University of Alberta

Learning and Classifying under Hard Budgets

by

Aloak Kapoor ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta
Fall 2005

Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

0-494-09203-3

Your file Votre référence
ISBN:
Our file Notre retérence
ISBN:

University of Alberta

Library Release Form

Name of Author: Aloak Kapoor

Title of Thesis: Learning and Classifying under Hard Budgets

Degree: Master of Science

Year this Degree Granted: 2005

Date: ____________________

Abstract

When learning a classifier for a function $Y = f(X)$, the features, X, often have an associated cost. Since resources for feature acquisition are usually finite, learners and classifiers must be able to act intelligently under hard budgets. In this thesis, the goal is a learner that spends its fixed learning budget b_L acquiring features of labelled training instances so as to produce the most accurate "active classifier" that spends at most b_C per instance. To produce this fixed budget classifier, the fixed budget learner must sequentially decide which feature values to collect to learn the relevant information about the distribution. We explore several approaches the learner can take, ranging from Reinforcement Learning techniques, to the obvious "round robin" strategy that spends equally on all features. We show empirically that round robin is problematic (especially for small b_L), and provide alternate learning strategies that achieve superior performance on a variety of datasets.

Acknowledgements

I would like to thank my supervisor, Russell Greiner, for his constant support, guidance, and seemingly endless supply of good ideas. I am a better person for having worked with Russ the last year. As well, this thesis could not have been written without the generous financial support provided by NSERC and iCORE throughout my graduate career. Finally, I wish to thank the three most outstanding people I know: my mother, father, and sister.

Table of Contents

1 Introduction 1

2 Active Model Selection 3
- 2.1 Introduction and Motivation . . . 3
- 2.2 Formal Description . . . 4
  - 2.2.1 Simplifying Assumptions . . . 6
  - 2.2.2 An Example Policy . . . 6
  - 2.2.3 Mapping to Budgeted Learning . . . 7
- 2.3 The Markov Decision Process Formulation . . . 7
  - 2.3.1 Value Functions . . . 8
  - 2.3.2 Simple Results . . . 8
- 2.4 Existing Algorithms . . . 9
  - 2.4.1 The Optimal Policy . . . 9
  - 2.4.2 Round Robin (RR) . . . 9
  - 2.4.3 Biased Robin (BR) . . . 9
  - 2.4.4 Single Coin Lookahead (SCL) . . . 10
- 2.5 Reinforcement Learning Background . . . 11
  - 2.5.1 Overview . . . 11
  - 2.5.2 Learning versus Planning . . . 11
  - 2.5.3 Learning the Optimal Value Function Using Temporal Differences . . . 11
  - 2.5.4 The Need for Function Approximation . . . 13
  - 2.5.5 Tile Coding: a Linear Function Approximator . . . 14
- 2.6 Adapting RL for Active Model Selection . . . 15
- 2.7 Empirical Results . . . 16
- 2.8 Unsuccessful Approaches . . . 19
  - 2.8.1 Supervised Learning of a Classifier . . . 19
  - 2.8.2 Search . . . 19
  - 2.8.3 Optimal Two Coin (OTC) . . . 19
- 2.9 Summary . . . 20

3 Budgeted Learning a Bounded Active Classifier 21
- 3.1 Introduction . . . 21
  - 3.1.1 A Motivating Example . . . 21
  - 3.1.2 Objective and Outline . . . 21
- 3.2 Background: Bounded Active Classifiers . . . 22
  - 3.2.1 Definition of an Active Classifier . . . 22
  - 3.2.2 Bounding Active Classifiers . . . 22
- 3.3 Formal Description . . . 24
  - 3.3.1 Simplifying Assumptions . . . 24
  - 3.3.2 Complexity Results . . . 25
- 3.4 The MDP Formulation . . . 25
  - 3.4.1 The Optimal Learning Policy . . . 27
- 3.5 Heuristic Learning Policies . . . 28
  - 3.5.1 Round Robin (RR) . . . 28
  - 3.5.2 Biased Robin (BR) . . . 29
  - 3.5.3 Single Feature Lookahead (SFL) . . . 29
  - 3.5.4 Randomized SFL (RSFL) . . . 29
- 3.6 Loss Functions . . . 30
- 3.7 Experimental Results . . . 31
- 3.8 Summary . . . 33

4 Related Literature 34
4.1 Introduction . . . 34
4.2 Budgeted Learning a Bounded Active Classifier . . . 34
4.3 Active Model Selection . . . 36

5 Conclusions 37
5.1 Contributions . . . 37
5.2 Research Directions . . . 38

Bibliography 40

A Proofs 42
A.1 Proposition 1 . . . 42
A.2 Proposition 2 . . . 43
A.3 Proposition 3 . . . 44
A.4 Proposition 4 . . . 44
A.5 Proposition 5 . . . 45

B Features for RL Function Approximation 47
B.1 Feature Groups . . . 47
B.2 Alternate Features . . . 48

List of Tables

2.1 Free parameters in ϵ–greedy, tile coding TD(λ) 15
2.2 Feature sets used for approximating the value function 16
2.3 Expected regret of various policies 16
2.4 Resources used by each policy on n=10, b=20 18

3.1 Reduction in computation time using Proposition 5 28

B.1 Other features tested for RL function approximation 49

List of Figures

2.1 An example of a policy for $b = 2$, $n = 2$, and uniform priors 7
2.2 A problematic state for the SCL policy 10
2.3 An example of tile coding in two dimensions 14
2.4 Various values of lambda — SCL still superior to RL 17
2.5 Various amounts of training — simple policies still superior to RL 18

3.1 An example of an active classifier when the features and class label are binary . . . 23
3.2 Identical costs and some irrelevant features — RSFL and BR outperform RR . . . 31
3.3 Identical costs, no irrelevant features — RR still suboptimal 32
3.4 Different feature costs — RSFL and BR dominate RR 32

4.1 Active learning versus budgeted learning 35

Chapter 1

Introduction

In classification learning, the goal is to learn a classifier for an unknown function $Y = f(\mathrm{X})$ such that the classifier can predict the class label, Y, when given the features, X. In many practical applications, the features are initially unknown to both the learner and the classifier, and must be acquired at a cost. In these cases, an *active classifier*, that can actively purchase the values of unknown features before making a classification, should be produced by the learner. Unfortunately, resources are seldom infinite; real-world tasks typically have finite budgets for both the learner and the classifier that limit the total value of features that can be collected. Thus, when feature costs exist, the machine learning researcher is faced with the following budgeted learning problem:

Given a pool of training instances with known class labels but unknown feature values, decide how to spend the fixed learning budget b_L purchasing features of training instances so as to produce the most accurate active classifier that can spend at most b_C per instance.

We refer to this problem as "budgeted learning a bounded active classifier". In this thesis, we investigate the aforementioned problem in detail. We concentrate on developing strategies for the learner that sequentially select which feature to purchase given the remaining b_L budget and the results of the previous purchases. Developing an effective spending strategy can be challenging because the true utility of the learner's purchases is not known until the b_L budget is exhausted and the final bounded active classifier is learned and applied. Although the topic of budgeted learning is not entirely new [17, 18], our work is unique because it places bounds on both the learner and the classifier and thus incorporates costs at training and testing time. By contrast, the previous budgeted learning research considers only costs at training time, and allows the classifier to see all feature values for free. The dual budget framework we consider in this thesis is a better model of many real-world problems.

We begin our investigation in the next chapter, in which we take a simplified version of our problem and allow the learner to use Reinforcement Learning (RL) techniques to learn a purchasing policy. We demonstrate empirically that despite extensive training, the RL methods that we employ are inferior to simple, heuristic policies. In Chapter 3, we explore the full problem of budgeted learning a bounded active classifier. We provide empirical evidence that the obvious round robin spending policy (purchasing every feature of every instance until the b_L budget is exhausted) is problematic, particularly when the budget is small relative to the number of features. We describe alternate learning strategies, and show that they significantly outperform round robin on a variety of real-world datasets. Finally, Chapter 4 provides a survey of related literature, while Chapter 5 summarizes contributions and discusses future work. We note that versions of Chapter 2, 3, and 4 have been published [15, 14].[1]

[1] ACM and co-author Russell Greiner kindly grant permission to reuse material in [15]. The use of material in [14] is granted with kind permission of Springer Science and Business Media and co-author Russell Greiner.

Chapter 2

Simplified Budgeted Learning: Active Model Selection

2.1 Introduction and Motivation

To gain insight into the budgeted learning issue, we consider a simpler problem known as *active model selection*.[1] Loosely speaking, active model selection involves finding the best object among a set of n, given a finite budget of probes with which to freely explore and test the objects, where a probe of an object returns a sample value drawn from that object's distribution. After the budget is exhausted, a single object must be selected, and a one-time reward is received that represents the expected value of the chosen object. This formulation allows for pure exploration of the objects with the budget, and delays all reward until the final time step. Notice that this problem corresponds to the training phase of budgeted learning, in which features of labelled training instances can be purchased in any way, with a single one-time reward (i.e., the classification accuracy) being received once the budget is exhausted and the final learned classifier is applied. In both the active model selection problem and the budgeted learning problem, the goal is to decide how to spend a finite number of probes in order to get the information required to make the best decision when the budget is exhausted.

In addition to the above relationship, previous research [17] has shown that algorithms which perform well on active model selection are also effective on a variant of our budgeted learning problem. As a result, we use active model selection as a low-dimensional testbed to prototype the performance of strategies for budgeted learning.

In the remainder of this chapter, we give the formal description of active model selection, and show that the problem can be viewed as a Markov Decision Process (MDP). The MDP framework allows us to describe the (intractable) optimal algorithm, and derive some new results about the problem. The main contribution of the chapter is to investigate the performance of standard algorithms from Reinforcement Learning on active model selection. We perform a variety of tests using

[1] We also refer to active model selection as the "coins problem" for reasons that will become clear during the formal problem statement in Section 2.2.

Reinforcement Learning techniques, and show that better performance is achieved with less computational effort using simpler, existing policies. In closing, we discuss other approaches to active model selection that appear promising, but are also inferior to the existing heuristic policies.

2.2 Formal Description

The input to the active model selection problem is:

- A set of n independent Bernoulli random variables $\{C_1, \ldots, C_n\}$ with unknown success probabilities. For simplicity of exposition, we can think of these C_i as a set of coins, where the unknown success probability is the probability of the coin turning up heads when flipped.
- A set of n prior distributions (i.e., density functions), indicating the uncertainty over the true head probability of each of the n coins. That is, the *head probability* of each coin C_i is itself treated as a random variable Z_i, and a prior density function $f_i(Z_i)$ is provided as a distribution over the possible head probabilities of coin C_i.
- A set of n (known) costs $\{S(C_1), \ldots, S(C_n)\}$ for flipping the coins, where $S(C_i) \in \Re^+$.
- A finite (known) budget $b \geq 0$ that can be spent flipping the coins.

Given these inputs, the active model selection problem proceeds as follows. Any coin C_i can be flipped at any time, as long as the remaining budget, denoted by b', satisfies $b' \geq S(C_i)$. We use the outcome of each coin flip to update the density function for the flipped coin. For example, if coin C_i is flipped and turns up heads, then its density function is updated to $f_i(Z_i|C_i = heads)$; of course, a similar update occurs for a tails outcome. (We describe the exact format of the density function and the updates in our simplifying assumptions below.) Coin flips and density updates continue until the budget is exhausted ($b' = 0$). We can view the sequence of flips and updates as a learning period, in which we improve our information about the true head probabilities of the coins. Once the budget is exhausted, the learning period is over, and a single coin must be chosen — this coin C^* (and only this coin) will be used in all future flips, for which we will receive rewards for head outcomes. Of course, even when $b' = 0$, we will still not *know* the true head probability for this (or any) coin, and so will not know whether coin C^* actually has a better head probability than the other coins. The best we can do given the observed coin flip outcomes $\mathbf{o}$, is to choose the coin that minimizes our future regret of selecting it. To do this, we define a new random variable Z_{max} to be the maximum head probability over all of the coins: $Z_{max} = max_i(Z_i)$, and now the Bayesian regret of choosing coin C_i given coin flip outcomes $\mathbf{o}$ is:

$$\text{Regret}(C_i) = \int_{\vec{Z}} (Z_{max} - Z_i) \prod_{j=1}^{n} f_j(Z_j|\mathbf{o}) \; d\vec{Z} \qquad (2.1)$$

Notice that we minimize regret by choosing the coin whose mean (posterior) head probability is largest [19]. Let this maximum mean coin be $C^*(\mathbf{o}) = \arg\max_{C_i} E[Z_i|\mathbf{o}]$. Thus, when the budget is exhausted and coin flip outcomes $\mathbf{o}$ have been observed, $C^*(\mathbf{o})$ *should* be selected.

Before introducing the overall (regret-related) objective function we wish to minimize, we must first introduce the notion of a policy. A policy π for active model selection specifies which coin to flip at each time step. Formally, a policy is a mapping $\pi: \langle b', f_1(Z_1), \ldots, f_n(Z_n) \rangle \rightarrow [1, n]$ that specifies the index of the coin to flip, given the current state defined by the remaining budget and the posterior distributions over the coins. Since the result of every coin flip is stochastic, a policy for flipping the coins can result in *several* different "outcome" states in which the budget is exhausted. Thus, a policy π for active model selection is scored based on its expected regret:

$$\mathrm{ER}(\pi) \quad = \sum_{\mathbf{o} \in outcomes(\pi)} P(\mathbf{o})\ \mathrm{Regret}(C^*(\mathbf{o})) \tag{2.2}$$

where the sum is over the various "outcomes" of the policy when the budget b' has been exhausted. The objective of active model selection is to find the optimal policy π^* that minimizes Equation 2.2.

As mentioned earlier, since regret can be minimized by choosing the coin with the highest expected head probability, an alternate (equivalent) way to score a policy π is to calculate the expected$_1$ maximum expected$_2$ head probability (EMEHP) of the chosen coin:

$$\mathrm{EMEHP}(\pi) \quad = \sum_{\mathbf{o} \in outcomes(\pi)} P(\mathbf{o})\ \max_i \{E(Z_i|\mathbf{o})\}\ . \tag{2.3}$$

Note that both "expected" are required as the first expectation$_1$ is over possible outcomes of the policy, while the second expectation$_2$ is over the head probability distribution of the chosen coin. Under this EMEHP score, the objective of active model selection is to find the optimal policy π^* that maximizes Equation 2.3. Since maximizing head probability is more intuitive than minimizing Bayesian regret, maximizing Equation 2.3 is usually an easier objective to remember for active model selection.

The two objective functions, Equations 2.2 and 2.3, consider the probability of reaching an outcome state in which the budget has been exhausted. Each outcome state corresponds to seeing some non-negative number of heads and tails on the coins in the set. A benefit of using the Bayesian formulation is that the probability of reaching *any* state is well defined. Specifically, the probability of reaching a state can be computed using the prior density functions over the coins and the posterior densities that result after each coin flip outcome. For example, if we let $f_i(Z_i|p_h, q_t)$ denote the posterior density over coin C_i's head probability after observing p heads and q tails on C_i, then the probability of seeing the outcome where coin C_i turns up heads twice followed by a tail on coin C_j is:

$$E(Z_i|f_i(Z_i)) \ \times\ E(Z_i|f_i(Z_i|1_h)) \ \times\ [1 - E(Z_j|f_j(Z_j))] \tag{2.4}$$

Thus, at any point in time, we use the expected head probability of a coin as a point estimate of the current probability of that coin turning up heads when flipped. Calculating transitions in this way, we can compute the probability of reaching any state using strictly the density functions $\{f_1(Z_1), \ldots, f_n(Z_n)\}$.

2.2.1 Simplifying Assumptions

Coin C_i's head probability is represented as a random variable $Z_i \in [0,1]$. We assume that Z_i is a Beta random variable with density function $f_i(Z_i) = W(\alpha,\beta) \ (Z_i)^{\alpha-1}(1-Z_i)^{\beta-1}$ (here $W(\alpha,\beta)$ is a normalizing constant and α and β are two positive hyperparameters that define the Beta distribution). For a Beta(α,β) distribution, the mean is $\mu = \frac{\alpha}{\alpha+\beta}$ while the variance is $\sigma^2 = \frac{\mu(1-\mu)}{\alpha+\beta+1}$. Loosely speaking, when α (β) is much larger than β (α), it means that a coin is likely to have a high (low) head probability. On the other hand, when both α and β are 1, the distribution over head probabilities is uniform. As we have an independent Beta distribution for each coin, we use α_i and β_i to denote the specific hyperparameters for coin C_i.

One attractive property of the Beta distribution is that it is computationally simple to calculate posterior densities. If coin C_i's initial head probability distribution is $Beta(\alpha_i, \beta_i)$, then after observing p heads and q tails on coin C_i, its posterior density is just $f_i(Z_i|p_h, q_t) = Beta(\alpha_i+p, \beta_i+q)$. Thus, the Beta hyperparameters can be viewed as simple frequency counts for a random variable with two possible outcomes.

Although the formal description allows for any coin costs, we will assume that the costs are uniform: $S(C_i) = 1 \ \forall i$, and that the budget b is a positive integer. Finally, as we are studying active model selection because of its relationship to budgeted learning, we are typically interested in values of b that are not much greater than n (typically $b = n \times k$, with k a small positive integer), as most budgeted learning algorithms will act reasonable when b is much larger than n. In fact, in the case where b is very large relative to n, even a simple policy (e.g. purchasing every feature of every instance) will yield a training set that can produce an accurate classifier, and so these scenarios are not of great interest from a budgeted learning point of view.

2.2.2 An Example Policy

Figure 2.1 shows an example of a policy for a two coin problem with identical Beta(1,1) priors, a budget of two, and uniform coin costs $S(C_i) = 1$. Each transition in the policy is labelled with its probability of occurrence, and the Beta densities over the coins are updated after each transition. Here left branches correspond to head outcomes and right branches correspond to tail outcomes. Notice that the policy is contingent, as the coin that is flipped on the second time step depends on the outcome of coin C_1's initial flip. The policy in Figure 2.1 has four outcome states corresponding to the leaves of the tree, and has an EMEHP of $\frac{7}{12}$, which can be verified using Equation 2.3.

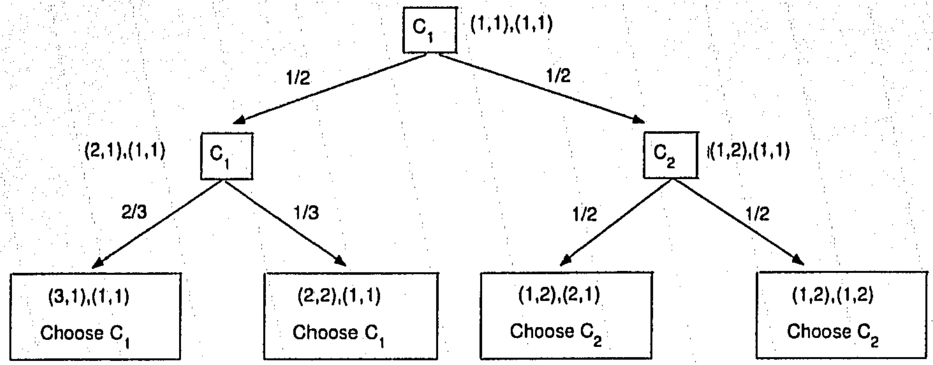


Figure 2.1: An example of a policy for $b = 2$, $n = 2$, and uniform priors

2.2.3 Mapping to Budgeted Learning

As mentioned in Section 2.1, active model selection is highly related to budgeted learning because it mimics the pure exploration phase (i.e., purchasing features of labelled training data), followed by the one-time reward phase (i.e., the classification accuracy of the final learned classifier). In addition to this relationship, we can also show that optimal active model selection is equivalent to optimal budgeted learning of a bounded active classifier, with some assumptions. Specifically, given a binary class Y and n binary features $\{R_i\}_{i=1..n}$ for a classification task, assume $P(R_i = 1|Y = 0) = 0 \ \forall i$, let all features have unit cost, and assume the bounded active classifier can collect only one feature (i.e., $b_C = 1$). Then the best feature to use for the bounded active classifier is: $\arg\max_{R_i} P(R_i = 1|Y = 1)$. Set coin C_i to be feature R_i, Z_i to be $P(R_i = 1|Y = 1)$, and let flipping coin C_i be equivalent to purchasing feature R_i on a random $Y = 1$ instance. Then a policy π^* that maximizes the expected head probability of the chosen coin (Equation 2.3) also maximizes the expected accuracy of the chosen bounded active classifier.

2.3 The Markov Decision Process Formulation

Active model selection can be formulated as a finite Markov Decision Process [25] consisting of a set of states S, a set of actions A, a reward function R, and a transition function T. Specifically, we identify a state $s \in S$ of the MDP by the remaining budget b', and by the collection of Beta hyperparameters over the coins. That is, a state is a $2n + 1$ element vector of the form: $\langle b', \alpha_1, \beta_1, \ldots, \alpha_n, \beta_n \rangle$. The complete set of reachable states corresponds to all the possible posterior Beta distributions that can occur over the n coins by spending some portion m of the original budget b, with $m \leq b$. Since no more actions can be taken once the budget is exhausted, the terminal states are those in which $b' = 0$. In general we denote the starting state by s_0, and denote the state encountered on the ith time step by s_i.

The set of actions in the MDP corresponds to the n different coins that can be flipped, where action $a_{c_i} \in A$ denotes flipping coin C_i. The reward function $R(s,a,s')$ specifies the reward of taking action a from state s and reaching state s'. For the coins problem, the reward received when reaching any non-terminal state (i.e. where the remaining budget is positive) is zero, while the reward at a terminal state is the maximum expected head probability over the coins.[2] We use r_i to denote the immediate reward received on the ith time step.[3]

In many MDPs, the reward at future time steps is valued less than immediate reward, and so a discount factor $\gamma \leq 1$ is used to multiply future rewards to reduce their value. In the coins problem, future rewards are no less valued than immediate rewards (in fact the *only* reward that matters is the one received on the last time step), and so we have $\gamma = 1$ in our MDP formulation. Finally, the transition function $T(s,a,s')$ specifies the probability of reaching state s' after taking action a from state s. Due to our Bayesian formulation, $T(s,a,s')$ is conveniently given by the Beta distributions over the coins. For example, $T(Beta(4,2), a_{C_i}, Beta(5,2))$ is just the probability of coin C_i turning up heads: $P(C_i = \text{heads}) = E(Z_i) = 4/6$. As the transition function specifies probabilities, we often use $P(s,a,s')$ in place of $T(s,a,s')$.

2.3.1 Value Functions

An advantage of the MDP formulation is that the true long-term value of states can be quantified using a *value function*. Specifically, a value function $V^\pi : S \to \Re$ for a policy π measures the total expected reward accumulated from any state s_t when following π:

$$V^\pi(s_t) = E\left(\sum_{i=0}^{\infty}(\gamma^i r_{i+t+1} | \pi, s_t)\right) \quad . \tag{2.5}$$

With the value function notation, we know that a state s is preferable to a state s' if we can achieve greater expected reward from s when following an optimal policy: $V^{\pi^*}(s) > V^{\pi^*}(s')$. Given this relationship, we often use the value function notation to compare values of different states.

2.3.2 Simple Results

Using the value function notation of the previous section, we can derive the following intuitive properties concerning active model selection. Both results can be obtained using induction on the budget (and proofs can be found in Appendix A). These results are helpful because they can be used to establish an upper or lower bound on the optimal value of a state s using the optimal value of a related state s'. In addition, these results can be used as starting points for deriving more complex properties of the coins problem (e.g., Proposition 2 can be extended to relate states that have fewer than $n-1$ matching coins).

[2] This choice of reward function assumes we are using the EMEHP objective as in Equation 2.3. We could also use the expected regret objective in Equation 2.2, and this would change our reward function to give Regret($C^*(o)$) at the terminal states. Since it is easier to think of maximizing rather than minimizing rewards, our EMEHP-based reward function is usually more intuitive than the regret version.

[3] With the understanding that $r_i = 0$ for any time step i that is past a terminal state, since such a state can never be reached.

Proposition 1 A head is always better than a tail. *Assume all coins have unit cost, let s be any non-terminal state, and assume some coin C_i is flipped in s. If s^{+h_i} denotes the next state in which a head outcome is observed, and s^{+t_i} denotes the next state in which a tail outcome is observed, then $V^{\pi^*}(s^{+h_i}) \geq V^{\pi^*}(s^{+t_i})$.*

Proposition 2 The more heads the better. *Given any state* s: *$(b', \alpha_1, \beta_1, \ldots \alpha_i, \beta_i, \ldots \alpha_n, \beta_n)$, consider another state* ŝ : *$(b', \alpha_1, \beta_1, \ldots \alpha_i + 1, \beta_i, \ldots \alpha_n, \beta_n)$ which is identical to* s *except that one additional head has been observed on coin C_i. Then, $V^{\pi^*}(\hat{s}) \geq V^{\pi^*}(s)$.*

2.4 Existing Algorithms

2.4.1 The Optimal Policy

Since the coins problem is an MDP, several techniques can be used to solve for the optimal policy exactly [28]. For example, a bottom-up dynamic program can use the Bellman optimality equation to learn V^{π^*}, the expected value of each state under an optimal policy:

$$V^{\pi^*}(s) = \max_a \sum_{s'} P(s, a, s')[R(s, a, s') + \gamma V^{\pi^*}(s')] \tag{2.6}$$

Beginning at the next-to-end states in which $b' = 1$ and performing a backward sweep toward the initial state where $b' = b$, the optimal value function V^{π^*} can be completely determined. With the known transition and reward functions, the optimal policy π^* then follows immediately via greedy one-step lookahead. Unfortunately, the state space of active model selection grows exponentially with b and n, making it intractable to compute the optimal policy using exact methods such as dynamic programming. A natural alternative is to perform approximate dynamic programming via Reinforcement Learning, which we consider in detail in Section 2.6. Although Reinforcement Learning has not been applied to the coins problem previously, [18] has considered some simple heuristic policies which we review next.

2.4.2 Round Robin (RR)

The most intuitive spending policy is to allocate flips evenly over the coins, proceeding in a round-robin fashion. When $b = n \times k$ for an integer k, and all coins have unit cost, RR will flip each of the n coins k times. Despite its fair distribution of flips, the ratio of RR's expected regret to the optimal policy's expected regret can be made arbitrarily large [19]. Fortunately, more effective policies than RR are known.

2.4.3 Biased Robin (BR)

The BR algorithm repeatedly flips a coin C_i until a tail outcome occurs. Once a tail is observed, BR moves to the next coin, C_{i+1}, and repeats the process. (Of course when the last coin C_n turns up tails, BR moves back to the first coin C_1.) This simple algorithm is well known in statistics as

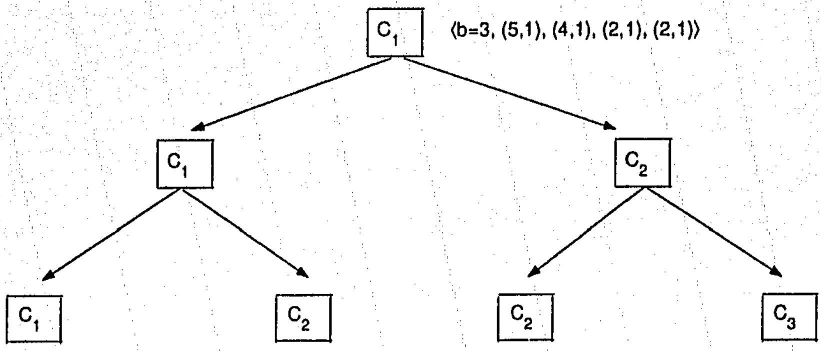


Figure 2.2: The optimal policy for the state $\langle b = 3, (5,1), (4,1), (2,1), (2,1)\rangle$ under unit coin costs. Notice that the optimal policy involves interactions between three of the four coins (e.g., coin C_2 should be flipped after C_1 turns up tails, and coin C_3 should be flipped after C_2 turns up tails). Since the SCL score for C_1 does not consider how flipping C_2 or C_3 could help C_1, it underestimates the value of flipping C_1, and SCL takes a suboptimal action from this state.

"Play the Winner" [23] and has been previously studied as a sampling method for clinical trials [13]. Its performance on the coins problem has been very strong in the case of identical starting priors. Despite its competitive performance, BR is a suboptimal policy. In fact, we can show that the number of states from which BR takes a suboptimal action can be made arbitrarily large:[4]

Proposition 3 *Given any positive integer* g $\geq$ 1, *there exists a problem with* n=(g+2) *Beta*$(1,1)$ *coins, and budget* b=(2n+3) *such that the BR policy takes a suboptimal action from at least* g *states.*

2.4.4 Single Coin Lookahead (SCL)

The SCL algorithm computes the EMEHP (Equation 2.3) of the policy that devotes *all* remaining flips in the budget to a single coin C_i. The coin that yields the policy with highest EMEHP is flipped *once*, and then SCL repeats the previous calculation with its reduced budget (and updated density functions) to choose the next coin to flip. Like BR, SCL has strong performance, but is still suboptimal. In particular, SCL suffers in situations where multiple coins must interact heavily to produce the optimal policy. This occurs because SCL computes a score for coin C_i without considering how the remaining $n-1$ coins could interact with C_i to improve its policy. To make this concrete, Figure 2.2 provides an example of a state where SCL takes a suboptimal action because it does not consider interactions among its coins. These deficiencies in the simple strategies offered by RR, BR and SCL motivate the need for a more robust policy that we consider next.

[4] Although such a result may help in proving the non-approximability of BR, it does not show non-approximability by itself. The reason is that the number of suboptimal actions is made arbitrarily large, but the probability of reaching states in which these actions occur is not considered.

2.5 Reinforcement Learning Background

The MDP formulation of the coins problem brings with it the possibility of using Reinforcement Learning techniques to develop effective spending policies. This section provides a brief introduction to RL, with a focus on the RL techniques that we employ later in this chapter when attempting to learn low-regret policies. We direct the interested reader to [28] for more details on any of the techniques discussed here.

2.5.1 Overview

Reinforcement Learning is a collection of techniques for learning (optimal) behaviour in sequential decision problems. In RL, an agent interacts directly with its environment and receives signals of reward as it takes actions. The goal is to develop a policy for taking actions that maximizes expected reward. The key characteristic that distinguishes RL from other learning methods (e.g. supervised, semi-supervised) is that the agent learns *on its own* by taking actions and directly observing the resulting rewards that are produced by the environment. With no explicit teacher or labelled training examples required, Reinforcement Learning is bounded mainly by the amount of environmental interaction available to the agent.

2.5.2 Learning versus Planning

A common distinction made in RL is between *learning* methods and *planning* methods. Planning methods require a known environment model (i.e. known transition and reward functions) and operate on simulated experience from this model. On the other hand, learning methods do not know the true environment model. Instead, they learn from "real" experience that they observe while acting in real-time in their environments. (Since the transition probabilities and the rewards are known in the active model selection task, we are faced with an RL planning problem.) An advantage of the planning problem is that experience is inexpensive to generate. Using only the model, large amounts of training episodes can be generated for the RL agent to test actions in. Furthermore, since the optimal policy can be defined in terms of the optimal value function:

$$\pi^*(s) = \arg\max_a \sum_{s'} P(s,a,s')[R(s,a,s') + \gamma V^{\pi^*}(s')] \tag{2.7}$$

the RL agent only needs to learn the optimal value function — with the known environment dynamics ($P(s,a,s')$ and $R(s,a,s')$), it can then calculate the optimal policy using greedy one-step lookahead.

2.5.3 Learning the Optimal Value Function Using Temporal Differences

There are many methods for learning the optimal value function, including value iteration, Monte Carlo methods, and temporal difference learning [28]. We focus on temporal difference learning

[27] in this thesis. Temporal difference learning is applicable to multi-step prediction problems in which the target value to be learned is observed gradually, as partial values comprising the target become available over time (just as rewards gradually accumulate in an MDP). The basis of temporal difference learning in an MDP is to shift the existing value estimate for a state s toward the newly observed values that occur over time. For example, suppose we take action a from state s_i, reach state s_{i+1} and produce an immediate reward r_{i+1}. In this case, a particular temporal difference learner, known as TD(0), uses the learning rule:

$$V(s_i) \leftarrow V(s_i) + \alpha(r_{i+1} + \gamma V(s_{i+1}) - V(s_i)) \tag{2.8}$$

to adjust its estimate of $V(s_i)$, with α a parameter controlling the learning rate. The TD(0) rule adjusts its estimate toward the one-step return, observed one step after leaving state s_i. By contrast, the general temporal difference algorithm, known as TD(λ), considers all j-step returns (for $j \in [1, \infty)$) that are observed j-steps after leaving state s_i. To incorporate all j-step returns in a simple, on-line fashion, TD(λ) augments the one-step return in Equation 2.8 with an *eligibility trace*. Specifically, TD(λ) maintains a positive, real-valued eligibility trace g_s for each state s that indicates how recently s was visited. (Intuitively, at the beginning of an episode, all eligibility traces are initialized to zero). By maintaining this eligibility trace, TD(λ) retains a record of which states have been visited previously and are therefore eligible to receive some credit for the current one-step return. Thus, after taking action a from state s_i and observing next state s_{i+1} and reward r_{i+1}, TD(λ) performs the following learning sequence for *all* states $s \in S$:

$$g_s \leftarrow g_s + 1 \qquad \text{if } s = s_i \tag{2.9}$$

$$V(s) \leftarrow V(s) + \alpha(r_{i+1} + \gamma V(s_{i+1}) - V(s_i))g_s \tag{2.10}$$

$$g_s \leftarrow \lambda\gamma g_s \tag{2.11}$$

Here $\lambda \in [0, 1]$ is a real-valued parameter that controls how the various j-step backups are averaged together. Notice that when $\lambda = 0$, all weight is assigned to the one-step backup, and the TD(λ) equations reduce to the simpler TD(0) learning rule. On the other hand, setting λ to an intermediate value such as 0.7 will assign some weight to each of the observed j-step returns, so that at the end of an episode, the value estimate for an observed state will have been adjusted toward a weighted sum of all j-step returns observed after that state.

To learn the value function for a policy π, an RL agent can use temporal difference learning while it experiences episodes of the MDP. For example, an RL agent can take actions according to π, and update its value function using a temporal difference learning rule after each state transition. This process of updates continues over multiple episodes of the MDP, gradually improving the value function estimate for π.

Under appropriate technical assumptions [28], the TD(0) rule (and the general TD(λ) algorithm) will converge to V^π for any policy π given that an RL agent chooses its actions according to π. In particular, to learn the value function for π, the TD learning updates should be distributed according to the states that would be encountered while following π. As stated previously, we are interested in learning the value function for the optimal policy π^*. Since this policy is *unknown*, an RL agent cannot act with it directly to generate the appropriate distribution of TD updates. However, it can still learn the optimal value function by acting according to a policy that is greedy in the limit of infinite exploration (GLIE) [25]. A GLIE policy performs every action from every state an infinite amount of times but reduces to a greedy policy in the limit. Since all actions are explored from every state, when a GLIE policy gets greedy in the limit, it is guaranteed to be an optimal policy π^*. Thus, an RL agent following a GLIE policy and using TD(λ) is guaranteed to learn the optimal value function V^{π^*} in the limit. Fortunately, a temporal difference learner with a GLIE-type policy can converge to V^{π^*} in a finite number of episodes in practice (see [28] for examples).

2.5.4 The Need for Function Approximation

The TD rules discussed so far assume that the value function is tabular, permitting exact representation of the value of every state in the state space. When state spaces are extremely large, however, it is impractical computationally to assume the RL agent can properly explore all states, and store a full tabular value function in memory. The standard solution is to utilize a function approximator to represent the value function, thereby allowing for an update to the value of state s to affect the value of other similar states. With a well-constructed function approximator, a value function over a large state space may be learned by visiting only a fraction of the total number of states in the space. The tabular temporal difference rules (from the previous section) can be re-derived to specify an update to a parameterized function rather than to a single tabular value. For instance, consider the popular linear function approximator:

$$V(s) = \vec{\theta} \cdot \vec{d_s} \tag{2.12}$$

where $\vec{\theta}$ is a vector of (learnable) parameters, and $\vec{d_s}$ is a vector of features for state s. For this linear function approximator, the TD(0) learning rule is

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha[r_{i+1} + \gamma V(s_{i+1}) - V(s_i)]\vec{d_{s_i}} \ . \tag{2.13}$$

On the other hand, the general TD(λ) algorithm maintains a vector of eligibility traces $\vec{g}$ (one trace for each learnable parameter), and its learning sequence for the linear function approximator is:

$$\vec{g} \leftarrow \lambda\gamma\vec{g} + \vec{d_{s_i}} \tag{2.14}$$

$$\vec{\theta} \leftarrow \vec{\theta} + \alpha[r_{i+1} + \gamma V(s_{i+1}) - V(s_i)]\vec{g} \tag{2.15}$$

Just as in the tabular case, these TD learning rules are applied after each transition from the current state s_i to the next state s_{i+1}. We next describe a specific linear function approximator that is often used in RL.

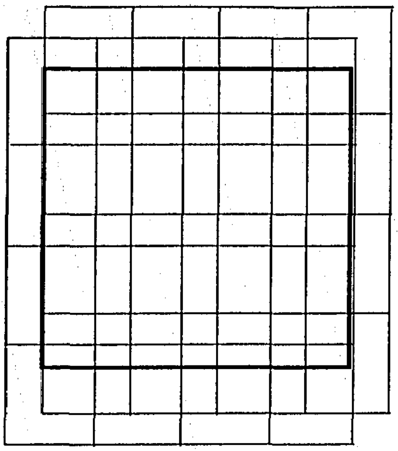


Figure 2.3: An example of tile coding over a two dimensional feature space. The feature space is outlined in bold, and two different tilings cover the space, with the position of each tiling offset by a small amount.

2.5.5 Tile Coding: a Linear Function Approximator

In tile coding, a group of tilings (i.e., grids) are laid over a feature space, with each tiling consisting of a set of h cells. The tilings are identical in size, but each tiling's position is offset by a small amount, so that each tiling covers the feature space in a different way (see Figure 2.3). Each cell h_i contains a real-valued (learnable) parameter θ_i. The value of a state s is formed by a linear combination over all the cells:

$$V(s) = \sum_i \theta_i d_{s_i} \tag{2.16}$$

where the coefficient d_{s_i} for cell i is 1 if state s is located in the cell, and 0 otherwise. Since the feature vector $\vec{d_s}$ for state s consists entirely of ones and zeros, the value of s is just the sum of the cell-values θ_i for all cells i which contain s.

The cell-values are modified by learning rules (such as Equation 2.13) as the RL agent acts in its environment. Moreover, since states that are nearby in feature space will occupy some of the same cells, these learning rules will adjust the value function estimate for several related states at once. As we expect states which are nearby in feature space to have similar value function estimates, this generalization can greatly speed up learning in large state spaces.

An advantage of tile coding is that there is great degree of flexibility in controlling how generalization occurs. For example, generalization can be controlled by the set of features used to represent the states, the number of different tilings laid over the space, as well as the shape and size of the individual cells. One can even choose to use several different feature sets for tile coding, and thus have a separate tile coding for each feature set. This requires laying a separate group of tilings over each one of these feature spaces. In this case, the value of a state is formed by summing all cell-values that contain the state, across all the different tile codings.

Table 2.1: Free parameters in ϵ–greedy, tile coding TD(λ)

Parameter	Description
α	step-size for learning
ϵ	exploration probability
λ	weighting of n-step returns
γ	discount factor
α–sched	schedule to decrease α
ϵ–sched	schedule to decrease ϵ
$\vec{d_s}$	features in function approximation
tile–shape	dimensions of each tile
num–tilings	density of tiles

2.6 Adapting RL for Active Model Selection

To apply RL to the coins problem, we attempt to learn the optimal value function with several separate RL agents. Each agent uses a unique set of features for function approximation (described in detail below), and gains the necessary experience by acting in a large number of simulated episodes generated from the known environment model. Each agent uses tile coding as its function approximation method, and employs a TD(λ) learner using an epsilon-greedy (GLIE-type) policy. Combining TD(λ) with a linear function approximator (such as tile coding) is attractive because upper bounds have been established on the mean squared error of the learned value function, under appropriate assumptions [29]. As noted in the RL background sections, the number of free variables that must be manually set for a TD(λ) tile coding agent is extensive. Table 2.1 contains a complete listing of these free variables. When designing our RL agents, we explored a wide range of values for the variables, including various choices for the probability of exploration (ϵ), the weight of n-step backups (λ), and the features ($\vec{d_s}$).

To collect features for function approximation, we gathered the obvious candidates (e.g. the Beta hyperparameters, the remaining budget, the means and standard deviations of the coins), along with some more subtle attributes (e.g. confidence intervals, budget based confidence intervals, modified lookaheads, variation among the coins, security of the best looking coin). We found these features to be relevant because they affected the optimal coin decision when we studied the optimal policy for small versions of the coins problem. Although we tested numerous combinations of features, we focus on five feature sets that are representative of the general trends observed. For each one of the five sets, Table 2.2 gives the names of the different*feature groups* that are included in the set. (The interested reader should refer to Appendix B.1 to see exactly which*features* are included in each *feature group*.) For our experiments of the next section, we trained five different TD(λ) tile coding agents, where each agent used one of the five feature sets for its function approximation.

Table 2.2: Feature sets used for approximating the value function

Set Number	Feature Groups Included In Set
1	Budget, Beta hyperparameters
2	Budget, Means and Standard Deviations
3	Budget, Confidence Interval Stats
4	Budget, Mean Stats, Confidence Interval Stats
5	Budget, Lookahead Stats, Confidence Interval Stats

Table 2.3: Expected regret of various policies

Policy	(n=5, b=15)	(n=8, b=16)	(n=10, b=20)
BR	0.05669	0.07544	**0.07210**
SCL	**0.05413**	**0.07342**	0.07211
RL(set1)	0.05747	0.07830	0.07473
RL(set2)	0.05791	0.07896	0.07390
RL(set3)	0.05555	0.07528	0.07385
RL(set4)	0.05545	0.07464	0.07248
RL(set5)	0.05537	0.07507	0.07280

2.7 Empirical Results

To test the effectiveness of our RL agents on active model selection, we conducted experiments on three problems of increasing difficulty, where each initial coin prior was a uniform Beta$(1, 1)$. Our five RL agents were given 1.8 million training episodes for the two smaller problems, and 2.8 million for the larger problem. The expected regret (Equation 2.2) was calculated for BR, SCL, and the policies learned using our RL agents. For this first set of experiments, we used TD(0) agents. The results are shown in Table 2.3.

The results indicate that for all problems considered, either BR or SCL produced the smallest expected regret. In fact, no RL policy is able to beat either of the heuristic policies in the case of ten coins and a budget of twenty, and no RL policy is able to beat SCL on *any* of the problems. We have observed that on even larger problems (e.g. ten coins and a budget of thirty), BR beats SCL and RL policies easily. The results of the experiments reveal that despite the extensive number of states observed during training, the RL policies are not generalizing well enough between states to beat the simpler policies.

In our next set of experiments, we tested the effect of varying λ for the TD(λ) learner. Figure 2.4 shows the results of varying λ when using the fifth set of features for function approximation on the $n = 8$, $b = 16$ problem. For all values of λ considered, the policies learned by RL do only slightly better than BR and are inferior to SCL. The difference between the various TD(λ) learners is not dramatic, but the expected regret is lowest with an intermediate value of $\lambda = 0.5$.

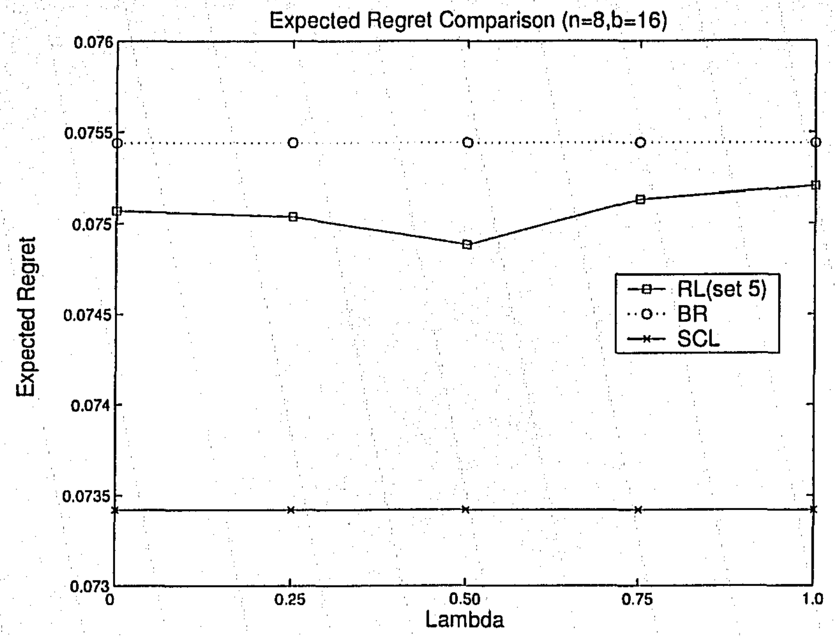


Figure 2.4: Various values of lambda — SCL still superior to RL

A possible explanation for the lower performance of RL is that not enough training episodes are being experienced. Additional training should permit an RL agent to increase its exploration of the state space, and yield a better policy. To test the effect of increased training, we conducted experiments on the $n = 10, b = 20$ problem in which we varied the number of training episodes from two and a half million up to an even more generous four and a half million. Learning took place with a TD(0.5) learner, using one of the strongest RL feature sets we tested, set number five. The downward sloping trend of Figure 2.5 suggests that increased training does improve the resulting policy; however, even after four million episodes, the expected regret of the RL policy is still larger than BR's or SCL's.

For further comparison, we consider the training time and memory required by BR, SCL, and the RL policy after four and a half million training episodes. The memory considered is only the policy specific storage (i.e., above and beyond the basic elements such as the Beta hyperparameters and the budget, that are generally required by all policies). Examining Table 2.4, we see that even using almost 800 MB of main memory, RL does not gain a significant advantage over the virtually memoryless BR and SCL routines.

As these experiments show, the performance, speed, and low memory requirements make the simper BR and SCL policies preferable to the use of Reinforcement Learning. Although it should be possible for an RL agent to do better than these heuristic policies, the experimental results indicate that (at least) more cleverly designed features or a better type of function approximator will be required to achieve this.

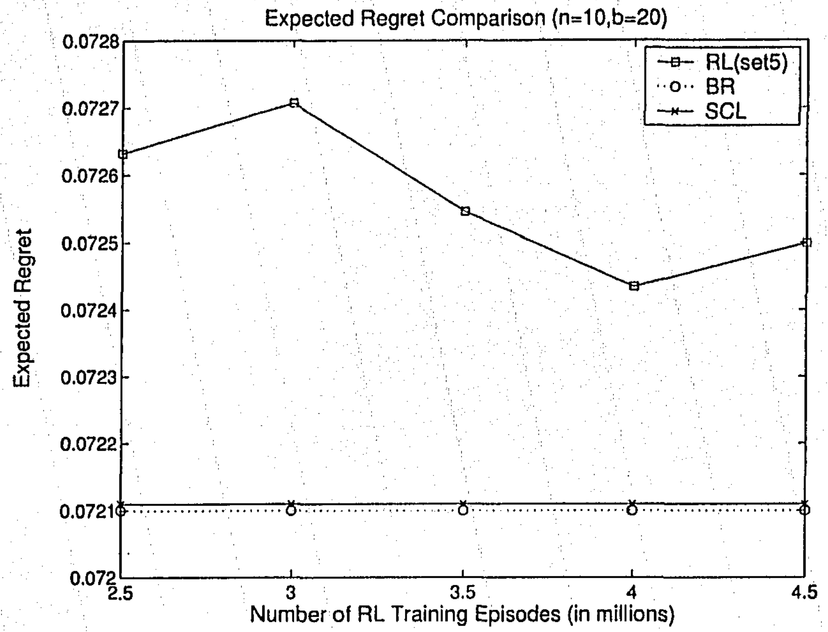


Figure 2.5: Various amounts of training — simple policies still superior to RL

Table 2.4: Resources used by each policy on n=10, b=20

Policy	Training time (mins)	Memory Used (MB)
BR	0	0
SCL	0	0
RL(set 5)	630	760

Perhaps the clearest argument *against* using RL for active model selection (and hence general budgeted learning) is the opportunity cost of conducting the necessary training. That is, although experience is easy to generate, the time and memory used to train RL agents could be equally well spent running a bottom-up dynamic program (as in Section 2.4.1) that solves for the optimal value of states. The dynamic program could compute the optimal policy from some select set of states in the same amount of time it takes a Reinforcement Learning agent to complete training. In effect, the optimal actions from this select set of states could be easily combined with the BR or SCL policies to lower their regret even further, and make it yet more difficult for RL methods to compete with these heuristic policies on active model selection. Overall, in the absence of better features for function approximation, these results suggest that the more tractable heuristic policies should be used instead of RL when considering the higher-dimensional and even more complicated problem of budgeted learning.

2.8 Unsuccessful Approaches

We have experimented with several other algorithms for active model selection which have not performed particularly well. We collect these negative results in this section, and briefly describe the algorithms and their shortcomings so that future research on active model selection can avoid these approaches and focus on more promising techniques.

2.8.1 Supervised Learning of a Classifier

It is possible to apply standard supervised learning techniques to learn a classifier for active model selection. Here the classifier implements a policy for active model selection by taking the Beta hyperparameters and the remaining budget as input, and returning the index of the best coin to flip as output. We used a dynamic program to generate the training data required for learning. Unfortunately, the dynamic program can only generate labelled data when the budget and number of coins is small, making it difficult to learn a classifier that can be applied to the more interesting (large) problems. In our experiments, we used training data to learn axis-parallel and oblique decision trees [10] and found that even on small problems, the learned classifiers had higher expected regret than simple policies such as BR.

2.8.2 Search

Blind search algorithms such as depth-first search can clearly find the optimal solution to the coins problem, albeit with time complexity on the order of $(2n)^b$. Nevertheless, we tested a depth-first search in the obvious way: truncating the lookahead depth to a reasonable level, and backing up heuristic estimates of $V^{\pi^*}(s)$ for all states s at the search horizon. The implementation was used in an on-line manner (similar to standard two player game tree search [25]) where a new search was conducted to the horizon level after each action was taken and the resulting next state observed. The search experiments confirmed that one does not require full lookahead to achieve reasonable performance. In fact, on the small and medium size problems tested, a lookahead of $b/2$ steps was fairly competitive with SCL. Of course, when the number of coins grows large, lookaheads become increasingly expensive and cannot be done to any effective depth, limiting the use of blind search for active model selection.

2.8.3 Optimal Two Coin (OTC)

In addition to search, we experimented with the optimal two coin algorithm. The OTC algorithm breaks up a large problem into several smaller, abstract problems that it can solve optimally. It then uses the solution to these abstract problems to choose an action for the original problem. Specifically, given n coins and a budget of b, OTC considers several abstract problems, each of which retain the budget b, but have only two coins from the original set of n. As there are $\binom{n}{2}$ possible pairs of coins, OTC computes an optimal policy for $\binom{n}{2}$ abstract problems. It then selects the abstract problem A_{best} whose optimal policy $\pi^*_{A_{best}}$ has the highest EMEHP, and it takes the first action of policy $\pi^*_{A_{best}}$. After observing the outcome of the action, the Beta distributions and the budget are updated, and a new set of $\binom{n}{2}$ abstract problems are solved to determine the next action to take.

To make the algorithm efficient, *all* possible two coin problems that can be encountered in the original n coin, budget b problem are solved optimally off-line (prior to running OTC) by a dynamic program. With only two coins involved, the dynamic program is quite fast and can typically compute the solutions to all abstract problems in a few seconds. Unfortunately, performance of OTC falls behind BR, particularly on problems with a large number of coins. When n is large, the optimal policy often involves interactions between many of the coins, and OTC is unable to consider interactions of more than 2 coins. Although using abstraction for active model selection may hold promise, our experiments with OTC show that a more clever type of abstraction will be required to be effective.

2.9 Summary

In this chapter, we explored the problem of active model selection. From a machine learning standpoint, active model selection is interesting because it is a simpler version of budgeted learning. The tight relationship between active model selection and budgeted learning has been described in previous research [17, 18], and also highlighted in this chapter. A particularly interesting property is that both problems have finite, episodic MDP formulations. As a result, Reinforcement Learning (a collection of techniques for developing intelligent behaviour in MDPs) appears to hold promise for solving budgeted learning. This chapter takes a first step toward testing this hypothesis, by extensively training several RL agents using different features for function approximation on the active model selection task. Our experiments demonstrate that simple heuristic policies are able to achieve lower expected regret with far less computation than the learned RL policies. Our results provide empirical evidence to the machine learning researcher that in the absence of more sophisticated function approximation (i.e., without better features or a better type of function approximator), applying RL techniques to the higher dimensional and more complex problem of budgeted learning will prove ineffective. Moreover, the experimental results reinforce the effectiveness of simple, heuristic policies for budgeted learning. We thus concentrate on heuristic approaches in the next chapter, when we consider budgeted learning a bounded active classifier.

Chapter 3

Budgeted Learning a Bounded Active Classifier

3.1 Introduction

3.1.1 A Motivating Example

Consider a doctor using a classifier to diagnose patient disease. The features of the classifier will typically be the results of medical tests such as X-rays, MRIs, or blood work on the patient. Due to the costs associated with running these tests, it is unrealistic to assume that the classifier will know the value of all features during classification. Instead, the doctor may be given a budget of \$100 to treat each patient, and the classifier can actively spend this \$100 to collect some features on which to base its classification. Since this classifier actively collects features and operates under a hard budget, we refer to it as a "bounded active classifier" (BAC) [9].

Learning this \$100 BAC will be an expensive proposition, because a complete training instance requires running all medical tests on a patient with a known disease. Here, the hospital may have only \$10 000 to allocate to *learn* the best \$100 classifier. That is, only \$10 000 are available to collect the features for labelled training instances. Faced with these dual budget constraints on the learner and the classifier, how should the machine learning researcher spend the \$10 000 collecting features of labelled training instances so as to learn the most accurate \$100 BAC?

3.1.2 Objective and Outline

The previous example demonstrates the real-world problem of budgeted learning a bounded active classifier. This chapter considers the problem in detail. More precisely, we study classification tasks in which feature values are initially unknown to the learner and classifier, and can be acquired at a cost. The learner is given a pool of labelled but otherwise unknown training examples, and it must decide how to spend its fixed learning budget b_L acquiring features of training instances so as to produce the most accurate active classifier that spends at most b_C per instance.

Before investigating our problem, we provide some background material on active classifiers in

Section 3.2. Following this review, we present the formal problem description for budgeted learning a bounded active classifier, as well as some complexity results. We also place our problem in the MDP framework, which allows us to describe the (intractable) optimal algorithm and to improve its running time (Section 3.4). The main contribution of the chapter is the description and empirical comparison of several tractable purchasing algorithms that the learner can employ. Sections 3.5 and 3.6 describe the details of these purchasing algorithms. Our experimental results (Section 3.7) demonstrate that when the learning budget is small, the obvious "round robin" algorithm (purchasing every feature of every instance until the b_L budget is exhausted) is problematic. As well, we show that our alternate learning strategies are able to outperform round robin on a variety of real-world datasets.

3.2 Background: Bounded Active Classifiers

3.2.1 Definition of an Active Classifier

An active classifier (AC) is a classifier that can actively purchase the value of unknown features before making its classification decision. Given some partially specified instance (e.g. $\langle x_1, ?, ?, x_4 \rangle$), an active classifier can either output a class label y, or it can choose to gather more information by requesting the value of an unspecified feature (e.g. X_2 or X_3). In general, the active classifier can recur indefinitely, continually purchasing unknown features for its current instance (as long as it can afford to pay for these features). Let us assume that we have a binary classification task in which there are r total features $\{X_i\}_{i=1..r}$ and two classes (Y= $+$ and Y= $-$), with the domain of feature X_i denoted by $dom(X_i)$, and with an unknown feature value denoted by "?". Then, formally, an active classifier is a function:

$$\text{AC} : \{dom(X_1) \cup \{?\} \times dom(X_2) \cup \{?\} \times \ldots \times dom(X_r) \cup \{?\}\} \rightarrow \{+, -, 1, \ldots r\} \quad (3.1)$$

where an integer output i indicates the request for the unknown feature X_i, and an output of $+$ or $-$ indicates a (final) classification decision. Contrast an AC with the traditional passive classifier (PC) that cannot request additional information. Since a PC can only output a class label based on the given feature values, it is poorly suited to tasks where features are initially unknown but can be acquired for a cost. To represent an AC, we can use a decision tree as in Figure 3.1. Notice that each interior node of the tree corresponds to a purchase of some feature, while a terminal (leaf) node corresponds to the AC's classification decision. As an example, if a test instance descends down the leftmost branch of the AC in Figure 3.1, then the AC must pay $Cost(X_2) + Cost(X_7)$ for the features it acquires before returning $Y = +$.

3.2.2 Bounding Active Classifiers

Many real-world tasks place a hard budget on the value of features that can be collected at classification time (e.g., a doctor who must diagnose patients using at most \$100 worth of tests). In these

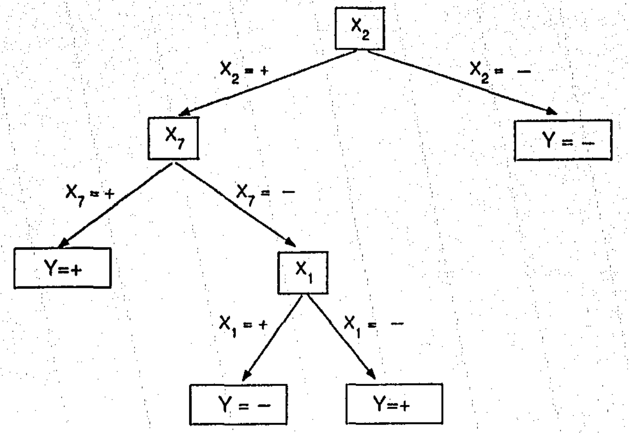


Figure 3.1: An example of an active classifier when the features and class label are binary

cases, a *bounded* active classifier (BAC) is required. A BAC with bound b_C is an active classifier that spends at most b_C for any test instance.

To score a BAC B, we consider its expected misclassification error over the distribution of labelled instances:

$$Q(B) \quad = \quad \sum_{\mathbf{x},y} P(\mathbf{x},y)\ L(B(\mathbf{x}),y) \tag{3.2}$$

where $L(i,j)$ denotes the misclassification error of classifying an instance as i when its true class is j. Let $All(b_C)$ denote the set of all bounded active classifiers that spend at most b_C per instance. We will typically be interested in the optimal bounded active classifier from $All(b_C)$, which is the one that minimizes the expected misclassification error:

$$\text{BAC}^* = \operatorname*{arg\,min}_{B \in All(b_C)} \ Q(B)\ . \tag{3.3}$$

In fact, when we present our formal problem description for budgeted learning a bounded active classifier in Section 3.3, the ultimate goal is to produce this BAC*. Previous research [9] has shown that it is possible to PAC-learn this BAC* by using a straightforward dynamic program that has *sufficiently accurate* estimates of the following distributions:

$$P(Y = y|\mathbf{X} = \mathbf{x}^*) \tag{3.4}$$

$$P(X_i = x_i|\mathbf{X} = \mathbf{x}^*) \tag{3.5}$$

where $\mathbf{x}^*$ is any partially specified feature vector with at most b_C dollars worth of feature values specified. As we describe formally in the next section, our problem involves learning BAC* when we have only a *limited learning budget* with which to estimate the two aforementioned sets of distributions.

3.3 Formal Description

The "budgeted bounded-active-classifier learner", BBACL, is given the (non-negative) cost $C(X_i) \in \Re^+$ of acquiring each individual feature X_i of any single specified instance[1] and the loss matrix $L = [\ell_{i,j}]$ whose (i, j) element specifies the penalty for returning the class y_i when the true class is y_j; by convention we assume $\ell_{i,i} = 0$ and $\ell_{i,j} > 0$ for $i \neq j$. BBACL also knows the total amount the learner can spend $b_L \in \Re^+$, and how much the resulting active classifier can spend per instance $b_C \in \Re^+$.

At any time, the BBACL can see the current $m \times (r+1)$ "tableau", whose rows each correspond to an instance $i \in \{1, \ldots, m\}$ and whose first r columns each correspond to a feature, and whose $r + 1$st column is the class label. Initially, only the class label is specified; the other $m \times r$ entries are all unknown. In general, we will let $x_i^{(j)}$ refer to the initially unknown value of the ith feature of the jth instance. At any point, BBACL can perform the $x_i^{(j)}$ "probe" to determine the value of $x_i^{(j)}$, at cost $C(X_i)$. This also reduces BBACL's remaining budget from b_L to $b_L - C(X_i)$. Once this budget reaches zero, BBACL stops collecting information and returns a bounded active classifier which corresponds to a decision tree of bounded depth [6]. Our goal is to produce BAC*, the bounded active classifier that has minimal expected misclassification error and spends at most b_C collecting features per instance (see Equation 3.3).

3.3.1 Simplifying Assumptions

Recall from Section 3.2.2 that in order to PAC-learn BAC* we require accurate estimates of distributions 3.4 and 3.5. In order to tractably estimate these distributions under our finite learning budget, we will make some simplifying assumptions. Firstly, the obvious frequentist approach of maintaining simple frequencies for probabilities is problematic, because many conditioning events will not occur given the sparsity of data. Instead, we will take a Bayesian stance by assuming that there is a prior distribution over labelled instances before seeing any data. In addition to this Bayesian approach, we will make the Naïve Bayes assumption, which means $P(x_i^{(j)})$ is independent of $x_k^{(j)}$ (for $k \neq i$) as we know the value of the class $Y = y_j$.[2]

Hence, if instance j is labelled with class $+$, and feature X_i has domain size $|X_i| = w$, we will model the distribution of the w multinomial parameters for $x_i^{(j)}$ as a Dirichlet distribution [11]: $\text{Dir}(\,\alpha_{1,+}^{(i)}, \ldots, \alpha_{w,+}^{(i)}\,)$, with Dirichlet parameters $\alpha_{j,+}^{(i)} > 0$. (Although technically it is the w multinomial parameters that are Dirichlet distributed, we will still write $x_i^{(j)} \sim \text{Dir}(\,\alpha_{1,+}^{(i)}, \ldots, \alpha_{w,+}^{(i)}\,)$ to simplify notation.) These Dirichlet parameters $\alpha_{j,+}^{(i)}$ are unrelated to the ones for negatively labelled instances $\alpha_{j,-}^{(i)}$ and also unrelated to the Dirichlet parameter values for other features X_h, for $h \neq i$.[3]

[1] We assume that these costs are independent of each other, both within and across instances. Moreover, if any test costs $C(X_i) = 0$, we can simply gather that information for each instance and then consider the resulting reduced problem where $C(X_i) > 0$ for all remaining X_is.

[2] Note that Naïve Bayes models often produce good classifiers even for datasets that violate this assumption.

[3] Thus, we maintain a single Dirichlet distribution for each $\langle$ feature, class-value $\rangle$ pair.

Initially, we will assume that each such distribution is uniform Dir($1, \ldots, 1$). If we later see a sample T with 29 $Y = +$ instances with $X_i = +$ and 14 $Y = +$ instances with $X_i = -$, the posterior distribution for $x_i^{(j)}$ for a new $Y = +$ instance would be Dir($1+29,\ 1+14$). The mean probability for $X_i = +$ here would be $P(X_i = +|T) = 30/(30+15) = 2/3$.

In general, if a variable X's prior distribution is $X \sim \text{Dir}(\alpha_1, \ldots, \alpha_w)$, then

$$P(X = i) \quad = \quad \frac{\alpha_i}{\sum_k \alpha_k} \tag{3.6}$$

If we then observe a sample T that includes a_i instances of $X = i$, then X's posterior distribution remains a Dirichlet, with new parameters

$$X|T \quad \sim \quad \text{Dir}(\alpha_1 + a_1, \ldots, \alpha_w + a_w) \ . \tag{3.7}$$

In the formal description given earlier in this section, a learning probe of the form $x_i^{(j)}$ specifies the feature to probe (X_i) and the specific instance in the tableau (instance j) on which to perform the probe. However, because of our Naïve Bayes assumption, we can treat all instances with the same class label identically. Thus, rather than querying specific instances, we only consider learning probes of the form (i, y) that request the ith feature of a randomly chosen instance in the tableau whose class label is y.[4] (By convention, this process selects the value of an (i, y) feature value that has not been seen before.) Finally, for our work we will assume a constant misclassification cost $\ell_{ij} = 1$ for $i \neq j$ and $\ell_{ii} = 0$.

3.3.2 Complexity Results

Unfortunately, the problem of budgeted learning a bounded active classifier is NP-hard in general. In fact, the reduction follows from the active model selection task we studied in Chapter 2. More precisely, Madani et al. [19] proves the following (active model selection) task is NP-hard: given a set of coins with known prior distributions and a fixed total number of flips, decide when to flip which coin to decide which coin has the highest head probability. Our framework inherits this negative NP-hardness result. (Identify each coin C_i with a binary feature X_i, where the head probability of coin C_i corresponds to the probability the class is true given X_i is true, $P(Y = +|X_i = +)$; we also let $P(Y = +|X_i = -) = 0$ for all features.) In addition, [9] shows that computing the best active classifier is NP-hard in general, even if we know the entire distribution. Our framework inherits that negative result as well.

3.4 The MDP Formulation

Budgeted learning a bounded active classifier is a sequential decision making problem: the learner sequentially specifies which feature-class pair to purchase from the tableau, and receives a one-time

[4] In other words, to make a purchase the learner only has to select a feature and a class – not a specific instance in the tableau.

reward (the expected misclassification error of the learned BAC) once the learning budget has been exhausted and the final learned BAC is applied. The task can be completely described as a finite, episodic Markov Decision Process with a (finite) set of states $s \in S$, a (finite) set of actions $a \in A$, a reward function $R(s)$, and a transition function $T(s, a, s')$. In fact, the mapping is very similar to the one described in Chapter 2. Each state of the problem $s \in S$ is identified by the remaining learning budget (denoted by b'_L), and by the posterior Dirichlet distributions over all the feature-class pairs. This representation of a state encapsulates all the information that has been learned so far about the various feature-class pairs. We identify the initial state s_0 as the one with full remaining learning budget ($b'_L = b_L$), and with the Dirichlets set to the (given) prior distributions. On the other hand, the terminal states are those in which the remaining learning budget is insufficient to make any more purchases ($b'_L < C(X_i) \quad \forall i$), and has thus resulted in a final set of posterior Dirichlet distributions. As in Chapter 2, we use s_i to denote the state encountered on the ith time step.

The space of possible actions corresponds to every distinct feature-class pair that the learner can purchase from the tableau. For the reward function, we use $R(s)$ to denote the immediate reward received in state s.[5] $R(s)$ is zero whenever s is a non-terminal state (i.e. no intermediate reward). On the other hand, if s is a terminal state, the reward received is the expected misclassification error of the best BAC that can be learned from the posterior Dirichlet distributions in s:

$$R(s) = \min_{B \in All(b_C)} \sum_{\mathrm{x},y} P(\mathrm{x}, y|s)\ L(B(\mathrm{x}), y) \quad \text{iff } s \text{ is a terminal state} \tag{3.8}$$

Since all reward is delayed until the final time step, our problem has a discount factor of $\gamma = 1$. In terms of notation, we will use r_i to denote the immediate reward accrued on the ith time step.

Finally, the transition function $T(s, a, s')$ specifies the probability of a particular feature purchase taking on a particular value. These transition probabilities are given by the *current* Dirichlet distributions over the feature-class pairs. For instance, suppose we purchase feature X_i on an instance where $Y = +$, and our current Dirichlet distribution for that feature-class pair is $X_i|+ \sim \text{Dir}(1, 4, 3)$. Then the probability of transitioning to the next state which has $X_i|+ \sim \text{Dir}(1, 5, 3)$ is the probability of X_i taking on its second value (x_{i_2}) given a positive class label: $P(X_i = x_{i_2}|Y = +) = 4/(1 + 4 + 3) = 1/2$. With S, A, R, and T specified, we have a complete MDP formulation for budgeted learning a bounded active classifier. The MDP formulation allows us to use the notation of policies and value functions. In our case, a policy $\pi : S \longrightarrow A$ specifies which action the learner should take (i.e., which feature-class pair the learner should purchase) given the current state. As well, the value function $V^\pi(s_t)$ specifies the expected reward accrued from state s_t when following policy π. In this chapter, we define the value function for a policy π as:

$$V^\pi(s_t) = E\left(\sum_{i=0} (\gamma^i r_{i+t} | \pi, s_t)\right) \quad . \tag{3.9}$$

[5] This reward notation is slightly different than the one used in Chapter 2; we make this slight notation change in this chapter strictly to simplify the proof of some upcoming results.

Notice that this definition is trivially different from the definition in Chapter 2 (Equation 2.5), because here we include the immediate reward received in state s_t as part of $V^{\pi}(s_t)$. This slightly different value function definition simplifies the proof of some upcoming results *without* changing our problem in any way.

3.4.1 The Optimal Learning Policy

As our problem is a finite Markov Decision Process, there exists a deterministic optimal policy for spending the learning budget [28]. The optimal learning policy is the one that minimizes the expected$_1$ expected$_2$ misclassification error of the final bounded active classifier. The first expectation$_1$ is over the set of possible Dirichlet distributions produced by the learner's purchases, and the second expectation$_2$ is over the possible labelled instances $(\mathbf{x}, y)$ that can occur *given* the resulting Dirichlets. Mathematically, the optimal learning policy π^* is defined as:

$$\pi^* = \arg\min_{\pi} \sum_{i \in Outcomes(\pi)} P(i) \sum_{\mathbf{x},y} P(\mathbf{x}, y|i)\ L(BAC_i^*(\mathbf{x}), y) \tag{3.10}$$

where each "outcome" corresponds to a terminal state in which our learning budget has been fully exhausted and has resulted in posterior Dirichlet distributions over the feature-class pairs. (Notice that the optimal BAC for each outcome state is contingent upon the probabilities that have been learned, and thus we write BAC_i^* to denote the optimal BAC with respect to the probabilities learned in outcome i.)

This optimal learning policy π^* can be computed via a bottom-up dynamic program. To see this, note that we can compute the optimal value function ($V^{\pi^*}(s)$) for all possible outcome states s where the learning budget has been exhausted, and then use these to compute the value for all possible "predecessor" states where there is only \$1 left in the learning budget, and then continue this backward sweep toward the initial state s_0. Unfortunately, the number of outcome states (and hence the computational complexity of the naïve dynamic program) has a prohibitive lower bound:

Proposition 4 *Let $|X_i|$ denote the domain size of feature X_i, $|Y|$ denote the number of classes, $t = |Y| \sum_i |X_i| - 1$, and each feature has unit cost. Then the bottom-up dynamic program must compute the value of $\Omega\left(\left(\frac{b_L+t}{b_L}\right)^{b_L}\left(\frac{b_L+t}{t}\right)^{t}\frac{1}{\sqrt{t}}\right)$ outcome states.*

We improved this naïve dynamic program by reducing the number of states whose value must be solved for. Below we show an interesting way to achieve this reduction by exploiting the equivalence of two "permuted" states under the conditional independence assumption.

Definition 1 *A* proper permutation *for a feature X_i with w domain values is a bijective function $f : [1, w] \longrightarrow [1, w]$ that applies the same reordering of the w parameters for every Dirichlet distribution on X_i.*

Example 1 *Let*

$$(X_i|Y = 0) \sim Dir(4, 2, 7), \quad (X_i|Y = 1) \sim Dir(3, 8, 5)$$

Table 3.1: Reduction in computation time using Proposition 5

b_L	b_C	Features	Domain Size	Naïve	Improved
2	4	6	4	161 sec	65 sec
3	2	4	3	888 sec	432 sec
4	3	4	3	8280 sec	3360 sec

Then a proper permutation *for feature X_i is:*

$$(X_i|Y=0) \sim Dir(7,2,4), \quad (X_i|Y=1) \sim Dir(5,8,3).$$

Proposition 5 *Assume the Naïve Bayes assumption holds, and consider any two states s_a and s_b that have equal values of b_L and are such that the Dirichlets of state s_a can be made equal to the Dirichlets of state s_b by specifying a set of r proper permutations, one for each feature X_i. Under these conditions, $V^{\pi^*}(s_a) = V^{\pi^*}(s_b)$, and $\pi^*(s_a) = \pi^*(s_b)$.*

This proposition allows us to improve the naïve dynamic program by reusing the computed value of a state s_a for properly permuted versions of s_a. The real-time improvement using Proposition 5 is shown in Table 3.1. In the last case ($b_L = 4$, $b_C = 3$), the naïve dynamic program ran out of memory after more than two hours, while our improved version finished properly in under an hour. Unfortunately such improvements are not sufficient to remove the exponential complexity of the dynamic program (recall that this task is NP-complete); therefore, we consider more tractable, suboptimal approaches in the next section.

3.5 Heuristic Learning Policies

This section summarizes a number of heuristic "budgeted bounded-active-classifier learners". We focus on only the data collection part of the algorithms; after collecting $\$b_L$ worth of feature values, each of the algorithms then passes its learned (posterior) Dirichlet distributions to a dynamic program that produces the BAC* in Equation 3.3. Our decision to focus on heuristic purchasing algorithms is partially motivated by the results of Chapter 2, in which we observed that simple heuristics were able to outperform more complex methods such as RL, and blind search on the related problem of active model selection. We note that many of the algorithms that follow are extensions or variants of heuristics used in other budgeted learning scenarios [17, 18].

3.5.1 Round Robin (RR)

This obvious algorithm simply purchases *complete* instances until its budget b_L is exhausted. It draws examples randomly, and so expects to have collected data about members of each class y in proportion to $P(Y = y)$. If there are r unit-cost features, we expect to know everything about roughly b_L/r instances. Notice RR implicitly assumes all features are equally valuable in learning the target concept.

3.5.2 Biased Robin (BR)

A more selective approach than round robin is to purchase a single feature and test whether or not its observed value has increased some measure of quality. The biased robin algorithm is more selective than RR, continually purchasing feature X_i as long as it improves quality, and otherwise moving to feature X_{i+1} (and of course looping back to X_1 after X_r). There are several choices for how to measure quality or loss; see Section 3.6. Of course, BR must also specify a class y from which to purchase its desired feature, and it does this by drawing from the class distribution $P(Y = y)$ on each purchase. As further motivation for this algorithm, [17] found it to be one of the best approaches for budgeted learning of a passive Naïve Bayes classifier, albeit with a different loss function. This method also corresponds to the "Play the Winner" approach discussed in [23].

3.5.3 Single Feature Lookahead (SFL)

One would always like to avoid wasting purchases on poor features, especially when faced with a limited learning budget. This motivates a prediction-based approach, which uses a loss function to estimate the expected loss incurred after making a sequence of purchases of a single, specified feature.

SFL uses this prediction based approach, and controls the level of myopia or "greediness" involved by providing an additional parameter, d = the lookahead depth. With a lookahead depth of d, SFL calculates the expected loss of spending its next \$$d$ sequentially purchasing feature i of instances of class j. That is, if s denotes our current set of Dirichlets and s' denotes any of the Dirichlet sets obtained after spending $\min(\$d, \$b'_L)$ purchasing feature X_i of $Y = j$ instances, then the expected loss for (i, j) is:

$$SFL(i,j) \quad = \quad \sum_{s'} P(s'|s)\, Loss(s')\ . \tag{3.11}$$

SFL determines the feature-class pair (i, j) with lowest expected loss, then purchases the value of this best (i, j) feature for *one* instance, and updates the Dirichlets based on the observed outcome of that purchase (and reduces the available remaining budget). It then recurs, using Equation 3.11 to compute the score for all feature-class pairs in this new situation — with its updated Dirichlets and a smaller budget. This process repeats until the learning budget is exhausted. The lookahead depth d can be set based on the computational resources available. If only the next one purchase is considered, then this reduces to the 1-step greedy algorithm. We note that SFL was originally used in [17, 18] (but with a different loss function).

3.5.4 Randomized SFL (RSFL)

Our experiments show that the SFL algorithm often spends the majority of its probes purchasing a single discriminative feature-class pair and neglects to explore other potentially good features. This property can be problematic, particularly when a dataset contains several discriminative features that can jointly yield a more accurate BAC than any single feature by itself. The randomized single feature lookahead algorithm (RSFL) alleviates this problem by increasing exploration among the best looking feature-class pairs. The RSFL algorithm is very similar to SFL, as it too calculates the expected loss in Equation 3.11 for each feature-class pair. However, rather than deterministically purchasing the pair with the best SFL score, RSFL considers the best K feature-class pairs and for each feature-class pair (i, j) in this set, it chooses to purchase feature i of class j with probability:

$$\frac{\exp \frac{-SFL(i,j)}{\tau}}{\sum_{i,j} \exp \frac{-SFL(i,j)}{\tau}} \tag{3.12}$$

Here, τ is a temperature controlling exploration versus exploitation. Although we set τ to one throughout this chapter, we include it in Equation 3.12 to show the relationship to the Gibbs distribution [28]. After experimenting with various values for the number of feature-class pairs, K, we found that K = (number of classes) × b_c seemed to perform well, particularly when the learning budget was not much greater than the number of features.

3.6 Loss Functions

As mentioned earlier, several of our algorithms rely on a loss function

$$Loss : \{\text{Dirichlet distributions over feature-class pairs}\} \rightarrow \Re \tag{3.13}$$

that attempts to measure the quality of a given probability distribution. After experimenting with several different choices of loss functions, we found Conditional Entropy Loss and Depth 1 BAC Loss to be effective.[6]

SFL, RSFL, and the greedy algorithm all use

$$\min_i \sum_x P(X_i = x) \min_y (1 - P(Y = y|X_i = x)) \tag{3.14}$$

which calculates the expected misclassification error of the best Depth 1 BAC. Since biased robin needs to detect small changes in a distribution, it tends to perform better with the more sensitive conditional entropy calculation, which measures the uncertainty of the class label Y given the value of a feature X_i:

$$-\sum_x P(X_i = x) \sum_y P(Y = y|X_i = x) \log_2 P(Y = y|X_i = x) \ . \tag{3.15}$$

The biased robin algorithm uses Equation 3.15 before and after the purchase of feature X_i to determine whether the purchase improved the ability of X_i to predict the class Y.

[6] The obvious loss function is just to use Equation 3.3 to compute the expected error of the optimal BAC. However, since loss functions can be called several times to decide on a single purchase, the computational expense of computing Equation 3.3 is prohibitive.

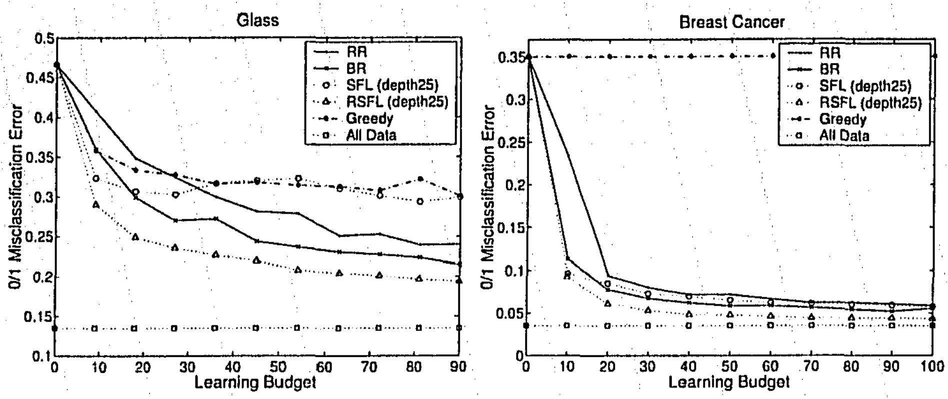


Figure 3.2: Identical costs and some irrelevant features — RSFL and BR outperform RR

3.7 Experimental Results

To compare the algorithms, we tested their performance on several datasets from the UCI Machine Learning Repository [12]. We used supervised entropy discretization [8] to discretize datasets with continuous values. Each dataset was then randomly partitioned into five folds. The algorithms were run five times, and on each run a single fold was set aside for testing, while the remaining four were available for purchasing. For each algorithm, we used the average value of these five runs as the algorithm's misclassification error on the whole dataset. We repeated this process 50 times to reduce the variance and get a measure of the average misclassification error. Thus, each point in the graphs that follow represents 50 repetitions of five-fold cross validation.

In the first set of experiments, all features have unit cost and the datasets contain some irrelevant features. We set the classifier's budget to $b_c = 3$, as this is large enough to allow several features to be used, but small enough to keep computations tractable. All Dirichlets parameters are uniformly initialized to 1. For reference, each graph also includes a gold standard "All Data" algorithm, which is allowed to see the *entire* dataset, and thus represents the best that one can do using the Naïve Bayes assumption on the data.

Figure 3.2 shows the performance of the algorithms on the Glass Identification dataset: a binary class problem with nine features whose domain sizes vary between one and three. The four features that have a domain size of one represent irrelevant information that any learning algorithm (especially one under a constraining budget) should avoid. Both RSFL and BR learn better than the obvious RR algorithm for all learning budgets considered. In fact, we found the optimal $b_C = 3$ BAC produced by the "All Data" algorithm involves four different features, and these four features are precisely the ones that RSFL and BR purchase heavily during learning. This is in contrast to the RR purchasing behaviour that spends equally on all features, despite their unequal predictive power. Finally, SFL and greedy spend their entire budget on only one or two features during learning, which accounts for their low accuracy BACs.

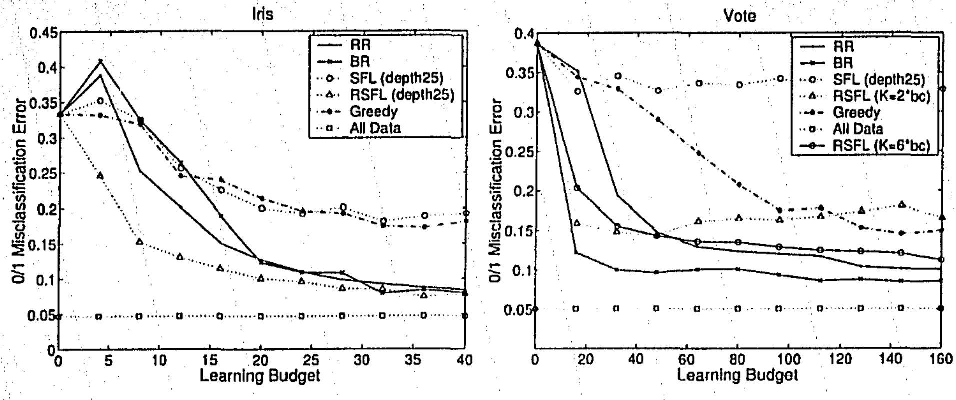


Figure 3.3: Identical costs, no irrelevant features — RR still suboptimal

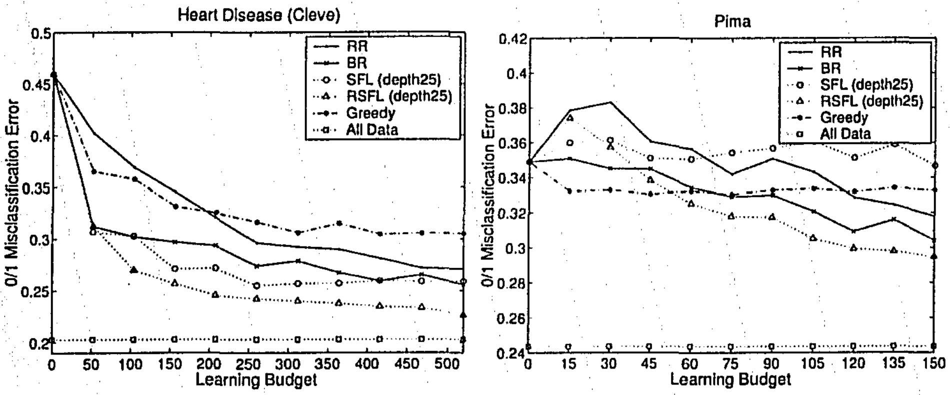


Figure 3.4: Different feature costs — RSFL and BR dominate RR

The Breast Cancer dataset contains ten features, only one of which is irrelevant to the concept. This dataset is particularly interesting because nearly all its features are good predictors, but three features have markedly lower conditional entropy than the rest. To produce the lowest error BAC, the learning algorithms must discover the superiority of these three features. We find RSFL does exactly this, spending 20%, 21%, and 32% of its budget respectively on the three strong features. In comparison, RR spends 10% of its budget on every feature which makes it much more difficult for it to separate the top features from the rest. BR also performs better than RR for all learning budgets considered.

The next set of experiments, shown in Figure 3.3, considers datasets without any irrelevant features. The Iris dataset has only four features and is a three class problem. Given that all four features are relevant, and that $b_C = 3$ in this experiment, the optimal BAC requests every feature at some point in its tree. With only four features to consider, RSFL is able to test them all effectively and produce better BACs than RR for all budgets considered. BR is also competitive with RR, except at some of the very low budgets where BR's exploration model prevents it from ever investigating some of the features.

Figure 3.3 (right) shows another binary class problem, the Vote dataset, that contains 16 features. Many of these features have similar (high) predictive power, and one feature in particular is nearly perfectly correlated with the class label. Once again we see that both RSFL and BR beat RR when the learning budget is small. RSFL asymptotes after about 50 purchases — it spends its budget finding a few strong features quickly and outputs a fairly low error BAC. As expected, at larger budgets RR collects enough information on every feature to find better candidates for its BAC than RSFL can. In particular, RR identifies the superiority of the "near perfect" feature more consistently than RSFL does at larger learning budgets. The graph shows that one can improve the performance of RSFL by increasing the number of top feature-class pairs, K, that RSFL considers on this dataset (thereby reducing the chance of RSFL skipping over the near perfect feature). We also observe that BR's exploration model is particularly well suited to this task because it is able to collect information on every feature at larger budgets, which is important on a dataset such as Vote with a large number of predictive features.

Our final set of experiments involved datasets where the features differed in cost. Both the Heart Disease dataset and the Pima Indians dataset have known cost data [12], which we scaled (so that costs were between \$1.00 and \$10.00) and then used in our tests. The scaled Heart Disease costs range from \$1 to \$7, and our tests are run with $b_c = \$7$. This dataset represents the worst case for RR, because the irrelevant features happen to be the most expensive ones. In fact, RSFL achieves the same error rate after \$100 that RR takes \$500 to reach. In the Pima dataset, feature costs are between \$1 and \$5, and we set $b_c = \$5$. The two irrelevant features have cost \$1, and the single best feature is \$4. Once again, BR and RSFL dominate RR for all budgets considered.

3.8 Summary

Many standard learning algorithms implicitly assume the features are always available for free, to both the learner at "training time" and later the classifier, at "performance time". This chapter extends those systems by explicitly considering these costs (at both training and performance time), when the learner and classifier have hard budgets that limit the total value of features that can be collected. In this chapter, we introduce the formal framework for budgeted learning a bounded active classifier, and present some complexity results for the problem. We also propose a more efficient way to implement the optimal algorithm, which we prove works effectively. Moreover, this chapter motivates and defines a variety of tractable learning strategies and shows they work effectively on various types of data — both with identical and with different feature costs. In particular, we demonstrated that our proposed strategies can often do much better than the obvious algorithm – "round robin" – especially when training data is limited.

Chapter 4

Related Literature

4.1 Introduction

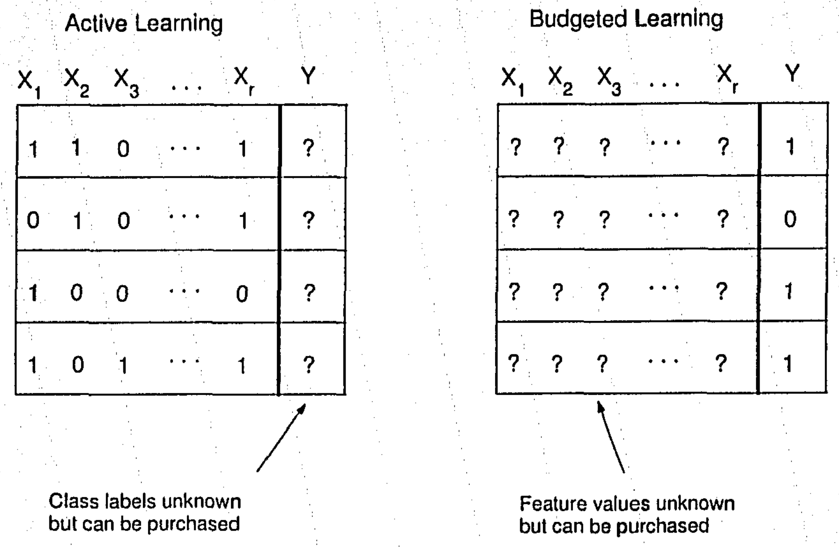
This chapter reviews some of the relevant literature from the fields of machine learning and sequential decision making. We divide the review into two parts. The first section highlights the work related to our main problem studied in Chapter 3, while the second section focuses on work related to the active model selection task investigated in Chapter 2.

4.2 Budgeted Learning a Bounded Active Classifier

There are a number of different senses of "costs" in the context of learning [31]. Our research considers two of these: the costs paid by the learner to acquire the relevant information at training time to produce an effective classifier, and also the costs paid by the classifier, at performance time, to acquire relevant information about the current instance. We impose hard constraints on the expenses paid by the learner, and on the total cost of tests that can be performed per instance by the classifier.

Many existing (sub)fields, such as active learning [5] and experimental design [3] (as well as earlier results such as [17]) focus on only the first of these costs – e.g., bounding how much the learner can spend to produce an accurate *passive* classifier. In addition, many of these systems request the *class label* for an otherwise *completely specified instance*. Thus they require only a single quantity per instance. Our problem is the complement of this: class labels are known but feature information must be purchased (see Figure 4.1). Unlike most of the other models, this means our work may need to consider the correlations amongst the many unknown properties of an instance.

There are numerous other machine learning results that focus on reducing the sample complexity for learning. Some of these include decision theoretic subsampling [21], on-line stopping rules [26], progressive sampling [22], and active feature value acquisition [20]. We note that these techniques differ from our approach because we place a firm prior budget on the learner's ability to acquire information, while these approaches typically allow the learner to purchase until some external



Active Learning

X_1	X_2	X_3	...	X_r	Y
1	1	0	···	1	?
0	1	0	···	1	?
1	0	0	···	0	?
1	0	1	···	1	?

Class labels unknown
but can be purchased

Budgeted Learning

X_1	X_2	X_3	...	X_r	Y
?	?	?	···	?	1
?	?	?	···	?	0
?	?	?	···	?	1
?	?	?	···	?	1

Feature values unknown
but can be purchased

Figure 4.1: Active learning versus budgeted learning

stopping criteria (for instance, accuracy) is satisfied.

Weiss and Provost [33] recently explored a problem related to one that we encounter in our overall framework: how to represent the class distribution when only a firm budget of n training examples can be used. For example, if our budget allows for ten training examples, should we select five from class one and five from class two, or draw our examples according to the true (underlying) class distribution? The results in [33] indicate that drawing from the true class distribution is the best choice for maximizing classifier accuracy when no additional experimentation can be done. On the other hand, when computational resources are available, Weiss and Provost suggest using a progressive sampling algorithm to choose the best class distribution. As discussed in Section 3.5, some of our algorithms (RR and BR) follow the results in [33] by drawing from the true class distribution when selecting which class to probe. We do not, however, utilize progressive sampling due to the computational expense, and the relatively small improvement reported in [33] over using the true class distribution.

Instead of considering the costs paid at learning time, some research has concentrated strictly on minimizing the costs paid by the classifier at performance time. In this vein, both [30] and [9] attempt to produce a decision tree that minimizes expected total cost. However, neither work assumes an a priori resource bound on the learner, thereby allowing for unconstrained amounts of training data with which to build these classifiers. Again, our work makes the more realistic assumption that if data costs money at performance time, it very likely costs money at learning time as well.

4.3 Active Model Selection

Active model selection was originally introduced in [18], although several similar problems have been previously studied. The well-known multi-armed bandit problem [23] is concerned with finding the best object within a set, but rewards are typically accrued throughout, without distinguishing training from testing phases. By contrast, active model selection gives no reward until the final coin is selected, and thus more accurately represents the pure training phase of budgeted learning. Strategies from the adversarial bandit formulation [2] could also be adopted for our problem, but the adversarial assumption is unnecessarily strong for our case, and thus less defensive algorithms can usually perform better on active model selection. A more recent bandit-variant, the max k-arm bandit [4], shares our notion of maximizing a *single* reward over a fixed number of sequential decisions. However, [4] allows the single reward to occur on any time step, as opposed to strictly at the terminal states.

Duff [7] studied the Bayesian MDP formulation in active model selection as a Bayes Adaptive Markov Decision Process (BAMDP). That study also considers various RL methods to approximate an optimal policy for BAMDPs, and chooses some of the same types of features for function approximation that we consider in Chapter 2. Moreover, the experimental results concur with our findings, as [7] also reports a gap between the reward of the learned RL policies and the optimal policy. Besides RL, another potential strategy for active model selection is on-line sparse lookahead [32, 16]. Unfortunately, given the size of the state space, we have found that any tractable (truncated) lookahead (as in [16]) usually yields a higher regret than the simple BR and SCL policies. It would be interesting to experiment with the recent ideas from [32] to see if a selectively grown lookahead tree could compete with the current heuristic policies.

Chapter 5

Conclusions

5.1 Contributions

This thesis examines classification learning when features have an acquisition cost, and the learner and classifier have only finite budgets to spend acquiring features of training and testing instances, respectively.

Chapter 3 explores this practical problem in detail, and provides the formal problem description. Other contributions from the chapter include a description of the optimal spending policy for the learner, as well as a method to effectively reduce the running time of the optimal algorithm. We also extend prior complexity results to our problem to establish it as NP-hard. Our main contributions are to propose several heuristic spending policies for the learner, and to test them empirically. The primary result of this dissertation is two fold. First, our experiments show that the obvious round robin purchasing policy that spends equally on all features is suboptimal — particularly when the learning budget is small relative to the number of features. Second, we observe empirically that our alternative purchasing algorithms (i.e. biased robin, randomized single feature lookahead) are able to outperform round robin on many datasets, both with identical and with different feature costs.

We also make a contribution to general budgeted learning by addressing an open question in the budgeted learning literature: can Reinforcement Learning techniques be used to learn an effective spending policy for the learner? Chapter 2 takes a first step toward answering this question by working with a simplified budgeted learning problem: active model selection. We extensively train multiple RL agents on active model selection, with each agent using a different combination of features for function approximation. Our experiments demonstrate that simple heuristic policies achieve lower expected regret on active model selection than the policies learned using the standard RL techniques and features we selected. These results suggest that (at least) better features for function approximation will be required if RL techniques are to be successfully applied to the higher dimensional and more complex problem of budgeted learning a bounded active classifier.

5.2 Research Directions

This dissertation has raised several interesting questions for future study. Beginning with the Reinforcement Learning investigation in Chapter 2, the most obvious open question is: what feature space should be used to represent the value function? A limitation of some of the features we used is that they do not incorporate the synergies that exist between coins (for instance, how the value of coin C_1 increases if coin C_2 and C_3 are also present in the current set of coins). A feature set that is able to approximately encode these dependencies may yield more promising results. Another area of future work is to use RL techniques to effectively learn the BR and SCL heuristic policies.[1] With some of the features we considered (in Appendix B), it is *theoretically* possible for an RL agent to represent the BR and SCL heuristics; however, we need more than just representation power in order to tractably learn BR or SCL. Specifically, the feature space must *generalize well* so that a TD(λ) agent only has to visit a reasonable number of states in order to learn a complete value function for the heuristic algorithms. Finding a feature set that can express the heuristics while permitting fast generalization would make RL competitive with BR and SCL (and thus more applicable to the full budgeted learning problem). Finally, given the low regret behaviour of BR and SCL, it would be interesting to prove or disprove the approximability characteristics of these heuristic algorithms under the common case of identical starting priors (this has remained an open problem since first posed by [19]).

We turn next to the full problem of budgeted learning a bounded active classifier. Although some of our proposed algorithms perform well on this problem, they might still be improved using some simple techniques. In the case of RSFL, for example, it may be better to *dynamically* choose the number K of top feature-class pairs to randomly select from (rather than fixing this threshold a priori). Alternatively, we might consider randomly selecting from all feature-class pairs, but with a decreasing temperature parameter τ, as in [4]. Another research direction is to experiment with algorithms that go beyond the "Naïve Bayes" assumption, and thus allow the learner to perform more powerful probes (e.g. requesting feature X_i on an instance where $X_j = +$ and $Y = -$). Related to this, we could consider maintaining additional probability estimates such as $P(X_i|Y, X_j)$ to incorporate dependencies among the features.

In our work, we have implicitly assumed that it is always worthwhile for the classifier to spend more on features (up to the budget b_C) if we can reduce our misclassification error. However, in many practical tasks, the cost of a misclassification error may be less than the cost of acquiring a feature. In these cases, it makes more sense to build a bounded active classifier that minimizes the expected total cost per instance[2] as opposed to the expected misclassification error. Our framework can be extended to this case by making some modifications to the dynamic program we use to build

[1] The motivation is that by learning the value function for these heuristics, an RL agent could then employ an exploratory action selection policy (during planning) to try and improve upon them.

[2] Where the total cost per instance is defined as: total cost = (misclassification cost) + (cost of features purchased)

BAC^* (see [9]).

The cost structure we assume in this work can be quite different from the complex (linked) cost structure that can exist in practice. For example, a linked cost structure might charge $10.00 for a blood test X_i by itself, but charge only $2.00 for the blood test if it is purchased in combination with a bone scan X_j. For these complicated cost structures, algorithms will have to consider the value of information of a feature under multiple scenarios (e.g. when purchased by itself, when purchased at a discount after acquiring linked feature X_j, etc.). Finally, the most important direction for future research is to build upon the empirical results herein to develop algorithms with strong theoretical guarantees on learning performance.

Bibliography

[1] George E. Andrews. The theory of partitions. In *Encyclopedia of Mathematics and its Applications*. Addison-Wesley, 1976.

[2] Peter Auer, Nicolò Cesa-Bianchi, Yoav Freund, and Robert E. Schapire. Gambling in a rigged casino: the adversarial multi-armed bandit problem. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, 1995.

[3] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: a review. *Statistical Science*, 1995.

[4] Vincent A. Cicirello and Stephen F. Smith. The max k-armed bandit: a new model of exploration applied to search heuristic selection. In *The Twentieth National Conference on Artificial Intelligence (AAAI)*, 2005.

[5] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. Active learning with statistical models. In *Advances in Neural Information Processing Systems 7 (NIPS)*, 1995.

[6] D. Dobkin, D. Gunopoulos, and S. Kasif. Computing optimal shallow decision trees. In *International Workshop on Mathematics in Artificial Intelligence*, 1996.

[7] Michael Duff. *Optimal learning: computational procedures for Bayes-adaptive Markov Decision Processes*. PhD thesis, University of Massachusetts Amherst, 2002.

[8] U. Fayyad and K. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1993.

[9] Russell Greiner, Adam J. Grove, and Dan Roth. Learning cost sensitive active classifiers. *Artificial Intelligence*, 2002.

[10] David G. Heath, Simon Kasif, and Steven Salzberg. Induction of oblique decision trees. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI)*, 1993.

[11] David Heckerman. A tutorial on learning in bayesian networks. In *Learning in Graphical Models*. The MIT Press, 1999.

[12] S. Hettich, C.L. Blake, and C.J. Merz. UCI repository of machine learning databases, 1998.

[13] David Hoel and Milton Sobel. Comparisons of sequential procedures for selecting the best binomial population. In *Sixth Berkeley Symposium on Mathematical Statistics and Probability*, 1971.

[14] Aloak Kapoor and Russell Greiner. Learning and classifying under hard budgets. In *The Sixteenth European Conference on Machine Learning (ECML)*, 2005.

[15] Aloak Kapoor and Russell Greiner. Reinforcement learning for active model selection. In *International Workshop on Utility-Based Data Mining (KDD)*, 2005.

[16] Michael Kearns, Yishay Mansour, and Andrew Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 2002.

[17] Daniel J. Lizotte, Omid Madani, and Russell Greiner. Budgeted learning of naive-bayes classifiers. In *Proceedings of the Nineteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2003.

[18] Omid Madani, Daniel J. Lizotte, and Russell Greiner. Active model selection. In *Proceedings of the Twentieth Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

[19] Omid Madani, Daniel J. Lizotte, and Russell Greiner. Active model selection. Technical report, University of Alberta, 2004.

[20] Prem Melville, Maytal Saar-Tsechansky, Foster Provost, and Raymond Mooney. Active feature-value acquisition for classifier induction. In *The Fourth IEEE International Conference on Data Mining (ICDM)*, 2004.

[21] Ron Musick, Jason Catlett, and Stuart Russell. Decision theoretic subsampling for induction on large databases. In *Proceedings of the Tenth International Conference on Machine Learning (ICML)*, 1993.

[22] Foster Provost, David Jensen, and Tim Oates. Efficient progressive sampling. In *Proceedings of the fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 1999.

[23] Herbert Robbins. Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 1952.

[24] Sheldon Ross. *A First Course in Probability*. Prentice Hall, 1997.

[25] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[26] Dale Schuurmans and Russell Greiner. Sequential pac learning. In *Proceedings of the Eighth Annual Conference on Computational Learning Theory (COLT)*, 1995.

[27] Richard S. Sutton. Learning to predict by the method of temporal differences. *Machine Learning*, 1988.

[28] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning*. The MIT Press, 1998.

[29] John N. Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 1997.

[30] Peter Turney. Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 1995.

[31] Peter Turney. Types of cost in inductive concept learning. In *Workshop on cost sensitive learning (ICML)*, 2000.

[32] Tao Wang, Daniel Lizotte, Michael Bowling, and Dale Schuurmans. Bayesian sparse sampling for on-line reward optimization.. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML)*, 2005.

[33] Gary M. Weiss and Foster Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 2003.

Appendix A

Proofs

Preface to Propositions 1 and 2

To simplify the proofs of Proposition 1 and 2, we prove them under the following value function definition:

$$V^{\pi}(s_t) = E\left(\sum_{i=0}^{\infty}(\gamma^i r_{i+t}|\pi, s_t)\right) \quad . \tag{A.1}$$

Note that this definition differs trivially from the one given in Chapter 2, because we include the reward received for reaching state s_t as part of $V^{\pi}(s_t)$. This small change in "accounting" simplifies the proofs of Proposition 1 and 2 *without* changing their effective meaning. Nevertheless, after observing the way we prove the results here, it is easy to see that both propositions also hold under the alternate value function definition of Chapter 2.

A.1 Proposition 1

$$\begin{aligned} \text{Let} \quad s^{+h_i} &= (b', \ \alpha_1, \beta_1, \ldots \alpha_i + 1, \beta_i, \ldots \alpha_n, \beta_n) \\ \text{and} \quad s^{+t_i} &= (b', \ \alpha_1, \beta_1, \ldots \alpha_i, \beta_i + 1, \ldots \alpha_n, \beta_n) \end{aligned}$$

We prove the result by induction on the remaining budget b'. For the base case, let $b' = 0$.

$$\begin{aligned} \text{Now} \quad V^{\pi^*}(s^{+t_i}) &= \max\left(\frac{\alpha_1}{\alpha_1+\beta_1}, \ldots \frac{\alpha_i}{\alpha_i+\beta_i+1}, \ldots \frac{\alpha_n}{\alpha_n+\beta_n}\right) \\ \text{and} \quad V^{\pi^*}(s^{+h_i}) &= \max\left(\frac{\alpha_1}{\alpha_1+\beta_1}, \ldots \frac{\alpha_i+1}{\alpha_i+\beta_i+1}, \ldots \frac{\alpha_n}{\alpha_n+\beta_n}\right) \quad . \end{aligned}$$

Since the latter max is term by term greater than or equal to the former max, it follows that $V^{\pi^*}(s^{+h_i}) \geq V^{\pi^*}(s^{+t_i})$ for the base case, $b' = 0$.

For the inductive step, assume the result holds for $b' \leq (j-1)$, and let $b' = j$. We will use

$s^{+t_i+h_k}$ to denote the state resulting from s^{+t_i} after coin C_k is flipped, turns up heads, and reduces the budget to $b' = j - 1$. We prove the inductive step by considering two mutually exclusive cases.

Case 1: The optimal action to take from (s^{+t_i}) is to flip some coin $C_k \neq C_i$.

$$\begin{aligned} V^{\pi^*}(s^{+t_i}) = & \quad \tfrac{\alpha_k}{\alpha_k+\beta_k} V^{\pi^*}(s^{+t_i+h_k}) + \tfrac{\beta_k}{\alpha_k+\beta_k} V^{\pi^*}(s^{+t_i+t_k}) & \text{(definition } V^{\pi^*}(s^{+t_i})) \\ \leq & \quad \tfrac{\alpha_k}{\alpha_k+\beta_k} V^{\pi^*}(s^{+h_i+h_k}) + \tfrac{\beta_k}{\alpha_k+\beta_k} V^{\pi^*}(s^{+h_i+t_k}) & \text{(inductive hypothesis)} \\ \leq & \quad V^{\pi^*}(s^{+h_i}) & \text{(definition } V^{\pi^*}(s^{+h_i})) \end{aligned}$$

Case 2: The optimal action to take from (s^{+t_i}) is to flip coin C_i.

$$\begin{aligned} V^{\pi^*}(s^{+t_i}) = & \quad \tfrac{\alpha_i}{\alpha_i+\beta_i+1} V^{\pi^*}(s^{+t_i+h_i}) + \tfrac{\beta_i+1}{\alpha_i+\beta_i+1} V^{\pi^*}(s^{+t_i+t_i}) & \text{(def. } V^{\pi^*}(s^{+t_i})) \\ \leq & \quad \tfrac{\alpha_i+1}{\alpha_i+\beta_i+1} V^{\pi^*}(s^{+t_i+h_i}) + \tfrac{\beta_i}{\alpha_i+\beta_i+1} V^{\pi^*}(s^{+t_i+t_i}) & \text{(inductive hyp.)} \\ \leq & \quad \tfrac{\alpha_i+1}{\alpha_i+\beta_i+1} V^{\pi^*}(s^{+h_i+h_i}) + \tfrac{\beta_i}{\alpha_i+\beta_i+1} V^{\pi^*}(s^{+h_i+t_i}) & \text{(inductive hyp.)} \\ \leq & \quad V^{\pi^*}(s^{+h_i}) & \text{(def. } V^{\pi^*}(s^{+h_i})) \end{aligned}$$

Thus, the result holds for all possible cases, completing the inductive step.

A.2 Proposition 2

The result is proved via induction on the remaining budget b'. For the base case, let $b' = 0$.

$$\begin{aligned} \text{Now} \quad V^{\pi^*}(s) = & \quad \max\left(\tfrac{\alpha_1}{\alpha_1+\beta_1}, \cdots \tfrac{\alpha_i}{\alpha_i+\beta_i} \cdots \tfrac{\alpha_n}{\alpha_n+\beta_n}\right) \\ \text{and} \quad V^{\pi^*}(\hat{s}) = & \quad \max\left(\tfrac{\alpha_1}{\alpha_1+\beta_1}, \cdots \tfrac{\alpha_i+1}{\alpha_i+\beta_i+1} \cdots \tfrac{\alpha_n}{\alpha_n+\beta_n}\right) \quad . \end{aligned}$$

Since the latter max is term by term greater than or equal to the former max, the base case holds. For the inductive step, assume the result holds for $b' \leq (j-1)$ and let $b' = j$. We will use s^{+h_k} to denote the state resulting from s after coin C_k is flipped, turns up heads, and reduces the budget to $b' = j - 1$. We prove the inductive step by considering two mutually exclusive cases.

Case 1: The optimal action from s is to flip some coin $C_k \neq C_i$.

$$\begin{aligned} V^{\pi^*}(s) = & \quad \tfrac{\alpha_k}{\alpha_k+\beta_k} V^{\pi^*}(s^{+h_k}) + \tfrac{\beta_k}{\alpha_k+\beta_k} V^{\pi^*}(s^{+t_k}) & \text{(Definition of } V^{\pi^*}(s)) \\ \leq & \quad \tfrac{\alpha_k}{\alpha_k+\beta_k} V^{\pi^*}(\hat{s}^{+h_k}) + \tfrac{\beta_k}{\alpha_k+\beta_k} V^{\pi^*}(\hat{s}^{+t_k}) & \text{(Inductive hypothesis)} \\ & \quad \leq V^{\pi^*}(\hat{s}) & \text{(Definition of } V^{\pi^*}(\hat{s})) \end{aligned}$$

Case 2: The optimal action from s is to flip coin C_i.

$$V^{\pi^*}(s) = \quad \tfrac{\alpha_i}{\alpha_i+\beta_i} V^{\pi^*}(s^{+h_i}) + \tfrac{\beta_i}{\alpha_i+\beta_i} V^{\pi^*}(s^{+t_i}) \qquad \text{(Definition of } V^{\pi^*}(s))$$

$$\begin{aligned} &\leq \tfrac{\alpha_i}{\alpha_i+\beta_i} V^{\pi^*}(\hat{s}^{+h_i}) + \tfrac{\beta_i}{\alpha_i+\beta_i} V^{\pi^*}(\hat{s}^{+t_i}) && \text{(Inductive hypothesis)} \\ &\leq \tfrac{\alpha_i+1}{\alpha_i+\beta_i+1} V^{\pi^*}(\hat{s}^{+h_i}) + \tfrac{\beta_i}{\alpha_i+\beta_i+1} V^{\pi^*}(\hat{s}^{+t_i}) && \text{(Proposition 1)} \\ &\leq V^{\pi^*}(\hat{s}) && \text{(Definition of } V^{\pi^*}(\hat{s})\text{)} \end{aligned}$$

Thus, the result holds for both possible cases, completing the inductive step.

A.3 Proposition 3

We use a non-terminal state to obtain the result. Consider a state, Q, in which $b' = 1$, there exists two Beta(3, 2) coins, and $(n-2)$ Beta(2, 2) coins. It is easy to verify (using Equation 2.3) that the optimal action in Q is strictly to flip a Beta(3, 2) coin. To prove the proposition, we show that BR encounters at least g different variants of Q in which it chooses to flip a Beta(2, 2) coin.

Let there be $n = g + 2$ coins, and a budget of $b = 2n + 3$. Notice the budget is such that state Q is guaranteed to occur under BR's strategy. In fact, Q occurs multiple times because there are $\binom{n}{2}$ distinct ways to place the two Beta(3, 2) coins. We also note that since the number of tails on all n coins is equal, we are guaranteed that BR will be currently flipping the first coin in the set. Thus, BR will make a suboptimal decision whenever it reaches state Q with the first coin being one of the Beta(2, 2)s. Observe that there are $\binom{n-1}{2}$ distinct versions of state Q in which the first coin is a Beta(2, 2). Now the proposition follows from the fact that: $\binom{n-1}{2} = \frac{(n-1)(n-2)}{2} = \frac{(g+1)g}{2} \geq g$ for all $g \geq 1$.

A.4 Proposition 4

We use two lemmas to aid in the proof. The first is a standard result from the theory of partitions [1]:

Lemma 1 *There are* $\binom{n-1}{b-1}$ *ways to express an integer* $n \geq 1$ *as the sum of exactly* b *positive integers.*

while the second lemma can be derived from the first [24]:

Lemma 2 *There are* $\sum_{j=1}^{b} \binom{n-1}{j-1}\binom{b}{j} = \binom{n+b-1}{b-1}$ *ways to express an integer* $n \geq 1$ *as the sum of* b *nonnegative integers.*

Let $d = |Y| \sum_i |X_i|$. Working from the bottom-up, the dynamic program (dp) must begin by calculating the value of all possible terminal states. Using our Naïve Bayes assumption and the unit cost of features, each unique terminal state corresponds to a complete allocation of the learning budget b_L over the d Dirichlet parameters. Thus, the number of distinct terminal states (that the dynamic program has to solve) is equal to the number of ways to express the learning budget b_L as the sum of d nonnegative integers. Using Lemma 2, the dp computes the value of

$$\frac{(b_L + d - 1)!}{(b_L)!(d-1)!} \tag{A.2}$$

states at the bottom level; using Stirling's formula on each factorial, we get

$$\begin{aligned} \frac{(b_L+d-1)!}{(b_L)!(d-1)!} &> \\ \frac{\left(\frac{b_L+d-1}{b_L}\right)^{b_L}\left(\frac{b_L+d-1}{d-1}\right)^{d-1}\sqrt{2\pi(b_L+d-1)}}{\sqrt{2\pi b_L}\sqrt{2\pi(d-1)}(1+\frac{1}{11b_L})(1+\frac{1}{11(d-1)})} &> \\ \frac{\left(\frac{b_L+d-1}{b_L}\right)^{b_L}\left(\frac{b_L+d-1}{d-1}\right)^{d-1}}{2\sqrt{2\pi}\sqrt{d-1}} &\in \\ \Omega\left(\left(\frac{b_L+d-1}{b_L}\right)^{b_L}\left(\frac{b_L+d-1}{d-1}\right)^{d-1}(d-1)^{\frac{-1}{2}}\right) \end{aligned}$$

and the result follows using $t = (d-1)$.

A.5 Proposition 5

To prove Proposition 5, the following lemma is required.

Lemma 3 *Let the Naïve Bayes assumption hold, and consider any set D of Dirichlets over the feature-class pairs and a bounded active classifier BAC_D (with bound b_C) constructed from D. Given any set of Dirichlets D' where D' can be made equal to D by specifying exactly one proper permutation for each feature, there exists a bounded active classifier $BAC_{D'}$ (also with bound b_C) constructed from D' such that the expected error of BAC_D is equal to the expected error of $BAC_{D'}$.*

We prove this lemma first, before moving on to Proposition 5. Let $P(.)_D$ denote a probability under D, and $P(.)_{D'}$ denote a probability under D'. Let b be a branch of BAC_D, which, without loss of generality, specifies some feature values $(X_i = x_i, X_j = x_j)$, and has classification label $Y = y$. Then the expected accuracy of branch b is

$$\begin{aligned} P(X_i = x_i, X_j = x_j, Y = y)_D &= \\ P(X_i = x_i|Y = y)_D P(X_j = x_j|Y = y)_D P(Y = y) &= \\ P(X_i = x'_i|Y = y)_{D'} P(X_j = x'_j|Y = y)_{D'} P(Y = y) \end{aligned}$$

where x'_i is the image of x_i under the proper permutation for X_i. Thus we have converted a branch b of BAC_D into a new branch b', where the expected accuracy of b' under D' is the same as the expected accuracy of b under D. We can repeat this conversion for each branch of BAC_D to get a set of new branches which, when summed together, have the same expected accuracy as BAC_D. Of course, since the expected misclassification error is 1−(expected accuracy), the new branches have the same expected misclassification error as BAC_D as well.

All that remains to be shown is that the set of new branches forms a valid BAC with bound b_C. To see this, note that we can apply our transformation by doing a preorder traversal of BAC_D, where at each non-leaf node specifying feature X_k, we reorder its subtrees using the proper permutation for feature X_k. A reordering of subtrees cannot invalidate the BAC, nor can it increase the bound

b_C. Once the entire tree has been traversed, we are guaranteed to have applied our transformation to each feature of each branch, ensuring that each branch has been fully converted. The converted tree is the desired $BAC_{D'}$.

This completes the proof of the lemma. Now we can prove the original proposition.

Let us adopt the notation that D_{s_a} denotes the Dirichlets of state s_a. Further, let $D_{s_a} + (ijd)$ denote the Dirichlets of state s_a after observing $X_i = d$ on a $Y = j$ instance. Finally, let f_i denote the proper permutation for feature X_i, and $dom(X_i)$ denote the domain of feature X_i.

The proof follows from induction on b_L. In the base case, $b_L = 0$. Since no learning budget remains in state s_a or s_b, there is no action to take, and hence trivially state s_a and s_b have the same (null) optimal action. When $b_L = 0$ the value of state s_a under an optimal policy is simply the expected misclassification error of the BAC^* constructed from state s_a's Dirichlets. By Lemma 3, state s_b must have a corresponding BAC with exactly the same expected misclassification error. Furthermore, the value of state s_b under an optimal policy cannot be any less, for if it were, then Lemma 3 implies that state s_a must have a corresponding BAC with lower expected error, which is a contradiction to the definition of BAC*. Thus states s_a and s_b have identical values under the optimal policy for the base case.

For the inductive step, assume the result holds for $b_L = n - 1$, and let states s_a and s_b have $b_L = n$. Now consider taking *any* initial action from state s_a, and then following an optimal policy. Let $V^{\pi^*}(s_a|X_i, Y = j)$ denote the value of purchasing feature X_i on a random $Y = j$ instance from state s_a, and then following an optimal policy. We have:

$$\begin{gathered}
V^{\pi^*}(s_a|X_i, Y = j) = \\
\textstyle\sum_{d\in dom(X_i)} P(X_i = d|Y = j)_{D_{s_a}} V^{\pi^*}(D_{s_a} + (ijd), b_L = n - 1) = \\
\textstyle\sum_{d\in dom(X_i)} P(X_i = f_i(d)|Y = j)_{D_{s_b}} V^{\pi^*}(D_{s_a} + (ijd), b_L = n - 1) = \\
\textstyle\sum_{d\in dom(X_i)} P(X_i = f_i(d)|Y = j)_{D_{s_b}} V^{\pi^*}(D_{s_b} + (ijf_i(d)), b_L = n - 1) = \\
V^{\pi^*}(s_b|X_i, Y = j)
\end{gathered}$$

where the second to last equality follows by an application of the inductive hypothesis, since $D_{s_a} + (ijd)$ can be made equal to $D_{s_b} + (ijf_i(d))$ by using the r proper permutations, one for each feature. Thus, we have just shown that the value of an action in state s_a is equal to the value of the same action from state s_b, when the action is followed by an optimal policy. This implies that the value of the two states under an optimal policy is equal, and that the two states have identical optimal actions. This completes the inductive step.

Appendix B

Features for RL Function Approximation

B.1 Feature Groups

The following list describes the features that were used to approximate the value function for our RL agents in Chapter 2.

Budget

- remaining budget (b')

Beta Hyperparameters

- $\alpha_i \quad \forall i = 1..n$
- $\beta_i \quad \forall i = 1..n$

Means and Standard Deviations

- $\mu_i \quad \forall i = 1..n$
- $\sigma_i \quad \forall i = 1..n$

Mean Stats

- $\max_i \mu_i$
- $\min_i \mu_i$
- $\sum_i \mu_i$

Lookahead Stats

- $\max_i \frac{\alpha_i + b'}{\alpha_i + \beta_i + b'}$
- $\frac{\sum_i \left(\frac{\alpha_i + b'}{\alpha_i + \beta_i + b'} \right)}{n}$

Confidence Interval Stats

- $\max_i \, (\mu_i + 1.96\sigma_i)$ (95% interval)
- $\max_i \, (\mu_i + 1.28\sigma_i)$ (80% interval)
- $\max_i \, (\mu_i + 0.67\sigma_i)$ (50% interval)
- $\max_i \, (\mu_i + 0.126\sigma_i)$ (10% interval)
- $\sum_i(\mu_i + 1.96\sigma_i)$
- $\sum_i(\mu_i + 0.126\sigma_i)$
- $\max_i \, (\mu_i + b' \times \sigma_i)$
- $\sum_i(\mu_i + b' \times \sigma_i)$

B.2 Alternate Features

This section describes several other features that we experimented with when trying to approximate the value function for our RL agents. Similar to the results of Chapter 2, these alternate feature were unable to consistently beat the simple heuristic policies.

Table B.1: Other features tested for RL function approximation

Feature	Comments
$\max_i \sigma_i$	
$\min_i \sigma_i$	
$\sum_i \sigma_i$	
standard dev. of greedy coin	
standard dev. of max mean coin	
max SCL score on any coin	helps simulate SCL
max # of flips on any coin	helps simulate RR
min # of flips on any coin	helps simulate RR
max # of heads on any coin	
min # of heads on any coin	
max # of tails on any coin	helps simulate BR
min # of tails on any coin	helps simulate BR
max $\|\mu_i - \mu_j\|$	
min $\|\mu_i - \mu_j\|$	
max $\|\sigma_i - \sigma_j\|$	
min $\|\sigma_i - \sigma_j\|$	
max $\|\alpha_i - \alpha_j\|$	
min $\|\alpha_i - \alpha_j\|$	
max $\|\beta_i - \beta_j\|$	
min $\|\beta_i - \beta_j\|$	
$\max_i \frac{\alpha_i}{\beta_i}$	
$\min_i \frac{\alpha_i}{\beta_i}$	
max # of identical coins	
$\max_i \mu_i \times \max_i \sigma_i$	
$\sum_i \mu_i \times \sum_i \sigma_i$	
$\max_i \frac{\alpha_i + b'}{\alpha_i + \beta_i + b'} \times \max_i \sigma_i$	
$\sum_i \frac{\alpha_i + b'}{\alpha_i + \beta_i + b'} \times \sum_i \sigma_i$	
min # of tails for the max mean coin to lose its max mean spot	
min # of heads for a non-max mean coin to become max mean coin	