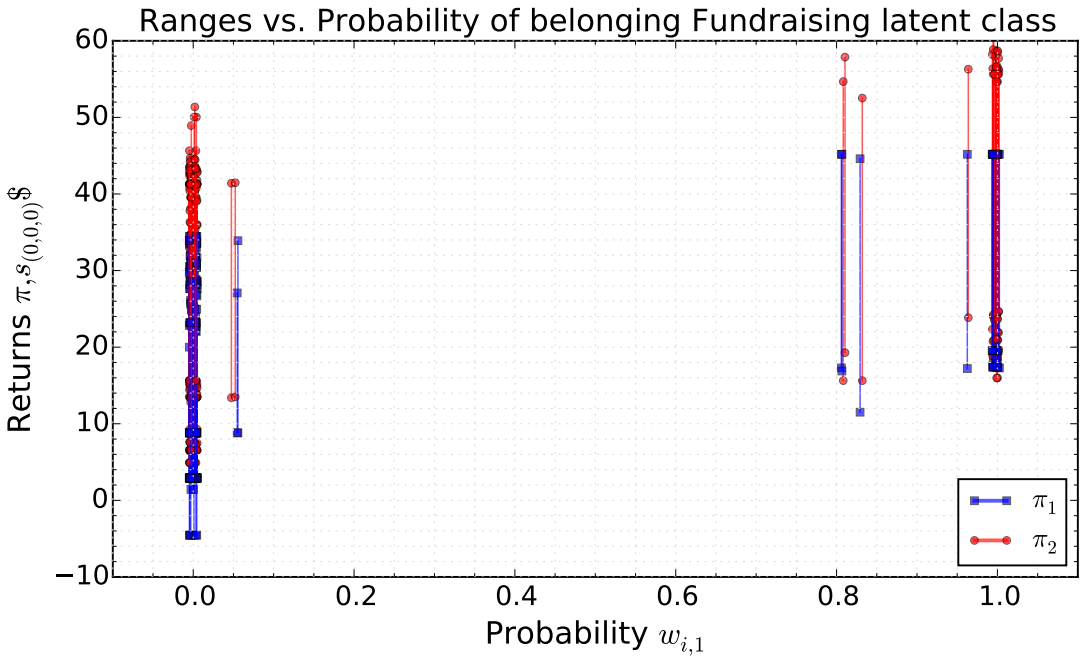


Figure 5.14: 60% returns of two policies versus being in donor Cluster 2

individuals may both have the same best intervention/treatment/policy, but one may expect his/her outcomes to be better than the other. In e-commerce, the same ad may be most preferred amongst customers, but one group of customers will only click on it 30% of the time, and another 50%, which greatly impacts a company's predicted revenue.

## 5.6 Summary and Discussion

I presented a novel method for using batch observational data to provide personalized estimates of the effectiveness of policies. The approach first searches for latent structure, and then quantifies uncertainty within and across the latent classes to compute effectiveness ranges for new individuals. I demonstrated that the approach helped capture important variability in HIV patient outcomes

as well as donation amounts of a fund-raising appeal, allowing us to make better predictions of the result of different policies. Such methods are of increasing importance given the huge number of potential applications (healthcare, e-commerce, online education, etc.) that could benefit from such algorithms.

One exciting next step is to expand the approach beyond policy evaluation to use uncertainty intervals to compute risk-dependent policy *recommendations* for individuals. A challenge for this direction, at least in the healthcare domain, is characterizing the individual's risk attitude—risk-seeking individuals prefer policies with a higher upside whereas risk-averse individuals prefer policies with a lower downside. Eliciting and incorporating this information into the search is an important problem to be resolved.

There is also an interesting question of how to leverage information gathered about the new individual as they respond to the proposed policy to refine the underlying estimates of their latent class and/or model parameter values. Framing the problem as a POMDP with a static latent class may be a fruitful way to start to address this issue. Finally, it would also be interesting to extend the model representation to handle continuous-state MDPs.

# Chapter 6
# Conclusion

In this chapter, I present concluding remarks as related to the thesis statement: *The applicability of reinforcement-learning methods to real-world challenges can be improved by novel evaluation methodologies, including online procedures for characterizing the capacity of reinforcement-learning algorithms and offline evaluation procedures that account for the uncertainties resulting from the use of noisy batch data*.

I first present a summary, draw some overarching conclusions, and finally suggest some areas of future work.

## 6.1 Summary

This dissertation's focus has been on improved evaluation methods to further the applicability of reinforcement learning in real-world challenges. The first challenge tackled was the development of online evaluation methods to better demonstrate the behavior and the capacity of reinforcement-learning algorithms with respect to the environments in which they are deployed (Chapter 3). By introducing the problem of meta-reinforcement learning, finding the best algorithm given a distribution of MDPs, we are able to focus our evaluation on how well sets of algorithms generalize to different MDPs. Two distinct evaluation methods were created. The first, using a uniform convergence bound, output a sample optimized generalization bound of the performance of a set

of algorithms given a sample of MDPs drawn from a larger MDP distribution. The second method, an application of cross-validation, provided a proxy to the sample optimized generalized bound that could be more useful for practitioners. Analyzing the application of these evaluation approaches, we show how one can detect overfitting and gain an intuition of the generalization error of a set of algorithms given only a sample of MDPs.

The second focus of the dissertation was offline evaluation of reinforcement-learning algorithms given batch data. Batch data allows us to tackle real-world problems that are harder to gain access to, to conduct online learning with or carry out online evaluation on. Ultimately, through all of our scientific pursuits, we would like to have our algorithms model real world behavior. In RL, that means we need to be able to deal with the shortcomings of using batch data to evaluate algorithms. First, the dissertation presented metrics that can be used to compare value-based reinforcement-learning algorithms (Chapter 4). A new metric, Relative Bellman Update Error, was developed and its properties presented. Ultimately, the offline metrics are hindered by a strong dependence on how the batch data is collected and how much of it is available. To deal with this shortcoming, one has to confront the uncertainty that arises when using batch data.

The latter part of the dissertation introduces a model-based evaluation method that tackles multiple sources of uncertainty resulting from using batch data (Chapter 5). I introduce the latent class MDP for capturing and quantifying the different sources of uncertainty. I introduce an expectation maximization-based algorithm for learn the parameters of the latent class MDP. To measure the accuracy of the ranges of outcomes predicted by the model, I introduce the

interval-based loss function. Through the application of the approach to a synthetic problem as well as two problems using real-world data, we explore the behavior of the approach. We observe that we increase our prediction accuracy when we find appropriate latent structure in the data we are modeling.

## 6.2 Conclusions

When researchers compare the performance of different algorithms, they wish to communicate to others the behavior of their algorithms. Through focusing on generalization in reinforcement-learning online evaluation, we discover that using single MDPs and tuned algorithms increases the chances of overfitting and further obscures the potential interactions between certain algorithms and certain MDPs. This challenge is not unique to RL but RL researchers have not actively focused on challenges relating to generalization. By using evaluation methods that characterize the capacity of algorithms and highlight the behavior of algorithms in relation to changing environment characteristics, we can better understand the limitations of these algorithms. The sample-optimized generalization bound provides theoretical underpinnings of relating a set of algorithms to an MDP distribution using only a sample of MDPs from that distribution. A more practical approach, cross-validation, still uses MDP samples from a distribution but is less conservative than the sample-optimized generalization bound. Both of these approaches can be used to detect over/under fitting of algorithms and provide information to researchers that can highlight the positive/negative aspects of choices of one algorithm set as opposed to another. This potentially increases the applicability of algorithms to more domains as researchers who would need RL algorithms would better understand their strengths and their shortcomings better.

To develop RL algorithms for real world problems, we need to have mechanisms for evaluating these algorithms on real world data. To do so, for multiple practical reasons, we have to use batch data. Metrics to compare value-based algorithms to each other on batch data are limited due to extreme sensitivity on the manner in which sampling is done. As such, a mechanism is needed for capturing the uncertainty inherent in using batch data for evaluation. By quantifying the uncertainty due to batch data, we can better evaluate policies resulting from algorithms using real-world data. As a demonstration, I applied these methods to two batch datasets captured in the context of medicine and marketing. The practical implication of this work is that it suggests we can create RL algorithms, train them using batch data and use the batch data to evaluate the outcomes of the policies resulting from the batch data. Having the ability to communicate ranges of outcomes of policies/interventions to domain experts can add valuable information in the sequential decision-making process. Dealing with real-world problems will extend the reach of reinforcement learning in general and the focus on quantifying uncertainty due to batch data is a small step in understanding a direction that can potentially bring in other disciplines and their challenges.

## 6.3 Future Work

Ultimately, given the new online evaluation approaches, there is an opportunity to create meta-reinforcement learning algorithms that are efficient. Earlier, we discussed the concept of mixture-ensembles as part of the sample optimized generalization bound (Section 3.3.1) and fusion ensembles as an example of an approach to create a meta-reinforcement learning algorithm (Section 3.4.2). If

we treat parameter tuning as a stochastic optimization problem, we can leverage the large body of work in relation to stochastic optimization (Spall, 2005; Goschin et al., 2013) as well as multi-armed bandits (Bubeck et al., 2011; Weinstein, 2014). Bringing these tools to bear would require augmenting existing RL algorithms; as such, might be attractive to practitioners as they would be applications of existing optimization techniques on the RL algorithm parameter space.

Recently, there has been more attention paid to creating RL algorithms in the meta-RL setting. Of particular interest is the recent combination of Deep Learning (Hinton et al., 2006) and reinforcement learning to tackle the Atari arcade environment Mnih et al. (2013). The original approach to learning in the Arcade environment involved using a small subset of video game MDPs to set parameters (training) for the reinforcement-learning algorithms. With the parameters set, the RL algorithms were then deployed on a larger set of test video game MDPs (Bellemare et al., 2013). Human-competitive performance has been achieved in these large problems, supporting the idea that evaluation methods like the meta-RL framework have an important role to play in scaling up RL approaches to complex real-world problems. I believe these published results should be applauded. I do not mean to diminish their impact, but I believe that there is still more to gain by varying the size of the training set, varying the specific MDPs, and measuring/approximating the generalization error with the larger testing set.

An obstacle to the adoption of evaluation methods that account for generalization in reinforcement learning will be the computational cost involved. In this dissertation, I presented two example environments that are small in comparison to the state-of-the-art benchmarks. Even so, overfitting is still very

much a danger that researchers have to be aware of, and finding efficient ways to compute evaluation statistics that can then be analyzed for generalization power will be critical.

The latent class MDP presented in this dissertation modeled the membership of sub-populations as well as transition models belonging to each of a set of latent populations. For simplicity, however, it did not model uncertainty in the reward function. An extension of this work would be to investigate the impact of having different reward functions for each sub-population and modeling the uncertainty that results. We can think of a situation in which one has two patients who belong to two subpopulations, their transition functions as per their transition models might be different but their ultimate reward functions as per the same states and actions might be slightly different as well. In the presented model, the reward function is ultimately intertwined with modeling choices made by a researcher and, as such, in some cases it might have no impact at all. In others, however, these choices might reveal patterns that can be used to improve the model.

Given an offline evaluation method that can provide the range of outcomes of a policy given batch data, a second set of questions arise: Can we we find the optimal policy given the uncertainty? Can we find an optimal policy with regard to the latent classes in the MDP? To do so, we believe one would have to incorporate the concept of risk in the policy-creation step. Concepts such as prospect theory (Kahneman and Tversky, 1979) could be incorporated into the step to find the optimal policy given different risk profiles that a practitioner is considering.

This dissertation has explored new evaluation methods in trying to increase the applicability of reinforcement learning to real-world challenges. What we

measure should ultimately assist us in understanding the real-world around us as well allow us to make better choices in how we model this real-world. This dissertation is but a small step in improving our understanding of some of the challenges reinforcement learning faces in expanding its reach to real-world challenges.

# Bibliography


András Antos, Csaba Szepesvári, and Rémi Munos. Learning near-optimal policies with Bellman-residual minimization based fitted policy iteration and a single sample path. *Machine Learning*, pages 89–129, 2008.

Christopher G Atkeson, Andrew W Moore, and Stefan Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML 1995)*, pages 30–37. 1995.

Peter L Bartlett and Shahar Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. *The Journal of Machine Learning Research*, 3:463–482, 2003.

Stephen D. Bay, Dennis Kibler, Michael J. Pazzani, and Padhraic Smyth. The UCI KDD archive of large data sets for data mining research and experimentation. *ACM SIGKDD Explorations Newsletter - Special issue on "Scalable data mining algorithms"*, pages 81–85, 2000.

Robert M. Bell, Yehuda Koren, and Chris Volinsky. All together now: A perspective on the Netflix Prize. *Chance*, 23(1):24–29, 2010.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: an evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47(1):253–279, 2013.

Richard Ernest Bellman. A markovian decision process. 1957.

Jeff A Bilmes. A gentle tutorial of the EM algorithm and its application to parameter estimation for Gaussian mixture and hidden Markov models. *International Computer Science Institute*, 4:126, 1998.

J.A. Boyan. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49(2):233–246, 2002.

Justin A. Boyan and Andrew W. Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in Neural Information Processing Systems 7*, pages 369–376. 1995.

R.I. Brafman and M. Tennenholtz. R-max-a general polynomial time algorithm for near-optimal reinforcement learning. *The Journal of Machine Learning Research*, 3:213–231, 2003.

R Scott Braithwaite, Mark S Roberts, Chung Chou H Chang, Matthew Bidwell Goetz, Cynthia L Gibert, Maria C Rodriguez-Barradas, Steven Shechter, Andrew Schaefer, Kimberly Nucifora, Robert Koppenhaver, et al. Influence of alternative thresholds for initiating HIV treatment on quality-adjusted life expectancy: A decision model. *Annals of Internal Medicine*, 148(3):178–185, 2008.

Sébastien Bubeck, Rémi Munos, and Gilles Stoltz. Pure exploration in finitely-armed and continuous-armed bandits. *Theoretical Computer Science*, 412(19):1832–1852, 2011.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement learning and dynamic programming using function approximators*. CRC press, 2010.

Rich Caruana and Alexandru Niculescu-Mizil. Data mining in metric space: an empirical analysis of supervised learning performance criteria. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–78. ACM, 2004.

Hsinchun Chen, Roger HL Chiang, and Veda C Storey. Business intelligence and analytics: From big data to big impact. *MIS quarterly*, 36(4):1165–1188, 2012.

William Dabney, Philip Thomas, and Andrew Barto. Performance metrics for reinforcement learning algorithms. 2013. Unpublished presentation at RLDM'13 Workshop.

Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)*, 27(3):326–327, 1995.

Chris Drummond. Machine learning an experimental science (revisited). Technical report, AAAI Workshop on Evauation Methods for Machine Learning, 2006.

M. Dudik, J. Langford, and L. Li. Doubly robust policy evaluation and learning. *Proceedings of the 28th International Conference on Machine Learning (ICML 2011)*, 2011.

Damien Ernst, G-B Stan, J Gongalves, and Louis Wehenkel. Clinical data based optimal STI strategies for HIV: A reinforcement learning approach. In *Decision and Control, 2006 45th IEEE Conference on*, pages 667–672. 2006.

Amir-massoud Farahmand, Doina Precup, and Mohammad Ghavamzadeh. Generalized classification-based approximate policy iteration. In *Tenth European Workshop on Reinforcement Learning (EWRL)*, volume 2. 2012.

D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A.A. Kalyanpur, A. Lally, J.W. Murdock, E. Nyberg, J. Prager, et al. Building watson: An overview of the deepqa projectverview of the deepqa project. *AI Magazine*, 31(3):59–79, 2010.

Radu Florian and David Yarowsky. Modeling consensus: classifier combination for word sense disambiguation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing*, volume 10, pages 25–32. 2002.

R. Fonteneau, S. Murphy, L. Wehenkel, and D. Ernst. Model-free Monte Carlo-like policy evaluation. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*. 2010.

Sergiu Goschin, Ari Weinstein, and Michael Littman. The cross-entropy method optimizes for quantiles. In *Proceedings of The 30th International Conference on Machine Learning (ICML 2013)*, pages 1193–1201. 2013.

Geoffrey Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Nicholas K. Jong and Peter Stone. Model-based function approximation in reinforcement learning. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007)*, page 95. 2007.

Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

Ron Kahan. Using database marketing techniques to enhance your one-to-one marketing initiatives. *Journal of Consumer Marketing*, 15(5):491–493, 1998.

Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica: Journal of the Econometric Society*, pages 263–291, 1979.

Robert Kleinberg, Aleksandrs Slivkins, and Eli Upfal. Multi-armed bandits in metric spaces. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 681–690. ACM, 2008.

Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, volume 14, pages 1137–1145. 1995.

J.Z. Kolter and A.Y. Ng. Regularization and feature selection in least-squares temporal difference learning. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML 2009)*, pages 521–528. ACM, 2009.

G. Konidaris, S. Osentoski, and PS Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2008)*, pages 380–385. 2008.

M.G. Lagoudakis and R. Parr. Least-squares policy iteration. *The Journal of Machine Learning Research*, pages 1107–1149, 2003.

Pat Langley. Machine learning as an experimental science. *Machine Learning*, 3(1):5–8, 1988.

Pat Langley. Crafting papers on machine learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 1207–1216. 2000.

Pat Langley and Dennis Kibler. The experimental study of machine learning. *Unpublished paper*, 1991.

B.R. Leffler, M.L. Littman, and T. Edmunds. Efficient reinforcement learning with relocatable action models. In *Proceedings of the National Conference on Artificial Intelligence*, page 572. 2007.

M. Lewis. A dynamic pricing approach to customer relationship pricing. *Management Science*, 51(6):986–994, 2005.

L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining*, pages 297–306. 2011.

L.J. Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, pages 293–321, 1992.

M.L. Littman, G.A. Keim, and N. Shazeer. A probabilistic approach to solving crossword puzzles. *Artificial Intelligence*, 134(1-2):23–55, 2002.

Shie Mannor, Duncan Simester, Peng Sun, and John N Tsitsiklis. Bias and variance approximation in value function estimates. *Management Science*, 53(2):308–322, 2007.

Vukosi Ntsakisi Marivate, Jessica Chemali, Emma Brunskill, and Michael Littman. Quantifying uncertainty in batch personalized sequential decision making. In *Workshops at the Twenty-Eighth AAAI Conference on Artificial Intelligence*. 2014.

Vukosi Ntsakisi Marivate and Michael L Littman. An ensemble of linearly combined reinforcement-learning agents. In *AAAI (Late-Breaking Developments)*. 2013.

Viktor Mayer-Schönberger and Kenneth Cukier. *Big data: A revolution that will transform how we live, work, and think*. Houghton Mifflin Harcourt, 2013.

Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. MIT press, 2012.

Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Nonparametric return distribution approximation for reinforcement learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 799–806. 2010a.

Tetsuro Morimura, Masashi Sugiyama, Hisashi Kashima, Hirotaka Hachiya, and Toshiyuki Tanaka. Parametric return density estimation for reinforcement learning. *In Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence (UAI 2010)*, 2010b.

Travis B Murdoch and Allan S Detsky. The inevitable application of big data to health care. *Journal of the American Medical Association*, 309(13):1351–1352, 2013.

Kevin P Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

DJ Newman, S. Hettich, CLS Blake, and CJ Merz. UCI Repository of Machine Learning Database, Irvine, CA: University of California, Dept. of Information and Computer Science. 1998.

Andrew Y Ng, Adam Coates, Mark Diel, Varun Ganapathi, Jamie Schulte, Ben Tse, Eric Berger, and Eric Liang. Autonomous inverted helicopter flight via reinforcement learning. In *Experimental Robotics IX*, pages 363–372. 2006.

A. Nouri, M.L. Littman, L. Li, R. Parr, C. Painter-Wakefield, and G. Taylor. A novel benchmark methodology and data repository for real-life reinforcement learning. In *Proceedings of the 26th International Conference on Machine Learning (ICML 2009)*. 2009.

Dirk Ormoneit and Śaunak Sen. Kernel-based reinforcement learning. *Machine Learning*, pages 161–178, 2002.

Ismail Parsa. KDD cup 1998 data. 1998. URL `http://www.kdnuggets.com/meetings/kdd98/kdd-cup-98.html`.

R. Polikar. Ensemble based systems in decision making. *Circuits and Systems Magazine, IEEE*, 6(3):21–45, 2006.

E. Porteus. Conditions for characterizing the structure of optimal strategies in infinite-horizon dynamic programs. *Journal of Optimization Theory and Applications*, pages 419–432, 1982.

Mattia CF Prosperi, Andre Altmann, Michal Rosen-Zvi, Ehud Aharoni, Gabor Borgulya, Fulop Bazso, Anders Sönnerborg, Eugen Schülter, Daniel Struck, Giovanni Ulivi, et al. Investigation of expert rule bases, logistic regression, and non-linear machine learning techniques for predicting response to antiretroviral treatment. *Antiviral Therapy*, 14(3):433–42, 2009.

L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1):1–39, 2010.

Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3rd Edition)*. Pearson Education, 2010.

Steven M Shechter, Matthew D Bailey, Andrew J Schaefer, and Mark S Roberts. The optimal time to initiate HIV therapy under ordered health states. *Operations Research*, 56(1):20–33, 2008.

S.M. Shortreed, E. Laber, D.J. Lizotte, T.S. Stroup, J. Pineau, and S.A. Murphy. Informing sequential clinical decision-making through reinforcement learning: an empirical study. *Machine Learning*, 84(1):109–136, 2011.

George Siemens and Phil Long. Penetrating the fog: Analytics in learning and education. *Educause Review*, 46(5):30–32, 2011.

S.P. Singh and R.C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, pages 227–233, 1994.

James C Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*. John Wiley & Sons, 2005.

Walter F Stewart, Jason Roy, Jimeng Sun, and Shahram Ebadollahi. Clinical utility of machine learning and longitudinal EHR data. In *Machine Learning in Healthcare Informatics*, pages 209–227. 2014.

Alexander L Strehl and Michael L Littman. A theoretical analysis of model-based interval estimation. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 856–863. 2005.

M. Strens. A Bayesian framework for reinforcement learning. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 943–950. 2000.

R. Sun and T. Peterson. Multi-agent reinforcement learning: weighting and partitioning. *Neural networks*, 12(4-5):727–753, 1999.

R.S. Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44, 1988.

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

Aviv Tamar, Dotan Di Castro, and Shie Mannor. Policy evaluation with variance related risk criteria in markov decision processes. In *JMLR Workshop and Conference Proceedings*, volume 28, pages 495–503. 2013.

Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

J.N. Tsitsiklis and B. Van Roy. An analysis of temporal-difference learning with function approximation. *Automatic Control, IEEE Transactions on*, 42(5):674–690, 1997.

HP van Hasselt. Insights in reinforcement learning: formal analysis and empirical evaluation of temporal-difference learning algorithms. *SIKS dissertation series*, 2011(04), 2011.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

Ari Weinstein. *Local Planning for Continuous Markov Decision Processes*. Ph.D. thesis, Rutgers, The State University of New Jersey, 2014.

Robin A Weiss. How does HIV cause AIDS? *Science*, 260(5112):1273–1279, 1993.

S. Whiteson, B. Tanner, ME Taylor, and P. Stone. Generalized domains for empirical evaluations in reinforcement learning. In *Proceedings of the 4th workshop on Evaluation Methods for Machine Learning at ICML-09*. 2009.

S. Whiteson, B. Tanner, and A. White. The reinforcement learning competitions. *AI Magazine*, 31(2):81–94, 2010.

Shimon Whiteson, Brian Tanner, Matthew E Taylor, and Peter Stone. Protecting against evaluation overfitting in empirical reinforcement learning. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 120–127. IEEE, 2011.

M.A. Wiering and H. van Hasselt. Ensemble algorithms in reinforcement learning. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 38(4):930–936, 2008.

Ronald J. Williams and Leemon C. Baird, III. Analysis of some incremental variants of policy iteration: First steps toward understanding actor-critic learning systems. Technical report, Northeastern University, College of Computer Science, 1993.

Jason Wolfe, Aria Haghighi, and Dan Klein. Fully distributed em for very large datasets. In *Proceedings of the 25th international conference on Machine learning*, pages 1184–1191. ACM, 2008.

L. Xu, H.H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In *24th AAAI Conference on Artificial Intelligence (AAAI 2010)*, pages 210–216. 2010.

Illhoi Yoo, Patricia Alafaireet, Miroslav Marinov, Keila Pena-Hernandez, Rajitha Gopidi, Jia-Fu Chang, and Lei Hua. Data mining in healthcare and biomedicine: A survey of the literature. *Journal of Medical Systems*, 36(4):2431–2448, 2012.

M. Zazzi, F. Incardona, M. Rosen-Zvi, M. Prosperi, T. Lengauer, A. Altmann, A. Sonnerborg, T. Lavee, E. Schülter, and R. Kaiser. Predicting response to antiretroviral treatment by machine learning: The EuResist project. *Intervirology*, 55:123–127, 2012.