**University of Bath**

**UNIVERSITY OF BATH**

**PHD**

**Bounded Rationality in Reinforcement Learning**
**(Alternative Format Thesis)**

Lichtenberg, Jan Malte

*Award date:*
2023

*Awarding institution:*
University of Bath

[Link to publication](#)

**Alternative formats**
If you require this document in an alternative format, please contact:
openaccess@bath.ac.uk

# BOUNDED RATIONALITY *in* REINFORCEMENT LEARNING

Jan Malte Lichtenberg

A thesis submitted for the degree of Doctor of Philosophy
University of Bath
Department of Computer Science
December 2021

DECLARATION OF ANY PREVIOUS SUBMISSION OF THE WORK

The material presented here for examination for the award of a higher degree by research has not been incorporated into a submission for another degree.
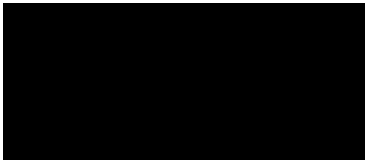
Jan Malte Lichtenberg

DECLARATION OF AUTHORSHIP

I am the author of this thesis, and the work described therein was carried out by myself personally.
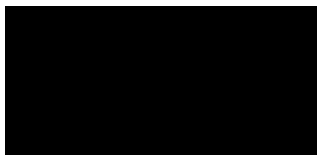
Jan Malte Lichtenberg

ABSTRACT

The broad problem I address in this dissertation is the design of autonomous agents that can efficiently learn goal-directed behavior in sequential decision-making problems under uncertainty. I investigate how certain models of bounded rationality—simple decision-making models that take into consideration the limited cognitive abilities of biological and artificial minds—can inform reinforcement learning algorithms to produce more resource-efficient agents. In the two main parts of this dissertation I use different existing models of bounded rationality to address different resource limitations present in sequential decision-making problems. In the first part I introduce a boundedly rational function approximation architecture for reinforcement learning agents to reduce the amount of training data required to learn a useful behavioral policy. In the second part I investigate how Herbert A. Simon's satisficing strategy can be applied in sequential decision making problems to reduce the computational effort of the action-selection process.

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF ALGORITHMS

Part I

INTRODUCTION

1

OVERVIEW

This dissertation investigates how simple models of bounded rationality can inform reinforcement learning algorithms to produce more resource-efficient agents.

In the following sections I argue that modern reinforcement learning is rooted in the pursuit of *perfect rationality*, discuss two limitations of this approach, and outline how algorithms inspired by the alternative notion of *bounded rationality* are able to address these limitations.

## 1.1  REINFORCEMENT LEARNING & THE TALE OF THE OPTIMAL POLICY

The broader goal of artificial intelligence considered in this dissertation is that of building agents that produce useful behavior using limited resources.[1] I focus on agents that learn useful behavior autonomously, that is, without being given explicit instructions about what to do in which situation.

One way to enable an agent to learn autonomously is to provide feedback on the usefulness of its behavior (rather than on the correctness of its actions). This feedback is usually given in the form of a numerical signal, called *reward*.[2] For example, a chess-playing agent could receive a positive reward for a game-winning move but a lower (or negative) reward for a move that immediately leads to the opponent's victory. Similarly, a self-driving car that crashes into a wall could receive a negative reward, while it could receive a positive reward for, say, every second that it drives safely.

The general idea behind learning from rewards is simple. By repeatedly interacting with its environment, the agent gathers experience on a) which actions yield which rewards in which kind of situations, and b) how the agent's environment changes as a function of the agent's actions. If the agent learns to choose actions that lead to high rewards or to states of the environment that lead to high rewards in the future, then the resulting behavior is useful—because this is how the rewards have been designed in the first place.

And yet, learning from rewards is difficult. In general, the reward is only known *after* the action has been executed. Also, every action not only yields an immediate reward but also changes the agent's situation in the environment and thus affects future rewards as well. Furthermore, in some domains meaningful rewards are extremely rare, resulting in a large temporal delay between the exe-

[1] As opposed to, for example, building agents that imitate human decision making. See Russell and Norvig (2010) for a comparison of different goals in artificial intelligence research.

[2] See Sutton and Barto (2018, Section 1.7) for a history of research on learning from rewards.

cution of an action and the corresponding feedback. For example, in chess, the first and only reward occurs at the end of the game. The agent has to learn good opening moves as well as end-game behavior from this single reward only.

In view of these difficulties, modern reinforcement learning has turned to one approach in particular: optimization of long-term reward. According to Sutton and Barto (2018, p. 1), "reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal". A mapping from situations (or states) to actions is called a *policy*. A policy that maximizes some notion of cumulative reward received by an agent over the course of a life time is called an *optimal policy*. Reinforcement learning, so to speak, is the quest for an optimal policy.

The formalization of the learning objective as a reward optimization problem is intuitively appealing: if the optimization procedure is carried out successfully, the agent learns the *most useful* behavior. Using optimization also has practical advantages. Mathematical optimization is a well-established discipline and provides a large range of proven and well-studied optimization algorithms. In fact, most existing reinforcement learning algorithms are at their core optimization algorithms from convex optimization or optimal control theory,[3] adapted to the problem of learning an optimal policy from delayed rewards.

These adapted optimization algorithms have led to impressive advances in learning how to play challenging two-player games with sparse rewards such as Backgammon,[4] Chess, and Go;[5] learning how to play video games such as Tetris[6] or ATARI games;[7] and learning to solve continuous control and robotics problems.[8]

And yet, existing reinforcement learning algorithms have seen limited adoption to real-life tasks.[9] Next I describe two limitations of the reward-optimization approach that contribute to this lack of adoption.

A. *Reward optimization requires too much data.* To choose optimally is to choose an action that maximizes long-term value. Unfortunately, the values of available actions are hardly ever known before the decision is made. They are usually estimated by a mathematical model, called a *value function*, which takes a description of a given action as input and outputs an approximation of the action's value. In the context of reward optimization, the goal then becomes to approximate the true action values as closely as possible. Most reinforcement learning algorithms use linear functions or artificial neural networks for value function approximation. Whilst these models are indeed capable of producing highly accurate value estimates, they require substantial amounts of data for training—data that is not provided but has to be arduously collected by the agent through exploration in the environment.

The problem is the following. To estimate a good value function, the agent requires large amounts of high-quality data. However, to collect more high-quality data, the agent requires a somewhat good value function in the first place. Many reinforcement learning algorithms are guaranteed to eventu-

[3] Policy gradient methods are hill-climbing algorithms. Value-based reinforcement learning algorithms are approximations of dynamic programming algorithms (Bellman, 1957). See also Section 2.3.

[4] Tesauro (1992, 2002)
[5] Silver et al. (2017b,a)
[6] Gabillon et al. (2013)
[7] Mnih et al. (2015)
[8] Lillicrap et al. (2015)
[9] Dulac-Arnold et al. (2019)

ally escape from this gridlock situation. However, they may require prohibitive amounts of time and computation to do so in many complex real-world tasks.

Note that when an accurate simulator of the environment is available, the required data can simply be created through simulation. And indeed, all successful reinforcement learning applications mentioned above use simulation to quickly generate huge amounts of data. However, rather than eliminating the problem altogether, simulation merely transforms the data problem into a computational problem (thus further increasing the computational burdens described in Limitation B further below). For example, the CBMPI[10] algorithm requires over 20 million calls to a Tetris simulator to generate the data required to learn eight parameters of a linear expert policy. An even more extreme example is AlphaGo Zero,[11] which learned its policy using information gathered throughout 29 million games of Go.[12]

Despite its horrendous computational costs, simulation can be an effective way to create enough high-quality data for learning purposes. In many real-world domains, however, the agent does not have access to an accurate or cheap-enough simulator,[13] underpinning the need for reinforcement learning algorithms that can learn useful behavior from few(er) data points.

B. *Reward optimization is too computationally expensive in large action spaces.* When faced with a decision, a reward-maximizing agent has to consider and evaluate every available action in order to select the action that is believed to maximize reward. The total amount of computational resources required for one such deliberation is thus proportional to the product of the *number of available actions* and the *computational resources required to evaluate one action*. The values of both variables have to be kept small for the reward-maximizing approach to be feasible.

As described in Limitation A, recent reinforcement learning algorithms have seen a rise in the complexity of action-evaluation functions, and thus a rise in the need for computational resources to evaluate a single action. To keep learning feasible, these algorithms were employed and evaluated in environments with relatively low-dimensional action spaces. For example, a maximum number of 17 actions is available in any of the video games from the popular "ATARI 2600" evaluation benchmark;[14] the maximum action size in the game of Go with original board size is 361.

The "action space of real life", on the other hand, is enormous. Not only do we have thousands of possible actions to choose from in every instant, but the identities of available actions change between different moments and situations. Moreover, the number of available actions increases even further if temporally extended action plans (also called macro actions[15] or skills[16]) are considered as well. In many real-life domains, it is currently simply impracticable to consider and evaluate all actions before making a decision.

[10] Scherrer et al. (2015)

[11] Silver et al. (2017b)

[12] It was estimated that the experiments required approximately 4.8 million TPU computation hours (Rocke, 2019). The cost of one TPU computation hour as of 2021 is about $4.50, resulting in estimated computational costs of over $20 million to replicate the experiments.

[13] Think, for example, of a drug-discovery application, in which effects and side-effects of a new drug can only be observed by giving the drug to biological organisms due to a lack of accurate simulations of entire biological organisms.

[14] Bellemare et al. (2013)

[15] Amarel (1968)
[16] Thrun and Schwartz (1995) and Şimşek (2008)

This dissertation addresses the limitations just described by exploring approaches to reinforcement learning that dispose with the idea of (approximate) reward optimization at all times. Instead, the approaches explored here build on the notion of bounded rationality, which is introduced in the next section.

## 1.2   NOTIONS OF RATIONAL BEHAVIOR

To act rationally is "to do the right thing".[17]   Different notions of rationality therefore refer to different interpretations of what it means to do the right thing.

The reward-maximizing approach to reinforcement learning presented in the previous section views a rational agent as one that behaves optimally in every instant. This view is consistent with the prevalent paradigm of rational choice in economics,[18] game theory,[19] and statistical decision theory:[20] the *maximum expected utility* principle postulates that decision makers should consider all available alternatives and choose the alternative that maximizes a subjective expected utility function. In artificial intelligence research, this notion of rationality is called *perfect rationality*.[21]   In economics, perfect rationality has also been called *objective rationality*[22], *full rationality*[23], or simply *rational choice theory*.

Rationality plays a key role in economics, where one of the main goals is to model how economic entities (people, firms, and countries) react to economic policy interventions. For example, it may be of interest to estimate the change in tax revenue of a country if the value-added tax is increased by, say, 5%. One way to estimate the overall effect is to first estimate how each individual would change its consumption behavior as a function of the changed tax rate, followed by an aggregation of the individual results. Neo-classical economic theory is built on the assumption of perfect rationality and thus assumes that individuals behave according to the maximum expected utility principle. In our tax example this would mean that every individual reconsiders their economic choices by recalculating the long-term utilities of all available alternatives under the modified tax rate before making any new economic decisions. Is this an accurate (or at least useful) assumption to make?

It is not according to Herbert Simon,[24] who in the 1950s considered the maximum expected utility principle to be an inadequate assumption to describe human economic decision making. Simon (1957, p. 198) formulated, what he called, the *principle of bounded rationality*:

> The capacity of the human mind for formulating and solving complex problems is very small compared with the size of the problems whose solution is required for objectively rational behavior in the real world—or even for a reasonable approximation to such objective rationality.

According to Simon, human decision making is instead a search process that is guided by aspiration levels. An aspiration level is a value of some objective function that must be reached or surpassed by a satisfactory decision alternative.[25] Alternatives are considered sequentially and the search is stopped as soon as

[17] Russell and Norvig (2010, p. 37)

[18] Stigler (1961)
[19] Von Neumann and Morgenstern (1944)
[20] Savage (1972)

[21] Russell and Norvig (2010)
[22] Simon (1956)
[23] Selten (1998)

[24] Apart from being a Nobel Prize-winning economist, cognitive psychologist and political scientist, Herbert Simon would later also become a pioneering figure in artificial intelligence research.

[25] The term "aspiration level" stems from experimental psychology, where similar concepts have been studied before Simon's work (for example, see Lewin et al., 1944; Sauermann and Selten, 1962).

a satisfactory alternative is found, a process for which Simon coined the term *satisficing*.[26]

More generally, a model of bounded rationality is any predictive model that takes into consideration the computational limitations of the agent's mind and the limited data available to estimate the model's parameters. Many models are boundedly rational by deliberately ignoring some of the available information or combining different pieces of information in simple ways, for example, by giving them equal weight. Often these models do not need to be defined as the solution to a mathematical optimization problem but instead can be described by simple, cognitively plausible processes such as counting, ordering, and pairwise comparisons of values.[27]

Simply put, "doing the right thing" for a boundedly rational, satisficing[28] agent is to find a "good enough" solution, whereas the perfectly rational, optimizing agent aims to find "the best" solution.

## 1.3   BOUNDED RATIONALITY IN ARTIFICIAL INTELLIGENCE

Simon's critique of perfect rationality spawned an ongoing debate about which notion of rationality is best suited to describe human decision making in both economics[29] and psychology.[30] The goal proclaimed at the beginning of this chapter, however, was to build agents that produce useful behavior, not to imitate human decision making—so why should we even care about descriptive models of human decision making?

Humans and animals are still the most resource-efficient general-purpose decision makers in complex real-world environments known to us. This resource efficiency is unlikely to be due to superior computational power or data-storage capacities in biological brains (when compared to today's supercomputers), but rather due to the successful adaption of our decision-making strategies to real-world decision environments throughout the course of evolution. If we manage to transfer elements of these decision-making strategies from biological organisms to artificial decision makers, we can hope to increase the resource efficiency of the latter.

One has to keep in mind that the physical architectures of most of today's artificial agents (that is, computers) are substantially different in form and function from biological decision making architectures such as animal brains. Trying to imitate and implement human decision-making strategies on the level of bio-chemical processes present in biological brains is a daunting task. We therefore step up several levels of abstraction and try to make the transfer from the biological to the artificial on the level of cognitive processes.

Economics and psychology have brought forward a large canon of cognitive decision making models, including some that are inspired by perfect rationality (such as, for example, expected utility theory or prospect theory[31]) and some that are inspired by bounded rationality (such as Simon's satisficing strategy or fast-and-frugal decision heuristics[32]).

[26] Simon (1959)

[27] Katsikopoulos et al. (2020)

[28] The term "satisficing" will have two meanings throughout this dissertation. It will refer to Simon's satisficing decision making strategy that uses aspiration levels to decide whether an alternative is deemed satisfactory or not. More generally, "satisficing" will also be understood as an alternative to "optimizing".

[29] See Chapters 1 and 2 of Gigerenzer and Selten (2002) for a history of bounded rationality in both economics and psychology.

[30] Gigerenzer et al. (1999) and Oaksford and Chater (2007)

[31] Kahneman and Tversky (2013)

[32] Gigerenzer and Goldstein (1996) and Gigerenzer et al. (1999). See also Section 2.2

As outlined in the previous two sections, modern reinforcement learning algorithms are predominantly built on the idea of perfect rationality. In this dissertation, I substitute various parts of reinforcement learning algorithms that are inspired by the maximum expected utility principle with models of bounded rationality. This leads to the following thesis, which I aim to defend in this dissertation.

> *In situations characterized by few available data or scarce computational resources, models of bounded rationality can help reinforcement learning algorithms to be more resource-efficient,*

where resource efficiency will manifest itself in two ways throughout this dissertation. First, a boundedly rational algorithm is more resource-efficient than its perfectly rational counterpart if it requires fewer data (Limitation A) or less computational effort (Limitation B) to achieve the same or better performance. Second, a boundedly rational algorithm is more resource-efficient than its perfectly rational counterpart if it achieves better performance while requiring the same or fewer resources.

The following section outlines the contents of this dissertation and summarizes the contributions that provide supporting evidence for my thesis.

## 1.4 OUTLINE & CONTRIBUTIONS

This introductory part of the dissertation is concluded by Chapter 2, which provides technical background and discusses existing work that accounts for limited resources in artificial intelligence research.

The rest of the dissertation is broken into two largely independent parts. The two approaches presented in these parts build on different types of boundedly rational models from the literature to address different resource limitations present in sequential decision-making processes.

▸ PART II: BOUNDEDLY RATIONAL ACTION EVALUATION. In this part I explore the usefulness of boundedly rational predictive models as function approximators in reinforcement learning. This part addresses Limitation A by reducing the amount of data required to learn a useful policy.

Using function approximation to approximate a policy or value function in reinforcement learning can be interpreted as a series of supervised learning tasks, with the particularity that the training data in a given iteration is created by the agent itself, using the policy learned in the previous iteration. Most reinforcement algorithms in the literature use data-hungry supervised learning models such as unconstrained linear functions or neural networks for function approximation.[33] In supervised learning these complex models can produce excellent predictions when they are trained on large amounts of high quality data. However, a reinforcement learning agent that uses such a complex function approximator throughout the entire learning process fails to take into account that

[33] Sutton and Barto (2018, Part II)

amount and quality of data is limited in the beginning of the learning process. This can result in a slow and tedious learning process as described in more detail in Limitation A further above.

A growing body of work has produced evidence that many models of bounded rationality can rival, and sometimes outperform, more complex statistical models across a wide range of supervised learning tasks.[34] This leads to the following questions. *How can we use existing models of bounded rationality from supervised learning to create boundedly rational function approximation architectures for reinforcement learning? Does a boundedly rational reinforcement learning agent require fewer data to learn a useful policy than agents that do not take resource limitations into account?*

The three chapters contained in this part present one approach to fruitfully integrate existing models of bounded rationality into reinforcement learning. In the first two chapters I address a series of questions on predictive models of bounded rationality in the traditional supervised learning setting. The insights obtained in these two chapters then directly inform the development of the reinforcement learning algorithm presented in the third chapter of this part.

Throughout this part of the dissertation, I assume that the agent has access to pre-selected, numerical features that describe the current situation and that can be used for making decisions and predictions.[35] Coming up with suitable features in the first place is beyond the scope of this thesis and will be considered in future work.

> In Chapter 3 I start with the questions: *How accurate are simple predictive models of bounded rationality? When do they outperform statistical methods from the machine learning literature?* I report the results of a large-scale empirical analysis of simple, boundedly rational regression models, including equal-weighting models and single-predictor models, on 60 real-world data sets. The main findings are as follows. Individual simple regression models routinely outperformed more complex statistical regression models, especially on small training data sets. Occasionally, simple models also performed on par with complex models on larger training sets. On average, however, the simple models failed to reach the mean predictive accuracy of their more complex counterparts on large training sets.

> In the context of reinforcement learning, these results suggest a solution in which the agent adapts the function approximation architecture to the amount and quality of data that is available: in the beginning of learning, when data is scarce, it might be beneficial to rely on a boundedly rational model. In later stage of the learning process, when more and higher-quality data is available, the agent might benefit from relying on a more data-driven statistical model as function approximation architecture.

> It is sensible to assume that the amount and quality of data available to a reinforcement learning agent increases gradually during the learning process. This suggests the use of a function approximation architecture that gradu-

[34] Czerlinski et al. (1999), Brighton (2006), Şimşek and Buckmann (2015), Buckmann and Şimşek (2017), and Katsikopoulos et al. (2018, 2020). See also Section 3.3 for a review of models of bounded rationality in the regression task.

[35] See also Section 2.1.

ally increases its reliance on the data accumulated (as opposed to a function approximator that makes a binary switch from a simple boundedly rational to a complex data-driven model at a certain point), leading to the question: How can such a transition be implemented?

In Chapter 4 I propose a regularized linear model, called *shrinkage toward equal weights* (or STEW), that can flexibly interpolate between a strongly constrained equal-weighting solution and the fully data-driven, unconstrained solution (that is, ordinary least squares in the context of a regression task). Apart from the use case as a smooth transition function between a boundedly rational model and a fully data-driven model (followed up on in Chapter 5, see below), the STEW regularization term is useful in the traditional supervised learning setting, as well.

The STEW regularization term provides a way of fruitfully incorporating a prevalent form of prior knowledge into the learning procedure: feature directions. The direction of a feature indicates whether the feature is associated positively or negatively with the response variable. In many real-world applications, feature directions are known or can be estimated with ease.[36] For instance, consider the task of predicting an apartment's rent price and assume we have access to a feature that indicates whether the apartment has hot water or not (yes = 1, no = 0). Most people would assume that this feature is correlated positively with the price variable, or in other words, that the feature has positive direction.

I present theoretical and empirical evidence that a linear regression model with STEW regularization can benefit from this type of prior knowledge in ways that other regularized linear models cannot. When information on feature directions is available, STEW outperforms existing regularized models including the non-negative Lasso model, which also incorporates knowledge about feature directions. Averaged across 13 real-world data sets, STEW regularization yields an error reduction of up to 20% compared to other regularized models on small training sets and performs as well as the other models on large training size sets.

Finally, in Chapter 5, I integrate ideas and findings from the two previous chapters to construct a new reinforcement learning algorithm, called *iterative policy space expansion*, or IPSE. The IPSE algorithm adapts its function approximation architecture to the amount and quality of data available. In the beginning of the learning process—when data is scarce—the algorithm learns a simple yet effective equal-weighting policy that has a catalytic effect on the learning process and helps the agent to quickly emerge from the gridlock situation described in Limitation A. In later stages of learning—when more and higher-quality data becomes available—the STEW regularization term is used to gradually transform the function approximator from an equal-weighting model to an unconstrained linear model.

[36] I discuss this claim in further detail in Section 2.2.1 in the context of single-shot decision making. In Section 5.3, I present an algorithm that learns feature directions in sequential decision making problems.

When applied to learning how to play the game of Tetris, the IPSE algorithm is substantially more resource-efficient than reinforcement learning algorithms that use a complex function approximator throughout the entire learning process. Specifically, a policy learned by the IPSE algorithm after 400 iterations[37] clears on average more than twice as many lines as the policies learned by competing algorithms using the same amount of training data. Conversely, to achieve the level of performance of the best rival algorithm after 400 iterations, the IPSE algorithm requires around 50 iterations, or in other words, about an eighth of the data.

[37] This corresponds to a training set size of 400 Tetris positions.

▸ PART III: SATISFICING IN REINFORCEMENT LEARNING. This part revolves directly around Herbert Simon's satisficing strategy for decision making and applies it to sequential decision making. This part addresses Limitation B by reducing the number of actions that have to be evaluated during each decision-making process.

In Chapter 6 I formulate a satisficing strategy for use in Markov decision processes. Similar to Simon's original work, the $\xi$-*satisficing policy* considers actions sequentially and selects the first action whose value is larger than some pre-defined aspiration level $\xi$. I study how this policy trades off computational effort against expected quality of the selected action as a function of the aspiration level.

The main challenge for a satisficing agent lies in determining a suitable aspiration level at every decision stage of the Markov decision process. In particular, the effort required for this deliberation process should not offset the computational savings gained by using the satisficing policy in the first place.

I develop three computationally simple *aspiration adaption rules* to set aspiration levels dynamically. The first rule is called *aspiration tracking*. Under the strong conditions that the Markov decision process is deterministic and the agent has access to an optimal value function, a satisficing agent using aspiration tracking follows an optimal policy (in terms of expected return) and requires provably less expected effort than the greedy policy.

I provide examples that illustrate how aspiration tracking fails to produce these results when these conditions are not met. This leads to the development of two other aspiration adaption rules, called *value tracking* and *valved value tracking*, which are more robust to approximation errors in the optimal value function than the aspiration tracking rule. I evaluate all three aspiration adaption rules in the game Lunar-Lander, where the optimal value function is approximated by an artificial neural network. On average, a satisficing agent using valved value tracking reaches the same return as the greedy policy while requiring around 76% of the greedy policy's effort.

Finally, this dissertation concludes with Chapter 7, which discusses the contributions of this dissertation.

## 1.5 PUBLICATIONS

This dissertation produced three publications.

- Jan M. Lichtenberg and Özgür Şimşek (2017). "Simple regression models". In: *Imperfect Decision Makers: Admitting Real-World Rationality*. PMLR 58, pp. 13–25.

- Jan M. Lichtenberg and Özgür Şimşek (2019b). "Regularization in directable environments with application to Tetris". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 3953–3962.

- Jan M. Lichtenberg and Özgür Şimşek (2019a). *Iterative policy-space expansion in reinforcement learning*. arXiv: `1912.02532 [cs.LG]`. This article was presented at the 2019 NeurIPS workshop on Biological and Artificial Reinforcement Learning.

Chapter 3 is based on Lichtenberg and Şimşek (2017), Chapter 4 is based on Lichtenberg and Şimşek (2019b), and Chapter 5 is based on Lichtenberg and Şimşek (2019a,b).

# BACKGROUND & RELATED LITERATURE

The two main elements of this dissertation—reinforcement learning and models of bounded rationality—originate from different research areas: computer science and economics, respectively. Furthermore, whereas reinforcement learning is concerned with sequential decision making, most existing models of bounded rationality were developed for (and studied in) single-shot decision-making or prediction tasks. It is therefore of little surprise that both research communities often use different names, conventions, and notation for similar concepts.

In this chapter I aim to provide a framework that allows to study models of bounded rationality in reinforcement learning, using a single language and a common set of notation. The binding element between both fields in Part II is the field of *supervised learning*, which generally is concerned with learning a function that maps an input to an output from existing input-output pairs. Supervised learning plays a fundamental role in many modern reinforcement learning algorithms. At the same time, we can formulate many existing models of bounded rationality as supervised learning models. This allows a straightforward integration of models of bounded rationality (formulated as supervised learning models) into reinforcement learning.

It is for its function as a binding element that I start by introducing supervised learning in Section 2.1. In Section 2.2 I present existing models of bounded rationality, formulated as supervised learning models. In Section 2.3 I provide a brief introduction to reinforcement learning. Finally, in Section 2.4 I discuss existing work that uses concepts of bounded rationality in machine learning.

## 2.1 SUPERVISED LEARNING

Supervised learning is learning an input-output association from data. For example, the output variable that we wish to predict (also called *response*) could be a quantitative measure of disease progression for diabetes, and the input variables that we would like to base our predictions on (also called *features*) could be a collection of measurements describing the patient, such as several blood serum measurements, age, body mass index, sex, and average blood pressure.[1] Assume that this measure of disease progression is highly expensive or difficult to obtain. Then it would be beneficial to have access to a prediction model that can estimate the progression of diabetes in a patient based on the other features,

[1] We will indeed work with such a data set in later sections. The *Diabetes* data set is described in more detail in Appendix C.

which are much easier (or cheaper) to measure.

To learn such a prediction model, supervised learning requires a *training data set*, which contains both the feature and response values for a collection of observations (such as patients in our diabetes example). During a typical training procedure, the learning algorithm produces response estimates for the given feature inputs and compares the resulting estimates to the true response values from the training data. The model parameters are then adjusted such that the estimates get closer to the true values. Supervised learning is called "supervised" because the learning process is guided by the presence of the response variable in the training data. Different learning algorithms (some of which are described further below) use the training data in different ways.

Once the model is trained, it can be used to predict the response for previously unseen objects, for which only the feature values are known. A good model is one that generalizes well to unseen data, that is, a model that accurately predicts the response values of objects that were *not* used to train the model. In our diabetes example, these would be newly incoming patients with diabetes symptoms.

Formally, a model $f$ parametrized by a vector of parameters $\boldsymbol{\beta}$ predicts the value of the response $y \in \mathcal{Y}$ by

$$\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta}), \tag{2.1}$$

where $\hat{y} \in \mathcal{Y}$ is the predicted value and $\boldsymbol{x} = (x_1, \dots, x_p)$ is a vector of $p$ real-valued[2] features. The training set of $n$ observations is denoted by

$$\mathcal{D} = \{(y_i, \boldsymbol{x}_i), i = 1, \dots, n\},$$

where $(y_i, \boldsymbol{x}_i)$ it the $i$-th observation. We will sometimes use matrix notation to denote the training data more compactly as $\mathcal{D} = (\boldsymbol{y}, \boldsymbol{X})$, where $\boldsymbol{y}$ is the response vector of length $n$, and $\boldsymbol{X}$ is the $n \times p$ feature matrix.

[2] This includes that the features could be binary (for example, the answer to a yes/no question). In some areas of statistical machine learning, it is important to distinguish between binary, categorical, and continuous (real-valued) features. Here we treat binary features as if they were continuous.

### 2.1.1 *Example data for supervised learning: Rent*

Here I introduce a real-world data set that will be used as an example for supervised learning throughout this dissertation. The *Rent* data set[3] contains information about 2053 apartments in Munich, Germany. The objective is to predict the *rent in € per $m^2$* for an apartment, based on eight features including the *number of rooms*, the *year of construction* (in years), and whether *the apartment has warm water* (yes = 1, no = 0). Table 2.1 shows the rent per $m^2$ and the values of five features for the first 10 apartments of the rent data set. A more detailed description about the Rent data set (and about all other supervised learning data sets used in this dissertation) can be found in Appendix C.

[3] Fahrmeir et al. (2012) and Schauberger and Tutz (2014)

### 2.1.2 *Prediction tasks*

Supervised learning can be applied to various prediction tasks. These prediction tasks differ from each other in the type of response variable they predict or in the number of observations that form an input.

| rentm | size | rooms | year | good | ... | kitchen |
|-------|------|-------|------|------|-----|---------|
| 10.90 | 68 | 2 | 1918 | yes | ... | no |
| 11.01 | 65 | 2 | 1995 | yes | ... | no |
| 8.38 | 63 | 3 | 1918 | yes | ... | no |
| 8.52 | 65 | 3 | 1983 | no | ... | no |
| 6.98 | 100 | 4 | 1995 | yes | ... | yes |
| 11.55 | 81 | 4 | 1980 | no | ... | no |
| 3.72 | 55 | 2 | 1924 | no | ... | no |
| 5.40 | 79 | 3 | 1924 | no | ... | no |
| 8.58 | 52 | 1 | 1957 | no | ... | no |
| 4.95 | 77 | 3 | 1948 | no | ... | no |
| ... | ... | ... | ... | ... | ... | ... |

Table 2.1: First 10 observations (apartments) of the *Rent* data set. Each apartment (line) is described by the rent in € per $m^2$ (*rentm*), the *size* of the apartment in $m^2$, the number of *rooms*, the *year* of construction, whether the apartment is situated in a upperclass neighborhood (*good*), and whether the apartment has an upper-market built-in *kitchen*.

In what follows, I present the four prediction tasks that will play a role in this dissertation. I will motivate and explain these prediction tasks by asking four different hypothetical questions about the Rent data set.

▸ REGRESSION. In regression, the goal is to predict a real-valued (or *continuous*) variable of a single observation. That is, the space of the response variable is $\mathcal{Y} = \mathbb{R}$. Figure 2.1 sketches input and output of a regression model. Because rent in € per $m^2$ is a continuous variable, the *Rent* data set naturally suggests the following regression problem:

*"How high is the rent for this apartment?"*

▸ CLASSIFICATION In classification, the goal is to which category out of a fixed set of $k$ classes or categories an observation belongs to. These classes are often represented mathematically by arbitrarily assigning a number form 1 to $k$ to each class, resulting in $\mathcal{Y} = \{1, \dots, k\}$. Figure 2.2 sketches input and output of a classification model.

On the *Rent* data set, we can build a classification problem by defining a "categorical rent variable", called *rent category*, as follows (the chosen categories and thresholds are arbitrary and chosen for illustrative purposes only):

$$rent\ category = \begin{cases} \text{``high''} & \text{if rent per m}^2 > 12; \\ \text{``middle''} & \text{if } 12 \geq \text{rent per m}^2 > 7; \\ \text{``low''} & \text{if } 7 \geq \text{rent per m}^2. \end{cases}$$

The so-constructed response variable has $k = 3$ categories. The corresponding question would be

*"What is the rent category of this apartment? High, middle, or low?"*

A classification problem with $k = 2$ classes is often called *binary classification*. We use $\mathcal{Y} = \{-1, 1\}$ for binary classification.

▸ PAIRED COMPARISON. The paired comparison problem asks which of two alternatives has the higher value in a continuous response variable. Figure 2.3



Figure 2.1: Regression. The model $f$ observes the features of one object and predicts the object's response value $y \in \mathbb{R}$ by $\hat{y} \in \mathbb{R}$. In this example, the predicted value is $\hat{y} = 1.5$.
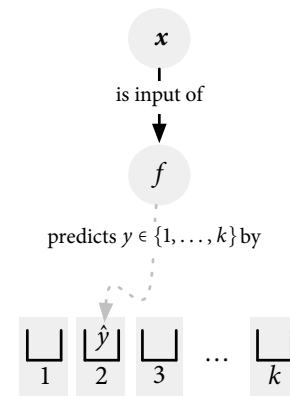


Figure 2.2: Classification. The model $f$ observes the features of one object and predicts the object's response value $y \in \{1, \dots, k\}$ by $\hat{y} \in \{1, \dots, k\}$. In the figure, the model predicts class "2".

sketches inputs and output of a paired comparison model. Using the *Rent* data set, a classical paired comparison question would be

*"Which of these two apartments has a higher rent?"*

The paired comparison problem can be formulated as a classification problem as follows. Let $A$ and $B$ denote the two observations that are to be compared, let $y_A$ and $y_B$ denote their response values, let $x_A$ and $x_B$ denote their feature values, and let *sgn* denote the mathematical sign function: $sgn(z)$ is 1 if $z > 0$, 0 if $z = 0$, and $-1$ if $z < 0$. The classification observation corresponding to the comparison of $A$ and $B$ is then defined as

$$(y, x) := (sgn(y_A - y_B), sgn(x_A - x_B)),$$

that is, the signs of the differences in response and feature values of $A$ and $B$. This leads to $\mathcal{Y} = \{-1, 0, 1\}$ with the obvious interpretation that if $y = 1$, $A$ has a higher response value than $B$; if $y = -1$, $A$ has a lower response value than $B$; and if $y = 0$, there is a tie between both observations.

Formulated this way, the set of possible response variables $\mathcal{Y}$ is a finite set of integers—just as it is in classification.[4] We can therefore use existing classification algorithms to model paired comparison tasks. In particular, if ties are ignored, then the response set of paired comparison is $\mathcal{Y} = \{-1, 1\}$, and paired comparison can be modeled as a binary classification problem.

▸ DISCRETE CHOICE. A discrete choice model predicts a choice among $k$ alternatives, made (or to be made) by a decision maker. Using the *Rent* data set, a classical discrete choice question (with $k = 4$ alternatives) would be

*"Presented with the choice between apartments A, B, C, and D, which one did (or would) the decision maker choose?"*

The training set for a discrete choice model is a set of observed choices. One observation in discrete choice is called a *choice set*. It consists of a collection of choice alternatives, along with their feature descriptions, and the response variable, which indicates the alternative that was chosen. Formally, let $k_i$ denote the size of a given choice set $i$, then the response variable is given by $y_i \in \{1, \ldots, k_i\}$ and the features are given by $x_i = \{x_{i1}, \ldots, x_{ik}\}$, where $x_{ij}$ is the feature vector belonging to alternative $j$ in choice set $i$.

The response in discrete choice is a single integer chosen from a finite set of integers and thus looks just like the response in classification. However, the input in discrete choice is a set of feature vectors rather than a single feature vector as in classification—the two decision problems therefore are different.

Table 2.2 shows an example choice data set consisting of four choice sets (as indicated by varying background colors and the column "choice set id"). Each row corresponds to one choice alternative (for example, an apartment). The observed choice is indicated by a one-hot-variable called "choice", showing the chosen alternative ("1") and the not-chosen alternatives ("0"), for each choice
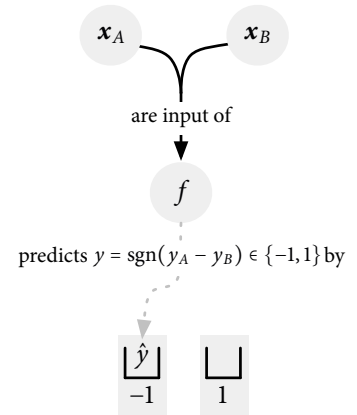


Figure 2.3: Paired comparison. The model $f$ observes the features of two alternatives $A$ and $B$ and predicts which alternative has a higher response value. In the figure, the model predicts that alternative $B$ has a higher response value, that is, $sgn(y_A - y_B) = -1$.

[4] The integers are just arbitrary labels for different classes. It does not matter whether the integers are negative, positive, ordered, or unordered.
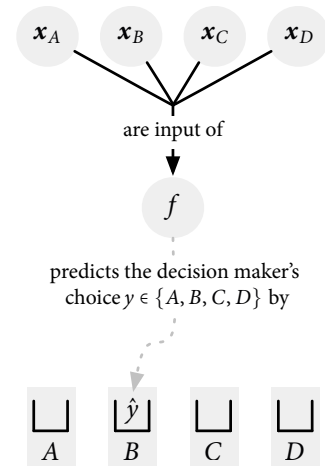


Figure 2.4: Discrete choice. The model $f$ observes the features of all alternatives $A$, $B$, $C$ and $D$ and predicts a decision maker's choice $y \in \{A, B, C, D\}$ by $\hat{y} \in \{A, B, C, D\}$. In the figure, the model predicts that the decision maker would choose alternative $B$.

set. In terms of Equation 2.1, these choices translate to $y_1 = 3$, $y_2 = 2$, $y_3 = 1$, and $y_4 = 5$. Choice sets can be of varying size.

| choice set id | choice | rentm | size | rooms | year | good | ... | kitchen |
|---:|---:|---:|---:|---:|---:|:---:|:---:|:---:|
| 1 | 0 | 10.90 | 68 | 2 | 1918 | yes | ... | no |
| 1 | 0 | 11.01 | 65 | 2 | 1995 | yes | ... | no |
| 1 | 1 | 8.38 | 63 | 3 | 1918 | yes | ... | no |
| 2 | 0 | 8.52 | 65 | 3 | 1983 | no | ... | no |
| 2 | 1 | 6.98 | 100 | 4 | 1995 | yes | ... | yes |
| 2 | 0 | 11.55 | 81 | 4 | 1980 | no | ... | no |
| 2 | 0 | 3.72 | 55 | 2 | 1924 | no | ... | no |
| 3 | 1 | 5.40 | 79 | 3 | 1924 | no | ... | no |
| 3 | 0 | 8.58 | 52 | 1 | 1957 | no | ... | no |
| 4 | 0 | 4.95 | 77 | 3 | 1996 | no | ... | yes |
| 4 | 0 | 7.95 | 90 | 3 | 1973 | no | ... | no |
| 4 | 0 | 13.02 | 110 | 5 | 1910 | yes | ... | yes |
| 4 | 0 | 9.75 | 62 | 3 | 1902 | yes | ... | no |
| 4 | 1 | 6.84 | 42 | 2 | 1989 | no | ... | yes |

Table 2.2: Example choice set data set using data from the *Rent* data set (§2.1.1). The data set contains 4 observations (that is, choice sets), as indicated by the *choice set id* and the alternating background colors. Each choice set consists of a variable number of alternatives (rows), which correspond to apartments in the Rent example. The *choice* variable denotes the observed choice for each choice set. Compared to the data set used for regression or classification (see, for example, Table 2.1), choice set data has two additional variables (*choice set id* and *choice*). Furthermore, one observation consists of multiple choice alternatives.

### 2.1.3    *Prediction models*

The last subsection was about supervised learning problems. This section is about the solutions to these problems and different approaches to tackle the central question: *How to construct a good model f?*

The literature has proposed many different model classes and algorithms to estimate their parameters. All models suitable for a particular prediction task take the same types of feature values as inputs and produce the same types of outputs, as described in the previous subsection. Model classes differ in the type and number of operations that can be applied to tweak and combine the input features in order to produce the desired output value.

Next I describe the linear model, which is one of the most-used model classes not only in machine learning but also in the social and natural sciences. Furthermore, many existing models of bounded rationality are special cases of the linear model (see Section 2.2). Further below, I then briefly describe (deep) artificial neural networks, which generalize linear models by combining multiple individual linear models into one single model.

▸ THE LINEAR MODEL. The linear model allows all features to be weighted independently (that is, multiplied by a real-valued weight parameter) before the weighted feature values are summed up to produce the output. Formally, a linear model is given by

$$\hat{y} = g(\sum_{j=1}^{p} x_j \beta_j), \qquad (2.2)$$

where $\beta_1, \ldots, \beta_p$ are the linear weights[5] and $g : \mathbb{R} \to \mathcal{Y}$ is a *link function* that relates the linear combination of features to the response variable, $y$. The link function is usually chosen in accordance with the prediction task:

In regression, $\mathcal{Y} = \mathbb{R}$ and here we simply use the identity function $g(x) = x$ as link function (that is, the linear model simplifies to $\hat{y} = \sum_{j=1}^{p} x_j \beta_j$).

[5] Sometimes, the linear model is defined as $\hat{y} = g(\beta_0 + \sum_{j=1}^{p} x_j \beta_j)$, that is, with an "intercept" $\beta_0$. Note that the intercept-free version (Equation 2.2) is equivalent to the version that uses an intercept if the feature set of the former is expanded by a constant feature variable $x_c = 1$. For notational simplicity, we stick with the intercept-free formulation in the remainder of this dissertation.

In binary classification, popular choices for the link function are the logistic function $g(x) = \frac{1}{1+e^{-x}}$ and the logistic function with subsequent thresholding,[6] given by $g(x) = sgn(\frac{1}{1+e^{-x}} - t)$, where $t \in [0,1]$ is the decision threshold.

Using vector notation and denoting $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$, we can rewrite the linear model as $\hat{y} = g(\boldsymbol{x}^\top \boldsymbol{\beta})$. Similarly, we can express the predicted values for an entire training set as $\hat{\boldsymbol{y}} = g(\boldsymbol{X}\boldsymbol{\beta})$, where the link function $g$ here is applied element-wise to the elements of the vector $\boldsymbol{X}\boldsymbol{\beta}$.

In discrete choice, the linear model is called *multinomial logistic regression*.[7] It is slightly more complicated than linear models in regression or binary classification because the input of the model in discrete choice consists of multiple feature vectors describing each of the $k$ choice alternatives, denoted by $\boldsymbol{x}_i$ for $i = 1, \ldots, k$. The model computes a latent utility (or *score*) for each alternative, given by

$$U(i) = \boldsymbol{\beta}^\top \boldsymbol{x}_i \quad \text{for } i = 1, \ldots, k.$$

A deterministic version of multinomial logistic regression outputs the utility-maximizing choice alternative, that is,

$$\hat{y} = \underset{i \in \{1, \ldots, k\}}{\operatorname{argmax}} U(i).$$

Alternatively, a stochastic version of multinomial logistic regression outputs choice alternative $i$ with probability

$$\mathbb{P}(\hat{y} = i) = \frac{e^{U(i)}}{\sum_{j=1}^{k} e^{U(j)}}.$$

▸ LEARNING A LINEAR MODEL. The linear model by itself only specifies the way in which the features $\boldsymbol{x}$ can be possibly tweaked and combined to produce an estimate of the response $y$. Here we are interested in how the model parameters are learned from a training data set.

The overall objective is to choose the model parameters such that the model predictions on unseen observations are as close as possible to the true response values. The obvious problem is that we generally do not know the true response values of any future observations. We do, however, have access to the true response values in the "supervised" training data.

For this reason we make an assumption that is central to most of supervised learning: future data is assumed to be generated from the same data generating process as the training data. Building on this consistency assumption, we can use the training data as a proxy for future data. In particular, if we find parameters such that the model's predictions of response values in the training data are close to their true values, then there is hope that predictions will be accurate on previously unseen data as well.

What it means for a prediction and a true value to be "close" to each other depends on the prediction task at hand. The following models maximize some form of closeness (or equivalently, minimize some form of distance function) between predictions and true values for linear models in various prediction tasks.[8]

*Ordinary least squares*, or OLS, is a linear regression model. The OLS estimate is defined as the minimizer of the residual sum of squares, or RSS. The RSS sums up the squared distances between predictions and true values on the training data and is given by

$$RSS(\boldsymbol{\beta}) = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \|\boldsymbol{y} - \hat{\boldsymbol{y}}\|_2^2 = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 \, .$$

The corresponding minimization problem allows the closed-form solution

$$\boldsymbol{\beta}^{OLS} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \; RSS(\boldsymbol{\beta}) = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}.$$

Minimizing the RSS is equivalent to maximizing the likelihood of the data under the assumption that the conditional distribution of the response given the features follows the Gaussian distribution $y|X \sim \mathcal{N}(\boldsymbol{x}^T\boldsymbol{\beta}, \sigma^2)$, where $\sigma^2 > 0$ is the unknown variance.

*Logistic regression* is a linear binary classification model. It searches the weights that maximize the likelihood of the training data under the assumption that the response variable $y$ is sampled from a random variable $Y$ that follows a Bernoulli distribution with probability of success given by the linear model with logistic link function. Formally, the probability that the response is "1" is given by

$$\mathbb{P}(Y = 1|\boldsymbol{x}, \boldsymbol{\beta}) = g(\boldsymbol{x}^T\boldsymbol{\beta}) = \frac{1}{1 + e^{-\boldsymbol{x}^T\boldsymbol{\beta}}} \, .$$

The corresponding likelihood function is given by

$$\mathcal{L}(\boldsymbol{\beta}|\boldsymbol{y}; \boldsymbol{X}) = \prod_{i=1}^{n}\left[\mathbb{P}(Y = 1|\boldsymbol{x}, \boldsymbol{\beta})^{y_i}(1 - \mathbb{P}(Y = 1|\boldsymbol{x}, \boldsymbol{\beta}))^{(1-y_i)}\right].$$

The maximization of this likelihood function does not allow a closed-form solution. However, the likelihood function for logistic regression is globally concave and easily differentiable. It thus has a unique maximum which can be easily approximated using standard numerical optimization algorithms such as gradient ascent.

*Multinomial logistic regression* (discrete choice). Multinomial logistic regression maximizes the likelihood of observed choices in the training data under the following assumption about the decision maker's choice behavior: The decision maker's choice is stochastic and can be described by a random variable $Y$ with support $\{1, \dots, k\}$ and the probability that the decision maker

chooses alternative $y$ is given by

$$\mathbb{P}(Y = i \mid \boldsymbol{\beta}; \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k) = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{\beta}}}{\sum_{j=1}^{k} e^{\boldsymbol{x}_j^T \boldsymbol{\beta}}},$$

where $\boldsymbol{x}_j$ is the feature vector describing alternative $j$. Further assuming that the observed choices in the choice data set $\mathcal{D}$ are independent of each other, the corresponding log-likelihood is given by

$$\log \mathcal{L}(\boldsymbol{\beta}|\mathcal{D}) = \sum_{i}^{n} \log \mathbb{P}(Y = y_i \mid \boldsymbol{\beta}; \boldsymbol{x}_{i1}, \ldots, \boldsymbol{x}_{ik}), \tag{2.3}$$

where $\boldsymbol{x}_{ij}$ is the feature vector of size $p$, corresponding to alternative $j$ in choice set $i$. Similar as in logistic regression, this likelihood function can be maximized numerically using gradient ascent techniques.[9]

[9] See Train (2009) for a comprehensive tutorial about estimating the parameters of a multinomial logistic regression model.

▸ DEEP NEURAL NETWORKS. Deep neural networks are a large class of machine learning models that are composed of multiple connected processing layers that learn progressively more complex representations of raw data.[10] In recent years, deep neural networks improved the state-of-the-art across many machine learning areas such as natural language processing and computer vision. Here I only discuss the deep learning architecture that is relevant and used in this dissertation: feed-forward deep neural networks. For an introduction to various other neural networks architectures as well as the algorithms to train them, the reader is referred to Goodfellow et al. (2016).

[10] LeCun et al. (2015)



Figure 2.5: Feed-forward neural network with $H$ hidden layers.

Figure 2.5 shows a deep feed-forward neural network with $H$ hidden layers (only the first and the last hidden layer are actually shown in the figure). Each circle represents one "neuron". Neurons are organized in layers. The shown network takes a single vector of feature values $\boldsymbol{x}$ as input and output a scalar

response value $y$ (in general, the output layer can consist of any number of neurons). In the simplest case, every neuron (including the output neurons but excluding the input features) is a function of the outputs provided by the neurons in the previous layer.
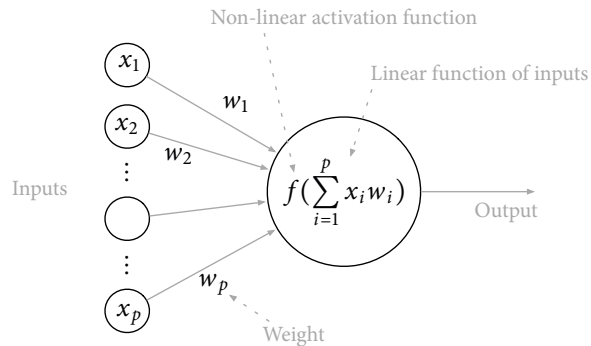


Figure 2.6: Single neuron of a feed-forward neural network.

Figure 2.6 shows a single neuron of the first hidden layer in greater detail. Each arrow can be thought of as a weight that determines the importance of the respective input feature for that neuron. The output of the neuron is computed as follows. First, the neuron computes the linear weighted sum of the features, given by $\sum_{i=1}^{p} w_i x_i$. The output of that linear function is then sent through a non-linear activation function $f$ to produce the neuron's output. Neurons in all other hidden layers look the same, except for the inputs, which are the outputs of the previous layer's neurons (rather than the input features). The output neurons do also take the same form, however, their activation functions are often different from those of the neurons in the hidden layers, in order to match the desired output format.

## 2.2    MODELS OF BOUNDED RATIONALITY

Models of bounded rationality are prediction or decision-making models that take into consideration the cognitive limitations of the decision-maker's mind, and the limited data and time available to the decision maker in many real-world decision tasks.[11] Many of these models can be described by cognitively plausible processes that build on simple concepts and building blocks such as pair-wise comparisons or the summation of few values. This stands in contrast to more involved techniques such as calculating numerical gradients or inverting large matrices, which are often required to learn the parameters of more complex models from the machine learning literature.

More often than not, simple models of bounded rationality do not need to be defined as the solution to a mathematical optimization problem but instead can be described by a simple mechanism that searches for a good-enough solution. The fact that most models of bounded rationally can also be formulated as the solution to some constrained optimization problem is irrelevant here. It

[11] Simon (1997) and Gigerenzer et al. (1999)

is the fact that these models arrive at good solutions *without having to perform costly optimization procedures* that makes them interesting candidates for the purposes of this dissertation.

In this section I discuss existing boundedly rational prediction models that can be formalized in the supervised learning framework. That is, these models take a set of features as input and output a response variable (the type of which depends on the prediction task at hand), and the model's parameters are learned using a training data set. Compared to more complex supervised learning models such as deep neural networks, boundedly rational supervised learning models do not necessarily use all of the available information (that is, features), or they combine different pieces of information in very simple ways, for example, by giving them equal weight.[12]

The psychological literature has produced and studied a range of boundedly rational, feature-based predictive models including *fast-and-frugal heuristics*[13] and *simple regression models*.[14] In the remainder of this section I will present two general strategies that are used by many boundedly rational models. I formalize the corresponding models as instances of the general linear model discussed in the previous section, which facilitates their comparison with other supervised learning models. First, however, I define and discuss an important building block of various models of bounded rationality: feature directions.

### 2.2.1 Feature directions

The *direction* of a feature in a linear model is defined as the sign of the corresponding weight, which can be positive or negative. Feature directions are building blocks for many models of bounded rationality including lexicographic heuristics and equal-weighting strategies, as described in the upcoming sections.

An environment is said to be *directable* if the directions of the features are known. In a directable environment, features can be "directed" so that the weights are all positive (for example, by recoding any feature with a negative weight by multiplying its values by −1).

In many problems feature directions are known beforehand, for instance, when the problem is naturally constrained to have only positive weights.[15] In other problems, features can be directed intuitively by the user, as supported by experimental evidence.[16] Even without any prior knowledge, directions can be estimated from relatively few training data in single-shot tasks.[17] In Chapter 5 I present an algorithm to learn feature directions in sequential decision-making tasks.

### 2.2.2 Lexicographic models

*Lexicographic models*[18] are a class of boundedly rational models for choosing among decision alternatives. A lexicographic model considers features sequentially in a given order and makes a prediction based on the first discriminating

[12] Şimşek (2013) and Gigerenzer et al. (1999), see also Section 3.3.

[13] Gigerenzer and Goldstein (1996) and Gigerenzer et al. (1999, 2011)

[14] A history and an extensive literature review of simple regression models is provided in Section 3.3.

[15] For example mixing problems, see Slawski and Hein (2013) and references therein.

[16] Dana and Thomas (2006) and Katsikopoulos et al. (2010)

[17] Şimşek and Buckmann (2015)

[18] Fishburn (1974), Gigerenzer and Goldstein (1996), and Şimşek (2020b)

feature that is found. A specific lexicographic model thus has to answer the two questions: What is a discriminating feature, that is, when does one stop the search for more information? And how to determine the order in which features are considered? Next follows a brief description of a widely studied lexicographic model for the paired comparison task.

The *take-the-best* heuristic[19] is a lexicographic paired comparison model that orders available features by decreasing validity[20] and makes a decision based on the first feature that discriminates between the two alternatives. A feature discriminates between two alternatives if the corresponding two feature values are different.

Take-the best can be formulated as a linear paired-comparison model as follows. Assume that the features variables $x_1, x_2, \ldots, x_p$ are ordered by decreasing validity and that all feature weights are positive.[21] Then the take-the-best model is equivalent to a linear paired-comparison model whose weight vector $\boldsymbol{\beta}$ satisfies the non-compensatoriness constraints

$$\beta_i > \sum_{j=i+1}^{p} \beta_j, \text{ for } i = 1, 2, \ldots, p-1,$$

where $p$ is the number of predictors.[22] For example, a set of non-compensatory weights is given by the sequence of exponentially decaying weights

$$\beta_1 = \frac{1}{2}, \ \beta_2 = \frac{1}{4}, \ \beta_3 = \frac{1}{8}, \ \ldots, \ \beta_p = \frac{1}{2^p}.$$

These models are called *non-compensatory* because the decision is made based on the satisfying feature alone—the remaining features that have not yet been considered cannot compensate or overturn this decision.

To use a take-the-best model, the decision maker thus needs to be able to rank features by decreasing validity and needs to know all feature directions. Despite its low requirements and limited expressiveness, the take-the-best heuristic has been found to perform remarkably well in various empirical comparisons to complex machine learning models, including artificial neural networks, support-vector machines, random forests, and elastic-net regularized linear regression.[23]

The next section presents a different model of bounded rationality, which is arguably even simpler than a lexicographic model because it does not require the decision maker to order the features in any way.

### 2.2.3 *Equal-weighting strategies*

Equal-weighting strategies attribute equal importance to each feature. They are regularly used in various real-world decision-making tasks under uncertainty.[24] One example is given by equal-weighted index funds for stock market investments, which present a simple and effective investment strategy.[25] Equal-weighting strategies are also frequently used in group decision-making across various politics and business contexts in the form of simple majority voting.[26]

[19] Gigerenzer and Goldstein (1996)

[20] The (empirical) validity of a feature in a paired comparison task is the accuracy (ratio of correct inferences) of that feature among pairwise comparisons on which the feature discriminates, see, for example, Şimşek and Buckmann (2015).

[21] If a certain feature weight is known to be negative, the corresponding feature can be *directed* by inverting the feature values, see also Section 2.2.1.

[22] Martignon and Hoffrage (1999, 2002)

[23] Czerlinski et al. (1999), Brighton (2006), Şimşek and Buckmann (2015), and Buckmann and Şimşek (2017)

[24] Katsikopoulos et al. (2020)

[25] DeMiguel et al. (2009b,a)

[26] Galesic et al. (2018)

Models of equal-weighting strategies (or simply, equal-weighting models) have a long history of use in the social sciences. They appeared in a seminal paper by Dawes and Corrigan (1974) that showed that even so-called improper models (such as equal-weighting models) could outperform human expert judgements. The article also demonstrated that these models can compete well with OLS on real-world data sets, stimulating further work on equal-weighting models in the 1970s, continuing to this day.[27]

[27] Einhorn and Hogarth (1975), Wainer (1976), Gigerenzer et al. (1999), and Katsikopoulos et al. (2020). See also Section 3.3 for a review on equal-weighting models specifically in the regression task.

Equal-weighting models can take different forms depending on the prediction task they are used in. In the regression task it makes sense to think of "equal importance of features" as "feature weights having equal magnitudes". That is, every feature contributes equally to the outcome, but that contribution can be positive or negative. Formally, this can be achieved by a linear model whose weights satisfy the constraints

$$|\beta_1| = |\beta_2| = \cdots = |\beta_p|.$$

In this case, the decision maker needs to decide the direction of each feature and the overall scale (that is, the magnitude of all weights).

In some prediction tasks, however, the linear model is invariant to scaling of feature weights and the magnitude of the equal-weighting constant can thus be set arbitrarily.[28] In such a case the decision maker only has to estimate feature directions to use the equal-weighting model.

[28] For instance, the linear discrete choice model considered in Chapter 5 is invariant to scale.

The model further simplifies in resource allocation problems, where the task is to distribute a finite set of resources among competing alternatives. A weight here corresponds to the proportion of resources allocated to the corresponding alternative and thus has to be non-negative. Furthermore, all the weights have to sum to one, leading to $\beta_i = \frac{1}{p}$, for all features $i = 1, \ldots, p$. For example, in a traditional portfolio allocation problem, the equal-weighting strategy is to allocate the same amount of funds to each asset or asset class under consideration. This strategy, also called the $\frac{1}{N}$-rule, was found to outperform many other, more data-driven investment strategies.[29]

[29] DeMiguel et al. (2009b,a)

## 2.3 REINFORCEMENT LEARNING

In this section I present the reinforcement learning algorithms that are used or modified in later parts of this dissertation. This section builds heavily on the textbook by Sutton and Barto (2018). The reader is referred to the same textbook for a more thorough introduction to reinforcement learning.

Reinforcement learning is concerned with a decision maker (called an *agent*) who learns how to solve a sequential decision making problem. The agent learns by repeatedly interacting with its (decision-making) *environment*. The agent receives feedback on its actions in the form of a *reward* signal. We make the convenient assumption that the interaction between agent and environment can be modeled as a Markov decision process, or MDP.[30]

[30] This assumption is made by most research work in reinforcement learning. And indeed, all sequential decision problems considered in this dissertation can be modeled as MDPs.

▸ MARKOV DECISION PROCESSES. Figure 2.7 summarizes the agent-environment interaction in a MDP. In a given decision stage (or time step) $t$, the agent observes the current configuration (or *state*) of the environment, given by $s_t$. The agent chooses an action $a_t$ based only on the information contained in the state description $s_t$. The agent then executes action $a_t$, which possibly changes the state of the environment. The agent receives the reward $r_t$ and observes the new state $s_{t+1}$. The information contained in the state description of $s_{t+1}$ is then used to choose the next action $a_{t+1}$, and so on.
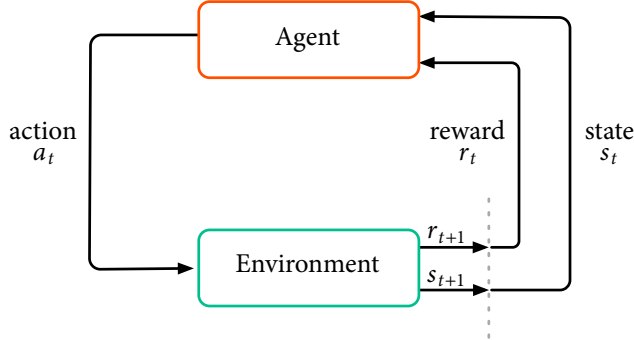


Figure 2.7: Agent-environment interface in a Markov decision process (MDP). The figure is adapted from Sutton and Barto (2018, p. 48).

More formally, an MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma, \rho_0)$, where

$\mathcal{S}$ is a set of states, which describe the current configuration of the decision environment;

$\mathcal{A}$ is a set of actions that the agent can execute ($\mathcal{A}(s)$ denotes the set of actions executable in state $s$);

$P(s_{t+1}|a_t, s_t) : \mathcal{S} \times \mathcal{A}(s) \times \mathcal{S} \to [0,1]$ defines transition probabilities, that is, the probability of arriving in state $s_{t+1}$ after executing action $a_t$ in state $s_t$;

$R(s_t, a_t, s_{t+1}) : \mathcal{S} \times \mathcal{A}(s) \times \mathcal{S} \to \mathbb{R}$ is the reward function;

$\gamma$ is a temporal discount factor, which defines the relative importance between future and immediate rewards; and

$\rho_0$ is the distribution from which the initial state $s_0$ is drawn.

In every time step $t$, the agent executes an action $a_t$ in state $s_t$, observes a new state $s_{t+1} \sim P(s_{t+1}|a_t, s_t)$, and receives a reward $r_t = R(s_t, a_t, s_{t+1})$. In most cases, neither the transition model nor the reward function is known to the agent and the agent can only learn through trial-and-error. In this dissertation, I focus on discrete actions sets. The number of actions in an action set is given by the cardinality of the action set, denoted by $|\mathcal{A}|$. The number of available actions in a state $s$ is given by $|\mathcal{A}(s)|$. The number of available actions can differ between different states.

The objective in reinforcement learning is to find an optimal behavioral *policy*. A policy defines the agent's behavior at a given time, and is usually a function of the current state of the environment. A *deterministic policy* $\pi(s) : \mathcal{S} \rightarrow \mathcal{A}(s)$ maps a state $s \in \mathcal{S}$ to an action $a$ from the set of actions available in that state, given by $\mathcal{A}(s)$. A *stochastic policy* $\pi(a|s) : \mathcal{A}(s) \times \mathcal{S} \rightarrow [0,1]$ is a probability distribution over actions in $\mathcal{A}(s)$. An optimal policy, denoted by $\pi_*$, is a policy that maximizes the expected future cumulative reward, which is also called expected *return*, and is given by $\mathbb{E}_{s_0 \sim \rho_0, a_t \sim \pi(s_t)} \left[ \sum_{t=0}^{\infty} r_t \right]$.

The following two subsections describes the two general reinforcement learning approaches that provide the foundations for the algorithms proposed in this dissertation. Value-based methods will play a central role in Chapter 6. Classification-based reinforcement learning is the basis for the algorithms described in Chapter 5.

2.3.1    *Value-based reinforcement learning*

Value-based control algorithms learn a state-action value function to structure the search for an optimal policy. The state-action value function (or shorter, *action-value* function) with respect to some policy $\pi$, denoted by

$$q_\pi(s, a) = \mathbb{E}_{a_t \sim \pi(s_t), t > 0} \left[ \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a \right],$$

is the expected reward obtained by the agent after taking action $a$ in state $s$ and following $\pi$ thereafter. Closely related to the action-value function is the so-called state-value function, given by

$$v_\pi(s) = \mathbb{E}_{a_t \sim \pi(s_t), t \geq 0} \left[ \sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s \right].$$

Given an optimal value-function, defined by $q_*(s, a) = \max_\pi q_\pi(s, a)$, an optimal policy can be obtained by calculating the *greedy policy* with respect to the optimal value function, given by

$$\pi_*(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \, q_*(s, a).$$

Value-based reinforcement learning algorithms try to approximate the optimal value function as closely as possible and then compute the greedy policy with respect to that approximated value function to obtain an approximately optimal policy. The agent usually starts by initializing the action-values of all actions in non-terminal states to a random number (or simply, to zero). During learning, these values are periodically updated with the aim to get the estimates closer to the true action-values. Next I describe a widely studied class of value-based reinforcement learning algorithms called temporal-difference learning.

▸ TEMPORAL-DIFFERENCE LEARNING. To make an update to the action-value function, temporal-difference algorithms require only information gathered in

the current transition as opposed to information gathered throughout an entire episode,[31] or access to the transition model of the MDP.[32]

One of the canonical tabular temporal-difference reinforcement learning algorithms is called *Sarsa*. In every time step the Sarsa algorithm updates the action value of the current state-action pair, using five pieces of information: $s_t, a_t, r_t, s_{t+1}$, and $a_{t+1}$. The value function update rule for the tabular Sarsa algorithm is given by

$$q(s_t, a_t) \leftarrow (1 - \alpha)q(s_t, a_t) + \alpha[r_t + q(s_{t+1}, a_{t+1})], \qquad (2.4)$$

where $0 < \alpha < 1$ is a learning rate parameter.

During learning, actions are chosen based on the current action-value estimates. If the agent always chooses the action that is currently believed to be best, it can get stuck in a local optimum. That is, the agent does not find out about a better action in a given state simply because that action is never tried out, or not often enough. We say, the agent does not *explore* (enough).

One approach that guarantees enough exploration during the learning process is to use an *ε-greedy* policy. The ε-greedy policy takes a random action with probability $\varepsilon$ and the greedy action with probability $1 - \varepsilon$. Algorithm 1 provides pseudo code for the tabular Sarsa algorithm with ε-greedy exploration. It is called "tabular", because the action-value estimates can be simply stored in a table.

---

**Algorithm 1** Tabular Sarsa with ε-greedy exploration. Adapted from Sutton and Barto (2018, p. 130).

---

**Output:**
$q$, an action-value function.
**Input:**
$q(s, a)$ for all $s \in \mathcal{S}, a \in \mathcal{A}(s)$      *// table of action-values*
$\alpha \in (0, 1]$      *// learning rate*
$\varepsilon \in [0, 1]$      *// exploration parameter*
$\gamma \in [0, 1]$      *// discount factor*

Initialize all $q(s, a)$ arbitrarily, except that $q(\text{terminal}, \cdot) = 0$
**for** each episode **do**
    Initialize $s_0$
    Choose $a_0$ from $s_0$ using ε-greedy policy derived from current $q$
    **for** each time step $t = 0, 1, 2, \ldots$ of the episode **do**
        Take action $a_t$, observe $r_t, s_{t+1}$
        Choose $a_{t+1}$ from $s_{t+1}$ using ε-greedy policy derived from current $q$
        $q(s_t, a_t) \leftarrow (1 - \alpha)q(s_t, a_t) + \alpha[r_t + \gamma q(s_{t+1}, a_{t+1})]$      *// Equation 2.4*
        $s_t \leftarrow s_{t+1}; a_t \leftarrow a_{t+1}$
    **end for**
**end for**

---

▸ Function approximation in reinforcement learning. The state spaces of interesting sequential decision making problems are usually too large to learn and maintain a table of action values for each state-action pair. Function approximation methods approximate the action-value function by a parametrized, mathematical function that takes as input a set of *features* describing the action (in the current state) and that produces as output the corresponding value estimate.[33] More formally, let $\phi(s, a) \in \mathbb{R}^p$ denote a feature vector describing the state action pair $(s, a)$. The approximated action-value function is then a function $\hat{q}(s, a, \boldsymbol{\theta}) : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \to \mathbb{R}$, where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a parameter vector.

For value-based reinforcement learning algorithms, learning then consists of finding $\boldsymbol{\theta}$ such that the approximated value function $\hat{q}(s, a, \boldsymbol{\theta})$ is close to the true value function $q(s, a)$ for all state-action pairs.

The function-approximated equivalent of the tabular Sarsa algorithm is called *semi-gradient Sarsa with function approximation*.[34] Its update rule is given by

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[r_t + \hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{\theta}) - \hat{q}(s_t, a_t, \boldsymbol{\theta})] \nabla_{\boldsymbol{\theta}} \hat{q}(s_t, a_t, \boldsymbol{\theta}), \qquad (2.5)$$

where $0 < \alpha < 1$ is a learning rate as in the tabular case and $\nabla_{\boldsymbol{\theta}}$ denotes the vector of partial derivatives with respect to the individual elements of $\boldsymbol{\theta}$ (the parameters), or formally, $\nabla_{\boldsymbol{\theta}} \hat{q}(s_t, a_t, \boldsymbol{\theta}) = \left[\frac{\partial \hat{q}(s_t, a_t, \boldsymbol{\theta})}{\partial \theta_1}, \frac{\partial \hat{q}(s_t, a_t, \boldsymbol{\theta})}{\partial \theta_2}, \ldots, \frac{\partial \hat{q}(s_t, a_t, \boldsymbol{\theta})}{\partial \theta_d}\right]$. Many successful applications parametrize $\hat{q}$ as a linear function or a deep neural network, both of which are easily differentiable with respect to their model parameters and therefore keep the computational requirements of an update step (Equation 2.5) relatively low. Algorithm 2 provides pseudo code for semi-gradient Sarsa with function approximation and $\varepsilon$-greedy exploration.

### 2.3.2 *Classification-based reinforcement learning*

In some domains, value functions are difficult to learn from limited data.[35] The reinforcement learning algorithms discussed in this subsection directly approximate the optimal policy without the help of an explicit action-value function. Instead, these algorithms interpret policy learning as a classification problem.[36]

More specifically, the policy is learned from a training set of existing input-output pairs, where the input is the feature description of a state and the output is an optimal action. This has the advantage that the algorithm designer can choose from a large range of proven classification algorithms including linear classifiers, random forests, support vector machines, or neural networks, which were shown to generalize well to unseen observations. The hope is that the policy, learned on a limited training data set of optimal decisions, generalizes well and produces optimal (or near-optimal) behavior on the entire MDP.

The problem is that such a data set of optimal decisions is not available for most MDPs. The agent therefore has to build the training data set on its own, through interaction with the environment, or a simulation thereof. The big challenge is that the agent usually starts off with a random policy, which makes it difficult to find an optimal (or at least near-optimal) action for a given state to begin with.

[33] The term "function approximation methods" is oddly unspecific but actually refers quite specifically to a set of well-known methods within reinforcement learning research (Sutton and Barto, 2018, Part II). Sometimes, the almost equally unspecific term "reinforcement learning with function approximation" is used.

[34] Sutton and Barto (2018, Section 10.1).

[35] For example, because the action value estimates inherently show high variance. A good example for this is Tetris, as discussed further in Section 5.6.1.

[36] Lagoudakis and Parr (2003), Fern et al. (2004), Li et al. (2007), Lazaric et al. (2016), and Scherrer et al. (2015)

---

**Algorithm 2** Semi-gradient Sarsa with function approximation and $\varepsilon$-greedy exploration. Adapted from Sutton and Barto (2018, p. 244).

---

**Output:**
$\hat{q}$, an approximated action-value function.
**Input:**
$\hat{q}(s, a, \boldsymbol{\theta}) : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^d \rightarrow \mathbb{R}$              // *differentiable action-value function*
$\alpha \in (0, 1]$                                     // *learning rate*
$\varepsilon \in [0, 1]$                               // *exploration parameter*
$\gamma \in [0, 1]$                                   // *discount factor*

Initialize $\boldsymbol{\theta}$ arbitrarily
**for** each episode **do**
    Initialize $s_0$
    Choose $a_0$ from $s_0$ using $\varepsilon$-greedy policy derived from current $\hat{q}$
    **for** each time step $t = 0, 1, 2, \ldots$ of the episode **do**
        Take action $a_t$, observe $r_t, s_{t+1}$
        **if** $s_{t+1}$ is terminal **then**
            $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[r_t - \hat{q}(s_t, a_t, \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}}\hat{q}(s_t, a_t, \boldsymbol{\theta})$
            Go to next episode
        **end if**
        Choose $a_{t+1}$ from $s_{t+1}$ using $\varepsilon$-greedy policy derived from current $v$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha[r_t + \gamma\hat{q}(s_{t+1}, a_{t+1}, \boldsymbol{\theta}) - \hat{q}(s_t, a_t, \boldsymbol{\theta})]\nabla_{\boldsymbol{\theta}}\hat{q}(s_t, a_t, \boldsymbol{\theta})$      // *Eq.* 2.5
        $s_t \leftarrow s_{t+1}; a_t \leftarrow a_{t+1}$
    **end for**
**end for**

---

▸ ROLLOUTS are one way of using the current policy to generate a training instance for the classification data set (that is, to find an approximately optimal action in a given state). Rollouts work as follows.

First, the value of each available action in a given state is approximated by the cumulative sum of rewards obtained in a finite-length forward simulation of the environment, in which the first action is the action to be evaluated and the following actions are chosen according to a *rollout policy*.[37] The action with the highest approximate value then becomes the class label for that state.

Formally, let $\mathcal{G}(s, a) : \mathcal{S} \times \mathcal{A}(s) \rightarrow \mathcal{S} \times \mathbb{R}$ denote a generative model (or simulator), which allows to sample a next state $s'$ and reward $r$ for a given state action pair $(s, a)$ without changing the true state of the environment and let $\hat{s}$ denote the state for which we wish to find the optimal action (sometimes called the *rollout starting state*). Furthermore, let $\hat{\pi} : \mathcal{S} \rightarrow \mathcal{A}$ denote a rollout policy. A rollout trajectory for action $a \in \mathcal{A}(\hat{s})$ in state $\hat{s}$ of length $T$ is given by

$$\hat{s}, a, r_0, s_1, a_1, r_1, \ldots, s_T, a_T, r_T,$$

where $s_1, r_0 \sim \mathcal{G}(\hat{s}, a), s_{t+1}, r_t \sim \mathcal{G}(s_t, a_t)$ for $t \geq 1$, and $a_t = \hat{\pi}(s_t)$ for $t \geq 1$. The approximated action value is given by $\hat{U}(\hat{s}, a) = \sum_{t=0}^{T} r_t$. Sometimes, the approximated action value is averaged across multiple rollouts. The classification label is determined by determining the action with the highest approximated value, given by $\hat{a} = \text{argmax}_{a \in \mathcal{A}(\hat{s})} \hat{U}(\hat{s}, a)$.

[37] In many cases the rollout policy is simply the agent's current policy or some modification thereof, for example, the current policy with added or reduced exploration behavior.

Algorithm 3 provides pseudo-code for the rollout procedure for a single state-action pair.

---

**Algorithm 3** ROLLOUT($s, a, \pi_r$): Rollout procedure for estimating the value of an action $a$ in state $s$ using rollout policy $\pi_r$.

---

**Output:**
$\hat{U} \in \mathbb{R}$,  *estimated value of taking action a in s*
**Input:**
$s \in \mathcal{S}$                                            *// rollout starting state*
$a \in \mathcal{A}(s)$                    *// action to be evaluated*
$\pi_r(s) : \mathcal{S} \to \mathcal{A}$                  *// rollout policy*
$M \in \mathbb{N}$                   *// number of rollouts*
$T \in \mathbb{N}$                   *// rollout length*
$\gamma \in [0, 1]$                   *// discount factor*
$\mathcal{G}(s, a) : \mathcal{S} \times \mathcal{A}(s) \to \mathcal{S} \times \mathbb{R}$             *// generative model*

**for all** $j = 1, \ldots, M$ **do**
  $(s', r) \leftarrow \mathcal{G}(s, a)$
  $\hat{U}_j \leftarrow r$
  $s \leftarrow s'$
  **for all** $t = 1, \ldots, T - 1$ **do**
    $(s', r) \leftarrow \mathcal{G}(s, \pi_r(s))$
    $\hat{U}_j \leftarrow \hat{U}_j + \gamma^t r$
    $s \leftarrow s'$
  **end for**
**end for**
**return** $\hat{U} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \hat{U}_j$

---

▸ CLASSIFICATION-BASED REINFORCEMENT LEARNING WITH ROLLOUTS.[38] Rollout-based reinforcement learning algorithms proceed in iterations, producing a series of classification training sets

$$\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k,$$

which are used to learn policies

$$\pi_1, \pi_2, \ldots \pi_k,$$

respectively. In the most basic form of the algorithm, the rollout policy used to create the training data set $\mathcal{D}_t$ is the policy learned in the previous iteration, $\pi_{t-1}$.[39]

The main idea is that the decision made with the help of the rollout procedure should be, on average, slightly better than the decision that would be made by the current policy without the help of any forward simulation.[40] The new policy, learned on these slightly improved decisions, is then expected to be slightly better than the previous policy. This new policy is then used as the new rollout policy to create an even better classification data set, which in turn yields an even better policy, and so on. Pseudo code for a general form of classification-based reinforcement learning with rollouts is provided in Algorithm 4.

[38] Lagoudakis and Parr (2003)

[39] The first $\mathcal{D}_1$ is constructed using an initial given policy $\pi_0$, which usually is given by the uniformly random policy.

[40] This is apparent at the beginning of learning, where the random policy would select a random action, whereas the rollout-based decision is informed by the rewards observed during the rollouts.

---

**Algorithm 4** Classification-based reinforcement learning with rollouts (general form).

**Output:**

$\pi(s) : \mathcal{S} \to \mathcal{A}$     // *policy that returns an action $a \in \mathcal{A}$ for given state $s \in \mathcal{S}$*

$\pi \leftarrow$ uniform random policy
**for** $k = 0, 1, 2, \ldots$ **do**
    create classification data set $\mathcal{D}$ using rollouts with rollout policy $\pi$
    $\pi \leftarrow$ train classifier on $\mathcal{D}$
**end for**

---

## 2.4  RELATED WORK

The infeasibility of perfectly optimal behavior in non-trivial problems due to resource constraints faced by the decision maker is widely acknowledged in the artificial intelligence literature in general,[41] and in sequential decision making problems in particular.[42] In this section, I discuss existing ideas that account for limited resources in artificial intelligence research. Each upcoming chapter of this dissertation has its own section discussing the literature related specifically to the contents of that chapter.[43]

▸ OPTIMIZATION UNDER CONSTRAINTS & META-REASONING. One attempt to account for limited resources is to directly incorporate resource constraints into the utility-maximization problem. Utility-maximization under constraints has been studied in economics,[44] psychology,[45] and in artificial intelligence research.[46] One problem with this approach is that the constrained optimization problem is usually not actually simpler to solve than the original, unconstrained optimization problem.[47] In other words, the constrained optimization problem requires even more computational resources than the original problem. Therefore, while constrained optimization can sometimes accurately model the *outcomes* of decisions made under limited resources, it is often not a good model of the decision process itself, and therefore of limited value for creating artificial decision makers.

Utility-maximization under constraints is a form of optimal *meta-reasoning*[48] because the agent actively deliberates on a meta-level, trying to find the optimal cost-benefit trade-off. Several authors[49] have noted that optimal meta-reasoning leads to a problem of infinite regress, briefly described next. The meta-reasoning effort itself is a costly process. Thus, the agent should also balance the benefits and costs of this meta-level effort in a meta-meta-reasoning effort to arrive at a truly optimal solution. However, this meta-meta-reasoning effort is itself costly and thus demands for yet another level of reasoning, and so on.

*Information-theoretic bounded rationality*[50] also formulates bounded rationality as a constrained utility-maximization problem, where any resource con-

[41] For example, Russell and Norvig (2010, Sections 1.1 and 27.3).

[42] Sutton and Barto (2018, Part II)

[43] The chapter-specific related-literature sections are §3.3, §4.3, §5.5, and §6.9.

[44] Sargent (1993) and Stigler (1961)

[45] Anderson and Milson (1989), see also the discussion in Gigerenzer et al. (1999, p.10)

[46] Russell and Wefald (1991) and Gershman et al. (2015)

[47] Usually the constrained optimization problem is transformed into an unconstrained optimization problem using the method of Lagrange multipliers or its generalization, the Karush-Kuhn-Tucker conditions (Kuhn and Tucker, 2014).

[48] Zilberstein (2008) and Russell and Wefald (1991)

[49] Elster (1977), Russell and Wefald (1991), Gigerenzer et al. (1999), and Ortega et al. (2015)

[50] Braun et al. (2011), Ortega (2011), Genewein et al. (2015), and Grau-Moya (2016). A review is provided in Ortega et al. (2015).

straints are formulated as information-processing costs. An optimal solution to this problem can be formulated as a simple rejection sampling mechanism that resembles the $\xi$-satisficing policy presented in Chapter 6 of this dissertation (see Section 6.9 for a more detailed discussion about this relationship).

▸ BOUNDED OPTIMALITY.[51] Russell and Norvig (2010, p. 1050) write that a "bounded optimal agent behaves as well as possible, *given its computational resources*." Bounded optimality is related to optimization under constraints and meta-reasoning,[52] but it shifts the focus from the agent itself to the designer of the agent architecture. The designer is tasked to construct an "optimal program", which the agent then executes to exhibit boundedly optimal behaviors. The authors admit, however, that it is difficult to construct such optimal programs for anything but "very simple machines and somewhat restricted kinds of environments" (Russell and Norvig, 2010, p. 1050).

▸ ANYTIME ALGORITHMS. Dean and Boddy (1988, p. 49) define *anytime algorithms* as algorithms that "can be interrupted at any point during computation and return a result" and the "answers returned improve in some well-behaved manner as a function of time" (p. 52). Anytime algorithms are useful because they allow good behavior even if the decision-maker's deliberation process is stopped early.[53] The concept of an anytime algorithm, however, is not a formal approach of how to get to a good decision outcome for any given decision problem with limited resources. "Anytime" can instead be seen as a desirable property of an algorithm.

At the risk of oversimplification, we can characterize the approaches presented so far as "top-down" approaches: The starting point is a constrained optimization problem, which is solved to derive a decision-making strategy. If the constrained optimization procedure is successful, this leads to some form of optimal behavior. However, it is generally not clear whether the optimization procedure itself is successful or feasible at all.

In this dissertation I more often than not follow a "bottom-up" approach: I start from a simple model of bounded rationality or a decision heuristic (which is usually inspired by human or animal decision-making) and study how this model can be used to improve the resource efficiency of existing machine learning algorithms.[54]

▸ ECOLOGICAL RATIONALITY. In that regard my approach is related to the "bottom-up" approach known as *ecological rationality*.[55] This notion is inspired by Herbert Simon's idea that the rationality and performance of a decision-making strategy largely depends on the match between the strategy's structure and the structure of the decision environment.[56] Put differently, a very simple, boundedly rational decision heuristic can routinely make good inferences if it can exploit reliable structures in real-world decision environments. Moreover,

[51] Horvitz (1987), Etzioni (1989), Russell and Wefald (1991), and Russell and Subramanian (1994)

[52] Zilberstein (2008)

[53] Zilberstein (1995)

[54] See Katsikopoulos (2014) for a related discussion about the distinction between "idealistic" and "pragmatic" cultures of bounded rationality in economics and psychology.

[55] Gigerenzer et al. (1999), Gigerenzer and Selten (2002), Katsikopoulos (2011, 2014), Todd et al. (2012), Todd and Brighton (2016), and Brighton (2020)

[56] Simon (1956)

a collection of simple heuristics can become a powerful decision-making system if the decision maker knows which heuristic to use in which situation. This idea is related to the notion of the *adaptive toolbox*,[57] a metaphorical view of the mind as a collection of heuristics and building blocks that allow humans and animals to search for information and make decisions under limited resources.

The main questions addressed in the ecological rationality literature are: Which tools work well in a given environment, and why?[58] Which environmental structures are prevalent in real-world decision environments?[59] How can an agent learn to identify match between strategy and environment and thus choose the right tool at the right time?[60]

Note that following a bottom-up approach does not preclude us from also asking the question whether a certain simple decision heuristic occurs as the solution to some optimization problem. On the contrary, such results help the cause of ecological rationality because the nature of the optimization problem can provide information about the environmental structures in which the corresponding (optimal) heuristic performs well.

[57] For example, Gigerenzer and Selten (2002).

[58] Todd et al. (2012), Martignon and Hoffrage (2002), Katsikopoulos and Martignon (2006), Baucells et al. (2008), Gigerenzer and Brighton (2009), and Katsikopoulos et al. (2018)

[59] Şimşek (2013) and Şimşek et al. (2016)

[60] Rieskamp and Otto (2006)

Part II

BOUNDEDLY RATIONAL FUNCTION APPROXIMATION

*3*

# THE PREDICTIVE POWER OF SIMPLE REGRESSION MODELS

This chapter is based on Jan M. Lichtenberg and Özgür Şimşek (2017). "Simple regression models". In: *Imperfect Decision Makers: Admitting Real-World Rationality*. PMLR 58, pp. 13–25.

In this chapter I study the predictive accuracy of models of bounded rationality in supervised learning, and more specifically, in the regression task. Simple regression models have a long history in the psychology literature, where they have been compared to human judgements and classical statistical models such as ordinary least squares. However, there is a lack of a comprehensive empirical comparison of these models with state-of-the-art regression models from the supervised learning literature.

I report the results of such an empirical analysis on 60 real-world data sets. Simple regression models such as equal-weights regression routinely outperformed their state-of-the-art peers, especially on small training sets. However, averaged across all data sets, simple models showed lower predictive accuracy than their more complex counterparts in situations where plenty of data was available.

The main contribution of this chapter in the context of this dissertation is conceptual. The results suggest to use the amount of data available to a reinforcement learning agent as a main determinant in deciding *when* to use models of bounded rationality as function approximation architectures. Specifically, when only little or low-quality data is available, simple models of boundedly rationality seem to provide a promising alternative to more complex models as function approximators. Conversely, a simple function approximator is unlikely to rival a more complex function approximator when a lot of data is available. Making the sensible assumption that the amount and quality of data available to a reinforcement learning agent generally increases over time, the results suggest the use of an adaptive function approximation architecture that starts simple and becomes more complex over time.

A secondary contribution of the present study is that it complements the growing literature on comparing the predictive performance of boundedly rational prediction models to more complex statistical models,[1] building towards a more general theory of when and why simple models perform well.

A further contribution of this chapter is based on the following observation

[1] See Katsikopoulos et al. (2018) for a recent synthesis of existing work.

from the empirical analysis. There was no simple model that predicted well in all data sets; but in almost all data sets, there was at least one simple model that predicted well (across the entire learning curve, that is, even for large training set sizes). This opens up an interesting research direction for future work, which could examine models that adaptively choose between a few simple—but maximally different—simple models, as discussed in more detail in Section 3.5.[2]

In Section 3.1 I list existing simple regression models, define new ones, and describe how to estimate their parameters (Section 3.2). Section 3.3 defines three desiderata of a comprehensive empirical study to assess the predictive performance of simple regression models and reviews the existing literature on simple regression models, concluding that none of the existing work meets the desiderata defined. In Section 3.4 I report the results of a comprehensive empirical study that compares simple models to state-of-the-art regression algorithms on 60 data sets. Finally, Section 3.5 discusses the results.

## 3.1   SIMPLE REGRESSION MODELS

Recall from Section 2.1 that a *regression model* $f$ is a model that predicts a real-valued output $\hat{y}$ given some $p$-dimensional input vector $\boldsymbol{x}$, that is,

$$\hat{y} = f(\boldsymbol{x}, \boldsymbol{\beta}).$$

The simple models we consider are special instances of the standard linear regression model[3]

$$\hat{y} = \beta_0 + \gamma \sum_{j=1}^{p} x_j \alpha_j, \tag{3.1}$$

and share the following properties: (a) parameters $\alpha_j$ are chosen heuristically (for example, equal weights), and (b) parameters $\alpha_j$ can be estimated or determined independently of the location parameter $\beta_0$ and the scale parameter $\gamma$. Intuitively, the weighted sum determines the nature of how the predictors are combined or selected. The two parameters $\beta_0$ and $\gamma$ then determine the location and scale of this weighted sum, respectively. Different simple models correspond to different ways of determining $\alpha_j$. Estimation of $\beta_0$ and $\gamma$ is the same for all simple models considered and will be explained in the following section.

Given some training data $(y_i, \mathbf{x_i})$, $i = 1, \ldots, n$, we assume that predictors and the response variable are *centered*, that is,

$$\bar{\boldsymbol{y}} = \frac{1}{n} \sum_{i=1}^{n} y_i = 0 \quad \text{and} \quad \bar{\boldsymbol{x}}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij} = 0 \text{ for all } j = 1, \ldots, p,$$

and *scaled*, that is,

$$s_{\boldsymbol{y}} = \frac{1}{n} \sum_{i=1}^{n} y_i^2 = 1 \quad \text{and} \quad s_{\boldsymbol{x}_j} = \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 = 1 \text{ for all } j = 1, \ldots, p.$$

[2] See also Şimşek and Buckmann (2017).

[3] Setting $\beta_j = \gamma \alpha_j$, we obtain the classical formulation of the linear regression model: $\hat{y} = \beta_0 + \sum_{j=1}^{p} x_j \beta_j$.

A centered and scaled variable will be called *standardized*. Scaling the response variable $y$ does not affect the relative model performance but simplifies the analysis across different data sets. Furthermore, predictors are said to be *directed* if they correlate non-negatively with the response.[4] We will now define the simple models considered in the upcoming empirical analysis.

▶ MEAN PREDICTION. This is the simplest available model and always predicts the mean value of the response calculated on the training data. The corresponding model can be written as

$$\hat{y} = \beta_0.$$

Mean prediction is appropriate when no predictive predictor is available. We normally do not consider data sets that do not contain predictive predictors. Therefore, mean prediction does not play a role in supervised learning in general. In this chapter the mean prediction model serves as a baseline model.

▶ RANDOM WEIGHTS. This is perhaps the most "improper" model[5] one could imagine. Once standardized and directed, each predictor is assigned a random weight stemming from a uniform distribution, that is,

$$\hat{y} = \beta_0 + \gamma \sum_{j=1}^{p} \omega_j x_j,$$

where $\omega_j \sim \mathcal{U}(a,b)$. Different authors used different values for $a$ and $b$. We used $a = 0$ and $b = 1$. More than 80 years ago, Wilks (1938) showed that the correlation of predictions of two independent random weights models tends to 1 with an increasing number of positively inter-correlated variables. Random weights should be outperformed by equal weights due to smaller variance of the latter.[6] We still shall include random weights as a benchmark model in our empirical analysis.

▶ EQUAL WEIGHTS. The *equal weights* (EW) regression model[7] takes all standardized and directed features into account and weights them equally, that is,

$$\hat{y} = \beta_0 + \gamma \sum_{j=1}^{p} x_j. \tag{3.2}$$

Using the assumption that all predictors are directed, equal weights has only two free parameters (location and scale) left.

▶ CORRELATION WEIGHTS. This model weights all predictors by their uni-variate correlation with the response, that is,

$$\hat{y} = \beta_0 + \gamma \sum_{j=1}^{p} r_{yx_j} x_j,$$

where $r_{yx_j}$ is the sample Pearson correlation coefficient between the response $y$ and predictor $x_j$. Correlation weights has to estimate $p + 2$ parameters. However, these coefficients are still easier to calculate than ordinary least squares

weights both in terms of computational complexity and numerical stability issues. Whereas OLS suffers, for example, from the multicollinearity problem, the correlation coefficients are calculated independently of each other and independently of $\beta_0$ and $\gamma$.

▸ RANKED CORRELATION WEIGHTS. This model does not need the exact values of the correlation weights but only their ranks, that is, their relative order. The corresponding model can be written as

$$\hat{y} = \beta_0 + \gamma \sum_{j=1}^{p} \rho_j\, x_j, \quad \text{where} \quad \rho_j = rank(r_{yx_j}).$$

The lowest correlated cue has rank 1 and the highest correlated cue has rank $p$. Ties are assigned the average rank.[8] The ranked correlation weights model might be easier to estimate and thus more robust than correlation weights[9] or OLS but still allows for differential weighting of multiple predictors as opposed to equal-weighting or single-cue strategies.

▸ SINGLE-CUE REGRESSION. This model considers only the predictor that has the highest uni-variate correlation with the response among all available predictors. The corresponding model can be written as

$$\hat{y} = \beta_0 + \gamma x_1,$$

where $x_1$ is the cue that is most correlated with the criterion $y$. In order to determine *the* single-cue, we estimate the correlations between all predictors and the response and choose the one with the highest absolute value.[10]

## 3.2 PARAMETER ESTIMATION FROM TRAINING DATA

Unless specifically stated otherwise, we assume that location parameter $\beta_0$ and the scale parameter $\gamma$ are calculated using *simple linear regression* (SLR).[11] The parameters $\alpha_j$ are set or estimated depending on the respective algorithm and always before the estimation of $\beta_0$ and $\gamma$.

SLR is much easier to estimate than OLS in general as it does not involve the inversion of matrices but only simple estimates of scale and co-variation. Let $c(\boldsymbol{x}_i) = \sum_{j=1}^{p} x_j \alpha_j$ denote the weighted sum of predictors for observation $i$ and let $\boldsymbol{c} = (c(\boldsymbol{x}_1), \ldots, c(\boldsymbol{x}_n))^T$ denote the vector of weighted sums for observations 1 to $n$. Then, the SLR estimates for the univariate regression model (3.1) are given by

$$\gamma = r_{yc} \frac{s_y}{s_c} \quad \text{and} \quad \beta_0 = \bar{y} - \gamma\bar{c},$$

where $r_{yc}$ is the sample correlation coefficient between $\boldsymbol{y}$ and $\boldsymbol{c}$, and $s_y$ and $s_c$ are the standard deviations of $\boldsymbol{y}$ and $\boldsymbol{c}$, respectively.

Note that $\beta_0$ can be omitted (set to 0) for all models when predictors and responses are standardized. Some authors do not include the scale parameter

[8] For example, the vector $(7, 4, 4, 2)$ has ranks $(4, 2.5, 2.5, 1)$.

[9] Our implementation of ranked correlation weights actually first estimates all correlations and then assigns ranks. However, there may be simpler ways to (approximately) determine the ranking of correlations.

[10] Estimation of single-cue regression is not less complex than estimation of correlation weights as we have to calculate all $p$ predictor-response correlations in order to determine the single-cue. However, there may be simpler ways to (approximately) determine the single-cue. In addition, single-cue regression is "simpler" at decision time, where it requires only the information of one predictor.

[11] SLR is sometimes also called uni-variate regression.

$\gamma$ when the loss function is invariant under scaling. In this chapter we are interested in regression under squared error loss, which is sensitive to scaling. Inclusion of $\gamma$ is therefore crucial.

### 3.3    DESIDERATA FOR AN EMPIRICAL ANALYSIS OF SIMPLE REGRESSION MODELS & LITERATURE REVIEW

Ideally, an empirical study assessing the predictive performance of simple regression models would have the following qualities:

1.  It compares simple models with state-of-the-art regression algorithms from the supervised learning literature,

2.  on a large variety of data sets from different domains (rather than just a single data set or just simulated data),

3.  using a regression-adequate evaluation metric in a prediction context (as opposed to a data-fitting context).

Table 3.1 shows all studies reviewed in the next section and previous comparative studies, in chronological order. The table shows that none of the empirical studies meet all three requirements listed above. Specifically, studies that use a regression-adequate evaluation function in a prediction context (highlighted in green color) either do not use real-world data or just use a single data set (for example, US election data in Cuzan and Bundrick (2009) and Graefe (2015)).

   In the remainder of this section we review the existing literature about simple regression models on a model-by-model basis.

▶ EQUAL WEIGHTS. The EW model has a long history of use in the social sciences. It appeared in a seminal paper by Dawes and Corrigan (1974) that showed that *even* so-called improper models (such as EW) could outperform human expert judgements. The article also demonstrated that EW can compete well with OLS on real-world data sets, stimulating further work on equal-weighting models in the 1970s, continuing to this day.[12] Equal-weighting models have also been found to be useful in other types of problems, including paired comparison[13] and portfolio optimization.[14]

   Equal weights has been discussed analytically in regression: Einhorn and Hogarth (1975) compare expected mean squared errors of EW and OLS with each other, providing intuition on how $n$ and $p$ influence the relative performance difference between the two algorithms. Their theorem depends on the knowledge of the true error distribution, though, and is thus of little practical value.

   Wainer (1976) states that "it makes no nevermind" whether one uses equal weights or OLS as the loss in variance explained is small under some general conditions on the true coefficients.[15] However, the proportion of variance explained on the training data is a bad proxy for assessing the prediction performance of a model on unseen data. Davis-Stober et al. (2010) provide analytical

[12] Einhorn and Hogarth (1975), Wainer (1976), Davis-Stober et al. (2010), and Graefe (2015)

[13] Gigerenzer et al. (1999)

[14] DeMiguel et al. (2009b)

[15] See also Laughlin (1978) and Wainer (1978).

| Paper | Model(s) | Research style | Performance metric |
|---|---|---|---|
| Wilks (1938) | RW | Theoretical | Correlation |
| Wesman and Bennett (1959) | UW | Empirical, College Grades | Correlation |
| Schmidt (1971) | UW | MC-Simulation; Gaussian Data | Correlation |
| Dawes and Corrigan (1974) | EW, RW | Theoretical and Empirical (Social sciences / Psychology) | Correlation |
| Einhorn and Hogarth (1975) | UW, EW | Theoretical | Correlation & $\frac{MSE^{EW}}{MSE^{OLS}}$ |
| Wainer (1976) | EW | Theoretical | $R^2$ |
| Laughlin (1978) | EW | Theoretical | $R^2$ |
| Wainer (1978) | EW | Theoretical | $R^2$ |
| Dawes (1979) | | Follow up to Dawes and Corrigan (1974), similar results. | |
| Ehrenberg (1982) | OLS | Theoretical | $R^2$ |
| Barron and Barrett (1996) | RCW | Synthetic Data | Accuracy |
| Hertwig et al. (1999) | QuickEst | Theoretical; City data set | MAE & Frugality |
| Dana and Dawes (2004) | CW, EW, SC | 5 real data sets and syn. data | validated $R$ |
| Bobko et al. (2007) | EW | Empirical | Correlation |
| Helversen and Rieskamp (2008) | Mapping Model, QuickEst | Simulation and human experiments | MSE |
| Cuzan and Bundrick (2009) | EW | US election data | MAE |
| Waller and Jones (2009) | CW, OLS | Theoretical | $\cos(\boldsymbol{\beta}^{OLS}, \boldsymbol{\beta}^{CW})$ & drop in $R^2$ |
| Davis-Stober et al. (2010) | EW, SC, UW | Theoretical and simulated data | $\mathrm{MSE}_{\boldsymbol{\beta}} = \mathbb{E}\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|$ |
| Davis-Stober (2011) | | Similar setup as in Davis-Stober et al. (2010) | |
| Woike et al. (2012) | QuickEst, Zig-QuickEst | Empirical on 99 data sets with dichotomized cues. | normalized MSE |
| Graefe (2015) | EW, UW | US Election data | MAE |

Table 3.1: Summary of papers that investigate simple regression models. Performance metrics that are highlighted with ▇ are not appropriate for the regression problem. Performance metrics that are highlighted with ▇ are used in regression, but are not appropriate for the *prediction* problem investigated here (these metrics measure how well the model fits the training data). Performance metrics that are highlighted with ▇ are appropriate for assessing the performance of regression models in a prediction context.

bounds on the expected mean squared error of the equal weights parameter vector, given by $MSE_{\boldsymbol{\beta}} = \mathbb{E}\|\boldsymbol{\beta}^{\text{EW}} - \boldsymbol{\beta}^{\text{true}}\|$.

▷ CORRELATION WEIGHTS. Waller and Jones (2009) compare the correlation weights model to the OLS model. They derive conditions under which the correlation weight $r_{yx_i}$ and the regression weight $\beta_i^{OLS}$ of a feature $x_i$ are equal or maximally similar. They also describe the loss in $R^2$ (the coefficient of determination) when switching from OLS to correlation weights. However, their analysis remains in a data *fitting* context. Dana and Dawes (2004) find that Correlation Weights is equal or outperforms OLS in 4 out of 5 data sets across different training set sizes. However, their analysis is not based on prediction error, but on "validated $R^2$". Several other studies compared Correlation Weights to (mostly) OLS on synthetic data sets[16] or one single data set.[17]

[16] Claudy (1972)

[17] Goldberg (1972)

▷ RANKED CORRELATION WEIGHTS. A model similar to the ranked correlation weights model is compared to true weights and equal-weighting models in the context of *multiattribute decision making*[18] in Barron and Barrett (1996). We do not know of any study that compares the prediction accuracy of ranked correlation weights models to other regression models in a regression context.

[18] Find the alternative with the highest response value among a set of $n \geq 2$ alternatives.

▷ SINGLE-CUE REGRESSION. Dana and Dawes (2004) find that single-cue regression[19] is inferior to OLS on 5 data sets overall, only outperforming the latter for the smallest examined training set size on 2 of the 5 data sets.

[19] Called "Take the Best Weights" in their paper.

Single-predictor models as well as related lexicographic models (use one predictor at a time, see also Section 2.2.2) have been studied in machine learning[20] and psychology[21]. However, all of these studies are concerned not with the regression problem but with classification or paired comparison problems.

[20] Holte (1993) and Şimşek and Buckmann (2015)

[21] Tversky and Kahneman (1973), Gigerenzer et al. (1999), and Hogarth and Karelaia (2005)

## 3.4    EMPIRICAL ANALYSIS

Here we report the results of a large-scale empirical analysis of simple regression models.

▷ DATA SETS. We used 60 publicly-available data sets from varying domains. Sources included online data repositories, statistics and data mining competitions, packages for R statistical software, textbooks, and research publications. The number of observations ranged from 31 to 39644, the number of predictors from 3 to 52. Table 3.2 shows the number of observations and the number of predictors in each data set. Detailed information about the data sets can be found in Appendix C.

For each data set, the response has been standardized and all predictors have been standardized and directed. Missing predictor values have been mean-imputed and observations with missing response values have been removed

| Id | Name | Obs. | Predictors | Id | Name | Obs. | Predictors |
|---|---|---|---|---|---|---|---|
| 1 | Abalone | 4177 | 8 | 31 | Land | 67 | 4 |
| 2 | Afl | 41 | 5 | 32 | Lung | 654 | 4 |
| 3 | Air | 41 | 6 | 33 | Mammal | 58 | 7 |
| 4 | Airfoil | 1503 | 5 | 34 | Medexp | 5574 | 14 |
| 5 | Algae | 340 | 11 | 35 | Men | 34 | 3 |
| 6 | Athlete | 202 | 8 | 36 | Mileage | 398 | 7 |
| 7 | Basketball | 96 | 4 | 37 | Mine | 44 | 4 |
| 8 | Birthweight | 189 | 8 | 38 | Monet | 430 | 4 |
| 9 | Bodyfat | 252 | 13 | 39 | Mortality | 60 | 15 |
| 10 | Bone | 42 | 6 | 40 | Movie | 62 | 12 |
| 11 | Car | 93 | 21 | 41 | Mussel | 44 | 8 |
| 12 | Cigarette | 528 | 7 | 42 | News | 39644 | 52 |
| 13 | Concrete | 1030 | 8 | 43 | Obesity | 136 | 11 |
| 14 | Contraception | 152 | 6 | 44 | Occupations | 36 | 3 |
| 15 | Cpu | 209 | 6 | 45 | Pinot | 38 | 6 |
| 16 | Crime | 47 | 15 | 46 | Pitcher | 176 | 15 |
| 17 | Diabetes | 442 | 10 | 47 | Plasma | 315 | 12 |
| 18 | Diamond | 308 | 4 | 48 | Prefecture | 45 | 5 |
| 19 | Dropout | 63 | 17 | 49 | Prostate | 97 | 8 |
| 20 | Excavator | 33 | 4 | 50 | Reactor | 32 | 10 |
| 21 | Fish | 413 | 3 | 51 | Rebellion | 32 | 6 |
| 22 | Fuel | 51 | 5 | 52 | Recycle | 31 | 7 |
| 23 | Gambling | 47 | 4 | 53 | Rent | 2053 | 10 |
| 24 | Highway | 39 | 11 | 54 | Salary | 52 | 5 |
| 25 | Hitter | 263 | 19 | 55 | Sat | 50 | 4 |
| 26 | Home | 3281 | 4 | 56 | Schooling | 3010 | 22 |
| 27 | Homeless | 50 | 7 | 57 | Tip | 244 | 6 |
| 28 | Infant | 101 | 3 | 58 | Vote | 159 | 5 |
| 29 | Laborsupply | 5320 | 5 | 59 | Wage | 4360 | 10 |
| 30 | Lake | 69 | 10 | 60 | Whitewine | 4898 | 11 |

Table 3.2: Data sets used in the empirical comparison.

from the data set.

▸ BENCHMARK MODELS. We chose to include three benchmark models, described below. Two of them are state-of-the-art regression models. We included OLS for historic reasons.

1. Ordinary least squares (OLS) minimizes the mean squared error between predicted and true values on the training data.

2. Elastic net regression[22] is a state-of-the-art regularized linear regression model. Regularized linear models originally have been developed to overcome the estimation difficulties of OLS.[23] They attempt to optimize prediction accuracy by finding the happy medium between simplicity and complexity. Regularized linear models are explained and discussed in detail in the next chapter (see, in particular, Section 4.1). We also tested ridge regression and the Lasso in our empirical study and found their results to be very similar to those of the elastic net and thus omit these two models from our analysis for brevity.

3. Random forest regression[24] is a non-parametric and non-linear regression model. The random forest regression model is an ensemble of regression

[22] Zou and Hastie (2005)

[23] Hoerl and Kennard (1970)

[24] Breiman (2001)

trees, whose individual predictions are combined (usually averaged) to produce a single prediction output. The model's classification-equivalent (often simply called random forests or random decision forests) was among the best models in a large-scale comparison of 179 general-purpose classification algorithms across 121 data sets.[25] It also has been shown to perform remarkably well on tiny training data sets.[26]

[25] Fernández-Delgado et al. (2014)

[26] Buckmann and Şimşek (2017)

Implementation details for each of the benchmark models are provided in Appendix A.

▶ RESULTS. We show three sets of results. Figure 3.1 shows the mean RMSE of each algorithm across 60 data sets, computed using 10-fold cross validation. These estimates of the prediction error correspond to large training set sizes relative to the total size of the data set (90% of available observations).
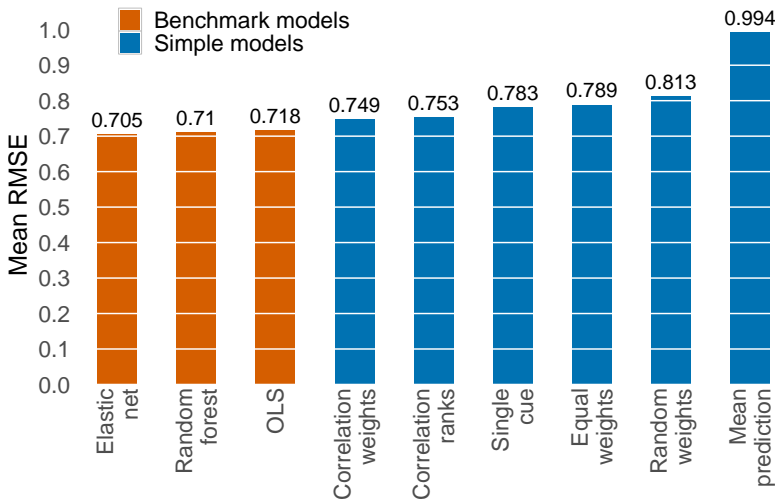


Figure 3.1: 100 x 10-fold cross validation-based root mean squared error (RMSE) across 60 data sets.

Figure 3.2 shows learning curves averaged across 60 data sets as the training set varied in size from 4 to 100. The figure shows learning curves for all models except mean prediction, random weights, and OLS.[27] The test set consisted of 10% of the total number of observations in the data set and did not overlap with the corresponding training set. The estimation procedure was repeated 100 times for each training set size and algorithm. The learning curves are not monotonically decreasing towards the end because only a subset of the data sets are large enough for higher training-set sizes. The number of data sets available for a given training set size is indicated at the top of the figure.

[27] OLS overfits for small ratios of $n/p$. The resulting average RMSEs were outside of the figure boundaries due to some data sets with a large number of predictors $p$.

Finally, we present learning curves of various algorithms in individual data sets. Figure 3.3 shows learning curves in data sets *diabetes*, *prostate*, and *sat*. Figures B.1 to B.3 in the Appendix present individual learning curves for all remaining data sets.

We first compare simple regression models to benchmark models collectively, then comment on results within the groups of simple models and bench-
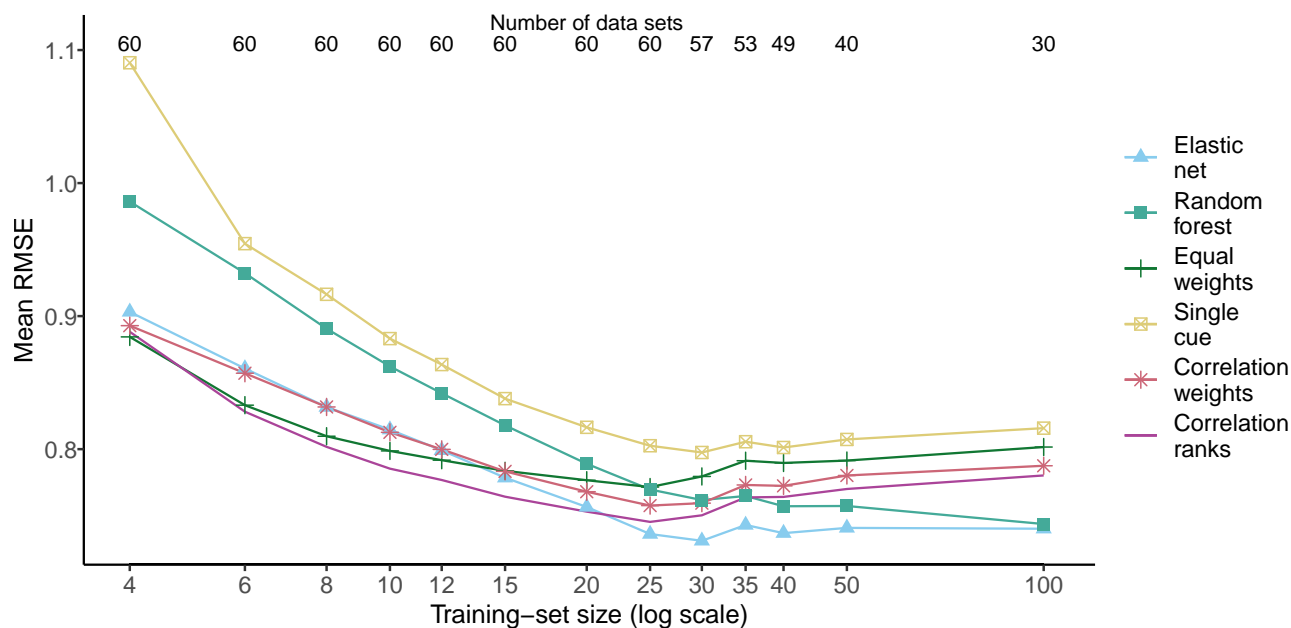
Figure 3.2: Learning curves averaged across 60 data sets.

mark models, respectively.

Averaged across 60 data sets, simple models were collectively outperformed by all benchmark models for larger training set sizes.[28] However, equal weights and ranked-correlation weights outperformed all competing models for training set sizes below 15 on the mean learning curve. In addition, the learning curves in individual data sets show that, for many data sets, there is at least one simple model that performed well across large parts of the learning curve. Let the minimum error curve be defined as the algorithm with minimum error among all algorithms as a function of training set size. Then, in 22 out of 60 data sets, simple models occupied the entire minimum error curve except for possibly one training-set size. In another 21 data sets, simple models occupied at least half of the minimum error curve. The 17 remaining data sets were dominated by benchmark models.

On many data sets we observed that some simple models performed remarkably well while other simple models performed quite poorly (as opposed to a situation in which all simple models performing approximately equally well or equally poorly). A good example is the *sat* data set shown in Figure 3.3, which is one of the few data sets for which both equal weights and ranked correlation weights perform poorly, but for which single-cue is the best-performing algorithm across the entire learning curve. Conversely, on the *diabetes* data set, single-cue performs considerably worse than most other models while correlation weights performs best until a training set size of 30.

The data sets *prostate* and *diabetes* have been used to illustrate the favorable prediction performance of the elastic net and other sophisticated regression

[28] The end of the learning curve shows the average across all 30 data sets that are large enough for training set sizes of 100. The cross validation-based analysis of Figure 3.1 shows the average across all 60 data sets for training set sizes ranging from 27 to 35679 observations, corresponding to 90% of the total size of the respective data sets. Both analyses show qualitatively similar results.
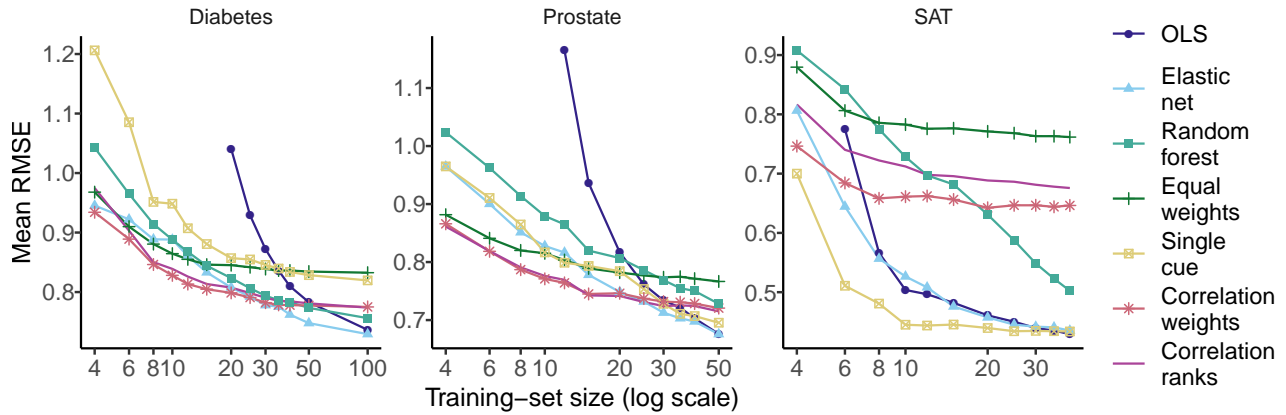
[29] Tibshirani (1996), Efron et al. (2004), and Zou and Hastie (2005)

models in the past.[29] Figure 3.3 shows that correlation weights outperformed the elastic net in both data sets in training set sizes smaller than 30.

On the mean learning curve, ranked-correlation weights outperformed all other simple models across the entire curve. However, on individual data sets, ranked-correlation weights was often outperformed by one or more of the other simple models. In fact, in almost all data sets, the learning curve of ranked-correlation weights lies in between those of equal weights and correlation weights, independent of which of the two latter models performed better. This confirms the intuition that ranked correlation weights is an intermediately-complex model that is able to perform well in situations of scarce information (similar to equal weights) but can also exploit the benefits of weighting predictors differently when there is enough information to reliably estimate the ranking of predictors.

Both equal weights and single-cue regression share the property of performing either very well or very poorly on many data sets. Single-cue regression was the second-worst or worst model over the entire range of training set sizes in 23 of the 60 data sets. However, it was also the best-performing model across the entire learning curve in *sat* and across large parts of the learning curve in *bodyfat*. Equal weights outperformed all other models across the entire learning curve in data sets *bone, fuel, pinot, reactor, rent*, and *wage*. But it was by far the worst model on large parts of the learning curve in *diamond, excavator, fish* and *sat*.

Among benchmark models, OLS was outperformed by random forest regression and the elastic net, both on average and individually on most of the data sets. Elastic net more often than not outperformed random forest regression, especially on small training sets.

3.5   DISCUSSION

The empirical analysis presented in this chapter shows that simple regression models, for example, equal-weights regression, routinely outperform not only

multiple linear regression but also state-of-the-art regression models, especially on small training sets.

We found that none of the simple models we examined predicted well in all data sets. But, in almost all data sets, there was at least one simple model that predicted well.

Because OLS has severe estimation difficulties with small training sets, it would be reasonable to expect simple regression models to perform better than OLS on small training sets. This was also observed previously in the literature.[30] However, we did not expect the simple models to be able to compete with state-of-the-art regularized linear models such as elastic-net regression or random forest regression.

Sparsity-inducing regularized linear models attempt to optimize prediction accuracy by searching through a possibly infinite-dimensional hypothesis space of linear models, ranging from a sparse linear model to the full, complex OLS solution. All simple models considered here are also special cases of the linear regression model. And yet—even though we tested only four such simple models—these models could sometimes outperform the carefully-optimized elastic net.

These results indicate that it may be possible to substantially reduce the size of the hypothesis space of linear models and to make good inferences nonetheless. In other words, it is possible to make good inferences based on a few simple models if only one knows which simple model to choose.

Future work could examine models that adaptively choose between a few but maximally different simple models. For example, a model that chooses between single-cue, equal weights, and ranked correlation weights, based only on information in the training data, could be a fast and robust alternative to current state-of-the-art models, while being computationally less challenging. The main question will be whether this algorithm can choose the most appropriate simple model based on only a small number of training examples.

An important research direction is to examine whether people can intuitively pick the right simple model for a given problem. Such a finding may explain how people often make good decisions despite their bounded cognitive capacities.

[30] For example, in Dana and Dawes (2004) and Davis-Stober et al. (2010).

4

# BOUNDED RATIONALITY AS REGULARIZATION: SHRINKAGE TOWARD EQUAL WEIGHTS

This chapter is based on Jan M. Lichtenberg and Özgür Şimşek (2019b). "Regularization in directable environments with application to Tetris". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 3953–3962.

The previous chapter showed that models of bounded rationality such as equal-weights regression encode useful prior knowledge and therefore can provide a computationally less expensive alternative to more complex machine learning models, especially in situations where only limited amounts of training data are available. The results also showed that, on average, these simple models cannot reach the performance of the more complex models when training data is abundant.

Here we are interested in creating a model that can get the best of both worlds by adapting to the amount of data available. Regularized linear models are a good candidate for doing exactly that: depending on the choice of a parameter, called regularization strength, these linear models interpolate between a purely data-driven approach and an approach that relies on the prior knowledge encoded in the regularization term.

One could expect that such an adaptively regularized linear model should always perform at least as well as a rigid simple linear model, because the former could simply emulate the latter in a situation of limited available data. Interestingly, the results in the previous chapter showed that simple models routinely outperformed elastic-net regression, a popular general-purpose regularized linear model.[1] One possible explanation is that the simple models investigated encode some relevant prior knowledge that is not covered by elastic-net regularization.[2] This leads to the question: *Can we inform statistical machine learning models with the prior knowledge encoded in existing models of bounded rationality?*

In this chapter, I propose a new regularization term for linear models called *shrinkage toward equal weights*, or STEW. With increasing regularization strength, STEW regularization shrinks the weights of a linear model toward each other, resulting in an equal-weighting solution in the limit of infinite regularization. We study the STEW regularization term in the regression task. In particular, we study properties of the equal-weights regression model as a source of intuition regarding when, and why, STEW can perform well. We provide theoretical

[1] Similar findings were reported for the paired comparison tasks in Buckmann and Şimşek (2017).

[2] A different, complementary explanation is that the regularization strength cannot always be tuned appropriately using only information in the training data.

and empirical evidence that the equal-weights estimator has relatively low bias compared to comparable constrained linear models and that this bias is further reduced when *feature directions* are known. The direction of a feature indicates whether the feature is associated positively or negatively with the response variable. In many applications, feature directions are known or can be estimated with ease.[3]

Our empirical analysis shows that these properties translate from the equal-weights model to STEW. When information on directions is available, STEW routinely outperforms existing regularized models including the non-negative Lasso, which also incorporates knowledge about feature directions. Unlike methods that are based on non-negativity constraints, we found STEW to be robust when the underlying assumption of known feature directions was violated, that is, when the information about directions was unreliable or absent.

This chapter provides three main contributions.

The first contribution of this chapter is conceptual. We show that statistical machine learning models can benefit from incorporating prior knowledge encoded in models of bounded rationality from the psychology literature. In particular, STEW regularization can fruitfully incorporate a prevalent and easy-to-estimate form of domain knowledge—feature directions—in ways that existing models cannot.

The second contribution is algorithmic. STEW regularization provides a relatively simple way of smoothly transitioning between a model of bounded rationality and a fully data-driven model. This suggests a straightforward way of building agents that use their bounded cognitive resources adaptively, based on how much time, computation power, and data are available. We develop this idea further in the upcoming chapter.

A third contribution is that our theoretical analysis of equal-weighting models provides insights on why equal-weighting strategies—which are used across a wide variety of academic studies as well as critical real-life situations[4] —perform so well.

The chapter is structured as follows. Section 4.1 provides technical background. Section 4.2 defines the STEW regularization term and Section 4.3 discusses related work. Section 4.4 contains a theoretical bias-variance analysis of equal-weighting models, followed by an empirical analysis of STEW-regularized regression models in both simulated (Section 4.5.1) and real-world environments (Section 4.5.2). Section 4.6 concludes with a discussion of the results presented in this chapter.

## 4.1 BACKGROUND

We consider the linear regression problem described in Section 2.1.2. The objective is to predict a response $y \in \mathbb{R}$ by

$$\hat{y} = \beta_0 + \sum_{j=1}^{p} \beta_j x_j, \tag{4.1}$$

[3] We briefly discussed this claim in Section 2.2.1 in the context of single-shot decision making. Chapter 5 addresses the problem of learning feature directions in sequential decision making problems.

[4] See, for example, Katsikopoulos et al. (2020, Section 1.1) or Graefe (2015) for work on equal-weighting strategies in the context of election forecasts. See also the discussion on equal-weighting strategies in Section 2.2.3.

where $x_1, \ldots, x_p$ are feature values and $\beta_0, \ldots, \beta_p$ are feature weights. To estimate feature weights, a training set of $n$ observations, $(y_i, x_{1i}, \ldots, x_{pi}), i = 1, \ldots, n$, is available. Following the standard in the regularization literature,[5] we assume that features and responses are standardized so that $\frac{1}{n} \sum_{i=1}^{n} y_i = 0$, $\frac{1}{n} \sum_{i=1}^{n} x_{ij} = 0$, and $\frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 = 1$, for $j = 1, \ldots, p$. It follows that $\beta_0$ is zero and thus can be omitted. We use matrix notation, with $y \in \mathbb{R}^n$ denoting the response vector, $X \in \mathbb{R}^{n \times p}$ the feature matrix, and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_p)$ the weight vector. The Ordinary least squares (OLS) estimate is the set of weights that minimizes the residual sum of squares $\|y - X\boldsymbol{\beta}\|_2^2$ on the training set.

[5] For example, Friedman et al. (2001).

▸ REGULARIZED LINEAR MODELS. Most regularized linear models minimize a penalized residual sum of squares of the form $\mathcal{L}(\boldsymbol{\beta}, \lambda) = \|y - X\boldsymbol{\beta}\|_2^2 + \lambda P(\boldsymbol{\beta})$, where $P$ is the penalty function and $\lambda \geq 0$ is the strength of the penalty. Well known penalty functions use the $l_q$-norm of the weight vector, $\|\boldsymbol{\beta}\|_q$. For example, ridge regression[6] uses the $l_2$ penalty, the Lasso[7] uses the $l_1$ penalty, and the elastic net[8] uses a convex combination of the $l_1$ and the $l_2$ penalties. These models shrink all weights toward zero as $\lambda \to \infty$. We refer to them as *models that shrink toward zero*.

[6] Hoerl and Kennard (1970)
[7] Tibshirani (1996)
[8] Zou and Hastie (2005)

▸ EQUAL-WEIGHTING MODELS. In this chapter the term "equal-weighting model" refers to a linear regression model where all feature weights have the same value, called the equal-weighting constant and denoted by $(\gamma)$. That is, the model is given by

$$\hat{y} = \gamma \sum_{j=1}^{p} x_j. \tag{4.2}$$

We define *Equal Weights* (EW) as the least-squares estimator of $\gamma$ and denote the corresponding estimate with $\gamma_{EW}$.

▸ DIRECTABILITY OF FEATURES. Feature directions were introduced in Section 2.2.1. Notice that the EW model as defined in Equation 4.2 is a sensible model only if the features are directed so that the true weights have identical signs. In this chapter we study the EW model and other models under various assumptions about the directability of features in the decision environment.

   The rationale for the use of EW in psychology and decision making is the assumption that people are good in choosing relevant features and know—through intuition or past experience—how to direct them.[9] The model we propose, STEW, can also use this knowledge fruitfully.

[9] Einhorn and Hogarth (1975)

## 4.2   SHRINKAGE TOWARD EQUAL WEIGHTS

Motivated by the surprisingly high performance of equal-weighting models in the literature—not only in regression but also in classification, paired comparison, and portfolio optimization—we propose to use the equal-weights model as a prior in regularization. In other words, we make the initial assumption that

*features have equal impact on the response variable.* This assumption leads to the regularization term $\sum_{i<j} ||\beta_i| - |\beta_j||^q$, for $q > 0$, which penalizes differences in the magnitude of the weights. It leaves the choice of feature directions free. However, the differences of absolute values within the penalty function make the loss function difficult to optimize. We therefore use a penalty function that assumes a directable environment.

We define *shrinkage toward equal weights*, or STEW, as the regularization term for linear models that penalizes the $l_q$-norm of *all* pairwise differences between weights. In the context of regression, STEW regularization leads to the minimization of the loss function below:

$$\mathcal{L}^{STEW}(\boldsymbol{\beta}, \lambda, q) = \|\boldsymbol{y} - \boldsymbol{X\beta}\|_2^2 + \lambda \sum_{i<j} |\beta_i - \beta_j|^q, \tag{4.3}$$

where $q > 0$ and $\lambda \geq 0$ determine the regularization behavior.

When $\lambda = 0$, STEW is equivalent to OLS, just like existing regularized linear models that shrink weights toward zero. However, with increasing regularization strength $\lambda$, STEW shrinks weights toward each other rather than toward zero. In the limit as $\lambda \to \infty$, STEW becomes equivalent to EW, with all weights converging to $\gamma_{EW}$. Figure 4.1 illustrates this difference in regularization behavior for STEW with $q = 1$ and $q = 2$ compared to Lasso and ridge regression on the *Rent* data set.
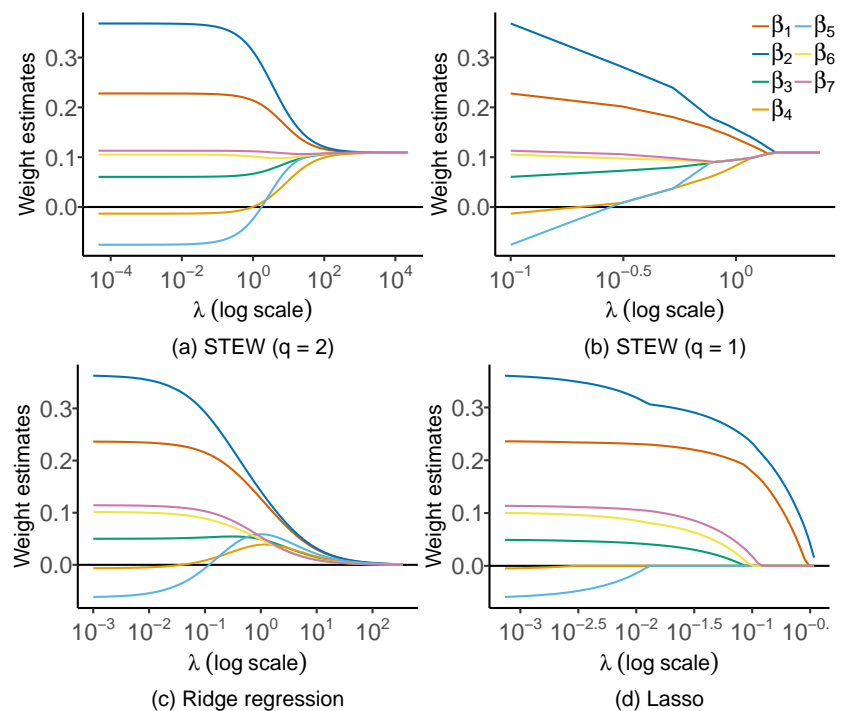


Figure 4.1: Weight estimates, as a function of the regularization strength $\lambda$, for STEW with $q = 1$ and $q = 2$, ridge regression, and the Lasso on the *Rent* data set with seven standardized features. The Rent data set was introduced in Section 2.1.1.

In the remainder of this chapter we refer to the application of STEW regularization in a linear regression model whenever we talk about STEW or the

STEW model. In Chapter 5, however, we will use the STEW regularization term to regularize a linear discrete choice model.

▸ PARAMETER ESTIMATION. In matrix notation, Equation 4.3 can be written as follows:

$$\mathcal{L}^{STEW}(\boldsymbol{\beta}, \lambda, q) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{D}\boldsymbol{\beta}\|_q^q, \qquad (4.4)$$

where $\boldsymbol{D}$ is a pairwise difference matrix with $p(p-1)/2$ rows and $p$ columns. The rows of $\boldsymbol{D}$ are the unique permutations of the vector $(1, -1, 0, \ldots, 0)$ with $p$ entries such that the entry '1' precedes the entry '−1'. For example, if there are $p = 4$ predictors the pairwise difference matrix is given by

$$\boldsymbol{D} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix}.$$

With $q = 1$, Equation 4.4 has no closed-form solution but can be solved numerically, for example, by using the generalized Lasso framework.[10] With $q = 2$, minimizing Equation 4.4 is a Tikhonov regularization problem[11] and admits the closed form solution below:

$$\underset{\boldsymbol{\beta}}{\text{argmin}} \, \mathcal{L}^{STEW}(\boldsymbol{\beta}, \lambda, 2) = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{D}^T\boldsymbol{D})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \qquad (4.5)$$

[10] Tibshirani and Taylor (2011)

[11] Tikhonov et al. (2013)

We use $q = 2$ in the remainder of this chapter due to the computational advantages of its closed-form solution. The matrix $\boldsymbol{D}$ enters the computation only via the scalar product $\boldsymbol{D}^T\boldsymbol{D}$, which has dimensions $p \times p$ and whose elements can be described as follows:

$$(\boldsymbol{D}^T\boldsymbol{D})_{ij} = \begin{cases} p - 1, & \text{for } i = j \\ -1, & \text{for } i \neq j. \end{cases}$$

This matrix can be calculated once, before conducting the search for the best value of $\lambda$ because it depends only on the number of features $p$.

The regularization strength $\lambda$ is treated as a hyper-parameter that is chosen before the weights $\boldsymbol{\beta}$ are estimated.

## 4.3 RELATED WORK

Here we discuss existing work that is related to STEW in that it is also motivated by knowledge about feature directions, also shrinks some weights toward each other, or also tries to integrate models of bounded rationality into a statistical machine learning framework.

▸ NON-NEGATIVITY CONSTRAINTS. Similar to STEW, non-negative least squares (NNLS) and non-negative Lasso (NNLasso) benefit from positive (or directable) features. NNLS minimizes the residual sum of squares while constraining weights to be positive. Although positivity constraints alone were found to have regularizing properties,[12] NNLS has been combined with $l_1$-penalty as well.[13] The resulting model is NNLasso and minimizes the loss function

$$\mathcal{L}^{NNLasso}(\boldsymbol{\beta}, \lambda) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \|\boldsymbol{\beta}\|_1, \text{ such that } \beta_i \geq 0, \forall i = 1, \dots, p.$$

Figure 4.2 shows regularization paths for the NNLasso model. Note how for NNLasso all weights are always positive or zero, whereas weights in a STEW-regularized linear model can be negative for weak regularization strengths. We compare STEW to NNLasso in the empirical analysis in Section 4.5.
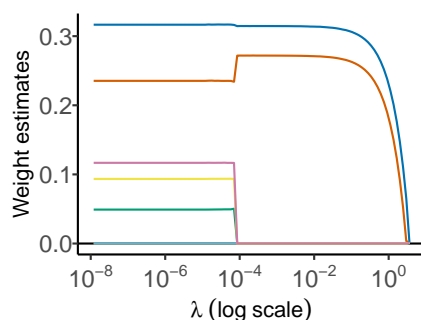
Figure 4.2: Weight estimates, as a function of the regularization strength $\lambda$, for NNLasso on the *Rent* data set with seven standardized features. Compare to figure 4.1.

▸ TOTAL-VARIATION MODELS. Total variation (TV) models[14] are motivated by environments in which features are spatially or temporally correlated, such as the pixels of an image or the measurements of a time series. TV models estimate smooth functions by penalizing the difference between the weights of adjacent features. In a one-dimensional setting, TV models minimize the loss function

$$\mathcal{L}^{TV}(\boldsymbol{\beta}, \lambda) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \lambda \sum_{j=2}^{p} |\beta_j - \beta_{j-1}|.$$

TV models have been developed for and used with data sets where a natural adjacency relationship exists. TV models are also used in biostatistics when the data allows a meaningful order of features, for example, in protein mass spectroscopy. The fused Lasso[15] considers a Lasso-type $l_1$-penalty in addition to a TV-type smoothness penalty.

There is a surface similarity between STEW and TV models: both models penalize differences between weights. But the exact form of the penalty differs between the models. TV models penalize the differences between adjacent weights while STEW penalizes all pairwise differences between the weights. This difference is a direct consequence of the different motivations behind the two models and it results in meaningful differences in regularization behavior. Specifically, TV models shrink weights together in patches or clusters that are

defined by the adjacency relationships (sample regularization paths are shown in Figure B.4 in the Appendix), which is quite different to the behavior of STEW (Figure 4.1). It should be noted that imposing an arbitrary adjacency relationship onto a dataset (to be used with a TV model) is not well justified: different adjacency relationships result in arbitrarily different solutions along the regularization path. Figure B.4 in the Appendix also presents a comparison of regularization paths taken by TV models with different orderings of features of the *Rent* data set.

▶ BAYESIAN LINEAR MODELS. Some regularized linear models are equivalent to certain Bayesian linear models.[16] This leads to the question of whether models of bounded rationality could also be formulated as Bayesian linear models.

    Parpart et al. (2018) pursued this question in the context of paired comparison. Specifically, they formulate simple decision heuristics for the paired comparison problem as Bayesian linear models under infinitely strong priors. They define and analyze two Bayesian linear models, called "half-ridge" and "half-Lasso", that correspond to using half-Gaussian and half-Laplacian prior distributions, respectively. The "half-"prefix indicates that weights are truncated at zero, that is, all weights are positive if features are directed. They find that both these models converge to the tallying heuristic for a prior-variance approaching 0. The tallying heuristic is the paired-comparison equivalent of the equal-weights regression model. Note that this convergence result relies on the fact that the weights in linear paired comparison models are scale-invariant.[17] By consequence, the results do not hold in the context of regression, where scale-invariance does not hold.

    Moreover, Parpart et al. (2018) define a Bayesian two-step paired comparison model called *covariance orthogonalizing regularization* (COR) that converges to the tallying heuristic or the take-the-best heuristic, depending on the decision rule used. Similar to the STEW regularizer defined in this chapter, the COR model allows the interpolation between one of the simple models and a fully-data-driven solution by varying the prior variance (similar to how we vary regularization strength). Unlike the regularization term defined in this chapter, the COR model is restricted to paired comparison and does not easily generalize to regression.

## 4.4    BIAS-VARIANCE ANALYSIS OF EQUAL-WEIGHTING MODELS

Regularized linear models search for a happy medium between OLS, which has low bias but high variance, and a model that has high bias but low variance. For both STEW and models that shrink toward zero, the low-variance model is an equal-weighting model: STEW regularizes toward the EW model ($\gamma = \gamma_{EW}$) while models that shrink toward zero regularize toward what we call the **0**-*model* ($\gamma = 0$).

    Theorem 1 shows results on the bias-variance decomposition of mean squared

[16] For example, ridge regression corresponds to a zero-mean Normal prior on the weights (Hoerl and Kennard, 1970); the Lasso corresponds to a zero-mean Laplace prior (Tibshirani, 1996). The variance of the prior is inversely related to the regularization strength $\lambda$ in both cases.

[17] That is, a linear paired comparison model with feature weights $\beta$ makes the same predictions as a model with feature weights $c\beta$ for any $c > 0$.

error for equal-weighting models, providing intuition on when and why EW—and therefore STEW—can perform well.

Mean squared error $MSE(\hat{\boldsymbol{\beta}}) = \mathbb{E}\|\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}\|^2$ can be decomposed into two components, squared bias and the trace of the variance-covariance matrix, $\boldsymbol{\Sigma}_{\hat{\boldsymbol{\beta}}}$, as follows:

$$MSE(\hat{\boldsymbol{\beta}}) = bias^2 + variance = \|\mathbb{E}[\hat{\boldsymbol{\beta}}] - \boldsymbol{\beta}\|^2 + tr(\boldsymbol{\Sigma}_{\hat{\boldsymbol{\beta}}}).$$

Let $\hat{\boldsymbol{\beta}}_{EW}$ and $\hat{\boldsymbol{\beta}}_0$ denote the weight estimates of the EW model and the **0**-model, respectively. Their differences in squared bias and mean squared error are defined as

$$\Delta bias^2 := bias^2(\hat{\boldsymbol{\beta}}_0) - bias^2(\hat{\boldsymbol{\beta}}_{EW})$$

and

$$\Delta MSE := MSE(\hat{\boldsymbol{\beta}}_0) - MSE(\hat{\boldsymbol{\beta}}_{EW}),$$

respectively.

---

**Theorem 1.** Let $\boldsymbol{y} \sim (\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 \boldsymbol{I}_{n \times n})$, where $\|\boldsymbol{\beta}\|^2 < \infty$, $\sigma^2 > 0$, and $\boldsymbol{I}_{n \times n}$ is the identity matrix of size $n$. Let $\bar{\boldsymbol{\beta}} := \frac{1}{p} \sum_{i=1}^{p} \beta_i$ denote the mean of the true weights. Then,
(1) The minimum-bias equal-weighting estimator of $\gamma$ is $\bar{\boldsymbol{\beta}}$.
(2) For orthonormal data matrix $\boldsymbol{X}$ (i.e., $\boldsymbol{X}^T \boldsymbol{X} = \boldsymbol{I}_{p \times p}$),
   (a) EW is the minimum-bias equal-weighting estimator,
   (b) $\Delta bias^2 = p\bar{\boldsymbol{\beta}}^2$,
   (c) $\Delta MSE = p\bar{\boldsymbol{\beta}}^2 - p\sigma^2$,
   (d) The squared mean weight $\bar{\boldsymbol{\beta}}^2$, and thus $\Delta bias^2$ and
       $\Delta MSE$, is higher on a directed set of weights than
       on an undirected set of weights.

---

*Proof of Theorem 1.* **(1)** The bias of an equal-weighting estimator can be written as follows:

$$\|\mathbb{E}[\hat{\boldsymbol{\beta}} - \boldsymbol{\beta}]\|_2^2 = \|\mathbb{E}[\gamma\mathbf{1} - \boldsymbol{\beta}]\|_2^2 \tag{4.6}$$

$$= \sum_{i=1}^{p}(\gamma - \beta_i)^2 \tag{4.7}$$

$$= \sum_{i=1}^{p}(\gamma^2 - 2\gamma\beta_i + \beta_i^2). \tag{4.8}$$

The derivative of the bias with respect to $\gamma$ is then given by the following:

$$\frac{\partial}{\partial\gamma} \sum_{i=1}^{p}(\gamma^2 - 2\gamma\beta_i + \beta_i^2) = \sum_{i=1}^{p}(2\gamma - 2\beta_i), \tag{4.9}$$

Equating this derivative to 0 yields the following:

$$\sum_{i=1}^{p}(2\gamma - 2\beta_i) = 0$$

$$2\gamma p = 2\sum_{i=1}^{p}\beta_i$$

$$\gamma = \frac{\sum_{i=1}^{p}\beta_i}{p} = \bar{\beta}.$$

**(2a)** From here on, we assume that $X$ is orthonormal, that is, $X^TX = I_{p\times p}$. Recall that $\gamma_{EW}$ is calculated using simple (univariate) linear regression on the model

$$\hat{y} = \gamma X\mathbf{1}, \tag{4.10}$$

where $\gamma$ is the equal-weighting parameter to be estimated and $\mathbf{1}$ is a column-vector of ones of length $p$. Defining $c := X\mathbf{1}$, the simple linear regression estimate of Equation 4.10 is given by

$$\gamma_{EW} = \frac{r_{yc}}{s_c}, \tag{4.11}$$

where $r_{yc}$ is the sample correlation coefficient between $y$ and $c$, and $s_c$ is the standard deviation of $c$. For $y = X\beta + \varepsilon$ with $\mathbb{E}[\varepsilon] = \mathbf{0}$, the expected value of EW's $\gamma_{EW}$ is given by

$$\begin{aligned}
\mathbb{E}[\gamma_{EW}] &= \mathbb{E}[\frac{r_{yc}}{s_c}]\\
&= \mathbb{E}[\frac{\mathbf{1}^TX^Ty}{\mathbf{1}^TX^TX\mathbf{1}}]\\
&= \mathbb{E}[\frac{\mathbf{1}^TX^T(X\beta + \varepsilon)}{\mathbf{1}^T\mathbf{1}}]\\
&= \mathbb{E}[\frac{\mathbf{1}^TX^TX\beta}{\mathbf{1}^T\mathbf{1}}] + \mathbb{E}[\frac{\varepsilon}{\mathbf{1}^T\mathbf{1}}]\\
&= \frac{\sum_{i=1}^{p}\beta_i}{p} + 0\\
&= \bar{\beta}.
\end{aligned}$$

**(2b)** We compute the squared biases for EW and the **0**-model. Let $\hat{\beta}_{EW} = \gamma_{EW}\mathbf{1}$ and $\hat{\beta}_0 = \mathbf{0}$ the weight estimates for EW and the **0**-model,

respectively. Then, the squared biases are given by

$$
\begin{aligned}
\text{bias}^2(\hat{\boldsymbol{\beta}}_{EW}) &= \|\mathbb{E}[\hat{\boldsymbol{\beta}}_{EW}] - \boldsymbol{\beta}\|^2 \\
&= (\bar{\beta}\mathbf{1} - \boldsymbol{\beta})^T(\bar{\beta}\mathbf{1} - \boldsymbol{\beta}) \\
&= p\left(\bar{\beta}\right)^2 - 2\sum_{i=1}^{p}\bar{\beta}\beta_i + \boldsymbol{\beta}^T\boldsymbol{\beta} \\
&= \boldsymbol{\beta}^T\boldsymbol{\beta} - p\left(\bar{\beta}\right)^2 \\
&= \|\boldsymbol{\beta}\|^2 - p\left(\bar{\beta}\right)^2
\end{aligned}
$$

$$
\begin{aligned}
\text{bias}^2(\hat{\boldsymbol{\beta}}_0) &= \|\mathbf{0} - \boldsymbol{\beta}\|^2 \\
&= \|\boldsymbol{\beta}\|^2.
\end{aligned}
$$

The difference in biases, $\Delta\text{bias}^2$, can be computed directly as follows:

$$
\begin{aligned}
\Delta\text{bias}^2 &= \text{bias}^2(\hat{\boldsymbol{\beta}}_0) - \text{bias}^2(\hat{\boldsymbol{\beta}}_{EW}) \\
&= \|\boldsymbol{\beta}\|^2 - (\|\boldsymbol{\beta}\|^2 - p\left(\bar{\beta}\right)^2) \\
&= p\bar{\beta}^2.
\end{aligned}
$$

**(2c)** The variance for the **0**-model is clearly 0. For the EW model, we use Equation 4.11 to note that the EW estimate $\hat{\boldsymbol{\beta}}_{EW} = \mathbf{1}(\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{X}\mathbf{1})^{-1}\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{y}$ is a linear function of $\boldsymbol{y}$. The trace of its variance $tr(Var(\hat{\boldsymbol{\beta}}_{EW}))$ can thus be written as follows:

$$
\begin{aligned}
&tr(\mathbf{1}(\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{X}\mathbf{1})^{-1}\mathbf{1}^T\boldsymbol{X}^T Var(\boldsymbol{y})\boldsymbol{X}\mathbf{1}(\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{X}\mathbf{1}^T)^{-1}\mathbf{1}^T) \\
&= \sigma^2 tr(\mathbf{1}(\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{X}\mathbf{1})^{-1}\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{I}\boldsymbol{X}\mathbf{1}(\mathbf{1}^T\boldsymbol{X}^T\boldsymbol{X}\mathbf{1}^T)^{-1}\mathbf{1}^T) \\
&= \sigma^2 p.
\end{aligned}
$$

The result follows directly by adding these variances to the squared biases from Result 2b.

**(2d)** It remains to show that the squared average weight $\left(\bar{\beta}\right)^2$ is larger on a set of directed predictors than on an undirected set of predictors. Let $\boldsymbol{\beta}_\| = (|\beta_1|, \ldots, |\beta_p|)$ denote the weights on a positively directed data set. Then the following holds:

$$
\begin{aligned}
\left(\overline{\boldsymbol{\beta}_\|}\right)^2 &\geq \left(\bar{\beta}\right)^2 \\
|\overline{\boldsymbol{\beta}_\|}| &\geq |\bar{\beta}| \\
\overline{\boldsymbol{\beta}_\|} &\geq |\bar{\beta}|.
\end{aligned}
$$

The last line follows directly from Jensen's inequality for the convex function $f(x) = |x|$. $\qquad\square$

Result 1 shows that minimum bias is achieved by setting the equal-weighting constant $(\gamma)$ to the mean of the true weights $(\bar{\beta}^2)$. Result 2a shows that, in the

special case of an orthonormal data matrix, $\gamma_{EW}$ is an unbiased estimate of the mean of the true weights and thus attains minimum bias. Result 2b shows that the difference in bias between the **0**-model and EW increases with the square of the mean of true weights, in other words, with increasing distance of the true mean of weights from zero. It follows that EW has lower mean squared error than the **0**-model if the decrease in bias is not canceled out by the increase in variance that results from estimating $\gamma_{EW}$. In the case of an orthonormal data matrix, this variance simply equals the product of the noise parameter $\sigma^2$ and the number of features $p$ (Result 2c).

Result 2d examines the impact of knowing feature directions. When feature directions are known, features can be recoded to have the same direction, for example, by multiplying the values of all negative features by $-1$. This simple operation does not change the biases of the **0**-model, OLS, ridge regression, and the Lasso. It does, however, reduce the bias of the EW model.

## 4.5    EMPIRICAL ANALYSIS

Next we present simulation experiments that examine to which extent the results of Theorem 1 transfer from EW to STEW in a diverse set of simulated linear environments. Further below we then assess the predictive performance of STEW in a set of single-shot real-world regression environments under varying assumptions about the directability of features.

More specifically, we compare the predictive accuracy of a STEW-regularized linear regression model, a Lasso-regularized regression model, ridge regression, the non-negative Lasso model (NNLasso), and the EW model. We also tested *non-negative least squares*, which is NNLasso without the lasso regularization but omit it from the plots because its performance always lagged behind NNLasso.

▸ IMPLEMENTATION DETAILS. The regularization strength $\lambda$ of all regularized linear models was chosen using $k$-fold cross validation. The cross-validation parameter $k$ was set to $\min(10, n)$, where $n$ is the training set size. Model-specific implementation details are provided in Appendix A.

### 4.5.1    *Simulated Environments*

We sampled data from the true model $Y = X_1\beta_1 + \cdots + X_{20}\beta_{20} + \varepsilon$, where $X_i \overset{i.i.d}{\sim} \mathcal{N}(0,1)$ and $\varepsilon \overset{i.i.d}{\sim} \mathcal{N}(0,1)$. The defining property of each environment was the prior distribution from which the weights $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_{20})$ were sampled. For each environment, 400 data sets were sampled to compare the predictive accuracy of STEW to that of the baseline models described above. In all environments, when training sets were large enough, STEW, ridge regression, and the Lasso performed equally well, with MSE converging to irreducible error. Our discussion will thus focus on small-to-medium sample sizes.
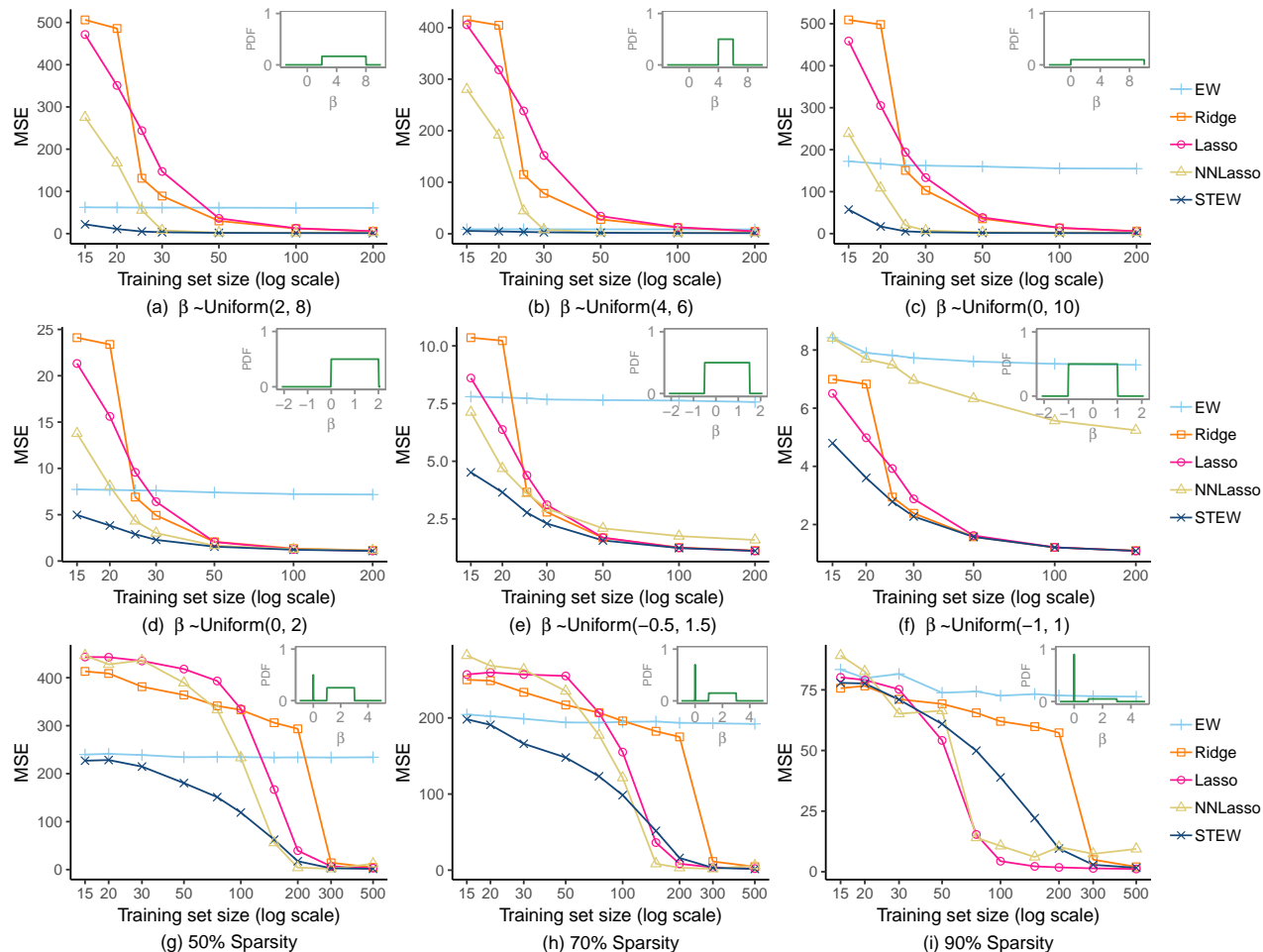
Figure 4.3: Prediction error in environments defined by uniform weight priors with the same mean but different variance (a–c), with shifting support (d–f), and varying degrees of sparsity (g–i). Probability density functions of the weight priors are shown in green in the top-right corner of each panel.

▶ DIRECTABLE ENVIRONMENTS. We first analyze the ideal use case for STEW: when weights are known to be positive (equivalently, if features are directable). Recall that, in such an environment, STEW, EW, and NNLasso are able to directly use the knowledge that the weights are positive. On the other hand, ridge regression and the Lasso cannot incorporate this information directly; they learn it from the data.

Figure 4.3a shows the predictive performance of various models when $\beta \sim \mathcal{U}(2, 8)$. STEW performed best overall. EW performed relatively well when training sets were small—although it was outperformed (as expected) by all adaptively regularizing models for large sample sizes. STEW was able to combine the strengths of different models. For small sample sizes, STEW regularized toward the EW solution and outperformed all competing models, including EW. For large sample sizes, STEW performed as well as the other adaptively regularizing linear models. Notice that, for small sample sizes, NNLasso was far behind STEW, even though it also directly used the knowledge that the weights are positive.

One possible explanation for the superior performance of STEW compared to NNLasso is that the prior distribution of the weights has relatively low variance. When variance is low, weights are relatively close to each other, creating an environment that supports EW, and therefore STEW. We therefore examine two additional environments, $\boldsymbol{\beta} \sim \mathcal{U}(4,6)$ and $\boldsymbol{\beta} \sim \mathcal{U}(0,10)$, that are identical to $\boldsymbol{\beta} \sim \mathcal{U}(2,8)$ in the shape of the distribution and their expected values but differ in their variance. The results are shown in panels b and c of Figure 4.3. STEW remained the best performing model in all three environments but its relative advantage compared to the next best model, NNLasso, decreased with increasing variance. In additional experiments, we increased the variance to unrealistically high levels, up to $\boldsymbol{\beta} \sim \mathcal{U}(0,50)$. The results, provided in Figure B.5 in the Appendix, remained qualitatively similar.

▸ EFFECT OF DIRECTABILITY. Weight priors used in panels d–f of Figure 4.3 follow a uniform distribution as before. They all have a support of length 2 but differ in the region of support. From panel d to f, the environments decrease in the proportion of weights that are positive. In the $\boldsymbol{\beta} \sim \mathcal{U}(0,2)$ environment, all weights are positive. This prior therefore represents a fully-directable environment. The slightly shifted $\boldsymbol{\beta} \sim \mathcal{U}(-0.5,1.5)$ environment can be interpreted as a situation in which the user can direct some but not all the weights. Finally, the $\boldsymbol{\beta} \sim \mathcal{U}(-1,1)$ environment is symmetric around 0; weights cannot be directed. With decreasing directability of weights, the performance of STEW, EW, and NNLasso decreased relative to the performance of models which do not use information about the direction of features. Yet STEW remained the best performing model even in an undirectable environment. In contrast, NNLasso performed considerably worse than ridge regression and the Lasso.

▸ HIGH-DIMENSIONAL ENVIRONMENTS WITH SPARSITY. On learning curves presented so far, the early parts of the curves correspond to situations in which the number of features ($p$) was moderately higher than the number of observations ($n$). But $p$ was of the same order of magnitude as $n$. For the following set of experiments, we increased the number of features to $p = 200$. In addition, we introduced sparsity by setting some proportion of weights to exactly zero. Weights were sampled from $\mathcal{U}(1,3)$ and subsequently, conditional on the outcome of a coin flip, set to zero. This coin flip had success probability $\mathbb{P}[\beta = 0] = \omega$, where $\omega$ is the expected degree of sparsity in the environment. For example, if $\omega = 0.7$, on average, 70% of the weights have a value of zero while 30% of the weights follow a $\mathcal{U}(1,3)$ distribution. Panels g to i of Figure 4.3 show the results. With 50% sparsity, STEW outperformed all other models on large parts of the learning curve, especially when $n \ll p$. With increasing sparsity, Lasso-type models increasingly benefited from their variable selection property. With 90% sparsity, Lasso-type models outperformed STEW across large parts of the learning curve.

▶ EMPIRICAL BIAS-VARIANCE ANALYSIS. Weights in the environment of Figure 4.4 follow a Gaussian distribution with zero mean and unit variance, $\boldsymbol{\beta} \sim \mathcal{N}(0, 1)$. A Gaussian prior represents the ideal environment for ridge regression from a Bayesian perspective.[18] Surprisingly, STEW outperformed all other models including ridge regression across the entire learning curve. The figure also shows the empirical bias-variance decomposition of mean squared error, revealing the different approaches ridge regression and STEW take towards regularization. For small sample sizes, ridge regression reduced variance to almost zero, with error consisting almost entirely of bias. On the other hand, STEW was able to substantially lower bias by allowing some variance.

[18] Hoerl and Kennard (1970)
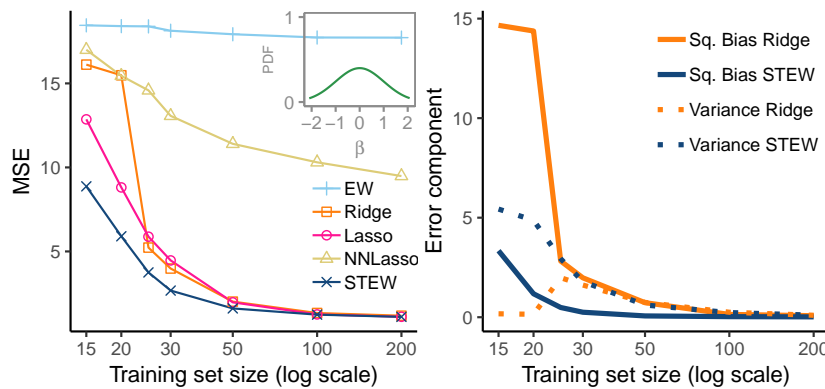


Figure 4.4: Prediction error (a) and empirical bias-variance decomposition (b) in a Gaussian environment.

### 4.5.2  *Real-World Environments*

Learning curves on real-world data sets (Figure 4) were computed as follows. We pre-processed each real-world data set by standardizing responses and predictors to have zero mean and unit variance. Missing predictor values were mean-imputed and observations with missing response values were removed from the data set. We set aside a random subset of 10% of the observations as test set. We then progressively sampled training sets of increasing size using the remaining observations. Results were averaged across 200 repetitions, each corresponding to a different train/test-split of the data.

We compared the prediction performance of STEW, EW, elastic net, and NNLasso on 13 real-world data sets under different conditions regarding how directable features are. Appendix C contains detailed descriptions of each data set.

We first consider the *Rent* data set (described in Section 2.1.1), where the problem is to estimate the response *rent per $m^2$* for 2053 apartments based on 10 features. In the first stage of our analysis, we directed features based on our intuition. For example, the features *the apartment has warm water* (yes = 1, no = 0) and the *year of construction* (in years) were both expected to be positively associated with the response. Figure 4.5a shows that both EW and STEW clearly outperformed competing models across the entire learning curve on the

(a) Rent
(intuitively directed)

(b) Diabetes
(Lasso–directed)

(c) Mean across 13 data sets
(Lasso–directed)

(d) Rent
(directed on training set)

(e) Diabetes
(directed on training set)

(f) Mean across 13 data sets
(directed on training set)
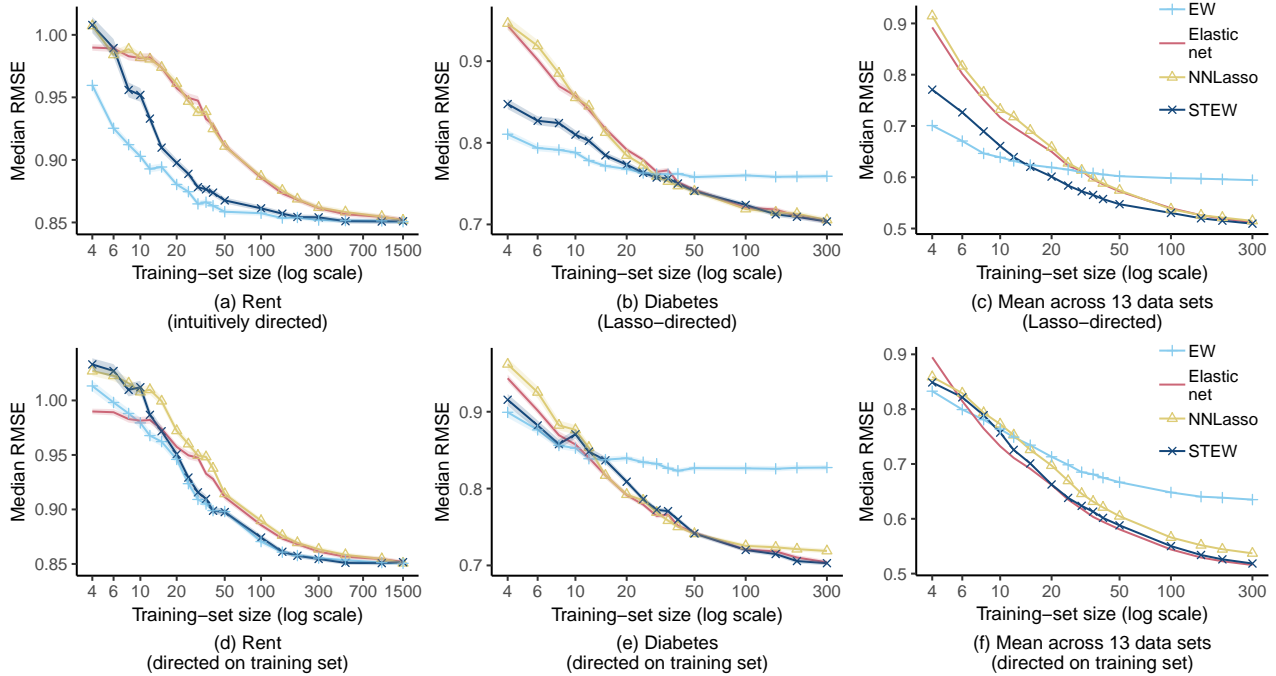
Figure 4.5: Median root mean squared error (RMSE) with standard-error band across 200 repetitions. The first column shows results on the *Rent* data set, the second column on the *Diabetes* data set, and the third column on all 13 data sets. Features were directed based on the intuition of the author (a), based on a Lasso estimate on the whole data set (b, c), or based on the training set (d–f).

intuitively directed *Rent* data set, with EW performing even better than STEW.

Intuitively guessing feature directions is not always easy. In the *Diabetes* data set, in which a quantitative measure of disease progression of 442 diabetes patients needs to be predicted based on *age, sex, body mass index, average blood pressure*, and six blood serum measurements, we could not intuitively guess the directions of most features. However, a physician probably could.

We simulated this type of expert knowledge as follows. We estimated a Lasso-regularized model on the entire data set and chose the regularization strength that resulted in the lowest cross-validated prediction error. We discarded all features whose Lasso weight was zero and positively directed the remaining features, that is, we multiplied all features with −1 whose Lasso weight was negative.

Figure 4.5b shows learning curves for *Diabetes* obtained in this way. EW performed best until a training set size of 25 but fell behind for training set sizes larger than 40. STEW could not match EW's performance on small training sample sizes. However, it clearly outperformed the elastic net and NNLasso on small training set sizes and performed equally well with larger training set sizes.

Average learning curves across all 13 data sets are shown in Figure 4.5c. EW performed best on very small training sets, STEW on small to medium training sets, and all adaptively-regularized linear models performed equally well on large training set sizes. Individual learning curves on the other data sets (available in Figure B.6 in the Appendix) show that STEW outperformed both the elastic net and NNLasso on 5 out of 13 data sets, while showing comparable performance in the remaining 8 data sets.

Even when no information about feature directions is available, directions can still be estimated from the training set, for example, from Pearson correlation coefficients between the features and the response. Panels d to f of Figure 4.5 show learning curves when directions were estimated in this way. Averaged across many data sets, STEW did not outperform the competing models but it was robust in the sense that it did not perform worse than the elastic net. Individual learning curves are available in Figure B.7 in the Appendix.

## 4.6 DISCUSSION

One may reasonably assume that STEW would perform well only in environments in which the true feature weights are almost equal. This is clearly not the case. STEW has proven to be useful in a wide range of synthetic and real-world environments where any assumption of equal weights is clearly violated.

To understand how STEW can outperform models that shrink toward zero, it has been instructive to contrast the two models that are obtained in the limit of infinite regularization: the equal-weights (EW) model and the **0**-model. Our theoretical results show that EW has lower bias than the **0**-model and that this difference increases with increasing directability of features. On data sets that require strong regularization (for example, small data sets), STEW inherits this relatively lower bias.

Sign-constrained models such as NNLS or NNLasso also utilize information on feature directions but generally did not perform as well as STEW in fully directable environments. Furthermore, when directions were not available, or were unreliable, these models failed to produce useful estimates whereas STEW performed on par with other regularized linear models.

STEW showed surprisingly high prediction accuracy across a variety of $p > n$ environments. However, unlike Lasso-type models, STEW has no built-in variable-selection mechanism. It is thus clearly not meant to be a model for *sparse recovery*, that is, STEW is not expected to identify the non-zero weights in a sparse environment. It could, however, potentially be developed further to include a sparsity component or used in conjunction with existing methods for variable selection. One possibility is a two-stage model, similar to *Lasso + OLS*.[19] The first stage of this model consists of fitting a Lasso model on the entire training data and subsequently discarding all features whose Lasso-estimates are zero. The final estimate is then obtained by fitting the second-stage model on the reduced set of features. STEW could prove useful as a second-stage model because the initial Lasso estimate not only takes care of discarding irrelevant features but also provides information about feature directions. STEW and Lasso-type models exploit different types of priors (or information) about the environment. Developing models that can exploit both types of information is a fruitful direction for future research.

In this chapter we studied the STEW regularization term in the regression task only. This seems to be usual in the regularization literature.[20]. I believe

[19] Efron et al. (2004) and Belloni and Chernozhukov (2013)

[20] For example, initial analysis of Lasso / $l_1$-regularization and ridge regression / $l_1$-regularization in machine learning was conducted in regression settings (Hoerl and Kennard, 1970; Tibshirani, 1996).

that the reason for this focus on regression is supported by the fact that mean-squared regression error can be easily decomposed into bias and variance components (which facilitates theoretical analysis), whereas similar decompositions are less intuitive for other loss functions.[21] Whatever the reason may be, the STEW regularization term can be easily used for linear models in other prediction tasks as well. In fact, a STEW-regularized multinomial logistic regression model will play a central role in the next chapter.

[21] Friedman (1997) and Domingos (2000)

# BOUNDEDLY RATIONAL WHEN IT MATTERS MOST: ITERATIVE POLICY SPACE EXPANSION IN REINFORCEMENT LEARNING.

This chapter is based on the articles *"Iterative policy-space expansion in reinforcement learning"* (Lichtenberg and Şimşek, 2019a) and *"Regularization in directable environments with application to Tetris"* (Lichtenberg and Şimşek, 2019b).

Reinforcement learning with function approximation can be interpreted as a series of supervised learning tasks with the particularity that the training data (pairs of states and their corresponding optimal actions) in a given iteration is created by the agent itself, using the policy learned in the previous iteration. The general idea behind these algorithms is that a better policy approximation allows the agent to create a better training data set for the next iteration, which in turn allows the agent to learn an even better policy, and so on.

In a typical reinforcement learning application, however, the agent does not know anything about its environment at the beginning of the learning process. The initial policy therefore is usually of low quality.[1] A low-quality policy leads to a noisy data set, that is, only few states are mapped to optimal actions. A noisy training data set in turn makes it difficult for the agent to learn a better policy in the next iteration.

Put differently, a good policy is not only the output of the learning process; it is a significant part of the learning process itself. Furthermore, this learning process is of self-reinforcing nature: a good policy early on facilitates finding an even better policy quickly, whereas a bad policy early on further slows down the learning process.

Existing classification-based reinforcement learning algorithms largely ignore this observation. Driven by the prospect of learning an expressive, near-optimal policy at the end of the learning process, these algorithms use complex policy approximation architectures, such as unconstrained linear functions or deep neural networks, throughout the entire learning process. The problem is that these complex models tend to overfit the noisy data typically present at the beginning of the learning process, resulting in weak policies early on, and thus ultimately in long and tedious learning processes.

The goal in this chapter is to accelerate the learning process by means of adapting the difficulty of the policy learning task to the amount and quality of data available to the agent. In particular, the policy learning task should be

[1] Typically the initial policy is a uniformly random policy or a deterministic policy approximator whose parameters are initialized randomly (and which therefore, on average, is not better than a uniformly random policy).

simple in the beginning of the learning process and only become more difficult as quality and amount of data increases.

We propose a specific instantiation of this general idea in the context of learning a linear policy using rollout-based reinforcement learning. The proposed algorithm is centered around the equal-weighting model and the shrinking toward equal weights (STEW) regularization term studied in the previous two chapters.

More specifically, the algorithm starts with the simple task of dividing the available features into two groups: features that correlate positively with good decision outcomes, called *positive* features; and features that correlate negatively with good decision outcomes, called *negative* features. "Negative" here does not mean that a feature is unimportant or irrelevant. The word exclusively refers to the sign of the feature's weight, also called the feature's *direction*.[2] In this early phase of the algorithm, the algorithm attributes equal importance to both positive and negative features. The agent thus effectively uses an equal-weighting policy, which serves as a catalyst for the learning process.

Once all feature directions have been learned, the algorithm uses the STEW regularization term to gradually deviate from the equal-weighting policy and to learn about the relative importances of features. Put differently, the agent first learns the signs of the weights (a simple task) before learning their magnitudes (a much more difficult task).

The idea that a sequence of progressively more difficult tasks could accelerate learning has been exploited in animal training where it is called *shaping*.[3] Previous research has raised the question of whether learning machines could benefit from similar ideas. In robotics, learned dynamics from regions of easy solvability are reused in more difficult regions of the task environment.[4] In *curriculum learning*,[5] neural networks are trained with progressively more noisy and less relevant training data. However, finding a good curriculum is a difficult problem and solutions are often task-specific.[6]

The algorithm proposed in this chapter does *not* require an external teacher who guides the learning agent with a carefully tailored curriculum of tasks with increasing difficulty. The task difficulty is instead regulated intrinsically along the following two dimensions. First, the agent initially learns weights naïvely (as in *naïve Bayes*), that is, without considering interdependencies among features. Eventually, weights are estimated jointly. Second, the agent learns in a decreasingly constrained policy space. We therefore call the algorithm *iterative policy space expansion*, or IPSE.

The main contribution of this chapter is both conceptual and algorithmic. We find that reinforcement learning can be accelerated by using a form of intrinsically regulated curriculum learning, in which the task difficulty is regulated by the capacity of the policy approximation architecture. In particular, the IPSE algorithm relies on the boundedly rational equal-weighting strategy in the beginning of the learning process, when resources are most limited. When applied to the problem of learning how to play the game of Tetris, the IPSE algorithm

[2] See also Section 2.2.1.

[3] Skinner (1958), Peterson (2004), Krueger and Dayan (2009), and Bengio et al. (2009)

[4] Sanger (1994)

[5] Elman (1993) and Bengio et al. (2009)

[6] But see, for example, Graves et al. (2017).

learns considerably faster than approaches that do not take into account that amount and quality of the resources available to the agent change over time.

This chapter produces two further algorithmic contributions. The *M-learning* algorithm is a rollout-based reinforcement learning algorithm that interprets the policy improvement step as a discrete choice problem. We use the M-learning algorithm as a workhorse for the IPSE algorithm, but M-learning is a novel algorithm by itself. It provides a useful alternative to existing classification-based reinforcement learning algorithms in environments for which policies are more naturally described by a discrete choice problem than by a classification problem.

The *learning feature directions* (LFD) algorithm provides a way of estimating feature directions in Markov decision processes. Feature directions have been identified as building blocks not only for equal-weighting strategies but also for other models of bounded rationality such as the *take-the-best* heuristic (see Section 2.2.2). Therefore, we believe that the LFD algorithm could prove useful as a building block in other reinforcement learning algorithms as well.

The remainder of this chapter is organized as follows. Section 5.1 defines the setting in which the remainder of this chapter operates. Section 5.2 defines the M-learning algorithm and Section 5.3 defines the LFD algorithm. These two algorithms are then combined to define the IPSE algorithm in Section 5.4. Section 5.5 compares the IPSE algorithm to related algorithms in the literature. Section 5.6.1 provides background on the Tetris domain and defines it as a Markov decision process. The remainder of Section 5.6 then presents the results of our Tetris experiments. Finally, 5.7 concludes this chapter and discusses possible future work.

## 5.1 BACKGROUND & OVERVIEW

This chapter is concerned with learning in Markov decision processes, where the agent informs its decision in a state $s$ by observing action features $\boldsymbol{\phi}(s, a) \in \mathbb{R}^p$ for each available action $a$. In particular, we are interested in learning linear policies of the form

$$\pi(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \; \boldsymbol{\beta}^T \boldsymbol{\phi}(s, a), \tag{5.1}$$

where $\boldsymbol{\beta} \in \Pi \subseteq \mathbb{R}^p$ denotes the vector of feature weights to be estimated and $\Pi$ denotes the *policy space*. The policy can be interpreted as a discrete choice problem:[7] choose one out of $k = |\mathcal{A}(s)|$ available actions.

[7] See Section 2.1.2 for a detailed description of the discrete choice problem and its differences to the classification problem.

In the following sections, we will define three reinforcement learning algorithms: learning feature directions (LFD), M-learning, and iterative policy space expansion (IPSE). These three algorithms are similar in that they all are rollout-based algorithms, which are discussed in detail in Section 2.3.2 in the introductory part of this dissertation. In particular, all three algorithms use the same rollout mechanism (Algorithm 3) to create a new data set from the current policy. The

three algorithms differ in how a new set of weights $\beta$ is learned from a given data set and, in particular, the policy space $\Pi$ they operate on.

We first define M-learning and the LFD algorithm in Sections 5.2 and 5.3, respectively. The IPSE algorithm is then obtained by sequentially applying the LFD algorithm and a regularized version of the M-learning algorithm, as explained in Section 5.4.

## 5.2 M-LEARNING

Here we present a novel rollout-based reinforcement learning algorithm to learn a linear policy, called *M-learning*. Similar to classification-based reinforcement learning algorithms,[8] M-learning learns a new policy by training a supervised learning algorithm on a data set of decisions made using rollouts. M-learning is different from classification-based algorithms in that it interprets policy learning as a discrete choice problem, rather than a classification problem.[9] Specifically, it uses multinomial logistic regression to train a new policy on choice set data.

Because discrete choice requires observed choices (or choice sets) as training data, the data-collecting process is slightly different to that of classification-based reinforcement learning algorithms. In what follows, we first describe how M-learning constructs a training data set of observed choices before we describe how (regularized) multinomial logistic regression can be used to learn a new policy. The pseudo-code for M-learning is provided in Algorithm 5.

▸ DATA-SET CONSTRUCTION. The agent starts with an empty training data set $\mathcal{D} = \varnothing$ to which one training sample is added after each interaction with the environment.[10] A single training sample (that is, one choice set) is generated as follows.

1. For every available action in the current state, use the ROLLOUT procedure (given in Algorithm 3) to generate a rollout estimate of the action's utility.

2. Let $\tilde{a}$ denote the action that returned the highest mean estimated rollout value. The agent executes this action and observes the next state of the environment.

3. Add the sample, $(\tilde{a}, \phi(s, a_1), ..., \phi(s, a_{|\mathcal{A}(s)|}))$ to the training set $\mathcal{D}$, where the predictors are the feature values of all available actions in state $s$ and the response variable is the identity of the selected action.

The agent only keeps the most recent $n \in \mathbb{N}$ choice sets in $\mathcal{D}$. The parameter $n$ should be high enough such that the supervised learning algorithm has enough data to train a new policy, but not too high to not learn on data that is too old (and thus was created using a not-as-good rollout policy).

▸ TRAINING A NEW POLICY. Periodically (in our case, after each rollout procedure), feature weights are updated through multinomial logistic regression on

[8] See Section 2.3.2.

[9] See Section 2.1.2 on a discussion about the differences between discrete choice and classification.

[10] That is, M-learning is an online algorithm in the sense that the training set is created using the current state of the environment. This stands in stark contrast with existing rollout-based algorithms that use a pre-existing set of rollout starting states in every iteration, see, for example, Lagoudakis and Parr (2003) and Scherrer et al. (2015).

---

**Algorithm 5** M-LEARNING to learn a policy $\pi$.

---

**Notation:** $p \in \mathbb{N}$ is the number of features, $\mathcal{A}(s)$ is the set of actions available in state $s$.

**Output:** $\boldsymbol{\beta} \in \mathbb{R}^p$, a vector of action-utility weights, initialized randomly.

**Input:**

$U(s, a) = f(\boldsymbol{\beta}, \boldsymbol{\phi}(s, a))$, where      *// action-utility function, e.g., linear*

     $\boldsymbol{\phi}(s, a) \in \mathbb{R}^p$            *// vector of state-action features*

$\mathcal{D} = \varnothing$      *// data structure to store choice sets (e.g., Table 2.2)*

$M \in \mathbb{N}$           *// number of rollouts*

$T \in \mathbb{N}$           *// rollout length*

$\gamma \in [0, 1]$           *// discount factor*

$n(k) : \mathbb{N} \to \mathbb{N}$           *// batch size at step k*

$\pi_r(s, \boldsymbol{\beta}) : \mathcal{S} \times \mathbb{R}^p \to \mathbb{R}$ *// rollout policy that returns an action for given s and $\boldsymbol{\beta}$*

$s \leftarrow$ state sampled from initial state distribution
**for** $k = 0, 1, 2, \ldots$ **do**
     **for all** $a \in \mathcal{A}(s)$ **do**
         $\hat{U}(s, a) \leftarrow$ ROLLOUT$(s, a, \pi_r(s, \boldsymbol{\beta}), M, T)$      *// see Algorithm 3*
     **end for**
     $\tilde{a} \leftarrow \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \hat{U}(s, a)$
     Take action $\tilde{a}$ and observe new state $s'$
     **if** $s'$ is not terminal **then**
         $\mathcal{D} \leftarrow \mathcal{D} \cup \{\{\tilde{a}, \boldsymbol{\phi}(s, a_1), \boldsymbol{\phi}(s, a_2), ..., \boldsymbol{\phi}(s, a_{|\mathcal{A}(s)|})\}\}$ *// append choice set*
         $s \leftarrow s'$
     **else**
         $s \leftarrow$ state sampled from initial state distribution      *// reset episode*
     **end if**
     Construct batch $\mathcal{D}_k$ using $n(k)$ most recent choice sets from $\mathcal{D}$
     Update $\boldsymbol{\beta}$ using multinomial logistic regression on batch $\mathcal{D}_k$
**end for**

---

the accumulated training set. The new set of weights maximizes the likelihood of the selected actions in the training set if the agent were to use action-selection probabilities

$$pr_\beta(s, a) = \frac{e^{U_\beta(s,a)}}{\sum_{a' \in \mathcal{A}(s)} e^{U_\beta(s,a')}},$$

where $U_\beta(s, a) := \boldsymbol{\beta}^T \boldsymbol{\phi}(s, a)$ is the *utility*[11] of an action $a$ in state $s$. The corresponding log-likelihood function of the parameter vector $\boldsymbol{\beta}$ given the data set $\mathcal{D}$ can be written as

$$\log \mathcal{L}(\boldsymbol{\beta}|\mathcal{D}) = \sum_{(s_i, \tilde{a}_i) \in \mathcal{D}} \log(pr_\beta(s_i, \tilde{a}_i)), \tag{5.2}$$

where $\tilde{a}_i$ is the action that was selected in the $i$-th choice set of the training set $\mathcal{D}$. Note that $\log(pr_\beta(s_i, \tilde{a}_i))$ depends on $\boldsymbol{\beta}$ because $U(s, a)$ depends on $\boldsymbol{\beta}$.

▸ REGULARIZED M-LEARNING. Let $P(\boldsymbol{\beta})$ denote a regularization term and let $\lambda \geq 0$ denote a regularization strength. A regularized version of M-learning is easily obtained by maximizing the following modified log-likelihood function

$$\log \mathcal{L}(\boldsymbol{\beta}|\mathcal{D}) - \lambda P(\boldsymbol{\beta}),$$

where $\lambda > 0$ is the regularization strength.

In Section 5.4, we will use M-learning with STEW regularization. In Chapter 4 we found, in the context of regression, that the STEW model performs particularly well if feature directions are known beforehand and used to *direct* all features to have positive weights.[12] In the following section we propose an algorithm that learns feature directions in a rollout-based reinforcement learning setting.

## 5.3 LEARNING FEATURE DIRECTIONS (LFD)

We present a learning algorithm, named LFD, that learns feature directions for a linear policy in a Markov decision process. In the context of Equation 5.1, the goal here is to learn a policy

$$\pi(s) = \underset{a \in \mathcal{A}(s)}{\operatorname{argmax}} \, \boldsymbol{d}^T \boldsymbol{\phi}(s, a), \tag{5.3}$$

where $\boldsymbol{d} \in \{-1, 0, 1\}^p$ are the feature directions to be estimated. Note that the weights in Equation 5.3 are invariant to scaling: the policy remains unchanged when all weights $\boldsymbol{d}$ are multiplied with the same positive scalar.

On a high level the LFD algorithm works as follows. Feature directions are initialized to zero; they are said to be *undecided*. The agent navigates the environment using a rollout mechanism to select actions and keeps track of how often each feature is associated positively and negatively with selected actions. A feature is assigned a direction when the difference between positive and negative associations is deemed to be significant. The algorithm terminates when

[11] We explicitly do not use the term "value" here, because the utility in this context is not necessarily an estimate of the entire expected cumulative return obtained after taking the action. The utility defined here is only meaningful relative to the utilities of other actions in the same state.

[12] See also Section 2.2.1.

all feature directions have been decided. Pseudo-code for LFD is provided in Algorithm 6.

---

**Algorithm 6** Learning feature directions (LFD)

---

**Output:**
$d \in \{-1, 0, 1\}^p$                            *// feature directions, initialized to $\mathbf{0}$*
**Input:**
$\alpha \in (0, 1)$                              *// significance threshold*
$\pi_r(s, d) : \mathcal{S} \times \{-1, 0, 1\}^p \to \mathcal{A}$       *// rollout policy using current feature directions*

  $s \leftarrow$ state sampled from initial state distribution
  $n_i^+ \leftarrow 0; n_i^- \leftarrow 0$, for $i = 1, \ldots, p$    *// initialize positive and negative training instances*
  **while** not all directions are learned **do**
    **for all** $a \in \mathcal{A}(s)$ **do**
      $\hat{U}(s, a) \leftarrow \text{ROLLOUT}(s, a, \pi_r(s, d))$            *// see Algorithm 3*
    **end for**
    $\tilde{a} \leftarrow \underset{a \in \mathcal{A}(s)}{\text{argmax}} \, \hat{U}(s, a)$
    Take action $\tilde{a}$ and observe new state $s'$
    **if** $s'$ is not terminal **then**
      **for all** $i = 1, \ldots, p$ **do**
        $\Delta_i = \text{sgn}\left( \sum_{a \neq \tilde{a}} \text{sgn}\left( \phi_i(s, \tilde{a}) - \phi_i(s, a) \right) \right)$
        $n_i^+ \leftarrow n_i^+ + \max(\Delta_i, 0)$
        $n_i^- \leftarrow n_i^- - \min(\Delta_i, 0)$
        *p-val* $\leftarrow$ test $H_0 \colon n_i^+/(n_i^+ + n_i^-) = 0.5$
        **if** *p-val* $< \alpha$ **then**
          $d_i \leftarrow \begin{cases} 1 & \text{if } n_i^+ > n_i^- \\ -1 & \text{otherwise} \end{cases}$
        **end if**
      **end for**
      $s \leftarrow s'$
    **else**
      *// reset episode*
      $s \leftarrow$ state sampled from initial state distribution
    **end if**
  **end while**

---

The LFD algorithm uses the same basic rollout mechanism as M-learning but uses the rollout data in a different way, as described next. The rollout policy uses features for which a direction has already been determined, while ignoring features with undecided directions. That is, the rollout policy is given by $\pi_r(s) = \underset{a \in \mathcal{A}(s)}{\text{argmax}} \, d^T \phi(s, a)$, where $d$ is the vector of current direction estimates. Let $\tilde{a}$ denote the action chosen by such a rollout procedure for a given state $s$ and let

$$\phi(s, a_1), \phi(s, a_2), \ldots, \phi(s, a_{|\mathcal{A}(s)|})$$

denote the feature values of all actions available in state $s$. Furthermore, let sgn

denote the mathematical sign function:

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0. \end{cases}$$

A training instance $\Delta_i$ for feature $\phi_i$ compares the feature values of the selected action $\tilde{a}$ to the feature values of all other available actions in the same state. A training instance can be positive or negative and is defined as

$$\Delta_i = \text{sgn}\left( \sum_{a \neq \tilde{a}} \text{sgn}\left( \phi_i(s, \tilde{a}) - \phi_i(s, a) \right) \right).$$

For example, a positive training instance means that feature value $\phi_i$ was larger for the chosen action $\tilde{a}$ than for other actions more often than it was smaller.

Let $n_i^+$ denote the number of positive training instances and let $n_i^-$ denote the number of negative training instances. A feature is assigned a direction only after the difference between $n_i^+$ and $n_i^-$ is found to be statistically significant. We use a two-sided exact binomial test[13] with null hypothesis that feature $\phi_i$ has no direction, that is,

$$H_0\colon \frac{n_i^+}{n_i^+ + n_i^-} = 0.5.$$

If the resulting p-value is smaller than some pre-defined threshold $\alpha$, the feature is assigned the direction $d_i = \text{sgn}(n_i^+ - n_i^-)$.

[13] See, for example, Howell (2009).

## 5.4 ITERATIVE POLICY-SPACE EXPANSION (IPSE)

Here we combine the LFD algorithm and M-learning with STEW regularization to construct a reinforcement learning algorithm that decouples the estimation of weight signs from the estimation of weight magnitudes. LFD is employed first, until all feature directions are learned. The algorithm then switches to M-learning, treating the learned directions as useful prior knowledge.

Under conditions defined further below, the combined algorithm learns in a monotonically expanding policy space. For this reason we call this algorithm *iterative policy-space expansion*, or IPSE. Note that building blocks other than LFD or M-learning could be used to create algorithms that learn in monotonically expanding policy-spaces. In the remainder of this chapter we will use the acronym IPSE to refer to the version that uses LFD and M-learning with STEW penalty.

▶ THE TRANSITION FROM LFD TO REGULARIZED M-LEARNING. The directions learned in the first phase of the IPSE algorithm (using the LFD algorithm) implicitly define an equal-magnitudes policy. In the second phase of the IPSE algorithm the weights are allowed to gradually deviate from this strongly constrained solution, as more high-quality training data is obtained. To this aim,

we use M-learning with the regularization term

$$P_d(\boldsymbol{\beta}) = \|\boldsymbol{\beta} - \boldsymbol{d}\|_2^2 = \sum_{i=1}^p (\beta_i - d_i)^2,$$

where $\boldsymbol{d}$ are the directions obtained by the LFD algorithm. The corresponding maximization problem that is solved in every iteration of the M-learning phase is given by

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\operatorname{argmax}} \log \mathcal{L}(\boldsymbol{\beta}|\mathcal{D}) - \lambda \|\boldsymbol{\beta} - \boldsymbol{d}\|_2^2, \qquad (5.4)$$

where $\lambda > 0$ is, as always, the regularization strength. The complete algorithm is given in Algorithm 7.

---

**Algorithm 7** Iterative policy space expansion (IPSE) to learn a linear policy $\pi$.

  **Output:**
  $\pi(s) : \mathcal{S} \to \mathcal{A}$       // *policy that returns an action $a \in \mathcal{A}$ for given state $s \in \mathcal{S}$*

  // *Phase 1: learn feature directions.*
  $\boldsymbol{d} \leftarrow$ LFD                           // *Algorithm 6*
  // *Define regularization term based on learned directions.*
  $P_d(\boldsymbol{\beta}) = \|\boldsymbol{\beta} - \boldsymbol{d}\|_2^2$
  // *Phase 2: regularized M-learning.*
  $\boldsymbol{\beta} \leftarrow$ M-LEARNING with $P_d(\boldsymbol{\beta})$-regularization     // *Algorithm 5*

---

In the context of the discrete choice problem considered here, and if all features are directed to be positive, the penalty term $P_d(\boldsymbol{\beta})$ is equivalent to the STEW penalty term defined in Section 4.2 in the previous chapter.[14] In the remainder of this section, we therefore refer to $P_d(\boldsymbol{\beta})$ as a STEW penalty.

[14] See Appendix A for a more detailed explanation of the equivalence between both regularization terms in the present context.

▸ THE POLICY SPACE OF IPSE. We derive necessary conditions on the regularization strength of the STEW penalty (controlled by the parameter $\lambda$) that ensure that IPSE learns in monotonically expanding policy spaces. The policy space at any given iteration is characterized by the values that the policy weight vector $\boldsymbol{\beta}$ can attain. Initially, IPSE learns directions using the LFD algorithm; the policy space is therefore constrained to be $\Pi_{\mathrm{LFD}} = \{-1, 1\}^p$. Figure 5.1 shows this policy space when there are $p = 2$ features.

During the M-learning phase, the policy space is a function of the regularization term used with multinomial logistic regression and the corresponding regularization strength $\lambda$. For $\lambda = 0$ (that is, no regularization at all), the policy space is simply given by $\mathbb{R}^p$. For $\lambda > 0$, we can reformulate the unconstrained optimization problem of Equation 5.4 as the constrained optimization problem

$$\underset{\boldsymbol{\beta} \in \mathbb{R}^p}{\operatorname{argmax}} \log \mathcal{L}(\boldsymbol{\beta}|\mathcal{D}), \quad \text{such that } \|\boldsymbol{\beta} - \boldsymbol{d}\|_2^2 \le c(\lambda),$$

where $c(\lambda) : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ is a decreasing function of $\lambda$, for which the following holds true: $c(\lambda) \to \infty$ for $\lambda \to 0$; and $c(\lambda) \to 0$ for $\lambda \to \infty$.
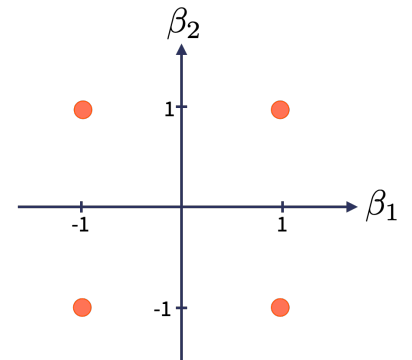


Figure 5.1: Policy space of the LFD algorithm for $p = 2$ features. The policy space is given by $\Pi_{\mathrm{LFD}} = \{(-1, -1), (-1, 1), (1, -1), (1, 1)\}$.

It follows that the policy space for a given $\lambda$ during the M-learning phase is given by

$$\Pi_\lambda = \{\boldsymbol{\beta} \in \mathbb{R}^p | \sum_{i=1}^{p} (\beta_i - d_i)^2 \le c(\lambda)\}, \tag{5.5}$$

which is a hypersphere around the equal-weights solution that was found by the LFD algorithm. The size of that hypersphere is a decreasing function of the regularization strength $\lambda$. Figure 5.2 sketches $\Pi_\lambda$, as a function of decreasing regularization strength $\lambda$, for $p = 2$ features and $\boldsymbol{d} = (d_1, d_2) = (-1, 1)$. For $p = 3$ the policy space $\Pi_\lambda$ would be a sphere, centered around the point $\boldsymbol{d} = (d_1, d_2, d_3)$, whose size increases with decreasing $\lambda$.

Let $\{\lambda_k\}_{k=1}^{\infty}$ denote a sequence of decreasing regularization strengths. It then follows that $\boldsymbol{d} \subset \Pi_{\lambda_k} \subset \Pi_{\lambda_{k+1}} \subset \mathbb{R}^p$, or in other words, the policy space is monotonically expanding.



Figure 5.2: Policy space $\Pi_\lambda$ (Equation 5.5) for $p = 2$ features as a function of the regularization strength $\lambda$, if the feature directions learned by the LFD algorithm in the first phase were $\boldsymbol{d} = (-1, 1)$. The policy space expands as $\lambda$ decreases.

▸ CHOICE OF $\lambda$. In practice, the regularization strength of regularized linear models is usually chosen using cross validation (compare, for example, Section 4.5). Here, we use a pre-defined schedule of decreasing regularization strengths in order to ensure a monotonically expanding policy space. We generally aim to find a schedule that satisfies the following two properties.

1. The regularization strength should initially be high enough to ensure a smooth transition from LFD to M-learning.

2. The regularization strength should decrease rapidly enough so that the policy space is not overly constrained for too long.

Both these properties are satisfied in the following example.

▸ EXAMPLE WEIGHT TRAJECTORIES. Figure 5.3 shows policy weight trajectories of the IPSE algorithm obtained while learning to play Tetris (see Section 5.6.1 for a detailed description of the experimental setup). IPSE used regularization strength $\lambda_k = 5/k$ in the $k$-th iteration of STEW-regularized M-learning. Each curve shows the weight estimate of one of the eight features as learning progresses.

All weights were initialized to zero and the LFD algorithm started to learn feature directions. In this particular example, the last missing direction was learned at iteration 36, as indicated by the dashed grey vertical line. Specifically, all features were assigned a *negative* direction of $-1$, except *Eroded cells*, which was assigned a *positive* direction of 1. In the iterations directly following the transition to M-learning, the estimated weights remained relatively close to the equal-weighting solution, indicating that the policy space was still tightly constrained around the LFD solution. As the regularization strength decreased, the policy space expanded and the feature weight estimates increasingly deviated from the equal-weighting solution.
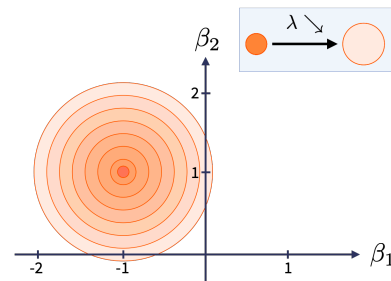
Figure 5.3: Policy weight trajectories of the IPSE algorithm in Tetris. The dashed vertical line signifies the transition from the LFD algorithm to M-learning with STEW penalty. Weights were rescaled such that the weight *rows with holes* always had an absolute value of 1.

## 5.5    RELATED LITERATURE

▸ CURRICULUM REINFORCEMENT LEARNING. Various curriculum learning[15] approaches have been proposed for reinforcement learning.[16] The most direct approach is to generate a sequence of related MDPs of increasing difficulty. For example, for the game of chess a curriculum could be created by considering smaller "sub-games" with smaller board sizes, fewer pieces, or simplified rules. One obvious downside of this explicit form of curriculum learning is that the sequence of MDPs has to be defined by an expert before learning starts or during the learning process (similar to *interactive reinforcement learning*[17]), which might not always be as easy to achieve as in the chess example.

▸ REGULARIZATION IN REINFORCEMENT LEARNING. The IPSE algorithm uses different forms of regularization to adapt the policy learning task difficulty throughout the learning process. Previous work has studied regularization in the context of reinforcement learning.[18] For example, Loth et al. (2007) use Lasso/$l_1$-regularization[19] to induce sparsity in the value function approximator in the context of a temporal difference learning. Farebrother et al. (2018) study whether $l_2$-regularization or dropout regularization[20] in DQN networks helps the reinforcement learning agent to generalize better to unknown domains.

There are several differences between this chapter and the studies mentioned above. First, the regularization used in the IPSE algorithm is centered around the equal-weighting strategy and uses the STEW regularization term, whereas the other works study $l_1$, $l_2$, or dropout regularization. Second, the IPSE algorithm studies regularization as intrinsically regulated curriculum learning to speed up learning, whereas existing work focusses on generalization to unknown domains and estimation error of value functions.

[15] Bengio et al. (2009)

[16] See Narvekar et al. (2020) for a recent survey.

[17] Thomaz and Breazeal (2006)

[18] Xu et al. (2007), Loth et al. (2007), Farahmand et al. (2008, 2009), and Farebrother et al. (2018)

[19] See Section 4.1 for a discussion of Lasso (or $l_1$) regularization in the context of regression.

[20] Srivastava et al. (2014)

## 5.6 EXPERIMENTS

We next present results from our experiments in Tetris. The main objective of these experiments is to examine whether the IPSE algorithm can learn an expert Tetris policy more quickly than existing state-of-the-art algorithms. Furthermore, we perform several ablation studies to analyze the relative importance of various elements of the IPSE algorithm.

### 5.6.1 *Tetris*

The video game Tetris (Figure 5.4 shows a screenshot of the game) is an important benchmark for artificial intelligence research and, in particular, reinforcement learning. A history of Tetris as well as a summary of existing machine learning solutions to the game can be found in Algorta and Şimşek (2019).

Tetris is played on a two-dimensional grid (also called board), which is empty at the beginning of the game. The board is filled up by pieces that are falling down from the top, one at a time. The pieces are of different shapes but always fill out 4 connected grid cells. They are shown in Figure 5.5.

While the pieces are falling down, the player can rotate the pieces and move them horizontally to determine where, and in which position, the piece lands. The player can create new space on the board by *clearing lines*: whenever an entire row on the board is filled with pieces, the whole row is deleted. The game ends when the board is so full that the next piece cannot appear on the board.

▸ TETRIS IN REINFORCEMENT LEARNING. Tetris can be formulated as a Markov decision process, where the state consists of the board configuration and the identity of the falling tetrimino. Available actions are the possible placements of the tetrimino on the board. The number of actions available in a given state ranges from 0 to 34. Figure 5.6 shows an example of possible placements (bottom row) for a given state (top row). A reward of 1 is received for each cleared line. The game ends when a state allows no further legal placement. We used a board size of $10 \times 10$ in all experiments.



Figure 5.4: Tetris on the Nintendo Game Boy.



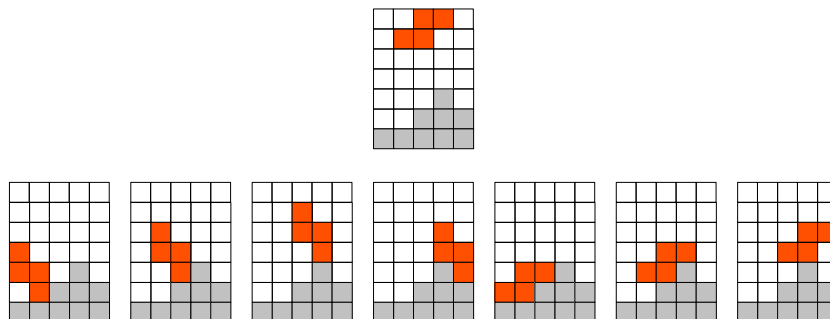Figure 5.5: The seven *Titriminos* of size 4 used in the classic version of Tetris.



Figure 5.6: Top row: example of a tetris state on a $7 \times 6$ board. Bottom row: all seven available actions as defined by the possible placements of the falling Tetrimino.

We use eight features to describe a state-action pair: *landing height*, *number*

*of eroded piece cells*, *row transitions*, *column transitions*, *number of holes*, *number of board wells*, *hole depth*, and *number of rows with holes*. These features are from earlier work by Thiery and Scherrer (2009a), who describe them in detail.

▸ ALGORITHMS. We compared the IPSE algorithm to a state-of-the-art algorithm from the literature and several modifications of the IPSE algorithm:

*Classification-based modified policy iteration*,[21] or (CBMPI), is the best-performing reinforcement learning algorithm for Tetris reported in the literature. The algorithm is described in detail in Appendix A.

[21] Gabillon et al. (2013) and Scherrer et al. (2015)

*LFD* learns feature directions using the LFD algorithm and uses these feature directions to define an equal-weighting policy.

*M-learning (STEW)* directly starts with the second phase of the IPSE algorithm, that is, without learning feature directions beforehand. This is Algorithm 5 with STEW regularization.

*M-learning (STEW, known directions)* is given prior knowledge about feature directions.[22] This algorithm thus can "skip" the LFD phase. The algorithm is considered an upper baseline.

[22] The feature directions were obtained from the weights of the BCTS policy (Thiery and Scherrer, 2009a). The same feature directions were also used in Şimşek et al. (2016).

*M-learning (no regularization)* is Algorithm 5 without any regularization.

Learning performance was measured by the quality of the policy as a function of the computational resources used during learning. The quality of a policy can be easily estimated by playing some test games and averaging the scores. Yet because different algorithms use different learning mechanisms, we require a common unit of measurement for the amount of resources used across all algorithms. In line with previous work on Tetris,[23] we used the number of *calls to the generative model* of the Tetris engine as the central metric for resource intensiveness. Next we describe how the this metric depends on the rollout parameter settings for different algorithms.

[23] Scherrer et al. (2015) and Lazaric et al. (2016)

▸ ROLLOUT PARAMETER SETTINGS. The IPSE algorithm, LFD + EW, as well as the various versions of M-learning all use a similar rollout procedure, in which only the current state of the environment is used as a rollout starting state. We used the same rollout parameters for all these algorithms. In a given state, the algorithms computed $M = 10$ rollouts of length $T = 10$ for each available action. Given that the number of actions is always smaller than 34, the maximum number of calls to the generative model of Tetris for one iteration of the respective algorithm was at most $34TM = 3400$.

This is different for CBMPI. The CBMPI algorithm uses a pre-computed set of rollout starting states. In each iteration of the algorithm, the algorithm performs a rollout on a large number of rollout starting states, thus requiring more calls to the generative model per iteration. The Tetris results for CBMPI reported by Scherrer et al. (2015) on the $10 \times 10$ board used a per-iteration budget

of 8,000,000 calls to the generative model. In comparison, the total budget (after 400 iterations) we used for all other algorithms was 1,360,000. In order to compare the algorithms meaningfully, we experimented with CBMPI using budgets in the same range as M-learning. We present results with CBMPI using a per-iteration budget of 170,000 (resulting in 8 iterations of the algorithm).

▸ RESULTS. After each iteration of the algorithm, the quality of the learned policy was evaluated by playing 10 games of Tetris. Figure 5.7 shows mean scores across 100 replications, with shaded areas corresponding to standard error of the mean.
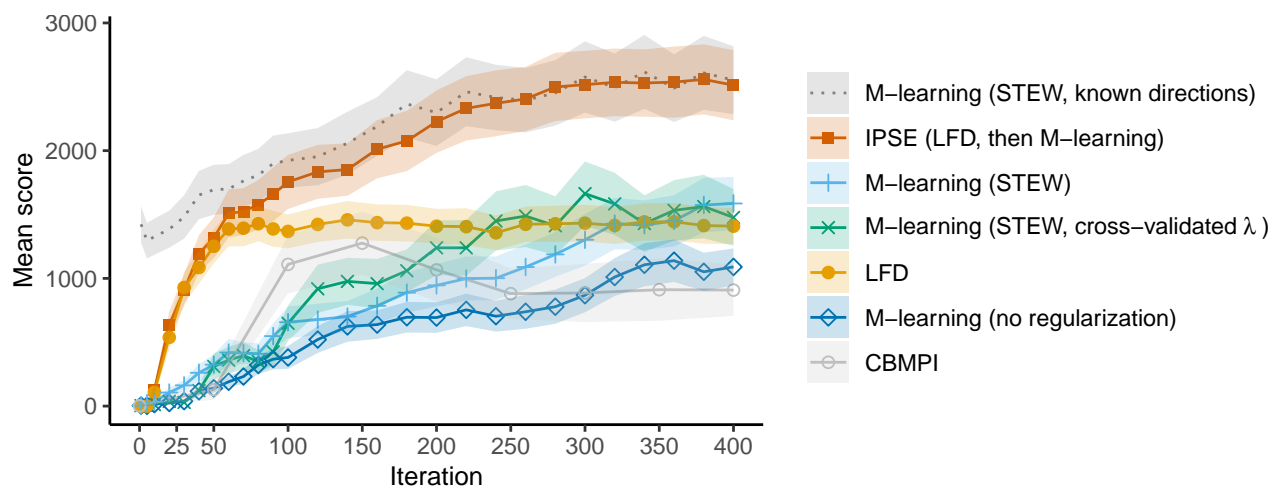


Figure 5.7: Quality of the policy learned as a function of the iterations of the algorithm. Each learning curve shows means across 100 replications of the algorithm. Quality of the policy is measured by the mean score obtained by the policy in 30 Tetris games.

M-learning with given feature directions represents an upper baseline. This is the dotted line in the figure. Among algorithms that were not given prior knowledge about feature directions, IPSE showed the highest learning rate and learned the best policies overall. Furthermore, it rapidly approached the ceiling performance obtained with known feature directions. All other algorithms learned more slowly. At 400 iterations, there was a large performance gap between IPSE and all other algorithms.

The hypothesis that IPSE benefits from learning directions independently was supported by the strong performance of the standalone LFD algorithm at the beginning of the learning curve. This indicates that IPSE could fruitfully use the naïve direction estimates as a stable basis for later learning.

The results obtained for the standalone LFD algorithm also support another hypothesis developed in the previous two chapters: Always using the same single model of bounded rationality (equal weighting, in this case) does not yield the best results. Using LFD (and thus, equal weighting) across the entire learning process did not perform as well as the IPSE algorithm, which used equal weighting at the beginning of the learning process but shifted to a more data-driven approach as more data became available.

In 1772, Benjamin Franklin received a plea for advice on a difficult career decision from his friend and fellow scientist Joseph Priestley. [24] In his reply, he described what later became known as *Franklin's rule*:[25]

> "My way is to divide half a sheet of paper by a line into two columns, writing over the one *Pro*, and over the other *Con*. […] I put down under the different heads short hints of the different motives that at different times occur to me for or against the measure. When I have thus got them all together in one view, I endeavor to estimate their respective weights […] If I judge some two reasons con equal to some three reasons pro, I strike out the five; and thus proceeding I find at length where the balance lies [and] come to a determination accordingly." (p. 878 in Franklin, 1987)

Almost 250 years later, pros-and-cons lists are still used extensively. What makes such a simple tool so effective? One aspect could be that the main problem of estimating the relative importance of arguments is facilitated by first solving the much simpler subproblem of deciding—for each feature individually—whether it is positively or negatively associated with the response.

The IPSE algorithm presented in this chapter is similar to Franklin's rule in that the algorithm first learns, for each feature individually, whether it is positively or negatively associated with good decision outcomes. Similar to how people structure their thoughts by categorizing arguments into *pro* and *contra*, learning feature directions has proven to be a useful building block for learning more complex policies.

The experimental results presented here showed that reinforcement learning algorithms can benefit from learning in a policy space that initially is strongly constrained but expands during the learning process. A decreasingly constrained policy space could also be seen as an increasingly capable cognitive mechanism, which is consistent with the view of the cognitive mechanism of humans and great apes that initially has very low capacity but grows during development.[26]

One limitation of the work presented in this chapter is that we considered a problem for which useful, hand-crafted features were available. An interesting direction for future work is to extend the presented approach to reinforcement learning agents that determine appropriate features in the first place, for example, by using non-linear function approximators such as artificial deep neural networks. Everything else being equal, deep neural networks are more prone to overfitting small data sets than linear functions. I believe that accounting for limited resources at the beginning of the learning process therefore could be even more beneficial for deep reinforcement learning agents than for linear agents. However, it is unclear whether the notion of feature directions is useful for certain neural network architectures such as convolutional neural networks.

[24] Franklin (1987)

[25] Gigerenzer et al. (1999)

[26] Krueger and Dayan (2009), Brown et al. (2005), and Miller and Cohen (2001)

Part III

BOUNDEDLY RATIONAL ACTION SELECTION

# 6

## SATISFICING POLICIES IN MARKOV DECISION PROCESSES

Herbert Simon's satisficing strategy[1] for decision making is simple: consider decision alternatives sequentially and select the first alternative whose estimated value is higher than a pre-specified aspiration level. The possible benefit of such a strategy compared to the value-maximizing approach is the following. If a satisfying alternative is found early, there is no need to consider and evaluate the remaining alternatives, which can result in considerable computational savings. And yet, despite its simplicity and apparent possible advantages, the satisficing strategy has seen only very limited applications in artificial intelligence research.

[1] Simon (1956)

The reason for the negligence of satisficing and other models of bounded rationality does not always seem to be a disbelief in or a rejection of the notion of bounded rationality *per se*. For example, Russell and Norvig (2010, p. 1049) write that

> ... [satisficing] appears to be a useful model of human behaviors in many cases. It is not a formal specification for intelligent agents, however, because the definition of 'good enough' is not given by the theory.

In this chapter I propose several strategies for setting aspiration levels in sequential decision making problems under uncertainty and compare these strategies to the classical value-maximizing strategy for decision making.

I start by formalizing a satisficing strategy for use in Markov decision processes. More specifically, I propose the *ξ-satisficing policy*, which, in the spirit of Simon's work, considers available actions sequentially and selects the first action whose value is larger than the policy aspiration level $\xi$. By selecting a possibly sub-optimal action before all actions are considered, the satisficing policy trades off action quality against computational effort.

I first analyze effort and quality of the $\xi$-satisficing policy, as a function of the aspiration level $\xi$, in the simplified setting of a single decision within a Markov decision process. This analysis indicates that the effort-quality trade-off curve offered by the $\xi$-satisficing policy is highly non-linear in the sense that it yields a large reduction of effort for a small loss in quality.

This initial analysis also directly informs the development of three *aspiration adaption rules*, which are rules for setting aspiration levels dynamically when the $\xi$-satisficing policy is used at each decision stage of the Markov decision process.

The first of these rules is *aspiration tracking.* Theoretical analysis in the context of deterministic Markov decision processes shows that when the agent has access to an optimal value function, the satisficing policy with aspiration tracking is provably more efficient than the value-maximizing (greedy) policy in the sense that the former requires less expected effort than the latter, without giving up any expected quality at all.

I evaluate the performance of aspiration tracking in a deterministic grid-world environment with a large discrete action space, where the evaluation of all actions (as required by the greedy policy) is a computational bottleneck. When given access to an optimal value function, a satisficing agent using aspiration tracking reaches optimal return while requiring, on average, less than 10% of the effort required by the greedy policy.

In many complex reinforcement learning domains, however, the optimal value function is usually only approximated.[2] I show that the aspiration tracking rule is prone to accumulating small approximation errors, which can deteriorate the performance of the satisficing policy. This leads to the development of two other aspiration adaption rules, called *value tracking* and *valved value tracking*, which are less vulnerable to approximation errors of the value function.

[2] See Sutton and Barto (2018, Part II) or Section 2.3.

I evaluate all three aspiration adaption rules in the game Lunar-Lander, where the optimal value function is approximated by an artificial neural network. On average, a satisficing agent using valved value tracking reaches the same return as the greedy policy while requiring around 76% of the greedy policy's effort.

A satisficing strategy is only useful if the corresponding effort reduction is not offset by the effort required to determine an appropriate aspiration level in the first place.[3] The aspiration adaption rules proposed in this chapter require little effort. They are defined by cognitively plausible recursive functions such as the simple difference between two real-valued numbers.

[3] See also the related discussion about meta-level optimization in Section 2.3.

The main contribution of this chapter is two-fold. First, I show that the satisficing strategy can be successfully integrated into Markov decision processes by proposing cognitively plausible update rules for setting aspiration levels dynamically. Second, I provide theoretical evidence and experimental support that a boundedly rational satisficing agent using these aspiration adaption can be more resource-efficient than the perfectly rational greedy policy in the sense that the former requires less expected effort but yields the same expected quality as the latter.

## 6.1 PRELIMINARIES

In this chapter, I consider value-based reinforcement learning in sequential decision making problems that can be modeled as Markov decision processes (MDP).[4] We assume that the action space $\mathcal{A}(s)$ in any given state $s$ is discrete and we are especially interested in domains where the number of available actions, denoted by $|\mathcal{A}(s)|$, is large. To reduce notation, we make the assumption

[4] MDPs were introduced in Section 2.3.

that there is only one initial state $s_0$. All results presented in this chapter hold conceptually if the initial state is instead sampled from an initial state distribution.

We will analyze the computational effort required by various value-based policies when an action-value function $q_\pi$ is given.[5] We will consider both tabular algorithms and algorithms that use function approximation to represent the value function. When function approximation is used, we consider deep neural networks that generalize values across states *and* actions, as used in early work on deep reinforcement learning,[6] as well as more recently, in work on reinforcement learning for large discrete action spaces.[7] These networks take a feature representation of a state-action pair, $\phi(s, a)$, as input, and output the corresponding state-action-value estimate $\hat{q}(s, a)$. In a given state $s$, an agent that uses the greedy policy, and thus requires value estimates for all actions in $\mathcal{A}(s)$, has to make $|\mathcal{A}(s)|$ forward passes through the network.

## 6.2 EFFORT-QUALITY TRADE-OFF IN THE SPACE OF POLICIES

We are interested in value-based policies that produce useful behavior while requiring less computational effort than the greedy policy in the sense that these other policies require fewer than $|\mathcal{A}(s)|$ action-value computations (or look-ups) before selecting an action.

We now define more formally the notions of *effort* required to choose an action and *quality* of the chosen action of a policy $\pi$ with respect to an action-value function $q$ in a given state $s$.

---

**Definition 1** (Effort of a policy). The *effort* $e(\pi, s)$ denotes the random variable that describes the number of action-values that the policy $\pi$ computes before making a decision. The *expected effort* of the policy $\pi$ in a state $s$ is the expected value of $e(\pi, s)$ and will be denoted by

$$f(\pi, s) = \mathbb{E}[e(\pi, s)].$$

---

For instance, the random policy, denoted by $\pi_r$, requires an expected effort of $f(\pi_r, s) = 0$ because no action value has to be computed to make a decision. The greedy policy $\pi_*$ requires an expected effort of $f(\pi_*, s) = |\mathcal{A}(s)|$ because all action values have to be computed to determine the value-maximizing action.

> **Definition 2** (Quality of a policy). The *quality* of a policy $\pi$ with respect to a value function $q$ in a state $s$ is the expected $q$-value of the selected action and will be denoted by
>
> $$u(\pi, s) = \int_{\mathcal{A}(s)} q(a|s) d\pi(a|s).$$

For instance, the random policy yields a quality of $u(\pi_r, s) = \mathbb{E}[q(a|s)]$ and the greedy policy yields a quality of $u(\pi_*, s) = \max_{a \in \mathcal{A}(s)} q(a|s)$.

To simplify notation, we will sometimes just write $f$ for expected effort or $u$ for quality,[8] without explicitly conditioning on the state or policy when the context is unambiguous.

▶ TRADING OFF EFFORT AGAINST QUALITY. The greedy policy yields maximum quality but also requires high effort. One policy that requires zero effort is the random policy. However, the quality of uniformly random behavior is generally too low to be useful.

One naïve way of obtaining a policy with intermediate effort is to use the $\varepsilon$-greedy policy, which selects a value-maximizing or a random action with probabilities of $\varepsilon$ and $1 - \varepsilon$, respectively. The resulting interpolation is linear in the sense that a reduction in effort is proportionally matched by a reduction in quality. For instance, if $\varepsilon = 0.5$, the $\varepsilon$-greedy policy requires, on average, half of the greedy policy's effort. However, the expected quality of the chosen action is only half-way in between the quality of a greedily chosen action and a randomly chosen action.[9]

In the following section I will define the $\xi$-satisficing policy, which, like the $\varepsilon$-greedy policy, provides an interpolation of the greedy policy and the random policy. Unlike the $\varepsilon$-greedy policy, the satisficing policy navigates the effort-quality tradeoff on a non-linear trajectory, allowing the agent to benefit from a large reduction of effort by giving up only a small amount of quality, or in some cases, by not giving up any quality at all.

[8] It is somewhat unfortunate that the symbol for quality will be denoted by the letter $u$ and not by the letter $q$ (as in *q*uality). In the reinforcement learning literature, the letter $q$ is already "reserved" to denote the action-value function. You can think of "u" as in *u*tility for a mnemonic aid.

[9] The expected quality of an $\varepsilon$-greedy policy is given by $u(\pi_r) + \varepsilon[u(\pi_*) - u(\pi_r)]$. See also Figure 6.3d.

## 6.3 $\xi$-SATISFICING POLICIES: LOW-EFFORT DECISION MAKING

We now define the value-based $\xi$-satisficing policy, where $\xi \in \mathbb{R}$ denotes the *aspiration level*.

> **Definition 3** ($\xi$-satisficing policy). The $\xi$-satisficing policy with respect to a state-action value function $q$, denoted by $\tilde{\pi}(s, \xi, q)$, is to consider actions in $\mathcal{A}(s)$ sequentially according to a proposal distribution $d$, and to accept the first satisfactory action. An action $a$ is satisfactory if its value, $q(s, a)$, is larger than the aspiration level $\xi$. If no action in $\mathcal{A}(s)$ is found to be satisfactory, the policy is to choose a value-maximizing action.

An algorithmic description of the $\xi$-satisficing policy is provided in Algorithm 8. In the remainder of this chapter, we assume that $d$ is the uniform random distribution without replacement. That is, actions are simply considered in random order, and no action is considered more than once.

In the context of a single decision, expected effort and quality of the satisficing policy both depend on the value of the aspiration level $\xi$ relative to the distribution of $q$-values in the current state $s$, given by $q(a|s)$. This relationship is summarized in Figures 6.1 and described in more detail next.
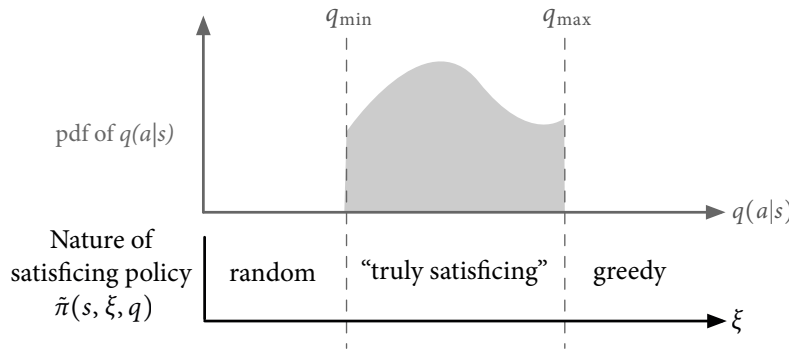


Figure 6.1: Nature of the $\xi$-satisficing policy as a function of the aspiration level $\xi$ in relation to the distribution of action-values in the current states, given by $q(a|s)$.

There are two extreme cases. Let $q_{\max} := \max_a q(a|s)$, then for $\xi > q_{\max}$, no action is ever satisfactory and the satisficing policy thus "falls back" to the greedy policy.[10] By contrast, let $q_{\min} := \min_a q(a|s)$, then for $\xi \leq q_{\min}$, the first (randomly chosen) action is always satisfactory and the satisficing policy thus yields the same quality as the random policy, while requiring one action evaluation.

In between these two extreme cases, a lower aspiration level generally leads to a higher number of actions deemed satisfactory, which reduces expected effort because fewer actions have to be evaluated until a satisfactory action is found. However, a lower aspiration also decreases the expected quality of the chosen action because more low-quality actions become satisfactory and could thus potentially be selected. Both effects are indicated in Figure 6.2.

In the remainder of this section, we aim to characterize the resulting trade-off in more detail by analyzing the relative magnitudes of both effects. Broadly speaking, we would like to know how large the reduction in expected effort is for a small reduction in quality (relative to the greedy policy). However, the desired effect sizes depend not only on the aspiration level chosen by the agent but also on the shape of the distribution $q(a|s)$. This distribution is generally not known to the agent and can therefore not be used to directly calculate a suitable aspiration level.

[10] If the algorithm keeps track of the currently highest $q$-value while searching for a satisfactory action, the greedy action can be determined immediately after consideration of all actions, thus not requiring any further computation.
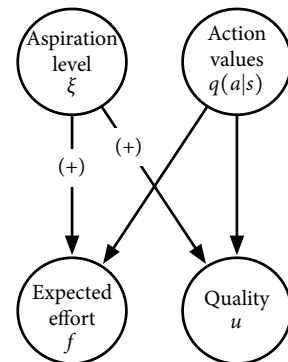


Figure 6.2: Influence of the aspiration level and the distribution of action values on quality and effort of the satisficing policy. The (+) sign on an arrow mean that the two variables connected by that arrow are positively correlated.

---

**Algorithm 8** $\tilde{\pi}(s, \xi, q)$ : a $\xi$-satisficing policy with respect to $q(s, a)$.

---

   **Output:** $a \in \mathcal{A}(s)$                                       *// action*

   **Input:**

   $\xi \in \mathbb{R}$                                    *// policy aspiration level*

   $d : \mathcal{A} \to \mathcal{A}$             *// action-sample distribution, e.g., uniform random*

   $s \in \mathcal{S}$                    *// state on which the policy is computed*

   $q(s, a) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$                 *// action-value function*

   $\varepsilon \in \mathbb{R}$            *// exploration parameter in case of non-satisfaction*

 

   $\mathcal{B} = \mathcal{A}(s)$                  *// set of actions to be considered*

   **while** $\mathcal{B} \neq \varnothing$ **do**

      sample $a$ from $\mathcal{B}$ according to $d$

      $\mathcal{B} \leftarrow \mathcal{B} \smallsetminus \{a\}$          *// remove action from consideration set*

      **if** $q(s, a) \geq \xi$ **then**

         **return** a       *// the procedure stops immediately and returns a*

      **end if**

   **end while**

   *// No satisficing action found*

   **return** $\varepsilon$-greedy action

---

### 6.3.1  *Characterizing the effort-quality tradeoff of satisficing using example distributions for $q(a|s)$*

Here we analyze quality and effort for three different example distributions of $q(a|s)$, shown in Figure 6.3a. These distributions correspond to different degrees of suitability of the satisficing policy to the decision environment, allowing us to study the satisficing policy in more or less "favorable" environments.

Specifically, the three distributions are Beta$(\alpha, \beta)$ distributions for different values of $\alpha$ and $\beta$. All distributions have the same minimum and maximum action values ($q_{\min} = 0$ and $q_{\max} = 1$). They differ in how the action-values are distributed between the two extremes. In a decision environment where action values follow a Beta$(3, 1)$ distribution (purple), most actions have high value and very few actions are of low value. Intuitively, this distribution is well suited for a satisficing agent because a random search can quickly find a high-quality action. By contrast, the Beta$(1, 3)$ distribution (green) corresponds to a situation in which most actions are of low value and very few actions are of high value; we expect that many actions have to be evaluated until a high-quality action is found. The Beta$(2, 2)$ distribution (orange) corresponds to an intermediate case. Or to summarize: purple is "favorable"; orange is "neutral"; and green is "unfavorable".

We use these three decision environments for our analysis as follows. Panels b to d of Figure 6.3 (each of which will be discussed in detail further below) show various quantities related to the effort-quality trade-off of the $\xi$-satisficing policy, as a function of the aspiration level $\xi$, in the three decision environments
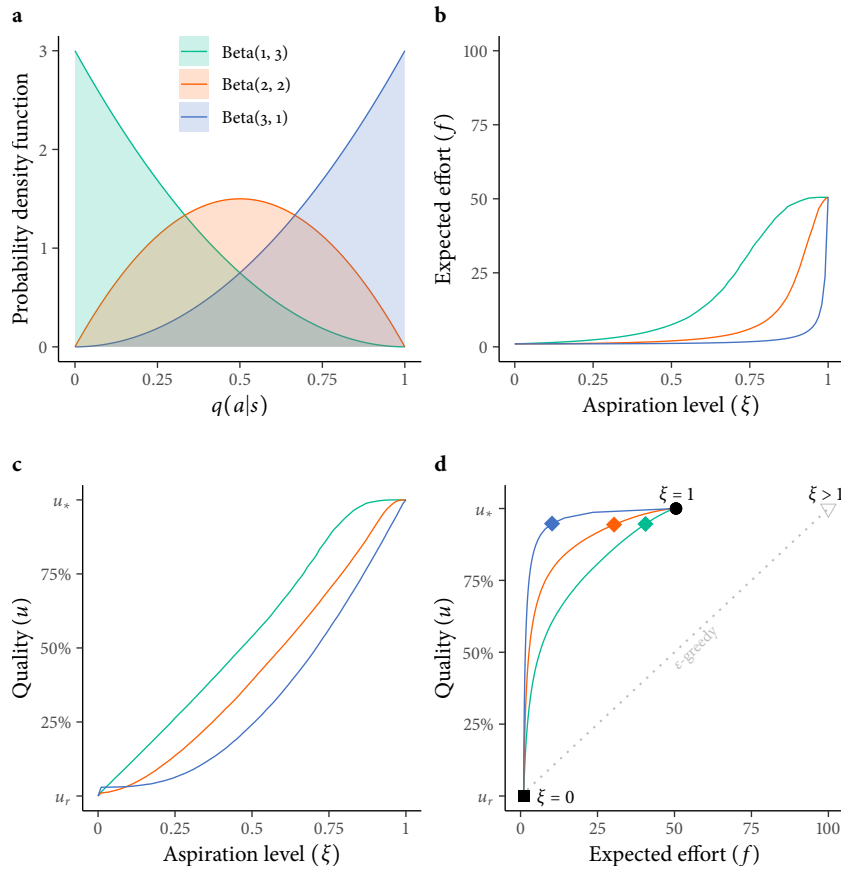
**a**, Probability density functions for various Beta$(\alpha, \beta)$ distributions. All solid lines in panels **b** to **d** show empirical means calculated across 1000 action sets, each consisting of $|\mathcal{A}| = 100$ actions whose values were randomly drawn according to the distributions shown in panel **a** (matching color-coding).

**b**, Expected effort $f$ as a function of the aspiration level $\xi$.

**c**, Quality of the satisficing policy as a function of the aspiration level $\xi$. The quality values (y-axis) are scaled to the range $[u_r, u_*]$ for each distribution of $q(a|s)$, where $u_r$ and $u_*$ are the qualities of the random and the greedy policy, respectively.

**d**, Effort-quality tradeoffs for the $\xi$-satisficing policy (solid lines) as the policy parameter is varied from $\xi = 0$ (■) to $\xi = 1$ (●). The colored diamonds ◆, ◆, and ◆ correspond to the aspiration levels that yield 95% of the quality of the greedy policy. The figure also shows the $\varepsilon$-greedy policy (dotted grey line), as the exploration parameter is varied from $\varepsilon = 1$ (■, random policy) to $\varepsilon = 0$ (▽, greedy policy). The point ▽ also corresponds to the $\xi$-satisficing policy for $\xi > 1$.

Figure 6.3:

just described. Each figure shows three solid colored lines, which correspond to the three distributions of $q(a|s)$ shown in Panel a, as indicated by matching colors.

▶ EFFORT OF SATISFICING. Figure 6.3b shows the expected effort $f$ of the satisficing policy as a function of the policy aspiration level $\xi$. Ideally, a policy is as close as possible to the bottom-right corner. In that regard, the Beta$(3, 1)$ distribution (blue) is preferred to the Beta$(2, 2)$ distribution (orange), which itself is preferred to the Beta$(1, 3)$ distribution (green), confirming our earlier intuition about which distribution of $q$-values is favorable for the satisficing policy.

▶ QUALITY OF SATISFICING. Figure 6.3c shows $u$ as a function of $\xi$. Ideally, we would like to reduce the aspiration level as much as possible (to reduce effort) while maintaining a high expected quality. That is, in this figure, the top-left corner is best. Note that the preference order between value distributions established in the previous paragraphs is reversed here: for a fixed aspiration level $\xi$, the Beta$(1, 3)$ distribution (green) maintains the highest quality whereas the Beta$(3, 1)$ distribution (blue) yields the lowest quality.

▷ Effort-quality trade-off for satisficing. We now combine the two individual effects discussed so far to obtain a more complete picture on how the satisficing policy trades off effort and quality. Figure 6.3d shows expected effort (x-axis) and quality (y-axis) of the $\xi$-satisficing policy as the aspiration level is varied from $\xi = 0$ (■) to $\xi = 1$ (●). A single point on any of these lines thus corresponds to the effort-quality tradeoff for one specific aspiration level. Ideally, a policy is as close as possible to the top-left corner.

To provide a baseline, the plot also shows the trade-offs obtained by the $\varepsilon$-greedy policy, as the exploration parameter is varied from $\varepsilon = 1$ (■, random policy) to $\varepsilon = 0$ (▽, greedy policy).

The three colored diamonds (◆, ◆, and ◆) show the satisficing policies that yield 95% of the quality of the greedy (relative to the random policy). To reach this level of quality, the satisficing policy requires 40% of the effort required by the greedy policy in the case of the unfavorable Beta$(1, 3)$ distribution of $q$-values. In the case of the favorable Beta$(3, 1)$ distribution, the satisficing policy requires only 10% of the effort to yield 95% of the quality that would be obtained by the greedy policy.

The effort-quality trade-offs obtained from using the satisficing policy generally seem useful in the sense that a relatively small reduction in quality led to a relatively large reduction in effort, even for the least favorable distribution of action values considered here. And yet it remains difficult to derive a general principle for setting aspiration levels from this analysis alone for the following reasons. First, the agent usually does not know which type of value distribution $q(s|a)$ is to be expected in the current state. Second, we analyzed three exemplary distributions which are not necessarily representative of action-value distributions found in the real world. Finally, the usefulness of a trade-off always depends on the relative importance given to effort and quality, and is thus ultimately application-dependent.

There is, however, one point in Figure 6.3d that represents a situation in which the $\xi$-satisficing policy outperforms the greedy policy, regardless of the relative importance given to quality and effort, and for all distributions of $q(a|s)$ that were considered: when $\xi = 1$ (indicated by ● in Figure 6.3d), the $\xi$-satisficing policy yielded maximum quality but required only half of the expected effort that is required by the greedy policy. I next characterize this situation more formally.

6.3.2 *Satisficing can be a Pareto improvement on the greedy policy.*

In welfare economics, a *Pareto improvement* is a change to a system that improves the (economic) well-being of at least one person without harming any other person's well-being in the same system.[11] In the context of the effort-quality trade-off discussed here, we say a policy $\pi_1$ is a Pareto improvement on a policy $\pi_2$, if $\pi_1$ requires strictly less effort or yields strictly higher quality

[11] The concept is named after economist Vilfredo Pareto (see, for example, Mas-Colell et al., 1995, Chap. 16).

than $\pi_2$ while being at least as good in the other dimension.

Ideally we would like to prove that the $\xi$-satisficing policy can be a Pareto improvement on the greedy policy for any action-value distribution $q(a|s)$. To this aim, I first introduce the *number of satisfactory actions*, denoted by $\tilde{n}$, as an intermediate variable in the analysis of effort, as shown in Figure 6.4. This allows us to prove Lemma 1, which quantifies the satisficing policy's expected effort $f$ as a function of $\tilde{n}$. This lemma then allows us to prove the Pareto improvement for a single decision (see Theorem 2) as well as an upper bound on the expected effort required by a satisficing agent when satisficing is used in each decision of an MDP (see Theorem 3). Moreover, the lemma provides intuition on why the satisficing policy's effort-quality trade-off shows the non-linearity observed in the previous section.

▷ EFFORT AS A FUNCTION OF THE NUMBER OF SATISFACTORY ACTIONS. Given a state $s$, an aspiration level $\xi$, and a distribution of $q$-values $q(a|s)$, the number of satisfactory actions $\tilde{n}(s, \xi, q(a|s)) = |\{a \in \mathcal{A}(s)|q(s,a) \geq \xi\}|$ is given by the number of actions that can possibly be selected by the satisficing policy in this situation. To simplify notation, we usually write just $\tilde{n}$ when the decision context is unambiguous.

Next we quantify the expected effort of the $\xi$-satisficing policy as a function of the number of satisfactory actions, $\tilde{n}$, when the action proposal distribution $d$ is uniformly random.



Figure 6.4: Analysis of effort using the number of satisfactory actions, $\tilde{n}$, as intermediary variable. The $(+)$ or $(-)$ sign on an arrow mean that the two variables connected by that arrow correlate positively or negatively, respectively. Compare to Figure 6.2.

---

**Lemma 1** (Expected effort of satisficing). For $\tilde{n} \in [1, |\mathcal{A}|]$ satisfactory actions, the expected effort of the satisficing policy is given by

$$f(\tilde{n}) = \frac{|\mathcal{A}| + 1}{\tilde{n} + 1}. \tag{6.1}$$

The expected effort of the satisficing policy decreases at a quadratically decreasing rate as $\tilde{n}$ increases. Specifically, for $\tilde{n} \in [2, |\mathcal{A}|]$, the difference in expected effort, as the number of satisfactory actions is increased from $\tilde{n} - 1$ to $\tilde{n}$, is given by

$$\delta_{\tilde{n}} := f(\tilde{n}) - f(\tilde{n} - 1) = -\frac{|\mathcal{A}| + 1}{\tilde{n}^2 + \tilde{n}}. \tag{6.2}$$

---

*Proof of Lemma 1.*

To prove Equation 6.1, we reformulate the satisficing policy as an "urn problem" from probability theory. Consider an urn that contains $|\mathcal{A}|$ balls (corresponding to available actions), of which $\tilde{n}$ balls are red (satisfactory actions) and the remaining balls are blue. The expected effort $f(\tilde{n})$ is given by the expected value of the following random variable, denoted by $X$. The random variable $X$ is the number of draws from the urn until the first red ball is drawn, if balls are drawn from the urn randomly without replace-

ment (because in satisficing, every action is evaluated at most once). This case has been treated in Ahlgren (2014), which provides the desired expression for $f(\tilde{n})$.

The expression for the difference in expected effort $\delta_{\tilde{n}}$ (Equation 6.2) is obtained by plugging in the expected values from Equation 6.1 and applying simple algebra.

$$
\begin{aligned}
\delta_{\tilde{n}} &:= \mathbb{E}[X|\tilde{n}] - \mathbb{E}[X|\tilde{n}-1] \\
&= \frac{|\mathcal{A}|+1}{\tilde{n}+1} - \frac{|\mathcal{A}|+1}{\tilde{n}} \\
&= \frac{\tilde{n}(|\mathcal{A}|+1)}{\tilde{n}(\tilde{n}+1)} - \frac{(\tilde{n}+1)(|\mathcal{A}|+1)}{\tilde{n}(\tilde{n}+1)}. \\
&= \frac{\tilde{n}(|\mathcal{A}|+1) - (\tilde{n}+1)(|\mathcal{A}|+1)}{\tilde{n}(\tilde{n}+1)}. \\
&= -\frac{|\mathcal{A}|+1}{\tilde{n}^2+\tilde{n}}.
\end{aligned}
$$

□
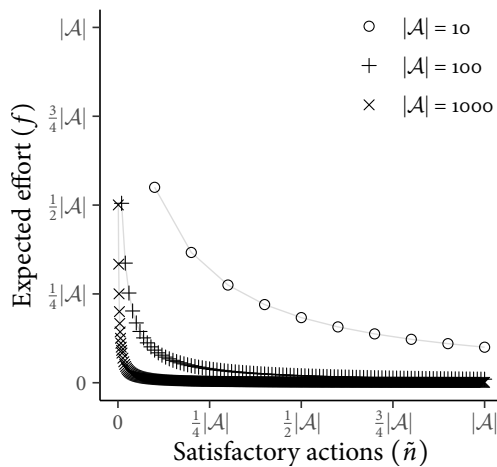


Figure 6.5: Expected number of action evaluations for the satisficing policy, $f(\tilde{n})$, as a function of the number of satisfactory actions in the action set, $\tilde{n}$, for different action set sizes $|\mathcal{A}|$. Both axes are scaled relative to the total number actions $|\mathcal{A}|$ in each case.

Figure 6.5 illustrates Lemma 1. The larger the action set $|\mathcal{A}|$, the larger the proportional reduction in effort for a given percentage of satisfactory actions $\tilde{n}$. For example, if 10% of available actions are deemed satisfactory, the reductions in effort for action set sizes 10, 100, and 1000 are 45.0%, 90.8%, and 99.0%, respectively. This indicates that satisficing could be especially useful for domains with large action sets.

Furthermore, the relationship between effort and $\tilde{n}$ is non-linear. The decrease in effort obtained by adding another action to set of satisfactory actions is large for small $\tilde{n}$ (that is, when not many actions are deemed satisfactory yet) but becomes negligible for large $\tilde{n}$. More specifically, if exactly one action is deemed satisfactory (that is, $\tilde{n} = 1$), the effort is $\frac{|\mathcal{A}|+1}{2}$, which for large $|\mathcal{A}|$, is almost half of the effort that is required by the greedy policy. For $\tilde{n} > 1$, the

effort decreases further as the number of satisfactory actions increases, albeit at a quadratically decreasing rate (Equation 6.2 in Lemma 1).

We now use Lemma 1 to provide conditions on when the $\xi$-satisficing policy is a Pareto improvement on the greedy policy.

> **Theorem 2** (Pareto improvement on greedy policy). Let $q_{max}$ denote the highest and $q_2$ denote the second-highest $q$-value obtainable in a state $s$ with at least two available actions (that is, $|\mathcal{A}(s)| > 1$). Then, for $\xi \in (q_2, q_{max}]$, the $\xi$-satisficing policy is a Pareto improvement on the greedy policy.

*Proof of Theorem 2.* In any given state $s$ and for a given action-value function $q$, the greedy policy requires effort of $e(\pi_*, s) = |\mathcal{A}(s)|$ and yields a quality of $u(\pi_*, s) = \max_a q(s, a)$. We will show that the $\xi$-satisficing policy for $\xi \in (q_2, q_{max}]$ yields the same quality as the greedy policy but has lower effort.

*Quality.* The quality of the satisficing policy is given by the expected value of all satisfactory actions, that is, $u(\tilde{\pi}_\xi, s) = \mathbb{E}[q(a|s)|q(s, a) \geq \xi]$. Yet for $\xi \in (q_2, q_{max}]$ only actions with value $q_{max}$ are satisfactory and thus

$$u(\tilde{\pi}_\xi, s|\xi \in (q_2, q_{max})) = \mathbb{E}[q(a|s)|q(s, a) = q_{max}] = q_{max},$$

which is equivalent to the quality obtained by the greedy policy.

*Effort.* For $\xi \in (q_2, q_{max}]$ there is at least one action in $\mathcal{A}(s)$ that is satisfactory, that is, $\tilde{n} \geq 1$. Therefore, for $\xi \in (q_2, q_{max}]$ and $|\mathcal{A}(s)| > 1$, and using Lemma 1,

$$f(\tilde{\pi}(\xi), s) \leq \frac{|\mathcal{A}(s)| + 1}{2} < |\mathcal{A}(s)| = f(\pi_*, s),$$

which was to be shown.   □

## 6.4   FROM ONE-SHOT TO SEQUENTIAL DECISION MAKING.

In the previous section I analyzed, in the context of a single decision in an MDP, how effort and quality of the $\xi$-satisficing policy vary as a function of the aspiration level $\xi$. In the following sections, I extend the analysis of the satisficing policy to entire MDPs and address the problem of how a satisficing agent can come up with suitable aspiration levels in the first place.

I develop three *aspiration adaption rules* to set aspiration levels dynamically and study how a satisficing agent using these rules trades off quality and effort across the entire decision-making process. Specifically, quality of a policy $\pi$ will be measured by the *expected episode return*, which is given by the value of the starting state, denoted by $v_\pi(s_0)$.[12] The *expected episode effort* is defined as the expected sum of individual efforts across all decisions if the agent follows policy

[12] Recall that at the beginning of this chapter we made the convenient assumption that there is only one starting state $s_0$. Without this assumption, the expected episode return of a policy $\pi$ would be given by $\mathbb{E}_{s_0 \sim \rho_0}[v_\pi(s_0)]$.

$\pi$. It is denoted by

$$F(\pi) = \mathbb{E}_{a_t \sim \pi(s_t), t \geq 0}\left[\sum_{t=0}^{\infty} f(\pi, s_t)\right], \tag{6.3}$$

where $f(\pi, s_t)$ is the expected effort required in state $s_t$ (see Definition 1).

## 6.5 ASPIRATION TRACKING

Here I present an aspiration adaption rule called *aspiration tracking*. It is directly informed by Theorem 2, which states that if the aspiration level can be set just below or equal to the maximum $q$-value in the current state, the satisficing policy requires less expected effort but yields optimal quality. Consequently, if an agent could set the policy aspiration to the maximum $q$-value at every decision stage of a sequential decision making problem, the agent would follow an optimal policy that requires less effort than the greedy policy.

The problem, of course, is that the maximum $q$-value in a given state is in general not known beforehand. Aspiration tracking is a simple aspiration adaption rule that estimates the maximum $q$-value in any given state based on the aspiration level used in the previously encountered state and the reward obtained in the current transition.

---

**Definition 4** (Aspiration tracking)**.** The *aspiration tracking* update rule with respect to a value function $q$ is defined by the recursive function

$$\xi_t = \frac{\xi_{t-1} - r_{t-1}}{\gamma}, \text{ for } t > 0, \tag{6.4}$$

and initial aspiration level $\xi_0 = v(s_0)$.

---

The initial aspiration level $\xi_0$ can be interpreted as the cumulative return the agent aspires to obtain across the entire episode. Intuitively speaking, the aspiration tracking rule tracks how much of the initially aspired return is already collected, or equivalently, how much return is yet to be obtained to achieve the goal set out in the beginning. This interpretation becomes apparent in the following toy example, which shows aspiration tracking applied to a toy MDP.

▷ EXAMPLE 1: ASPIRATION TRACKING IN A DETERMINISTIC MDP. Consider the episodic, deterministic, and undiscounted MDP, shown in Figure 6.6.



$r(s, a) = -1$ for all states $s$ and actions $a$

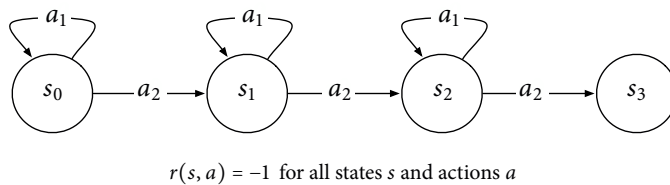Figure 6.6: Episodic MDP with four states $\mathcal{S} = \{s_0, s_1, s_2, s_3\}$ and two actions $\mathcal{A} = \{a_1, a_2\}$ available in every non-terminal state. The state $s_0$ is the starting state and $s_3$ is the only terminal state. A reward of $r(s, a) = -1$ is received for all state-action pairs $(s, a) \in \mathcal{S} \times \mathcal{A}$. The temporal discount factor is given by $\gamma = 1$.

In what follows, we will walk through the decisions made by a satisficing agent that uses aspiration tracking to set the aspiration level in every time step of the episode. We assume that the agent has access to the optimal value function $q_*$, shown in Figure 6.7. The initial aspiration level is set to $\xi_0 = v_*(s_0) = -3$. Because the discount factor is $\gamma = 1$, the aspiration tracking update rule simplifies to $\xi_t = \xi_{t-1} - r_{t-1}$.

The agent observes the initial state $s_0$ and starts considering actions in random order until a satisfactory action is found. The two actions in $s_0$ have action values $q_*(s_0, a_1) = -4$ (which is lower than the aspiration level $\xi_0$) and $q_*(s_0, a_2) = -3$ (which is equal to the aspiration level $\xi_0$). That is, only $a_2$ is satisfactory. Regardless of the order in which actions are considered, the agent eventually selects action $a_2$. The agent receives a reward of $r_0 = -1$ and observes the new state of the environment, $s_1$.

In time step $t = 1$, the aspiration level is updated to $\xi_1 = \xi_0 - r_0 = -3 - (-1) = -2$. Just as in the previous time step, only action $a_2$ is satisfactory (see Figure 6.7) and thus eventually selected by the agent. In time step $t = 2$, the aspiration level is set to $\xi_2 = \xi_1 - r_1 = -2 - (-1) = -1$. The agent thus selects action $a_2$ yet again and arrives at the terminal state $s_3$.

| $q_*$ | $s_0$ | $s_1$ | $s_2$ |
|-------|-------|-------|-------|
| $a_1$ | −4 | −3 | −2 |
| $a_2$ | −3 | −2 | −1 |

Figure 6.7: Optimal value function $q_*(s, a)$ for the toy MDP shown in Figure 6.6.



Figure 6.8: Episode summary of a satisficing agent in the toy MDP. For each decision stage $t = 0, 1, 2,$ and $3$, the figure shows the state of the environment ($s$), the aspiration level as determined by aspiration tracking ($\xi_t$), the maximum $q_*$-value in the current state ($\max_a q_*(a|s_t)$), and the action selected by the agent ($a$).

Figure 6.8 summarizes the episode just described. The satisficing agent always selected action $a_2$ and thus followed an optimal policy. Of course, an optimal policy would also be obtained by following the greedy policy. However, the greedy policy requires more effort than the satisficing policy to compute the optimal policy, as described next

In any given state $s$, the greedy policy needs to compute both action values; its expected effort for a single decision is thus $f(\pi_*, s) = 2$. The satisficing policy requires an effort of 1 if it first considers $a_2$, and an effort of 2 if it first considers $a_1$ (and thus has to consider $a_2$ as well). The expected effort required by the satisficing policy is thus $f(\tilde{\pi}, s) = 0.5 \times 1 + 0.5 \times 2 = 1.5$. Because in both cases the agent makes three such decisions, the expected episode effort for the greedy policy is given by $F(\pi_*) = 3 \times 2 = 6$ and the expected episode effort for the

satisficing policy is given by $F(\tilde{\pi}) = 3 \times 1.5 = 4.5$. In this example, satisficing yields a 25% reduction of expected effort. Notice that the reduction in effort would be larger if the MDP had a larger action space.[13]

In short, using aspiration tracking in every time step yielded the same (optimal) quality as the greedy policy but required strictly less expected effort. Put differently, satisficing was a Pareto improvement on the greedy policy *across the entire sequential decision process*.

In this particular example, the aspiration tracking rule is indeed able to correctly estimate the maximum $q$-value in every step,[14] which leads to a Pareto-improvement on the greedy policy in every time step (via Theorem 2), and thus to the long-term Pareto improvement just described.

In the following section, I will provide conditions under which this long-term Pareto improvement is guaranteed to hold. I will also provide a different toy example in which these conditions are not met and where, consequently, aspiration tracking does not lead to such strong results.

## 6.6 LONG-TERM PARETO IMPROVEMENT BY ASPIRATION TRACKING.

Let $\mathcal{S}^*$ denote the set of *relevant* states, that is, all non-terminal states that have a positive visitation probability under any optimal policy. Furthermore, let

$$\mu := \min_{s \in \mathcal{S}^*} |\mathcal{A}(s)|$$

denote the minimum size of all state-specific action sets across all relevant states.

> **Theorem 3** (Long-term Pareto improvement). Consider a deterministic MDP and assume that the optimal action-value function $q_*$ is known. Let $\pi_*$ denote the greedy policy with respect to $q_*$. Furthermore, let $\tilde{\pi}$ denote a satisficing policy with respect to $q_*$ that uses aspiration tracking. Then,
>
> A. the satisficing policy $\tilde{\pi}$ is an optimal policy, and
>
> B. the total expected effort of the satisficing policy can be upper-bounded as follows:
> $$F(\tilde{\pi}) \leq \frac{\mu + 1}{2\mu} F(\pi_*). \tag{6.5}$$
>
> Equality in Eq. 6.5 is attained if and only if all relevant state-dependent action sets have the same size ($|\mathcal{A}(s)| = \mu$ for all $s \in \mathcal{S}^*$) and there is exactly one value-maximizing action in every relevant state (that is, $|\mathrm{argmax}_{a \in \mathcal{A}(s)} q(a|s)| = 1$ for all $s \in \mathcal{S}^*$).

*Proof of Theorem 3.* We first prove that the aspiration level in any time step $t$ is equal to the maximum $q_*$-value in state $s_t$. That is, we aim to show that

---

[13] For example, consider the toy MDP shown in Figure 6.6, but with an enlarged action set that adds 98 "clones" of action $a_2$ to the existing actions $a_1$ and $a_2$. This would result in a total action set size of $|\mathcal{A}| = 100$ and thus an expected effort per decision of $f(\pi_*) = 100$ for the greedy policy, whereas a satisficing policy using aspiration tracking would require an expected effort of $f(\tilde{\pi}) = (99 \times 1 + 2 \times 1)/100 = 1.01$, corresponding to a 98.99% decrease in expected effort.

[14] Compare the third and fourth lines in Figure 6.8.

$$\xi_t = \max_a q_*(a|s_t) \tag{6.6}$$

for all $t$. For $t = 0$, Equation 6.6 is true by definition of the aspiration tracking adaption rule, that is, $\xi_0 = \max_a q_*(a|s_0)$. We now show that if Eq. 6.6 holds true for a time step $t \geq 0$, then it also holds for $t + 1$. In time step $t$, the satisficing agent selects a value-maximizing action $a_t \in \text{argmax}_a q_*(a|s_t)$, receives reward $r_t$, and observes the new state $s_{t+1}$. The new aspiration level in time step $t + 1$ is given by

$$
\begin{aligned}
\xi_{t+1} &= \frac{\xi_t - r_t}{\gamma} \\
&= \frac{\max_a q_*(a|s_t) - r_t}{\gamma} \\
&= \frac{q_*(a_t|s_t) - r_t}{\gamma} \qquad\qquad \text{Because } a_t \in \text{argmax}_a q_*(a|s_t). \\
&= \frac{\mathbb{E}[R_t + \gamma \max_{a'} q_*(S_{t+1}|a')|S_t = s_t, A_t = a_t] - r_t}{\gamma} \qquad \text{By the Bellman equation.} \\
&= \frac{r_t + \gamma \max_{a'} q_*(a'|s_{t+1}) - r_t}{\gamma} \qquad\qquad \text{By assumption of a deterministic transition model.} \\
&= \max_{a'} q_*(a'|s_{t+1}).
\end{aligned}
$$

By induction it follows that Eq. 6.6 holds true for all time steps $t$. It follows that the satisficing policy $\tilde{\pi}$ chooses a value-maximizing action in every time step, and is thus an optimal policy. This concludes what was to be shown for part A.

Now to part B. According to Lemma 1, the effort required by $\tilde{\pi}$ in time step $t$ is given by $e(\tilde{\pi}, s_t) = \frac{|\mathcal{A}(s_t)|+1}{\tilde{n}(s_t)+1}$, where $\tilde{n}(s_t) \geq 1$ is the number of satisfactory actions in state $s_t$. The number of satisfactory actions is equal to or greater than one in every state because there is always at least one value-maximizing action (according to Equation 6.6). We now compare the total efforts of the satisficing policy $\tilde{\pi}$ and the greedy policy $\pi_*$ with each other. Recall that the total effort of a policy $\pi$ is given by the expected sum of efforts across the entire episode, given by $F(\pi) = \mathbb{E}_{a_t \sim \pi(s_t)}[\sum_{t=0}^{\infty} e(\pi, s_t)]$, where the expectation is taken over possible trajectories under $\pi$. Note that both $\pi_*$ and $\tilde{\pi}$ are optimal policies. Therefore, the distributions of trajectories produced by these policies are the same (using the minor technical assumption that if there is more than one value-maximizing action in a state, the policy is to select uniformly randomly among all value-maximizing ac-

tions). The total effort of the satisficing policy is then given by

$$
\begin{aligned}
F(\tilde{\pi}) &= \mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} e(\tilde{\pi}, s_t)\Big] \\
&= \mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} \frac{|\mathcal{A}(s_t)| + 1}{\tilde{n}(s_t) + 1}\Big] \\
&\leq \mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} \frac{|\mathcal{A}(s_t)| + 1}{2}\Big] \\
&= \mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} \frac{|\mathcal{A}(s_t)|(1 + \frac{1}{|\mathcal{A}(s_t)|})}{2}\Big] \\
&\leq \mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} \frac{|\mathcal{A}(s_t)|(1 + \frac{1}{\mu})}{2}\Big] \\
&= \mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} |\mathcal{A}(s_t)|\frac{\mu + 1}{2\mu}\Big] \\
&= \frac{\mu + 1}{2\mu}\mathbb{E}_{a_t \sim \tilde{\pi}(s_t)}\Big[\sum_{t=0}^{\infty} |\mathcal{A}(s_t)|\Big] \\
&= \frac{\mu + 1}{2\mu}\mathbb{E}_{a_t \sim \pi_*(s_t)}\Big[\sum_{t=0}^{\infty} |\mathcal{A}(s_t)|\Big] \\
&= \frac{\mu + 1}{2\mu}F(\pi_*),
\end{aligned}
$$

Using that $\tilde{n}(s_t) \geq 1$ for all $s_t \in \mathcal{S}^*$.

Using that $\mu = \min_{s_t \in \mathcal{S}^*} |\mathcal{A}(s)|$.

Using that both $\tilde{\pi}$ and $\pi_*$ are optimal policies and their expected trajectories therefore the same.

which was to be shown. Equality holds only if $\tilde{n}(s_t) = 1$ and $|\mathcal{A}(s_t)| = \mu$ (that is, all state-specific action sets have the same size) for all $s_t \in \mathcal{S}^*$. $\square$

Theorem 3 shows that an aspiration-tracking satisficing policy with respect to an optimal value function is an optimal policy. Moreover, in any non-trivial[15] MDP, the satisficing policy requires strictly less expected episode effort than the greedy policy, that is, $F(\tilde{\pi}) < F(\pi_*)$. In particular, if no relevant state is trivial, then $\mu \geq 2$, and consequently, $F(\tilde{\pi}) \leq \frac{3}{4}F(\pi_*)$. The relative advantage of the satisficing policy's effort increases as the minimum size of state-dependent action sets, $\mu$, increases.

The strong results of Theorem 3 were obtained under the following two main assumptions: first, the MDP is deterministic; and second, the optimal value function is known. These assumptions were met in Example 1, which was discussed in the previous section. In what follows, we consider situations in which at least one of these assumptions is not met. We first provide an example of a stochastic MDP, where aspiration tracking fails to produce an optimal policy. Later in Section 6.8.2 we then consider a case when the optimal value function is approximated by a neural network.

[15] A state is considered trivial if its action set contains only one action and there is thus no decision to be made. A non-trivial MDP is an MDP with at least one non-trivial relevant state.

▸ EXAMPLE 2: FAILURE OF ASPIRATION TRACKING IN STOCHASTIC MDPs. Consider the stochastic MDP shown in Figure 6.9. Similar to what we did in Example 1, we compare quality and effort of the greedy policy and a satisficing policy using aspiration tracking. Both policies are defined with respect to the
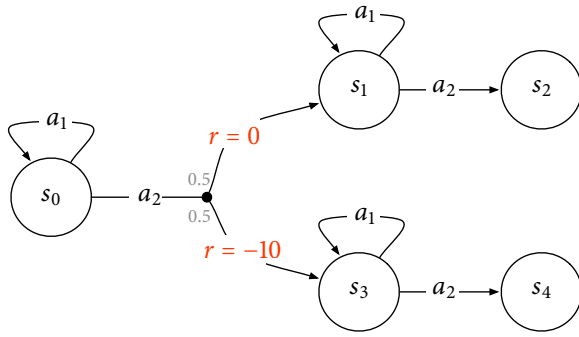
optimal action-value function $q_*$, which is shown in Figure 6.10 and assumed to be known to the agent.

The optimal policy is to select action $a_2$ in every state. Depending on the outcome of the coin-flip after taking $a_2$ in state $s_0$, the agent either follows the trajectory $s_0 \rightarrow s_1 \rightarrow s_2$ and receives a return of $0 - 1 = -1$ or follows the trajectory $s_0 \rightarrow s_3 \rightarrow s_4$ and receives a return of $-10 - 1 = -11$. The expected return of an optimal policy is therefore $v_*(s_0) = -6$. Regardless of the outcome of the coin-flip, the greedy agent makes two decisions among two actions each; the episode effort of the greedy agent is therefore given by $F(\pi_*) = 2 + 2 = 4$.

Now to the satisficing agent that uses aspiration tracking. The initial aspiration level is set to $\xi_0 = v_*(s_0) = -6$. The satisficing agent therefore starts off just like the greedily behaving agent and selects the optimal action $a_2$ (because $a_1$ is not satisfactory).

We next consider both outcomes of the coin-flip in turn. If the agent transitions to state $s_3$ and thus receives a reward of $r_0 = -10$, the new aspiration level is given by $\xi_1 = -6 - (-10) = 4$. The maximum action-value in $s_3$ is $q_*(s_3, a_2) = -1$, which is smaller than the aspiration level $\xi_1$. The satisficing policy therefore falls back to the greedy policy, selecting the optimal action $a_2$. In this case, the aspiration-tracking satisficing agent still follows an optimal policy and requires less expected episode effort than the greedy agent.[16]

However, if the coin-flip makes the agent transition to $s_1$, the agent receives a reward of $r_0 = 0$. In this case the new aspiration level is $\xi_1 = -6 - 0 = -6$. This aspiration level is lower than both action-values in state $s_1$ (see middle column in Figure 6.10). The satisficing agent therefore selects whichever action is considered first. If that first action happens to be $a_1$, the satisficing agent no longer follows an optimal policy and receives a lower return than the greedy policy. To make things worse, the total episode effort could become higher than the one required by the greedy policy because the agent ends up having to make more decisions.

Example 2 showed how satisficing with aspiration tracking can result in sub-optimal behavior in stochastic MDPs. In the following section I focus on the

| $q_*$ | $s_0$ | $s_1$ | $s_3$ |
|-------|-------|-------|-------|
| $a_1$ | −7 | −2 | −2 |
| $a_2$ | −6 | −1 | −1 |

Figure 6.10: Optimal value function $q_*(s, a)$ for the toy MDP shown in Figure 6.9.

[16] Specifically, the expected episode effort of the satisficing policy would be given by $F(\tilde{\pi}) = f(\tilde{\pi}, s_0) + f(\tilde{\pi}, s_1) = 1.5 + 2 = 3.5 < 4 = F(\pi_*)$.

other important assumption in Theorem 3: the availability of an exact optimal value function.

## 6.7 VALUE TRACKING

The MDPs used to model many real-world applications have too many states or actions, or both, to learn and maintain a tabular value function. The standard solution to this problem is to approximate the action-value function using a parametrized mathematical function $\hat{q}_{\theta}(s, a)$, where $\boldsymbol{\theta}$ is a vector of parameters. Learning then consists of finding a set of parameters such that $\hat{q}_{\theta}(s, a)$ is as close as possible to $q_*(s, a)$ for all state-action pairs.

Such an approximation is hardly ever perfect—and it usually does not have to be. Small approximation errors do not pose a problem for the greedy policy as long as the approximated value of an optimal action is still larger than the approximated values of all non-optimal actions (and the greedy policy therefore still correctly identifies a value-maximizing action). For satisficing with aspiration tracking, however, a small approximation error in the evaluation of the starting state can negatively affect all future decisions. This is because every later aspiration level is a function of the initial aspiration level, which is set to the value of the initial state, $\xi_0 = v(s_0)$.

For instance, in Example 1 in Section 6.5, if the optimal value of the initial state $s_0$ was estimated to be, say, $\xi_0 = \hat{v}(s_0) = -2.9$ (the correct value is $-3$), then the aspiration levels in future states ($\xi_1 = -1.9$ and $\xi_2 = -0.9$) would be higher than the maximum $q$-values in the respective states ($q_{\max}(s_1) = -2$ and $q_{\max}(s_2) = -1$). As a result, the satisficing policy would fall back to the greedy policy and thus lose the improvements that would be guaranteed by Theorem 3 if the initial value estimate was correct. A severe under-estimation is arguably even worse: if the initial value was estimated to be lower than any true action value in the initial state (for example, $\hat{v}(s_0) = -4.1$), the satisficing policy would be equivalent to a random policy in the first step (and possibly in later steps, as well).

To break the dependence of all aspiration levels on the value estimate of the initial state, I propose an alternative aspiration adaption rule called *value tracking*.

---

**Definition 5** (Value tracking). The *value tracking* aspiration adaption rule with respect to a value function $q$ is defined by the recursive function

$$\xi_t = q(s_{t-1}, a_{t-1}) - r_{t-1}, \text{ for } t > 0, \tag{6.7}$$

and initial aspiration level $\xi_0 = v(s_0)$.

---

In value tracking, the new aspiration level is simply given by the difference between the value of the action chosen in the previous state, given by $q(s_{t-1}, a_{t-1})$, and the reward obtained in the current decision, $r_{t-1}$. Just like aspiration track-

ing, value tracking provides an unbiased estimate of the maximum $q$-value of the current state based only on information from the previous time step. Unlike aspiration tracking, however, the value-tracking update rule "re-bases" this estimate on the value estimate from the previous time step, rather than estimating it based on the value estimate of the initial state value (and the stream of rewards observed afterwards). In doing so, value tracking makes the bet that short-term approximation errors are smaller than long-term, accumulated approximation errors.

We will compare the aspiration tracking and value tracking update rules in the experiments presented in the following sections. Before doing so, however, we will introduce yet another aspiration adaption rule.

This third rule is based on the observation that a severe over-estimation of the maximum $q$-value in the current state has a qualitatively different effect compared to a severe under-estimation of that same value. Specifically, if the aspiration level is strictly larger than $q_{max}$ (no matter by how much), satisficing simply falls back to the greedy policy and thus yields optimal quality and requires the same effort as the greedy policy across the entire episode. However, for aspiration levels that strongly under-estimate $q_{max}$, satisficing can result in low-quality decision-making (in the worst case, uniformly random behavior) and thus eventually even *more* effort across the entire episode than the greedy policy.[17] In other words, cost is an asymmetric function of estimation error.

The *valved value tracking* update rule provides one (rather extreme) way of taking this cost asymmetry into account.

> **Definition 6** (Valved value tracking). The *valved value tracking* aspiration adaption rule is defined recursively by
>
> $$\xi_t = \max(q(s_{t-1}, a_{t-1}) - r_{t-1}, \xi_{t-1}), \text{ for } t > 0, \tag{6.8}$$
>
> and initial aspiration level $\xi_0 = v(s_0)$.

Valved value tracking uses value tracking (Equation 6.7) only if the updated value is higher than the current aspiration level, and sticks to the current estimate otherwise. Unlike previous aspiration level update rules, valved value tracking does not, in general, provide an unbiased estimate of the maximum $q$-value in the current state. Instead, valved value tracking is biased toward overestimating the maximum $q$-value in the current state. By consequence, in the case of estimation error, valved value tracking is more likely to be rescued by the "safety net" that is the greedy policy rather than falling all the way down and ending up as a random policy.

## 6.8   EXPERIMENTS

The objective of these experiments is two-fold. First, we aim to provide experimental illustration for the theoretical results developed in the previous sections.

[17] Low aspiration levels can result in higher *total* episode effort because even though the effort in a single decision decreases with decreasing aspiration levels (Section 6.3.2), the total number of decisions might increase by following a non-optimal policy, as discussed in Example 2 in Section 6.6.

Second, we assess the performance of various aspiration adaption rules when the assumptions of Theorem 3 are not met.

### 6.8.1 *Macro-action gridworld: tabular value function and large action set*

Here I compare the satisficing policy with aspiration tracking to the greedy policy in a gridworld domain with macro actions. *Macro actions* are fixed sequences of primitive (or *atomic*) actions.[18] For example, the macro action "drink coffee" could be divided into the sequence of more primitive (yet still quite high-level) actions "move hand to coffee cup", "grab cup of coffee", "move hand to mouth", and "make drinking motion".

The use of macro actions allows the agent to abstract and focus on the important parts of the decision-making process (e.g., "getting sufficient caffeine" rather than "optimally grabbing a cup of coffee"), and it usually results in fewer decisions to be made (and thus potentially reduced effort). The downside, however, is that the action set of macro actions is usually much larger than the action set of primitive actions. The number of possible macro actions is an exponential function of the length of the number of atomic actions used to compose the macro action.[19] Even moderately sized atomic action sets and macro-action lengths can result in huge action set sizes.

▸ THE DOMAIN: GRIDWORLD WITH MACRO ACTIONS. Figure 6.11 shows an example state of an $8 \times 8$ gridworld environment with 4-step macro actions. The agent starts each episode in the bottom-left corner. The only terminal state is the "gold state" in the top-right corner. Each macro action consists of a chain of four primitive actions, which are given by the four cardinal directions. The action set size is $|\mathcal{A}(s)| = 4^4 = 256$. The figure shows six macro actions. The agent receives a negative reward of $r = -4$ for every decision. The agent receives an additional reward of $r = 16$ upon arriving on the terminal state (where the gold is). The optimal episode return is $v_*(s_0) = 0$.
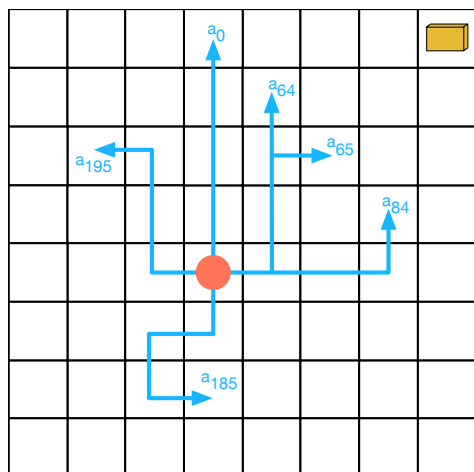


[18] Amarel (1968)

[19] For $k$ atomic actions, the number of different $n$-step macro actions is given by $k^n$. For example, for only $k = 2$ atomic actions "←" and "→", and $n = 3$ steps, the following $2^3 = 8$ macro actions are available:
$\{\leftarrow, \leftarrow, \leftarrow\}$,
$\{\rightarrow, \leftarrow, \leftarrow\}$,
$\{\leftarrow, \rightarrow, \leftarrow\}$,
$\{\leftarrow, \leftarrow, \rightarrow\}$,
$\{\rightarrow, \rightarrow, \leftarrow\}$,
$\{\rightarrow, \leftarrow, \rightarrow\}$,
$\{\leftarrow, \rightarrow, \rightarrow\}$, and
$\{\rightarrow, \rightarrow, \rightarrow\}$.

Figure 6.11: $8 \times 8$ gridworld environment with macro actions.

▶ EXPERIMENTAL SETUP. I measured episode effort and return of various value-based policies. All policies were given access to the tabular optimal value function $q_*$. Specifically, we compared the satisficing policy using aspiration tracking (and $\xi_0 = v_*(s_0) = 0$) to the $\varepsilon$-greedy policy for different values of the exploration parameter $\varepsilon$, including the random policy ($\varepsilon = 1$) and the greedy policy ($\varepsilon = 0$). I do not show results for value tracking and valved value tracking because both rules are equivalent to aspiration tracking when the optimal value function is given and the MDP is deterministic.

▶ RESULTS. Figure 6.12 shows episode return as a function of the total episode effort (number of considered actions), averaged across 1000 episodes. The greedy policy required an effort of 1000 action considerations to reach the optimal return of 0. The satisficing policy considered, on average, 70 actions to reach the same optimal return.
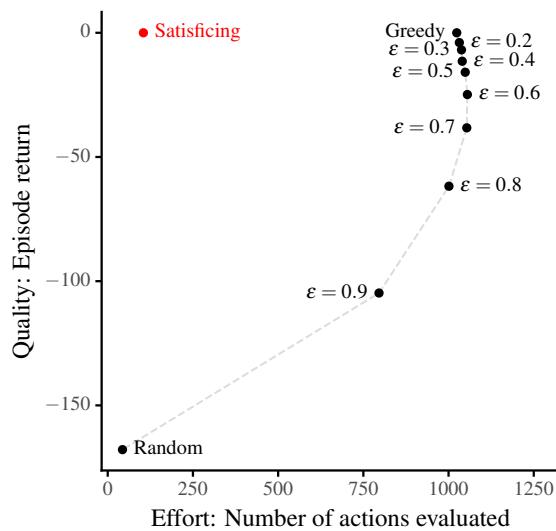


Figure 6.12: Effort-quality trade-offs for $\xi$-satisficing with aspiration tracking and $\varepsilon$-greedy policies.

According to Theorem 3, the expected episode effort required by the satisficing policy in this environment should be at most around half the effort required by the greedy policy, or $F(\tilde{\pi}) \leq \frac{256+1}{2\times256} F(\pi_*) \approx 0.5 F(\pi_*)$. Yet the results show that the satisficing policy did not even require a tenth of the greedy policy's effort. The upper bound is not tight in this case because there are $\tilde{n} = 16$ optimal (and thus satisfactory) actions available in every state. The bound in Theorem 3 assumes the "worst case" (in terms of expected improvement on the greedy policy) that every state has only $\tilde{n} = 1$ optimal action.

### 6.8.2  *Lunar-Lander: Approximated value functions.*

In this subsection I analyze the satisficing policy in a domain where the state space is too large for a tabular value representation to be feasible. The action-value function is instead approximated by an artificial deep neural network. I

compare different aspiration level update rules to each other and to the greedy policy.

▶ DOMAIN: LUNAR LANDER. Figure 6.13 shows one frame of the Lunar-Lander environment. The agent controls the purple space ship and tries to land it safely on the moon, on the landing pad between the two yellow flagpoles. For a landing to be safe, the space ship should not be too fast and reasonably leveled (so as to land on the space ship's feet rather than on its "back"). An episode ends if the space ship lands safely and comes to rest or if the space ship crashes.

The agent can control the space ship using four discrete actions: "fire main engine" (resulting in upwards moving force), "left orientation engine", "right orientation engine", and "do nothing". To guide its actions, the agent observes an 8-dimensional continuous state vector, which contains information about the position, velocity and orientation of the space ship. The initial state varies across episodes.

The agent receives small positive rewards for getting closer to the landing pad[20] and, conversely, receives negative rewards if moving away from (or landing outside of) the landing pad. An additional reward of +100 or −100 is obtained if the agent lands safely or crashes, respectively. Fuel is infinite but the agent receives a negative reward of −0.3 for every frame in which the main engine (upwards force) is used.

▶ EXPERIMENTAL SETUP. I represented the value function using a fully-connected feed-forward neural network with one hidden layer consisting of 64 neurons. The input of that network was the 8-dimensional state representation and the output layer consisted of 4 neurons, one for each of the four actions. The network parameters were trained using the DQN algorithm[21] for $600,000$ steps, using double Q-learning[22] and learning parameters as described in Table 6.1.



Figure 6.13: Lunar-Lander environment. We used the *LunarLander-v2* environment available in *OpenAI gym* (Brockman et al., 2016).

[20] The agent receives a total reward between 100 and 140 for moving relatively directly from the starting position to the landing pad.

[21] Mnih et al. (2015)

[22] Van Hasselt et al. (2016)

Table 6.1: Learning parameters for the DQN algorithm.

| Description | Parameter | Value |
|---|---|---|
| Learning rate | $\alpha$ | 0.001 |
| Exploration (linearly decreasing) | $\varepsilon_{\text{init}} \to \varepsilon_{\text{final}}$ | $1.00 \to 0.05$ |
| Discount factor | $\gamma$ | 0.99 |
| Batch size | - | 32 |
| Replay buffer size | - | $100,000$ |

The so-trained approximated value function is denoted by $\hat{q}_{\text{DQN}}(s, a)$. I evaluated three satisficing policies with respect to $\hat{q}_{\text{DQN}}(s, a)$. The satisficing policies differed only in the way aspiration levels are set. I used the aspiration adaption rules aspiration tracking, value tracking, and valved value tracking. The baseline policy was the greedy policy with respect to the approximated value function $\hat{q}_{\text{DQN}}(s, a)$.

▶ RESULTS. Figure 6.14 shows episode return as a function of episode effort averaged across 100 episodes. On average, satisficing with valved value tracking
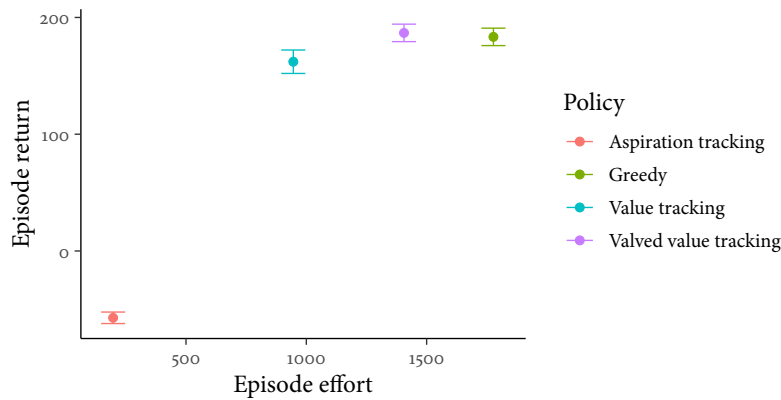
Figure 6.14: Results for the Lunar Lander domain. Effort-quality trade-offs for the greedy policy and the $\xi$-satisficing policy with varying aspiration adaption rules. The error bars show standard error of the mean.

showed approximately the same return[23] as the greedy policy while requiring around 79.1% of the greedy policy's effort.

The pure value tracking update rule required even less effort (53.2% of the greedy policy's effort) but also yielded slightly lower quality than the valved value tracking policy. Both effects are consistent with the intuition that led to the development of valved value tracking in the first place. Value tracking sometimes sets the aspiration level too low, which leads to smaller expected effort but can also deteriorate quality. Valved value tracking on the other hand tends to over-estimate the maximum q-value in the current state and thus sometimes leads to value-maximizing behavior, which leads to higher quality but also more effort.

Aspiration tracking failed to produce a useful policy and achieved an average return of around $-57.2$ (that is, the space ship was crashing more often than not). One possible explanation for the bad performance of aspiration tracking could be that the initial aspiration level, which was given by the approximated value of the starting state, was generally too low. In aspiration tracking all aspiration levels directly depend on the initial aspiration. If the initial aspiration level is too low, the aspiration levels in all further states are likely to underestimate the respective highest $q$-values too, resulting in what is essentially random behavior and a likely crash.[24]

## 6.9    RELATED LITERATURE

▸ EXISTING MODELS OF SATISFICING BEHAVIOR. Most closely related to this chapter's work is an article by Russo and Van Roy (2018), which discusses a satisficing strategy in multi-armed non-contextual bandit problems[25] with many arms (actions), including the infinite-armed bandit. They establish information-theoretic regret bounds[26] for the satisficing policy. Their satisficing strategy settles for the first action that is "within $\varepsilon$ of the optimum", that is, $\varepsilon$ is similar to the

[23] The average return of valved value tracking (186.8) was slightly higher than the average return of the greedy policy (183.4). However, the standard error bands overlap so that no conclusion is drawn from this small difference.

[24] See also the toy example given in Section 6.7, where an initial aspiration level that is too low can lead to random behavior across the entire episode.

[25] A non-contextual bandit can be thought of as an MDP with only one state and every action available in that state leading back to the same state.

[26] The regret of a policy $\pi$ in bandit problems is the performance gap between the optimal policy and the policy $\pi$.

aspiration level $\xi$ used here. One main difference to our work is that they study satisficing for a given $\varepsilon$ whereas our main focus is on determining a suitable aspiration level in each time step.

Ortega et al. (2015) derive a rejection sampling algorithm can be seen as a stochastic alternative to the satisficing algorithm studied here. The algorithm emerges as a solution to the information-theoretic bounded rationality problem (also discussed in Section 2.4) of maximizing the free energy potential of a policy $\pi$ in a single-shot decision-making problem, defined as

$$ FE(\pi) := \sum_a \pi(a)U(a) - \frac{1}{\alpha} \sum_a \pi(a) \log \frac{\pi(a)}{Q(a)}, \qquad (6.9) $$

where $U$ is a utility function, $Q$ is a prior policy, and $\alpha \in \mathbb{R}$ is the inverse temperature which regulates the trade-off between utility and information cost. The regularization term can be interpreted as the cost of information needed to search for a policy that is different from the prior policy $Q$.

The rejection-sampling algorithm samples choice possibilities in random order and selects the first action that is not rejected, where the rejection probability is a function of the action's value relative to the maximum value attainable (see Algorithm 1 in Ortega et al., 2015). The rejection probability (which loosely corresponds to the aspiration level in our work) is thus calculated from the maximum value attainable, which has to be provided to the algorithm as an input. Our work, on the other hand, focuses on coming up with a suitable aspiration level in the first place, when the maximum value in the current state is not available.

Simon's satisficing strategy has also been formalized in various settings within economics and game theory.[27] Yet most of these settings consider either one-shot choice decisions, simple two-player zero-sum games such as the "Prisoner's dilemma", or two-armed bandits.[28] Furthermore, some of this work is descriptive in that it aims to describe human choice behavior as closely as possible. This is useful for marketing science, where satisficing has been found to explain consumer choice behavior.[29]

▶ DEALING WITH LARGE DISCRETE ACTION SPACES IN REINFORCEMENT LEARNING. Large discrete action spaces have been identified previously as a problem for deep reinforcement learning problems.[30] Existing approaches to tackle this problem include to learn action representations that can generalize across actions.[31] Dulac-Arnold et al. (2015) learn a proposal network that proposes a hypothetical target action in a continuous action embedding, which is then mapped to a real action from $\mathcal{A}(s)$ that is close or closest to the hypothetical target action. These approaches are somewhat orthogonal to—and could be combined with—the $\xi$-satisficing policy described here.

[27] Winter (1971), Radner (1975), Selten (1998), Bendor et al. (2009), and Caplin et al. (2011)

[28] Bendor et al. (2009)

[29] Stüttgen et al. (2012)

[30] Dulac-Arnold et al. (2015)

[31] Chandak et al. (2019) and Tennenholtz and Mannor (2019)

In this chapter we developed and studied strategies to dynamically set aspiration levels in Markov decision processes. An initial analysis of satisficing in a single decision of the Markov decision process led us to the following guiding principle: try to set the aspiration level to the maximum q-value in the current state because the satisficing policy then yields a Pareto improvement on the greedy policy. The maximum q-value in a decision stage is generally not known beforehand. We therefore developed simple aspiration adaption rules that estimate the maximum q-value in the current state based on information available from the previous decision stage. We found that satisficing agents equipped with these aspiration adaption rules can follow optimal or near-optimal policies while requiring considerably less effort than agents that use a greedy policy.

Klein et al. (1995) provide experimental support that chess players generate (end evaluate) only relatively few actions (chess moves) before making a decision. Furthermore, they found that the action-generation process is unlikely to be uniformly random. Instead chess players use their experience and intuition to guide the action-generating process so that, on average, good or optimal moves are likely to be considered earlier than bad moves. In this chapter we made the simplifying assumption that the satisficing policy's action-generating process followed a uniformly random distribution. If, instead, the agent had access to an action proposal distribution that was likely to propose high-value actions earlier than low-value actions, the expected effort required by the satisficing policy could possibly be reduced even further.

Part IV

DISCUSSION & APPENDIX

## DISCUSSION OF CONTRIBUTIONS

In this dissertation I explored two approaches of how models of bounded rationality can inform reinforcement learning algorithms to be more resource-efficient.

▸ PART II: BOUNDEDLY RATIONAL FUNCTION APPROXIMATION. The goal in this part was to reduce the amount of training data required to learn a useful policy. In Chapter 5 I presented a novel reinforcement learning algorithm, called *iterative policy space expansion* (or IPSE), which adapts its policy approximation architecture to the amount and quality of data available. The high-level idea of this algorithm is as follows. At the beginning of the learning process, when resources are most limited, the algorithm learns in a strongly constrained policy space and relies on a model of bounded rationality to quickly learn an initial policy. This initial policy may not be optimal but it is good enough to allow the agent to efficiently explore its environment and to collect more and better data. With increasing learning progress, the algorithm then gradually expands the policy space and thus allows the agent to learn a more refined, data-driven policy.

The IPSE algorithm incorporates the general idea that learning in a sequence of progressively more difficult learning tasks allows the learner to gradually build more complex concepts and thus to accelerate the learning process. Contrary to existing algorithms that exploit this idea, the IPSE algorithm does not require the algorithm designer to explicitly construct a series of progressively more difficult tasks. Instead, the task difficulty is regulated intrinsically by a policy space that initially is strongly constrained and expands over time. This property makes the IPSE algorithm applicable to domains where creating a curriculum of increasingly difficult tasks is difficult or costly.

I studied a specific version of the IPSE algorithm that is centered around the equal-weighting heuristic. I evaluated this algorithm in the challenging domain of learning how to play Tetris. The IPSE algorithm showed a substantially higher learning rate than competing algorithms that did not use a boundedly rational model as building block for learning, including the state-of-the-art algorithm CBMPI.

Future work can build on the ideas presented in this chapter in a number of ways.

One direction is to extend these ideas from policy-based reinforcement learning to value-based reinforcement learning. Put differently, the same or similar regularization techniques can be used to constrain (and expand) the value-function space.

Another possible direction is to use a different model of bounded rationality as central building block. For instance, instead of first learning an equal-weighting model, the algorithm could first learn a non-compensatory[1] decision rule or a lexicographic-tallying model[2] before expanding the policy space.

A third direction is to extend these ideas to non-linear function approximation architectures. In its current form, the IPSE algorithm relies on the availability of powerful features that can be meaningfully combined using a linear function. In domains in which such features are not available, deep reinforcement learning algorithms can be used to discover features during the learning process. A boundedly rational non-linear function approximator could lead to faster feature discovery in reinforcement learning algorithms.

The IPSE algorithm was directly inspired by the results obtained in the first two chapters of this part, which were concerned with models of bounded rationality in supervised learning.[3] In addition, these first two chapters provided insights into how existing models of bounded rationality can be useful in machine learning algorithms and applications more generally, beyond their application in reinforcement learning.

In Chapter 3 I presented the most comprehensive empirical study about the predictive performance of simple regression models to date. Simple regression models routinely outperformed state-of-the-art regression models on small training sets. Averaged across all data sets, simple models showed lower predictive accuracy than their more complex counterparts on large training sets. However, even for larger training set sizes, there was almost always at least one simple model that rivaled the best-performing model. Put differently, an agent equipped with a collection of simple models can make excellent inferences if the agent is capable to select the right model for the present decision environment.[4]

In Chapter 4 I presented a regularization term, called *shrinkage toward equal weights* (or STEW), which has two useful properties in addition to its variance-reduction capabilities similar to other regularization techniques. First, it allows the user to interpolate between a simple, boundedly rational equal-weighting solution and a fully data-driven solution such as the least-squares estimator. Second, a STEW-regularized model can incorporate prior knowledge about feature directions in ways that other regularized linear models cannot. When knowledge about feature directions was available, linear regression with STEW regularization outperformed competing regularized linear models across various real-world data sets.

---

[1] See Section 2.2.2.

[2] Şimşek and Buckmann (2017)

[3] Previous work (Şimşek et al., 2016; Şimşek, 2020a) has exploited and transferred insights from the study of bounded rationality in supervised learning to reinforcement learning.

[4] Similar results were also obtained in Buckmann and Şimşek (2017) for the paired comparison task. These findings are also sympathetic to the idea of the mind as an *adaptive toolbox* (Gigerenzer and Selten, 2002; Rieskamp and Otto, 2006; Brighton, 2006).

▶ Part III: Boundedly rational action selection. The goal in this part was to reduce the computational resources required by the agent by means of reducing the number of actions that have to be considered and evaluated before a decision is made. I presented an approach based on Herbert Simon's satisficing strategy for decision making. The satisficing policy is to consider actions sequentially and to select the first action whose estimated long-term value is above some pre-defined aspiration level. By selecting a possibly sub-optimal action before all actions are considered and evaluated, the satisficing policy trades off action quality against computational effort.

The present work departs from existing literature by studying the satisficing strategy in the context of Markov decision processes, which is one of the most widely studied mathematical frameworks for sequential decision-making problems. The main challenge for a satisficing agent in this context is to determine a suitable aspiration level at every decision stage. I defined three simple rules to set aspiration levels dynamically and studied how a satisficing agent using these aspiration adaption rules trades off effort and quality across the entire sequential decision-making process. Under the strong conditions that the Markov decision process is deterministic and that the agent has access to an optimal value function, the satisficing policy provably yields optimal expected quality but requires provably less expected effort than the value-maximizing policy. The empirical results I presented indicate that the satisficing policy can be more resource-efficient than the greedy policy even if the optimal value function is only approximated.

This work is limited in that it studied the properties of the satisficing policy only after the actual learning process, that is, when an optimal value function (or an approximation thereof) is already available. However, the presented work laid the groundwork for exploring the satisficing strategy in the learning process itself. For instance, an agent can use the $\xi$-satisficing policy in combination with any of the proposed aspiration adaption rules as exploration policy in a temporal-difference learning algorithm. I believe that this could yield a useful approach to perform *directed exploration*.[5] Directed exploration methods use some form of exploration-specific knowledge to guide the exploration process, with the goal to explore promising actions more often than actions that are unlikely to have a positive impact. Provided a suitably chosen aspiration level, the satisficing policy is a directed exploration method in the sense that it explores among satisfying (and thus promising) actions only.

[5] Thrun and Möller (1991). Directed exploration is also sometimes called *structured exploration* (Gupta et al., 2018).

In the two main parts just presented, I used different types of boundedly rational models from the literature to address different resource limitations. One interesting direction for future work is to combine the different approaches presented in this dissertation. For example, an agent could use satisficing in combination with a boundedly rational value function approximation architecture to address both data and computational limitations.

Throughout this dissertation, I assumed that the agent was given hand-crafted,

pre-selected features (for instance, Tetris features such as *number of holes* or the *size* and number of rooms features in the Rent data set). Future work could study how models of bounded rationality can be used to come up with and select useful features in the first place.

On a more fundamental level, the study of bounded rationality in machine learning could be extended to the problem formulation itself, for instance, by reformulating the reinforcement learning problem as a satisficing problem: find any policy that achieves at least a certain amount of average reward. I believe that such relaxations could lead to faster discovery of "good enough" policies in complex real-world sequential decision-making environments.

# A

## APPENDIX A. CODE & IMPLEMENTATION DETAILS

---

▸ OPEN-SOURCED CODE. The GitHub repository `https://github.com/janmaltel/stew-tetris` contains a Python implementation of STEW-regularized multinomial logistic regression, as used in Chapter 5. The repository also contains a Tetris implementation and example scripts to run Tetris experiments.

▸ IMPLEMENTATION DETAILS FOR REGRESSION MODELS IN CHAPTER 3.

Ordinary least squares (OLS). We used the R[1] builtin function *lm()* for estimating OLS. Whenever there were more predictors than observations (that is, $p > n$), we used only the $n - 1$ predictors that are most correlated with the response to avoid numerical instabilities.

[1] R Core Team (2015)

Elastic-net regression.[2] The elastic net has two main parameters. Parameter $\lambda \geq 0$ controls the overall strength of regularization. The elastic net reduces to OLS for $\lambda = 0$. Parameter $0 \leq \alpha \leq 1$ controls the amount of *ridge* versus *Lasso* characteristics. The elastic net generalizes two other regularized linear models, ridge regression[3] when $\alpha = 0$ and the Lasso[4] when $\alpha = 1$. We used the R package *glmnet*[5] in order to estimate the elastic net. The parameters $\alpha$ and $\lambda$ were jointly optimized using 10-fold cross-validation using a two-dimensional grid search with $\alpha \in \{0, 0.25, 0.5, 0.75, 1\}$ and $\lambda$ on the built-in search path of glmnet, which used a log-spaced grid with a maximum of 100 candidate values.

[2] Zou and Hastie (2005)

[3] Hoerl and Kennard (1970)
[4] Tibshirani (1996)
[5] Friedman et al. (2015)

Random forest regression[6] is a non-parametric and non-linear regression model. Random forests are ensembles of regression trees.[7] We used the R package *randomForest*[8] and use `ntree = 500` trees per forest.

[6] Breiman (2001)
[7] Breiman et al. (2017)
[8] Liaw and Wiener (2002)

▸ IMPLEMENTATION DETAILS FOR REGRESSION MODELS IN CHAPTER 4.

Shrinkage toward equal weights (STEW). We used STEW with $q = 2$ for the experiments. Solutions were computed using the closed-form solution given in Equation 4.5. The regularization strength $\lambda$ was optimized on the training set using $k$-fold cross validation on a log-spaced grid with maximum 100 candidate values (the search stops early when the norm of the difference be-

tween actual and previously estimated weights is smaller than some small $\varepsilon > 0$).

Non-negative lasso (NNLasso). We used the R package *nnlasso* (Mandal and Ma, 2016) to estimate non-negative least squares with lasso penalty (NNLasso). The regularization strength $\lambda$ was optimized using $k$-fold cross-validation and using the built-in search path of the nnlasso package.

Elastic-net regression. See in implementation details for regression models in the previous chapter described further above in this appendix.

▸ THE STEW PENALTY IN THE IPSE ALGORITHM.

In Chapter 4, the STEW penalty is defined as

$$P_{\text{STEW}}(\boldsymbol{\beta}) = \sum_{i<j}(\beta_i - \beta_j)^2.$$

In Chapter 5, the penalty term used to regularize multinomial regression is

$$P_{\boldsymbol{d}}(\boldsymbol{\beta}) = \|\boldsymbol{\beta} - \boldsymbol{d}\|_2^2 = \sum_{i=1}^{p}(\beta_i - d_i)^2,$$

where $\boldsymbol{d} = (d_1, \ldots, d_p) \in \{-1, 1\}^p$ are feature direction estimates. Here we show that $P_{\text{STEW}}$ can be used to implement $P_{\boldsymbol{d}}$ in the policy learning context described in Section 5.4.

We start by showing that the STEW penalty is equivalent (up to a constant factor) to the squared Euclidean distance between the weight vector and the average weight vector, that is, $P_{\text{STEW}}(\boldsymbol{\beta}) = p\|\boldsymbol{\beta} - \bar{\boldsymbol{\beta}}\|_2^2 = p\left(\sum_{i=1}^{p}(\beta_i - \bar{\boldsymbol{\beta}})^2\right)$.

$$P_{\text{STEW}}(\boldsymbol{\beta}) = \sum_{i<j}(\beta_i - \beta_j)^2$$

$$= \sum_{i<j}(\beta_i^2 - 2\beta_i\beta_j + \beta_j^2)$$

$$= (p-1)\sum_{i=1}^{p}\beta_i^2 - 2\sum_{i<j}\beta_i\beta_j$$

$$= (p-1)\sum_{i=1}^{p}\beta_i^2 - 2\sum_{i<j}\beta_i\beta_j + \sum_{i=1}^{p}\beta_i^2 - \sum_{i=1}^{p}\beta_i^2$$

Adding a "constructive zero": $\sum_{i=1}^{p}\beta_i^2 - \sum_{i=1}^{p}\beta_i^2 = 0$.

$$= p\sum_{i=1}^{p}\beta_i^2 - 2\sum_{i<j}\beta_i\beta_j - \sum_{i=1}^{p}\beta_i^2$$

$$= p\sum_{i=1}^{p}\beta_i^2 - \left(\sum_{i=1}^{p}\beta_i\right)^2$$

Via expression for the square of a finite sum: $\left(\sum_{i=1}^{n}a_i\right)^2 = \sum_{i=1}^{n}a_i^2 + 2\sum_{i=1}^{n}\sum_{j=1}^{i-1}a_i a_j$

$$= p\sum_{i=1}^{p}\beta_i^2 - p^2\bar{\boldsymbol{\beta}}^2$$

$$= p\left(\sum_{i=1}^{p}\beta_i^2 - p\bar{\boldsymbol{\beta}}^2\right)$$

$$= p\left(\sum_{i=1}^{p}\beta_i^2 - 2p\bar{\boldsymbol{\beta}}^2 + p\bar{\boldsymbol{\beta}}^2\right)$$

$$= p\left(\sum_{i=1}^{p}\beta_i^2 - 2\bar{\boldsymbol{\beta}}\sum_{j=1}^{p}\beta_j + p\bar{\boldsymbol{\beta}}^2\right)$$

$$= p\left(\sum_{i=1}^{p}(\beta_i^2 - \beta_j\bar{\boldsymbol{\beta}} + \bar{\boldsymbol{\beta}}^2)\right)$$

$$= p\left(\sum_{i=1}^{p}(\beta_i - \bar{\boldsymbol{\beta}})^2\right)$$

$$= p\|\boldsymbol{\beta} - \bar{\boldsymbol{\beta}}\|_2^2$$

Equivalence between this term and $P_d(\boldsymbol{\beta})$ then follows from the following considerations and pre-processing steps:

1. The factor $p$ can be absorbed into the regularization strength $\lambda$.

2. Knowledge of the feature direction vector $\boldsymbol{d}$ allows us to direct all features.[9] After all features are directed, the vector of feature directions is given by $\boldsymbol{d} = (1, \ldots, 1)$.

   [9] See also Section 2.2.1.

3. In the context of the discrete choice linear model considered in Section 5.4, the policy in invariant to scale. That is, the policy remains unchanged if all weights are multiplied by the same positive constant. It follows that shrinking weights to a constant produces the same behavior, irrespective of the value of that constant.

Therefore, in the given context the STEW penalty can be used to implement $P_d$. Note that this is not true in a regression context because the corresponding linear model is not invariant to scale.

▶ CLASSIFICATION-BASED MODIFIED POLICY ITERATION,[10] (or CBMPI) is a benchmark algorithm used in the experiments in Chapter 5. CBMPI is a rollout-based reinforcement learning algorithm with the following particularities, compared to classical classification-based reinforcement learning.[11]

[10] Gabillon et al. (2013) and Scherrer et al. (2015)

[11] Lagoudakis and Parr (2003). Classification-based reinforcement learning is discussed in detail in Section 2.3.2.

1. *Value-function approximation.* The CBMPI algorithm uses rollouts not only to approximate a policy function but also to approximate a state-value function, which is a regression task. The approximated action-values are bootstrapped in the rollout procedure itself and for cost-sensitive policy approximation, as explained below.

2. *Bootstrapped rollouts.* The CBMPI algorithm bootstraps the approximate state-value function to estimate action-values. Specifically, the approximated value of the last state of a rollout is added to the cumulative reward obtained during the rollout to obtain the following action-values estimate:

$$\hat{Q}(s, a) = \sum_{t=1}^{T} r_t + \hat{v}(s_T),$$

where $\hat{v}(s_T)$ is the approximated value of the last state of the rollout, $s_T$.

3. *Cost-sensitive classification.* For their Tetris experiments, Scherrer et al. (2015) used const-sensitive classification to approximate a new policy. In the context of policy approximation, cost-sensitive classification minimizes the loss function

$$\mathcal{L}(\pi) = \sum_{s \in \mathcal{R}} \left[ \max_{a \in \mathcal{A}(s)} \hat{Q}(s, a) - \hat{Q}(s, \pi(s)) \right], \tag{A.1}$$

where $\hat{Q}(s, a)$ is the rollout value for action $a$, and $\mathcal{R}$ is the set of rollout starting states. In their Tetris experiments, Scherrer et al. used a linear policy $\pi(s) = \underset{a \in \mathcal{A}(s)}{\arg\max} \, \phi(s, a)^\top \beta$ and optimized the feature weights $\beta$ using an evolutionary algorithm called covariance matrix adaptation evolution strategy (CMA-ES).[12]

[12] Hansen and Ostermeier (2001)

4. *Rollout set.* Similar to Lagoudakis and Parr (2003), CBMPI uses a pre-computed data set of rollout states. The initial rollout states are sampled from a large, pre-computed *rollout data set* that is generated by an expert policy.[13]

[13] Scherrer et al. (2015) used games played by the DU controller (Thiery and Scherrer, 2009b) to generate the rollout data set. We used the exact same rollout data set as Scherrer et al. (2015) in our experiments.

---

**Algorithm 9** Classification-Based Modified Policy Iteration (CBMPI, Scherrer et al., 2015).

---

**Output:**

$\pi(s) : \mathcal{S} \rightarrow \mathcal{A}$      *// policy that returns an action $a \in \mathcal{A}$ for given state $s \in \mathcal{S}$*

**Input:**

$\Pi$                                                   *// policy space*

$\mathcal{D}$                                       *// set of rollout starting states*

$\pi \leftarrow$ uniform random policy

$v \leftarrow$ arbitrary state value function

**for** $k = 0, 1, 2, \ldots$ **do**

    **for all** $s \in \mathcal{D}$ **do**

        $\tilde{v}(s) \leftarrow \textsc{BootstrappedRollout}(s, \pi(s), \pi, v)$     *// see Algorithm 10*

        **for all** $a \in \mathcal{A}(s)$ **do**

            $\tilde{Q}(s, a) \leftarrow \textsc{BootstrappedRollout}(s, a, \pi, v)$

        **end for**

    **end for**

    *// Regression to learn new value function*

    $v \leftarrow \underset{v}{\mathrm{argmin}} \ \sum_{s \in \mathcal{D}} \left( v(s) - \tilde{v}(s) \right)^2$

    *// Cost-sensitive classification to learn new policy*

    $\pi \leftarrow \underset{\pi \in \Pi}{\mathrm{argmin}} \ \sum_{s \in \mathcal{D}} \left[ \max_{a \in \mathcal{A}(s)} \tilde{Q}(s, a) - \tilde{Q}(s, \pi(s)) \right]$

**end for**

---

---

**Algorithm 10** BOOTSTRAPPEDROLLOUT($s, a, \pi_r, v$): Rollout procedure for estimating the value of an action $a$ in state $s$ using rollout policy $\pi_r$ and value function $v$.

---

**Output:**

$\hat{Q} \in \mathbb{R}$, *estimated value of taking action a in s*

**Input:**

$s \in \mathcal{S}$                                          *// rollout starting state*

$a \in \mathcal{A}(s)$                                *// action to be evaluated*

$\pi_r(s) : \mathcal{S} \to \mathcal{A}$                                *// rollout policy*

$M \in \mathbb{N}$                                   *// number of rollouts*

$T \in \mathbb{N}$                                      *// rollout length*

$\gamma \in [0, 1]$                                   *// discount factor*

$\mathcal{G}(s, a) : \mathcal{S} \times \mathcal{A}(s) \to \mathcal{S} \times \mathbb{R}$            *// generative model*

$v(s) : \mathcal{S} \to \mathbb{R}$                            *// state-value function*

**for all** $j = 1, \ldots, M$ **do**

   $(s', r) \leftarrow \mathcal{G}(s, a)$

   $\hat{Q}_j \leftarrow r$

   $s \leftarrow s'$

   **for all** $t = 1, \ldots, T - 1$ **do**

      $(s', r) \leftarrow \mathcal{G}(s, \pi_r(s))$

      $\hat{Q}_j \leftarrow \hat{Q}_j + \gamma^t r$

      $s \leftarrow s'$

   **end for**

   $\hat{Q}_j \leftarrow \hat{Q}_j + \gamma^T v(s)$          *// bootstrap value of the last rollout state*

**end for**

**return** $\hat{Q} \leftarrow \frac{1}{M} \sum_{j=1}^{M} \hat{Q}_j$

---

# B

**Learning curves on individual real-world data sets.** The following figures show learning curves on the individual data sets that were used to compute average learning curves shown in the main part of the dissertation.

- Chapter 3. Additional figures B.1 to B.3 correspond to Figure 3.2.

- Chapter 4. Figures B.6 and B.7 correspond to panels c and f of Figure 4.5, respectively.

**Comparison of regularization paths.** Figure B.4 shows regularization paths (weight estimates as a function of the regularization strength $\lambda$) for the STEW with $l_1$ and $l_2$ penalties, and the total variation (TV) regularization period with various orders of the predictors. The differences in regularization behavior are discussed in Section 4.3.

**STEW in an environment with high variance of weights.** Figure B.5 shows learning curves for STEW in an environment in which the true weights are highly non-equal. We used the experimental setup of Section 4.5.1 with prior weight distribution $\boldsymbol{\beta} \sim \mathcal{U}(0, 50)$.
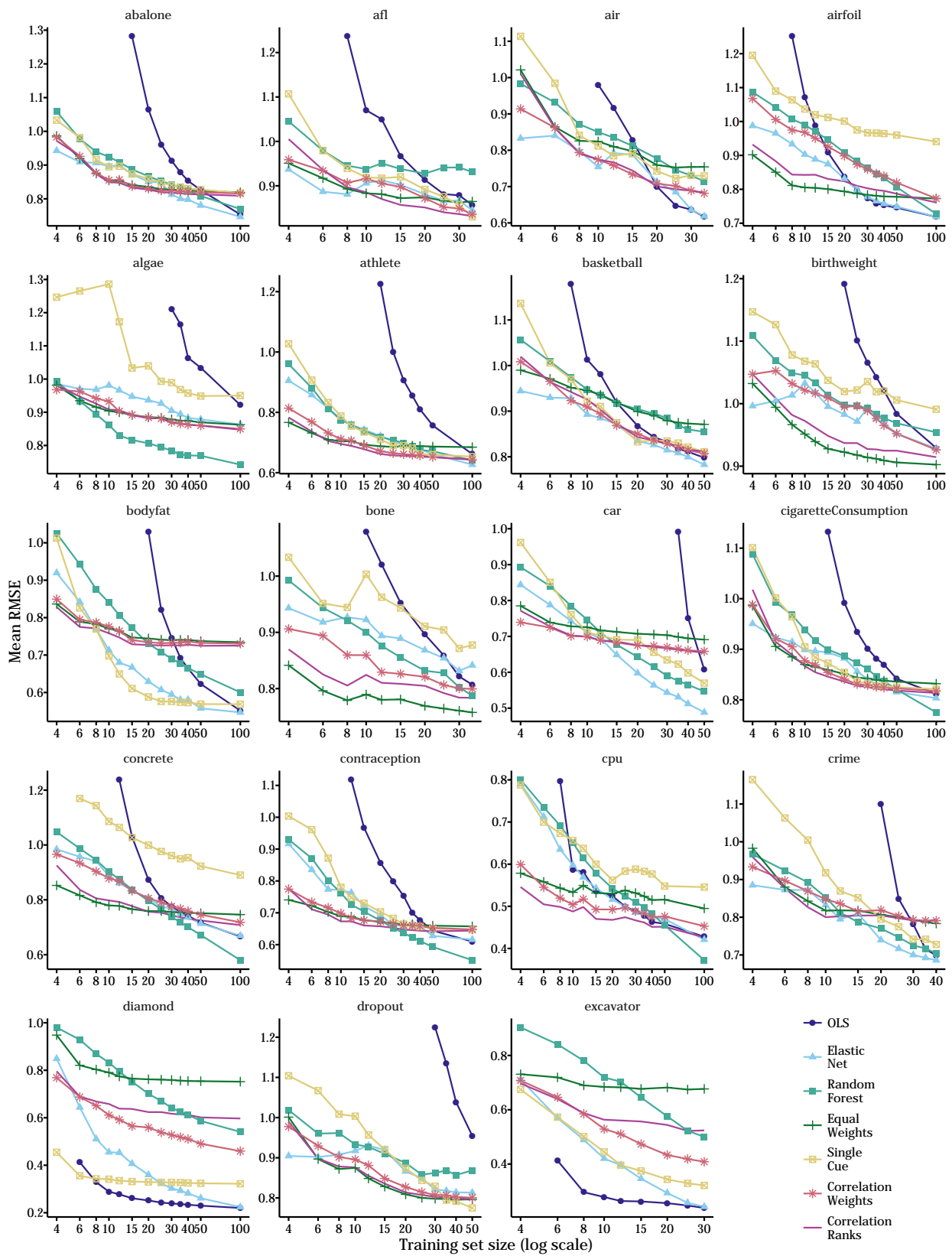
Figure B.1: Learning curves for simple regression models. Data sets 1 to 18.
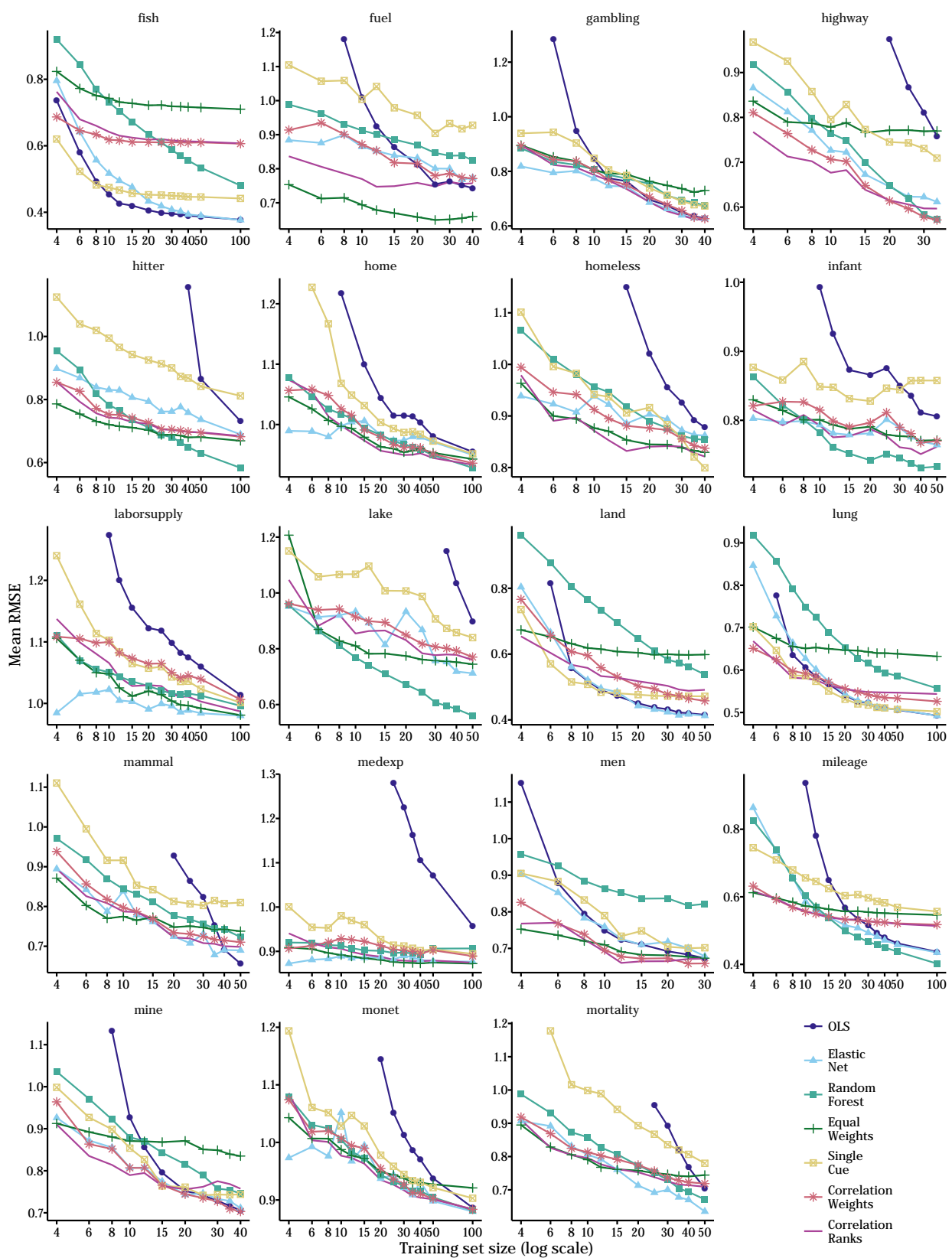
Figure B.2: Learning curves for simple regression models. Data sets 19 to 37.
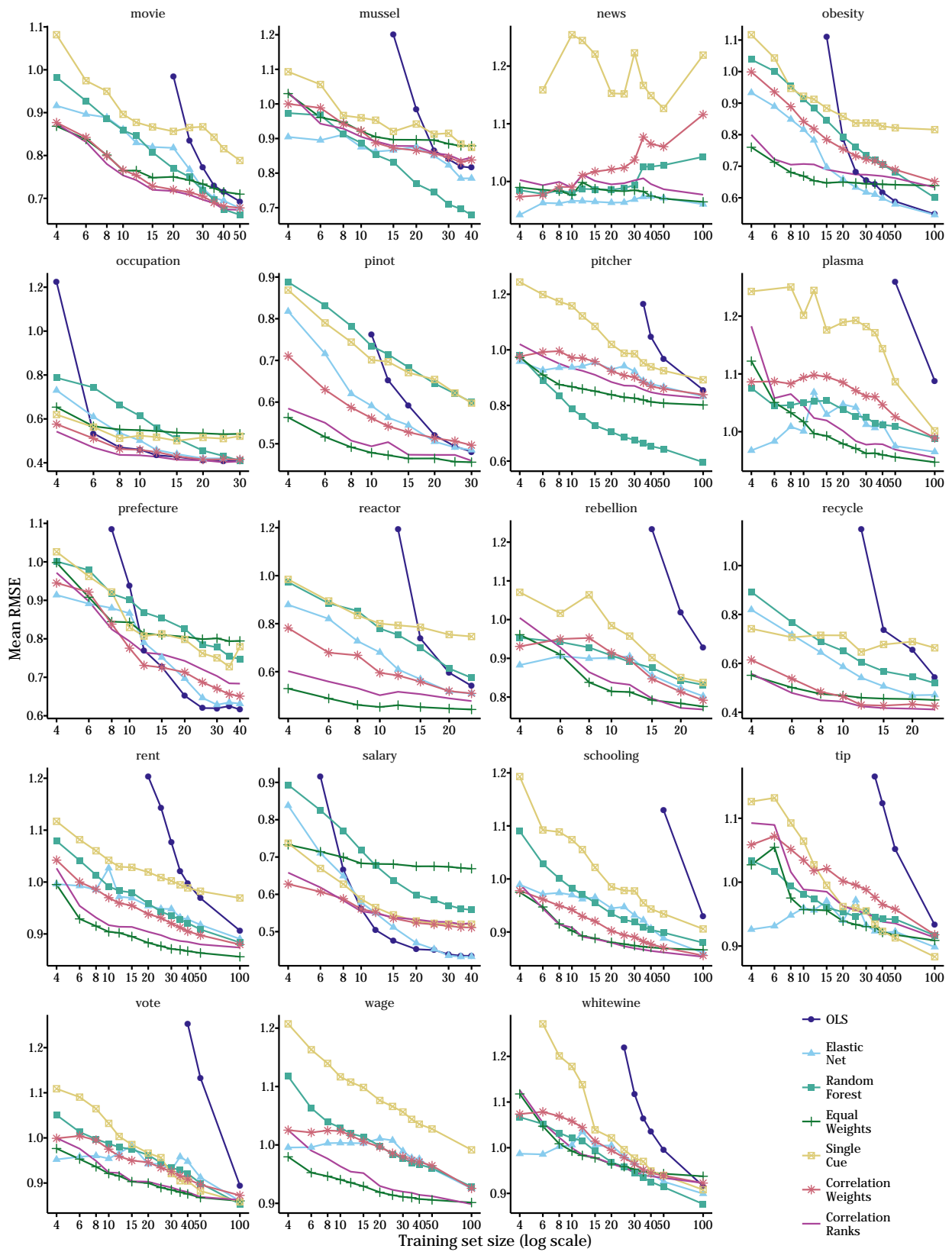
Figure B.3: Learning curves for simple regression models. Data sets 38 to 57.
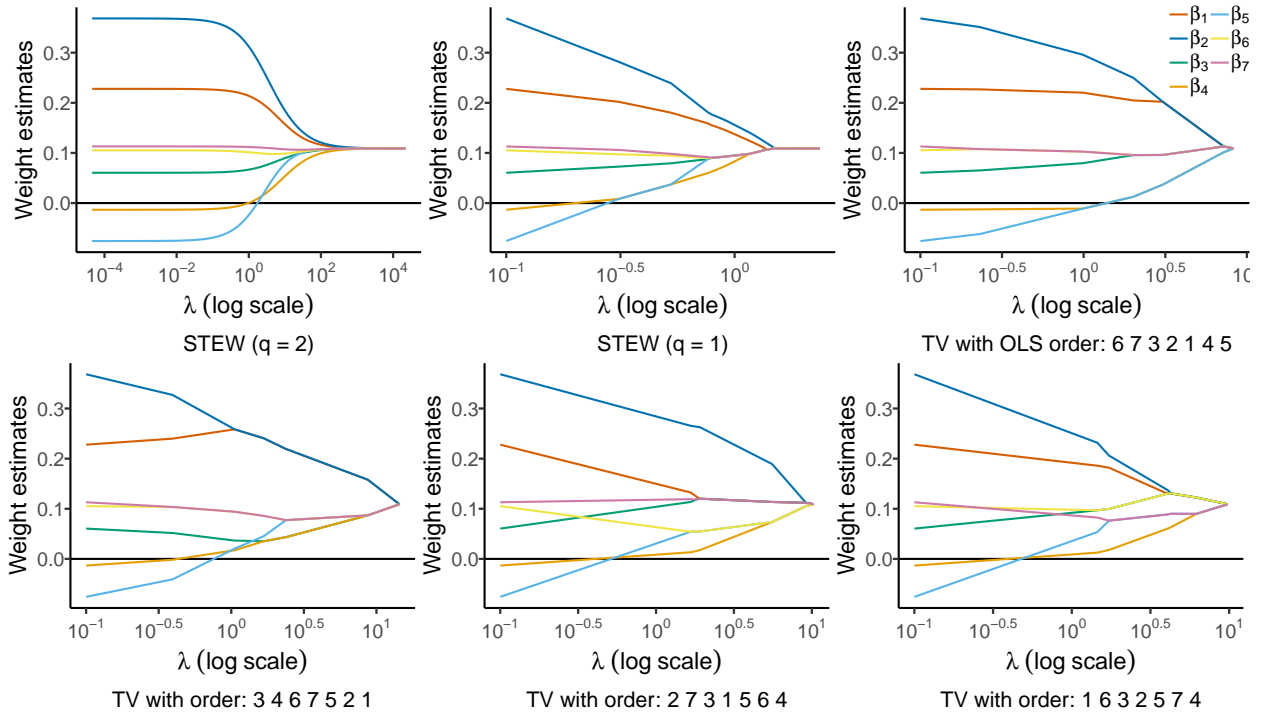
Figure B.4: Weight estimates on the *Rent* data set as a function of regularization strength $\lambda$ for STEW with $l_1$ and $l_2$ penalties, and total variation (TV) with various orders of the predictors.
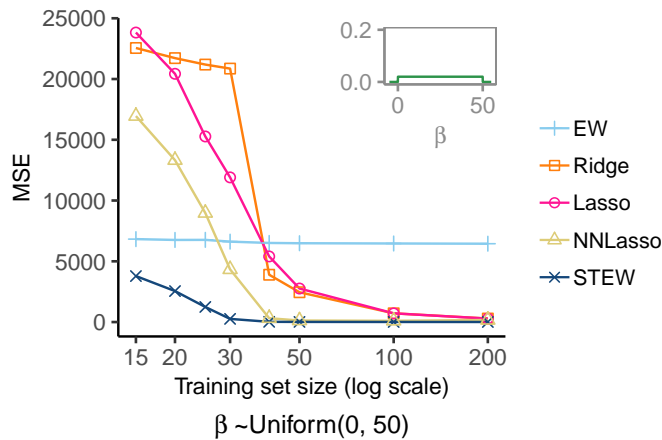


Figure B.5: Mean squared error (MSE) across 400 repetitions for equal weights (EW), ridge regression, the Lasso, the non-negative Lasso (NNLasso) and shrinkage toward equal weights (STEW) as a function of training set size in an environment where the true weights are sampled from $\beta \sim \mathcal{U}(0, 50)$.
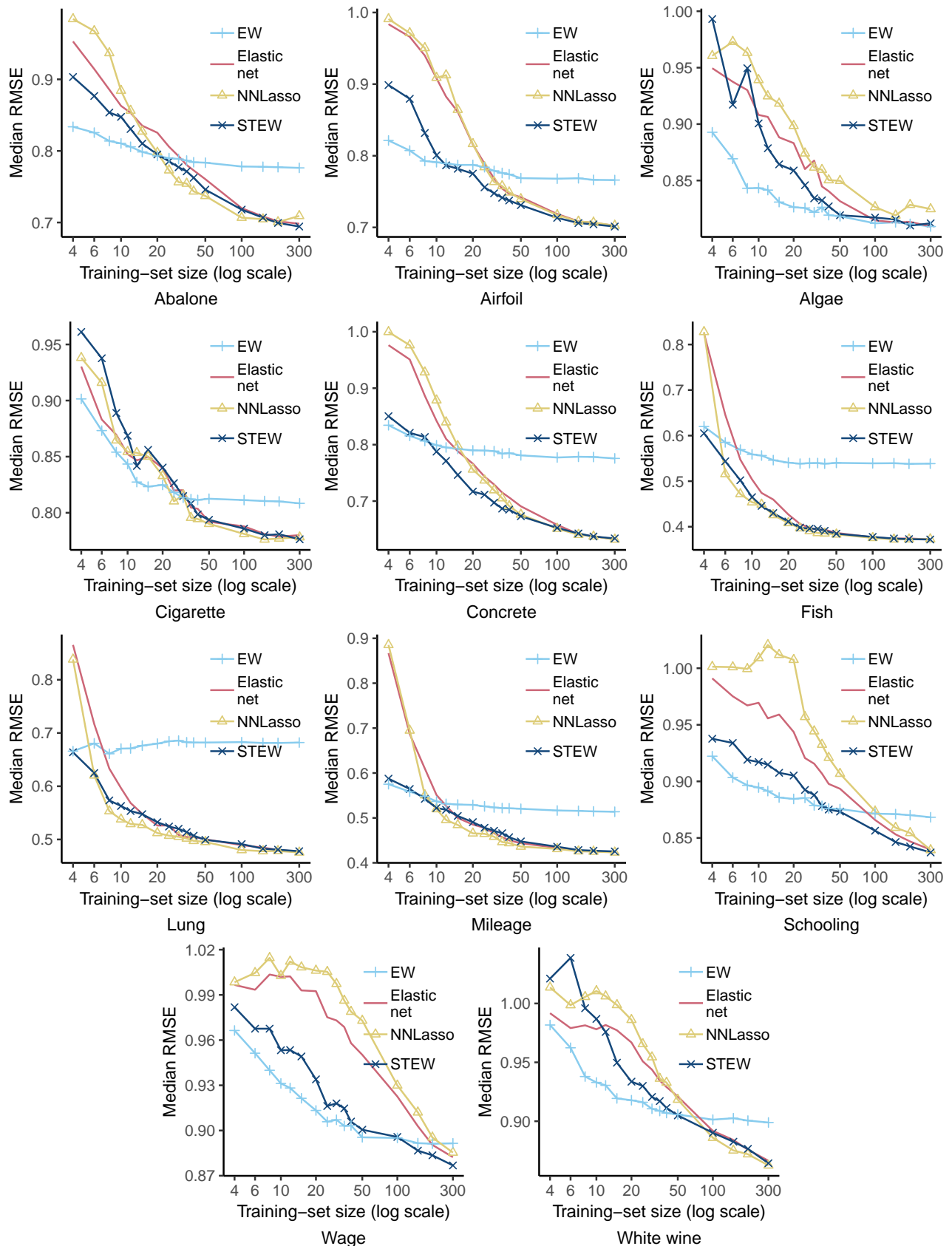
Figure B.6: Median root mean squared error (RMSE) across 200 repetitions on individual data sets. Predictors were directed based on a Lasso estimate on the entire data set.
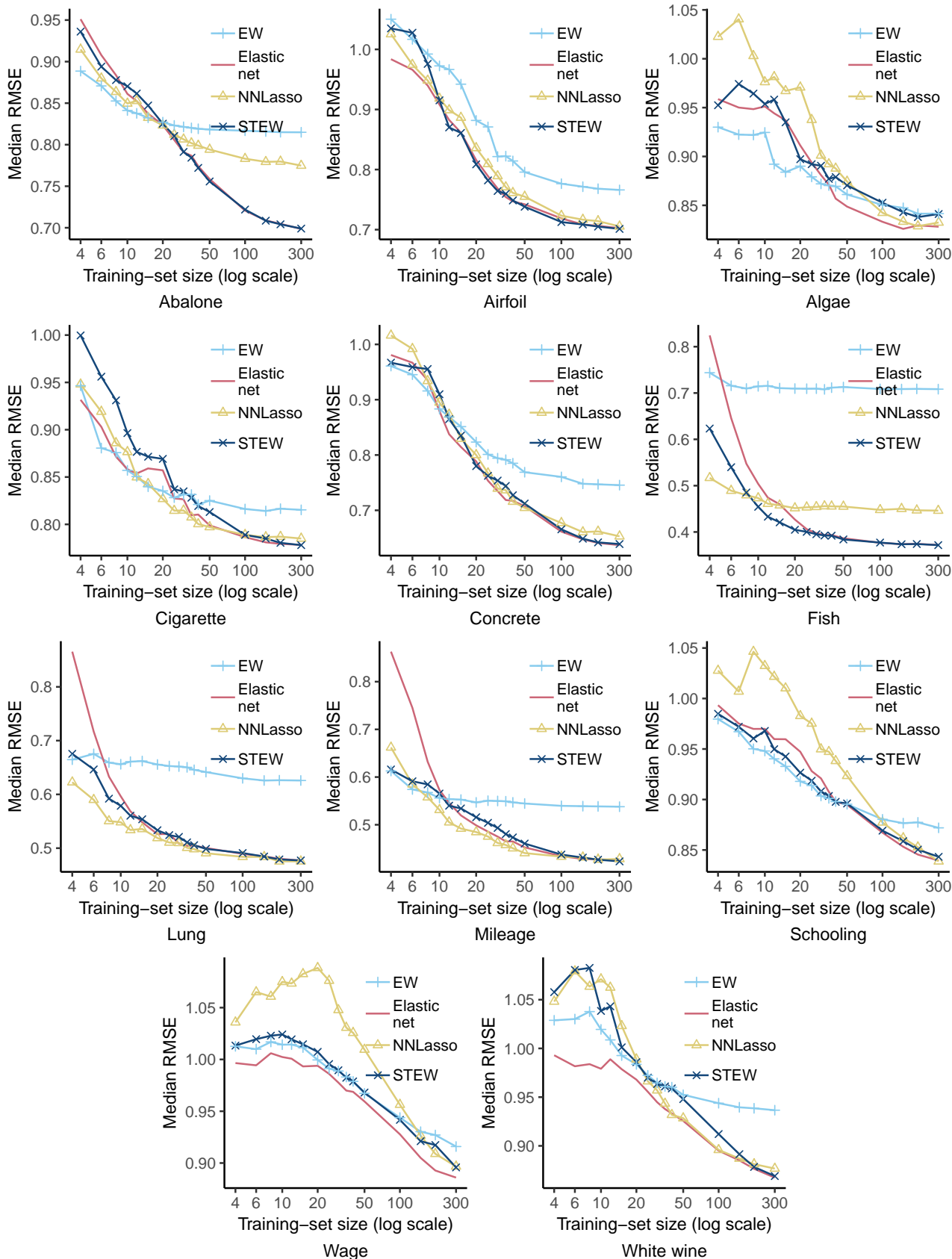
Figure B.7: Median root mean squared error (RMSE) across 200 repetitions on individual data sets. Predictors were directed based on the training set using Pearson correlation coefficients between features and response.

APPENDIX C. DATA SETS

This appendix describes each data set from Table 3.2 in more detail. It shows the number of objects (or observations), the criterion (response) variable, the attributes (predictors), and the source of the data set.

*Abalone*　Objects: 4177 abalones (sea snails). Criterion: age (measured in visible rings). Attributes: sex, length, diameter, height, whole weight, shucked weight, viscera weight, shell weight. Source: This data set comes from a study by Nash et al. (1994). It is available from the UCI Machine Learning Repository (Bache and Lichman, 2013).

*AFL*　Objects: 41 Australian Football League (AFL) games at the Melbourne Cricket Ground in 1993 and 1994. Criterion: attendance. Attributes: forecasted maximum temperature on the day of the game, total attendance at other AFL games in Melbourne and Geelong on the day of the game, total membership in the two clubs whose teams were playing, number of players in the top 50 who participated in the game, number of days since the earliest game of the season. Source: This data set was assembled by Rowan Todd and Mark Mc-Naughton for a class project at the University of Queensland in a statistics course taught by Margaret Mackisack. The data sources were *The Football Bible '94* by Rex Hunt, *The Weekend Australian*, *Inside Football*, and *Football Record*. The data set is available from OzDASL data library (Smyth, 2011), where it is listed with the name *AFL Crowd Attendance at the MCG*.

*Air*　Objects: 41 cities in the United States. Criterion: annual mean concentration of sulfur dioxide. Attributes: average annual temperature, number of manufacturing enterprises employing 20 or more workers, population, average annual wind speed, average annual rainfall, average number of days with rainfall per year. Source: The data were gathered by Sokal and Rohlf (1981) from several publications of the United States government. The data set is reported in a book by Hand et al. (1994) with identifying number 26 and label *air pollution in US cities*.

*Airfoil*　Objects: 1503 airfoils at various wind tunnel speeds and angles of attack. Criterion: scaled sound pressure level, in decibels. Attributes: fre-

quency, angle of attack, chord length, free-stream velocity, suction side displacement thickness. SOURCE: This data set comes from a study by Brooks et al. (1989). It is available from the UCI Machine Learning Repository (Bache and Lichman, 2013).

*Algae*   OBJECTS: 340 samples from European rivers taken over a period of approximately one year. CRITERION: density of algae type a. ATTRIBUTES: concentrations of eight chemicals, season (fall, winter, spring, summer), river size (small, medium, large), fluid velocity (low, medium, high). SOURCE: The data set is from the 1999 Computational Intelligence and Learning (COIL) competition. It is available from the UCI data repository (Bache and Lichman, 2013), where it is labeled *COIL 1999 competition data*.

*Athlete*   OBJECTS: 202 nationally-ranked athletes in Australia. CRITERION: blood hemoglobin concentration. ATTRIBUTES: body mass index, sum of skin folds, percent body fat, lean body mass, height, weight, sex, the sport the athlete competes in (basketball, field, gymnastics, netball, rowing, track 400m, swimming, sprint, tennis, water polo). SOURCE: The data were collected by Telford and Cunningham (1991) at the Australian Institute of Sport. The data set is reported by Maindonald and Braun (2010) and is available from associated R package *DAAG* (Maindonald and Braun, 2013) with label *ais*.

*Basketball*   OBJECTS: 96 basketball players. CRITERION: points scored per minute. ATTRIBUTES: assists per minute, height, time played, age. SOURCE: The data set is reported by Simonoff (2003) and is available from a website maintained by the author (Simonoff, 2015), where it is labeled *baskball.dat*.

*Birthweight*   OBJECTS: 189 newborns. CRITERION: birth weight. ATTRIBUTES: age of mother, weight of mother at last menstrual period, race (white, black, other), number of previous premature labors, number of physician visits during the first trimester, presence of uterine irritability, whether the mother smoked during pregnancy, whether the mother has a history of hypertension. SOURCE: The data were collected at Baystate Medical Center in Springfield, Massachusetts in 1986 (Hosmer and Lemeshow, 2000). The data set is electronically available from R package *MASS* (Venables and Ripley, 2002; Ripley et al., 2013), where it is labeled *birthwt*.

*Bodyfat*   OBJECTS: 252 males. CRITERION: percentage of body fat determined by underwater weighing. ATTRIBUTES: age, weight, height, and various body circumference measurements: neck, chest, abdomen, hip, thigh, knee, ankle, biceps, forearm, wrist. SOURCE: The data were collected by Penrose et al. (1985). The data set is available from StatLib (Meyer and Vlachos, 1989) with label *bodyfat*.

*Bone*    OBJECTS: 42 male skeletons buried in coffins. CRITERION: nitrogen content. ATTRIBUTES: deposition time, depth of burial, age of the person, whether quicklime was added to the coffin at burial, whether skeleton was contaminated with oil, burial site (2 sites 130 km apart in northern England). SOURCE: The data were collected by Jarvis (1997). The data set is available electronically from a data repository maintained by Winner (2015), where it is listed with the name *nitrogen levels in skeletal bones of various ages and internment lengths.*

*Car*    OBJECTS: 93 passenger cars on sale in the United States in 1993. CRITERION: sale price of the most basic version of the car. ATTRIBUTES: city mileage, highway mileage, cylinders (3, 4, 5, 6, 8, rotary), engine size, maximum horsepower, engine revolutions per mile in highest gear, fuel tank capacity, passenger capacity, length, wheelbase, width, weight, rear seat room, luggage capacity, u-turn space, airbag (none, driver only, both driver and passenger), whether a manual transmission version is available, whether the manufacturer is from the United States, type of car (small, sporty, compact, midsize, large, van), drivetrain type (rear, front, four-wheel drive). SOURCE: The data set was assembled by  Lock (1993) using information from *PACE New Car & Truck 1993 Buying Guide* and *Consumer Reports April 1993 Annual Auto Issue.* It is available from the R package *MASS* (Venables and Ripley, 2002; Ripley et al., 2013) with label *Cars93.*

*Cigarette*    OBJECTS: 528 states in the USA (in different years). CRITERION: packs per capita. ATTRIBUTES: year, consumer price index, state population, state personal income, average state, federal, and average local excise taxes for fiscal year. SOURCE: The data set was assembled by Professor Jonhatan Gruber, MIT. It has been used in an introductory econometrics textbook (Stock and Watson, 2003). It is available electronically from R package *Ecdat* (Croissant, 2013).

*Concrete*    OBJECTS: 1030 concrete samples. CRITERION: concrete compressive strength. ATTRIBUTES: cement (kg/$m^3$), blast furnace slag (kg/$m^3$), fly ash (kg/$m^3$), water (kg/$m^3$), superplasticizer (kg/$m^3$), coarse aggregate (kg/$m^3$), fine aggregate (kg/$m^3$), age in days. SOURCE: This data set comes from a study by Yeh (1998). It is available from the UCI Machine Learning Repository (Bache and Lichman, 2013).

*Contraception*    OBJECTS: 210 localities in the world (most are United Nations members but includes areas like Hong Kong that are not independent countries). CRITERION: percentage of unmarried women using a modern method of contraception. ATTRIBUTES: annual population growth rate, per capita 2001 gross domestic product, percentage of females over the age of 15 who are economically active, population, expected number of live births per female in 2000, percentage of population that is urban in 2001. SOURCE: The data set is reported

by Weisberg (2005) who notes that the source of the data is the United Nations. It is electronically available from R package *alr3* (Weisberg, 2011) where it is labeled *UN3*.

*CPU*    OBJECTS: 209 central processing units on the market in 1981–1984. CRITERION: published performance on a benchmark mix relative to an IBM 370/158 Model 3. ATTRIBUTES: cycle time, minimum main memory, maximum main memory, cache memory, minimum number of channels, maximum number of channels. SOURCE: The data set was assembled by Ein-Dor and Feldmesser (1987) using information from *Computerworld* magazine. It is electronically available from the R package *MASS* (Venables and Ripley, 2002; Ripley et al., 2013) with label *cpus*.

*Crime*    OBJECTS: 47 states of the United States. CRITERION: crime rate in 1960. ATTRIBUTES: percentage of males aged 14–24 in state population, indicator variable for a southern state, mean years of schooling of the population aged 25 years or older, per capita expenditure on police protection in 1960, per capita expenditure on police protection in 1959, labor force participation rate of civilian urban males in the age-group 14–24, number of males per 100 females, state population in 1960, percentage of nonwhites in the population, unemployment rate of urban males 14–24, unemployment rate of urban males 35–39, wealth (median value of transferable assets or family income), income inequality (percentage of families earning below half the median income), probability of imprisonment (ratio of number of commitments to number of offenses), average time served by offenders in state prisons before their first release. SOURCE: The data set was assembled by Ehrlich (1973) from various publications of the United States government, including *Uniform Crime Reports* of the Federal Bureau of Investigation, United States Census, and *National Prison Statistics Bulletin*. Rounded data, taken from Vandaele (1978), is electronically available from OzDASL (Smyth, 2011), where it is labeled *uscrime*.

*Diabetes*    OBJECTS: 442 diabetes patients. CRITERION: a quantitative measure of disease progression one year after baseline. ATTRIBUTES: age, sex, body mass index, average blood pressure and six blood serum measurements. SOURCE: The data was used in Efron et al. (2004). It is available electronically from the R package *lars* (Efron and Hastie, 2013).

*Diamond*    OBJECTS: 308 round diamond stones. CRITERION: sale price. ATTRIBUTES: weight in carats, color purity (D, E, F, G, H, I), clarity (internally flawless, very very slight inclusion 1, very very slight inclusion 2, very slight inclusion 1, very slight inclusion 2), certification (Gemmological Institute of America, International Gemmological Institute, Hoge Raad Voor Diamant). SOURCE: The data set was assembled by Chu (2001) from advertisements in Singapore's *Business Times* edition of February 18, 2000. It is available electronically from the R

package *Ecdat* (Croissant, 2013).

*Dropout*    OBJECTS: 63 public high schools in Chicago. CRITERION: dropout rate. ATTRIBUTES: enrollment, attendance rate, parental involvement rate, percent limited-English students, percent low-income students, average class size, percent White students, percent Black students, percent Hispanic students, percent Asian students, percent minority teachers, average composite ACT score, IGAP scores: reading, math, science, social science, writing. SOURCE: This prediction problem is from a study by Czerlinski et al. (1999). Their data sources are two articles in the February 1995 issue of *Chicago* magazine (Morton, 1995; Rodkin, 1995), where the authors note that their primary data source is Illinois State Board of Education's 1994 School Report Card.

*Excavator*    OBJECTS: 33 hydraulic excavators operating in the opencast mining industry in the United Kingdom. CRITERION: annual maintenance cost. ATTRIBUTES: weight, type of machine (front shovel, backacter), type of industry (opencast coal, opencast slate), company attitude to used oil analysis (regular use, not). SOURCE: The data are from a study by Edwards et al. (2000). The data set is available electronically from a data repository maintained by Winner (2015), where it is listed with the name *construction plant maintenance costs*.

*Fish*    OBJECTS: 413 female Arctic charr. CRITERION: number of eggs. ATTRIBUTES: age, weight, mean egg weight. SOURCE: This prediction problem is from a study by Czerlinski et al. (1999). The data were collected by Christian Gillet from the French National Institute for Agricultural Research. The data set used in this study was obtained via personal communication.

*Fuel*    OBJECTS: 51 states and the District of Columbia of the United States. CRITERION: per capita motor fuel consumption in 2001. ATTRIBUTES: population, fuel tax rate, per capita income, miles of federal-aid primary highways, proportion of the population who are licensed drivers. SOURCE: The data set is reported by Weisberg (2005) who notes that the source of the data is the Federal Highway Administration. The data set is available from R package *alr3* (Weisberg, 2011) where it is labeled *Fuel2001*.

*Gambling*    OBJECTS: 47 British teenagers. CRITERION: annual gambling expenditure. ATTRIBUTES: sex, socio-economic status, weekly income, verbal score. SOURCE: The data were collected by Ide-Smith and Lea (1988). The data set is reported by Faraway (2005) and is electronically available from the associated R package *faraway* (Faraway, 2011), where it is labeled *teengamb*.

*Highway*    OBJECTS: 39 segments of highway in Minnesota. CRITERION: accident rate. ATTRIBUTES: segment length, average daily traffic count, truck volume as a percent of total volume, speed limit, number of lanes, lane width,

shoulder width, number of signalized interchanges per mile, number of freeway-type interchanges per mile, number of access points per mile, highway type (federal interstate highway, principal arterial highway, major arterial, other). SOURCE: The data set is reported by Weisberg (2005) who notes that the data were taken from an unpublished master's paper in civil engineering by Carl Hoffstedt. The data set is available electronically from R package *alr3* (Weisberg, 2011).

*Hitter*    OBJECTS: 322 hitters in North American Major League Baseball. CRITERION: annual salary at the beginning of the 1987 season. ATTRIBUTES: 1986 performance: number of at bats, hits, home runs, runs scored, runs batted in, walks, putouts, assists, errors; career performance: number of at bats, hits, home runs, runs scores, runs batted in, walks; number of years in the major leagues; division at the end of the 1986 season (East, West); league at the end of the 1986 season (American, National); league at the beginning of the 1987 season (American, National). SOURCE: The data set was prepared by the Statistical Graphics Section of the American Statistical Association for the 1988 Annual Statistical Meetings and is available from StatLib (Meyer and Vlachos, 1989). The version used in this work is from Fox (2008), includes corrections by Hoaglin and Velleman (1995), and is electronically available from a website maintained by Fox (2015).

*Homeless*    OBJECTS: 50 cities in the United States. CRITERION: rate of homelessness. ATTRIBUTES: mean temperature, unemployment rate, percentage of inhabitants with incomes below the poverty line, vacancy rate, population, percentage of public housing, whether the city has rent control. SOURCE: The data set was assembled by Tucker (1987) from Department of Housing and Urban Development's 1984 *Report to the Secretary on the Homeless and Emergency Shelters* and other sources.

*Home*    OBJECTS: 3281 homes sold in San Francisco. CRITERION: sales price. ATTRIBUTES: number of bedrooms, interior area of the property in squarefeet, lotsize of the property, year the property was built. SOURCE: The data were reported in Adler (2010) and are available from the associated R package *nutshell* (Adler, 2012) with label *sanfrancisco.home.sales*.

*Infant*    OBJECTS: 105 nations. CRITERION: infant-mortality rate. ATTRIBUTES: per-capita income, geographic location (Africa, Americas, Asia, Europe), whether the country exports oil. SOURCE: Rates of infant mortality were obtained by Leinhardt and Wasserman (1979) from the editorial section of the *New York Times* (Crittenden, 1975). The data set is reported by Fox (2008) and is electronically available from a website maintained by the author (Fox, 2015).

*Laborsupply*    OBJECTS: 5320 working men (in the US). CRITERION: log of hourly wage. ATTRIBUTES: log of annual hours worked, number of children, age, disability (yes/no), year. SOURCE: The data comes from the Panel Study of Income Dynamics (PSID). It has been studied in Ziliak (1997). It is available electronically from R package *Ecdat* (Croissant, 2013).

*Lake*    OBJECTS: 69 world lakes. CRITERION: number of known crustacean zooplankton species present. ATTRIBUTES: surface area, maximum depth, mean depth, specific conductance, elevation, latitude, longitude, distance to nearest lake, number of lakes within 20 km, rate of photosynthesis. SOURCE: The data set is reported by Weisberg (2005) who notes that the data were provided by S. Dodson and discussed in part in Dodson (1992). The data set is electronically available from the R package *alr3* (Weisberg, 2011).

*Land*    OBJECTS: 67 counties in Minnesota. CRITERION: rent per acre paid in 1977 for agricultural land planted in alfalfa. ATTRIBUTES: average rent for all tillable land, density of dairy cows, proportion of pasture land, whether liming is required to grow alfalfa. SOURCE: The data set is reported by Weisberg (2005) who notes that the data were collected by Douglas Tiffany. The data set is electronically available from R package *alr3* (Weisberg, 2011) where it is labeled *landrent*.

*Lung*    OBJECTS: 654 children. CRITERION: forced expiratory volume in liters. ATTRIBUTES: age in years, height in inches, gender, exposure to smoking. SOURCE: The data were collected by Tager et al. (1979). The data set is reported in Ekstrom and Sørensen (2010) and is electronically available from the associated R package *isdals* (Ekstrom and Sørensen, 2014) where it is labeled *fev*.

*Mammal*    OBJECTS: 62 mammal species. CRITERION: average daily sleep. ATTRIBUTES: body weight, brain weight, maximum life span, gestation time, predation index, sleep exposure index, overall danger index.. SOURCE: The data are from a study by Allison and Cicchetti (1976). The data set is available from StatLib (Meyer and Vlachos, 1989), where it is labeled *sleep*.

*Medical Expenditure*    OBJECTS: 5574 US citizens. CRITERION: annual medical expenditures. ATTRIBUTES: coinsurance rate, whether the person has an individual deductible plan, log of the annual participation incentive payment, whether the person has a physical limitation, the number of chronic diseases, self-rate health (excellent, good, fair poor), log of annual family income, log of family size, years of schooling of household head, exact age, sex, child, whether household head is black. SOURCE: The data comes from the RAND Health Insurance Experiment (RHIE). It has been studied in Deb and Trivedi (2002). It is available electronically from R package *Ecdat* (Croissant, 2013) where it is called *medexp*.

*Men*    OBJECTS: 34 famous men. CRITERION: mean attractiveness rating. AT-
TRIBUTES: mean likeability rating, name recognition, whether the man is Amer-
ican. SOURCE: This prediction problem is from a study by Czerlinski et al.
(1999). The data were collected by Henss (1996) with the participation of 115
male and 131 female Germans, in ages ranging from 17 to 66 years old.

*Mileage*    OBJECTS: 398 cars built in 1970–1982. CRITERION: mileage. AT-
TRIBUTES: number of cylinders, engine displacement, horsepower, vehicle weight,
time to accelerate from 0 to 60 mph, model year, origin (American, European,
Japanese). SOURCE: The data set was prepared by the Committee on Statistical
Graphics of the American Statistical Association for its Second Exposition of
Statistical Graphics Technology, held in conjunction with the Annual Meetings
in Toronto, August 15–18, 1983. It is electronically available from StatLib (Meyer
and Vlachos, 1989), where it is labeled *cars*. The version used in the current
work is from the UCI Machine Learning Repository (Bache and Lichman, 2013),
named *Auto+MPG*, in which 8 of the original cars were removed because their
mileage values were missing.

*Mine*    OBJECTS: 44 coal mines in the Appalachian region of western Virginia.
CRITERION: number of fractures in upper seams of coal mines. ATTRIBUTES:
inner burden thickness, percent extraction of the lower previously mined seam,
lower seam height, duration of operation. SOURCE: The data set is reported by
Montgomery et al. (2001) and is electronically available from the associated R
package *mpg* (Braun, 2012) where it is labeled *p13.7*.

*Monet*    OBJECTS: 430 sales of paintings by Monet. CRITERION: sale price.
ATTRIBUTES: height of the painting, width of the painting, whether the painting
is signed, auction house where sale took place. SOURCE: The data set is reported
by Greene (2003). It is electronically available from a website maintained by the
author (Greene, 2015), where it is labeled *data on sales of Monet paintings*.

*Mortality*    OBJECTS: 60 metropolitan areas in the United States. CRITERION:
mortality rate. ATTRIBUTES: average annual precipitation, average January tem-
perature, average July temperature, percent population aged 65 or older, average
household size, median school years completed by those over 22, percent hous-
ing units that are sound and with all facilities, humidity, population density in
urbanized areas, percent nonwhite population in urbanized areas, percent em-
ployed in white collar occupations, percentage of families with income less than
$3000, relative hydrocarbon pollution potential, relative nitric oxides pollution
potential, relative sulfur dioxide pollution potential, annual average relative hu-
midity. SOURCE: The data set was assembled by McDonald and Schwing (1973).
It is electronically available from StatLib (Meyer and Vlachos, 1989), where it is
labeled *pollution*.

*Movie*   OBJECTS: 62 movies. CRITERION: first-run box office in the United States. ATTRIBUTES: production budget, index of star poser, whether the movie is a sequel, indicator for an action film, indicator for comedy, indicator for animation, indicator for horror, MPAA rating(G, PG, PG13, R), trailer views at traileraddict.com, message board comments at comingsoon.net, attention at fandango.com, percentage of Fandango votes for "can't wait to see". SOURCE: The data set is reported by Greene (2003) and is electronically available from a website maintained by the author (Greene, 2015), where it is labeled *movie buzz data*.

*Mussel*   OBJECTS: 44 rivers in eastern United States. CRITERION: number of freshwater mussel species. ATTRIBUTES: area of drainage basins, amount of dissolved solids, nitrate concentration, hydronium concentration, number of intervening rivers to four major species-source river systems: Alabama-Coosa, Apalachicola, Savannah, and St. Lawrence. SOURCE: The data are from an article by Sepkoski and Rex (1974). The data set is available electronically from an online repository maintained by Winner (2015), where the data set is described as *freshwater mussel species in US Rivers*.

*News*   OBJECTS: 39797 news articles (online). CRITERION: number of shares in social networks. ATTRIBUTES: 58 attributes in total, of which many are word statistics[1]. SOURCE: The original data are from the website www.mashable.com. The data set was studied in Fernandes et al. (2015). It is available from the UCI Machine Learning Repository (Bache and Lichman, 2013).

[1] For a complete description of all predictors, please see https://archive.ics.uci.edu/ml/datasets/Online+News+Popularity.

*Obesity*   OBJECTS: 136 children. CRITERION: somatotype (a scale of body type, ranging from 1, very thin, to 7, obese). ATTRIBUTES: sex, body measurements at ages 2, 9, and 18: height, weight, leg circumference, strength. SOURCE: The data were collected by Tuddenham and Snyder (1954) on children born in Berkeley, California, between January 1928 and June 1929. The data set is reported by Weisberg (2005) and is electronically available from the associated R package *alr3* where it is labeled *BGSall*.

*Occupation*   OBJECTS: 36 occupations. CRITERION: prestige rating of the National Opinion Research Center (NORC). ATTRIBUTES: suicide rate among males aged 20–64, median income, median number of school years completed.. SOURCE: The data set was assembled by Labovitz (1970) using data from the U.S. Census of 1950 and prestige rankings obtained by NORC in its 1947 survey. It is reported in a book by Hand et al. (1994) with identifying number 490 and label *prestige, income, education, and suicide rates for 36 occupations*.

*Pinot*   OBJECTS: 38 samples of Pinot Noir wine. CRITERION: quality. ATTRIBUTES: clarity, aroma, body, flavor, oakiness, region. SOURCE: The data set

is reported by Montgomery et al. (2001) and is electronically available from associated R package *MPV* (Braun, 2012), where it is labeled *table.b11*.

*Pitcher*    OBJECTS: 206 pitchers in North American Major League Baseball. CRITERION: annual salary at the beginning of the 1987 season. ATTRIBUTES: 1986 performance: wins, losses, earned run average, game appearances, innings pitched, games saved; career performance: wins, losses, earned run average, game appearances, innings pitched, games saved; years in major leagues; league at the end of 1986 (American, National); league at the beginning of the 1987 season (American, National). SOURCE: The data set was prepared by the Statistical Graphics Section of the American Statistical Association for the 1988 Annual Statistical Meetings and is available from StatLib (Meyer and Vlachos, 1989). The version used in this work is from Fox (2008) and is electronically available from a website maintained by the author (Fox, 2015).

*Plasma*    OBJECTS: 315 adults. CRITERION: Plasma retinol level. ATTRIBUTES: age, sex, body mass index, daily caloric intake, daily fat intake, daily fiber intake, daily cholesterol intake, dietary beta-carotene consumed per day, dietary retinol consumed per day, number of alcoholic drinks consumed per week, smoking status (never smoked, former smoker, current smoker), vitamin use (often, used but not often, not used). SOURCE: The data set was made available by Therese Stukel, Dartmouth Hitchcock Medical Center, at StatLib (Meyer and Vlachos, 1989), where it is labeled *Plasma_Retinol*. Dr. Stukel notes that a related publication is by Nierenberg et al. (1989).

*Prefecture*    OBJECTS: 45 prefectures in Japan. CRITERION: number of emigrants in the Pacific Northwest in 1911–1912 from the prefecture (per million of the prefecture's population). ATTRIBUTES: percentage of land cultivated by tenant farmlands, change in ratio of tenant farmlands between 1883 and 1907, average area of arable land per farm, number of government contracted laborers sent to Hawaii, whether any of the 18 pioneer Japanese immigrants to the Pacific Northwest were from the prefecture. SOURCE: The data are from an article by Murayama (1991). The data set is available electronically from an online repository maintained by Winner (2015), where the data set is described as *Japanese emigration to Pacific Northwest 1880–1915*.

*Prostate*    OBJECTS: 97 patients with prostate cancer. CRITERION: logarithm of prostate-specific antigen. ATTRIBUTES: log(cancer volume), log(prostate weight), age, log(amount of benign prostatic hyperplasia), seminal vesicle invasion, log(capsular penetration), Gleason score, percentage Gleason score 4 or 5. SOURCE: The data appears in Stamey et al. (1989). The data set is publicly available from the R package *lasso2* (Lokhorst et al., 2014), where it is labeled *Prostate*.

*Reactor*   OBJECTS: 32 light water reactors constructed in the United States in the late 1960s and early 1970s. CRITERION: construction cost. ATTRIBUTES: date on which the construction permit was issued (measured in years since January 1, 1900), time between application for and issue of the construction permit, time between issue of operating license and construction permit, net capacity, whether a prior light water reactor existed at the same site, whether the location is in the north-east region of the United States, Whether a cooling tower is used, whether the nuclear steam supply system was manufactured by Babcock-Wilcox, cumulative number of power plants constructed by each architect-engineer, whether there was a partial turnkey guarantee. SOURCE: The data set is reported by Cox and Snell (1981) and Davison (2003). It is electronically available from R package *SMPracticals* (Davison, 2013), where it is labeled *nuclear*.

*Rebellion*   OBJECTS: 32 Romanian counties in 1907. CRITERION: proportion of villages in which rebellious events took place in the Romanian peasant rebellion of 1907, labelled *spread*. ATTRIBUTES: proportion of arable land devoted to wheat, proportion of rural population that is illiterate, strength of middle peasantry (measured by the proportion of land owned in units of 7 to 50 hectares), Gini coefficient of inequality of landownership, population, region (Northern, South Central, Southwest, Eastern). SOURCE: The data set was assembled by Chirot and Ragin (1975). Partial data set is reported by Fox (2008) and is electronically available from a website maintained by the author (Fox, 2015).

*Recycle*   OBJECTS: 31 Scottish local authorities. CRITERION: weekly recyclate yield. ATTRIBUTES: weekly recycling capacity, weekly residual capacity, number of principal materials collected, number of extended materials collected, frequency of recycling collection, frequency of residual collection, type of sort (comingled, curbside sort, dual service, single material). SOURCE: The data were obtained by Baird et al. (2013) from Scottish local authorities. Partial data set is available electronically from an online repository maintained by Winner (2015), where the data set is described as *recycling capacity, items collected and average yield for Scottish local authorities*.

*Rent*   OBJECTS: 2053 apartments in Munich, Germany. CRITERION: rent per square-meter in euros. ATTRIBUTES: size, number of rooms, year of construction, whether the apartment is located at a good address, whether the apartment is located at the best address, whether the apartment has warm water, whether the apartment has central heating, whether the bathroom has tiles, whether there is special furniture in the bathroom, whether the apartment has an upmarket kitchen. SOURCE: The data set is reported in Fahrmeir et al. (2012) and is electronically available from R package *catdata* (Schauberger and Tutz, 2014).

*Salary*    OBJECTS: 52 professors at a Midwestern college in the United States. CRITERION: academic year salary. ATTRIBUTES: sex, rank (assistant professor, associate professor, full professor), number of years in current rank, the highest degree earned (doctorate, masters), number of years since highest degree was earned. SOURCE: The data set is reported by Weisberg (2005) and is electronically available from associated R package *alr3* (Weisberg, 2011).

*SAT*    OBJECTS: 50 US states. CRITERION: average total score on the SAT, 1994-95. ATTRIBUTES: avg. expenditure per pupil, avg. pupil/teacher ratio, avg. salary of teachers, percentage of eligible students. SOURCE: The data were collected by Guber (1999). The data set is electronically available from the R package *faraway* (Faraway, 2011).

*Schooling*    OBJECTS: 3010 individuals in the US. CRITERION: log of wage. ATTRIBUTES: lived in smsa 1966, lived in smsa in 1976, grew up near 2-yr college, grew up near 4-yr college, grew up near 4-year public college, grew up near 4-year private college, education in 1976, education in 1966, age in 1976, lived with mom and dad at age 14, single mom at 14, step parent at 14, lived in south 1966, lived in south in 1976, mom-dad education class (1-9), black, enrolled in 1976, the kww score, normed IQ score, married in 1976, library card in home at age 14, experience in 1976. SOURCE: The data set comes from the National Longitudinal Survey of Young Men (NLSYM) and has been used by Card (1993). It is available electronically from R package *Ecdat* (Croissant, 2013).

*Tip*    OBJECTS: 244 parties dining in a restaurant. CRITERION: tip rate. ATTRIBUTES: dollar amount of the bill, size of the party, sex of the bill payer, day of the week, time of the day, whether there were smokers in the party. SOURCE: Data were recorded by a food server in a restaurant located in a suburban shopping mall in the United States during an interval of two and a half months in early 1990. The data set is reported in a collection of case studies for business statistics (Bryant and Smith, 1995). It is electronically available from the R package *reshape* (Wickham, 2007).

*Vote*    OBJECTS: 159 counties in Georgia, USA. CRITERION: proportion of uncounted votes in the 2000 presidential election. ATTRIBUTES: type of voting equipment used (optical scan with central count, optical scan with precinct count, punch card, lever, paper), whether the county is in Atlanta, whether the county is urban or rural, proportion of African Americans, economic status (rich, middle, poor). SOURCE: The data set was assembled by Meyer (2002). It is reported by Faraway (2005) and is electronically available from the associated R package *faraway* (Faraway, 2011), where it is labeled *gavote*.

*Wages*    OBJECTS: 4360 males in the US (from 1980 to 1987). CRITERION: log of wage. ATTRIBUTES: year, years of schooling, years of experience, whether the

wage has been set by collective bargaining, ethnicity, whether married, whether health problem, industry (12 levels), occupation (9 levels), residence (rural area, north east, nothern central, south). SOURCE: The data set comes from the National Longitudinal Survey (NLS Youth Sample) and has been used by Vella and Verbeek (1998). It is available electronically from R package *Ecdat* (Croissant, 2013) where it is called *Males*.

*White wine*    OBJECTS: 4898 white wines. CRITERION: quality score (between 0 and 10). ATTRIBUTES: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol. SOURCE: This data set comes from a study by Cortez et al. (2009). It is available from the UCI Machine Learning Repository (Bache and Lichman, 2013).

# BIBLIOGRAPHY

Adler, Joseph (2010). *R in a nutshell: A desktop quick reference*. O'Reilly Media.

Adler, Joseph (2012). *nutshell: Data for "R in a nutshell"*. R package.

Ahlgren, John (2014). *The probability distribution for draws until first success without replacement*. arXiv: 1404.1161 [math.PR].

Algorta, Simón and Özgür Şimşek (2019). *The game of Tetris in machine learning*. arXiv: 1905.01652 [cs.LG].

Allison, Truett and Domenic V. Cicchetti (1976). "Sleep in mammals: Ecological and constitutional correlates". In: *Science* 194.4266, pp. 732–734.

Amarel, Saul (1968). "On representations of problems of reasoning about actions". In: *Machine Intelligence 3*, pp. 131–171.

Anderson, John R. and Robert Milson (1989). "Human memory: An adaptive perspective." In: *Psychological Review* 96.4, p. 703.

Bache, Kevin and Moche Lichman (2013). *UCI machine learning repository*. http://archive.ics.uci.edu/ml.

Baird, Jim, Robin Curry, and Tim Reid (2013). "Development and application of a multiple linear regression model to consider the impact of weekly waste container capacity on the yield from kerbside recycling programmes in Scotland". In: *Waste Management & Research* 31.3, pp. 306–314.

Barron, Francis H. and Bruce E. Barrett (1996). "Decision quality using ranked attribute weights". In: *Management Science* 42.11, pp. 1515–1523.

Baucells, Manel, Juan A. Carrasco, and Robin M. Hogarth (2008). "Cumulative dominance and heuristic performance in binary multiattribute choice". In: *Operations Research* 56.5, pp. 1289–1304.

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

Bellman, Richard (1957). *Dynamic programming*. Princeton University Press.

Belloni, Alexandre and Victor Chernozhukov (2013). "Least squares after model selection in high-dimensional sparse models". In: *Bernoulli* 19.2, pp. 521–547.

Bendor, Jonathan B., Sunil Kumar, and David A. Siegel (2009). "Satisficing: A 'pretty good' heuristic". In: *The BE Journal of Theoretical Economics* 9.1.

Bengio, Yoshua, Jérôme Louradour, Ronan Collobert, and Jason Weston (2009). "Curriculum learning". In: *Proceedings of the 26th International Conference on Machine Learning*, pp. 41–48.

Bobko, Philip, Philip L. Roth, and Maury A. Buster (2007). "The usefulness of unit weights in creating composite scores: A literature review, application to content validity, and meta-analysis". In: *Organizational Research Methods* 10.4, pp. 689–709.

Braun, Daniel A., Pedro A. Ortega, Evangelos Theodorou, and Stefan Schaal (2011). "Path integral control and bounded rationality". In: *2011 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (AD-PRL)*, pp. 202–209.

Braun, W. John (2012). *MPV: Data sets from Montgomery, Peck and Vining's book*. R package.

Breiman, Leo (2001). "Random forests". In: *Machine Learning* 45.1, pp. 5–32.

Breiman, Leo, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone (2017). *Classification and regression trees*. Routledge.

Brighton, Henry (2006). "Robust inference with simple cognitive models." In: *AAAI Spring Symposium: Between a Rock and a Hard Place: Cognitive Science Principles Meet AI-Hard Problems*, pp. 17–22.

Brighton, Henry (2020). "Statistical foundations of ecological rationality". In: *Economics* 14.1.

Brockman, Greg, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba (2016). *Openai gym*. arXiv: `1606.01540 [cs.LG]`.

Brooks, Thomas F., D. Stuart Pope, and Michael A. Marcolini (1989). *Airfoil self-noise and prediction*. National Aeronautics, Space Administration, Office of Management, Scientific, and Technical Information Division.

Brown, Stacey L., Joby Joseph, and Mark Stopfer (2005). "Encoding a temporally structured stimulus with a temporally structured neural representation". In: *Nature Neuroscience* 8.11, p. 1568.

Bryant, Peter G. and Marlene A. Smith (1995). *Practical data analysis: Case studies in business statistics*. Richard D. Irwin Publishing.

Buckmann, Marcus and Özgür Şimşek (2017). "Decision heuristics for comparison: How good are they?" In: *Imperfect Decision Makers: Admitting Real-World Rationality*. PMLR 58, pp. 1–11.

Caplin, Andrew, Mark Dean, and Daniel Martin (2011). "Search and satisficing". In: *American Economic Review* 101.7, pp. 2899–2922.

Card, David (1993). *Using geographic variation in college proximity to estimate the return to schooling*. Tech. rep. National Bureau of Economic Research.

Chambolle, Antonin (2004). "An algorithm for total variation minimization and applications". In: *Journal of Mathematical Imaging and Vision* 20.1, pp. 89–97.

Chandak, Yash, Georgios Theocharous, James Kostas, Scott Jordan, and Philip S. Thomas (2019). *Learning action representations for reinforcement learning*. arXiv: `1902.00183 [cs.LG]`.

Chirot, Daniel and Charles Ragin (1975). "The market, tradition and peasant rebellion: The case of Romania in 1907". In: *American Sociological Review*, pp. 428–444.

Chu, Singat (2001). "Pricing the C's of diamond stones". In: *Journal of Statistics Education* 9.2.

Claudy, John G. (1972). "A comparison of five variable weighting procedures". In: *Educational and Psychological Measurement* 32.2, pp. 311–322.

Cortez, Paulo, António Cerdeira, Fernando Almeida, Telmo Matos, and José Reis (2009). "Modeling wine preferences by data mining from physicochemical properties". In: *Decision Support Systems* 47.4, pp. 547–553.

Cox, David R. and E. Joyce Snell (1981). *Applied statistics: Principles and examples*. Chapman and Hall.

Crittenden, Ann (1975). "Vital dialogue is beginning between the rich and the poor". In: *The New York Times, September 28 1975*, E–3.

Croissant, Yves (2013). *Ecdat: Data sets for econometrics*. R package.

Cuzan, Alfred G. and Charles M. Bundrick (2009). "Predicting presidential elections with equally weighted regressors in Fair's equation and the fiscal model". In: *Political Analysis* 17.3, pp. 333–340.

Czerlinski, Jean, Gerd Gigerenzer, and Daniel G. Goldstein (1999). "How good are simple heuristics?" In: *Simple heuristics that make us smart*. Ed. by Gerd Gigerenzer, Peter M. Todd, and the ABC Research Group. Oxford University Press, pp. 97–118.

Dana, Jason and Robyn M. Dawes (2004). "The superiority of simple alternatives to regression for social science predictions". In: *Journal of Educational and Behavioral Statistics* 29.3, pp. 317–331.

Dana, Jason and Rick Thomas (2006). "In defense of clinical judgment... and mechanical prediction". In: *Journal of Behavioral Decision Making* 19.5, pp. 413–428.

Davis-Stober, Clintin P. (2011). "A geometric analysis of when fixed weighting schemes will outperform ordinary least squares". In: *Psychometrika* 76.4, pp. 650–669.

Davis-Stober, Clintin P., Jason Dana, and David V. Budescu (2010). "A constrained linear estimator for multiple regression". In: *Psychometrika* 75.3, pp. 521–541.

Davison, Anthony C. (2003). *Statistical models*. Cambridge University Press.

Davison, Anthony C. (2013). *SMPracticals: Practicals for use with Davison's (2003) 'Statistical models'*. R package.

Dawes, Robyn M. (1979). "The robust beauty of improper linear models in decision making." In: *American Psychologist* 34.7, p. 571.

Dawes, Robyn M. and Bernard Corrigan (1974). "Linear models in decision making". In: *Psychological Bulletin* 81.2, pp. 95–106.

DeMiguel, Victor, Lorenzo Garlappi, Francisco J. Nogales, and Raman Uppal (2009a). "A generalized approach to portfolio optimization: Improving per-

formance by constraining portfolio norms". In: *Management Science* 55.5, pp. 798–812.

DeMiguel, Victor, Lorenzo Garlappi, and Raman Uppal (2009b). "Optimal versus naive diversification: How inefficient is the 1/N portfolio strategy?" In: *Review of Financial Studies* 22.5, pp. 1915–1953.

Dean, Thomas L. and Mark S. Boddy (1988). "An analysis of time-dependent planning." In: *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*. Vol. 88, pp. 49–54.

Deb, Partha and Pravin K. Trivedi (2002). "The structure of demand for health care: Latent class versus two-part models". In: *Journal of Health Economics* 21.4, pp. 601–625.

Dodson, Stanley (1992). "Predicting crustacean zooplankton species richness." In: *Limnology and Oceanography* 37.4, pp. 848–856.

Domingos, Pedro (2000). "A unified bias-variance decomposition". In: *Proceedings of the 17th International Conference on Machine Learning*, pp. 231–238.

Dulac-Arnold, Gabriel, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin (2015). *Deep reinforcement learning in large discrete action spaces*. arXiv: 1512.07679 [cs.LG].

Dulac-Arnold, Gabriel, Daniel Mankowitz, and Todd Hester (2019). *Challenges of real-world reinforcement learning*. arXiv: 1904.12901 [cs.LG].

Edwards, David J., Gary D. Holt, and Frank C. Harris (2000). "A comparative analysis between the multilayer perceptron 'neural network' and multiple regression analysis for predicting construction plant maintenance costs". In: *Journal of Quality in Maintenance Engineering* 6.1, pp. 45–61.

Efron, Bradley and Trevor Hastie (2013). *lars: Least angle regression, Lasso and forward stagewise*. R package.

Efron, Bradley, Trevor Hastie, Iain Johnstone, and Robert Tibshirani (2004). "Least angle regression". In: *The Annals of Statistics* 32.2, pp. 407–499.

Ehrenberg, Andrew S. C. (1982). "How good is best?" In: *Journal of the Royal Statistical Society. Series A (General)* 145.3, p. 364.

Ehrlich, Isaac (1973). "Participation in illegitimate activities: A theoretical and empirical investigation". In: *Journal of Political Economy* 81, pp. 521–565.

Ein-Dor, Phillip and Jacob Feldmesser (1987). "Attributes of the performance of central processing units: A relative performance prediction model". In: *Communications of the ACM* 30.4, pp. 308–317.

Einhorn, Hillel J. and Robin M. Hogarth (1975). "Unit weighting schemes for decision making". In: *Organizational Behavior and Human Performance* 13.2, pp. 171–192.

Ekstrom, Claus T. and Helle Sørensen (2010). *Introduction to statistical data analysis for the life sciences*. CRC Press.

Ekstrom, Claus T. and Helle Sørensen (2014). *isdals: Provides datasets for "Introduction to statistical data analysis for the life sciences"*. R package.

Elman, Jeffrey L. (1993). "Learning and development in neural networks: The importance of starting small". In: *Cognition* 48.1, pp. 71–99.

Elster, Jon (1977). "Ulysses and the sirens: A theory of imperfect rationality". In: *Social Science Information* 16.5, pp. 469–526.

Etzioni, Oren (1989). "Tractable decision-analytic control." In: *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning* 89, pp. 114–125.

Fahrmeir, Ludwig, Rita Künstler, Iris Pigeot, and Gerhard Tutz (2012). *Statistik: Der weg zur datenanalyse*. Springer Berlin Heidelberg.

Farahmand, Amir-massoud, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor (2008). "Regularized policy iteration." In: *Advances in Neural Information Processing Systems*, pp. 441–448.

Farahmand, Amir-massoud, Mohammad Ghavamzadeh, Csaba Szepesvári, and Shie Mannor (2009). "Regularized fitted Q-iteration for planning in continuous-space Markovian decision problems". In: *2009 American Control Conference*. IEEE, pp. 725–730.

Faraway, Julian (2005). *Extending linear models with R: Generalized linear, mixed effects and nonparametric regression models*. Chapman & Hall/CRC.

Faraway, Julian (2011). *faraway: Functions and datasets for books by Julian Faraway*. R package.

Farebrother, Jesse, Marlos C. Machado, and Michael Bowling (2018). *Generalization and regularization in DQN*. arXiv: 1810.00123 [cs.LG].

Fern, Alan, SungWook Yoon, and Robert Givan (2004). "Approximate policy iteration with a policy language bias". In: *Advances in Neural Information Processing Systems*, pp. 847–854.

Fernandes, Kelwin, Pedro Vinagre, and Paulo Cortez (2015). "A proactive intelligent decision support system for predicting the popularity of online news". In: *Portuguese Conference on Artificial Intelligence*, pp. 535–546.

Fernández-Delgado, Manuel, Eva Cernadas, Senén Barro, and Dinani Amorim (2014). "Do we need hundreds of classifiers to solve real world classification problems?" In: *Journal of Machine Learning Research* 15.1, pp. 3133–3181.

Fishburn, Peter C. (1974). "Lexicographic orders, utilities and decision rules: A survey". In: *Management Science* 20.11, pp. 1442–1471.

Fox, John (2008). *Applied regression analysis and generalized linear models*. 2nd. SAGE Publications.

Fox, John (2015). *Data sets for 'Applied regression analysis and generalized linear models, second edition'*. http://socserv.socsci.mcmaster.ca/jfox/Books/Applied-Regression-2E/datasets/.

Franklin, Benjamin (1987). *Writings*. Library of America.

Friedman, Jerome H. (1997). "On bias, variance, 0/1—loss, and the curse-of-dimensionality". In: *Data Mining and Knowledge Discovery* 1.1, pp. 55–77.

Friedman, Jerome H., Trevor Hastie, Noah Simon, and Rob Tibshirani (2015). *glmnet: Lasso and elastic-net regularized generalized linear models*. R package.

Friedman, Jerome H., Trevor Hastie, and Robert Tibshirani (2001). *The elements of statistical learning*. Vol. 1. Springer.

Gabillon, Victor, Mohammad Ghavamzadeh, and Bruno Scherrer (2013). "Approximate dynamic programming finally performs well in the game of Tetris". In: *Advances in Neural Information Processing Systems*, pp. 1754–1762.

Galesic, Mirta, Daniel Barkoczi, and Konstantinos V. Katsikopoulos (2018). "Smaller crowds outperform larger crowds and individuals in realistic task conditions". In: *Decision* 5.1, p. 1.

Genewein, Tim, Felix Leibfried, Jordi Grau-Moya, and Daniel A. Braun (2015). "Bounded rationality, abstraction, and hierarchical decision-making: An information-theoretic optimality principle". In: *Frontiers in Robotics and AI* 2, p. 27.

Gershman, Samuel J., Eric J. Horvitz, and Joshua B. Tenenbaum (2015). "Computational rationality: A converging paradigm for intelligence in brains, minds, and machines". In: *Science* 349.6245, pp. 273–278.

Gigerenzer, Gerd and Henry Brighton (2009). "Homo heuristicus: Why biased minds make better inferences". In: *Topics in Cognitive Science* 1.1, pp. 107–143.

Gigerenzer, Gerd and Daniel G. Goldstein (1996). "Reasoning the fast and frugal way: Models of bounded rationality." In: *Psychological Review* 103, pp. 650–669.

Gigerenzer, Gerd, Ralph Hertwig, and Thorsten Pachur (2011). *Heuristics: The foundations of adaptive behavior*. Oxford University Press.

Gigerenzer, Gerd and Reinhard Selten (2002). *Bounded rationality: The adaptive toolbox*. MIT Press.

Gigerenzer, Gerd, Peter M. Todd, and the the ABC Research Group (1999). *Simple heuristics that make us smart*. Oxford University Press.

Goldberg, Lewis R. (1972). "Parameters of personality inventory construction and utilization: A comparison of prediction strategies and tactics." In: *Multivariate Behavioral Research Monographs*.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep learning*. Vol. 1. MIT Press.

Graefe, Andreas (2015). "Improving forecasts using equally weighted predictors". In: *Journal of Business Research* 68.8, pp. 1792–1799.

Grau-Moya, Jordi (2016). "Decision-making under bounded rationality and model uncertainty: An information-theoretic approach". PhD Thesis. Eberhard Karls Universität Tübingen.

Graves, Alex, Marc G. Bellemare, Jacob Menick, Rémi Munos, and Koray Kavukcuoglu (2017). "Automated curriculum learning for neural networks". In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 1311–1320.

Greene, William H. (2003). *Econometric analysis*. Pearson Education India.

Greene, William H. (2015). *Online data archive*. http://pages.stern.nyu.edu/~wgreene/Text/econometricanalysis.htm.

Guber, Deborah C. (1999). "Getting what you pay for: The debate over equity in public school expenditures". In: *Journal of Statistics Education* 7.2.

Gupta, Abhishek, Russell Mendonca, YuXuan Liu, Pieter Abbeel, and Sergey Levine (2018). *Meta-reinforcement learning of structured exploration strategies*. arXiv: 1802.07245 [cs.LG].

Hand, David J., Fergus Daly, D. Lunn, Kevin J. McConway, and E. Ostrowski (1994). *A handbook of small data sets*. Chapman & Hall/CRC.

Hansen, Nikolaus and Andreas Ostermeier (2001). "Completely derandomized self-adaptation in evolution strategies". In: *Evolutionary Computation* 9.2, pp. 159–195.

Helversen, Bettina von and Jörg Rieskamp (2008). "The mapping model: A cognitive theory of quantitative estimation." In: *Journal of Experimental Psychology: General* 137.1, pp. 73–96.

Henss, Ronald (1996). "The attractiveness of prominent people". Unpublished manuscript at the Fachrichtung Psychologie, University of Saarbrücken.

Hertwig, Ralph, Ulrich Hoffrage, and Laura Martignon (1999). "Quick estimation". In: *Simple heuristics that make us smart*. Ed. by Gerd Gigerenzer, Peter M. Todd, and the ABC Research Group. Oxford University Press, pp. 209–234.

Hoaglin, David C. and Paul F. Velleman (1995). "A critical look at some analyses of Major League Baseball salaries". In: *American Statistician* 49.4266, pp. 277–285.

Hoerl, Arthur E. and Robert W. Kennard (1970). "Ridge regression: Biased estimation for nonorthogonal problems". In: *Technometrics* 12.1, p. 55.

Hogarth, Robin M. and Natalia Karelaia (2005). "Ignoring information in binary choice with continuous variables: When is less 'more'?" In: *Journal of Mathematical Psychology* 49.2, pp. 115–124.

Holte, Robert C. (1993). "Very simple classification rules perform well on most commonly used datasets". In: *Machine Learning* 11.1, pp. 63–90.

Horvitz, Eric J. (1987). "Reasoning about beliefs and actions under computational resource constraints". In: *Proceedings of the Third Conference on Uncertainty in Artificial Intelligence*, pp. 429–447.

Hosmer, David W. and Stanley Lemeshow (2000). *Applied logistic regression*. Wiley.

Howell, David C. (2009). *Statistical methods for psychology*. Cengage Learning.

Ide-Smith, Susan G. and Stephen E. G. Lea (1988). "Gambling in young adolescents". In: *Journal of Gambling Behavior* 4.2, pp. 110–118.

Jarvis, David R. (1997). "Nitrogen levels in long bones from coffin burials interred for periods of 26–90 years". In: *Forensic Science International* 85.3, pp. 199–208.

Kahneman, Daniel and Amos Tversky (2013). "Prospect theory: An analysis of decision under risk". In: *Handbook of the fundamentals of financial decision making: Part I*. World Scientific, pp. 99–127.

Katsikopoulos, Konstantinos V. (2011). "Psychological heuristics for making inferences: Definition, performance, and the emerging theory and practice". In: *Decision Analysis* 8.1, pp. 10–29.

Katsikopoulos, Konstantinos V. (2014). "Bounded rationality: The two cultures". In: *Journal of Economic Methodology* 21.4, pp. 361–374.

Katsikopoulos, Konstantinos V., Ian N. Durbach, and Theodor J. Stewart (2018). "When should we use simple decision models? A synthesis of various research strands". In: *Omega* 81, pp. 17–25.

Katsikopoulos, Konstantinos V. and Laura Martignon (2006). "Naive heuristics for paired comparisons: Some results on their relative accuracy". In: *Journal of Mathematical Psychology* 50.5, pp. 488–494.

Katsikopoulos, Konstantinos V., Lael J. Schooler, and Ralph Hertwig (2010). "The robust beauty of ordinary information." In: *Psychological Review* 117.4, p. 1259.

Katsikopoulos, Konstantinos V., Özgür Şimşek, Markus Buckmann, and Gerd Gigerenzer (2020). *Classification in the wild*. MIT Press.

Klein, Gary, Steve Wolf, Laura Militello, and Caroline Zsambok (1995). "Characteristics of skilled option generation in chess". In: *Organizational Behavior and Human Decision Processes* 62.1, pp. 63–69.

Krueger, Kai A. and Peter Dayan (2009). "Flexible shaping: How learning in small steps helps". In: *Cognition* 110.3, pp. 380–394.

Kuhn, Harold W. and Albert W. Tucker (2014). "Nonlinear programming". In: *Traces and Emergence of Nonlinear Programming*. Ed. by G. Giorgi and T. Kjeldsen. Birkhäuser, Basel, pp. 247–258.

Labovitz, Sanford (1970). "The assignment of numbers to rank order categories". In: *The American Sociological Review* 35, pp. 515–524.

Lagoudakis, Michail G. and Ronald Parr (2003). "Reinforcement learning as classification: Leveraging modern classifiers". In: *Proceedings of the 20th International Conference on Machine Learning*, pp. 424–431.

Laughlin, James E. (1978). "Comment on estimating coefficients in linear models: It don't make no nevermind." In: *Psychological Bulletin* 85.2, pp. 247–253.

Lazaric, Alessandro, Mohammad Ghavamzadeh, and Rémi Munos (2016). "Analysis of classification-based policy iteration algorithms". In: *Journal of Machine Learning Research* 17.1, pp. 583–612.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton (2015). "Deep learning". In: *Nature* 521.7553, pp. 436–444.

Leinhardt, Samuel and Stanley S. Wasserman (1979). "Exploratory data analysis: An introduction to selected methods". In: *Sociological Methodology* 10, pp. 311–365.

Lewin, Kurt, Tamara Dembo, Leon Festinger, and Pauline S. Sears (1944). "Level of aspiration." In: *Personality and the behavior disorders*. Ed. by J. M. Hunt. Ronald Press.

Li, Lihong, Vadim Bulitko, and Russell Greiner (2007). "Focus of attention in reinforcement learning." In: *Journal of Universal Computer Science* 13.9, pp. 1246–1269.

Liaw, Andy and Matthew Wiener (2002). "Classification and regression by randomForest". In: *R news* 2.3, pp. 18–22.

Lichtenberg, Jan M. and Özgür Şimşek (2017). "Simple regression models". In: *Imperfect Decision Makers: Admitting Real-World Rationality*. PMLR 58, pp. 13–25.

Lichtenberg, Jan M. and Özgür Şimşek (2019a). *Iterative policy-space expansion in reinforcement learning*. arXiv: 1912.02532 [cs.LG].

Lichtenberg, Jan M. and Özgür Şimşek (2019b). "Regularization in directable environments with application to Tetris". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 3953–3962.

Lillicrap, Timothy P., Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). *Continuous control with deep reinforcement learning*. arXiv: 1509.02971 [cs.LG].

Lock, Robin H. (1993). "1993 New car data". In: *Journal of Statistics Education* 1.1.

Lokhorst, Justin, Bill Venables, Berwin Turlach, and Martin Maechler (2014). *lasso2: L1 constrained estimation aka 'lasso'*. R package.

Loth, Manuel, Manuel Davy, and Philippe Preux (2007). "Sparse temporal difference learning using LASSO". In: *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 352–359.

Maindonald, John and W. John Braun (2010). *Data analysis and graphics using R: An example-based approach*. Cambridge University Press.

Maindonald, John and W. John Braun (2013). *DAAG: Data analysis and graphics data and functions*. R package.

Mandal, Baidya N. and Jun Ma (2016). *nnlasso: Non-negative Lasso and Elastic Net penalized generalized linear models*. R package.

Martignon, Laura and Ulrich Hoffrage (1999). "Why does one-reason decision making work". In: *Simple heuristics that make us smart*. Ed. by Gerd Gigerenzer, Peter M. Todd, and the ABC Research Group, pp. 119–140.

Martignon, Laura and Ulrich Hoffrage (2002). "Fast, frugal, and fit: Simple heuristics for paired comparison". In: *Theory and Decision* 52.1, pp. 29–71.

Mas-Colell, Andreu, Michael D. Whinston, and Jerry R. Green (1995). *Microeconomic theory*. Vol. 1. Oxford University Press.

McDonald, Gary C. and Richard C. Schwing (1973). "Instabilities of regression estimates relating air pollution to mortality". In: *Technometrics* 15, pp. 463–482.

Meinshausen, Nicolai (2013). "Sign-constrained least squares estimation for high-dimensional regression". In: *Electronic Journal of Statistics* 7, pp. 1607–1631.

Meyer, Mary C. (2002). "Uncounted votes: Does voting equipment matter?" In: *Chance* 15.4, pp. 33–38.

Meyer, Mike and Pantelis Vlachos (1989). *StatLib*. http://lib.stat.cmu.edu/.

Miller, Earl K. and Jonathan D. Cohen (2001). "An integrative theory of prefrontal cortex function". In: *Annual Review of Neuroscience* 24.1, pp. 167–202.

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, and Georg Ostrovski (2015). "Human-level control through deep reinforcement learning". In: *Nature* 518.7540, p. 529.

Montgomery, Douglas C., Elizabeth A. Peck, and G. Geoffrey Vining (2001). *Introduction to linear regression analysis*. 3rd. Wiley.

Morton, F. B. (1995). "Charting a school's course". In: *Chicago*, pp. 86–95.

Murayama, Yuzo (1991). "Information and emigrants: Interprefectural differences of Japanese emigration to the Pacific Northwest, 1880–1915". In: *The Journal of Economic History* 51.1, pp. 125–147.

Narvekar, Sanmit, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone (2020). *Curriculum learning for reinforcement learning domains: A framework and survey*. arXiv: 2003.04960 [cs.LG].

Nash, Warwick J., Tracy L. Sellers, Simon R. Talbot, Andrew J. Cawthorn, and Wes B. Ford (1994). "The population biology of abalone (haliotis species) in Tasmania. I. Blacklip Abalone (h. rubra) from the north coast and islands of Bass Strait". In: *Sea Fisheries Division, Technical Report* 48.

Nierenberg, David W., Therese A. Stukel, John A. Baron, Bradley J. Dain, and E. Robert Greenberg (1989). "Determinants of plasma levels of beta-carotene and retinol". In: *American Journal of Epidemiology* 130.3, pp. 511–521.

Oaksford, Mike and Nick Chater (2007). *Bayesian rationality: The probabilistic approach to human reasoning*. Oxford University Press.

Ortega, Pedro A. (2011). "A unified framework for resource-bounded autonomous agents interacting with unknown environments". PhD thesis. University of Cambridge.

Ortega, Pedro A., Daniel A. Braun, Justin Dyer, Kee-Eung Kim, and Naftali Tishby (2015). *Information-theoretic bounded rationality*. arXiv: 1512.06789 [stat.ML].

Parpart, Paula, Matt Jones, and Bradley C. Love (2018). "Heuristics as Bayesian inference under extreme priors". In: *Cognitive Psychology* 102, pp. 127–144.

Penrose, Keith W., A. G. Nelson, and A. G. Fisher (1985). "Generalized body composition prediction equation for men using simple measurement techniques". In: *Medicine & Science in Sports & Exercise* 17.2, p. 189.

Peterson, Gail B. (2004). "A day of great illumination: BF Skinner's discovery of shaping". In: *Journal of the Experimental Analysis of Behavior* 82.3, pp. 317–328.

R Core Team (2015). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing.

Radner, Roy (1975). "Satisficing". In: *Optimization Techniques IFIP Technical Conference*. Springer, pp. 252–263.

Riedmiller, Martin (2005). "Neural fitted Q iteration—first experiences with a data efficient neural reinforcement learning method". In: *European Conference on Machine Learning*, pp. 317–328.

Rieskamp, Jörg and Philipp E. Otto (2006). "SSL: A theory of how people learn to select strategies." In: *Journal of Experimental Psychology: General* 135.2, p. 207.

Ripley, Brian, Bill Venables, Douglas M. Bates, Kurt Hornik, Albrecht Gebhardt, and David Firth (2013). *MASS: Support functions and datasets for Venables and Ripley's MASS*. R package.

Rocke, Aidan (2019). *The true cost of AlphaGo Zero*. URL: https://keplerlounge.com/artificial/intelligence/2019/03/24/alpha-go-zero.html.

Rodkin, D (1995). "10 Keys for creating top high schools". In: *Chicago*, pp. 78–85.

Russell, Stuart J. and Peter Norvig (2010). *Artificial intelligence: A modern approach*. Prentice Hall Press.

Russell, Stuart J. and Devika Subramanian (1994). "Provably bounded-optimal agents". In: *Journal of Artificial Intelligence Research* 2, pp. 575–609.

Russell, Stuart J. and Eric Wefald (1991). "Principles of metareasoning". In: *Artificial Intelligence* 49.1-3, pp. 361–395.

Russo, Daniel and Benjamin Van Roy (2018). *Satisficing in time-sensitive bandit learning*. arXiv: 1803.02855 [cs.LG].

Sanger, Terence D. (1994). "Neural network learning control of robot manipulators using gradually increasing task difficulty". In: *IEEE Transactions on Robotics and Automation* 10.3, pp. 323–333.

Sargent, Thomas J. (1993). *Bounded rationality in macroeconomics: The Arne Ryde memorial lectures*. Oxford University Press.

Sauermann, Heinz and Reinhard Selten (1962). "Anspruchsanpassungstheorie der Unternehmung". In: *Zeitschrift für die gesamte Staatswissenschaft/Journal of Institutional and Theoretical Economics* 4, pp. 577–597.

Savage, Leonard J. (1972). *The foundations of statistics*. Courier Corporation.

Schauberger, Gunther and Gerhard Tutz (2014). *catdata: Categorical data*. R package.

Scherrer, Bruno, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist (2015). "Approximate modified policy iteration and its application to the game of Tetris." In: *Journal of Machine Learning Research* 16, pp. 1629–1676.

Schmidt, Frank L. (1971). "The relative efficiency of regression and simple unit predictor weights in applied differential psychology". In: *Educational and Psychological Measurement* 31.3, pp. 699–714.

Selten, Reinhard (1998). "Aspiration adaptation theory". In: *Journal of Mathematical Psychology* 42.2-3, pp. 191–214.

Sepkoski, J. John and Michael A. Rex (1974). "Distribution of freshwater mussels: Coastal rivers as biogeographic islands". In: *Systematic Biology* 23.2, pp. 165–188.

Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis (2017a).

*Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm*. arXiv: `1712.01815 [cs.AI]`.

Silver, David, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, and Adrian Bolton (2017b). "Mastering the game of Go without human knowledge". In: *Nature* 550.7676, p. 354.

Simon, Herbert A. (1956). "Rational choice and the structure of the environment." In: *Psychological Review* 63.2, p. 129.

Simon, Herbert A. (1957). *Models of man*. J. Wiley and Sons.

Simon, Herbert A. (1959). "Theories of decision-making in economics and behavioral science". In: *American Economic Review* 49.3, pp. 253–283.

Simon, Herbert A. (1997). *Models of bounded rationality: Empirically grounded economic reason*. Vol. 3. MIT Press.

Simonoff, Jeffrey S. (2003). *Analyzing categorical data*. New York: Springer.

Simonoff, Jeffrey S. (2015). *Online data archive*. `http://people.stern.nyu.edu/jsimonof/SmoothMeth/`.

Şimşek, Özgür (2008). "Behavioral building blocks for autonomous agents: Description, identification, and learning". PhD thesis. University of Massachusetts Amherst.

Şimşek, Özgür (2013). "Linear decision rule as aspiration for simple decision heuristics". In: *Advances in Neural Information Processing Systems*, pp. 2904–2912.

Şimşek, Özgür (2020a). "Bounded rationality for artificial intelligence". In: *Routledge handbook of bounded rationality*. Ed. by Riccardo Viale. Routledge, pp. 338–348.

Şimşek, Özgür (2020b). "Lexicographic decision rule". In: *Oxford research encyclopedia of politics*. Oxford University Press.

Şimşek, Özgür, Simon Algorta, and Amit Kothiyal (2016). "Why most decisions are easy in Tetris–And perhaps in other sequential decision problems, as well". In: *Proceedings of the 33rd International Conference on Machine Learning*, pp. 1757–1765.

Şimşek, Özgür and Marcus Buckmann (2015). "Learning from small samples: An analysis of simple decision heuristics". In: *Advances in Neural Information Processing Systems*, pp. 3141–3149.

Şimşek, Özgür and Marcus Buckmann (2017). "On learning decision heuristics". In: *Imperfect Decision Makers: Admitting Real-World Rationality*. PMLR 58, pp. 75–85.

Skinner, Burrhus F. (1958). "Reinforcement today." In: *American Psychologist* 13.3, p. 94.

Slawski, Martin and Matthias Hein (2013). "Non-negative least squares for high-dimensional linear models: Consistency and sparse recovery without regularization". In: *Electronic Journal of Statistics* 7, pp. 3004–3056.

Smyth, Gordon K. (2011). *Australasian data and story library (OzDASL)*. `http://www.statsci.org/data`.

Sokal, Robert R. and F. James Rohlf (1981). *Biometry: The principles and practice of statistics in biological research*. 2nd. W. H. Freeman and Company.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: a simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1, pp. 1929–1958.

Stamey, Thomas A., John N. Kabalin, John E. McNeal, Iain M. Johnstone, Fuad Freiha, Elise A. Redwine, and Norman Yang (1989). "Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. II. Radical prostatectomy treated patients." In: *Journal of Urology* 141.5, pp. 1076–1083.

Stigler, George J. (1961). "The economics of information". In: *Journal of Political Economy* 69.3, pp. 213–225.

Stock, James H. and Mark W. Watson (2003). *Introduction to econometrics*. Addison Wesley Boston.

Stüttgen, Peter, Peter Boatwright, and Robert T. Monroe (2012). "A satisficing choice model". In: *Marketing Science* 31.6, pp. 878–899.

Sutton, Richard S. and Andrew G. Barto (2018). *Reinforcement learning: An introduction*. MIT Press.

Tager, Ira B., Scott T. Weiss, Bernard Rosner, and Frank E. Speizer (July 1979). "Effect of parental cigarette smoking on the pulmonary function of children". In: *American Journal of Epidemiology* 110.1, pp. 15–26.

Telford, Richard D. and Ross B. Cunningham (1991). "Sex, sport, and body-size dependency of hematology in highly trained athletes". In: *Medicine & Science in Sports & Exercise* 23, pp. 788–794.

Tennenholtz, Guy and Shie Mannor (2019). *The natural language of actions*. arXiv: 1902.01119 [cs.AI].

Tesauro, Gerald (1992). "Practical issues in temporal difference learning". In: *Advances in Neural Information Processing Systems*, pp. 259–266.

Tesauro, Gerald (2002). "Programming backgammon using self-teaching neural nets". In: *Artificial Intelligence* 134.1-2, pp. 181–199.

Thiery, Christophe and Bruno Scherrer (2009a). "Building controllers for Tetris". In: *Icga Journal* 32.1, pp. 3–11.

Thiery, Christophe and Bruno Scherrer (2009b). "Improvements on learning Tetris with cross entropy". In: *Icga Journal* 32.1, pp. 23–33.

Thomaz, Andrea L. and Cynthia Breazeal (2006). "Reinforcement learning with human teachers: Evidence of feedback and guidance with implications for learning performance". In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 1000–1005.

Thrun, Sebastian B. and Knut Möller (1991). "Active exploration in dynamic environments". In: *Advances in Neural Information Processing Systems*, pp. 531–538.

Thrun, Sebastian and Anton Schwartz (1995). "Finding structure in reinforcement learning". In: *Advances in Neural Information Processing Systems*, pp. 385–392.

Tibshirani, Robert (1996). "Regression shrinkage and selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288.

Tibshirani, Robert, Michael Saunders, Saharon Rosset, Ji Zhu, and Keith Knight (2005). "Sparsity and smoothness via the fused lasso". In: *Journal of the Royal Statistical Society Series B*, pp. 91–108.

Tibshirani, Ryan J. and Jonathan Taylor (2011). "The solution path of the generalized lasso". In: *The Annals of Statistics* 39.3, pp. 1335–1371.

Tikhonov, Andrei N., A. V. Goncharsky, V. V. Stepanov, and Anatoly G. Yagola (2013). *Numerical methods for the solution of ill-posed problems*. Springer.

Todd, Peter M. and Henry Brighton (2016). "Building the theory of ecological rationality". In: *Minds and Machines* 26.1-2, pp. 9–30.

Todd, Peter M., Gerd Gigerenzer, and the the ABC Research Group (2012). *Ecological rationality: Intelligence in the world*. Oxford University Press.

Train, Kenneth (2009). *Discrete choice methods with simulation*. 2nd. Cambridge University Press.

Tucker, W. (1987). "Where do the homeless come from?" In: *National Review*, pp. 34–44.

Tuddenham, Read D. and Margaret M. Snyder (1954). "Physical growth of California boys and girls from birth to eighteen years." In: *University of California Publications in Child Development* 1.2, p. 183.

Tversky, Amos and Daniel Kahneman (1973). "Availability: A heuristic for judging frequency and probability". In: *Cognitive Psychology* 5.2, pp. 207–232.

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). "Deep reinforcement learning with double q-learning". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1.

Vandaele, Walter (1978). "Participation in illegitimate activities: Ehrlich revisited". In: *Deterrence and Incapacitation*. Ed. by A. Blumstein, J. Cohen, and D. Nagin. National Academy of Sciences, pp. 270–335.

Vella, Francis and Marno Verbeek (1998). "Whose wages do unions raise? A dynamic model of unionism and wage rate determination for young men". In: *Journal of Applied Econometrics* 13.2, pp. 163–183.

Venables, Bill N. and Brian D. Ripley (2002). *Modern applied statistics with S*. Fourth. Springer.

Von Neumann, John and Oskar Morgenstern (1944). *Theory of games and economic behavior*. Princeton University Press.

Wainer, Howard (1976). "Estimating coefficients in linear models: It don't make no nevermind". In: *Psychological Bulletin* 83.2, pp. 213–217.

Wainer, Howard (1978). "On the sensitivity of regression and regressors." In: *Psychological Bulletin* 85.2, pp. 267–273.

Waller, Niels G. and Jeff A. Jones (2009). "Correlation weights in multiple regression". en. In: *Psychometrika* 75.1, pp. 58–69.

Weisberg, Sanford (2005). *Applied linear regression*. 3rd. Wiley.

Weisberg, Sanford (2011). *alr3: Data to accompany Applied Linear Regression 3rd edition*. R package.

Wesman, Alexander G. and George K. Bennett (1959). "Multiple regression vs. simple addition of scores in prediction of college grades." In: *Educational and Psychological Measurement*.

Wickham, Hadley (2007). "Reshaping data with the reshape package". In: *Journal of Statistical Software* 21.12.

Wilks, Samuel S. (1938). "Weighting systems for linear functions of correlated variables when there is no dependent variable". In: *Psychometrika* 3.1, pp. 23–40.

Winner, Larry (2015). *Online data archive*. http://www.stat.ufl.edu/~winner/datasets.html.

Winter, Sidney G. (1971). "Satisficing, selection, and the innovating remnant". In: *The Quarterly Journal of Economics* 85.2, pp. 237–261.

Woike, Jan K., Ulrich Hoffrage, and Ralph Hertwig (2012). "Estimating quantities: Comparing simple heuristics and machine learning algorithms". In: *Artificial Neural Networks and Machine Learning–ICANN 2012*, pp. 483–490.

Wu, Lan, Yuehan Yang, and Hanzhong Liu (2014). "Nonnegative-lasso and application in index tracking". In: *Computational Statistics & Data Analysis* 70, pp. 116–126.

Xu, Xin, Dewen Hu, and Xicheng Lu (2007). "Kernel-based least squares policy iteration for reinforcement learning". In: *IEEE Transactions on Neural Networks* 18.4, pp. 973–992.

Yeh, I-Cheng (1998). "Modeling of strength of high-performance concrete using artificial neural networks". In: *Cement and Concrete Research* 28.12, pp. 1797–1808.

Zilberstein, Shlomo (1995). "Operational rationality through compilation of anytime algorithms". In: *AI Magazine* 16.2, pp. 79–79.

Zilberstein, Shlomo (2008). "Metareasoning and bounded rationality". In: *Proceedings of the AAAI Workshop on Metareasoning: Thinking about Thinking*.

Ziliak, James P. (1997). "Efficient estimation with panel data when instruments are predetermined: An empirical comparison of moment-condition estimators". In: *Journal of Business & Economic Statistics* 15.4, pp. 419–431.

Zou, Hui and Trevor Hastie (2005). "Regularization and variable selection via the Elastic Net". In: *Journal of the Royal Statistical Society, Series B* 67, pp. 301–320.