Learning from Interaction: Models and Applications

Dorota Głowacka

A dissertation submitted in partial fulfilment

of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

October 2012

University College London

I, Dorota Głowacka confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the thesis.

Abstract

A large proportion of Machine Learning (ML) research focuses on designing algorithms that require minimal input from the human. However, ML algorithms are now widely used in various areas of engineering to design and build systems that interact with the human user and thus need to "learn" from this interaction. In this work, we concentrate on algorithms that learn from user interaction. A significant part of the dissertation is devoted to learning in the bandit setting. We propose a general framework for handling dependencies across arms, based on the new assumption that the mean-reward function is drawn from a Gaussian Process. Additionally, we propose an alternative method for arm selection using Thompson sampling and we apply the new algorithms to a grammar learning problem.

In the remainder of the dissertation, we consider content-based image retrieval in the case when the user is unable to specify the required content through tags or other image properties and so the system must extract information from the user through limited feedback. We present a novel Bayesian approach that uses latent random variables to model the systems imperfect knowledge about the users expected response to the images. An important aspect of the algorithm is the incorporation of an explicit exploration-exploitation strategy in the image sampling process. A second aspect of our algorithm is the way in which its knowledge of the target image is updated

given user feedback. We considered a few algorithms to do so: variational Bayes, Gibbs sampling and a simple uniform update. We show in experiments that the simple uniform update performs best. The reason is because, unlike the uniform update, both variational Bayes and Gibbs sampling tend to focus on a small set of images aggressively.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Broadly speaking, machine learning studies the principles that govern learning processes, be it in artificial systems (as used in modern technology) or natural systems (e.g. in humans or animals). Theories and algorithms from machine learning are relevant to understanding aspects of human learning (e.g. Tenenbaum et al. (2011)). For example, neural networks have been a valuable tool for psychologists as a computational model of the way brains function (e.g. Arbib (1995)); reinforcement learning agrees well with the neural activity of dopaminergic neurons during reward-based learning (e.g. Glimcher (2011)); and sparse representations in computer vision predict well the visual features found in the early visual cortex (e.g. Boureau et al. (2010)). At the other end of the spectrum, many aspects of research conducted in the fields of psychology, cognitive science, philosophy or even physiology can be of great use to machine learning researchers, since people still learn languages,

concepts, and causal relationships from far less data than any automated system. Deeper understanding of human cognition and insights from human learning can help us to improve machine learning systems. Most research in the area of machine learning, or, broadly speaking, artificial intelligence, concentrates on designing systems that mimic human behaviour. However, as artificial systems are becoming more and more advanced and autonomous, new challenges for machine learning arise. Thus, it is not sufficient to build a system that is pre-programmed to mimic human behaviour or to achieve a certain goal. Present-day artificial systems must adjust their behaviour in accordance with the needs of the users and the surrounding environment, e.g. employing robots in rescue operations or the health care system. Consequently, machine learning researchers must design algorithms that will allow the artificial system to be constantly "learning" through interaction with its environment and its users.

In this work, we concentrate on algorithms and systems that learn from user interaction. A significant portion of the dissertation is devoted to the study of learning in the bandit setting. We propose a general framework for handling dependencies across arms, based on a new assumption that the mean-reward function is drawn from a Gaussian Process. Additionally, we propose an alternative method for arm selection using Thompson sampling. We successfully apply the new algorithms to the grammar learning problem in the *Principles and Parameters* setting (Chomsky (1981)).

In the remainder of the dissertation, we consider content-based image re-

trieval in the case when the user is unable to specify the required content through tags or other image properties and so the system must extract information from the user through limited feedback. We present a novel Bayesian approach that uses latent random variables to model the systems imperfect knowledge about the users expected response to the images. The proposed approach compares favourably with previous work. An important aspect of the proposed algorithm is the incorporation of an explicit exploration-exploitation strategy in the image sampling process, which greatly improves the performance of the algorithm when compared to its main competitors that do not employ exploration-exploitation strategies. A second aspect of our algorithm is the way in which its knowledge of the target image is updated given user feedback. We considered a few algorithms to do so: variational Bayes, Gibbs sampling and a simple uniform update. We show in experiments that the simple uniform update performs best. The reason is because, unlike uniform updates, both variational Bayes and Gibbs sampling tend to focus on a small set of images (which may or may not contain the target) aggressively.

The dissertation is organised as follows: in Chapter 2, we lay out the theoretical background necessary for understanding the algorithms and applications developed in the remainder of the thesis. The main theoretical concepts described in Chapter 2 are Bayesian inference, the Dirichlet distribution and the related Dirichlet Process as well as the Gaussian Process. Chapter 3 is devoted to independent bandit algorithms, where we introduce

the concept of simple bandits and describe the UCB family of algorithms (Auer et al. (2002a)) as well as algorithms based on empirical variance of each arm such as UCB-V (Audibert et al. (2009)) and Thompson sampling bandits (Chapelle and Li (2011)). We propose a new approach based on a Beta prior for tuning the regret bound. The experimental results show that our approach outperforms UCB algorithms as well as empirical variance based bandit algorithms. In Chapter 4, we concentrate on dependent arm bandits. Initially, we review recent literature on contextual bandits algorithms, such as LinRel (Auer (2002)) and LinUCB (Chu et al. (2011)). The main idea of both algorithms is to compute the expected reward of each arm by finding a linear combination of the previous rewards of the arm. Next, we propose a new Gaussian Process Bandits algorithm, where the mean reward function is drawn from a Gaussian Process with a given covariance matrix, which allows us to assess how similar two arms are, and, unlike LinRel and LinUCB, does not rely on a simple linear combination of previous awards. Using the 2-dimensional Rosenbrock function, we assess the performance of the Gaussian Process Bandit against LinRel and LinUCB with favourable results. In Chapter 5, we apply Gaussian Process Bandits to a grammar learning problem. We show that the Gaussian Process Bandit can be successfully applied to the scenario of modelling language acquisition thus combining the fields of machine learning and cognitive science. One of the main contributions of this chapter is applying computational techniques to modelling acquisition of metrical stress system, while most literature on modelling language ac-

quisition concentrates mostly on the syntax level. In Chapter 6, we consider content-based image retrieval in the case when the user is unable to specify the required content through tags or other image properties. We present a novel Bayesian approach that uses latent random variables to model the system's imperfect knowledge about the user's expected response to the images. The algorithm uses a Dirichlet process over the variables θ, which are the probabilities how likely each image is to the one that the user is looking for, to represent the state of its knowledge. At each iteration, the system samples k images to present to the user, from which the user selects the one closest to the target. Thus, we call the algorithm Dirichlet Sampling algorithm. The aim of the algorithm is to allow the user to find the target image in as few iterations as possible. An important aspect of the proposed algorithm is the incorporation of an explicit exploration-exploitation strategy in the image sampling process, which greatly improves the performance of the algorithm compared to its main competitors, such as *PicHunter* (Cox et al. (2000)), that do not employ exploration-exploitation strategies. Additionally, we propose a number of heuristics that will allow the algorithm to be incorporated into real-life online retrieval systems that may contain databases consisting of millions of images. The initial experiments involving 4 million Flickr images show that when combined with heuristics, such as *sparse data representation*, the Dirichlet Search algorithm can be successfully applied to very large datasets.

The work presented in this thesis is largely based on the following work:

1. Głowacka, D. and John Shawe-Taylor, "Simple bandits revisited" (breaking-news abstract), *AISTATS 15*, La Palma, 2012.

2. Głowacka, D., Medlar, A. and J. Shawe-Taylor, "Sifting through images with multinomial relevance feedback", *NIPS Workshop Beyond Classification: Machine Learning for Next Generation Computer Vision Challenges*, Whistler, Canada, 2010.

3. Głowacka, D. and J. Shawe-Taylor, "Content-based image retrieval with multinomial relevance feedback", In: Sugiyama, M. & Q. Yang (eds.) *JMLR Workshop and Conference Proceedings Volume 13: 2nd Asian Conference on Machine Learning, Tokyo, 2010.* 111-125, 2010.

4. Głowacka, D. and J. Shawe-Taylor, "Multinomial relevance feedback for content-based image retrieval", *ICML Workshop on Reinforcement Learning and Search in Very Large Spaces*, Haifa, Israel, 2010.

5. Głowacka, D., Dorard, L., Medlar, A. and J. Shawe-Taylor, "Prior knowledge in learning finite parameter spaces". In: de Groote, P., Egg, M. and L. Kallmeyer (eds.) *Proceedings of the 14th Conference on Formal Grammar, Bordeaux, 2009.* Springer, LNAI 5591, 199 – 213, 2011.

6. Dorard, L., Głowacka, D. and J. Shawe-Taylor, "Gaussian processes modelling of dependencies in multi-armed bandit problems". In: *Pro-*

ceedings of the 10th International Symposium on Operational Research, Nova Gorica, Slovenia, 2009.

Chapter 2

Theoretical Background

2.1 Introduction

The task of data modelling occurs in many areas of research. We design a model for a particular system and when data arrives from the system, we adapt the model in light of the data. The model provides us with a representation of both our prior beliefs about the system and the information about the system that the data has provided. We can then use the model to make inferences. However, we need a consistent framework within which to construct our model incorporating any prior knowledge and within which to consistently compare our model with others. In this thesis, we concentrate mostly on Bayesian modelling. The Bayesian approach is based upon the expression of knowledge in terms of probability distributions. Given the data and a specific model, we can deterministically make inferences using

the rules of probability theory. In principle Bayesian theory always provides us with a unique solution to our data modelling problems. In practice such solutions may be difficult to find. As mentioned in the previous chapter, in this thesis we concentrate on a number of applications based on Gaussian Process and Dirichlet Process priors. In this chapter, we briefly introduce the basic concepts related to Bayesian inference and their applications to Gaussian Processes and Dirichlet Processes.

2.2 Bayesian Inference

We have constructed a model or hypothesis $\mathcal{H}_i$ with a set of parameters w_i which we shall use to model some data $\mathcal{D}$. Within the Bayesian approach to data modelling there are then two steps. The first step is to infer the parameters of the model given the data. In the next step, we wish to compare our present model with others. We start by expressing our prior beliefs about which models are *a priori* most suitable for the data in terms of a probability distribution over all possible models $P(\mathcal{H}_i)$. If a model has a set of parameters w_i, then we express our prior beliefs about the value of these parameters in terms of another prior distribution $P(w_i \mid \mathcal{H}_i)$. The data now arrives and we infer the parameters of the model given the data. Each model makes predictions about how likely the data is given that it was generated by that model with a specific set of parameters w_i. These predictions are embodied in the distribution $P(\mathcal{D} \mid w_i, \mathcal{H}_i)$.

Next, we need to combine the prior knowledge that we had about the setting of the parameters $P(w_i \mid \mathcal{H}_i)$, with the knowledge that we have just gained from the data $P(\mathcal{D} \mid w_i, \mathcal{H}_i)$. We can use Bayes' theorem to combine the two pieces of information:

$$P(w_i \mid \mathcal{D}, \mathcal{H}_i) = \frac{P(\mathcal{D} \mid w_i, \mathcal{H}_i)P(w_i \mid \mathcal{H}_i)}{P(\mathcal{D} \mid \mathcal{H}_i)}. \tag{2.1}$$

In other words, the total information we have about the value of the parameters, which is embodied in the posterior probability of the parameters $P(w_i \mid \mathcal{D}, \mathcal{H}_i)$ is the product of $P(\mathcal{D} \mid w_i, \mathcal{H}_i)$ (the likelihood of the data) and $P(w_i \mid \mathcal{H}_i)$ (the prior), with $P(\mathcal{D} \mid \mathcal{H}_i)$ (the evidence) as an appropriate normalising constant. Given that we can calculate $P(\mathcal{D} \mid \mathcal{H}_i)$, we can apply Bayes' theorem once more. $P(\mathcal{D} \mid \mathcal{H}_i)$ embodies what the data is telling us about the plausibility of each of the models $\mathcal{H}_i$ and we can combine this with our prior beliefs:

$$P(\mathcal{H}_i \mid \mathcal{D}) = \frac{P(\mathcal{D} \mid \mathcal{H}_i)P(\mathcal{H}_i)}{P(\mathcal{D})} \tag{2.2}$$

where $P(\mathcal{D})$ is a normalising constant. The posterior distribution $P(\mathcal{H}_i \mid \mathcal{D})$ allows us to rank our different models.

Applying Bayesian methods, however, is not always straightforward. There are three principal areas of difficulty. The first one centres around the mathematical complexity that often occurs in Bayesian approaches - approximations often have to be made to avoid intractable integrals and awkward equations. Most commonly, problems arise in evaluating the evidence. Another

potential problem with the Bayesian approach is the prior. The problem with priors lies with the difficult process of assigning probabilities to our beliefs about the priors. How do we decide what numbers truly reflect what we believe? Finally, Bayesian model comparison has a closed hypothesis space. If we fail to include the "true" model amongst our set of possible candidates, then this model will never be compared with others. There exists no Bayesian criterion for assessing whether or not our hypothesis space is correct. Thus, in certain parts of this thesis, we will apply various approximation techniques that will allow us to apply Bayesian inference to our problems.

2.3 Dirichlet Distribution and Dirichlet Processes

The Dirichlet Process, and the related Dirichlet distribution, are used extensively in Bayesian inference. In this section, we briefly introduce both terms and describe how to sample from the Dirichlet distribution.

2.3.1 Dirichlet Distribution

The Dirichlet distribution (Sjolander et al. (1996); Teh (2010); Ferguson (1973); Blackwell and MacQueen (1973); Neal (1992); Rasmussen (2000)) is a multi-parameter generalisation of the Beta distribution and defines a distribution over distributions. Let $\Theta = \{(\theta_1, \theta_2, \ldots, \theta_n) \mid \theta_i \geq 0 \wedge \sum_i^n \theta_i = 1\}$ be a

multinomial probability distribution on the discrete space $\mathscr{X} = \{\mathscr{X}_1, \ldots, \mathscr{X}_n\}$ such that $P(x = \mathscr{X}_i) = \theta_i$, where x is a random variable in the space $\mathscr{X}$. The Dirichlet distribution on Θ is given by the following formula:

$$P(\Theta \mid \alpha, M) = \frac{\Gamma(\alpha)}{\prod_{i=1}^{n} \Gamma(\alpha m_i)} \prod_{i=1}^{n} \theta_i^{\alpha m_i - 1} \tag{2.3}$$

where $\mathscr{M} = \{M = (m_1, m_2, \ldots, m_n) \mid m_i > 0 \wedge \sum_i^n m_i = 1\}$ is the set of possible base measures defined on $\mathscr{X}$ and the selected M being the mean value of Θ, and α the precision parameter that specifies how concentrated the distribution is around M. α can be regarded as the number of (pseudo) measurements observed to obtain M, i.e. the number of events relating to the random variable x observed apriori. The greater the value of α, the more the distribution is concentrated around M. $\Gamma(\alpha)$ denotes the Gamma function (see Equation 2.19 below), which is a generalisation of the factorial function. If α is a positive integer, then

$$\Gamma(\alpha) = (\alpha - 1)!$$

We describe Θ as having a Dirichlet distribution with parameters M and α, which is denoted by

$$\Theta \sim Dir(\alpha, M)$$

Beta Distribution

When $n = 2$, the Dirichlet distribution reduces to the Beta distribution. The Beta distribution, $Beta(\alpha, \beta)$, is defined on the (0, 1) interval and has density function:

$$P(p \mid \alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}. \tag{2.4}$$

To make the connection clear, note that if

$$p \sim Beta(a, b)$$

then

$$\Theta = (p, 1-p) \sim Dir(\alpha, M)$$

where

$$M = \{\frac{a}{a+b}, \frac{a}{a+b}\}$$

and

$$\alpha = a + b$$

2.3.2 Posterior Update using the Multinomial Distribution

The multinomial distribution is characterised by two parameters: $k > 0$, which is the number of trials, and $p = \{(p_1, \ldots, p_n) \mid p_i \geq 0 \wedge \sum_{i=1}^{n} p_i = 1\}$, which indicates the probabilities of each outcome i. Thus, if we have k

independent events, and for each event, the probability of outcome i is p_i, then the multinomial distribution specifies the probability that outcome i occurs y_i times, for $i = 1, 2, \ldots, c$. If $Y = (y_1, \ldots, y_c) \sim Mult(k, p)$, then its probability mass function is given by:

$$P(y_1, \ldots, y_c \mid k, p = (p_1, \ldots, p_c)) = \frac{k!}{y_1! y_2!, \ldots, y_c!} \prod_{i=1}^{c} p_i^{y_i} \tag{2.5}$$

The Dirichlet distribution serves as a conjugate prior for the probability parameter p of the multinomial distribution.

We can use this relationship between the Dirichlet distribution and the multinomial distribution in order to obtain the posterior updates for Θ. Consider n observations $x_1, \ldots, x_n$ that are multinomially distributed according to Θ. If y_i is the number of times the event i is observed in the n observations, the posterior probability of Θ can be obtained using Bayes Rule as follows:

$$P(\Theta \mid \alpha, M, x_{1:n}) \propto P(x_{1:n} \mid \alpha, M, \Theta) P(\Theta \mid \alpha, M) \tag{2.6}$$

$$\propto (\prod_{i=1}^{n} \theta_i^{y_i})(\prod_{i=1}^{n} \theta_i^{\alpha m_i - 1}) \tag{2.7}$$

$$\propto \prod_{i=1}^{n} \theta_i^{\alpha m_i + y_i - 1} \tag{2.8}$$

$$\propto Dir(\Theta; \alpha^\star, M^\star) \tag{2.9}$$

where

$$\alpha^\star = \alpha + n \tag{2.10}$$

$$M^\star = \frac{\alpha M + n\hat{F}}{\alpha + n} \tag{2.11}$$

where $\hat{F}$ is the empirical distribution, i.e. the proportion of occurrences of the n events in the observations. Thus, the posterior is again a Dirichlet distribution with updated parameters.

Posterior Updates Using Binomial Distribution

When $n = 2$, the multinomial distribution reduces to the binomial distribution. The binomial distribution is the discrete probability distribution of the number of successes in a sequence of k independent experiments, each of which yields success with probability p. The probability of getting exactly y successes in k trials is given by:

$$P(y \mid k, p) = \frac{k!}{y!(k-y)!} p^y (1-p)^{k-y} \tag{2.12}$$

The Beta distribution can be seen as the posterior probability of the parameter p of a binomial distribution after observing $\alpha - 1$ successes and $\beta - 1$ failures given a uniform prior. More generally, if $(y \mid p) \sim Bin(k, p)$ and

$(p \mid \alpha, \beta) \sim Beta(\alpha, \beta)$, then

$$P(p \mid \alpha, \beta, y) \propto P(y \mid k, p)P(p \mid \alpha, \beta) \tag{2.13}$$

$$\propto (p^y(1-p)^{k-y})(p^{\alpha-1}(1-p)^{\beta-1}) \tag{2.14}$$

$$\propto p^{y+\alpha-1}(1-p)^{k-y+\beta-1} \tag{2.15}$$

$$\propto Beta(\alpha^\star, \beta^\star) \tag{2.16}$$

where

$$\alpha^\star = y + \alpha - 1 \tag{2.17}$$

$$\beta^\star = k - y + \beta - 1 \tag{2.18}$$

Thus, the posterior is a Beta distribution with updated parameters.

2.3.3 Sampling from the Dirichlet Distribution

An important aspect with regards to any distribution is how to sample from it. There are a number of methods to sample from the Dirichlet distribution:

1. Polya's urn (Johnson and Kotz (1977));
2. the "stick breaking" approach (Teh et al. (2007));
3. a method based on transforming Gamma distributed random variables.

In Polya's urn, we start off with placing α_i balls of colour $i = 1, \ldots, k$ in an urn. At each iteration, we draw one ball uniformly at random from the

urn, put it back in the urn and add a ball of the same colour. As the number of iterations increases, the proportions of each type of ball will converge to a pmf that is a sample from the distribution $Dir(\alpha)$.

Mathematically, the process can be described as follows:

1. Set a counter to $n = 1$ and draw ball $x_1 \sim \frac{\alpha}{\alpha_0}$.

2. Update the counter to $n+1$ and draw ball $x_{n+1}|x_1, \ldots, x_n \sim \frac{\alpha_n}{\alpha_{n0}}$, where $\alpha_n = \alpha + \sum_{i=1}^{n} \sigma_{x_i}$ and α_{n0} is the sum of entries of α_n.

Asymptotically, the probability of drawing balls of each colour is given by a pmf that is a realisation of the distribution $Dir(\alpha)$.

The "stick breaking" approach involves iteratively breaking a stick of length 1 into k pieces in such a way that the lengths of the k pieces follow a $Dir(\alpha)$ distribution. To illustrate the procedure, let us assume that $k = 3$. Over the course of the stick breaking process, we will be keeping track of a set of intermediate values $\{u_i\}$, which will be used to calculate the realization $Q \sim Dir(\alpha)$. First, we generate q_1 from $Beta(\alpha_1, \alpha_2 + \alpha_3)$ and set $u_1 = q_1$. Next, we generate $(\frac{q_2}{1-q_1}|q_1$ from $Beta(\alpha_2, \alpha_3)$, denote the result by u_2, and set $q_2 = (1-u_1)u_2$. The resulting vector $u = (u_1, (1-u_1)u2, 1-u_1-(1-u_1)u_2)$ comes from a Dirichlet distribution with parameter vector α. This procedure can be generalised to $k > 3$.

In this work, we sample from the Dirichlet distribution using the third method as this is the most computationally efficient one (Frigyik et al. (2010)). Below, we describe how to produce Dirichlet distributed random

variables based on transforming Gamma distributed random variables (Frigyik et al. (2010)). This method has two steps:

- Step 1 – Generate gamma random variables, i.e. for $i = 1, \ldots, n$, draw a number z_i from $\Gamma(\alpha m_i, 1)$.
- Step 2 – Normalise z to obtain $p_i = \frac{z_i}{z}$ with $z = \sum_{j=1}^{n} z_j$, then p is a realisation of $Dir(\alpha, M)$.

The Gamma distribution $\Gamma(\alpha m_i, \beta)$ is defined by the following probability density:

$$P(x; \alpha m_i, \beta) = \frac{\beta^{\alpha m_i} x^{\alpha m_i - 1} e^{-\beta x}}{\Gamma(\alpha m_i)} \tag{2.19}$$

$\alpha m_i > 0$ is called the shape parameter, and $\beta > 0$ is called the scale parameter. The Gamma distribution has the following important property: Suppose $x_i \sim \Gamma(\alpha m_i, \beta)$ are independent for $i = 1, \ldots, n$, i.e. they are on the same scale but have different shapes, then

$$S = \sum_{i=1}^{n} x_i \sim \Gamma(\sum_{i=1}^{n} \alpha m_i, \beta)$$

.

To show that the procedure described above creates Dirichlet samples, we use the change-of-variables formula to prove that the density of p is the density corresponding to the $Dir(\alpha, M)$ distribution (Frigyik et al. (2010)). The variables obtained in step 1 are $\{z_1, \ldots, z_n\}$, and the new variables obtained

in step 2 are $\{z, p_1, \ldots, p_{n-1}\}$. We relate them using the transformation T:

$$(z_1, \ldots, z_n) = T(z, p_1, \ldots, p_{n-1}) = (zp_1, \ldots, zp_{n-1}, z(1 - \sum_{i=1}^{k-1} p_i)). \quad (2.20)$$

The standard change-of-variables formula tells us that the density of $(z, p_1, \ldots, p_{n-1})$ is $f = g \circ T \times \mid \det(T) \mid$, where

$$g(z_1, \ldots, z_n; \alpha m_1, \ldots, \alpha m_n) = \prod_{i=1}^{n} z_i^{\alpha m_i - 1} \frac{e^{-z_i}}{\Gamma(\alpha m_i)} \quad (2.21)$$

is the joint density of the original (independent) random variables. Substituting the above equation into the change of variables formula, we obtain the joint density of the new random variables:

$$\begin{aligned} f(z, p_1, \ldots, p_{n-1}) &= (\prod_{i=1}^{n-1} (zp_i)^{\alpha m_i - 1} \frac{e^{-zp_i}}{\Gamma(\alpha m_i)})[(z(1 - \sum_{i=1}^{n-1} p_i))^{\alpha m_n - 1} \frac{e^{-z(1-\sum_{i=1}^{n-1} p_i)}}{\Gamma(\alpha m_n)}] z^{n-1} \\ &= \frac{(\prod_{i=1}^{n-1} p_i^{\alpha m_i - 1})(1 - \sum_{i=1}^{n-1} p_i)^{\alpha m_n - 1}}{\prod_{i=1}^{n} \Gamma(\alpha m_i)} z^{(\sum_{i=1}^{n} \alpha m_i) - 1} e^{-z}. \quad (2.22) \end{aligned}$$

Integrating over z, the marginal distribution of $\{p_1, \ldots, p_{n-1}\}$ is:

$$\begin{aligned} f(p) = f(p_1, \ldots, p_{n-1}) &= \int_0^\infty f(z, p_1, \ldots, p_{n-1}) dz \quad (2.23) \\ = \frac{(\prod_{i=1}^{n-1} p_i^{\alpha m_i - 1})(1 - \sum_{i=1}^{n-1} p_i)^{\alpha m_n - 1}}{\prod_{i=1}^{n} \Gamma(\alpha m_i)} &\int_0^\infty z^{(\sum_{i=1}^{n} \alpha m_i) - 1} e^{-z} dz \quad (2.24) \\ = \frac{\Gamma(\sum_{i=1}^{n} \alpha m_i)}{\prod_{i=1}^{n} \gamma(\alpha m_i)} &(\prod_{i=1}^{n-1} p_i^{\alpha m_i - 1})(1 - \sum_{i=1}^{n-1} p_i)^{\alpha m_n - 1}, \quad (2.25) \end{aligned}$$

which is the same as the Dirichlet density.

2.3.4 The Dirichlet Process

The Dirichlet dsitribution is a probability distribution over pmfs. The Dirichlet distribution is limited in that it assumes a finite set of events. The Dirichlet Process (DP) (Teh (2010)) is an extension of the Dirichlet distribution to continuous spaces and enables us to work with an infinite set of events. The set of all probability distributions over an infinite sample space is unmanageable. To deal with this, the Dirichlet process defines the class of distributions under consideration through the corresponding set of discrete probability distributions over all finite partitions of the infinite sample space that can be written as an infinite sum of weighted indicator functions.

We can think of the infinite sample space as a dartboard (Frigyik et al. (2010)), and a realisation from a Dirichlet is a probability distribution on the dartboard marked by an infinite set of darts of different lengths (weights). The n^{th} indicator δ_{y_n} marks the location of the n^{th} dart-of-probability such that $\delta_{y_n}(\mathscr{X}) = 1$ if dart $y_n \in \mathscr{X}$, and $\delta_{y_n}(\mathscr{X}) = 0$ otherwise. Each realisation of a Dirichlet process has a different and infinite set of these dart locations. Further, the n^{th} dart has a corresponding probability weight $p_n \in [0, 1]$ and $\sum_{n=1} p_n = 1$. So, for some set $\mathscr{X}$ of the infinite sample space, a realisation of the Dirichlet process will assign probability $P(\mathscr{X})$ to $\mathscr{X}$, where $P(\mathscr{X}) = \sum_{n=1} p_n \delta_{y_n}(\mathscr{X})$. The locations of the darts are independent, and the probability weight associated with the n^{th} dart is independent of its location. However, the weights on the darts are not independent. Instead of a vector α with one component per event as we have in the Dirichlet distribu-

tion, the DP is parameterised by a function α over the sample space of all possible events $\mathscr{X}$. α is a finite positive function, so it can be normalised to be a probability distribution θ so that the locations of the darts y_n are drawn iid from θ.

The Dirichlet Process - Formal Definition

For a random distribution G to be distributed according to a DP, its marginal distributions have to be Dirichlet distributed. Let H be a distribution over Θ and α be a positive real number, then G is DP distributed on the space $\mathscr{X}$ with base distribution H and concentration parameter α, written $G \sim DP(\alpha, H)$ if

$$(G(\mathscr{X}_1), \ldots, G(\mathscr{X}_n)) \sim Dir(\alpha, (H(\mathscr{X}_1), \ldots, H(\mathscr{X}_n))) \tag{2.26}$$

for every measurable partition $\mathscr{X}_1, \ldots, \mathscr{X}_n$ of Θ. The base distribution H is the mean of the DP, i.e. for any measurable set $\mathscr{X}_0 \subset \Theta$, we have

$$\mathbb{E}[G(\mathscr{X}_0)] = H(\mathscr{X}_0)$$

whenever $\mathscr{X}_0$ forms a part of the partition. The concentration parameter α can be understood as an inverse variance:

$$V[G(\mathscr{X}_0)] = H(\mathscr{X}_0)(1 - H(\mathscr{X}_0))/(\alpha + 1)$$

The larger α is, the smaller the variance, and the DP will concentrate more of its mass around the mean.

Posterior Distribution

Let $G \sim DP(\alpha, H)$. Since G is a distribution, we can draw samples from G itself. Let $\theta_1, \ldots, \theta_n$ be a sequence of independent draws from G. Note that θ_i takes values in Θ since G is a distribution over Θ. We are interested in the posterior distribution of G given observed values of $\theta_1, \ldots, \theta_n$. Let $\mathscr{X}_1, \ldots, \mathscr{X}_r$ be a finite measurable partition of Θ, and let $n_j = \sharp\{i \mid \theta_i \in \mathscr{X}_j\}$ be the number of observed values in $\mathscr{X}_j$. By the conjugacy between the Dirichlet and the multinomial distributions, we have:

$$(G(\mathscr{X}_1), \ldots, G(\mathscr{X}_r)) \mid \theta_1, \ldots, \theta_n \sim Dir(\alpha H(\mathscr{X}_1) + n_1, \ldots, \alpha H(\mathscr{X}_r) + n_r)$$

The posterior updates for the DP parameters are therefore:

$$\alpha^\star = \alpha + n \tag{2.27}$$

$$H^\star = \frac{\alpha H + \sum_{i=1}^{n} \delta_{\theta_i}}{\alpha + n}, \tag{2.28}$$

where δ_i is a point mass located at θ_i and $n_j = \sum_{i=1}^{n} \delta_i(\mathscr{X}_j)$. Thus, the DP provides a conjugate family of priors over distributions that is closed under

posterior updates given observations. Rewriting the posterior DP, we obtain:

$$G \mid \theta_1, \ldots, \theta_n \sim DP(\alpha + n, \frac{\alpha}{\alpha + n} H + \frac{n}{\alpha + n} \frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n}) \tag{2.29}$$

It can be seen that the posterior base distribution H is a weighted average between the prior base distribution H and the empirical distribution $\frac{\sum_{i=1}^{n} \delta_{\theta_i}}{n}$. The weight associated with the prior base distribution is proportional to α, while the empirical distribution has weight proportional to the number of observations n.

The concepts briefly described in this section are of particular relevance to the algorithms introduced in Chapter 6.

2.4 The Gaussian Process

Given noisy observations of n datapoints $X_n = \{x_1, x_2, \ldots, x_n\}$ and the outputs $\{y_1, y_2, \ldots, y_n\}$, we may want to make predictions for unseen datapoints $x_\star$. In order to do this, we need to move from the finite training data to a function f that makes predictions for all possible input values. We need to make assumptions about the characertistics of the underlying function. A possible solution is to give a prior probability to every possible functions, with functions considered to be more likely given higher probabilities. There is, however, a problem with this approach as we need to consider an uncountably infinite set of possible functions. This where we can use the Gaussian

Process (GP). Gaussian Processes (Doob (1994); Rasmussen and Williams (2006)) are a generalisation of finite-dimensional Gaussian distribution over vectors (functions), which are defined by a mean vector and covariance matrix. A GP is defined by a mean function and a covariance function, which indicates how correlated the values of the function f are at locations x_1 and x_2. The covariance function encodes our assumptions about the problem, e.g. that the function is smooth and continuous, and will influence the quality of the predictions.

To illustrate the approach, let us consider a 1-d regression problem mapping from an input x to an output $f(x)$ and we draw a number of samples drawn from the prior distribution over functions specified by a particular Gaussian process. Suppose that we now have two observations $\{(x_1, y_1), (x_2, y_2)\}$ and we are interested in functions that pass through these data points. As we draw more samples, the mean function adjust itself to pass through the data points and the posterior is reduced close to the observations. An important aspect of GP inference the specification the prior as this will affect the properties of the functions that we consider. We can obtain different priors through manipulating the mean and variance of the functions, or the covariance function of the GP.

2.4.1 Formal Definition

A Gaussian Process is a collection of random variables, any finite number of which have a joint Gaussian distribution (Rasmussen and Williams (2006)).

A GP is completely specified by its mean function

$$\mu(x) = \mathbb{E}[f(x)] \tag{2.30}$$

and covariance function

$$C_{x_1,x_2} = \mathbb{E}[(f(x_1) - \mu(x_1))(f(x_2) - \mu(x_2))]. \tag{2.31}$$

We write the GP as

$$f(x) \sim GP(\mu(x), C_{x_1,x_2}). \tag{2.32}$$

The random variables in GP represent the value of the function f:

$$f = \{f(x) : x \in X\}$$

whose marginal distribution for any finite collection of points

$$X_n = \{x_1, x_2, \ldots, x_n\} \subseteq X$$

has Gaussian joint distribution

$$P(f \mid C, \mu, \{X_n\}) = \frac{1}{Z} \exp\left(-\frac{1}{2}(f(X_n) - \mu(X_n))^T C^{-1}_{X_n X_n}(f(X_n) - \mu(X_n))\right)$$

The only constraint on the covariance function C_{x_1,x_2} is that it should generate a positive definite covariance matrix for any set of points X. Dif-

ferent choices of C_{x_1,x_2} can give rise to different priors ranging from straight lines, as in $y = w_0 + w_1 x$, to very rough sample paths associated with Brownian motion. A very common covariance function is the Gaussian covariance function:

$$C_{x_1,x_2} = v_0 \exp\left\{ -\sum_{l=1}^{d} \frac{(x_{1,l} - x_{2,l})^2}{\lambda_l^2} \right\} \tag{2.33}$$

The function is the product of d squared covariance functions with different length scales λ_l on each dimension. The general form of the covariance function expresses the idea that nearby inputs will have highly correlated outputs, and the λ parameters allow a different distance measure for each input dimension. For irrelevant inputs, the corresponding λ_l will become large and the model will effectively ignore that input.

It is typical for more realistic modelling situations that we do not have access to function values themselves, but only noisy versions, therefore the reward is given by $f(x) + \epsilon$, where

$$\epsilon \sim \mathcal{N}(0, \sigma^2_{noise})$$

is additive independent identically distributed Gaussian noise with mean 0 and variance σ^2_{noise}. With the added noise, the prior on the noisy observation becomes $C_{x_1,x_2} + \sigma^2_{noise}\delta_{1,2}$ where $\delta_{1,2}$ is a Kronecker delta which is 1 iff $x_1 = x_2$ and 0 otherwise.

2.4.2 The Gaussian Process Posterior

Let us denote by $\mathcal{D} = \{f(x_1), f(x_2), \ldots, f(x_n)\}$ a data vector generated by a GP with a covariance matrix C_n and a mean vector $\mu = 0$. By explicitly stating the probability of the data we have bypassed the step of assigning individual priors on the noise and the modelling function. It is worth mentioning that in GP, we are not constraining the function used to model the data to be a mixture of Gaussians, but rather to express the correlations between outputs at different points in the input space.

Given the data $\mathcal{D} = \{f(x_1), f(x_2), \ldots, f(x_n)\}$, we can calculate the Gaussian conditional distribution over $f(x_{n+1})$:

$$P(f(X_{n+1}) \mid \mathcal{D}, C_n, X_{n+1}) = \frac{P(f(X_{n+1}) \mid C_n, x_{n+1}, \{X_n\})}{P(f(X_n) \mid C_n, \{X_n\})} \tag{2.34}$$

$$= \frac{Z_n}{Z_{n+1}} \exp\left[-\frac{1}{2}(f(X_{n+1})C_{n+1}^{-1}f(X_{n+1}) - f(X_n)C_n^{-1}f(X_n))\right] \tag{2.35}$$

where Z_n and Z_{n+1} are the normalising constants. The Gaussian conditional distribution of $f(x_{n+1})$ is:

$$P(f(x_{n+1}) \mid \mathcal{D}, C_n, x_{n+1}) = \frac{1}{Z} \exp\left(-\frac{(f(x_{n+1}) - \mu(x_{n+1}))^2}{2\sigma^2_{\mu(x_{n+1})}}\right) \tag{2.36}$$

where

$$\mu(x_{n+1}) = k_{N+1}C_n^{-1}f(x_n) \tag{2.37}$$

and

$$\sigma^2_{\mu(x_{n+1})} = \kappa - k_{n+1} C_n^{-1} k_{n+1} \tag{2.38}$$

with $\kappa = C(x_{n+1}, x_{n+1})$ and $k_{n+1} = (C(x_1, x_{n+1}), \ldots, C(x_n, x_{n+1}))$. $\mu(x_{n+1})$ is the mean prediction at the new point and $\sigma_{\mu(x_{n+1})}$ is the standard deviation of this prediction.

The concepts briefly described in this section are of particular relevance to the algorithms introduced in Chapters 4 and 5.

Chapter 3

Bandit Algorithms

3.1 Introduction

An important aspect of the nature of learning is interaction with the environment. Reinforcement learning (Sutton and Barto (1998); Kaelbling et al. (1996)) is a computational approach to learning from iteraction. The main goal of reinforcement learning is learning which actions to take in order to maximise a reward function. The learner is not explicitly told which actions to take in order to obtain the best reward, but instead must discover, through trial and error, which actions will yield high rewards.

The trial and error approach to learning about one's environment leads to one of the main challenges in reinforcement learning – the trade-off between exploration and exploitation. In order to obtain high rewards, the learner must find out which actions can produce high rewards. However, in order to

discover such actions, the learner has to try actions not tested before. Thus, the learner has to exploit what he has already learnt from previous actions, but he also has to explore new actions in order to make better selections in the future. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward.

3.2 Bandits

One of the simplest reinforcement learning problems is known as the multi-armed bandit problem, which has been the subject of a great deal of study in the statistics and applied mathematics literature (Bellman (1956); Berry and Fristedt (1985); Narendra and Thathachar (1989); Robbins (1952); Thompson (1934); Auer et al. (2002a); Gittins (1979); Gittins and Jones (1979)). In this chapter, we propose a new extension to simple bandit algorithms, where, we use a Beta prior to evaluate the upper confidence bound of bandits with independent arms.

The multi-armed bandit problem is an analogy with a traditional slot machine, known as a one-armed bandit, but with multiple arms. In the bandit scenario, the player, after pulling (or "playing") an arm selected from a finite set of arms, receives a reward. It is assumed that the reward obtained when playing arm i is a sample from a distribution R_i with mean μ_i, both of which are unknown to the player. A bandit problem is characterised by the set of probability distributions $R_{i,1\leq i\leq N}$. The objective of the player is

to maximise the cumulative reward through iterative plays of the bandit. The optimal arm selection policy $S^\star$, i.e. the policy that yields maximum expected cumulative reward, consists in selecting arm $i^\star = \text{argmax}_i\{\mu_i\}$ to play at each iteration. The expected cumulative reward of $S^\star$ at time t (after t plays) is $t\mu_{i^\star}$. The performance of a policy S is assessed by the analysis of its expected regret ρ at time t, defined as the difference between the expected cumulative reward of $S^\star$ and S at time t:

$$\rho = t\mu_{i^\star} - \sum_{i=1}^{t} \mu_i,$$

where μ_i is the reward at time t. A good policy balances the learning of the distributions R_i and the exploitation of arms which have been learnt as having high expected rewards. As the number of arms is finite (and usually smaller than the number of experiments allowed), it is possible to explore all the possible options (arms), thus building empirical estimates of μ_i's, and also exploiting the best performing ones. As the number of times we play the same arms with the highest rewards grows, we expect our reward estimates to improve.

3.3 Independent Arm Bandits

Lai and Robbins (1985) were the first to show that the regret for the multi-armed bandit problem has to grow at least logarithmically in the number of

plays. The upper confidence bound of an algorithm is obtained by maximizing the expected payoff when the parameters are varied within an appropriate confidence set. Since then, policies which asymptotically achieve this regret have been devised by Lai and Robbins (1985) and many others (e.g. Agrawal (1995); Burnetas and Katehakis (1996)). Let $T(j)$ be the number of times arm j has been played and t be the number of times of all plays so far. In their classic paper, Lai and Robbins (1985) found, for specific families of reward distributions (including: normal, Bernoulli, Poisson), policies satisfying

$$\mathbb{E}[T(j)] \leq \left(\frac{1}{D(p_j \parallel p^\star)} + o(1) \right) \log t \tag{3.1}$$

where $o(1) \to 0$ as $t \to \infty$ and $D(p_j \parallel p^\star)$ is the Kullback-Leibler divergence between the reward density p_j of arm j and the reward density $p^\star$ of the arm with the highest reward expectation $\mu^\star$. Under these policies, the optimal arm is played exponentially more often than any other arm, at least asymptotically.

Auer et al. (2002a) considered the case when the rewards come from a bounded support, say [0, b], but otherwise the reward distributions are unconstrained. They have studied several policies, most notably UCB1 which maintains the number of times that each arm has been played. UCB1 constructs the Upper Confidence Bounds (UCB) for arm j at time t by adding

the bias factor σ, defined as follows,

$$\sigma_j(t) = \sqrt{\frac{2b^2 \log t}{T(j)}}, \tag{3.2}$$

to its sample mean.

Let us denote by $\mu_j(t)$ the empirical mean reward for arm j. In order to optimise the reward at each time step in the face of uncertainty, confidence intervals $[\mu_j(t) - \sigma_j(t); \mu_j(t) + \sigma_j(t)]$ are determined for the reward value we would get for each arm j at time t, and the algorithm plays at time t the arm which maximises the value of the upper confidence bound $\mu_j(t) + \sigma_j(t)$. The bigger $\sigma_j(t)$, the more likely it is that the reward will be in the confidence interval. Auer et al. (2002a) have proven that the expected regret of this algorithm is at most

$$8 \left(\sum_{i:\mu_i<\mu^\star} \frac{b^2}{\Delta_i} \right) \log(t) + O(1) \tag{3.3}$$

where

$$\Delta_i = \mu^\star - \mu_i \tag{3.4}$$

The UCB1 arm selection strategy is summarised in Algorithm 1.

In the same paper, Auer et al. (2002a) also propose UCB1-NORMAL, which is designed to work with normally distributed rewards only. This algorithm estimates the variance of the arms and uses these estimates to refine the bias factor. They show that for this algorithm when the rewards

Algorithm 1 UCB1

play each arm $(1, \ldots, n)$ once,
set $t = n + 1$
for $t = 1, \ldots, end$ **do**
  play $j = \text{argmax}\{\mu_j(t) + \sigma_j(t)\}$
  get reward r_j
  $T(j) = T(j) + 1$,
  $t = t + 1$.
end for

are indeed normally distributed with means μ_i and variances σ_i^2, the expected regret is at most

$$8 \left(\sum_{i:\mu_i<\mu^\star} \frac{32\sigma_i^2}{\Delta_i} + \Delta_i \right) \log(t) + O(1) \tag{3.5}$$

One of the major differences between the two algorithms is that the regret-bound for UCB1 scales with b^2, while the regret bound for UCB1- NORMAL scales with the variances of the arms. Since b is typically just an a priori guess on the size of the interval containing the rewards, which might be overly conservative, it might be desirable to lessen the dependence on it.

Auer et al. (2002a) introduced another algorithm, UCB1-TUNED, in the experimental section of their paper. This algorithm, similarly to UCB1-NORMAL, uses the empirical estimates of the variance in the bias sequence. For practical purposes, the regret bound of UCB can be tuned more finely

by replacing the bias factor $\sqrt{\frac{2\log(t)}{T(j)}}$ of UCB1 with

$$\sqrt{\frac{\log t}{T(j)}\min\{1/4, V(T(j))\}}, \tag{3.6}$$

where $V(T(j))$ is the variance of rewards obtained by playing arm j. Although no theoretical guarantees were derived for UCB1-Tuned, this algorithm has been shown to outperform UCB1 and UCB1-NORMAL. Intuitively, algorithms using variance estimates should work better than UCB1 when the variance of some suboptimal arms is much smaller than b^2, since these arms will be less often drawn: suboptimal arms are more easily spotted by algorithms using variance estimates.

Audibert et al. (2009) consider a variant of the basic UCB algorithm that takes into account the empirical variance of different arms. UCB-V has a major advantage over its alternatives that do not use such estimates provided that the variances of the payoffs of the suboptimal arms are low. According to the UCB-V policy, at each time t, we play arm x_j maximising:

$$B_{j,T(j),t} = \mu_j(t) + \sqrt{\left(\frac{2V(T(j))\varepsilon_t}{T(j)}\right)} + \frac{3b\varepsilon_t}{T(j)}, \tag{3.7}$$

where $V(T(j))$ is the empirical variance associated with the first $T(j)$ draws of arm j, and ε is the so called exploration function. A typical choice for this function is $\varepsilon_t = \log(t)$.

Let us summarise the main ideas behind UCB-V (Audibert et al. (2007)). As long as an arm is never selected its bound is infinite. Hence, initially the algorithm tries all the arms at least once. In future iterations, the more an arm j has been tested, the closer $B_{j,T(j),t}$ gets to the sample-mean and hence to the expected reward μ_j. So the procedure will hopefully tend to draw more often arms having the largest expected rewards. However, since the obtained rewards are stochastic it might happen that during the first draws the (unknown) optimal arm always gives low rewards. This might make the sample-mean of this arm smaller than that of the other arms and hence an algorithm that only uses sample-means might get stuck with not choosing the optimal arm any more. The UCB-V policy uses the exploration function ε to prevent this situation. Assuming that ε increases without bounds in t, after a while the last term of $B_{j,T(j),t}$ will start to dominate the two other terms and will also dominate the bound associated with the arms drawn very often. This will allow the algorithm to draw the optimal arm again, giving it a chance to develop a better estimate of the mean. An appropriate choice of ε encourages exploration and so it must be carefully chosen so as to balance exploration and exploitation. The major idea of upper-confidence bounds algorithms is that ε should be selected such that $B_{j,T(j),t}$ is a high probability upper bound on the payoff of arm j. The novelty UCB-V is that $B_{j,T(j),t}$ involves the empirical variance.

3.4 Beta Bandits

We propose a revised approach, based on Beta prior, for tuning the regret bound (Głowacka and Shawe-Taylor (2012)). We show that our approach greatly outperforms the UCB-V (Audibert et al. (2009)) and Thompson sampling (Chapelle and Li (2011)) – two bandit algorithms that have already been shown to outperform standard bandit algorithms such as UCB and UCB-TUNED.

Chapelle and Li (2011) propose a setting where so-called Thompson sampling (Thompson (1934)) is applied to the Bernoulli bandit. Thompson sampling chooses the arm that is best in a random sample of arm probabilities from the posterior distribution. It therefore selects arm j with probability equal to it being the best arm in the posterior distribution. Algorithm 2 gives this algorithm for the case of modelling the arm probabilities with independent Beta distributions. The reward of the arm x_j follows a Bernoulli distribution with mean θ_j The mean reward of each arm is modelled using the Beta distribution:

We propose a new strategy also based on Beta prior, which we term UCB-BETA. We modify the upper confidence bound used UCB-TUNED and replace it with:

$$\sigma_j(t) = \sqrt{\log t}\sqrt{V(j)} \tag{3.8}$$

where the term $\sqrt{\log t}$ is analogous to the first term found in UCB-TUNED in Equation 3.3 and $V(j)$ is Beta variance of arm j:

Algorithm 2 Thompson sampling for the Bernoulli bandit

Require: α and β parameters for a Beta distribution
$S_i = 0, F_i = 0, \forall i$, {Success and failure counters}
for $t = 1, \ldots, end$ **do**
  for $j = 1, \ldots, n$ **do**
    Draw θ_j according to Beta$(S_j + \alpha, F_j + \beta)$
  end for
  Play arm $j^\star = \text{argmax}_j \theta_j$ and observe reward r
  if $r == 1$ **then**
    $S_{j^\star} = S_{j^\star} + 1$
  else
    $F_{j^\star} = F_{j^\star} + 1$
  end if
end for

$$V(j) = \frac{\alpha_j \beta_j}{(\alpha_j + \beta_j)^2(\alpha_j + \beta_j + 1)} \tag{3.9}$$

while μ_j is the expectation of the Beta distribution:

$$\mu_j = \frac{\alpha_j}{\alpha_j + \beta_j} \tag{3.10}$$

3.4.1 Experimental Results

We compared the performance of UCB-BETA with UCB-V and Thompson sampling. The first problem consisted of selecting one of 10 different arms with the probability of obtaining a reward ranging from 0.0002 to 0.01. The probabilities were as follows: 0.001, 0.003, 0.01, 0.009, 0.0011, 0.0025, 0.0006, 0.007, 0.0002, 0.005. The experiments consisted of 1000 runs, with 10,000 iterations each, of the three policies. We also tested the influence of the prior

on the performance of UCB-BETA and Thompson sampling in two separate experiments. We set the initial value of α in the Beta distribution to 1, 0.1 and 0.01. Table 3.1 shows the the mean and variance of the 1000 runs as well as the maximum and minimum regret with the three Beta priors.

	MEAN	STD	MAX	MIN
UCB-BETA, $\alpha = 1$	73.18	26.47	169.1	30.77
UCB-BETA, $\alpha = 0.1$	71.73	66.49	307.92	6.06
UCB-BETA, $\alpha = 0.01$	140.98	160.29	689.23	1.76
Thompson, $\alpha = 1$	100.79	25.74	190.79	62.82
Thompson, $\alpha = 0.1$	75.85	33.52	282.87	20.74
Thompson, $\alpha = 0.01$	79.15	53.5	354.4	10.93
UCB-V	177.43	18.75	229.52	128.17

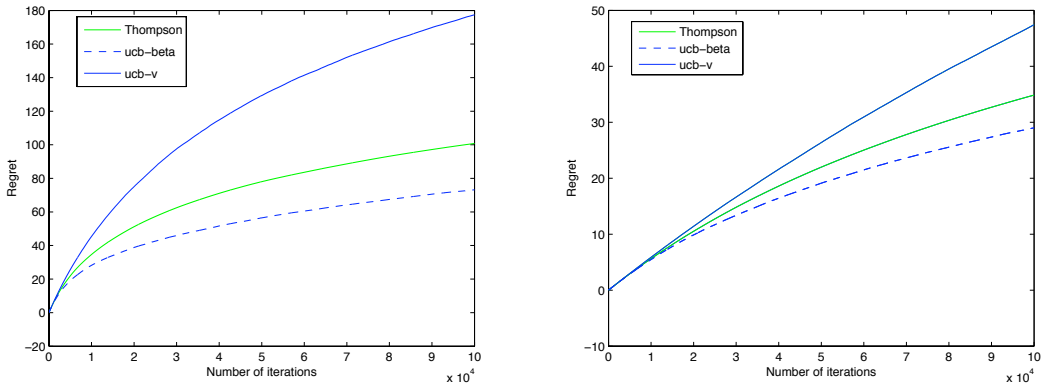
Table 3.1: Regret over 1000 runs of UCB-Beta, Thompson sampling and UCB-V with maximum probability of obtaining reward = 0.01.

In the next set of experiments, we decreased the reward probability by a factor of 10. The probabilities of the 10 arms of obtaining the reward were as follows: 0.0001, 0.0003, 0.001, 0.0009, 0.00011, 0.00025, 0.00006, 0.0007, 0.00002, 0.0005. The results of the experiments are reported in Table 3.2.

Figure 3.1 shows average regret over 100,000 iterations of the three policies with $\alpha = 1$ for UCB-BETA and Thompson sampling. As the results show, UCB-BETA outperforms UCB-V in both sets of experiments, which indicates that using Beta variance allows better tuning of the regret bound than using the UCB-V variance estimates. In general, UCB-BETA performs better than Thompson sampling in both sets of experiments. However, as the first set of experiments presented in Table 3.1 suggests, setting

	MEAN	STD	MAX	MIN
UCB-BETA, $\alpha = 1$	29.01	5.45	50.36	17.35
UCB-BETA, $\alpha = 0.1$	17.29	10.35	65.06	4.79
UCB-BETA, $\alpha = 0.01$	21.52	19.04	74	1.41
Thompson, $\alpha = 1$	34.86	5.2	57.02	24.46
Thompson, $\alpha = 0.1$	24.17	7.25	57.9	11.28
Thompson, $\alpha = 0.01$	23.44	15.38	85.88	4.07
UCB-V	47.43	2.61	53.98	40.98

Table 3.2: Regret over 1000 runs of UCB-Beta, Thompson sampling and UCB-V with maximum probability of obtaining reward = 0.001.



(a) (b)

Figure 3.1: Average regret of UCB-BETA, UCB-V and Thompson sampling with $\alpha = 1$ when the maximum probability of award is (a) 0.01 and (b) 0.001

the wrong prior has a detrimental effect on the performance of UCB-BETA. When $\alpha = 0.01$ and the maximum probability of obtaining reward is 0.01, UCB-BETA performs significantly worse than Thompson sampling. This result might point to the fact that when given the wrong prior, UCB-BETA cannot balance exploration-exploitation very well. The performance of the algorithm is highly variable depending on the initial settings of α. The α prior affects the empirical variance of each arm, which, in turn, will affect the algorithm's estimate of the size of the interval containing the highest rewards. When our prior "guess" of α is too large or too small, the algorithm frequently fails to detect which arms are suboptimal.

3.5 Conclusions

In this chapter, we introduced the multi-armed bandit problem and a new bandit policy based on Beta priors. Preliminary experimental results indicate that the proposed policy significantly outperforms existing bandit algorithms, such as UCB-V and Thompson sampling. Further theoretical and experimental analysis are required to fully understand the strengths and weaknesses of the proposed approach.

Chapter 4

Dependent Arm Bandits

As mentioned in the previous chapter, in order to obtain a good estimate of the reward of each arm, we need to try each arm, which renders the exploration phase very long if the number of arms is large. Modelling dependencies across arms – if such dependencies exist – makes the exploration phases shorter, since, when playing arm i, we gain knowledge not only about this particular arm but also about similar arms. Thus, at each play, we can update the μ_i estimates of several arms. There are relatively few studies of multi-armed bandit algorithms with dependent arms and most of the literature on the subject is focused on practical applications of dependent arm bandits. For example, the EXP4 algorithm (Auer et al. (2002b)) uses the exponential weighting technique to achieve an $O(\sqrt{T})$ regret but the computational complexity may be exponential in the number of features. Another general contextual bandit algorithm is the epoch-greedy algorithm (Lang-

ford and Zhang (2008)) that is similar to ϵ-greedy with shrinking ϵ. This algorithm is computationally efficient given an oracle optimiser but has the weaker regret guarantee of $O(T^{2/3})$. Pandey et al. (2007) have developed an algorithm that exploits cluster structures among arms and have shown that the accumulated reward obtained with their algorithm increases faster than that of UCB in an application involving web advertisements. Bubeck et al. (2008) and Kleinberg et al. (2008) focus on applications with infinitely many arms which are indexed in a metric space. Similarly, Lu et al. (2010) also consider metric spaces. However, the important difference between Kleinberg et al. (2008) and Lu et al. (2010) is the non-uniformity over the payoff function μ. Namely, Lu et al. (2010)'s bounds do not depend on μ whereas Kleinberg et al. (2008)'s do.

Auer (2002) uses confidence bounds to deal with an exploration-exploitation trade-off in a rather complicated model that is reminiscent of a dependent arms situation. The paper defines a model for associative reinforcement learning with linear value functions. In this model, a learning algorithm has to choose an alternative $j \in \{1, \ldots, K\}$, where K is the number of alternatives. In each trial $t = 1, \ldots, T$, the algorithm observes the reward $r_j(t)$ of the chosen alternative j, and tries to maximize its cumulative reward $\sum_{t=1}^{T} r_j(t)$. The main difference in comparison with the independent bandit problem is that the algorithm is provided with additional information. For each alternative j a feature vector $z_j \in \mathbb{R}^d$ is given to the learning algorithm and the algorithm chooses an alternative based on these feature vectors. This

feature vector determines the expected reward for alternative j in trial t. It is assumed that there is an unknown weight vector $w \in \mathbb{R}^d$ which is fixed for all trials and alternatives, such that $\langle w, z_j \rangle$ gives the expected reward $\mathbb{E}[r_j]$ for all $j \in \{1, \ldots, K\}$ and all $t = 1, \ldots, T$. LinRel calculates upper confidence bounds for the means $\mathbb{E}[r_j] = \langle w, z_j \rangle$ and chooses the alternative with the largest upper confidence bound, again trading-off exploitation controlled by the estimation of the mean, and exploration controlled by the width of the confidence interval. The main idea of the algorithm is to estimate the mean $\mathbb{E}[r_j]$ from a weighted sum of previous rewards. The LinRel algorithm is summarised in Algorithm 3.

Algorithm 3 LinRel

parameters: $\delta \in [0, 1]$, number of trials T
inputs: indexes of selected feature vectors $\Psi(t) \subseteq \{1, \ldots, t-1\}$; new feature vectors $z_1(t), \ldots, z_K(t) \in \mathbb{R}^d$; matrix $Z(t)$ of selected feature vectors; vector $r(t)$ of corresponding rewards
for $t = 1, \ldots, T$ **do**
  calculate the eigenvalue decomposition
  $Z(t) \cdot Z(t)' = U(t)' \cdot \Delta(\lambda_1(t), \ldots, \lambda_d(t)) \cdot U(t)$
  for $i = 1, \ldots, K$ **do**
    $\tilde{z}_i(t) = (\tilde{z}_{i,1}(t), \ldots, \tilde{z}_{i,d}(t))' = U(t) \cdot z_i(t)$
    $\tilde{u}_i(t) = (\tilde{z}_{i,1}(t), \ldots, \tilde{z}_{i,k}(t), 0, \ldots)'$
    $\tilde{v}_i(t) = (0, \ldots, 0, \tilde{z}_{i,k+1}(t), \ldots, \tilde{z}_{i,d}(t))'$
    $a_i(t) = \tilde{u}_i(t)' \cdot \Delta(\frac{1}{\lambda_1(t)}, \ldots, \frac{1}{\lambda_k(t)}, 0, \ldots, 0) \cdot U(t) \cdot Z(t)$
    $\text{width}_i(t) = \| a_i(t) \| (\sqrt{\ln(2TK/\delta)}) + \| \tilde{v}_i(t) \|$
    $\text{ucb}_i(t) = r(t) \cdot a_i(t)' + \text{width}_i(t)$
  end for
  play $z_i(t) = \text{argmax}\{\text{ucb}_i(t)\}$
end for

Auer (2002) was not able to bound the performance of LinRel. Instead, a master algorithm SupLinRel is used and calls LinRel as a subroutine. A $O(\sqrt{t}\log(t))$ regret bound was derived for SupLinRel, but none for LinRel. However, for most practical applications, LinRel is believed to be able to achieve the same or even better performance.

Algorithm 4 LinUCB

inputs: $\alpha \in \mathbb{R}_+$, number of trials T; new feature vectors $z_1, \ldots, z_K \in \mathbb{R}^d$
$A = I_d$
$b = 0_d$
for $t = 1, \ldots, T$ **do**
  $\theta_t = A^{-1}b$
  for $i = 1, \ldots, K$ **do**
    $p_{t,i} = \theta_t^T z_{t,i} + \alpha\sqrt{z_{t,i}^T A^{-1} z_{t,i}}$
  end for
  choose action $i_t = \text{argmax}_i\{p_{t,i}\}$
  observe payoff $r_t \in \{0, 1\}$
  $A = A + z_{t,i}z_{t,i}^T$
  $b = b + z_{t,i}r_t$
end for

The LinUCB algorithm (Li et al. (2010), Chu et al. (2011)) is motivated by the UCB algorithm (Auer et al. (2002a)) and the KWIK algorithm Walsh et al. (2009). LinUCB is also similar to the LinRel algorithm – the main idea of both algorithms is to compute the expected reward of each arm by finding a linear combination of the previous rewards of the arm. To do this, LinUCB decomposes the feature vector of the current round into a linear combination of feature vectors seen on previous rounds and uses the computed coefficients and rewards on previous rounds to compute the expected

reward on the current round. LinRel, however, is a more complicated algorithm as it requires solving an SVD (or eigen decomposition) of a symmetric matrix, while LinUCB only requires inverting the same matrix. The LinUCB algorithm is summarised in Algorithm 4.

4.1 The Gaussian Process Bandits Algorithm

We propose a general framework for handling dependencies across arms, based on a new assumption that the mean-reward function is drawn from a Gaussian Process (GP), with a given arm covariance matrix (Dorard et al. (2009))[1]. We assume in our model that the reward of an arm j is determined by a function f applied at point j to which Gaussian noise is added. The variance of the noise corresponds to the variability of the reward when always playing the same arm. Our assumption is that the rewards of arms are correlated, i.e. the more similar two arms are, the more similar the rewards obtained by playing them will be. Thus, playing an arm "close" to j gives information on the expected gain of playing j. These correlations can be modelled by assuming that f is a function drawn from a GP. By default, we take the mean of the GP prior to be 0, and we incorporate prior knowledge on how correlated arms are in the GP covariance matrix. Each entry (i, j) of this matrix specifies how "close" or "similar" arms i and j are. We consider arms indexed by any type of structured data as long as a kernel can be de-

[1]The experimental work reported in this chapter is joint work with Louis Dorard.

fined between the data points. Hence, whereas Pandey et al. (2007)'s model of dependencies is based on a clustering of arms, our model is based on a covariance/kernel matrix between the arms. Using GPs permits an approach similar to arm selection in UCB, where we look for the arm which maximises an upper confidence function. This function is a sum of a mean-reward estimate $\mu_j(t)$ and an uncertainty term $\sigma_j(t)$. In our case, the estimated reward is given by the mean of the GP posterior, and the uncertainty term given by the posterior standard deviation.

We consider a space $\mathscr{X}$ whose elements will be referred to as arms. Let κ denote a kernel defined on pairs of elements of $\mathscr{X}$. The reward after playing arm $x \in \mathscr{X}$ is given by $f(x) + \epsilon$ as defined in Section 2.4.1. Arms played up to time t are $x_1, \ldots, x_t$ with rewards $y_1, \ldots, y_t$. The GP posterior at time t after seeing data $(x_1, y_1), \ldots, (x_t, y_t)$ has mean $\mu_t(x)$ with variance $\sigma_t^2(x)$ as defined as follows (see Section 2.4.1 for more details):

$$\mu_t(x) = k_t(x)^T C_t^{-1} y_t \tag{4.1}$$

$$\sigma_t^2(x) = \kappa(x, x) - k_t(x)^T C_t^{-1} k_t(x) \tag{4.2}$$

4.1.1 Arm Selection

The algorithm plays a sequence of arms and aims to optimally balance exploration and exploitation. We select arms iteratively according to a UCB-

inspired formula:

$$x_{t+1} = \text{argmax}_{x \in \mathscr{X}} \{f_t(x) = \mu_t(x) + B(t)\sigma_t(x)\} \tag{4.3}$$

This can be seen as active learning where we want to learn accurately in regions where the function looks good, while ignoring other areas. The $B(t)$ term balances exploration and exploitation: the bigger it gets, the more it favours points with high $\sigma_t(x)$. In the original UCB formula, $B(t) \sim \sqrt{\log t}$.

The objective of the bandit algorithm is to minimise the regret over time, i.e. to find as quickly as possible a good approximation $f(x^\star)$ of the maximum of f. Let us define $f_t(x)$ as:

$$f_t(x) = \mu_t(x) + B(t)\sigma_t(x) \tag{4.4}$$

$$= k_t(x)^T C_t^{-1} y_t + B(t)\sqrt{\kappa(x,x) - k_t(x)^T C_t^{-1} k_t(x)} \tag{4.5}$$

Our approximation of $f(x^\star)$ at time t is $f(x_{t+1})$, where $x_{t+1} = \text{argmax}_{x \in \mathscr{X}} \{f_t(x)\}$. The argmax is found by exhaustive search of $\mathscr{X}$ when the space is finite. Here, we replace the problem of finding the maximum of the function f by a simpler one, i.e. by maximising the function f_t, which is known. At each iteration, we learn new information which enables us to improve our approximation of $f(x^\star)$ over time.

4.1.2 Complexity of Iteration t of GPB

There are n arms for which we need to compute f_t. The cost of computing f_t is equal to the cost of multiplying and adding the terms of Equation 4.3, which is of the order of $O(t^2)$, and the cost of computing the inverse of the covariance matrix C_t, which is in the order of $O(t^3)$. However, If we consider the following representation of C_t:

$$C_t = \begin{pmatrix} C_{t-1} & k_{t-1}(x_t) \\ k_{t-1}(x_t)^T & \kappa(x_t, x_t) + \sigma^2_{noise} \end{pmatrix} \tag{4.6}$$

then we can apply the block inversion lemma (Banachiewicz (1937)), which allows us to invert the matrix iteratively, reducing the cost from $O(t^3)$ to $O(t^2)$. Thus, the total cost of iteration t of GPB is:

$$O(t^2) \cdot n + O(t^2) = O(t^2 n). \tag{4.7}$$

In comparison, the cost of iteration t of UCB is $O(d)$, which is equal to d times the cost of computing the upper confidence function on a given arm. Thus, UCB is more efficient per iteration because its cost is constant over time.

4.2 Optimisation

As mentioned earlier, the cost of iteration t of GPB is $O(\mid T(t) \mid^2 n)$, where $T(t)$ is the set of visited arms at time t containing t elements. We can reduce the computational cost by reducing the size of set $T(t)$. At each iteration we add an arm to T, which increases the cost of future iterations. However, we can remove the "oldest" arm in T if $T(t)$ reaches a certain size $S(t)$. The intuition behind this approach is that as the training progresses, the impact of the "oldest" observed datapoints on $f(x^\star)$ become less and less relevant. A sensible choice for $S(t)$ may vary depending on the particular type of problem considered. In our experiments, we consider $S(t) = \sqrt{n}$ so that the amount of computation that GPB uses is in the order of $O(\sqrt{n^2}) = O(n)$ and is thus of the same order as UCB. The cost of iteration t of GPB becomes $O(n^2)$ plus, possibly, the cost of removing an element from the set $T(t)$. Two operations are required in order to remove an element from the $T(t)$ set:

- remove an element from the list of arms that have been tried so far, which is in the order of $O(1)$;
- update C_t^{-1} by using the block inversion formula, which is of the order of $O(\mid T(t) \mid^2)$

Thus, the cost of iteration t of GPB is:

$$O(n^2) + O(n) + O(1) = O(n^2). \tag{4.8}$$

By limiting the size of the training set to a fixed value, the cost of iteration t of GPB becomes constant over time (although the cost is still bigger than that of UCB). We will refer to the modified version of GPB as *GPB-red*. It is worth mentioning that when removing a data point from the training set, we increase the GP posterior variance at this point and points close to it and therefore we give them a bigger chance of being selected (as increasing the variance increases f_t). Thus, at each iteration of the algorithm, we discard the "oldest" datapoint (arm) in the training set and replace it with the datapoint (arm) just "played" by the algorithm. Effectively, the size of the training set remains constant and consists only of arms tried in the latest n iterations.

4.3 Experimental Results

In order to test the performance of GBP, we devised a toy problem where rewards are given as values of a function f plus Gaussian noise. We selected as f the 2-dimensional Rosenbrock function, defined in $[-1, 1] \times [-1, 1]$ and scaled so that its values are in [0, 1] range and its maximum is 1. Knowing the maximum of f will allow us to determine the regret of arm selection policies. We selected the Rosenbrock function as it is frequently used as a test function in optimisation algorithms as it has a shallow minimum inside a deeply curved valley. f is defined as follows on the unit disk (see Figure 4.1):

$$f(x) = (-100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 161)/161 \tag{4.9}$$

This can be extended to $x \in [-1, 1] \times [-1, 1]$ by setting x:

$$f(x) = f\left(\frac{x}{\parallel x \parallel}\right) \tag{4.10}$$

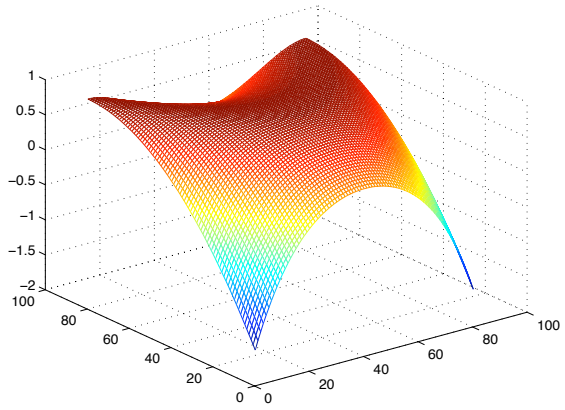


Figure 4.1: The Rosenbrock function as defined in Equation 4.9

In our experiments, arms are elements of a grid of $[-1; 1]^2$ of size n. In the experiments reported below, there are $n = 81$ arms obtained by taking intervals of 0.25 on the $[-1, 1] \times [-1, 1]$ space. We set the value of σ in the Gaussian kernel to 0.5, and the value of the added Gaussian noise standard deviation $\sigma_{noise} = 0.3$. We use a Gaussian kernel in $\mathbb{R}^2$ between the elements of the grid to build the covariance matrix:

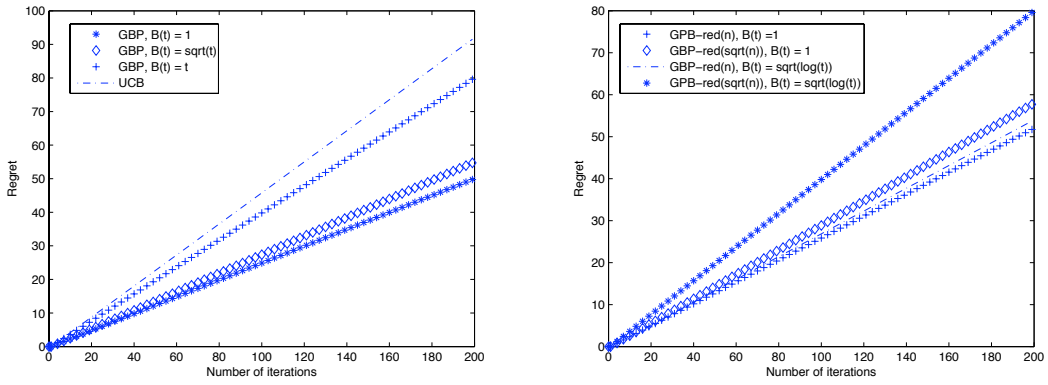
$$\kappa(x_i, x_j) = \exp\left(-\frac{\parallel x_i - x_j \parallel}{2\sigma^2}\right) \tag{4.11}$$

The regrets reported here are averaged over 100 runs of each algorithm.

We experimented with three b functions: $b(t) = t, b(t) = \sqrt{\log t}$ and $b(t) = 1$ and three S functions: $S(t) = t$ (Figure 4.2a), $S(t) = n$ (Figure 4.2b) and $S(t) = \sqrt{n}$ (Figure 4.2b). As mentioned in the previouse chapter, b is a "guess" on the size of the interval containing the rewards. When the interval is large, it may contain a large number of suboptimal arms and so the algorithm will concentrate too much trying out these subptimal arms, which will harm the performance of the algorithm. The b function also allows us to control the exploration/exploitation trade-off, i.e. the large the value of b, the more the algorithm explores. As Figure 4.2a shows, the regret of UCB is similar to that of the random policy. We can clearly see that GPB with $b(t) = t$ performs too much exploration and chooses arms quasi randomly. GPB with a smaller emphasis on exploration, such as $b(t) = \sqrt{\log t}$ and $b(t) = 1$, performs much better than UCB from the very start. Its regret curves are also smoother than the ones produced by UCB.

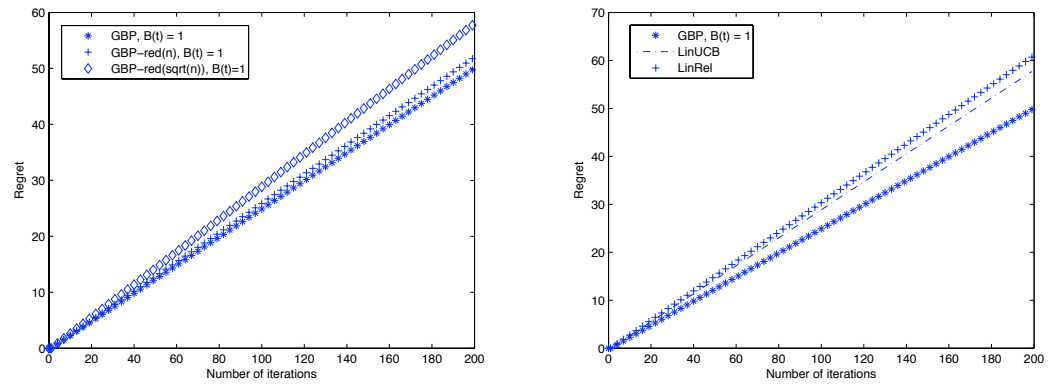
In the next set of experiments (Figure 4.2b), we tested the influence of the size of GPB's training set on the performance of the algorithm. *GPB-red(n)*, where n indicates the size of the training set, with $b(t) = \sqrt{\log t}$ and $b(t) = 1$ performs better than UCB. However, further reduction in the size of the training set to $\sqrt{n}$ adds exploration to the arm selection policy as a result of which *GPB- red*($\sqrt{n}$) with $b(t) = \sqrt{\log t}$ now performs similarly to a random selection policy. As expected, the regret of GPB with fixed $b(t)$ becomes worse when the size of the $T(t)$ set is reduced (Figure 4.3a).

In the last set of experiments, we compared the performance of GPB with



(a) (b)

Figure 4.2: Comparison of regrets of (a) UCB and GPB, and (b) GPB-red(n) and GPB-red(sqrt(n)) .



(a) (b)

Figure 4.3: (a) Comparison of regrets of GPB-red($S(t)$) for different functions of S; (b) Comparison of GPB with LinRel and LinUCB.

LinRel and LinUCB (Figure 4.3b). As mentioned earlier, the performance of LinRel and LinUCB is very similar, however, GPB outperforms both the algorithms. Given the similarities between the three algorithms, one might

expect similar performance. The better-than-expected performance of GPB might be explained by the fact that GPB uses a kernel function to assess the similarities between the arms. An appropriately selected kernel function will allow a better estimation of arms similarities than the linear combination of feature vectors of the arms, which the method applied in LinRel and LinUCB.

4.4 Bayesian Arm Selection

Although the arm selection method described in Section 4.1.1 seems quite natural, in some cases, finding the maximum of the upper confidence function f_t may prove costly, particularly as we would expect the function to become flatter as iterations proceed, as the algorithm aims to explore regions only as our uncertainty about their reward offsets the difference in our estimated reward. In this section, we propose an alternative method for selecting the next arm rather than choosing the point that maximises the upper confidence function based on the sampling of functions from the GP posterior (Głowacka et al. (2009)). Our strategy for selecting arms is to perform Thompson sampling, that is to sample a function f from the posterior distribution and then select $x_t = \text{argmax}_{x \in \mathcal{X}} f(x)$. This implements sampling an arm with the probability that it is the maximum in the posterior distribution, and so implements a Bayesian approach to trading exploration and exploitation. We can interpolate between these methods by sampling a variable number K of functions $f_1, \ldots, f_K$ from the posterior and selecting

$x_t = \text{argmax}_{\substack{\mathbf{x}\in\mathcal{X} \\ 1\leq k\leq K}} \{f_k(\mathbf{x})\}$.

A naive sampling from the posterior distribution would require inverting a matrix indexed by the full grid. We avoid this by iteratively unveiling the posterior sample $f(x)$ only sampling points that are likely to lead to a maximal value. Since the function $f(x)$ is not expected to be flat, only a small number of samples should be required in practice. We would envisage selecting these samples by simple hill climbing heuristics that could work efficiently on the non-flat f.

Below, we briefly describe the arm selection method:

1. Initialisation (t=1): x_1 chosen randomly in $\mathcal{X}$. y_1 is the reward obtained when "playing" x_1. The GP posterior after seeing the data (x_1, y_1) is sampled (f_1) to give iteration at time t and $x_2 = \text{argmax}\{f_1(x)\}$.

2. Iteration at time t: we have played $x_1, \ldots, x_{t-1}$ and have obtained rewards $y_1, \ldots, y_{t-1}$. The GP posterior is sampled to give f_t at the point $x_t = \text{argmax}\{f_{t-1}(x)\}$.

In Algorithm 5 below, we describe the sampling method in detail, which returns the next selected arm x_t. The method is Thompson sampling applied to the GP bandit setting. After seeing only the data $((x_1, y_1)), \ldots, (x_{t=1}, y_{t-1}))$, the posterior has mean $\mu^{(1)}_{t=1}$ and variance $\sigma^{(1)^2}_{t=1}$.

In Figure 4.4, we compare the performance of GPB with and without the inclusion of the Bayesian arm selection method. The results show that combining GPB with the sampling method improves the performance of the

Algorithm 5 Bayesian arm selection

input: T number of iterations, n number of arms,
$x_1, \ldots, x_{t-1}$ arms played up to time $t-1$,
$y_1, \ldots, y_{t-1}$ rewards obtained up to time $t-1$
for $t = 1, \ldots, T$ **do**
  $S = \{x_1, \ldots, x_{t-1}\}$, $V = \{y_1, \ldots, y_{t-1}\}$
  for $j = 1, \ldots, n$ **do**
    if $\mu_t^{(j)}(f_t(x_{t,j})) + B(t)\sigma_t^{(j)}(f_t(x_{t,j})) \geq \max(V)$ **then**
      $v_j = \mathcal{N}(\mu_t^{(j)}(f_t(x_{t,j})), \sigma_t^{(j)^2}(f_t(x_{t,j})))$
      $V = V \cup v_j$
      $S = S \cup x_{t,j}$
      the GP posterior after seeing the data $(x_1, y_1), \ldots, (x_{t-1}, y_{t-1})$ with observation noise σ^2_{noise} and $(x_{t,1}, v_1), \ldots, (x_{t,j}, v_j)$ without noise has mean $\mu_t^{(j+1)}$ and variance $\sigma_t^{(j+1)}$
    end if
  end for
  play arm x_k where $v_k = \max(V)$
  obtain reward y_t
end for

algorithm, confirming our intuition that our Bayesian sampling approach is more likely to lead to choosing the most optimal arm than simply selecting the point that maximises the upper confidence bound. The sampling method relies on the fact the function $f(x)$ becomes flatter as the iterations proceed. It would be interesting to see whether the sampling method would perform equally well for functions that are far more complicated than the Rosenbrock function that we tested it on.

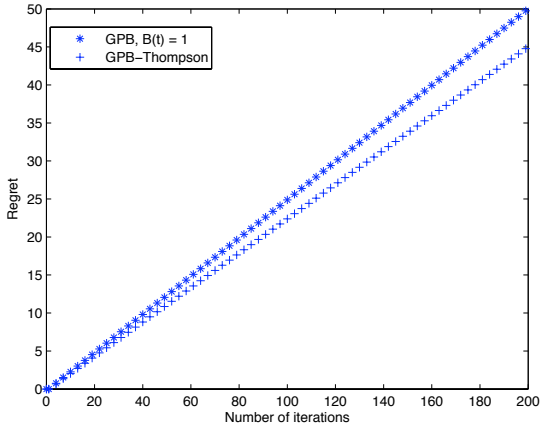


Figure 4.4: Comparison of the GPB with and without the inclusion of the Bayesian arm selection method.

In the next chapter, we consider a problem of grammar learning, where we apply the GP bandit setting combined with the sampling method described in Algorithm 5 with favourable results.

4.5 Conclusions

Exploiting arm dependencies allows GPB and $GPB - red$ to achieve better regrets than UCB, however, $GPB - red$ is more costly than UCB in the number of arms. Using $GPB - red$ would be beneficial in applications where error costs are high enough to warrant the extra computational cost of exploiting arm dependencies. One particular advantage of our approach is that we model dependencies with kernel functions. Consequently, we can consider applications where arms are characterised by sets of features, but also where arms represent objects such as text, images, music, or any structured data for which kernels exist. Thus, our framework can be applied to the web advertisement problem presented by Pandey et al. (2007) by defining a kernel between ads. Deciding which ads to put on which webpage can be considered a content matching problem, and such problems can be modelled as multi-armed bandit problems with dependent arms. Another possible application is product recommendation, where we want to match elements from a set of products with elements from a set of consumers.

Kocsis and Szepesvari (2006) used UCB to devise the UCT (Upper Confidence Trees) tree search algorithm, which was successfully applied to Go game playing (Gelly and Wang (2006)). The UCT algorithm selects paths to explore in the tree in a depth-first manner. At each node, the next node is selected by applying the UCB formula, where each child of that particular node is considered to be an arm with $\mu_j(t)$ calculated as the average of the re-

wards obtained by all tree paths containing node j. In the case of Go, nodes are labelled with Go boards and the reward of a game tree path is given by a Monte-Carlo simulation starting from the deepest node in the path: 1 if at the end of the simulation the game is won and 0 otherwise. It would be interesting to see whether UCT for Go can be improved by using GPB with a Go board kernel instead of UCB. In this particular case, arms are not independent since similar Go boards are likely to lead to similar rewards (Dorard and Shawe-Taylor (2010)).

Chapter 5

Grammar Learning and Gaussian Process Bandits

5.1 Introduction

A major aspect of linguistic theory is to provide an explanation as to how children, after being exposed to limited data, acquire the language of their environment[1]. The rise of "principles and parameters" (Chomsky (1981)) provided a new context for the study of language acquisition. In this approach, a class of languages can be viewed as fixed by parametric variation of a finite number of variables. Thus, acquisition of language (grammar)[2] amounts to fixing correctly the parameters of the grammar that the child is

[1] This chapter is largely based on (Głowacka et al. (2009)).

[2] In the remainder of this chapter, we will use the terms *language* and *grammar* interchangeably

trying to learn. The notion of finite parametrisation of grammar can be applied to syntax (Chomsky (1981)), phonological stress systems (Dresher and Kaye (1990)) or even lexical knowledge (Hale and Keyser (1993)). In this chapter, we will concentrate on the analysis of the "principles and parameters" framework as applied to stress systems (Dresher and Kaye (1990)). This choice has been prompted mainly by two considerations. Firstly, stress systems can be studied in relative independence of other aspects of grammars, i.e. syntax or semantics. Secondly, the parameters of metrical theory exhibit intricate interactions that exceed in complexity the syntactic parameters.

Starting with Gold's seminal paper (Gold (1967)), most research on learnability concentrates on the issue of convergence in the limit. The learner receives a sequence of positive examples from the target language. After each example, the learner either stays in the same state (does not change any of the parameters) or moves to a new state (changes its parameter setting). If, after a finite number of examples, the learner converges to the target language and never changes his guess, then the target language has been identified in the limit. In the Triggering Learning Algorithm (TLA) (Gibson and Wexler (1994)) two additional constraints were added: the single-value constraint, i.e. the learner can change only one parameter value at a time, and the greediness constraint, i.e. if, after receiving an example he cannot recognise, the learner changes one parameter and now can accept the new data, the learner retains the new parameter setting. The TLA is an online learning algorithm that performs local hill climbing. This algorithm, how-

ever, is problematic as positive-only examples can lead to local maxima, i.e. an incorrect hypothesis from which the learner cannot move, thus rendering the parameter space under consideration unlearnable. In order to acquire the target grammar, the learner has to start from very specific points in the parameter space.

Niyogi and Berwick (1996) and Niyogi (2006) model the TLA as a Markov chain and modify the TLA by replacing the local single-step hill climbing procedure with a simple Random Walk Algorithm (RWA). RWA renders the learning process faster and always converges to the correct target language irrespective of the initialisation of the algorithm. Niyogi and Berwick (1996) tested the algorithm on a very small three-parameter system that produced 8 grammars, where each grammar consisted of only up to 18 acceptable examples. Following Niyogi and Berwick (1996) and Niyogi (2006)'s observation that a RWA greatly improves the accuracy and speed of the learning process, we propose a new computational analysis of language acquisition within the "principles and parameters" setting. Our algorithm is set within the general framework of multi-armed bandit problems. The proposed *prior knowledge multi-armed bandit* problem abstracts the idea of a casino of slot machines in which a player has to play machines in order to find out how good they are, but where he has some prior knowledge that some machines are likely to have similar rates of reward. Each grammar is represented as "an arm of a bandit machine" with the mean-reward function drawn from a Gaussian Process specified by a covariance function between grammars. We test

our algorithm on the metrical stress ten-parameter space (Dresher and Kaye (1990)), which gives rise to 216 possible stress systems (as not all parameters are independent). The use of the algorithm, however, can be easily extended to much larger systems. We also compare the performance of our algorithm to that of TLA and RWA and show that it "learns" the correct grammar faster than both TLA and RWA.

As emphasised by Gibson and Wexler (1994) and Niyogi and Berwick (1996), arriving at the correct parameter setting is only one aspect of the language acquisition problem. As noted by Chomsky (1965), an equally important point is how the space of possible grammars is "scattered" with respect to the primary language data. It is possible for two grammars to be so close to each other that it is almost impossible to separate them by psychologically realistic input data. This leads to the question of sample complexity (Niyogi and Berwick (1996)), i.e. how many examples it will take to identify the target grammar. It is of not much use to the learner to be able to arrive at the correct target grammar within the limit if the time required to do so is exponentially long, which renders the learning process psychologically implausible. Thus, rather than concern ourselves with identifying the correct grammar, we will measure the number of errors made by the learner in acquiring the correct grammar. We will give experimental evidence that the number of iterations required to reach a state where the learner makes virtually no mistakes is smaller than the number of grammars to be explored. We will also consider the impact of variations in data distribution and the

presence of noise in the data on the performance of the algorithm.

5.2 Metrical Stress Parameters and the Learning Problem

In this section, we describe the syllable structure and its relevance to stress assignment. Further, we present the stress parameters that we refer to throughout the rest of this chapter. Lastly, we discuss how the data is presented to and assessed by the learner in our system.

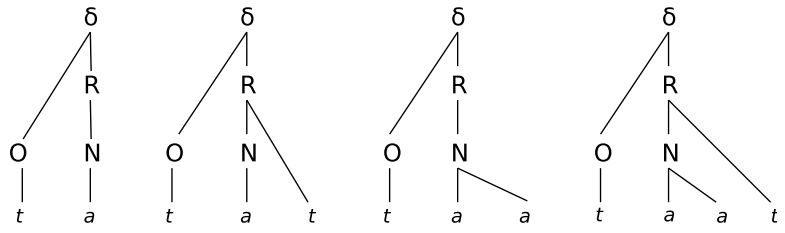
5.2.1 Syllable Structure and Stress Assignment

We assume that the input to the learning process are words. One of the principles shared by most theories of stress systems is that stress is sensitive to representations built on projections from syllable structure. In many languages, stress is sensitive to syllable weight, or quantity. Thus, we also assume the prior operation of rules that convert the speech signal into words and smaller word segments, such as syllables.

In general, syllables can be divided into two parts: an onset (O) and a rhyme (R). The onset consists of the consonant(s) before the syllable peak, which is usually a vowel. The rhyme consists of the vowel, i.e. the nucleus(N), and the consonant(s) following the nucleus, i.e. the coda (C). It is generally agreed that the onset plays no part in stress assignment. How-

ever, in quantity-sensitive languages, the structure of the rhyme plays an important role in stress assignment (see (Davies (1988)) for possible counter examples). Syllables that have only one element in the nucleus position and no coda are classified as light (Figure 5.1a) and as such do not attract stress. Syllables with two elements in the nucleus position count as heavy (Figure 5.1c, d) and attract stress, while syllables with one element in the nucleus and at least one element in the coda position can count as either light or heavy (depending on the setting of parameter 6 below) (Figure 5.1b).

Furthermore, we also assume that various acoustic cues that indicate phonological stress are mapped into one of three degrees of stress. The three levels of stress are primary stress (marked as 2), secondary stress (marked as 1), and lack of stress (marked as 0). For the purpose of our analysis, we assume that every word must have a primary stress.



(a) /ta/ (b) /tat/ (c) /taa/ (d) /taat/

Figure 5.1: Four examples of syllable with different rhyme structure. δ signifies a syllable node; O, R, N and C represent the constituents of the syllable to which the segmental material is attached.

5.2.2 The Stress System Parameters

In metrical theory, stress patterns, and the corresponding differences between languages, are due to metrical structures built on the rhyme of the syllable. The various possibilities of metrical structure construction can be expressed in terms of a series of binary parameters. In our analysis, we consider a 10-parameter model with the following parameters (Dresher and Kaye (1990)):

- P1: The word-tree is strong on the left/right;
- P2: Feet (see definition below) are binary/unbounded;
- P3: Feet are built from left/right;
- P4: Feet are strong on the left/right;
- P5: Feet are quantity sensitive/insensitive;
- P6: Feet are quantity sensitive to the rhyme/nucleus;
- P7: A strong branch of the foot must/must not itself branch;
- P8: There is/is not an extrametrical syllable;
- P9: It is extrametrical on the left/right;
- P10: Feet are/are not non-iterative.

If all the parameters were independent, then we would have $2^{10} = 1024$ possible grammars. However, due to built-in dependencies, there only 216 distinct stress systems (see Dresher and Kaye (1990) for more details).

Let us consider the effect that different parameter settings can have on language structure. For example, P1 tells us where in the word the main stress should fall. If P1 is set to left, then the main stress will fall on the initial syllable, as in Latvian or Hungarian, if, however, we set P1 to right, then the main stress will fall on the final syllable, as in French or Farsi. In many languages, secondary stress can also be observed. In such languages, syllables are first grouped together into feet and every foot receives a stress. If a language has feet, a number of other parameters come into play. P2 allows feet to be at most binary or else unbounded. Selecting binary feet will give an alternating pattern of weak (with stress level 0) and strong (stress level 1 or 2) syllables. We must also set P3, which will trigger the direction of construction from left to right or from right to left. Further, we must also set P4, which allows each foot to be left dominated or right dominated. For example, Maranungku, spoken in Australia, (Hayes (1995)) has the following setting P1[left], P2[binary], P3[left], P4[left], which gives rise to the following alternating pattern of stresses: 201, 2010, etc. On the other hand, Warao, spoken in Venezuela, (Hayes (1995)) has the following setting: P1[right], P2[binary], P3[right], P4[left], which results in the following stress pattern: 020, 01020, 10101020, 010101020.

5.2.3 Inclusion of Prior Knowledge

In Gibson and Wexler (1994) and Niyogi and Berwick (1996), the transition probabilities from one parameter setting state to another are calculated by

counting the number of overlapping input data between each grammar corresponding to each parameter setting. We consider this to be an unrealistic model in that it is not clear how the learner would be able to assess this overlap without knowledge of the grammars and the sentence frequencies. We prefer to work with a weaker assumption of the prior knowledge that learners are equipped with, namely that learners are able to assess similarity of grammars by the *Hamming distance* (Hamming (1950)) between their parameter vectors. This accords with the expectation that the entries in the parameter vector control aspects of the production of sentences that involve varying levels of processing by the learner. Hence, our conjecture is that the parameter settings described above have cognitive correlates that enable the learner to compute the *Hamming distance* between the grammars. One of the questions addressed by the experiments reported below (and answered in the affirmative) is whether this prior knowledge will be sufficient to enable subjects to learn to identify the correct grammar.

5.2.4 The Learning Procedure

Our learning procedure is inspired by the TLA (Gibson and Wexler (1994)). However, contrary to Gibson and Wexler (1994), we do not obey the single-value constraint or the greediness constraint. Our learning procedure is summarised below. In the next section, we describe in more detail how the grammar learning process can be analysed in the GP setting.

- Step 1 [Initialise]: Select a random grammar as your starting point.

- Step 2 [Process input data]: Receive n positive example words from the target grammar (g_{target}). The words are drawn at random from a fixed probability distribution.

- Step 3 [Learnability on error detection]: Check if the currently hypothesised grammar (g_h) can generate the input data and receive a reward r ranging from 0 to 1. $r = 0$ corresponds to a situation, where none of the n words can be found in the hypothesis grammar, while if $r = 1$, all the n words are analysable in the currently hypothesised grammar. The reward function has expected value

$$r = \frac{\sum_{w \in g_h} pr(w)}{\sum_{w \in g_h} pr(w) + \sum_{w \notin g_h \cap w \in g_{target}} pr(w)}, \tag{5.1}$$

  where $pr(w)$ indicates the probability of word w. In the bandit setting applied in this chapter, each grammar corresponds to a bandit arm. Thus, the currently hypothesised grammar can be thought of as a bandit arm that is currently being "played". After testing the hypothesised grammar, or "playing" the bandit arm, the learner receives a reward (as described above).

- Step 4 [Update] After observing a hypothesis grammar, i.e. a bandit arm, and the corresponding reward r, update the GP posterior as described in Section 4.1.

- Step 5 [Grammar selection]: Select a new grammar hypothesis, i.e. a new bandit arm to "play". The new grammar selection procedure is described in detail in Sections 5.3 and 4.4. The newly hypothesised grammar does not necessarily have to allow the learner to analyse all or any of the n input examples.

5.3 The Grammar Learning Problem

As suggested earlier, the problem of grammar learning can be considered as a many-armed bandit problem and the Gaussian Process approach can be used with a covariance function/kernel which applies to different grammars. Learning consists in looking for the "best" grammar, i.e. the one that maximises the reward function.

We assume in our model that the reward of an arm $\mathbf{x}$ is determined by a function f applied at point $\mathbf{x}$ to which Gaussian noise is added. The variance of the noise corresponds to the variability of the reward when always playing the same arm. In order to cope with large numbers of arms, our assumption will be that the rewards of arms are correlated. This can be modelled with a Gaussian Process: by default, we take the mean of the Gaussian Process prior to be $\mathbf{0}$, and we can incorporate prior knowledge on how correlated the arms are in the covariance function between arms. We assume that f is a function drawn from a GP. If arms are indexed in $\mathbb{R}^d$, for example, the covariance function can be chosen to be a Gaussian

kernel $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2}\right)$, whose smoothness σ is adjusted to fit the characteristic length scale which is assumed for f. In the parametric grammar learning problem, each grammar can be associated with an arm, so that looking for the optimal arm corresponds to looking for the optimal grammar given a certain criterion which is incorporated into the reward function.

Let us denote by $\mathcal{X}$ the set of parametrised grammars. In the "principles and parameters" framework, a grammar $\mathbf{x}$ is a binary vector of length d, where d is the number of parameters under consideration. In our case, $d = 10$. We need to define a kernel $\kappa(\mathbf{x}, \mathbf{x}')$/covariance function between grammars. In our experiments, we consider a Gaussian kernel that takes the form:

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\| \mathbf{x} - \mathbf{x}' \|^2}{2\sigma^2}\right) \tag{5.2}$$

where $\| \mathbf{x} - \mathbf{x}' \|^2$ is the *Hamming distance* between the two grammars[3]. In our case, we consider the 216 grammars described earlier. The algorithm plays a sequence of arms and aims at optimally balancing exploration and exploitation. The method for arm (grammar) selection is described in more detail in Section 4.4.

[3]Note that in this formulation, all parameters have equal effect on the produced data. As (Dresher and Kaye, 1990, p. 155, ft. 11) point out, theoretically, it is possible for a small change in the parameter setting to have large effects on the produced data and small changes to have small effects. This problem is not studied in Dresher and Kaye (1990). However, if the parameters have a non-uniform effect on the output data, we could incorporate this information in the covariance function. Our experiments show that in spite of these inaccuracies in the link between the parameters and their effects, our approach is able to learn a correct grammar reliably and quickly.

5.4 Experiments

The aim of the experiments is to investigate the following issues:

1. Whether the learning process can be completed in fewer iterations than the number of grammars under consideration.

2. Whether the learning process can be successful irrespective of the initial grammar selected in Step 1 of the learning algorithm.

3. Whether the learning can take place in an on-line fashion, i.e. the number of errors made as the learning process progresses is gradually being reduced until it reaches 0.

4. Is the algorithm robust with respect to the presence of noise and the input data distribution.

5.4.1 The Data

In our experiments, every input word consists of two parts: syllable representation followed by its stress assignment. In our analysis, we represent light syllables as L, syllables with a branching nucleus as N, and syllables with a branching rhyme as R. For example, a string of the form RLRL2010 represents a four-syllable word with primary stress on the initial syllable and secondary stress on the penultimate syllable; a pattern that can be found in, e.g. Icelandic or Czech. We consider words of up to a length of 7 syllables. This results from 2901 to 3279 words for each of the 216 possible grammars.

Needless to say, a given word can belong to more than one grammar. The number of overlapping words between grammars ranges from 3 to 3189. The average number of overlapping words is 262.

As mentioned earlier, in our stress systems analysis, we follow the principles of metrical stress theory (Hayes (1995) and further developed by Dresher and Kaye (1990)). Hayes (1995) set the standard for much subsequent research in this field by bringing together data from around 400 natural languages and dialects and incorporating them into a unified metrical framework. The 216 grammars discussed here represent stress patterns of a wide range of natural languages from ancient languages (e.g. Latin) through well-known Indo-European languages (e.g. French or Czech) to native Australian or native America languages (e.g. Maranungku or Warao). A more exhaustive list of natural languages corresponding to each of the 216 grammars can be found in Hayes (1995) and Dresher and Kaye (1990).

5.4.2 The Experiment Design and General Results

All the experiments reported below are averaged over 600 runs with a random starting point. This random initialisation allowed us to study the influence of the initialisation step of the learning process. Each run consisted of 400 iterations of the algorithm which we described in detail in Section 4.4. At each iteration the learner is presented with 5 words selected from the target grammar. The frequency with which each word is presented to the learner corresponds to the probability distribution of this word. Note that the learner

is not allowed to use the particular words to inform his learning but only the average error of the currently hypothesised grammar on these words.

Below, we report results for the target grammar with the following parameter setting: P1[right], P2[binary], P3[right], P4[left], P5[QI], P6[rhyme], P7[no], P8[yes], P9[right], P10[yes], although the simulation experimental show that similar results can be reported for the remaining 215 grammars. The data is drawn from a uniform distribution. As mentioned earlier, the error convergence to 0 corresponds to identifying the target grammar. As illustrated in Figure 5.2, the target grammar is identified within 30 - 50 iterations, irrespective of the initialisation step and assuming a uniform distribution over all the input data. Note that an exhaustive search would require "trying" all the 216 possible grammars, thus lengthening the learning process. The faster error convergence results from online nature of our learning algorithm and the incorporation of prior knowledge in our learning scenario. The algorithm is more efficient than one where at each iteration a new grammar was selected completely at random which results in a larger number of errors and a slower convergence rate.

5.4.3 Varying the Probability Distributions

In the second set of experiments, we test the convergence time in two scenarios: (1) when the input data is presented to the learner from a uniform distribution, i.e. the probability to see every word is $1/n$, where n is the number of possible words produced by a given grammar; (2) certain words

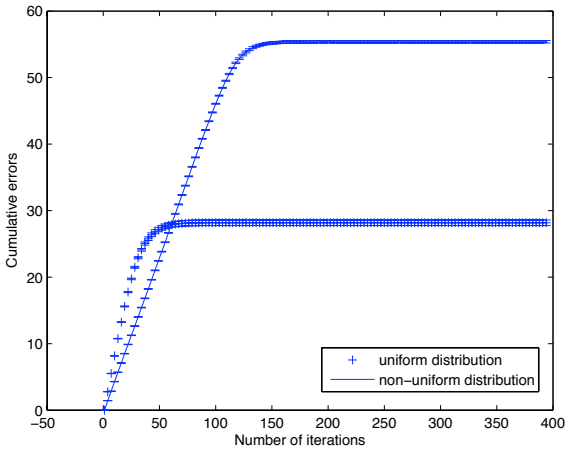


Figure 5.2: Error convergence rate (with standard deviation) of the Prior Knowledge Multi-armed Bandit algorithm when the data is drawn from (a) a uniform distribution and (b) the probability distribution is correlated with the word length and compared to the uniform probability distribution (non-uniform distribution).

are more likely to occur than others. In case (2), the probability distribution is correlated with the word-length, i.e. the shorter a given word is, the higher its probability of occurrence. As can be seen in Figure 5.2, varying the probability distribution affects the convergence rate. When the data is drawn from a non-uniform distribution, the convergence rate is slower, i.e. the target grammar is identified within 80 - 150 iterations, which is still lower than the number of all the 216 possible grammars. As mentioned earlier, the number of overlapping words between the target grammar and the remaining candidate grammars. In certain cases, the overlap between the vocabulary of the target grammar and the hypothesised grammar is over 90%, which means that the grammars can be distinguished by a small number of words and the

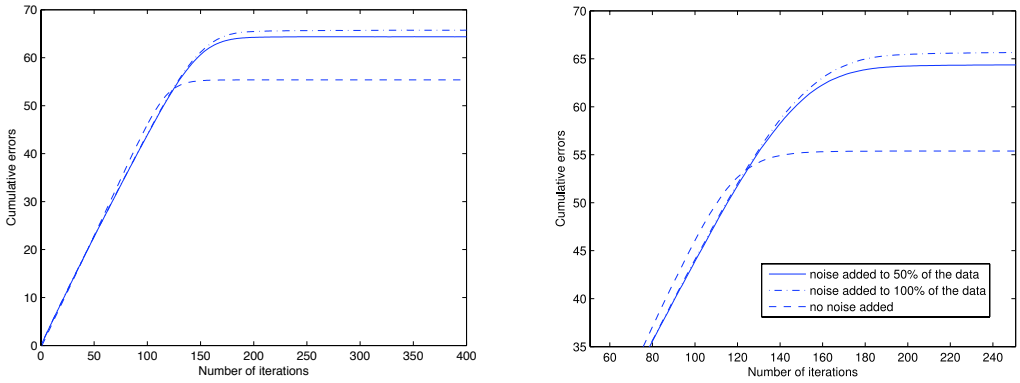
user has to be presented with one of these words in order to reject the hypothesised grammar. If the probabilities of occurrence of the non-overlapping are very low, then the learner might have to wait longer to "come across" them, i.e. it takes more iterations to be presented with a non-overlapping word and thus reject the candidate grammar.

5.4.4 The Impact of Noise

In the third set of experiments, we added noise (ω) to the input data, or, to be more precise to the reward function. Thus, the reward was $r + \omega$, where $\omega = randn * 0.05$. $randn$ is a random value drawn from a normal distribution with mean = 0 and standard deviation = 1. We tested the influence of noise when the noise was added to varying percentage of data ranging from 0% to 100%. Figure 5.3 compares the error rate convergence for cases where noise was added to 0%, 50% and 100% of data, where the data was drawn from a non-uniform distribution. The algorithm performs best with no noise present. However, even with the addition of noise, the correct grammar is identified within 110 – 170 iterations.

5.4.5 Comparison with TLA and RWA

In the last set of experiments, we compared the performance of the "prior knowledge multi-armed bandit" algorithm with that of TLA and RWA. Following Niyogi (2006), we implemented TLA as a Markov chain. Both in

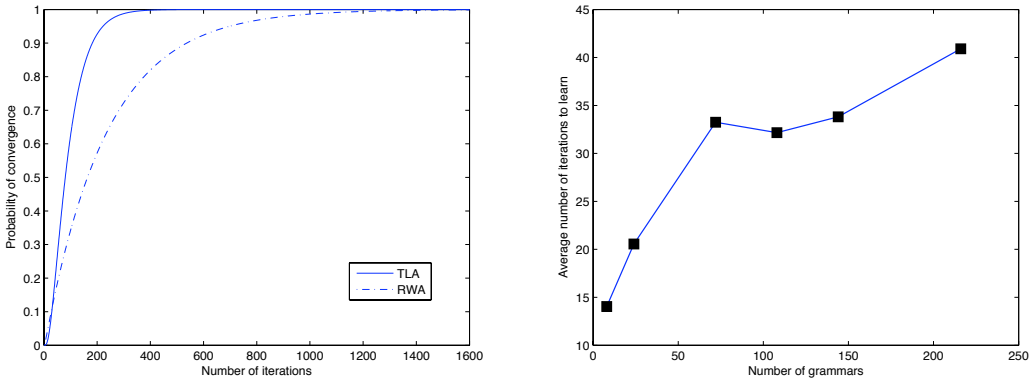


(a) (b)

Figure 5.3: (a) Error convergence rates with noise added to 0%, 50% and 100% of data. (b) expansion of Figure 5.3a in the critical region.

TLA and RWA, the words are drawn from a uniform distribution. The target grammar is the same as the one used in the previous experiments. As discussed earlier, it takes on average 30 - 50 iterations to learn the correct grammar with the prior knowledge multi-arm bandit algorithm. As can be seen in Figure 5.4a, it takes 311 iterations of TLA and 1071 iterations of RWA to learn the target grammar. It must be noted that at each iteration of our algorithm the learner is given a set of 5 words, while in the case of TLA and RWA the learner is given only one word at a time. However, even if we assume the worst-case scenario, where the learner needs 50 iterations of the prior knowledge multi-arm bandit algorithm to acquire the target grammar, we still require only 250 words to learn the language. Learning with TLA and RWA requires 311 and 1071 words, respectively. The prior knowledge multi-arm bandit algorithm converges faster than TLA and RWA in spite

of the fact that TLA and RWA provide the learner with additional information of transition probabilities. Note that the prior knowledge multi-arm bandit algorithm does not take into account this type of extensional information. In particular, it does not have any information about the correctness or otherwise of the grammar for specific words.

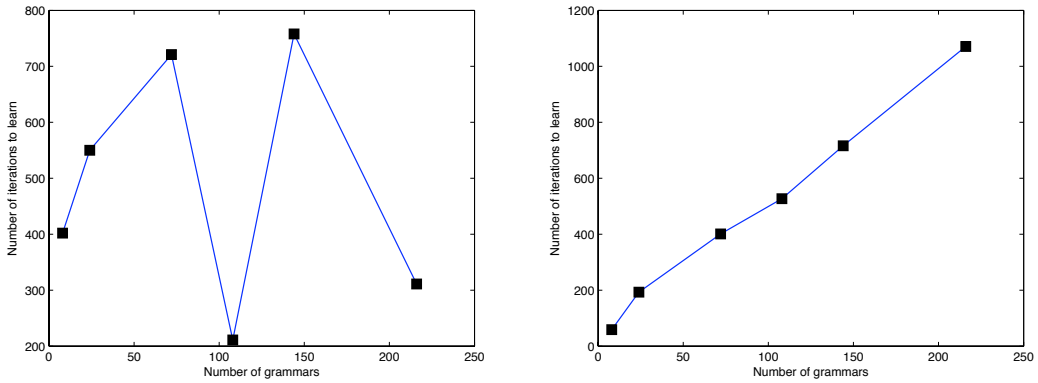


(a) (b)

Figure 5.4: (a) Probability of convergence of the Triggering Learning Algorithm (TLA) and the Random Walk Algorithm (RWA). (b) Average number of iterations required to learn the correct grammar with the prior knowledge multi-armed bandit algorithm as the size of the learning space increases.

Niyogi and Berwick (1996) and Niyogi (2006) showed that in a three-parameter setting with 8 grammars, RWA converges faster than TLA. However, our experiments on a 10-parameter space show that the convergence rate of RWA is much slower than that of TLA. We further compared the convergence rate of the three algorithms as the size of the parameter space, and consequently the number of grammars, increases. We looked at a scenario, where the number of possible grammars was: 8, 24, 72, 108, 144 and 216.

As can be seen in Figures 5.4b and 5.5b, in the case of the prior knowledge multi-arm bandit algorithm and RWA, the complexity of the learning error is affected by the size of the learning space, i.e. the smaller the number of grammars the faster the learning process. The size of the learning space does not have the same effect on the TLA, i.e. there is no correlation between the number of parameters/grammars and the probability of convergence (Figure 5.5a).



(a) (b)

Figure 5.5: (a) Number of iterations required to learn the correct grammar with TLA as the size of the learning space increases. (b) Number of iterations required to learn the correct grammar with RWA as the size of the learning space increases.

5.5 Discussion and Future Directions

The problem of learning parametrised grammars can be approached from many different perspectives. In this chapter, we concentrated on the prob-

lem of error convergence, i.e. how many examples it will take the learner to reach a stage where he can parse correctly all the incoming words. We have presented a new algorithm "prior knowledge multi-armed bandit" and have shown that the algorithm can successfully tackle the problem of sample complexity. The algorithm enables the learner to acquire the target language in an online fashion without the need to resort to searching the entire parameter space and without the danger of getting stuck in a local maximum. We have also shown that the learner can "discover" the parameter setting of the target grammar without direct access to the set difference of words belonging to different grammars, but from the more cognitively realistic access to the *Hamming distance* between the grammars parameter vectors.

A number of directions for future research arise. As the number of parameters increases, so does the complexity of the learning process. It is worth investigating how the error convergence rate will change as the parameter space grows/decreases. Further, we also need to conduct a more extensive empirical analysis of the impact of noise and data distribution on the convergence rate, i.e. how increasing the level of noise or a very unfavourable data distribution will affect the learning process.

Another possible direction is the derivation of a language change model from the current language acquisition model as well as a language acquisition model where the learner is exposed to data coming from different languages or dialects. The procedure discussed in this chapter concentrates on modelling the language acquisition process of a single child. Needless to say, a

language change model would require scaling up the present model to an entire population.

Chapter 6

Image Search with Dirichlet Prior

6.1 Introduction

We consider content-based image retrieval in the case when the user is unable to specify the required content through tags or other image properties. In this type of scenario, the system must extract information from the user through limited feedback. We consider a protocol that operates through a sequence of rounds. In each round, a set of images is displayed and the user must indicate which image is closest to their ideal target. Note that we do not always assume that the target is in the database but rather that there is a hypothetical target image in the user's mind and that the user's likelihood of choosing an image is proportional to a polynomially decaying function of the

distance between the displayed images and this target. While this problem has been studied before (e.g. Cox et al. (2000); Auer and Leung (2009)), we present a novel Bayesian approach that uses latent random variables to model the system's imperfect knowledge about the user's expected response to the images. The proposed approach compares favourably with previous work.

Suppose we are given a database $\mathscr{D}$ of images. For each image $x_i \in \mathscr{D}$, we use a latent variable θ_i taking values in $[0, 1]$ to represent the probability that x_i is an image the user is searching for. Supposing that the user has a single ideal target, the variables have to sum to one:

$$\sum_{x_i \in \mathscr{D}} \theta_i = 1.$$

Since the system has incomplete knowledge about the user's target image, it uses a Dirichlet process over the variables (θ_i) to represent the state of its knowledge. At each iteration, the system samples k images to present to the user, from which the user selects the one closest to the target. Thus, we call our algorithm *Dirichlet Sampling* (DS). The aim of the algorithm is to allow the user to find the target image in as few iterations as possible. Thus, at each iteration based on the user selection, the algorithm updates the probabilities of all the images in the databse with the aim of increasing the probabilities of images that are most likely to be the target and decreasing the probabilities of those that are least likely to be the target. As the iterations progress, the

user is most likely to be presented with images that have high probability (and hence are highly relevant). Thus, the objective of the algorithm is to increase the probabilities of the images in the region of interest to the user thus giving them a better chance to be sampled.

An important aspect of our DS algorithm is the incorporation of an explicit exploration-exploitation strategy in the image sampling process, which greatly improves the performance of the algorithm compared to its main competitors that do not employ exploration-exploitation strategies. Our approach is similar to Thompson sampling (Chapelle and Li (2011)): A sample is drawn from the Dirichlet distribution which represents the knowledge state of the system, adjusted using a temperature to trade-off exploration and exploitation, and the image with the largest probability (in the sample) of being the target is selected; this is repeated k times to obtain the k images presented.

A second aspect of our DS algorithm is the way in which its knowledge of the target is updated given user feedback (selection of one of the k images). We considered a few algorithms to do so: variational Bayes, Gibbs sampling and a simple uniform update. We show in experiments that the simple uniform update performs best. The reason is because both variational Bayes and Gibbs sampling tend to focus on a small set of images (which may or may not contain the target) aggressively, while the uniform update is less aggressive. We discuss this issue further in Section 6.4.1.

Performance of the algorithm is assessed by (1) the number of rounds

needed before the target image is presented to the user or an image is presented that is among the t nearest neighbours of the target in the database, where $t \geq 1$ is a parameter of the problem; (2) the average distance of the k images presented to the user at each iteration from the target (where each presented image is unique). Finally, performance is compared against earlier solutions with favourable results.

An important aspect of the chapter is an attempt to incorporate the algorithm into a real-life online system, where real users are involved and speed is of high importance. In order to account for these factors, we introduced a number of heuristics into the system, such as updating the probabilities of the images or sampling the k images presented to the user. Additionally, in order to be able to cope with very large datasets of images in a timely fashion, we propose a sparse representation for large datasets, which will allow us to roughly approximate the distribution of the images in these large datasets using only a small subset at a time.

6.2 Previous Work

One of the main research problems in content-based image retrieval with relevance feedback is finding a suitable image in as few iterations as possible. In previous research (Zhang and Chen (2002); Tong and Chang (2001); Gosselin et al. (2008); Chang et al. (2005)), active learning was used to select images around the decision boundary for user feedback to speed up the search pro-

cess. However, the user might find it difficult to label images lying close to the decision boundary, which results in noise being present in the user feedback. Recent work by Auer and Leung (2009) explicitly models noisy user feedback by incorporating an exploration-exploitation strategy into their system. In this model, after obtaining the user feedback, the algorithm can efficiently search for suitable images by eliminating the ones that do not match the user's query (see Section 6.7.4). Cox et al. (2000) apply a Bayesian approach to model the system's knowledge about the user's search interest (see Section 6.7.4).

Traditionally, in content-based image retrieval with user feedback, it is assumed that the images in the dataset are not labelled (Chen et al. (2001); Rui and Huang (2000); Rocchio (1971); Tong and Chang (2001)). Metric functions measuring similarity based on low-level visual features can be obtained by discriminative methods. Long-term learning is used with training datasets from the feedback of different users (He et al. (2002); Fournier and Cord (2002); Tao et al. (2006); Koskela and Laaksonen (2003); Tao et al. (2007); Linenthal and Qi (2008); Wacht et al. (2006); Tao and Tang (2004)). However, because of different perceptions about the same object, different users may give different kinds of feedback for the same query target. Short-term learning using feedback from a single user in a single search session can be used to deal with different perceptions of objects.

Recently, a large amount of work (Veltkamp and Tanase (1999); Smeulders et al. (2000); Lew et al. (2006); Crucianu et al. (2004); Datta et al.

(2008)) explored the use of user feedback as training data. Feedback is used to label data points as positive or negative for the training purposes. The problem with this approach is that at each iteration, the user selects the most relevant image, which may not necessarily be very similar to the ideal target image. Images predicted to be positive examples by discriminative methods are usually selected for presentation in each round. This might hinder progress in the search significantly as parts of the search space with images incorrectly predicted as negative are ignored.

6.3 Comparative Feedback

As mentioned earlier, the aim of the content-based image retrieval algorithm proposed in this chapter is to find an image that is close to the user's ideal target image within a small number of query iterations. In this case, the user's feedback is comparative, indicating which images among the ones presented to the user are more similar to the ideal target image. Auer and Leung (2009) and Auer et al. (2011) describe a formal user model for this search scenario and provide some results of initial experiments, indicating that the model captures the issue of delayed feedback very well. The proposed model predicts that the user selects an image from a set of k images with some probability that increases with the similarity between the image and the ideal target image. In the off-line experiments (simulations), we will rely on the user model proposed by Auer and Leung (2009). Additionally, we

will compare the predictions of the user model with the actual user behaviour in order to validate the results of the off-line experiments.

6.3.1 The User Model

Auer and Leung (2009)'s distance based user model specifies the probability of the user choosing a particular image in a given collage of images, assuming that the user has an ideal target image in mind. Let $\mathbf{x}_j$ be the j^{th} image in a set (collage) of k images and let $\mathbf{t}$ be the ideal target image, then the probability of choosing image $\mathbf{x}_j$ is given by:

$$D\{\mathbf{x}^* = \mathbf{x}_j \mid \mathbf{x}_1, \ldots, \mathbf{x}_k; \mathbf{t}\} = (1-\lambda)\frac{S(\mathbf{x}_j, \mathbf{t})}{\sum_{j'=1}^{k} S(\mathbf{x}_{j'}, \mathbf{t})} + \frac{\lambda}{k} \tag{6.1}$$

where S is a similarity measure between images and $0 \leq \lambda \leq 1$ accounts for uniform random noise.

Assuming a distance function $d(\cdot, \cdot)$ between images, a possible choice for the similarity measure $S(\cdot, \cdot)$ is:

$$S(\mathbf{x}, \mathbf{t}) = d(\mathbf{x}, \mathbf{t})^{-a} \tag{6.2}$$

This particular measure decreases polynomially with increasing distance. The parameter $a > 0$ indicates the user "sharpness". Large a implies that the user favours images closer to the target image. With the polynomial similarity measure, the user's response depends on the relative size of the

image distances to the ideal target image[1]. In all the experiments reported below, the values of a and λ in the user model were kept constant at 2 and 0.1, respectively.

In the user model considered here, the search engine supports the user in finding an image that matches their query sufficiently well. In each iteration, the search engine presents a set of k images from a database $\mathscr{D}$ to the user and the user selects the most relevant image from this set. The protocol is described below:

For each iteration $z = 1, 2, \ldots$ of the search:

- The search engine calculates a set of images $\mathbf{x}_{z,1}, \ldots, \mathbf{x}_{z,k} \in \mathscr{D}$ and presents them to the user.
- If one of the presented images matches the user's query, then the search terminates.
- Otherwise, the user chooses one of the images $\mathbf{x}_{z,1}, \ldots, \mathbf{x}_{z,k}$ as most relevant $\mathbf{x}_z^*$ according to a given distribution (see equation 6.1 above) .

6.3.2 The User Model and the Image Selection Problem

As mentioned earlier, the image search algorithm proposed in this chapter is based on Dirichlet Process, however, we are not only interested in developing

[1]We also conducted experiments with exponential similarity measure $\exp\{-ad(x_j, t)\}$, where the user's response depends on the absolute difference of the distances between images, but obtained inferior experimental results.

a new algorithm that can be applied to image retrieval; we are also interested in building a practical application, which inevitably involves a trade-off between algorithm design, available computer architecture and usability, just to name of few requirements that need to be taken into consideration when designing a new system, in particular one that needs to in online and be highly responsive to the user needs. One of the assumptions of our system is that at each iteration, the user is presented with a fixed number of k images and selects one of them, after which we want to be able to divide our dataset into k partitions and treat all the images close to the selected by the user as if they have been selected as well. However, these requirements are not fully compatible with the assumptions behind the DP. On one hand, we would like to be able to work with the assumption that our image database consists of a number of partitions, as in DP, but on the other hand, we want to treat our database as simply consisting of a set of individual images, particularly when it comes to sampling individual images. In this and the following sections, we introduce a number of techniques, some based on heuristics, as to how to combine these two issues.

Let $\mathscr{D}$ be a dataset of n images $\mathscr{D} = \{\mathbf{x}_i, \ldots, \mathbf{x}_n\}$. Let $M = m_1, m_2, \ldots, m_n$ be the base measure defined on $\mathscr{D}$. Initially, we set $m_i = \frac{1}{n}$ for $i = 1, \ldots, n$. Let $\mathbf{x}_z^* \in \{\mathbf{x}_{z,1}, \mathbf{x}_{z,2}, \ldots, \mathbf{x}_{z,k}\}$ be the image chosen by the user at iteration z from among the k presented images $\{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,k}\}$. In the remainder of this section, we suppress the index i to simplify the exposition. In our model, the user only sees k images at each iteration and so we can only observe user's

preference with respect to the k displayed images. However, we want to be able to model the user's preferences with respect to the entire dataset of images. Thus, we view the set of images $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ as partitioning the complete space of images into sets $\mathscr{X}_1, \mathscr{X}_2, \ldots, \mathscr{X}_k$ with

$$\mathscr{X}_j = \{\mathbf{x} : d(\mathbf{x}_j, \mathbf{x}) < d(\mathbf{x}_{j'}, \mathbf{x}), j' \neq j\}. \tag{6.3}$$

In the initial set of experiments, we simulate the user behaviour, thus we need to define how the image $\mathbf{x}_z^*$ is chosen by the user model. If we take the "true" response probability as

$$m_i^* \propto d(\mathbf{x}_i, \mathbf{t})^{-a} \tag{6.4}$$

where $\mathbf{t}$ is the target image, then the user model should choose partition $\mathscr{X}_j$ with probability

$$P(\mathscr{X}_j) = \sum_{i:\mathbf{x}_i \in \mathscr{X}_j} m_i^* \tag{6.5}$$

In order to be able to update the base measures of all images in a given partition at each iteration and be able to approximate the true DP posterior, we derive the base measure updates using Variational Bayes (VB) (Attias (2000); Beal (2003); Jaakkola (2001); Jordan et al. (1999)).

6.4 Variational Bayes

The Bayesian inference problem is one where we have a series of observations $X = \{x_1, \ldots, x_n\}$ and we wish to use them to determine the parameters W of our model:

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)}, \tag{6.6}$$

We are not too concerned with the correct normalisation of the posterior probability and so we can omit the evidence term to obtain:

$$P(W|X) \propto P(X|W)P(W) \tag{6.7}$$

For a general model it may not be possible to evaluate the posterior analytically, in which case it may be necessary to approximate the posterior with a simpler form $q(W)$. We can measure the fit between the approximate distribution and the true one by the Kullback-Leibler divergence:

$$F = \int q(W) \log[\frac{P(X|W)P(W)}{q(W)}]dW \tag{6.8}$$

Inferring the posterior distribution $P(W|X)$ is now a matter of maximising the free energy over $q(W)$. The maximisation of F is equivalent to minimising the Kullback-Leibler (KL) divergence between $q(W)$ and the true posterior:

$$\mathrm{KL}(q|p) = -\int q(W) \log \frac{p(W|X)}{q(W)} dW \tag{6.9}$$

Thus, the log marginal probability is:

$$\log p(X) = F(q) + \mathrm{KL}(q|p) \tag{6.10}$$

If we allow any choice of $q(W)$, then the maximum of the lower bound occurs when the KL divergence disappears, which occurs when $q(W)$ equals the posterior distribution $p(W|X)$. However, computing the true posterior may not be tractable, in which case we might consider a family of distributions $q(W)$ and then look for a member of this family for which the KL divergence is smallest. Thus, it is important to restrict the family membership only to tractable distributions that can also provide a good approximation to the true posterior. A possible family in which to restrict the distributions $q(W)$ is to use a factorised approximation to the true posterior (Attias (2000)). We partition W into groups:

$$q(W) = \prod_{i=1}^{M} q_i(W_i) \tag{6.11}$$

and then, amongst all the distributions $q(W)$, we seek the one for which the lower bound $F(q)$ is the largest.

6.4.1 VB Parameter Updates

Below we illustrate the factorized variational approximation for the Dirichlet Sampling algorithm. As mentioned earlier, Dirichlet distributions $Dir(\alpha)$ are

probability distributions over multinomial parameter vectors. The distribution is parametrised by a vector $\alpha = \{\alpha_1, \ldots, \alpha_n\}$, where $\alpha = (\alpha_1, \ldots, \alpha_n) = \alpha_0(m_1, \ldots, m_n)$, where $(m_1, \ldots, m_n)$ has 1-norm 1 and $\alpha_0 > 1$. The Dirichlet distribution is conjugate to the Multinomial distribution, which gives us the following generative model:

$$\theta|\alpha \backsim Dir(\alpha)$$

$$\beta_i|\theta \backsim Mult(\theta)$$

At each iteration of the DS algorithm, the user is presented with k images and selects one of them as being "most similar" to the ideal target image they have in mind. The selected image is a "proxy" for similar images, which we also consider to be selected by the user. Thus, we want to apply different updates depending on whether a given image was in a partition π selected by the user. Given a set $\{x_1, \ldots, x_k\}$ of observed images, the user selects image $x_j^\star$, which is the proxy for the partition containing that particular image. We denote the partition containing image $x_j^\star$ as $\pi_{x_j^\star}$. Our generative model looks

as follows:

$$P(\theta|\alpha) \propto \prod_{i}^{N} \theta_i^{\alpha_i - 1} \tag{6.12}$$

$$P(\beta|\theta) = \prod_{i}^{N} \theta_i^{\beta_i} \tag{6.13}$$

$$P(x_j^\star \mid \beta, \pi) = \delta \begin{cases} 1 & \text{if} \quad \beta_i \in \pi_{x_j^\star} \\ 0 & \text{otherwise} \end{cases} \tag{6.14}$$

Given our variables θ and β, and the selected image $x_j^\star$, we wish to obtain $p(\theta, \beta)|x_j^\star$. For most models, this is intractable and so we consider distribution $q(\theta, \beta)$, which can be factorised as:

$$q(\theta, \beta) = q_\theta(\theta) q_\beta(\beta) \tag{6.15}$$

which gives us the following definition of $F(q)$:

$$F(q) = \int q_\theta(\theta) q_\beta(\beta) \log \frac{p(x_j^\star, \theta, \beta)}{q_\theta(\theta) q_\beta(\beta)} d\theta d\beta \tag{6.16}$$

The computation of $q(\theta)$ proceeds by its maximisation of $F(q)$:

$$\log q_\theta(\theta) = \int q_\beta(\beta) \log p(x_j^\star, \theta, \beta) d\beta \tag{6.17}$$

We can rewrite Equation 6.17 in terms of an expectation:

$$\log q_\theta(\theta) = \mathbb{E}_{q(\beta)}[\log p(x_j^\star, \theta, \beta) d\beta] \tag{6.18}$$

$$q_\theta(\theta) \propto \exp(\mathbb{E}_{q(\beta)}[\log p(x_j^\star, \theta, \beta) d\beta]) \tag{6.19}$$

Applying the same procedure with respect to $q(\beta)$ will give us:

$$q_\beta(\beta) \propto \exp(\mathbb{E}_{q(\theta)}[\log p(x_j^\star, \theta, \beta) d\beta]) \tag{6.20}$$

We apply the result (6.18) to find the expression for the optimal factor $q_\theta(\theta)$. We only need to retain those terms that have functional dependency on θ as all the remaining terms are absorbed into the normalising constant. Thus, we have:

$$\log q_\theta(\theta_i) = \mathbb{E}_{q(\beta_i)}[\log p(\theta_i) + \log p(\beta_i)] + C \tag{6.21}$$

$$= \mathbb{E}_{q(\beta_i)}[\log(\theta_i^{\alpha_i - 1}) + \log(\theta_i^{\beta_i})] + C \tag{6.22}$$

$$= \mathbb{E}_{q(\beta_i)}[(\alpha_i - 1 + \beta_i) \log \theta_i] + C \tag{6.23}$$

$$= \alpha_i - 1 + \mathbb{E}[\beta_i \log \theta_i] + C \tag{6.24}$$

We can identify that

$$q_\theta(\theta_i) = Dir(\theta | \alpha_i) \tag{6.25}$$

where

$$\alpha_i^\star = \alpha_i + \mathbb{E}[\beta_i] \tag{6.26}$$

Similarly, we can obtain the expression for the optimal $q_\beta(\beta)$:

$$\log q_\beta(\beta_i) = \mathbb{E}_{q(\theta_i)}[\log(\beta_i) + \log(x_j^\star)] + C \tag{6.27}$$

$$= \mathbb{E}_{q(\theta_i)}[\log(\theta_i^{\beta_i}) + \log \delta] + C \tag{6.28}$$

$$= \mathbb{E}_{q(\theta_i)}[\beta_i \log \theta_i + \log \delta(x_i)] + C \tag{6.29}$$

$$= \beta_i \mathbb{E}[\log \theta_i] + \log \delta(x_i) + C \tag{6.30}$$

We can see that

$$q_\beta(\beta_i) = Mult(\beta_i|\gamma_i) \tag{6.31}$$

where

$$\gamma_i \propto \begin{cases} 0 & \text{if} \quad x_i \notin \pi_{x_j^\star} \\ e^{\mathbb{E}[\log \theta_i]} & \text{otherwise} \end{cases} \tag{6.32}$$

Note that for Dirichlet distributions, $e^{\mathbb{E}[\log \theta_i]} \approx \max(0, \alpha_i - .5)$ (Asuncion et al. (2009)), so that the update for α_i is approximately:

$$\alpha_i^* \approx \alpha_i + \begin{cases} \frac{\max(0,\alpha_i - .5)}{\sum_{i:x_i \in \mathscr{X}_j} \max(0,\alpha_i - .5)} & \text{if } x_i \in \mathscr{X}_j \\ 0 & \text{otherwise} \end{cases} \tag{6.33}$$

In other words, the parameters of images in the chosen partition are incremented, with the total increment equal to 1. Further, images x_i whose parameter α_i is already large tends to get a larger share of the increment, while those with small α_i will not get much increment at all. This effect, where the

"rich-gets-richer" can easily force the VB algorithm into a situation where the parameters for a small set of images dominate, even though they are not the true image. The resulting image search algorithm will then never converge to the true image since it will always show one of these dominating images.

To address this problem of premature (and incorrect) convergence, we propose a simple "naïve" change to the parameter update where all images in the chosen partition get incremented by an equal amount.

6.5 Naïve Parameter Update

We consider all images in a given partition as if seen by the user and as being of equal relevance to the user. The intuition behind the algorithm is that each sample is a proxy for the entire partition which comprises it. Thus, when we consider the user's choice, we need to update the base measures (weights) accordingly. Since we are not able to distinguish between the images in a partition, we use the update described in Algorithm 6.

Algorithm 6 Updates of parameters of the DS algorithm

if $x_i \in \mathscr{X}_j$ **then**
  $m_i \leftarrow \frac{\alpha m_i + 1}{\alpha + |\mathscr{X}_j|}$
else
  if $x_i \notin \mathscr{X}_j$ **then**
    $m_i \leftarrow \frac{\alpha m_i}{\alpha + |\mathscr{X}_j|}$
  end if
end if
$\alpha \leftarrow \alpha + \mid \mathscr{X}_j \mid$

In Section 6.7.2, we compare the performance of the DS algorithm using the two types of base measure updates.

6.6 Balancing Exploration versus Exploitation

The final ingredient of the algorithm is how the images presented to the user should be chosen. This involves a trade-off between presenting images that appear promising based on best current estimates of the mean given by the posterior measure $\{m_1, \ldots, m_n\}$ (exploitation), and trying areas where our current estimate could be too pessimistic (exploration). The strategy we adopt to solve this problem is to draw k samples from the posterior distribution, where each sample corresponds to distribution over all the images, and select the images $\mathbf{x}_j$, where $j = 1, \ldots, k$, that have the highest probability in each of these samples.

There is a slight problem with this selection. If we use the individual base measures of the images when drawing the k samples, we will sample from the underlying DP rather than the Dirichlet distribution corresponding to the partition defined by the chosen images. The k selected images are effectively proxies for the approximately n/k images in their respective partitions as the these are the only images that the user can see and give a response to. However, our assumption is that the user is interested not only in the particular image that he selected but also in similar images. Effectively, it is the partition that we wish to choose rather than individual images, i.e. at the

next iteration of the search, we want to present the user with images that are similar (i.e. in the same partiton) to the one selected at the previous iteration. This problem is overcome by multiplying each of the m_j base measures by n/k before drawing each sample, which will allow us to select images with probability that the partition they define contains the target. The n/k multiplier is a very crude measure of the size of each partition - after each iteration of the search algorithm, we cluster the entire dataset based on the distance of each image in the dataset from each of the k presented images and so at each iteration the size and membership of each cluster can change drastically. The experimental data show that the n/k multiplier leads to good experimental results.

We obtain the k images to display by sampling from the Dirichlet distribution. A fast method to sample a random vector from a n-dimensional Dirichlet distribution with parameters $m = \{\alpha m_1, \ldots, \alpha m_n\}$ is to draw n independent random samples from the Gamma distribution: $r_i \sim Gamma(\alpha m_i, 1) = \frac{r_i^{\alpha m_i - 1} e^{-r_i}}{\Gamma(\alpha m_i)}$ and normalise the resulting vector (see Section 2.3.3 for more details). Since we are interested only in the maximum, we can omit the normalisation step. The image selection procedure is described in Algorithm 7.

Algorithm 7 Image selection algorithm.

for $\mathtt{j} = 1, \ldots, k$ **do**
  for $i = 1, \ldots, n$ **do**
    $\mathtt{r}_i \leftarrow \mathtt{randg}(\alpha m_i * n/k)$
  end for
  $[\mathtt{value}_j, \mathtt{index}_j] \leftarrow \max(\mathtt{r})$; $\mathtt{images}_\mathtt{j} \leftarrow \mathtt{index}_j$
end for
Return: array `images` with indices of selected images

6.7 Experiments

In this section, we report experimental results involving the Dirichlet Sampling algorithm. The aim of the experiments was to investigate the following issues:

1. comparison between VB parameter updates and naïve parameter updates;

2. scaling properties of the algorithm;

3. comparison with previous work;

4. real-world performance of the algorithm to compare the fit between our user model and real users' performance.

6.7.1 Simulation Experiments

For the initial set of experiments, we used a subset of the Tiny Images Dataset (Torralba et al. (2008)). The subset that we used consisted of 37900 images

comprising 758 categories. We used the Basic Image Features (BIFs) technique (Crosier and Griffin (2010)) to extract the image features. All reported results are averaged over 1000 searches for randomly selected target images. In the simulation experiments, we used the user model introduced in Section 6.3.1 above. In the first set of experiments we controlled the value of the k parameter, i.e. how many images are presented to the user at each iteration. We ran 1000 experiments where the user was presented with 2, 5, 10 images, respectively. The search terminated when the user was presented with the target image or after 500 iterations of the algorithm.

In the second set of experiments, we introduced an additional parameter, i.e. the size of the image target set. The assumption behind this set of experiments is that the user may not necessarily look for the ideal image but instead terminate the search when presented with an image that is close enough to the ideal image. Thus, the search terminates if the user is presented with at least one of the r images closest to the target. We tested the algorithms when $r = 1$, i.e. the search terminates when the user is presented with the ideal target image, $r = 5$, i.e. the user is presented either with the ideal target image or one of the 4 images closest to the target, $r = 10$, i.e. the search terminates when the user is presented with either the target image or one of the 9 closest images.

6.7.2 VB vs Naïve Parameter Updates

We compared the performance of the DS algorithm using the updates described in Section 6.5 as well as the updates obtained through variational Bayes. Table 6.1 shows the average number of iterations to find the target image with the DS algorithm with both type of updates. The results are averaged over 1000 searches using the dataset mentioned above.

	k	**= 2**	**k**	**= 5**	**k**	**= 10**
Target Size	naïve	VB	naïve	VB	naïve	VB
1	69 (32)	450 (70)	24 (10)	198 (158)	17 (7)	86 (142)
5	53 (31)	431 (135)	19 (8)	89 (141)	13 (5)	38 (89)
10	45 (25)	39 (120)	17 (7)	60 (145)	12 (3)	24 (69)

Table 6.1: Comparison of the performance of the DS algorithm with VB and naïve updates. Standard deviation included in parenthesis

As the experimental results show, the DS algorithm with the "naïve" base measure updates proposed in Algorithm 6 significantly outperform the DS algorithm when combined with the VB updates. A possible explanation might be the value of the updates, which are smaller in case of VB updates. In case of the DS algorithm with non-VB updates, the weights of more promising images are increased by a higher value, which allows the algorithm to sample more relevant images early on in the search. The other possibility is that due to the variable value of each parameter update obtained through VB, as opposed to uniform updated by 1 in non-VB DS, VB zooms in on a particular image too quickly and through future updates cannot easily recover from the initial wrong choices by the user. This conclusion could be supported by the

fact that the inclusion of VB updates leads to a greater increase of standard deviation. A closer analysis of the experimental data shows that in up to 10% of VB experiments, the user fails to find the target image and the simulation run goes through the 500 iterations before it terminates. The data also shows that a large number of VB-updates experiments end after very few iterations, i.e. fewer than 10 iterations, in particular with a larger target size and larger k. However, the 10% of cases when the search fails to lead to the target image increase the mean and the standard deviation, and, from the user perspective, make the algorithm unreliable. In the case of the DS algorithm with naïve updates, there was not a single instance of a search where the target image could not be found within the first 500 iterations of the search session; both the mean and the standard deviation scale with the target size and the size of k.

6.7.3 Gibbs Sampling and Dirichlet Search Algorithm

As the experimental results reported in the previous section show, parameter updates obtained through VB do not improve the performance of the search algorithm. VB provides only an approximation of the "true" posterior. In the next set of experiments, we apply Gibbs sampling (Geman and Geman (1984)) to obtain a better approximation of the posterior. Gibbs sampling is applicable when the joint distribution is not known explicitly or is difficult to sample from directly, but the conditional distribution of each variable is known and is easy to sample from. Gibbs sampling generates an

instance from the distribution of each variable in turn, conditional on the current values of the other variables. The sequence of samples constitutes a Markov chain, and the stationary distribution of that Markov chain is the sought–after joint distribution that we try to approximate, which makes the approximation of the joint distribution derived through Gibbs sampling usually more accurate than that provided by VB.

The Gibbs sampler that we used in our experiments is summarised in Algorithm 8 below. We run the algorithm for only 100 steps as the initial simulations indicate that in the specific application 100 is sufficient for the Markov chain to mix.

Algorithm 8 Gibbs sampler for the DS algorithm

input: base measures $m_{1,\ldots,n} = \frac{1}{n}$;
matrix $C \quad z \times k$ with k images presented at iterations $1, \ldots, z$
for $j = 1, \ldots, k$ **do**
  for $i = 1, \ldots, 100$ **do**
    for $l = 1, \ldots, z$ **do**
      $s = \sum_{h \in O_l}^{n} m_i^{i-1}$, where O_l is partition containing image $C_{y,l}$
      $r_h = \frac{m_h}{s} \quad h \in O_l$
      $v_l \sim Mult((r_h)_{h \in O_l})$
    end for
    $b_h = m_h^{i-1} + \sum_{l=1}^{z} 1(v_l = h)$
    $m_h^s \sim Dir(b_h)$
  end for
  $g_j = Gamma(m^s)$
  [value, index] = max(g_j)
end for
return: array *images* with indices of selected images

Figure 6.1 shows the average distance from the target of 10 images pre-

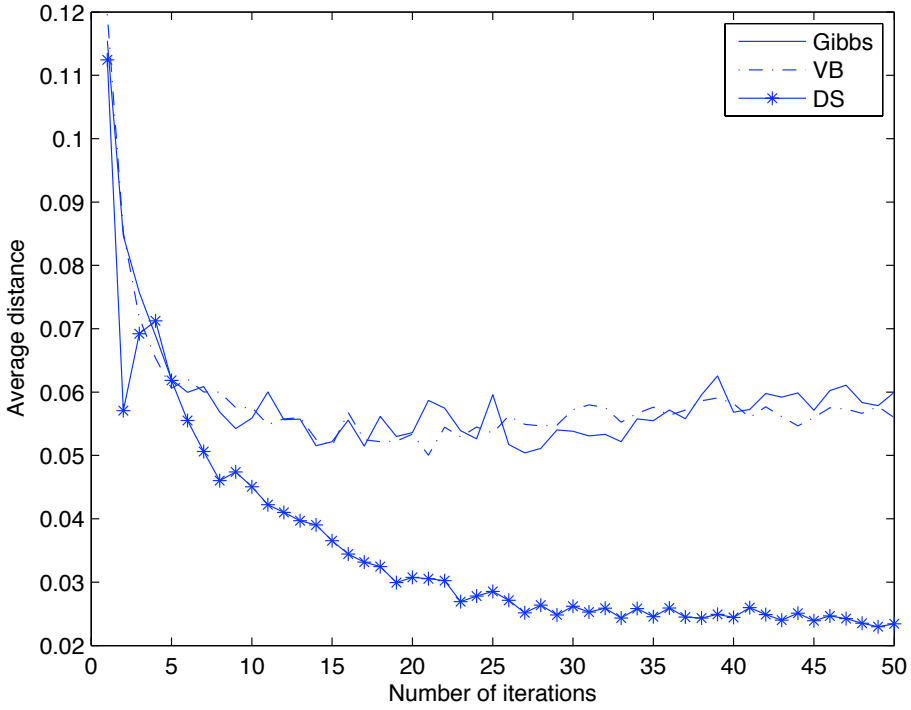


Figure 6.1: Comparison of the convergence of the DS algorithm with a Gibbs sampler (Gibbs), naïve updates (DS) and variational Bayes (VB).

sented to the user over first 50 iterations of the algorithm. The results are averaged over 1000 runs of the search algorithm with a random target image selected in each run. As the results show, incorporating Gibbs sampling into the search procedure does not improve the performance of the search algorithm, indicating that the poor performance is not as a result of the approximate inference but of a poor model, i.e. the correct Bayesian model is inferior to our somewhat ad-hc model proposed in Algorithm 6. The Bayesian model assumes that the user has a single target image in mind and it tries

to get to it from the very beginning of the search, i.e. it zooms in on a small region very quickly by giving the images in that region much higher weights than the rest of the dataset. As a result of these updates, the images in this small region are more likely to be sampled in future iterations. However, the experimental data indicate that this not how the users behave - the users spent the first couple of iterations of the search trying to "familiarise" themselves with the dataset sometimes making suboptimal choices. It's only after a few initial iterations, and after having a better understanding of what type of images the database contains, that the users really try to zoom in on a more specific region. This behaviour, however, is not compatible with the model assumed in VB. Thus, if in the first few iterations the user selects an image in the area where the final target image is placed, then the search with VB updates produces the desired results very quickly and after a small number of iterations the user is presented with the target image. If, however, after the first couple of iterations, the user wishes to steer the search into a slightly different area, it may take the algorithm a few iterations to recover from the suboptimal weight updates from the initial iterations. However, due to the non-uniform weight updates, it takes the DS algorithm with with the VB updates longer converge (since there is a large difference in weights of various images) when compared to the DS algorithm with naïve updates.

6.7.4 Related Image Retrieval Algorithms

In the next set of experiments, we compare the DS algorithm with naïve parameter update against two alternatives: the search algorithm proposed by Auer and Leung (2009) (AL algorithm) and *PicHunter* (Cox et al. (2000)). We selected these two algorithms as a benchmark due to the similarity of the updates between these two approaches and ours. Below, we briefly describe the two systems.

The AL Algorithm

The AL algorithm maintains weights $\mathbf{w} = \{w_1, w_2, \ldots, w_n\}$ on all the images in a given database $\mathscr{D}$. Initially, all $w_j = 1$. At each iteration, k images are presented to the user and the user selects one of the k images as being most relevant to their search. The images presented to the user are randomly selected (without repetition) from the database according to their weights. Let $\mathbf{x}_i^* \in \{\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,k}\}$ be the image chosen by the user as the most relevant at iteration i. If the search has not terminated, then all the images $\mathbf{x}_{i,1}, \ldots, \mathbf{x}_{i,k}$ are not sufficiently relevant and thus their weights are set to 0. All the remaining images are divided into k sets based on their proximity to the k images presented to the user. Weights of the images closest to $\mathbf{x}_i^*$ remain unchanged, while the weights of all the remaining images are demoted by a constant discount factor $0 \leq \beta < 1$ so that $\mathbf{x}_i = \mathbf{x}_i * \beta$. In all the experiments reported below, $\beta = 0.5$ as this was the value that gave the best experimental results.

The *PicHunter* Algorithm

The *PicHunter* image retrieval system (Cox et al. (2000)) uses Bayes' rule to predict the user's target image based on the user's actions. The system maintains a set of probabilities $p = \{p_1, p_2, \ldots, p_n\}$ for every image x_j in a dataset. Initially, all the probabilities $p_i = \frac{1}{n}$. During each iteration i, the search engine displays a set of images $\{x_1, \ldots, x_k\}$ and the user selects[2] an image $x_{i,*}$. After each iteration, the system estimates the probability that image x_j is the user's target image given the session history, and k images with the highest probability are presented to the user. The probabilities are updated as follows:

$$p_{i+1,j} = p_{i,j} \cdot G(d(x_j, x_{i,*})) \tag{6.34}$$

$d(x_j, x_{i,z}) = \parallel x_j - x_{i,*} \parallel$ is the distance between an image x_j and the image $x_{i,*}$ selected by the user in iteration i. In all the experiments reported in this section, we used the Euclidean distance. G is the similarity function and is defined as:

$$G = \exp\left(\frac{-d(x_j, x_{i,*})}{\sigma^2}\right) \tag{6.35}$$

In all the experiments reported in this section, $\sigma = 0.3$ as that was the value that lead to the best experimental results. After the update, all the

[2] *PicHunter* allows the user to select more than one image. For the sake of comparison, the user was allowed to select only one image in every iteration. We also performed experiments where the user was allowed to select more than one image but this fact did not significantly affect the performance of *PicHunter*

probabilities p_j are normalised.

Experimental Results

The results of the experiments are presented in Table 6.2. We report the average number of iterations required to terminate the search and compare the performance of our algorithm against the AL algorithm and *PicHunter*.

	k	**=**	**2**	**k**	**=**	**5**	**k**	**=**	**10**
Target Size	AL	DS	PH	AL	DS	PH	AL	DS	PH
1	227(38)	69(32)	482(15)	117(31)	24(10)	422(44)	92(36)	17(7)	374(58)
5	167(34)	53(31)	461(35)	81(34)	19(8)	390(70)	59(28)	13(5)	328(66)
10	139(35)	45(25)	454(38)	64(30)	17(7)	370(57)	48(24)	12(3)	308(68)

Table 6.2: Comparison of the performance of the AL algorithm, the DS algorithm and *PicHunter* (PH)

The DS algorithm significantly outperforms the AL algorithm and *PicHunter* in all the experiments reported above, which is rather surprising given the similarities of the three algorithms. The disappointing performance of the AL algorithm might be attributed to downgrading the weights by a constant value. If during the initial couple of iterations the user is presented with images that are far away from the target and picks one of these as most relevant, then consequently the weights of the images close to the target will be demoted early on in the search thus making them less likely to be sampled in the following iterations, while the weights of those further away from the target will remain unaffected and thus more likely to be sampled in the future. In this scenario, the lower the value of β, the worse the search results. However, if the user was "lucky" enough to be randomly presented

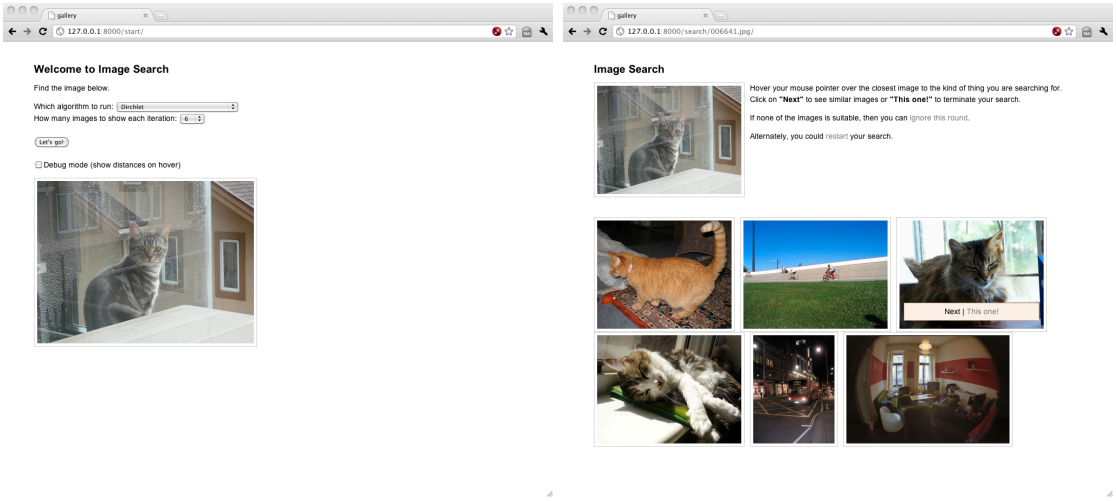
with an image not far away from the target during the first iteration, then the weights of the more relevant images would not be demoted and the search will terminate within a relatively small number of iterations. By comparison, the weight updates of the DS algorithm are more gradual, where the images close to the one selected by the user are gradually increased, while all the remaining ones are gradually decreased. Thus, in the DS algorithm, although the images that are close to the target may have their weights demoted during the initial few iterations of the search algorithm, they still stand a good chance of being sampled later on, thus giving the user an opportunity to pick more relevant images in future iterations. The DS algorithm and the AL algorithm treat the image selected by the user as a proxy for the entire partition of images. Both algorithms aim to identify an area containing images that are most likely to be of interest rather than one particular target image. In the case of *PicHunter*, the probabilities of all the images in the dataset are updated with respect to their distance from the image selected by the user. The algorithm tries to identify a particular image rather than an area from where to sample. Further, *PicHunter* does not really have an exploration stage – rather than sampling images that are most likely to be of interest to the user, at each iteration only the k images with the highest probabilities are displayed. Thus, if all the images displayed initially are far away from the target and all the probabilities are updated with respect to one of these images, then there is little hope for the algorithm to "recover" from the initial bad probability updates as the search progresses. It can ob-

served that the number of iterations required for *PicHunter* to complete the search uniformly large irrespective of the target size and the value of k

6.7.5 Real-life Experiments

In order to assess the compatibility between our user model and the real-life performance of the algorithm, we tested the DS algorithm on real users using the same Tiny Images subset as in our simulation experiments. For this purpose, we built a prototype search engine based on the DS algorithm. The system uses a simple user interface designed to search for target images with minimum training. The user interface is shown in Figure 6.2. At the start of the session, the user is shown the target image that he is expected to search for. The user clicks the start button and is taken to the next page where he is presented with k images. In the example shown in Figure 6.2, 6 images are displayed at each iteration. The target image is always present in the top left corner of the display to avoid possible interference from memory problems. The user selects the image that is most similar to the target image by clicking "next" on that particular image. The process is repeated until the desired image is found, at which point the user clicks "this one!" on the selected image.

The system was tested on 15 users who performed 4 searches each using the DS algorithm. At each iteration, 10 images were displayed. The users were instructed to terminate the search when they found the target image. If after 50 iterations, the target still has not been found, the user was asked



(a) (b)

Figure 6.2: User interface of the image search prototype system.

to select the image most similar to the target and the search terminated. The average number of iterations that was required to find the target image was 20. We expect the results to improve if a larger number of images are displayed at each iteration.

In spite of the fact that the target image was always on display during the search process, some of the users continued with the search even though the target had already been presented at an earlier iteration, while others terminated the search when presented with an image similar to the target image. For this reason, at each iteration we calculated the average distance of the currently displayed images from the target image. Our expectation was that at each iteration the average distance would be getting smaller until eventually it flattens out. In order to assess the compatibility of our

user model with real-life performance of the system, we plotted the average distance of the displayed images from the target image for the simulations as well as real-life experiments. As Figure 6.3 shows, in both cases the algorithm displays a similar convergence patterns.

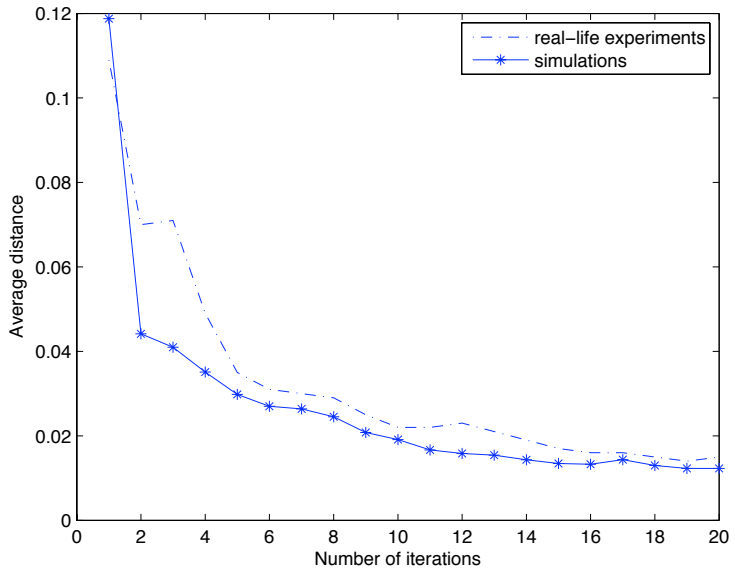


Figure 6.3: Comparison of the convergence of the DS algorithm in real-life experiments and in simulations.

6.8 Sparse Data Representation

Although the performance of the DS algorithm is encouraging, when the image dataset is very large, calculating the distances of all the data points from the k images displayed at each iteration becomes computationally expensive. Assuming that the user is presented with k images and each datapoint is rep-

resented as a d-dimensional feature vector, then each iteration of the search with n images is of the order $O(k \cdot n \cdot d)$. Thus, scaling the algorithm to searches based on datasets consisting of millions of images may be infeasible. We propose a revised version of the DS algorithm, where we work only with a subset, or with a sparse representation, of a given dataset of images. The revised version of the algorithm will allow us to reduce the number of calculations required in this step of the search thus enhancing the scaling properties of the algorithm.

Assuming that we have an image dataset $\mathscr{D}$ consisting of n images, we initially create a small dataset $\mathscr{A} = \{y_1, y_2, \ldots, y_l : y_i \in \mathscr{D} : l \ll n\}$ by randomly selecting l images from $\mathscr{D}$ and we run the algorithm using only this small dataset. At each suscessive iteration t, we replace f, where $f < l$, images with the lowest base measures from $\mathscr{A}$ with new images taken from $\mathscr{D}$. The new f images selected from $\mathscr{D}$ must be different from the ones contained in $\mathscr{A}$. The procedure can be briefly described as follows:

1. Randomly select f images $\{z_1, z_2, \ldots, z_f : z_j \in \mathscr{D} \wedge z_j \notin \mathscr{A}\}$.

2. Assign base measure (weight) values to the newly selected images using the subsampling algorithm described in Algorithm 9.

3. Order the f selected images as well as the images contained in the dataset $\mathscr{A}$ according to the value of their base measures.

4. Remove f datapoints with the lowest base measures from the combined dataset. Thus, the size of $\mathscr{A}$ at iteration t is the same as it was at

iteration $t-1$

Algorithm 9 Subsampling of replacement images.

for $j = 1, \ldots, f$ **do**
  for $i = 1, \ldots, l$ **do**
    $b_i = \mathtt{d}(y_i, z_j)$
  end for
  $[\mathtt{val}, \mathtt{ind}] = \mathtt{min}(b)$
  $s_j = m_{ind}$
end for
return: array $\{s_1, \ldots, s_f\}$ with base measures for images $\{z_1, \ldots, z_f\}$

The base measure s_j of a newly added image z_j takes the value of the base measure of the closest point in $\mathscr{A}$. In all the experiments reported in this section, we used the Euclidean distance to calculate the distance between images y_i and z_j. It should be noted that selecting f images for replacement does not necessarily mean that f new images will be added to $\mathscr{A}$. After the weights of the f images have been calculated, it is possible that some of the newly selected images will be discarded without being added to $\mathscr{A}$.

The intuition behind the algorithm is that in each iteration we remove images with the lowest weights, and consequently the lowest probability of being in the proximity of the target image, and replace them with images with higher weights and hence being of more interest to the user. Thus, although the dataset $\mathscr{A}$ is much smaller than the original image dataset $\mathscr{D}$, as the search progresses, $\mathscr{A}$ contains more and more images close to the target image with fewer and fewer images of little interest to the user. In this way, we can scale up the application of the DS algorithm to very large

datasets as we can still explore the dataset without the need to calculate the weights of all the images at each iteration. The sparse data representation is mainly aimed at applications where a large number of images are involved. In this type of scenario, it is reasonable to expect tens or even hundreds of images to be a very good proxy for the user's idealised target image. Thus, it is not necessary to search the entire dataset to find the target image as long as a subset of the images that are in the proximity of the idealised target are captured by the sparse data representation.

6.8.1 Experimental Results

We tested the performance of the DS algorithm using this sparse data representation and compared it against its performance with a full dataset. The size of the subset $\mathscr{A}$ was 1000 and in each iteration we replaced 200 images with the lowest base measures with new images sampled randomly from the remaining set. We applied the user model described earlier. In the first set of experiments we used the subset of the Tiny Images dataset. As in the earlier simulation experiments, the search terminated when the user was presented with the target image (or an image close to the target). All the results are averaged over 1000 runs of the search algorithm. We report the average number of iterations required to terminate the search. The results of the experiments, summarised in Table 6.3, indicate that even this simple subsampling procedure allows the DS algorithm to find the target image, although the number of iterations required to terminate the search might be

larger than using the entire dataset of images.

Target Size	**k=2**	**k=5**	**k=10**
1	178(38), *69(32)*	102(44), *24(10)*	64(46), *17(7)*
5	145(52), *53(31)*	79(52), *19(8)*	50(34), *13(5)*
10	115(38), *45(25)*	67(42), *17(7)*	39(23), *12(3)*

Table 6.3: Performance of the DS algorithm with sparse data representation. Standard deviation included in parenthesis. Results of experiments with full dataset included in italics.

In Figure 6.4, we plot the average distance between the target image and the k images presented to the user at each iteration using the full dataset and the subsampling algorithm. The results are averaged over 1000 searches for a randomly selected target image. Not surprisingly, the DS algorithm performs better when the entire dataset is available at each iteration, however, the average distance decreases at each iteration for both type of experiments indicating that more relevant images are being presented as the search progresses. The results show that the subsampling technique allows the user to find images close/similar to the target but does not guarantee finding the exact target image in as few iterations as using the entire dataset. However, when working with image databases consisting of millions of images, it is highly likely that a large number (hundreds or even thousands) of images will be virtually indistinguishible to the user and so, from the user prespective, being presented with an image very similar to the ideal target will not adversely affect their experience.

In the next set of experiments, we compared the performance of the DS

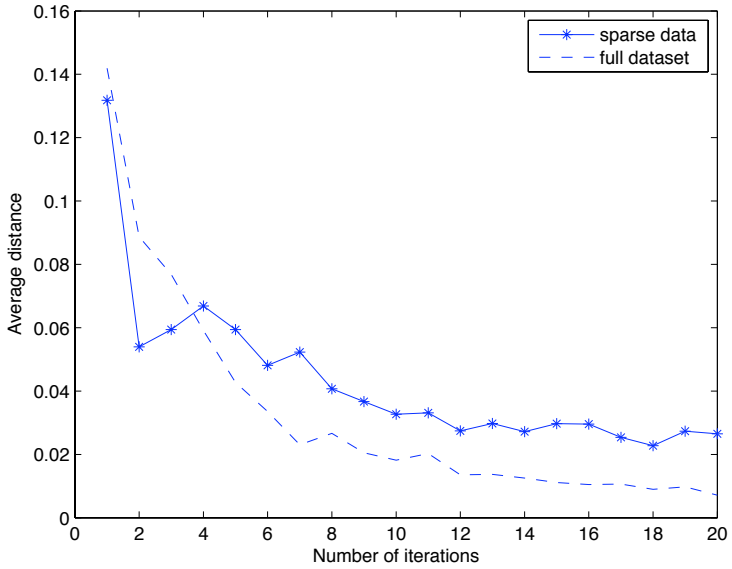


Figure 6.4: Convergence of the DS algorithm in simulations when using the the full dataset and sparse dataset.

algorithm with full and sparse data in real life using the same dataset as before. The experiments included the same setting and the same 15 subjects that took part in the experiments described in Section 6.7.5. Each subject performed 4 searches. The average number of searches using the sparse dataset representation was 29, while the average number of iterations using the full dataset was 20. The averages of the first 20 iterations of the two search strategies are presented in Figure 6.5. The algorithm displays a similar convergence pattern when used with full as well as sparse dataset although in the case of sparse data representation the average distance from the target is slightly higher.

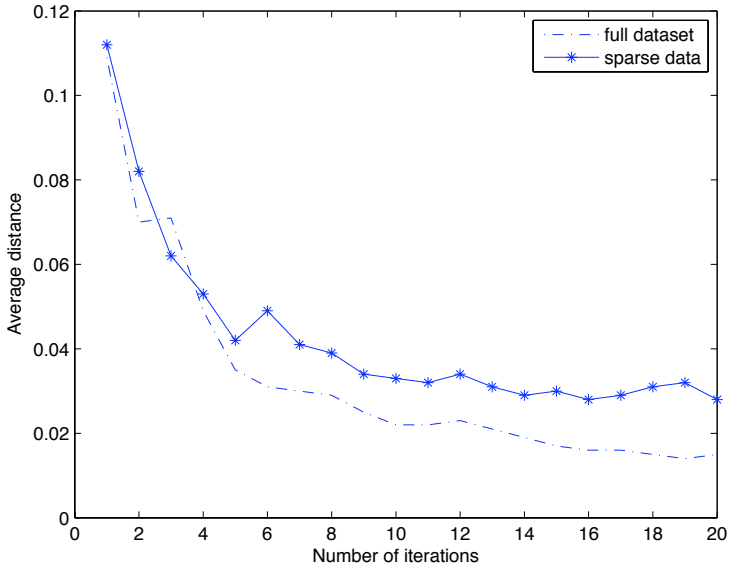


Figure 6.5: Convergence of the DS algorithm in real-life experiments using full and sparse data.

6.8.2 Sparse Data Representation and 4 Million Images Dataset

In order to test the efficiency of the subsampling algorithm on a large dataset, we conducted a set of experiments on a collection of 4 million Flickr images. For every image in the dataset, we used scale-invariant feature transform (SIFT) (Lowe (1999)) to extract the features. We used the PicSOM system (Laaksonen et al. (2000)) to obtain the features. We tested the DS algorithm using sparse data representation with 1000 subset of the 4 million dataset and replacing 200 images at each iteration. Due to the large size of the dataset, we only tested a scenario where the search terminates when the user

is presented with an image that is within 0.01%, 0.1%, 0.5% and 1% distance from the target image. The results, summarised in Table 6.4, are averaged over 1000 searches for randomly selected target image using the user model introduced earlier. The search terminated when the user found, or got closely enough to, the target image or after 500 iterations of the search algorithm.

Target Size	**k=5**	**k=10**	**k=20**
0.01%	251(52)	143(35)	157(42)
0.1%	151(44)	86(39)	50(23)
0.5%	36(22)	20(21)	12(6)
1%	20(23)	11(7)	7(4)

Table 6.4: DS algorithm with sparse data representation using a subset of 1000 and replacing 200 images at each iteration.

The initial results indicate that even when using only 1000 subset of a 4 million dataset, the algorithm preserves its scaling properties with respect to the size of the target set and the number of images displayed at each iteration. It is worth noting that we used only a very small subset consisting of less than 1 % of the entire dataset. Further testing and analysis are required to determine the optimal size of the subset used at each iteration of the search. As in the experiments involving the Tiny Images subset, both the mean and the standard deviation increase with the target size and the size of k. When the target size is 1, there is a sharp increase in the standard deviation as compared to larger target sizes, indicating that the algorithm allows the user to get to the general area of interest, however the number of iterations to find the precise target image may vary considerably.

Figure 6.6 shows the average distance from the target of the k images presented to the user at each iteration as well as the distance of the closest image to the target. As in the previous set of experiments, as the search progresses, more relevant images are displayed.

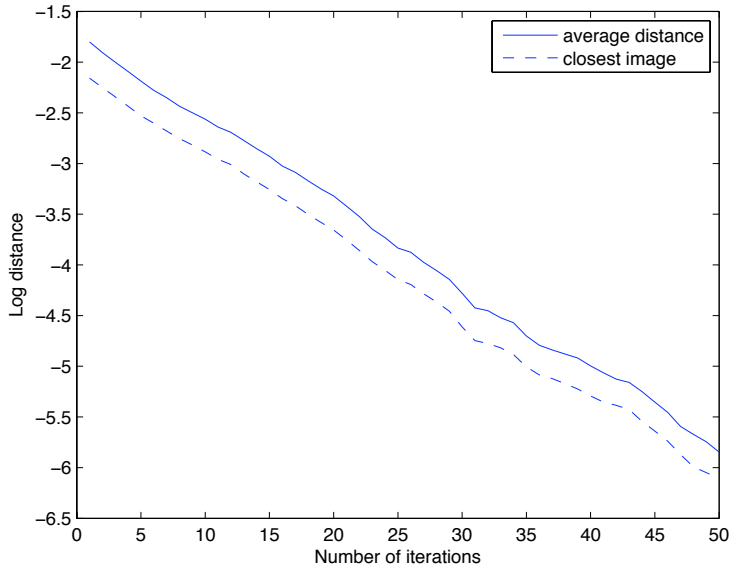


Figure 6.6: Average distance from the target of the images shown to the user in the first 50 iterations of the DS algorithm, and distance of the image closest to the target in each iteration.

6.9 Conclusions and Future Research

We have presented a new approach to content-based image retrieval based on multinomial relevance feedback. We model the knowledge of the system using a Dirichlet process. The model suggests an algorithm for generating

images for presentation that trades exploration and exploitation. The experiments confirm that the new approach outperforms earlier work using a more heuristic strategy. In spite of the encouraging results, further analysis of the algorithm might still lead to further performance enhancements. For example, additional aspects that might be taken into consideration are:

- developing the model so that the user can select more than one image at each iteration. This might be particularly useful when the number of the k displayed images are very similar to each other and the user has difficulties choosing between them, or when the user is interested in an image with a number of features each of which are present in a different image in the k ones on display.

- using different manners of calculating image distances. In all the experiments reported in this chapter we used Euclidean distance, which may not be the best way to assess the similarity between images. In future research, it might be worth investigating other similarity measures.

- subsampling the images for the sparse data representation. Again, we used a very crude and simple way to subsample the images. Further analysis is required to determine the optimal size of the sparse dataset as well as the optimal number of images that should be replaced at each iteration.

- allowing the user to select only a specific part (or parts) of an image thus indicating which aspects of an image are most relevent.

Chapter 7

Summary and Conclusions

In this dissertation, we proposed a number of algorithms that "learn" through interaction with the environment and presented empirical assessments of the proposed methods. We concentrate on algorithms that that trade off exloration/exploitation. In Chapter 3, we introduced a simple bandit algorithm with a beta prior. The algorithm is easy to implement and it has been shown to outperform other alternatives. In its simplest form, the algorithm does not have any parameters to tune, however, the experimental results show that adjusting the prior to reduce exploration can improve the performance. In Chapter 4, we introduced Gaussian Process Bandit and showed that it can be successfully applied in a setting where the bandit arms are not independent. Chapter 5 describes the application of Gaussian Process Bandits to the grammar learning problem. In Chapter 6, we propose an image retrieval algorithm that uses a Dirichlet process over the images to represent the state

of its knowledge. An important aspect of the algorithm is the incorporation of an explicit exploration – exploitation strategy in the image sampling process, which greatly improves the performance.

The dissertation starts off with simple bandit algorithms that assume that the bandit arms under consideration are independent. In the following two chapters, we proceed to more complex settings, where "similar" bandit arms give similar rewards. These type of algorithms can be very useful in scenarios where, unlike in the independent bandit arm setting, it may not always be possible to try every arm in order to gain some information about the reward that it can provide. The application of the Gaussian Process Bandit to modelling the acquisition of metrical structure indeed shows that it possible to find the correct grammar setting without having to try all the possible grammars. The last part of the dissertation was an attempt to apply an exploration-exploitation strategy in an information retrieval setting, where we need to optimise not only accuracy of the algorithms but also speed and, indirectly, the user experience. In theory, we could simply use GP bandits in this setting as well. However, the latter two constraints, i.e. the online nature of information retrieval systems, made the application of GP bandits to the problem described in Chapter 6 impractical. That is why we proposed a new exploration/exploitation strategy based on Dirichlet Process, which is computationally more efficient than GP bandits and, when combined with heuristics such as *sparse data representation*, can be applied to very large datasets. However, one can expect that with the development

of new and faster computer architectures, e.g. graphics cards, even such computationally complex algorithms as GP bandits could form a part of online retrieval systems with millions of images or documents.

Most current theoretical guarantees do not accurately represent the real-world performance of reinforcement learning algorithms, hence we concentrated mostly on possible applications of the algorithms and their performance. The experimental evaluation carried out in this dissertation reveals that the proposed algorithms provide very effective heuristics for addressing the exploration/exploitation trade-off. In fact, the experimental evaluation of the Dirichlet Search Algorithm shows that the suggested heuristic for the algorithm's parameter updates performs better than the more "principled" approach obtained through Variational Bayes or Gibbs sampling.

An important issue, which we briefly touched upon in Chapter 6, is the scalability of the algorithms. The experimental results show that the heuristic approach with sparse data representation allows the algorithm to scale up to a dataset with 4 million images. We will address the issue of scalability in more detail in future research.

Another important direction for future research is the incorporation of human-computer interaction into machine learning (or, depending on the point of view, incorporation of machine learning into human-computer interaction). Machine learning researchers concentrate mostly on creating new ways to define or approximate the learning process. However, in an era where machine learning tries to make itself "useful" in real-life applications, the ac-

curacy of the algorithms is as important as presenting the results to the user in a user-friendly fashion, being able to obtain an input from the user or making the user interaction with the system easy and intuitive.

Bibliography

Agrawal, R. (1995). Sample mean based index policies with o(log n) regret for the multi-armed bandit problem. *Advances in Applied Probability*, 27:1054 – 1078.

Arbib, M. (1995). *The Handbook of Brain Theory and Neural Networks*. The MIT Press.

Asuncion, A., Welling, M., Smyth, P., and Teh, Y. W. (2009). On smoothing and inference for topic models. In *Proceedings of the International Conference on Uncertainty in Artificial Intelligence*.

Attias, H. (2000). A variational bayesian framework for graphical models. In *Proceedings of Neural Information Processing Systems*.

Audibert, J., Munos, R., and Szepesvari, C. (2007). *Variance estimates and exploration function in multi-armed bandit*. CERTIS Research Report 07-31.

Audibert, J., Munos, R., and Szepesvari, C. (2009). Exploration-exploitation trade-off using variance estimates in multi-armed bandits. *Theoretical Computer Science*, 419:1876 – 1902.

Auer, P. (2002). Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3:397 – 422.

Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002a). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47:235 – 256.

Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. (2002b). The nonstochastic multiarmed bandit problem. *SIAM Journal on Computing*, 32 (1):48–77.

Auer, P., Głowacka, D., Leung, A., Lim, S. H., Medlar, A., and Shawe-Taylor, J. (2011). Study of exploration-exploitation trade-offs with delayed feedback. In *FP7–216529 PinView*. European Community's Seventh Framework Programme.

Auer, P. and Leung, A. (2009). Relevance feedback models for content-based image retrieval. In *Multimedia Analysis, Processing and Communications*. Springer.

Banachiewicz, T. (1937). Zur berechnung der determinanten, wie auch der inversen, und zur darauf basierten ausung der systeme linearer gleichungen. *Acta Astronomica, Series C*, 3:41 – 67.

Beal, M. J. (2003). Variational algorithms for approximate bayesian inference. PhD Thesis, Gatsby Computational Neurosicence Unit, University College London.

Bellman, R. (1956). A problem in the sequential design of experiments. *Sankhya*, 16:221 – 229.

Berry, D. and Fristedt, B. (1985). *Bandit Problems*. Chapman and Hall.

Blackwell, D. and MacQueen, J. (1973). Ferguson distributions via polya urn schemes. *Annals of Statistics*, 1:353 – 355.

Boureau, Y., Bach, F., LeCun, Y., and Ponce, J. (2010). Learning mid-level features for recognition. In *CVPR*.

Bubeck, S., Munos, R., Stoltz, G., and Szepesvari, S. (2008). Online optimization in x-armed bandits. In *Proceedings of Neural Information Processing Systems*.

Burnetas, A. and Katehakis, M. (1996). Optimal adaptive policies for sequential allocation problems. *Advances in Applied Mathematics*, 17:122 – 142.

Chang, E., Tong, S., Goh, K., and Chang, C. (2005). Support vector machine concept-dependent active learning for image retrieval. *IEEE Transactions on Multimedia*.

Chapelle, O. and Li, L. (2011). An empirical evaluation of thompson sampling. In *Proceedings of Neural Information Processing Systems*, volume 25.

Chen, Y., Zhou, X., and Huang, T. (2001). One-class svm for learning in image retrieval. In *Proceedings of ICIP*, pages 34 – 37.

Chomsky, N. (1965). *Aspects of the Theory of Syntax*. MIT Press.

Chomsky, N. (1981). *Lectures on government and binding*. Fortis.

Chu, W., Li, L., Reyzin, L., and Schapire, R. E. (2011). Contextual bandits with linear payoff functions. In *Proceedings of the 14th International Conference on Arificial Intelligence and Statistics*.

Cox, I., Miller, M., Minka, T., Papathomas, T., and Yianilos, P. (2000). "the baysian image retrieval system', *pichunter*: theory, implementation, and psychophysical experiments. *IEEE Transactions on Image Processing*, 9:20 – 37.

Crosier, M. and Griffin, L. (2010). Using basic image features for texture classification. *International Journal of Computer Vision*, 88(3):447 – 460.

Crucianu, M., Ferecatu, M., and Boujemaa, N. (2004). Relevance feedback for image retrieval: a short survey. In *State of the art in audiovisual content-based retrieval, information universal access and interaction, including data models and languages*. European Network of Excellence (FP6).

Datta, R., Joshi, D., Li, J., and Wang, J. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*

Davies, S. (1988). Syllable onsets as a factor in stress rules. *Phonology*, 5:1 – 19.

Doob, J. L. (1994). The elementary gaussian processes. *The Annals of Mathematical Statistics*, 3:229 – 282.

Dorard, L., Głowacka, D., and Shawe-Taylor, J. (2009). Gaussian processes modelling of dependencies in multi-armed bandit problems. In *Proceeding of 10th International Symposiums on Operational Research SOR'09*, volume 10.

Dorard, L. and Shawe-Taylor, J. (2010). Gaussian process bandits for tree search. *CoRR*, 1009.0605.

Dresher, B. E. and Kaye, J. D. (1990). A computational learning model for metrical phonology. *Cognition*, 34:137 – 195.

Ferguson, T. (1973). A bayesian analysis of some nonparametric problems. *Annals of Statistics*, 1(2):209 – 230.

Fournier, J. and Cord, M. (2002). Long-term similarity learning in content-based image retrieval. In *Proceedings of ICIP*, pages 441– 444.

Frigyik, B., Kapila, A., and Gupta, M. (2010). Introduction to the dirichlet distribution and related processes. UWEE Technical Report Number UWEETR-2010-0006.

Gelly, S. and Wang, Y. (2006). Exploration exploitation in go: Uct for monte-carlo go. In *Proceedings of Neural Information Processing Systems*, volume 20.

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721 – 741.

Gibson, T. and Wexler, K. (1994). Triggers. *Linguistic Inguiry*, 25(4):407 – 474.

Gittins, J. C. (1979). Bandit processes and dynamic allocation indices. *Journal of the Royal Statistical Society. Series B (Methodological)*, 41:148 – 177.

Gittins, J. C. and Jones, D. M. (1979). A dynamic allocation index for the discounted multiarmed bandit problem. *Biometrika*, 66:561 – 565.

Glimcher, P. (2011). Understanding dopamine and reinforcement learning: The dopamine reward prediction error hypothesis. *Proceedings of the National Academy of Sciences of the United States of America*, 108:15647–15654.

Głowacka, D., Dorard, L., Medlar, A., and Shawe-Taylor, J. (2009). Prior knowledge in learning finite parameter spaces. In *Proceedings of the 14th Conference on Formal Grammar, Bordeaux, 2009*, volume 14.

Głowacka, D. and Shawe-Taylor, J. (2012). Simple bandits revisited (breaking-news abstract). In *15th Conference on Artificial Intelligence and Statistics (AISTATS 15)*.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10(4):407 – 454.

Gosselin, P., Cord, M., and Philipp-Foliguet, S. (2008). Active learning methods for interactive image retrieval. *IEEE Transactions on Image Processing*.

Hale, K. and Keyser, J. (1993). On argument structure and the lexical expression of syntactic relations. In *The view from building 20*, pages 53 – 110. MIT Press.

Hamming, R. (1950). Error detecting and error correcting codes. *Bell System Technical Journal*, 29 (2):147–160.

Hayes, B. (1995). *Metrical Stress Theory: Principles and Case Studies*. The University of Chicago Press.

He, X., King, O., W.Ma, Li, M., and Zhang, H. (2002). Learning a semantic space from user's relevance feedback for image retrieval. *IEEE Trans. Circuits Syst. Video Techn.*, pages 39 – 48.

Jaakkola, T. S. (2001). Tutorial on variational approximation methods. In Opper, M. and Saad, D., editors, *Advances in Mean Field Methods*, pages 129 – 159. MIT Press.

Johnson, N. and Kotz, S. (1977). *Urn Models and their Applications*. John Wiley and Sons.

Jordan, M., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. In Jordan, M., editor, *Learning in Graphical Models*, pages 105 – 162. MIT Press.

Kaelbling, L., Littman, M., and Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237 – 285.

Kleinberg, R., Slivkins, A., and Upfal, E. (2008). Multi-armed bandits in metric spaces. In *Proceedings of STOC*.

Kocsis, L. and Szepesvari, C. (2006). Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning*, volume 20.

Koskela, M. and Laaksonen, J. (2003). Using long-term learning to improve efficiency of content- based image retrieval. In *Proceedings of PRIS*, pages 72–79.

Laaksonen, J., Koskela, M., Laakso, S., and Oja, E. (2000). Picsom – content-based image retrieval with self-organizing maps. *Pattern Recognition Letters*, 21:1199 – 1207.

Lai, T. and Robbins, H. (1985). Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6:4 – 22.

Langford, J. and Zhang, T. (2008). The epoch-greedy algorithm for contextual multi-armed bandits. In *Advances in Neural Information Processing Systems 20*.

Lew, M., Sebe, N., Djeraba, C., and Jain, R. (2006). Content-based multimedia information retrieval: state of the art and challenges. In *TOMCCAP*, pages 1 – 19.

Li, L., Chu, W., Langford, J., and Schapire, R. (2010). A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web*.

Linenthal, J. and Qi, X. (2008). An effective noise-resilient long-term semantic learning approach to content-based image retrieval. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'08)*.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of ICCV 2*.

Lu, T., Pal, D., and Pal, M. (2010). Contextual multi-armed bandits. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*.

Narendra, K. and Thathachar, M. (1989). *Learning Automata: An Introduction*. Prentice Hall.

Neal, R. (1992). Bayesian mixture modeling. In *Proceedings of the Workshop on Maximum Entropy and Bayesian Methods of Statistical Analysis*, pages 197 – 211.

Niyogi, P. (2006). *The Computational Nature of Language Learning and Evolution*. MIT Press.

Niyogi, P. and Berwick, R. C. (1996). A language learning model for finite parameter spaces. *Cognition*, 61:161 – 193.

Pandey, S., Chakrabarti, D., and Agarwal, D. (2007). Multi-armed bandit problems with dependent arms. In *Proceedings of the International Conference on Machine Learning*, volume 20.

Rasmussen, C. (2000). The infinite gaussian mixture model. In *Advances in Neural Information Processing Systems (12)*.

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Robbins, H. (1952). Some aspects of the sequential design of experiments. *Bulletin of the American Mathematical Society*, 58:527 – 535.

Rocchio, J. (1971). Relevance feedback in information retrieval. In Salton, J., editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*, pages 313 – 323. Prentice–Hall.

Rui, Y. and Huang, T. (2000). Optimizing learning in image retrieval. In *Proceedings of CVPR*, pages 1236 –1236.

Sjolander, K., Karplus, K., Brown, M., R.Hughey, Krogh, A., Mian, I., and D., H. (1996). Dirichlet mixtures: a method for improved detection of weak but significant protein sequence homology. *Computational Applied Bioscience*, 12(4):327 – 45.

Smeulders, A., Worring, M., Santini, S., Gupta, A., and Jain, R. (2000). Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 1349 – 1380.

Sutton, R. and Barto, A. (1998). *Reinforcement Learning: an Introduction*. The MIT Press.

Tao, D., Li, X., and Maybank, S. (2007). Negative samples analysis in relevance feedback. *IEEE Trans. Knowl. Data Eng.*, 19(4):568 – 580.

Tao, D. and Tang, X. (2004). Nonparametric discriminant analysis in relevance feedback for content-based image retrieval. In *IEEE International Conference on Pattern Recognition (ICPR)*, pages 1013 – 1016.

Tao, D., Tang, X., Li., X., and Wu, X. (2006). Asymmetric bagging and random subspace for support vector machines –based relevance feedback in image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7):1088 – 1099.

Teh, Y. (2010). Dirichlet processes. In *Encyclopedia of Machine Learning*. Springer.

Teh, Y. W., Görür, D., and Ghahramani, Z. (2007). Stick-breaking construction for the Indian buffet process. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 11.

Tenenbaum, J., Kemp, C., Griffiths, T. L., and Goodman, N. D. (2011). How to grow a mind: Statistics, structure, and abstraction. *Science*, 331 (6022):1279 – 1285.

Thompson, W. (1934). On the theory of apportionment. *American Journal of Mathematics*, 57:450 – 457.

Tong, S. and Chang, E. (2001). Support vector machine active learning for image retrieval. In *Proceedings of ACM Multimedia*, pages 107 – 118.

Torralba, A., Fergus, R., and Freeman, W. (2008). 80 million tiny images: a large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958 – 1970.

Veltkamp, R. and Tanase, M. (1999). Content-based image retrieval systems: a survey. In *State-of-the-Art in Content-Based Image and Video Retrieval*, pages 97 – 124.

Wacht, M., Shan, J., and Qi, X. (2006). A short-term and long-term learning approach for content-based image retrieval. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP'06)*, pages 389 – 392.

Walsh, T. J., Szita, I., Diuk, C., and Littman, M. L. (2009). Exploring compact reinforcement-learning representations with linear regression. In *roceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI-09)*.

Zhang, C. and Chen, T. (2002). An active learning framework for content-based information retrieval. *IEEE Transactions on Multimedia*, pages 260 – 268.