ACTIVE LEARNING: THEORY AND APPLICATIONS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Simon Tong
August 2001

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Daphne Koller
Computer Science Department
Stanford University
(Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

David Heckerman
Microsoft Research

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Christopher Manning
Computer Science Department
Stanford University

Approved for the University Committee on Graduate Studies:

To my parents and sister.

Abstract

In many machine learning and statistical tasks, gathering data is time-consuming and costly; thus, finding ways to minimize the number of data instances is beneficial. In many cases, active learning can be employed. Here, we are permitted to actively choose future training data based upon the data that we have previously seen. When we are given this extra flexibility, we demonstrate that we can often reduce the need for large quantities of data. We explore active learning for three central areas of machine learning: classification, parameter estimation and causal discovery.

Support vector machine classifiers have met with significant success in numerous real-world classification tasks. However, they are typically used with a randomly selected training set. We present theoretical motivation and an algorithm for performing active learning with support vector machines. We apply our algorithm to text categorization and image retrieval and show that our method can significantly reduce the need for training data.

In the field of artificial intelligence, Bayesian networks have become the framework of choice for modeling uncertainty. Their parameters are often learned from data, which can be expensive to collect. The standard approach is to data that is randomly sampled from the underlying distribution. We show that the alternative approach of actively targeting data instances to collect is, in many cases, considerably better.

Our final direction is the fundamental scientific task of causal structure discovery from empirical data. Experimental data is crucial for accomplishing this task. Such data is often expensive and must be chosen with great care. We use active learning to determine the experiments to perform. We formalize the causal learning task as that of learning the structure of a causal Bayesian network and show that active learning can substantially reduce the number of experiments required to determine the underlying causal structure of a domain.

Acknowledgments

My time at Stanford has been influenced and guided by a number of people to whom I am deeply indebted. Without their help, friendship and support, this thesis would likely never have seen the light of day.

I would first like to thank the members of my thesis committee, Daphne Koller, David Heckerman and Chris Manning for their insights and guidance. I feel most fortunate to have had the opportunity to receive their support.

My advisor, Daphne Koller, has had the greatest impact on my academic development during my time at graduate school. She had been a tremendous mentor, collaborator and friend, providing me with invaluable insights about research, teaching and academic skills in general. I feel exceedingly privileged to have had her guidance and I owe her a great many heartfelt thanks.

I would also like to thank the past and present members of Daphne's research group that I have had the great fortune of knowing: Eric Bauer, Xavier Boyen, Urszula Chajewska, Lise Getoor, Raya Fratkina, Nir Friedman, Carlos Guestrin, Uri Lerner, Brian Milch, Uri Nodelman, Dirk Ormoneit, Ron Parr, Avi Pfeffer, Andres Rodriguez, Merhan Sahami, Eran Segal, Ken Takusagawa and Ben Taskar. They have been great to knock around ideas with, to learn from, as well as being good friends.

My appreciation also goes to Edward Chang. It was a privilege to have had the opportunity to work with Edward. He was instrumental in enabling the image retrieval system to be realized. I truly look forward to the chance of working with him again in the future.

I also owe a great deal of thanks to friends in Europe who helped keep me sane and happy during the past four years: Shamim Akhtar, Jaime Brandwood, Kaya Busch, Sami Busch, Kris Cudmore, James Devenish, Andrew Dodd, Fabienne Kwan, Andrew Murray and too many others – you know who you are!

My deepest gratitude and appreciation is reserved for my parents and sister. Without their constant love, support and encouragement and without their stories and down-to-earth banter to keep my feet firmly on the ground, I would never have been able to produce this thesis. I dedicate this thesis to them.

Contents

List of Tables

List of Figures

Part I

Preliminaries

Chapter 1

Introduction

"Computers are useless. They can only give answers."

— Pablo Picasso, (1881-1973).

1.1 What is Active Learning?

The primary goal of machine learning is to derive general patterns from a limited amount of data. The majority of machine learning scenarios generally fall into one of two learning tasks: *supervised learning* or *unsupervised learning*.

The supervised learning task is to predict some additional aspect of an input object. Examples of such a task are the simple problem of trying to predict a person's weight given their height and the more complex task of trying to predict the topic of an image given the raw pixel values. One core area of supervised learning is the *classification* task. Classification is a supervised learning task where the additional aspect of an object that we wish to predict takes discrete values. We call the additional aspect the *label*. The goal in classification is to then create a mapping from input objects to labels. A typical example of a classification task is document categorization, in which we wish to automatically label a new text document with one of several predetermined topics (e.g., "sports", "politics", "business"). The machine learning approach to tackling this task is to gather a training set by manually labeling some number of documents. Next we use a *learner* together with the labeled training set to generate a mapping from documents to topics. We call this mapping a *classifier*. We can then use the classifier to label new, unseen documents.

The other major area of machine learning is the unsupervised learning task. The distinction between supervised and unsupervised learning is not entirely sharp, however the essence of unsupervised learning is that we are not given any concrete information as to how well we are performing. This is in contrast to, say, classification where we are given manually labeled training data. Unsupervised learning encompasses *clustering* (where we try to find groups of data instances that are similar to each other) and *model building* (where we try to build a model of our domain from our data). One major area of model building in machine learning, and one which is central to statistics, is *parameter estimation* . Here, we have a statistical model of a domain which contains a number of parameters that need estimating. By collecting a number of data instances we can use a learner to estimate these parameters. Yet another, more recent, area of model building is the discovery of correlations and causal structure within a domain . The task of causal structure discovery from empirical data is a fundamental problem, central to scientific endeavors in many areas. Gathering experimental data is crucial for accomplishing this task.

For all of these supervised and unsupervised learning tasks, usually we first gather a significant quantity of data that is randomly sampled from the underlying population distribution and we then induce a classifier or model. This methodology is called *passive learning* . A passive learner (Fig. 1.1) receives a random data set from the world and then outputs a classifier or model.

Often the most time-consuming and costly task in these applications is the gathering of data. In many cases we have limited resources for collecting such data. Hence, it is particularly valuable to determine ways in which we can make use of these resources as much as possible. In virtually all settings we assume that we randomly gather data instances that are independent and identically distributed. However, in many situations we may have a way of guiding the sampling process. For example, in the document classification task it is often easy to gather a large pool of *unlabeled* documents. Now, instead of randomly picking documents to be manually labeled for our training set, we have the option of more carefully choosing (or *querying*) documents from the pool that are to be labeled. In the parameter estimation and structure discovery tasks, we may be studying lung cancer in a

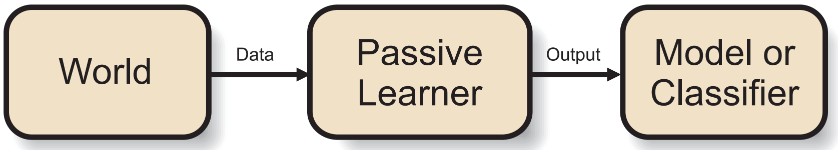


Figure 1.1: General schema for a passive learner.

Figure 1.2: General schema for an active learner.

medical setting. We may have a preliminary list of the ages and smoking habits of possible candidates that we have the option of further examining. We have the ability to give only a few people a thorough examination. Instead of randomly choosing a subset of the candidate population to examine we may query for candidates that fit certain profiles (e.g., “We want to examine someone who is over fifty and who smokes”).

Furthermore, we need not set out our desired queries in advance. Instead, we can choose our next query based upon the answers to our previous queries. This process of guiding the sampling process by querying for certain types of instances based upon the data that we have seen so far is called *active learning*.

1.1.1 Active Learners

An *active learner* (Fig. 1.2) gathers information about the world by asking queries and receiving responses. It then outputs a classifier or model depending upon the task that it is being used for. An active learner differs from a passive learner which simply receives a random data set from the world and then outputs a classifier or model. One analogy is that a standard passive learner is a student that gathers information by sitting and listening to a teacher while an active learner is a student that asks the teacher questions, listens to the answers and asks further questions based upon the teacher’s response. It is plausible that this extra ability to adaptively query the world based upon past responses would allow an active learner to perform better than a passive learner, and indeed we shall later demonstrate that, in many situations, this is indeed the case.

Querying Component

The core difference between an active learner and a passive learner is the ability to ask queries about the world based upon the past queries and responses. The notion of what exactly a query is and what response it receives will depend upon the exact task at hand. As we have briefly mentioned before, the possibility of using active learning can arise naturally in a variety of domains, in several variants.

1.1.2 Selective Setting

In the *selective* setting we are given the ability to ask for data instances that fit a certain profile; i.e., if each instance has several attributes, we can ask for a full instance where some of the attributes take on requested values. The selective scenario generally arises in the *pool-based* setting (Lewis & Gale, 1994). Here, we have a pool of instances that are only partially labeled. Two examples of this setting were presented earlier – the first was the document classification example where we had a pool of documents, each of which has not been labeled with its topic; the second was the lung cancer study where we had a preliminary list of candidates' ages and smoking habits. A *query* for the active learner in this setting is the choice of a partially labeled instance in the pool. The *response* is the rest of the labeling for that instance.

1.1.3 Interventional Setting

A very different form of active learning arises when the learner can ask for experiments involving interventions to be performed. This type of active learning, which we call *interventional*, is the norm in scientific studies: we can ask for a rat to be fed one sort of food or another. In this case, the experiment causes certain probabilistic dependencies in the model to be replaced by our intervention (Pearl, 2000) – the rat no longer eats what it would normally eat, but what we choose it to eat. In this setting a *query* is a experiment that forces particular variables in the domain to be set to certain values. The *response* is the values of the untouched variables.

1.2 General Approach to Active Learning

We now outline our general approach to active learning. The key step in our approach is to define a notion of a **model** $\mathcal{M}$ and its **model quality** (or equivalently, **model loss**, $Loss(\mathcal{M})$) . As we shall see, the definition of a model and the associated model loss can be tailored to suit the particular task at hand.

Now, given this notion of the loss of a model, we choose the next query that will result in the future model with the lowest model loss. Note that this approach is *myopic* in the sense that we are attempting to greedily ask the single next best query. In other words the learner will take the attitude: "If I am permitted to ask just one more query, what should it be?" It is straightforward to extend this framework so as to optimally choose the next query given that we know that we can ask, say, ten queries in total. However, in many situations this type of active learning is computationally infeasible. Thus we shall just be considering the myopic schema. We also note that myopia is a standard approximation used in sequential decision making problems (Horvitz & Rutledge, 1991; Latombe, 1991; Heckerman et al., 1994) .

When we are considering asking a potential query, $\mathbf{q}$, we need to assess the loss of the subsequent model, $\mathcal{M}'$. The posterior model $\mathcal{M}'$ is the original model $\mathcal{M}$ updated with query $\mathbf{q}$ and response $\mathbf{x}$. Since we do not know what the true response $\mathbf{x}$ to the potential query will be, we have to perform some type of averaging or aggregation. One natural approach is to maintain a distribution over the possible responses to each query. We can then compute the *expected* model loss after asking a query where we take the expectation over the possible responses to the query:

$$Loss(\mathbf{q}) = E_{\mathbf{x}} Loss(\mathcal{M}'). \tag{1.1}$$

If we use this definition in our active learning algorithm we would then be choosing the

For $i := 1$ **to** *totalQueries*
  ForEach q **in** *potentialQueries*
    Evaluate $Loss(q)$
  End ForEach
  Ask query q for which $Loss(q)$ is lowest
  Update model $\mathcal{M}$ with query q and response $\mathbf{x}$
End For
Return model $\mathcal{M}$

Figure 1.3: General schema for active learning. Here we ask *totalQueries* queries and then return the model.

query that results in the *minimum expected* model loss.

In statistics, a standard alternative to minimizing the expected loss is to minimize the maximum loss (Wald, 1950) . In other words, we assume the worst case scenario: for us, this means that the response $\mathbf{x}$ will always be the response that gives the highest model loss.

$$Loss(\mathbf{q}) = \max_{\mathbf{x}} Loss(\mathcal{M}'). \tag{1.2}$$

If we use this alternative definition of the loss of a query in our active learning algorithm we would be choosing the query that results in the *minimax* model loss.

Both of these averaging or aggregation schema are useful. As we shall see later, it may be more natural to use one rather than the other in different learning tasks.

To summarize, our general approach for active learning is as follows. We first choose a **model** and **model loss** function appropriate for our learning task. We also choose a method for computing the potential model loss given a potential query. For each potential query we then evaluate the potential loss incurred and we then chose to ask the query which gives the lowest potential model loss. This general schema is outlined in Fig. 1.2.

1.3 Thesis Overview

We use our general approach to active learning to develop theoretical foundations, supported by empirical results, for scenarios in each of the three previously mentioned machine learning tasks: classification, parameter estimation, and structure discovery. We tackle each of these three tasks by focusing on two particular methods prevalent in machine learning: support vector machines (Vapnik, 1982) and Bayesian networks (Pearl, 1988).

For the classification task, support vector machines have strong theoretical foundations and excellent empirical successes. They have been successfully applied to tasks such as handwritten digit recognition, object recognition, and text classification. However, like most machine learning algorithms, they are generally applied using a randomly selected training set classified in advance. In many classification settings, we also have the option of using pool-based active learning. We develop a framework for performing pool-based active learning with support vector machines and demonstrate that active learning can significantly improve the performance of this already strong classifier.

Bayesian networks (Pearl, 1988) (also called directed acyclic graphical models or belief networks) are a core technology in density estimation and structure discovery. They permit a compact representation of complex domains by means of a graphical representation of a joint probability distribution over the domain. Furthermore, under certain conditions, they can also be viewed as providing a causal model of a domain (Pearl, 2000) and, indeed, they are one of the primary representations for causal reasoning. In virtually all of the existing work on learning these networks, an assumption is made that we are presented with a data set consisting of randomly generated instances from the underlying distribution. For each of the two learning problems of parameter estimation and structure discovery, we provide a theoretical framework for the active learning problem, and an algorithm that actively chooses the queries to ask. We present experimental results which confirm that active learning provides significant advantages over standard passive learning.

Much of the work presented here has appeared in previously published journal and conference papers. The chapters on active learning with support vector machines is based on (Tong & Koller, 2001c; Tong & Chang, 2001) and work on active learning with Bayesian networks is based on (Tong & Koller, 2001a; Tong & Koller, 2001b).

Chapter 2

Related Work

There have been several studies of active learning in the supervised learning setting. Algorithms have been developed for classification, regression and function optimization.

For classification, there are a number of active learning algorithms. the Query by Committee algorithm (Seung et al., 1992; Freund et al., 1997) uses a prior distribution over hypotheses. The method samples a set of classifiers from this distribution and queries an example based upon the degree of disagreement between the committee of classifiers. This general algorithm has been applied in domains and with classifiers for which specifying and sampling from a prior distribution is natural. They have been used with probabilistic models (Dagan & Engelson, 1995) and specifically with the naive Bayes model for text classification in a Bayesian learning setting (McCallum & Nigam, 1998). The naive Bayes classifier provides an interpretable model and principled ways to incorporate prior knowledge and data with missing values. However, it typically does not perform as well as discriminative methods such as support vector machines, particularly in the text classification domain (Joachims, 1998; Dumais et al., 1998). Liere and Tadepalli (1997) tackled the task of active learning for text classification by using a committee-like approach with Winnow learners. In Chapter 4, our experimental results show that our support vector machine active learning algorithm significantly outperforms these committee-based alternatives.

Lewis and Gale (1994) introduced *uncertainty sampling* where they choose the instance that the current classifier is most uncertain about. They applied it to a text domain using logistic regression and, in a companion paper, using decision trees (Lewis & Catlett, 1994).

In the binary classification case, one of our methods for support vector machine active learning is essentially the same as their uncertainty sampling method, however they provided substantially less justification as to why the algorithm should be effective.

In the regression setting, active learning has been investigated by Cohn *et al.* (Cohn et al., 1996). They use squared error loss of the model as their measure of quality and approximate this loss function by choosing queries that reduce the statistical variance of a learner. More recently it has been shown that choosing queries that minimize the statistical bias can also be an effective approximation to the squared error loss criteria in regression (Cohn, 1997). MacKay (MacKay, 1992) also explores the effects of different information-based loss functions for active learning in a regression setting, including the use of KL-divergence.

Active learning has also been used for function optimization. Here the goal is to find regions in a space $\mathcal{X}$ for which an unknown function f takes on high values. An example of such an optimization problem is finding the best setting for factory machine dials so as to maximize output. There is a large body of work that explores this task both in machine learning and statistics. The favored method in statistics for this task is the response surface technique (Box & Draper, 1987) which design queries so as to hill-climb in the space $\mathcal{X}$. More recently, in the field of machine learning, Moore *et al.* (Moore et al., 1998) have introduced the $Q2$ algorithm which approximates the unknown function f by a quadratic surface and chooses to query "promising" points that are furthest away from the previously asked points.

To our best knowledge, there is considerably less published work on active learning in *unsupervised* settings. Active learning is currently being investigated in the context of refining theories found with ILP (Bryant et al., 1999). Such a system has been proposed to drive robots that will perform queries whose results would be fed back into the active learning system.

There is also a significant body of work on the design of experiments in the field of optimal experimental design (Atkinson & Bailey, 2001); there, the focus is not on learning the causal structure of a domain, and the experiment design is typically fixed in advanced, rather than selected actively.

One other major area of machine learning is *reinforcement learning* (Kaebling et al., 1996). This does not fall neatly into either a supervised learning task, or an unsupervised learning task. In reinforcement learning, we imagine that we can perform some series of actions in a domain. For example, we could be playing a game of poker. Each action moves us to a different part (or *state*) of the domain. Before we choose each action we receive some (possibly noisy) observation that indicates the current state that we are in. The domain may be stochastic and so performing the same action in the same state will not guarentee that we will end up in the same resulting state. Unlike supervised learning, we are often never told how good each action for each state is. However, unlike in unsupervised learning, we are usually told how good a sequence of actions is (although we still may not know exactly which states we were in when we performed them) by way of receiving a *reward*. Our goal is find a way of performing actions so as to maximize the reward. There exists a classical trade-off in reinforcement learning called the exploration/exploitation trade-off: if we have already found a way to act in the domain that gives us a reasonable reward, then should we continue exploiting what we know by continuing to act the way we are now, or should we try to explore some other part of the domain or way to act in the hope that it may improve our reward. One approach to tackling the reinforcement problem is to build a model of the domain. Furthermore, there are model based algorithms that explicitly have two modes of operation: an explore mode that tries to estimate and refine the parameters of the whole model and an exploit mode that tries to maximize the reward given the current model (Kearns & Singh, 1998; Kearns & Koller, 1999). The explore mode can be regarded as being an active learner; it tries to learn as much about the domain as possible, in the shortest possible time.

Another related area to active learning is the notion of value of information in decision theory. The value of information of a variable is the expected increase in utility that we would gain if we were to know its value. For example, in a printer troubleshooting task (Heckerman et al., 1994), where the goal is to successful diagnose the problem, we may have the option of observing certain domain variables (such as "ink warning light on") by asking the user questions. We can use a value of information computation to determine which questions are most useful to ask.

Although we do not tackle the reinforcement or value of information problems directly in this thesis, we shall re-visit them in the concluding chapter.

Part II

Support Vector Machines

Chapter 3

Classification

"When you have eliminated the impossible,
whatever remains, however improbable, must be the truth."

— Sherlock Holmes,
The Sign of the Four.

3.1 Introduction

Classification is a well established area in engineering and statistics. It is a task that humans perform well, and effortlessly. This observation is hardly surprising given the numerous times in which the task of classification arises in everyday life: reading the time on one's alarm clock in the morning, detecting whether milk has gone bad merely by smell or taste, recognizing a friend's face or voice (even in a crowded or noisy environment), locating one's own car in a parking lot full or other vehicles.

Classification also arises frequently in scientific and engineering endeavors: for example, handwritten character recognition (LeCun et al., 1995), object detection (LeCun et al., 1999), interstellar object detection (Odewahn et al., 1992), fraudulent credit card transaction detection (Chan & Stolfo, 1998) and identifying abnormal cells in cervical smears (Raab & Elton, 1993). The goal of classification is to induce or *learn* a *classifier* that automatically categorizes input data instances. For example, in the handwritten digit task, we would like the learned classifier to classify scanned handwritten digit image data into one of the ten possible digits.

We now come to the issue of how to learn such classifiers. Notice that we ourselves are very good at recognizing the gender of a person's face. However, if we are asked to manually list the set of rules that a computer could use to perform such a task we find it particularly hard. Rather than being manually encoded by humans, classifiers can be learned by analyzing statistical patterns in data. To learn a classifier that distinguishes between male and female faces we could gather a number of photographs of people's faces, manually label each photograph with the person's gender and use the statistical patterns present in the photographs together with their labels to induce a classifier. One could argue that, for many tasks, this process mimics how humans learn to classify objects too – we are often not given a precise set of rules to discriminate between two sets of objects; instead we are given a set of positive instances and negative instances and we learn to detect the differences between them ourselves.

3.2 Classification Task

3.2.1 Induction

By far the most standard and general classification task is the *inductive* classification task. This task is broken into two phases. The first phase is the *training phase*:

- **Input**: independent and identically distributed data from some underlying population: $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ where each data instance resides in some space $\mathcal{X}$. We are also given their labels $\{y_1 \ldots y_n\}$ where the set of possible labels $\mathcal{Y}$, is discrete. We call this labeled data the *training set*.

- **Output**: a classifier. This is a function: $f : \mathcal{X} \to \mathcal{Y}$.

Once we have a classifier, we can then use it to automatically classify new, unlabeled data instances in the *testing phase*:

- We are presented with independent and identically distributed data from the same underlying population as in the training phase: $\{\mathbf{x}'_1 \ldots \mathbf{x}'_{n'}\}$. This previously unseen, unlabeled data is called the *test set*.
- We use our classifier f to label each of the instances in turn.

We measure performance of our classifier by seeing how well it performs on the test set.

3.2.2 Transduction

An alternative classification task is the *transductive* task. In contrast to the inductive setting where the test set was unknown, in the transductive setting we know our test set before we start learning anything at all. The test set is still unlabeled, but we know $\{\mathbf{x}'_1 \ldots \mathbf{x}'_{n'}\}$. Our goal is to simply provide a labeling for the test set. Thus, our task now consists of just one phase:

- **Input**: independent and identically distributed data from some underlying population: $\{\mathbf{x}_1 \ldots \mathbf{x}_n\}$ where each data instance resides in some space $\mathcal{X}$. We are also given their labels $\{y_1 \ldots y_n\}$ where the set of possible labels $\mathcal{Y}$, is discrete. We are also given unlabeled i.i.d. data $\{\mathbf{x}'_1 \ldots \mathbf{x}'_{n'}\}$.
- **Output**: a labeling $\{y'_1 \ldots y'_{n'}\}$ for the unlabeled data instances.

Notice that we can simply treat the transductive task as an inductive task by pretending that we do not know the unlabeled test data and then proceeding wit the standard inductive training and testing phases. However, there are a number of algorithms (Dempster et al., 1977; Vapnik, 1998; Joachims, 1998) that can take advantage of the unlabeled test data to improve performance over standard learning algorithms which just treat the task as a standard inductive problem.

3.3 Active Learning for Classification

In many supervised learning tasks, labeling instances to create a training set is time-consuming and costly; thus, finding ways to minimize the number of labeled instances is beneficial.

Usually, the training set is chosen to be a random sampling of instances. However, in many cases active learning can be employed. Here, the learner can actively choose the training data. It is hoped that allowing the learner this extra flexibility will reduce the learner's need for large quantities of labeled data.

Pool-based active learning was introduced by Lewis and Gale (1994). The learner has access to a pool of unlabeled data and can request the true class label for a certain number of instances in the pool. In many domains this is a reasonable approach since a large quantity of unlabeled data is readily available. The main issue with active learning in this setting is finding a way to choose good queries from the pool.

Examples of situations in which pool-based active learning can be employed are:

- **Web searching.** A Web based company wishes to gather particular types of pages (e.g., pages containing lists of people's publications). It employs a number of people to hand-label some web pages so as to create a training set for an automatic classifier that will eventually be used to classify and extract pages from the rest of the web. Since human expertise is a limited resource, the company wishes to reduce the number of pages the employees have to label. Rather than labeling pages randomly drawn from the web, the computer uses active learning to request targeted pages that it believes will be most informative to label.

- **Email filtering.** The user wishes to create a personalized automatic junk email filter. In the learning phase the automatic learner has access to the user's past email files. Using active learning, it interactively brings up a past email and asks the user whether the displayed email is junk mail or not. Based on the user's answer it brings up another email and queries the user. The process is repeated some number of times and the result is an email filter tailored to that specific person.

- **Relevance feedback.** The user wishes to sort through a database/website for items (images, articles, etc.) that are of personal interest; an "I'll know it when I see it" type of search. The computer displays an item and the user tells the learner whether the item is interesting or not. Based on the user's answer the learner brings up another item from the database. After some number of queries the learner then returns a number of items in the database that it believes will be of interest to the user.

The first two examples involve induction. The goal is to create a classifier that works well on unseen future instances. The third example is an example of transduction. The learner's performance is assessed on the remaining instances in the database rather than a totally independent test set.

We present a new algorithm that performs pool-based active learning with support vector machines (SVMs). We provide theoretical motivations for our approach to choosing the queries, together with experimental results showing that active learning with SVMs can significantly reduce the need for labeled training instances.

The remainder of this chapter is structured as follows. Section 3.4 discusses the use of SVMs both in terms of induction and transduction. Section 3.5 then introduces the notion of a *version space*. Section 3.6 provides theoretical motivation for using the version space as our **model** and its size as the measure of **model quality** leading us to three methods for performing active learning with SVMs. In the following chapter, Sections 4.1 and 4.2 present experimental results for text classification and image retrieval domains that indicate that active learning can provide substantial benefit in practice.

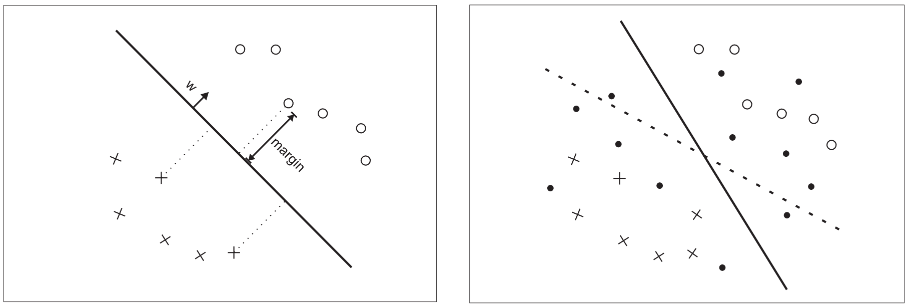
3.4 Support Vector Machines

3.4.1 SVMs for Induction

Support vector machines (Vapnik, 1982) have strong theoretical foundations and excellent empirical successes. They have been applied to tasks such as handwritten digit recognition (LeCun et al., 1995), object recognition (Nakajima et al., 2000), and text classification (Joachims, 1998; Dumais et al., 1998).

We consider SVMs in the binary classification setting. We are given training data $\{\mathbf{x}_1 \dots \mathbf{x}_n\}$ that are vectors in some space $\mathcal{X} \subseteq \mathbb{R}^d$. We are also given their labels $\{y_1 \dots y_n\}$ where $y_i \in \{-1, 1\}$. In their simplest form, SVMs are hyperplanes that separate the training data by a maximal margin (see Fig. 3.1(a)). All vectors lying on one side of the hyperplane are labeled as -1, and all vectors lying on the other side are labeled as 1. The training instances that lie closest to the hyperplane are called *support vectors*.

More generally, SVMs allow one to project the original training data in space $\mathcal{X}$ to a



(a) (b)

Figure 3.1: (a) A simple linear support vector machine. (b) A SVM (dotted line) and a transductive SVM (solid line). Solid circles represent unlabeled instances.

higher dimensional feature space $\mathcal{F}$ via a Mercer kernel operator K . In other words, we consider the set of classifiers of the form: [1]

$$f(\mathbf{x}) = \left(\sum_{i=1}^{n} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \right). \tag{3.1}$$

When K satisfies Mercer's condition (Burges, 1998) we can write: $K(\mathbf{u}, \mathbf{v}) = \Phi(\mathbf{u}) \cdot \Phi(\mathbf{v})$ where $\Phi : \mathcal{X} \rightarrow \mathcal{F}$ and "$\cdot$" denotes an inner product. We can then rewrite f as:

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}), \text{ where } \mathbf{w} = \sum_{i=1}^{n} \alpha_i \Phi(\mathbf{x}_i). \tag{3.2}$$

Thus, by using K we are implicitly projecting the training data into a different (often higher dimensional) feature space $\mathcal{F}$. It can be shown that the maximal margin hyperplane in $\mathcal{F}$ is of the form of Eq. (3.1).[2] The SVM then computes the α_is that correspond to the maximal margin hyperplane in $\mathcal{F}$. By choosing different kernel functions we can implicitly project the training data from $\mathcal{X}$ into spaces $\mathcal{F}$ for which hyperplanes in $\mathcal{F}$ correspond to more complex decision boundaries in the original space $\mathcal{X}$.

Two commonly used kernels are the polynomial kernel given by $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$

[1]Note that, as we define them, SVMs are functions that map data instances $\mathbf{x}$ into the real line $(-\infty, +\infty)$, rather than to the set of classes $\{-1, +1\}$. To obtain a class label as an output, we typically threshold the SVM output at zero so that any point $\mathbf{x}$ that the SVM maps to $(-\infty, 0]$ is given a class of -1, and any point $\mathbf{x}$ that the SVM maps to $(0, +\infty]$ is given a class of $+1$.

[2]In our description of SVMs we are only considering hyperplanes that pass through the origin. In other words, we are asuming that there is no bias weight. If a bias weight is desired, one can alter the kernel or input space to accomodate it.

which induces polynomial boundaries of degree p in the original input space[3] $\mathcal{X}$, and the radial basis function kernel $K(\mathbf{u}, \mathbf{v}) = (e^{-\gamma(\mathbf{u}-\mathbf{v})\cdot(\mathbf{u}-\mathbf{v})})$ which induces boundaries by placing weighted Gaussians upon key training instances. Fig. 3.2 shows the decision boundary in the input space $\mathcal{X}$ of an SVM using a polynomial kernel of degree 5. The curved decision boundary in $\mathcal{X}$ corresponds to the maximal margin hyperplane in feature set $\mathcal{F}$.

Algorithmically, the α_i parameters that specify the SVM can be found in polynomial time by solving a convex optimization problem (Vapnik, 1995):

$$\begin{array}{ll} \text{maximize}_{\boldsymbol{\alpha}} & \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}) \\ \text{subject to:} & \qquad \alpha_i > 0 \quad i = 1 \ldots n. \end{array}$$

For the majority of this chapter we assume that the modulus of the training data feature vectors are constant , i.e., for all training instances $\mathbf{x}_i$, $||\Phi(\mathbf{x}_i)|| = \lambda$ for some fixed λ. The quantity $||\Phi(\mathbf{x}_i)||$ is always constant for radial basis function kernels, and so the assumption has no effect for this kernel. For $||\Phi(\mathbf{x}_i)||$ to be constant with the polynomial kernels we require that $||\mathbf{x}_i||$ be constant. It is possible to relax this constraint on $\Phi(\mathbf{x}_i)$ and we discuss this possibility at the end of Section 3.6.

We also assume linear separability of the training data in the feature space. This restriction is much less harsh than it might at first seem. First, the feature space often has a very high dimension and so in many cases it results in the data set being linearly separable. Second, as noted by Shawe-Taylor and Cristianini (1999), it is possible to modify any kernel so that the data in the new induced feature space is linearly separable.[4]

3.4.2 SVMs for Transduction

The previous section discusses SVMs within the framework of induction. It assumes a labeled training set of data and the task is to create a classifier that has good performance on

[3] Note that, unlike the simple Euclidean inner product, a polynomial kernel of degree one induces a hyperplane in $\mathcal{X}$ that does not need to pass through the origin.

[4] This modification is done by redefining for all training instances $\mathbf{x}_i$: $K(\mathbf{x}_i, \mathbf{x}_i) := K(\mathbf{x}_i, \mathbf{x}_i) + \nu$ where ν is a positive regularization constant. This transformation essentially achieves the same effect as the soft margin error function (Cortes & Vapnik, 1995) commonly used in SVMs. It permits the training data to be linearly non-separable in the original feature space.

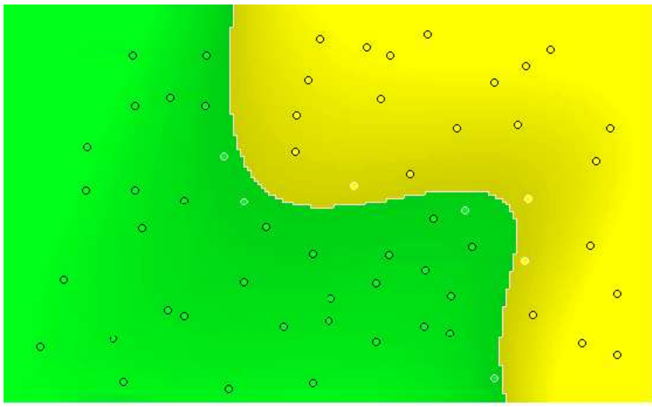


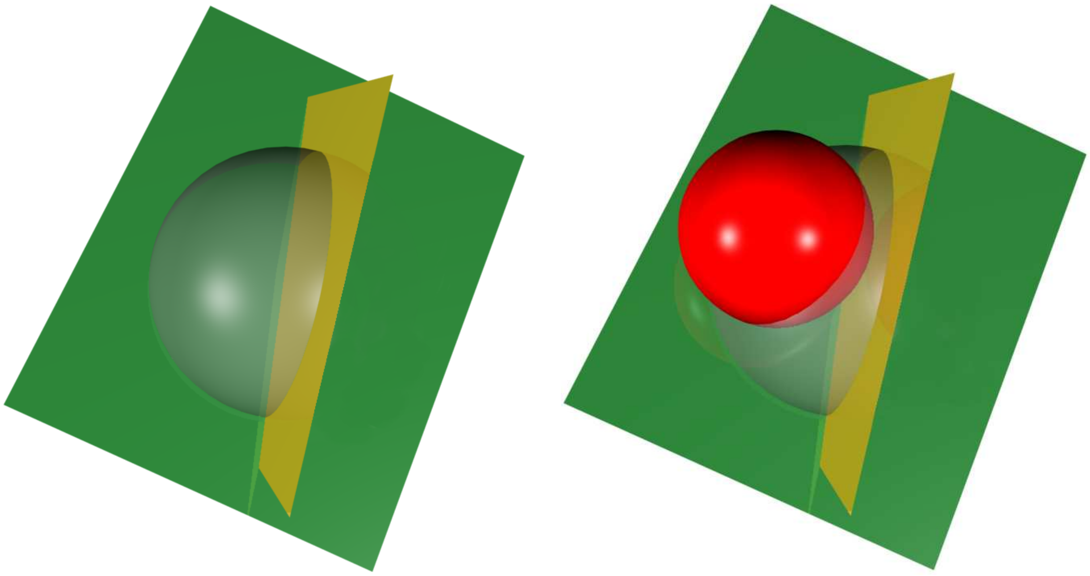
Figure 3.2: A support vector machine using a polynomial kernel of degree 5.

unseen test data. In addition to regular induction, SVMs can also be used for transduction. Here, we are first given a set of both labeled and unlabeled data. The learning task is to assign labels to the unlabeled data as accurately as possible. SVMs can perform transduction by finding the hyperplane that maximizes the margin relative to both the labeled and unlabeled data. See Figure 3.1(b) for an example. Recently, *transductive SVMs* (TSVMs) have been used for text classification (Joachims, 1999), attaining some improvements in precision/recall breakeven performance over regular inductive SVMs.

Unlike an SVM, which has polynomial time complexity, the cost of finding the global solution for a TSVM grows exponentially with the number of unlabeled instances. Intuitively, we have to consider all possible labelings of the unlabeled data, and for each labeling, find the maximal margin hyperplane. Therefore one generally uses an approximate algorithm instead. For example, Joachims (Joachims, 1999) uses a form of local search to label and relabel the unlabeled instances in order to improve the size of the margin.

3.5 Version Space

Given a set of labeled training data and a Mercer kernel K, there is a set of hyperplanes that separate the data in the induced feature space $\mathcal{F}$. We call this set of consistent hypotheses the *version space* (Mitchell, 1982) . In other words, hypothesis f is in the version space if for every training instance $\mathbf{x}_i$ with label y_i we have that $f(\mathbf{x}_i) > 0$ if $y_i = 1$ and $f(\mathbf{x}_i) < 0$ if $y_i = -1$. More formally:



(a) (b)

Figure 3.3: (a) Version space duality. The surface of the hypersphere represents unit weight vectors. Each of the two hyperplanes corresponds to a labeled training instance. Each hyperplane restricts the area on the hypersphere in which consistent hypotheses can lie. Here version space is the surface segment of the hypersphere closest to the camera. (b) An SVM classifier in a version space. The dark embedded sphere is the largest radius sphere whose center lies in version space and whose surface does not intersect with the hyperplanes. The center of the embedded sphere corresponds to the SVM, its radius is proportional to the margin of the SVM in $\mathcal{F}$ and the training points corresponding to the hyperplanes that it touches are the support vectors.

Definition 3.5.1 *Our set of possible hypotheses is given as:*

$$\mathcal{H} = \left\{ f \mid f(\mathbf{x}) = \frac{\mathbf{w} \cdot \Phi(\mathbf{x})}{\|\mathbf{w}\|} \textit{ where } \mathbf{w} \in \mathcal{W} \right\},$$

where our parameter space $\mathcal{W}$ *is simply equal to* $\mathcal{F}$*. The* version space*,* $\mathcal{V}$ *is then defined as:*

$$\mathcal{V} = \{f \in \mathcal{H} \mid \forall i \in \{1 \dots n\} \;\; y_i f(\mathbf{x}_i) > 0\}.$$

Notice that since $\mathcal{H}$ *is a set of hyperplanes, there is a bijection between unit vectors* $\mathbf{w}$ *and hypotheses* f *in* $\mathcal{H}$*. Thus we will redefine* $\mathcal{V}$ *as:*

$$\mathcal{V} = \{\mathbf{w} \in \mathcal{W} \mid \|\mathbf{w}\| = 1, \; y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0, i = 1 \dots n\}.$$

Definition 3.5.2 *The size or* area *of a version space, Area*$(\mathcal{V})$ *is the surface area that it occupies on the hypersphere* $\|\mathbf{w}\| = 1$.

Note that a version space only exists if the *training* data are linearly separable in the feature space. As we mentioned in Section 3.4.1, this restriction is not as limiting as it first may seem.

There exists a duality between the feature space $\mathcal{F}$ and the parameter space $\mathcal{W}$ (Vapnik, 1998; Herbrich et al., 1999) which we shall take advantage of in the next section: points in $\mathcal{F}$ correspond to hyperplanes in $\mathcal{W}$ and *vice versa*.

By definition points in $\mathcal{W}$ correspond to hyperplanes in $\mathcal{F}$. The intuition behind the converse is that observing a training instance $\mathbf{x}_i$ in the feature space restricts the set of separating hyperplanes to ones that classify $\mathbf{x}_i$ correctly. In fact, we can show that the set of allowable points $\mathbf{w}$ in $\mathcal{W}$ is restricted to lie on one side of a hyperplane in $\mathcal{W}$. More formally, to show that points in $\mathcal{F}$ correspond to hyperplanes in $\mathcal{W}$, suppose we are given a new training instance $\mathbf{x}_i$ with label y_i. Then any separating hyperplane must satisfy $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0$. Now, instead of viewing $\mathbf{w}$ as the normal vector of a hyperplane in $\mathcal{F}$, think of $\Phi(\mathbf{x}_i)$ as being the normal vector of a hyperplane in $\mathcal{W}$. Thus $y_i(\mathbf{w} \cdot \Phi(\mathbf{x}_i)) > 0$ defines a half space in $\mathcal{W}$. Furthermore $\mathbf{w} \cdot \Phi(\mathbf{x}_i) = 0$ defines a hyperplane in $\mathcal{W}$ that acts as one of the boundaries to version space $\mathcal{V}$. Notice that version space is a connected region on the surface of a hypersphere in parameter space. See Figure 3.3(a) for an example.

SVMs find the hyperplane that maximizes the margin in the feature space $\mathcal{F}$. One way to pose this optimization task is as follows:

$$\begin{array}{ll} \text{maximize}_{\mathbf{w}\in\mathcal{F}} & \min_i\{y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i))\} \\ \text{subject to:} & \|\mathbf{w}\| = 1 \\ & y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i)) > 0 \;\; i = 1\ldots n. \end{array}$$

By having the conditions $\|\mathbf{w}\| = 1$ and $y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i)) > 0$ we cause the solution to lie in the version space. Now, we can view the above problem as finding the point $\mathbf{w}$ in the version space that maximizes the distance: $\min_i\{y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i))\}$. From the duality between feature and parameter space, and since $\|\Phi(\mathbf{x}_i)\| = \lambda$, each $\frac{\Phi(\mathbf{x}_i)}{\lambda}$ is a unit normal vector of a hyperplane in parameter space. Because of the constraints $y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i)) > 0 \;\; i = 1\ldots n$ each of these hyperplanes delimit the version space. The expression $y_i(\mathbf{w}\cdot\Phi(\mathbf{x}_i))$ can be regarded as:

$\lambda \times$ the distance between the point $\mathbf{w}$ and the hyperplane with normal vector $\Phi(\mathbf{x}_i)$.

Thus, we want to find the point $\mathbf{w}^*$ in the version space that maximizes the minimum distance to any of the delineating hyperplanes. That is, SVMs find the center of the largest radius hypersphere whose center can be placed in the version space and whose surface does not intersect with the hyperplanes corresponding to the labeled instances, as in Figure 3.3(b).

The normals of the hyperplanes that are touched by the maximal radius hypersphere are the $\Phi(\mathbf{x}_i)$ for which the distance $y_i(\mathbf{w}^*\cdot\Phi(\mathbf{x}_i))$ is minimal. Now, taking the original rather than dual view, and regarding $\mathbf{w}^*$ as the unit normal vector of the SVM and $\Phi(\mathbf{x}_i)$ as points in features space we see that the hyperplanes that are touched by the maximal radius hypersphere correspond to the support vectors (i.e., the labeled points that are closest to the SVM hyperplane boundary).

The radius of the sphere is the distance from the center of the sphere to one of the touching hyperplanes and is given by $y_i(\mathbf{w}^*\cdot\frac{\Phi(\mathbf{x}_i)}{\lambda})$ where $\Phi(\mathbf{x}_i)$ is a support vector. Now, viewing $\mathbf{w}^*$ as a unit normal vector of the SVM and $\Phi(\mathbf{x}_i)$ as points in feature space, we have that the distance $y_i(\mathbf{w}^* \cdot \frac{\Phi(\mathbf{x}_i)}{\lambda})$ is:

$$\frac{1}{\lambda} \times \text{ the distance between the support vector } \Phi(\mathbf{x}_i) \text{ and the hyperplane with normal vector } \mathbf{w},$$

which is the margin of the SVM divided by λ. Thus, the radius of the sphere is proportional to the margin of the SVM.

3.6 Active Learning with SVMs

3.6.1 Introduction

In pool-based active learning we have a pool of unlabeled instances. It is assumed that the instances $\mathbf{x}$ are independently and identically distributed and their labels are distributed according to some conditional distribution $P(Y \mid \mathbf{x})$.

Given an unlabeled pool U, an *SVM active learner* ℓ has three components: (f, q, X). The first component is an SVM classifier, $f : \mathcal{X} \to [-1, 1]$, trained on the current set of labeled data X (and possibly unlabeled instances in U too). The second component $q(X)$ is the querying function that, given a current labeled set X, decides which instance in U to query next. The active learner can return a classifier f after each query (*online learning*) or after some fixed number of queries.

The main difference between an active learner and a passive learner is the querying component q. This component tells us which unlabeled pool instance to query next, which brings us to the issue of how to design such a function. We will use our general approach for active learning presented in Section 1.2. We shall first define a **model** and **model quality** or, equivalently, its **model loss**. We shall then choose the pool instance that improves the model quality the most.

3.6.2 Model and Loss

We choose to use the version space as our **model** , and the size of version space as the **model loss** . Thus, we shall choose to query pool instances that attempt to reduce the size of the version space as much as possible. Why should this be a good choice of model and model loss? Suppose $\mathbf{w}^* \in \mathcal{W}$ is the unit parameter vector corresponding to the SVM that we would have obtained had we known the actual labels of *all* of the data in the pool. We know that $\mathbf{w}^*$ must lie in each of the version spaces $\mathcal{V}_1 \supset \mathcal{V}_2 \supset \mathcal{V}_3 \ldots$, where $\mathcal{V}_i$ denotes the version space after i queries. Thus, by shrinking the size of the version space as much as possible with each query, we are reducing as fast as possible the space in which $\mathbf{w}^*$ can lie. Hence, the SVM that we learn from our limited number of queries will lie close to $\mathbf{w}^*$.

We need one more definition before we can proceed:

Definition 3.6.1 *Given an active learner ℓ, let $\mathcal{V}_i$ denote the version space of ℓ after i queries have been made. Now, given the $(i+1)$th query $\mathbf{x}_{i+1}$, define:*

$$\begin{aligned} \mathcal{V}_i^- &= \mathcal{V}_i \cap \{\mathbf{w} \in \mathcal{W} \mid -(\mathbf{w} \cdot \Phi(\mathbf{x}_{i+1})) > 0\}, \\ \mathcal{V}_i^+ &= \mathcal{V}_i \cap \{\mathbf{w} \in \mathcal{W} \mid +(\mathbf{w} \cdot \Phi(\mathbf{x}_{i+1})) > 0\}. \end{aligned}$$

So $\mathcal{V}_i^-$ and $\mathcal{V}_i^+$ denote the resulting version spaces when the next query $\mathbf{x}_{i+1}$ is labeled as -1 and 1 respectively.

We wish to reduce the version space as fast as possible. Intuitively, one good way of doing this is to choose a query that halves the version space. More formally, we can use the following lemma to motivate which instances to query:

Lemma 3.6.2 *Suppose we have an input space $\mathcal{X}$, finite dimensional feature space $\mathcal{F}$ (induced via a kernel K), and parameter space $\mathcal{W}$. Suppose active learner ℓ^* always queries instances whose corresponding hyperplanes in parameter space $\mathcal{W}$ halves the area of the current version space. Let ℓ be any other active learner. Denote the version spaces of ℓ^* and ℓ after i queries as $\mathcal{V}_i^*$ and $\mathcal{V}_i$ respectively. Let $\mathcal{P}$ denote the set of all conditional distributions of y given $\mathbf{x}$. Then,*

$$\forall i \in \mathbb{N}^+ \ \sup_{P \in \mathcal{P}} E_P[\mathit{Area}(\mathcal{V}_i^*)] \leq \sup_{P \in \mathcal{P}} E_P[\mathit{Area}(\mathcal{V}_i)],$$

with strict inequality whenever there exists a query $j \in \{1 \ldots i\}$ by ℓ that does not halve version space V_{j-1}.

Proof. The proof is straightforward. The learner ℓ^* always chooses to query instances that halve the version space. Thus $Area(\mathcal{V}^*_{i+1}) = \frac{1}{2}Area(\mathcal{V}^*_i)$ no matter what the labeling of the query points are. Let r denote the dimension of feature space $\mathcal{F}$. Then r is also the dimension of the parameter space $\mathcal{W}$. Let S_r denote the surface area of the unit hypersphere of dimension r. Then, under any conditional distribution P, $Area(\mathcal{V}^*_i) = \frac{S_r}{2^i}$.

Now, suppose ℓ does not always query an instance that halves the area of the version space. Then after some number, k, of queries, ℓ first chooses to query a point $\mathbf{x}_{k+1}$ that does not halve the current version space $\mathcal{V}_k$. Let $y_{k+1} \in \{-1, 1\}$ correspond to the labeling of $\mathbf{x}_{k+1}$ that will cause the larger half of the version space to be chosen.

Without loss of generality assume $Area(\mathcal{V}^-_k) > Area(\mathcal{V}^+_k)$ and so $y_{k+1} = -1$. Note that $Area(\mathcal{V}^-_k) + Area(\mathcal{V}^+_k) = \frac{S_r}{2^k}$, so we have that $Area(\mathcal{V}^-_k) > \frac{S_r}{2^{k+1}}$.

Now consider the conditional distribution P_0:

$$P_0(-1 \mid \mathbf{x}) = \begin{cases} \frac{1}{2} & \text{if } \mathbf{x} \neq \mathbf{x}_{k+1} \\ 1 & \text{if } \mathbf{x} = \mathbf{x}_{k+1} \end{cases}$$

Then under this distribution, $\forall i > k$,

$$E_{P_0}[Area(\mathcal{V}_i)] = \frac{1}{2^{i-k-1}} Area(\mathcal{V}^-_k) > \frac{S_r}{2^i}.$$

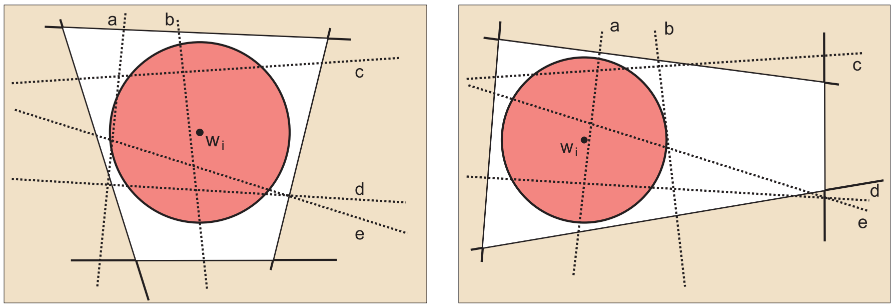
Hence, $\forall i > k$,

$$\sup_{P \in \mathcal{P}} E_P[Area(\mathcal{V}^*_i)] > \sup_{P \in \mathcal{P}} E_P[Area(\mathcal{V}_i)].$$

□

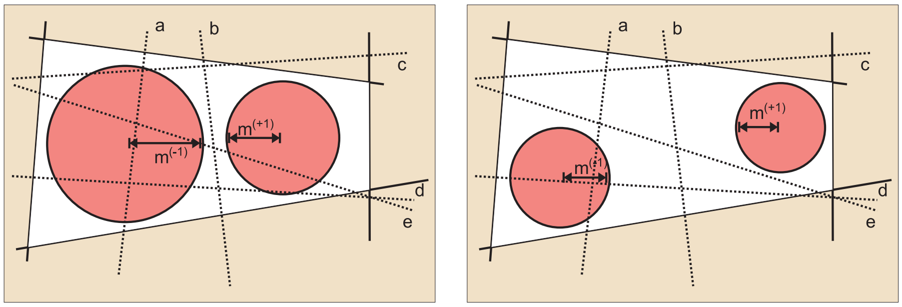
This lemma says that, for any given number of queries, ℓ^* minimizes the maximum expected size of the version space, where the maximum is taken over all conditional distributions of y given $\mathbf{x}$. In other words ℓ^* will be choosing queries that reduce the *minimax* loss of the model.

Seung et al. (Seung et al., 1992) also use an approach that queries points so as to attempt to reduce the size of the version space as much as possible. If one is willing to assume that there is a hypothesis lying within $\mathcal{H}$ that generates the data and that the generating hypothesis is deterministic and that the data are noise free, then strong generalization performance



(a) (b)

Figure 3.4: (a) Simple Margin will query $\mathbf{b}$. (b) Simple Margin will query $\mathbf{a}$.



(a) (b)

Figure 3.5: (a) MaxMin Margin will query $\mathbf{b}$. The two SVMs with margins m^- and m^+ for $\mathbf{b}$ are shown. (b) MaxRatio Margin will query $\mathbf{e}$. The two SVMs with margins m^- and m^+ for $\mathbf{e}$ are shown.

properties of an algorithm that halves version space can also be shown (Freund et al., 1997). For example one can show that the generalization error decreases exponentially with the number of queries.

3.6.3 Querying Algorithms

The previous discussion provides motivation for an approach where we query instances that split the current version space into two equal parts as much as possible. Given an unlabeled instance $\mathbf{x}$ from the pool, it is not practical to explicitly compute the sizes of the new version spaces $\mathcal{V}^-$ and $\mathcal{V}^+$ (i.e., the version spaces obtained when $\mathbf{x}$ is labeled as -1 and $+1$ respectively). We next present three ways of approximating this procedure.

- **Simple Margin.** Recall from Section 3.5 that, given some data $\{\mathbf{x}_1 \ldots \mathbf{x}_i\}$ and labels $\{y_1 \ldots y_i\}$, the SVM unit vector $\mathbf{w}_i$ obtained from this data is the center of the largest hypersphere that can fit inside the current version space $\mathcal{V}_i$. The position of $\mathbf{w}_i$ in the version space $\mathcal{V}_i$ clearly depends on the shape of the region $\mathcal{V}_i$; however, it is often approximately in the center of the version space. Now, we can test each of the unlabeled instances $\mathbf{x}$ in the pool to see how close their corresponding hyperplanes in $\mathcal{W}$ come to the centrally placed $\mathbf{w}_i$. The closer a hyperplane in $\mathcal{W}$ is to the point $\mathbf{w}_i$, the more centrally it is placed in the version space, and the more it bisects the version space. Thus we can pick the unlabeled instance in the pool whose hyperplane in $\mathcal{W}$ comes closest to the vector $\mathbf{w}_i$. For each unlabeled instance $\mathbf{x}$, the shortest distance between its hyperplane in $\mathcal{W}$ and the vector $\mathbf{w}_i$ is simply the distance between the feature vector $\Phi(\mathbf{x})$ and the hyperplane $\mathbf{w}_i$ in $\mathcal{F}$,. This distance is easily computed by: $|\mathbf{w}_i \cdot \Phi(\mathbf{x})|$. This results in the natural rule: learn an SVM on the existing labeled data and choose as the next instance to query the instance that comes closest to the hyperplane in $\mathcal{F}$.

  Figure 3.4(a) presents an illustration. In the stylized picture we have flattened out the surface of the unit weight vector hypersphere that appears in Figure 3.3(a). The white area is version space $\mathcal{V}_i$ which is bounded by solid lines corresponding to labeled instances. The five dotted lines represent unlabeled instances in the pool. The circle represents the largest radius hypersphere that can fit in the version space. Note that the edges of the circle do not touch the solid lines – just as the dark sphere in 3.3(b) does not meet the hyperplanes on the surface of the larger hypersphere (they meet somewhere under the surface). The instance $\mathbf{b}$ is closest to the SVM $\mathbf{w}_i$ and so we will choose to query $\mathbf{b}$.

  Two other studies (Campbell et al., 2000; Schohn & Cohn, 2000) independently developed our Simple method for active learning with support vector machines and provided different formal analyses. Campbell, Cristianini and Smola extend their analysis for the Simple method to cover the use of soft margin SVMs (Cortes & Vapnik, 1995) with linearly non-separable data. Schohn and Cohn note interesting behaviors of the active learning curves in the presence of outliers and both suggest the heuristic optimal stopping criterion of "stop querying when there are no more pool instances within the margin of the current hyperplane". Also, as we mentioned in Chapter 2, Lewis and Gale's (1994) uncertainty sampling is essentially the same as the Simple method.

- **MaxMin Margin.** The Simple Margin method can be a rather rough approximation. It relies on the assumption that the version space is fairly symmetric and that $\mathbf{w}_i$ is centrally placed. It has been demonstrated, both in theory and practice, that these assumptions can fail significantly (Herbrich et al., 1999). Indeed, if we are not careful we may actually query an instance whose hyperplane does not even intersect the version space. The MaxMin approximation is designed to somewhat overcome these problems. Given some data $\{\mathbf{x}_1 \ldots \mathbf{x}_i\}$ and labels $\{y_1 \ldots y_i\}$ the SVM unit vector $\mathbf{w}_i$ is the center of the largest hypersphere that can fit inside the current version space $\mathcal{V}_i$ and the radius m_i of the hypersphere is proportional[5] to the size of the margin of $\mathbf{w}_i$. We can use the radius m_i as an indication of the size of the version space (Vapnik, 1998). Suppose we have a candidate unlabeled instance $\mathbf{x}$ in the pool. We can estimate the relative size of the resulting version space $\mathcal{V}^-$ by labeling $\mathbf{x}$ as -1, finding the SVM obtained from adding $\mathbf{x}$ to our labeled training data and looking at the size of its margin m^-. We can perform a similar calculation for $\mathcal{V}^+$ by relabeling $\mathbf{x}$ as class $+1$ and finding the resulting SVM to obtain margin m^+.

  Since we want an equal split of the version space, we wish *Area*$(\mathcal{V}^-)$ and *Area*$(\mathcal{V}^+)$ to be similar. Now, consider $\min(\mathit{Area}(\mathcal{V}^-), \mathit{Area}(\mathcal{V}^+))$. It will be small if *Area*$(\mathcal{V}^-)$ and *Area*$(\mathcal{V}^+)$ are very different. Thus we will consider $\min(m^-, m^+)$ as an approximation and we will choose to query the $\mathbf{x}$ for which this quantity is largest. Hence, the MaxMin query algorithm is as follows: for each unlabeled instance $\mathbf{x}$ compute the margins m^- and m^+ of the SVMs obtained when we label $\mathbf{x}$ as -1 and $+1$ respectively; then choose to query the unlabeled instance for which the quantity $\min(m^-, m^+)$ is greatest.

  Figures 3.4(b) and 3.5(a) show an example comparing the Simple Margin and MaxMin

[5] To ease notation, without loss of generality we shall assume the the constant of proportionality is 1, i.e., the radius is equal to the margin.

Margin methods.

- **Ratio Margin.** This method is similar in spirit to the MaxMin Margin method. We use m^- and m^+ as indications of the sizes of $\mathcal{V}^-$ and $\mathcal{V}^+$. However, we shall try to take into account the fact that the current version space $\mathcal{V}_i$ may be quite elongated and for some $\mathbf{x}$ in the pool *both* m^- and m^+ may be small simply because of the shape of version space. Thus we will instead look at the *relative* sizes of m^- and m^+ and choose to query the $\mathbf{x}$ for which $\min(\frac{m^-}{m^+}, \frac{m^+}{m^-})$ is largest (see Figure 3.5(b)).

The above three methods are approximations to the querying component that always halves version space. After performing some number of queries we then return a classifier by learning a SVM with the labeled instances. The Simple method is significantly less computationally intensive than the other two methods since it needs to learn only one SVM per querying round, while the MaxRatio and MaxMin methods need to learn two SVMs for each pool instance during each querying round. Notice that we are not forced to stay with just one of these querying methods for all of our rounds of queries. For computational reasons, it may be beneficial to swap between the different methods after a number of queries have been asked: we call this type of querying method a Hybrid method.

We now address the assumption of having training feature vectors with constant moduli. The notions of a version space and of the size of version space still hold without the assumption. Furthermore, the margin of an SVM can be used as an indication of a version space size irrespective of whether the feature vectors have constant moduli (see (Vapnik, 1998) for further details). Thus the explanation for the MaxMin and MaxRatio methods still holds even without the constraint on the modulus of the training feature vectors.

The constant moduli assumption was necessary for our geometric view of version space to hold. The Simple method can still be used when the training feature vectors do not have constant modulus, but the motivating explanation no longer holds since the SVM can no longer be viewed as the center of the largest allowable sphere. However, for the Simple method, alternative motivations have recently been proposed by Campbell, Cristianini and Smola (2000) that do not require the constraint on the modulus.

For inductive learning, after performing some number of queries we then return a classifier by learning a SVM with the labeled instances. For transductive learning, after querying some number of instances we then return a classifier by learning a Transductive SVM with the labeled *and* unlabeled instances.

3.7 Comment on Multiclass Classification

A number of scenarios are inherently multiclass classification problems. For example, detecting which of several topics a particular document or image is about. Furthermore, there are two different types of multiclass settings. One multiclass setting is the *overlapping* classes setting where each data instance can belong to multiple classes at the same time (for example, a news article could belong to multiple different topics). The second type of multiclass setting is the *non-overlapping* , or *mutually exclusive* setting where each data instance belongs to exactly one of several classes.

A basic SVM is a binary classifier. SVMs can be easily extended to the overlapping multiclass setting by using the *one-vs-all* technique. For an k-class problem we learn k classifiers $f_1, \ldots, f_k$ where classifier f_i determines if an instance is in class i or not.

There are a number of techniques for extending SVMs to the more complicated mutually-exculsive multiclass case (Vapnik, 1998; Platt et al., 2000; Friedman, 1996). In this scenario the *one-vs-all* technique is one of the best performing and more common, albeit perhaps not the most computationally efficient, strategies. A difficulty arises because the outputs of the k different SVMs are uncalibrated reals values.[6] For example, it could be the case that $f_1(\mathbf{x}) = 2$ means that f_1 is very confident about $\mathbf{x}$'s label whereas $f_2(\mathbf{x}) = 2$ may mean that f_2 is only marginally confident about $\mathbf{x}$'s label. So, for the specific purpose of measuring an SVM's confidence in its prediction relative to other SVMs' predictions, the output of an SVM is uncalibrated. There have been a number of studies (Hastie & Tibshirani, 1998; Vapnik, 1998; Platt, 1999; Sollich, 1999) that explore ways of transforming each SVM's output into a calibrated conditional probability $P(i \mid \mathbf{x})$. Nevertheless, for mutually exclusive multiclass classification, uncalibrated values are typically used as measures of each f_i classifier's confidence, and the approximation of taking the predicted class

[6]Although still uncalibrated, the output of an SVM $f_i(\mathbf{x})$ is typically normalized by the margin, so that all of the support vectors are distance 1 from the hyperplane.

label to be:

$$y = \text{argmax}_i f_i(\mathbf{x}),$$

appears to work well in practice (Vapnik, 1998; Platt et al., 2000).

In both of these settings, we focus on the one-vs-all algorithm. Designing active learning algorithms for the alternative ways of performing multiclass classification is left as future work. With the one-vs-all approach we have k version spaces, one for each classifier. If we wish to use active learning we need to determine the model loss. In the binary classification task we used the area of the version space as our model loss. The area of the version space $\mathit{Area}(\mathcal{V})$ can be regarded as being proportional to the probability that a hypothesis chosen at random will correctly classify the current training data. Extending this notion to the multiclass case, our hypothesis is now a set of i hyperplanes and if we sample a hypothesis uniformly at random, the probability that we will have a hypothesis that correctly labels every point in our training set is proportional to:

$$\prod_i \mathit{Area}(\mathcal{V}^{(i)}). \tag{3.3}$$

Thus, perhaps one possible measure of model loss is the product of the version space areas. Fig. 3.6 shows why, intuitively, this measure of model loss is better than, say, the sum of areas. Class 3 is easily separated from the other two classes and so the version space of f_3 is much larger than that of f_1 and f_2. Querying points between classes 1 and 2 would intuitively be most useful since they will narrow down where f_1 and f_2 should lie. The product of version spaces criterion will query these points since they tend to halve the version spaces for f_1 and f_2. The sum of version spaces loss criterion will be distracted by the unlabeled points near class 3 since, although they do not halve the version space of 3, knowing their labels will remove a large area of the total sum of version space simply because f_3's version space is naturally large.

Eq. (3.3) is our **model loss** . Recall from Section 1.2 that we want to choose the unlabeled pool instance $\mathbf{x}$ that minimizes the maximum model loss:

$$\mathit{Loss}(\mathbf{x}) = \max_y \prod_i \mathit{Area}(\mathcal{V}^{(i)}_{\mathbf{x},y}), \tag{3.4}$$

Figure 3.6: Multiclass classification

where $\mathcal{V}^{(i)}_{\mathbf{x},y}$ is the version space after having asked $\mathbf{x}$ and received the label y. Note that, unlike in the binary classification case, this method no longer reduces to finding the pool instance bisects each of the k version spaces.

Now, evaluating the volumes of these versions spaces is intractable. To obtain an efficient algorithm we need to use an approximation to enable us to compute the model loss. The above definition of model loss allows us to extend the MaxRatio and MaxMin approximation methods to the multiclass case in the obvious manner. Extending the Simple method is more subtle.

For the Simple method, recall that the margin is proportional to the radius of the largest sphere that we can embed in the version space. Thus, unlike in the task of measuring an SVM's confidence in its own prediction, here the quantity $f_i(\mathbf{x})$ (where we normalize the output so that support vectors are distance one from the hyperplane) is actually a calibrated approximation of the extent to which $\mathbf{x}$ splits the version space. It is calibrated for this purpose since the scale of each $f_i(\mathbf{x})$ distance is measured relative to the radius of the sphere for that f_i.

Given the SVMs learned on the current labeled data, $f_1, \ldots, f_k$, and a pool instance $\mathbf{x}$, we wish to approximate the quantities $Area(\mathcal{V}^{(i)}_{\mathbf{x},y})$ for each i and each possible label y. In Fig. 3.7 we have the version space for one of the f_is. Imagine we are looking at pool instance $\mathbf{x}$ and we are considering the case where $\mathbf{x}$ is labeled as class i. Thus we wish to approximately find the area of the region A^+.

Notice that if $f_i(\mathbf{x}) = 0$ then we are approximately halving the version space. If $f_i(\mathbf{x})$ is close to 1 then $\mathbf{x}$ is a hyperplane that nearly touches the edge of the sphere and, so the

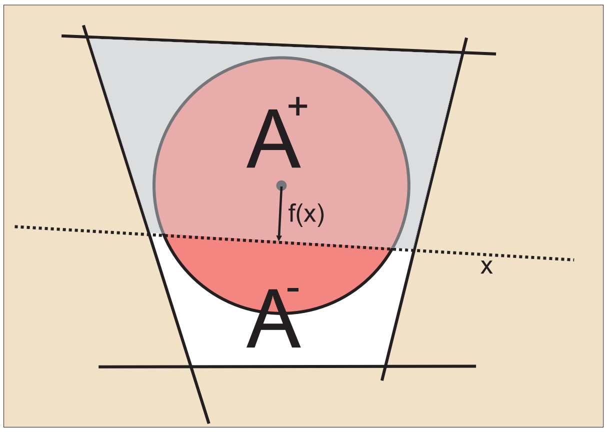


Figure 3.7: A version space.

area of the new version space will be close to the area of the original version space. If $f_i(\mathbf{x})$ is close to -1 then $\mathbf{x}$ is also a hyperplane that nearly touches the edge of the sphere, but this time the new version space lies on the side of the hyperplane furthest from the center of the sphere[7] and so the area of the new version space will be close to 0. In Fig. 3.7, $f_i(\mathbf{x}) = 0.5$ and the area of A^+ is approximately 0.75 of the old version space. When we look at the case where $\mathbf{x}$ is not labeled as class i, then we will wish to approximate the region A^-. In this case, $f_i(\mathbf{x}) = 0.5$ still, and the area of A^- is approximately 0.25 of the old version space.

These observations prompt the following mapping from $f_i(\mathbf{x})$ distances to sizes of version spaces:

- If the label y for pool instance $\mathbf{x}$ is class i, then:

$$Area(\mathcal{V}^{(i)}_{\mathbf{x},y}) \approx \left(\frac{f_i(\mathbf{x})+1}{2}\right) Area(\mathcal{V}^{(i)}). \tag{3.5}$$

- If the label y for pool instance $\mathbf{x}$ is not class i, then:

$$Area(\mathcal{V}^{(i)}_{\mathbf{x},y}) \approx \left(\frac{1-f_i(\mathbf{x})}{2}\right) Area(\mathcal{V}^{(i)}). \tag{3.6}$$

[7]One way to see this is because the center of the sphere is the current SVM, and it does not classify $\mathbf{x}$ correctly, so is cannot be in the new version space.

Notice that this approximation breaks down if $|f_i(\mathbf{x})| > 1$. However, we are performing a minimax computation, and so these outlier $\mathbf{x}$ instances will either be discarded at the "max" step if they cause $Area(\mathcal{V}^{(i)}_{\mathbf{x},y})$ to be too large and negative, or they will get rejected at the "min" step if they cause $Area(\mathcal{V}^{(i)}_{\mathbf{x},y})$ to be too large and positive.

Thus, by viewing the distance $f_i(\mathbf{x})$ as an approximate to how much the current version space is split, we get the following extension to the Simple algorithm:

Learn k classifiers, $f_1, \ldots, f_k$, one for each class.
For each unlabeled pool instance $\mathbf{x}$
  For each possible label y for $\mathbf{x}$
    For each classifier f_i
      Compute approximation to $Area(\mathcal{V}^{(i)}_{\mathbf{x},y})$ using either Eq. (3.5) or Eq. (3.6)
    End For
  End For
  $Loss(\mathbf{x}) = max_y \prod_i Area(\mathcal{V}^{(i)}_{\mathbf{x},y})$
End For
Query pool instance $\mathbf{x}$ for which $Loss(\mathbf{x})$ is lowest
Receive true label y'
Repeat

This multiclass Simple approximation is still very efficient: for each querying round we need to only learn k SVMs (one for each class), and we need to only sweep through the pool once for each classifier (to compute $f_i(\mathbf{x})$ for all $\mathbf{x}$).

Chapter 4

SVM Experiments

4.1 Text Classification Experiments

Text classification is the task of determining to which pre-defined topic a given text document belongs. Text classification has an important role to play, especially with the recent explosion of readily available text data. There have been many approaches to providing effective, automatic classification systems (Rocchio, 1971; Dumais et al., 1998). Furthermore, it is also a domain in which SVMs have shown notable success (Joachims, 1998; Dumais et al., 1998) and it is of interest to see whether active learning can offer further improvement over this already highly effective method.

For our empirical evaluation of the above methods we used two real-world text classification domains: the Reuters-21578 data set and the Newsgroups data set.

4.1.1 Text Classification

Rather than working directly with the raw text, learners typically work with features that are extracted from the document. The "bag-of-words" representation is particularly common: the ordering of the words within each document is ignored and the features are chosen to be particular words.

Sometimes some preprocessing of the documents is done. Common words on a stop list (such as "to", "it", "and") are ignored since they provide little discriminative information.

Also, words in the documents are stemmed so that, for example, "acquire", "acquiring", "acquired" all get mapped to the same stem (Porter, 1980). One other form of preprocessing is similar to stop list removal, but more extreme. One can perform feature selection and remove all words that are not "informative" with respect to the particular set of pre-defined topics (Yang & Pedersen, 1997). In our experiments we only consider stop word removal and stemming.

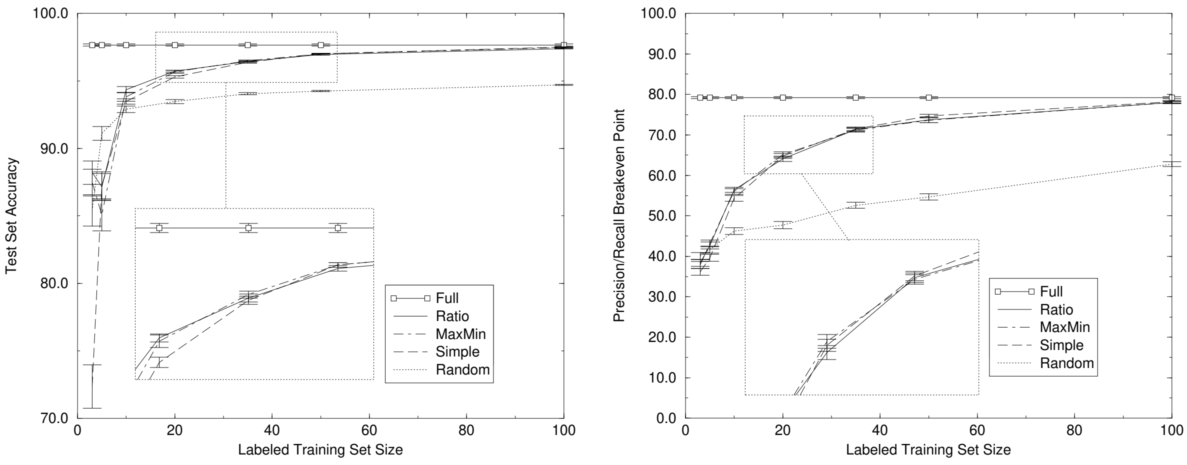
Given a set of n documents, a typical representation for documents is via *TFIDF* weighting (Salton & Buckley, 1988). There are a number of different variants of the TFIDF weighting scheme (Manning & Schütze, 1999). We describe one of the commonly used versions. Each document is represented by a fixed length unit vector $\mathbf{x}_i$ of dimension d. Each one of the d features, w_j, corresponds to a particular word (for example w_1 may correspond to the word "dog"). The vocabulary of d words is often chosen to be the words occurring in the entire set of (preprocessed) documents. Given a document, we construct the value for the j-th component of its corresponding vector $\mathbf{x}_i$ as follows: let $TF(w_j)$ be the number of times the word w_j occurs in the document. Let $IDF(w_j) = log(n/N_j)$ where N_j is the number of documents that contain the word w_j. Then give the j-th component of $\mathbf{x}_i$ a value of $TF(w_j).IDF(w_j)$. Intuitively, w_j is given a large value for a particular document if that word occurs many times in the document and very rarely in the other documents.

4.1.2 Reuters Data Collection Experiments

The Reuters-21578 data set[1] is a commonly used collection of newswire stories categorized into hand labeled topics. Each news story has been hand-labeled with some number of topic labels such as "corn", "wheat" and "corporate acquisitions". Note that some of the topics overlap and so some articles belong to more than one category. We used the 12902 articles from the "ModApte" split of the data and we considered the top ten most frequently occurring topics. We learned ten different binary classifiers, one to distinguish each topic. Each document was represented as a stemmed, TFIDF weighted word frequency vector.[2] Each vector had unit modulus. A stop list of common words was used and words occurring in less than three documents were also ignored. Using this representation, the document

[1]Obtained from www.research.att.com/~lewis.

[2]We used Rainbow (www.cs.cmu.edu/~mccallum/bow) for text processing.



(a) (b)

Figure 4.1: (a) Average test set accuracy over the ten most frequently occurring topics when using a pool size of 1000. (b) Average test set precision/recall breakeven point over the ten most frequently occurring topics when using a pool size of 1000.

vectors had around 10000 dimensions.

We first compared the three querying methods in the inductive learning setting. Our test set consisted of 3299 documents.

For each of the ten topics we performed the following. We created a pool of unlabeled data by sampling 1000 documents from the remaining data and removing their labels. We then randomly selected two documents in the pool to give as the initial labeled training set. One document was about the desired topic, and the other document was not about the topic. Thus we gave each learner 998 unlabeled documents and 2 labeled documents. After a fixed number of queries we asked each learner to return a classifier (an SVM with a polynomial kernel of degree one[3] learned on the labeled training documents). We then tested the classifier on the independent test set.

The above procedure was repeated thirty times for each topic and the results were averaged. We considered the Simple Margin, MaxMin Margin and MaxRatio Margin querying methods as well as a Random Sample method. The Random Sample method simply randomly chooses the next query point from the unlabeled pool. This last method reflects what

[3]For SVM and transductive SVM learning we used T. Joachims' SVMlight: ais.gmd.de/˜thorsten/svm_light/.

Table 4.1: Average test set accuracy over the top 10 most frequently occurring topics (most frequent topic first) when trained with ten labeled documents. Boldface indicates first place.

Topic	Simple	MaxMin	MaxRatio	Equivalent Random size
Earn	86.39 ± 1.65	87.75 ± 1.40	$\mathbf{90.24 \pm 2.31}$	34
Acq	77.04 ± 1.17	77.08 ± 2.00	$\mathbf{80.42 \pm 1.50}$	> 100
Money-fx	93.82 ± 0.35	94.80 ± 0.14	$\mathbf{94.83 \pm 0.13}$	50
Grain	95.53 ± 0.09	95.29 ± 0.38	$\mathbf{95.55 \pm 1.22}$	13
Crude	95.26 ± 0.38	95.26 ± 0.15	$\mathbf{95.35 \pm 0.21}$	> 100
Trade	96.31 ± 0.28	$\mathbf{96.64 \pm 0.10}$	96.60 ± 0.15	> 100
Interest	96.15 ± 0.21	$\mathbf{96.55 \pm 0.09}$	96.43 ± 0.09	> 100
Ship	97.75 ± 0.11	$\mathbf{97.81 \pm 0.09}$	97.66 ± 0.12	> 100
Wheat	98.10 ± 0.24	$\mathbf{98.48 \pm 0.09}$	98.13 ± 0.20	> 100
Corn	98.31 ± 0.19	$\mathbf{98.56 \pm 0.05}$	98.30 ± 0.19	15

happens in the regular passive learning setting – the training set is a random sampling of the data.

To measure performance we used two metrics: test set classification error and, to stay compatible with previous Reuters corpus results, the *precision/recall breakeven point* (Joachims, 1998). *Precision* is the percentage of documents a classifier labels as **relevant** that are truly labeled as **relevant**.[4] *Recall* is the percentage of truly relevant documents that are labeled as **relevant** by the classifier. By altering the decision threshold on the SVM we can trade precision for recall and can obtain a precision/recall curve for the test set. The precision/recall breakeven point is a one-number summary of this graph: it is the point at which precision equals recall.

Figures 4.1(a) and 4.1(b) present the average test set accuracy and precision/recall breakeven points over the ten topics as we vary the number of queries permitted. The horizontal line is the performance level achieved when the SVM is trained on all 1000 labeled documents comprising the pool. Over the Reuters corpus, the three active learning methods perform almost identically with little notable difference to distinguish between them. All three methods also appreciably outperforms random sampling. Tables 4.1 and 4.2 show the test set accuracy and breakeven performance of the active methods after they have asked

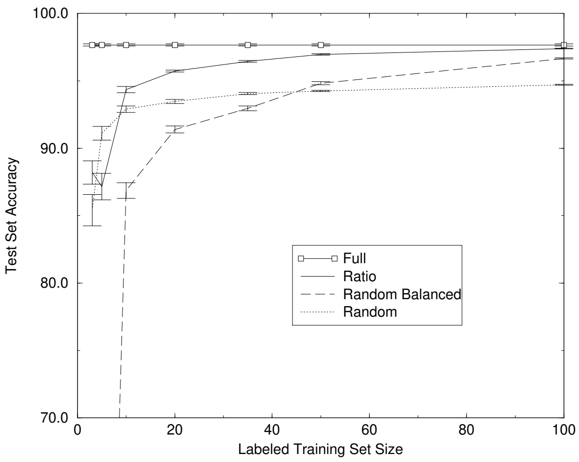
[4]For example, if our goal is to detect documents about corporate acquisitions, then articles about corporate acquisitions would be truly labeled as **relevant** and every other document would have a true label of **irrelevant**.

Table 4.2: Average test set precision/recall breakeven point over the top ten most frequently occurring topics (most frequent topic first) when trained with ten labeled documents. Boldface indicates first place.

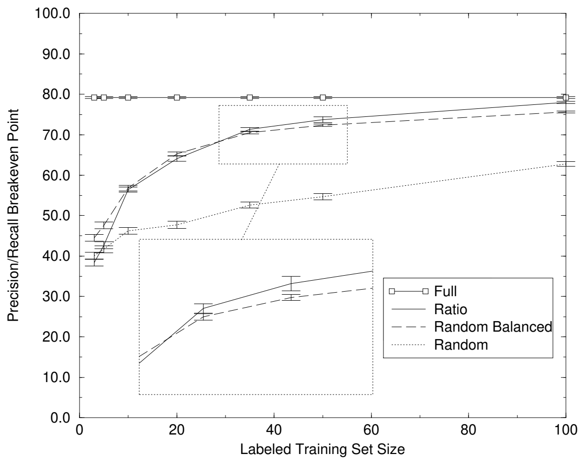
Topic	Simple	MaxMin	MaxRatio	Equivalent Random size
Earn	86.05 ± 0.61	89.03 ± 0.53	$\mathbf{88.95 \pm 0.74}$	12
Acq	54.14 ± 1.31	56.43 ± 1.40	$\mathbf{57.25 \pm 1.61}$	12
Money-fx	35.62 ± 2.34	$\mathbf{38.83 \pm 2.78}$	38.27 ± 2.44	52
Grain	50.25 ± 2.72	58.19 ± 2.04	$\mathbf{60.34 \pm 1.61}$	51
Crude	58.22 ± 3.15	55.52 ± 2.42	$\mathbf{58.41 \pm 2.39}$	55
Trade	$\mathbf{50.71 \pm 2.61}$	48.78 ± 2.61	50.57 ± 1.95	85
Interest	40.61 ± 2.42	$\mathbf{45.95 \pm 2.61}$	43.71 ± 2.07	60
Ship	$\mathbf{53.93 \pm 2.63}$	52.73 ± 2.95	53.75 ± 2.85	> 100
Wheat	64.13 ± 2.10	$\mathbf{66.71 \pm 1.65}$	66.57 ± 1.37	> 100
Corn	$\mathbf{49.52 \pm 2.12}$	48.04 ± 2.01	46.25 ± 2.18	> 100

for just eight labeled instances (so, together with the initial two random instances, they have seen ten labeled instances). The tables demonstrate that the three active methods perform similarly on this data set after eight queries, with the MaxMin and MaxRatio methods showing a very slight edge in performance. The last columns in each table are of more interest. They show approximately how many instances would be needed if we were to use Random to achieve the same level of performance as the MaxRatio active learning method. In this instance, passive learning on average requires over six times as much data to achieve comparable levels of performance as the active learning methods. The tables indicate that active learning provides more benefit with the infrequent classes, particularly when measuring performance by the precision/recall breakeven point. This last observation has also been noted before in previous empirical tests (McCallum & Nigam, 1998).

We noticed that approximately half of the queries that the active learning methods asked tended to turn out to be positively labeled, regardless of the true overall proportion of positive instances in the domain. We investigated whether the gains that the active learning methods had over regular Random sampling were due to this biased sampling. We created a new querying method called BalancedRandom which would randomly sample an equal number of positive and negative instances from the pool. Obviously in practice the ability to randomly sample an equal number of positive and negative instances without having to label an entire pool of instances first may or may not be reasonable depending upon the



(a)

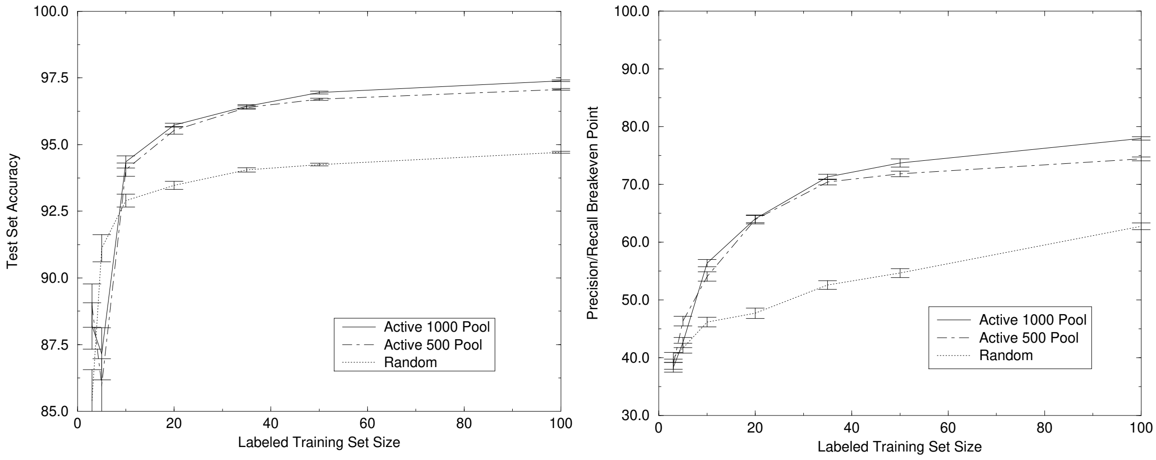


(b)

Figure 4.2: (a) Average test set accuracy over the ten most frequently occurring topics when using a pool size of 1000. (b) Average test set precision/recall breakeven point over the ten most frequently occurring topics when using a pool size of 1000.

domain in question. Figures 4.2(a) and 4.2(b) show the average accuracy and breakeven point of the BalancedRandom method compared with the MaxRatio active method and regular Random method on the Reuters dataset with a pool of 1000 unlabeled instances. The MaxRatio and Random curves are the same as those shown in Figures 4.1(a) and 4.1(b). The MaxMin and Simple curves are omitted to ease legibility. The BalancedRandom method has a much better precision/recall breakeven performance than the regular Random method, although it is still matched and then significantly outperformed by the active method. For classification accuracy, the BalancedRandom method initially has extremely poor performance (less than 50% which is even worse than pure random guessing) and is always consistently and significantly outperformed by the active method. This behavior indicates that the performance gains of the active methods are not merely due to their ability to bias the class of the instances they query. The active methods are choosing special targeted instances and approximately half of these instances happen to have positive labels.

Figures 4.3(a) and 4.3(b) show the average accuracy and breakeven point of the MaxRatio method with two different pool sizes. Clearly the Random sampling method's performance will not be affected by the pool size. However, the graphs indicate that increasing the pool of unlabeled data will improve both the accuracy and breakeven performance of active



(a) (b)

Figure 4.3: (a) Average test set accuracy over the ten most frequently occurring topics when using a pool sizes of 500 and 1000. (b) Average breakeven point over the ten most frequently occurring topics when using a pool sizes of 500 and 1000.

learning. This behavior is quite intuitive since a good active method should be able to take advantage of a larger pool of potential queries and ask more targeted questions.

We also investigated active learning in a transductive setting. Here we queried the points as usual except now each method (Simple and Random) returned a transductive SVM trained on both the labeled and remaining unlabeled data in the pool. The breakeven point for a TSVM was computed by gradually altering the number of unlabeled instances that we wished the TSVM to label as positive. This approach involves re-learning the TSVM multiple times and was computationally intensive. Since our setting was transduction, the performance of each classifier was measured on the pool of data rather than a separate test set. This experiment reflects the relevance feedback transductive inference example presented in the introduction.

Figure 4.4 shows that using a TSVM provides a slight advantage over a regular SVM in both querying methods (Random and Simple) when comparing breakeven points. However, the graph also shows that active learning provides notably more benefit than transduction. Indeed, using a TSVM with a Random querying method needs over 100 queries to achieve

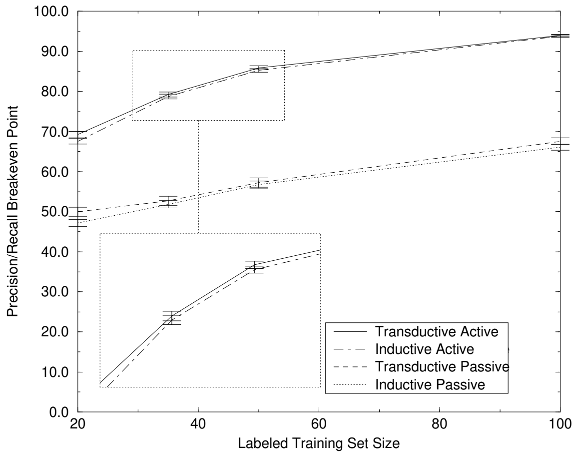


Figure 4.4: Average pool set precision/recall breakeven point over the ten most frequently occurring topics when using a pool size of 1000.

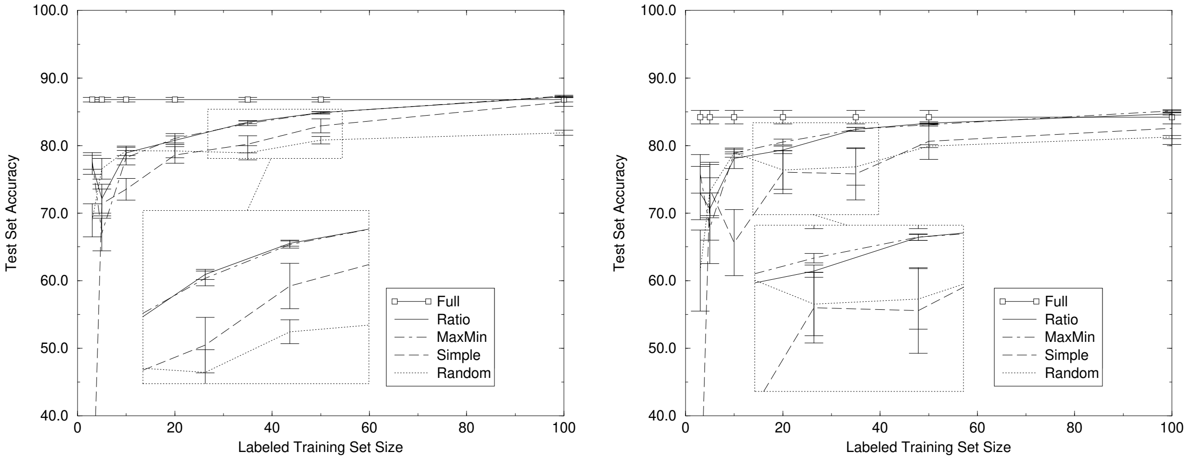
the same breakeven performance as a regular SVM with a Simple method that has only seen 20 labeled instances.

4.1.3 Newsgroups Data Collection Experiments

Our second data collection was Ken Lang's Newsgroups collection.[5] We used the five `comp.*` groups, discarding the Usenet headers and subject lines. We processed the text documents exactly as before resulting in vectors of around 10000 dimensions.

We placed half of the 5000 documents aside to use as an independent test set, and repeatedly, randomly chose a pool of 500 documents from the remaining instances. We performed twenty runs for each of the five topics and averaged the results. We used test set accuracy to measure performance. Figure 4.5(a) contains the learning curve (averaged over all of the results for the five `comp.*` topics) for the three active learning methods and Random sampling. Again, the horizontal line indicates the performance of an SVM that has been trained on the entire pool. There is no appreciable difference between the MaxMin and MaxRatio methods but, in two of the five newsgroups (`comp.sys.ibm.pc.hardware`

[5]Obtained from www.cs.cmu.edu/~textlearning.



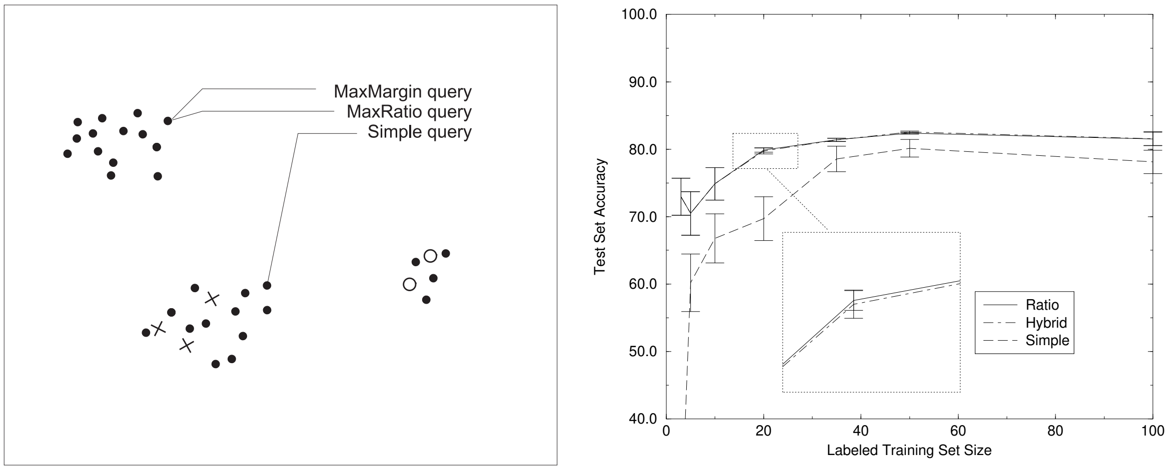
(a) (b)

Figure 4.5: (a) Average test set accuracy over the five `comp.*` topics when using a pool size of 500. (b) Average test set accuracy for `comp.sys.ibm.pc.hardware` with a 500 pool size.

and `comp.os.ms-windows.misc`) the Simple active learning method performs notably worse than the MaxMin and MaxRatio methods. Figure 4.5(b) shows the average learning curve for the `comp.sys.ibm.pc.hardware` topic. In around ten to fifteen per cent of the runs for both of the two newsgroups the Simple method was misled and performed extremely poorly (for instance, achieving only 25% accuracy even with fifty training instances, which is worse than random guessing!). This experiment indicates that the Simple querying method may be more unstable than the other two methods. Lewis and Gale (1994) also noted that the performance of the uncertainty sampling method (which is our Simple method) can be variable, performing quite poorly on occasions.

One reason for this instability could be that the Simple method tends not to explore the feature space as aggressively as the other active methods, and can end up ignoring entire clusters of unlabeled instances. In Figure 4.6(a) the Simple method takes several queries before it even considers an instance in the unlabeled cluster while both the MaxMin and MaxRatio query a point in the unlabeled cluster immediately.

While MaxMin and MaxRatio appear more stable they are much more computationally intensive. With a large pool of s instances, they require around $2s$ SVMs to be learned for each query. Most of the computational cost is incurred when the number of queries that have already been asked is large. The reason is that the cost of training an SVM



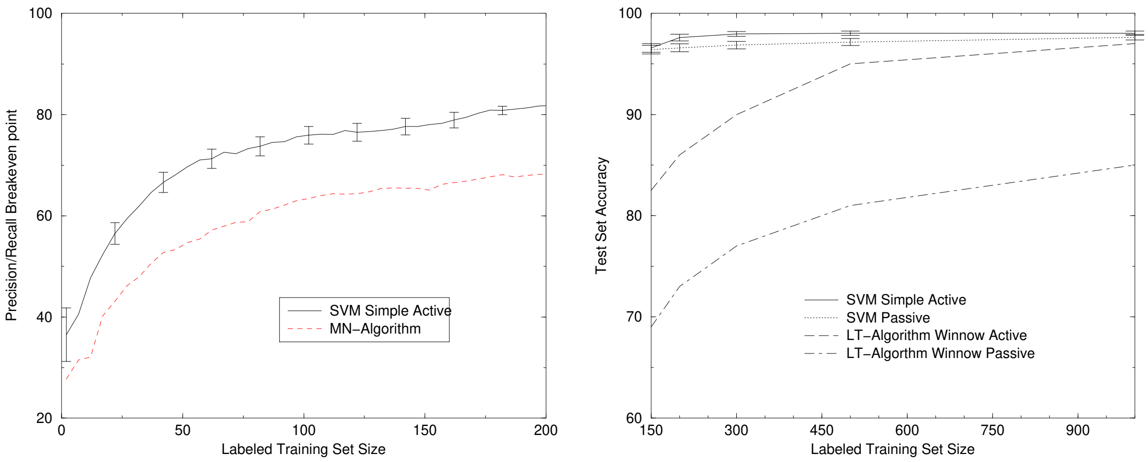
(a) (b)

Figure 4.6: (a) A simple example of querying unlabeled clusters. (b) Macro average test set accuracy for `comp.os.ms-windows.misc` and `comp.sys.ibm.pc.hardware` where Hybrid uses the MaxRatio method for the first ten queries and Simple for the rest.

grows polynomially with the size of the labeled training set and so now training each SVM is costly (taking around a minute to generate the 50th query on a Sun Ultra 60 450Mhz workstation with a pool of 1000 documents). However, when the quantity of labeled data is small, even with a large pool size, MaxMin and MaxRatio are fairly fast (taking a few seconds per query) since now training each SVM is fairly cheap. Interestingly, it is in the first ten queries that the Simple method seems to suffer the most through its lack of aggressive exploration. This observation prompts us to consider a Hybrid method. We can use MaxMin or MaxRatio for the first few queries and then use the Simple method for the rest. Experiments with the Hybrid method show that it maintains the stability of the MaxMin and MaxRatio methods while allowing the scalability of the Simple method. Figure 4.6(b) compares the Hybrid method with the MaxRatio and Simple methods on the two newsgroups for which the Simple method performed poorly. The test set accuracy of the Hybrid method is virtually identical to that of the MaxRatio method while the Hybrid method's run time was about the same as the Simple method, as indicated by Table 4.3.

Table 4.3: Typical run times in seconds for the Active methods on the Newsgroups dataset

Query	Simple	MaxMin	MaxRatio	Hybrid
1	0.008	3.7	3.7	3.7
5	0.018	4.1	5.2	5.2
10	0.025	12.5	8.5	8.5
20	0.045	13.6	19.9	0.045
30	0.068	22.5	23.9	0.073
50	0.110	23.2	23.3	0.115
100	0.188	42.8	43.2	0.2



(a) (b)

Figure 4.7: (a) Average breakeven point performance over the Corn, Trade and Acq Reuters-21578 categories. (b) Average test set accuracy over the top ten Reuters-21578 categories.

4.1.4 Comparision with Other Active Learning Systems

There have been a number of alternative approaches to active learning for text classification. McCallum and Nigam used a general purpose active learning algorithm called *Query by Committee* (Seung et al., 1992; Freund et al., 1997) together with a *naive Bayes* (Duda & Hart, 1973) model. They also used the *Expectation Maximization (EM)* (Dempster et al., 1977) algorithm to take further advantage of the unlabeled instances. We re-created McCallum and Nigam's (1998) experimental setup on the Reuters-21578 corpus and compared the reported results from their algorithm (*MN*-algorithm hereafter) with ours. In line with their experimental setup, queries were asked five at a time, and this was achieved by picking the five instances closest to the current hyperplane. Figure 4.7(a) compares McCallum and Nigam's reported results with ours. The graph indicates that the Active SVM performance is significantly better than the *MN*-algorithm.

An alternative committee approach to Query by Committee was explored by Liere and Tadepalli (1997, 2000). Although their algorithm (*LT*-algorithm hereafter) lacks the theoretical justifications of the Query by Committee algorithm, they successfully used their committee based active learning method with Winnow classifiers in the text domain. Figure 4.7(b) was produced by emulating their experimental setup on the Reuters-21578 data set and it compares their reported results with ours. Their algorithm does not require a positive and negative instance to seed their classifier. Rather than seeding our Active SVM with a positive and negative instance (which would give the Active SVM an unfair advantage) the Active SVM randomly sampled 150 documents for its first 150 queries. This process virtually guaranteed that the training set contained at least one positive instance. The Active SVM then proceeded to query instances actively using the Simple method. Despite the very naive initialization policy for the Active SVM, the graph shows that the Active SVM accuracy is significantly better than that of the *LT*-algorithm.

SVM active learning outperforms the other systems for two main reasons. First, SVMs are already a highly competative method for text classification (Joachims, 1998; Dumais et al., 1998). Second, our active method boosts the SVM performance so as to maintain the performance advantage over other classifiers when they use their own active learning methods.

4.2 Image Retrieval Experiments

4.2.1 Introduction

One key design task, when constructing image databases, is the creation of an effective browsing and searching component. While it is sometimes possible to arrange images within an image database by creating a hierarchy, or by hand-labeling each image with descriptive words, it is often time-consuming, costly and subjective. Alternatively, requiring the end-user to specify an image query in terms of low level features (such as color and texture) is challenging to the end-user, because an image query is hard to articulate, and articulation can again be subjective.

Thus, there is a need for a way to allow a user to implicitly inform a database of his or her desired output or *query concept* . To address this requirement, *relevance feedback* can be used as a query refinement scheme to derive or learn a user's query concept. To solicit feedback, the refinement scheme displays a few image instances and the user labels each image as **relevant** or **irrelevant**. Based on the answers, another set of images from the database are brought up to the user for labeling. After some number of such querying rounds, the refinement scheme returns a number of items in the database that it believes will be of interest to the user.

A query refinement scheme that uses relevance feedback can be regarded as a pool-based active learning task. In pool-based active learning the learner has access to a pool of unlabeled data and can request the user's label for a certain number of instances in the pool. In the image retrieval domain, the unlabeled pool would be the entire database of images. An instance would be an image, and the two possible labelings of an image would be **relevant** and **not relevant**. The goal for the learner is to learn the user's *query concept*. In other words, the goal is to give a label to each image within the database such that for any image, the learner's labeling and the user's labeling will agree.

In general, and for the image retrieval task in particular, such a learner must meet two critical design goals. First, the learner must learn target concepts accurately, with only a small number of labeled instances. Second, the learner must ask queries quickly since most users do not wish to wait around.

4.2.2 The $\mathsf{SVM}_{\mathit{Active}}$ Relevance Feedback Algorithm for Image Retrieval

Given the interactive nature of image retrieval, we used the Simple querying method only. The other querying methods proved too computationally costly in this domain.

For the image retrieval domain, we also have a need for performing multiple queries at the same time. It is not practical to present one image at a time for the user to label because the user is likely to quickly lose patience after a few rounds of querying. Hence, we would like to present the user with multiple images (say, twenty) at each round of querying. Thus, for each round, the active learner has to choose not just one image to be labeled but twenty. Theoretically it would be possible to consider the size of the resulting version spaces for each possible labeling of each possible set of twenty queries but clearly this approach is impractical. Instead our system takes the simple approach of choosing the queries to be the twenty images closest to its separating hyperplane.

In our text experiments, we noted that the Simple querying algorithm used by SVM_{Active} can sometimes be unstable during the first few queries. To address this issue, SVM_{Active} always randomly chooses twenty images for the first relevance feedback round. Then it uses the Simple active querying method on the second and subsequent rounds.

To summarize, our SVM_{Active} system performs the following:

1. Initialize with one **relvant** and one **irrelevant** image.
2. For the first round of querying, ask the user to label twenty randomly selected images.
3. Learn an SVM on the current labeled data
4. Ask the user to label the twenty pool images closest to the SVM boundary.
5. Perform additional querying rounds by going to step 3.

After the relevance feedback rounds have been performed SVM_{Active} retrieves the top-k most relevant images:

1. Learn a final SVM on the labeled data.
2. The final SVM boundary separates **relevant** images from **irrelevant** ones. Display the k **relevant** images that are farthest from the SVM boundary.

The follow section describes the features that we used for our SVM_{Active} image retrieval system.

4.2.3 Image Characterization

In order to be able to perform relevance feedback we first need to decide how to represent an image. We extract two key types of features from each image: its color and texture.

Filter Name	Resolution	Representation
Color Masks	Coarse	Appearance of culture colors
Color Spread	Coarse	Spatial concentration of a color
Color Elongation	Coarse	Shape of a color
Color Histograms	Medium	Distribution of colors
Color Average	Medium	Similarity comparison within the same culture color
Color Variance	Fine	Similarity comparison within the same culture color

Table 4.4: Multi-resolution Color Features.

Clearly a great deal of additional information is lost when using these simple types of features. However, just as document classifiers that ignore word ordering are still very effective, the image retrieval retrieval task can be effectively performed just by using these two simple types of features.

Color

Although the wavelength of visible light ranges from 400 nanometers to 700 nanometers, research (Goldstein, 1999) shows that the colors that can be named by all cultures are generally limited to eleven. In addition to *black* and *white*, the discernible colors are *red*, *yellow*, *green*, *blue*, *brown*, *purple*, *pink*, *orange* and *gray*.

We first divide color into 12 color bins including 11 bins for culture colors and one bin for outliers (Hua et al., 1999). At the coarsest resolution, we characterize color using a color mask of 12 bits. To record color information at finer resolutions, we record eight additional features for each color. These eight features are color histograms (the percentage of that color in the image), color means in the hue (H), saturation (S) and value (V) channels, color variances in the H, S and V channels, and two shape characteristics: elongation and spreadness. For each color bin, the color means indicate the average shade of that particular color. The color variances characterize the number of different shades of that color that are present in the image. For example, in a forest image we would expect a large variance for the H, S and V channels in the green color bin. Color spreadness is given by the second moment of that color's pixels' locations. Spreadness characterizes how that color scatters within the image (Leu, 1991). Color elongation characterizes the shape of a color and, for efficiency, it is compute simply by taking the ratio of the variances of that color's pixels' locations in the vertical and horizontal directions. Table 4.4 summarizes color features in coarse, medium and fine resolutions.

Texture

Texture is an important cue for image analysis. Studies (Manjunath et al., 2001; Smith & Chang, 1996; Tamura et al., 1978; Ma & Zhang, 1998) have shown that characterizing texture features in terms of structuredness, orientation, and scale (coarseness) fits well with models of human perception. A wide variety of texture analysis methods have been proposed in the past. We choose a discrete wavelet transformation (DWT) using quadrature mirror filters(Smith & Chang, 1996) because of its computational efficiency.

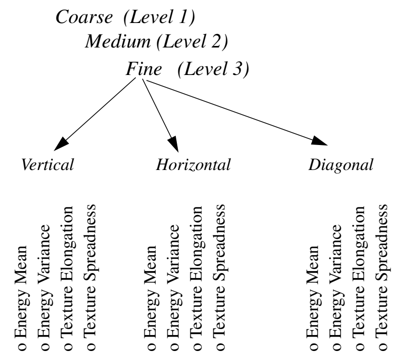


Figure 4.8: Multi-resolution texture features.

Each wavelet decomposition on a 2-D image produces four subimages: a $\frac{1}{2} \times \frac{1}{2}$ scaled-down version of the input image and its wavelets in three orientations: horizontal, vertical and diagonal. The energies of the horizontal, vertical and diagonal wavelet images capture the amount of fine texture present for those particular orientations in the original image. Now, applying the wavelet transformation to the $\frac{1}{2} \times \frac{1}{2}$ scaled-down version of the original image produces another set of four subimages. This time, the energies of the horizontal, vertical and diagonal wavelet images capture the amount of medium texture present in the original image. Similarly, applying the wavelet to the $\frac{1}{4} \times \frac{1}{4}$ version of original image yields a measure for the amount of coarse texture. Thus, we obtain a total of nine texture combinations from subimages of three scales and three orientations.

Each of the wavelet images is similar to the result produced by using a standard edge dectection filter in that it maintains spatial information. For example. if there is a large degree of fine horizontal texture in the center of the original image (e.g, because the center of the image contains a tree trunk) then the will be a high degree of energy in the center of the corresponding wavelet image for horiziontal fine texture. Thus, we can also extract elongation and spreadness information from the nine wavelet images. Figure 4.8 summarizes texture features.

4.2.4 Experiments

For our empirical evaluation of our learning methods we used three real-world image datasets: a four-category, a ten-category, and a fifteen-category image dataset where each category consisted of 100 to 150 images. These image datasets were collected from Corel Image CDs and the Internet.

- *Four-category* set. The 602 images in this dataset belong to four categories – *architecture*, *flowers*, *landscape*, and *people*.
- *Ten-category* set. The 1277 images in this dataset belong to ten categories – *architecture*, *bears*, *clouds*, *flowers*, *landscape*, *people*, *objectionable images*, *tigers*, *tools*, and *waves*. In this set, a few categories were added to increase learning difficulty. The tiger category contains images of tigers on landscape and water backgrounds to confuse with the landscape category. The objectionable images can be confused with people wearing little clothing. Clouds and waves have substantial color similarity.
- *Fifteen-category* set. In addition to the ten categories in the above dataset, the total of 1920 images in this dataset includes *elephants*, *fabrics*, *fireworks*, *food*, and *texture*. We added elephants with landscape and water backgrounds to increase learning difficulty between landscape, tigers and elephants. We added colorful fabrics and food to interfere with flowers. Various texture images (e.g., skin, brick, grass, water, etc.) were added to raise learning difficulty for all categories.

To provide an objective measure of performance, we assumed that a query concept was an image category. The SVM_{Active} learner has no prior knowledge about image categories[6]. It treats each image as a 144-dimension vector described in Section 4.2.3. The goal of SVM_{Active} is to learn a given concept through a relevance feedback process. In this process, at each feedback round SVM_{Active} selects twenty images to ask the user to label as **relevant** or **irrelevant** with respect to the query concept. It then uses the labeled instances to successively refine the concept boundary. After the relevance feedback rounds have finished SVM_{Active} then retrieves the top-k most relevant images from the dataset based on the final concept it has learned.

Accuracy is then computed by looking at the fraction of the k returned result that belongs to the target image category. Notice that this is equivalent to computing the precision on the top-k images. This measure of performance appears to be the most appropriate for the image retrieval task – particularly since, in most cases, not all of the relevant images will be able to be displayed to the user on one screen. As in the case of web searching, we typically wish the first few screens of returned images to contain a high proportion of relevant images. We are less concerned that not every single instance that satisfies the query concept is displayed.

As with all SVM algorithms, SVM_{Active} requires at least one relevant and one irrelevant image to function. In practice a single relevant image could be provided by the user (e.g., via an upload to the system) or could be found by displaying a large number of randomly selected images to the user (where, perhaps, the image feature vectors are chosen to be mutually distant from each other so as to provide a wide coverage of the image space). In either case we assume that we start off with one randomly selected relevant image and one randomly selected irrelevant image.

SVM_{Active} Experiments

Figures 4.9(a-c) show the average top-k accuracy for the three different sizes of data sets. We considered the performance of SVM_{Active} after each round of relevance feedback. The

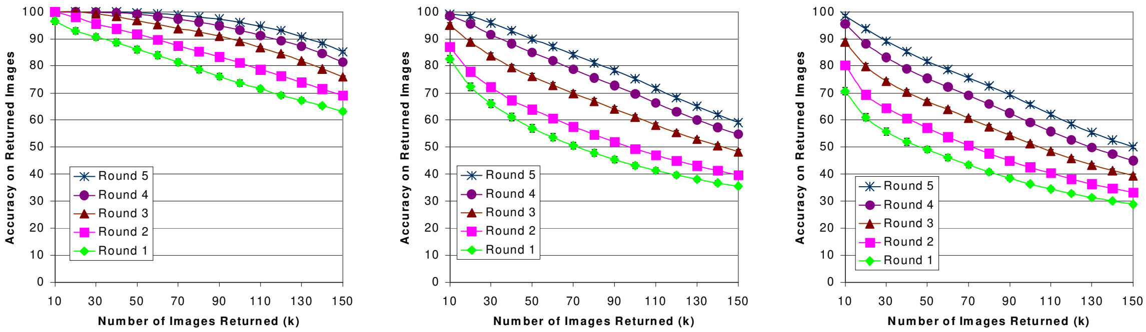
[6]Unlike some recently developed systems (Wang et al., 2000) that contain a semantic layer between image features and queries to assist query refinement, our system does not have an explicit semantic layer. We argue that having a layer can make a retrieval system restrictive. Rather, dynamically learning the semantics of a query concept is more flexible and hence makes the system more useful.

graphs indicate that performance clearly increases after each round. Also, the SVM_{Active} algorithm's performance degrades gracefully when the size and complexity of the database is increased – for example, after four rounds of relevance feedback it achieves an average of 100%, 95%, 88% accuracy on the top-20 results for the three different data sets respectively. It is also interesting to note that SVM_{Active} is not only good at retrieving just the top few images with high precision, but it also manages to sustain fairly high accuracy even when asked to return larger numbers of images. For example, after five rounds of querying it attains 99%, 84% and 76% accuracy on the top-70 results for the three different sizes of data sets respectively[7].

SVM_{Active} uses the Simple active querying method outlined in Section 3.6. We examined the effect that the active querying method had on performance. Figures 4.10(a) and 4.10(b) compare the active querying method with the regular passive method of sampling. The passive method chooses random images from the pool to be labeled. This method is the one that is typically used with SVMs since it creates a randomly selected data set. It is clear that the use of active learning is beneficial in the image retrieval domain. There is a significant increase in performance from using the active method and the boost in performance grows with the number of querying rounds.

SVM_{Active} displays 20 images per pool-querying round. There is a tradeoff between the number of images to be displayed in one round, and the number of querying rounds. The fewer images displayed per round, the lower the performance. However, with fewer images per round we may be able to conduct more rounds of querying and thus increase our performance. Figure 4.11 considers the effect of displaying different images per round. In Figures 4.11(a-b) we consider one of the topics in the four-category dataset. We start out by initializing with one relevant and one irrelevant image and then ask 20 randomly selected images. We then compare asking different numbers of images per round. Fig. 4.11(a) displays the top-100 accuracy for different numbers of images seen, and Fig. 4.11(b) displays the top-100 accuracy for different numbers of rounds. In Fig. 4.11(c) we consider the fifteen category dataset. We initialize with one relevant and one irrelevant image. Our first

[7]We note that, in general, the state-of-the-art performance levels of classifiers in the image domain is worse than in the text classification domain. This is because is harder to find meaningful image features. Thus the image features that are typically used are less informative about the topic of an image than the words features are about the topic of a document.

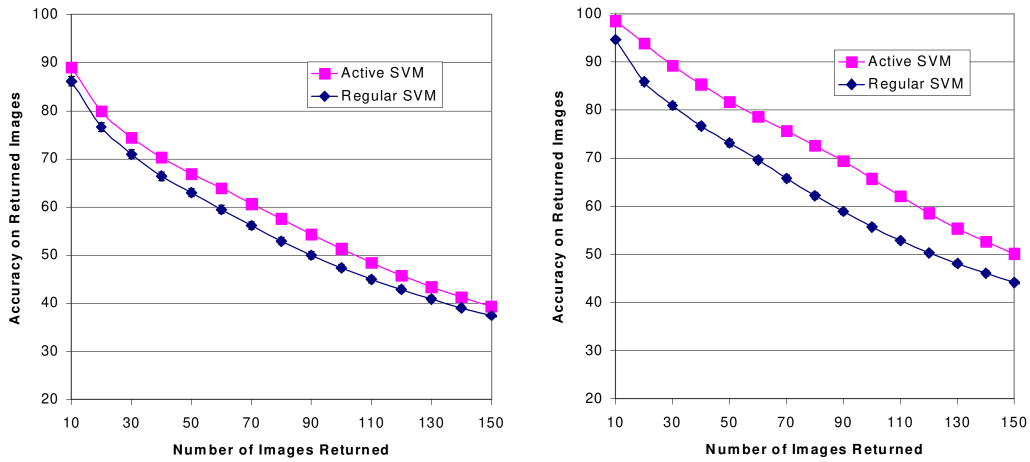


(a) (b) (c)

Figure 4.9: (a) Average top-k accuracy over the four-category dataset. (b) Average top-k accuracy over the ten-category dataset. (c) Average top-k accuracy over the fifteen-category dataset. Standard error bars are smaller than the curves' symbol size. Legend order reflects order of curves.

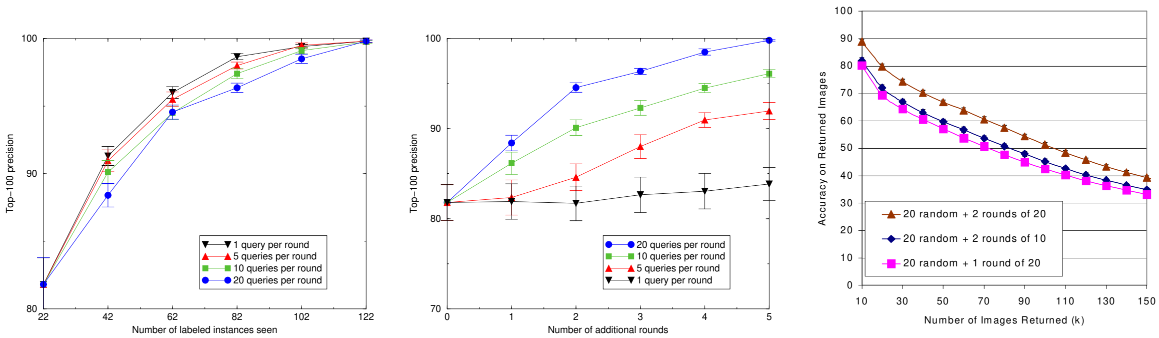
round consisted of displaying twenty random images and then, on the second and subsequent rounds of querying, active learning with 10 or 20 images is invoked. We notice that in all graphs there is indeed a little benefit to asking (20 random + two rounds of 10 images) over asking (20 random + one round of 20 images). This observation is unsurprising since the active learner has more control and freedom to adapt when asking two rounds of 10 images rather than one round of 20. What is interesting is that asking (20 random + two rounds of 20 images) is far better than asking (20 random + two rounds of 10 images). The increase in the cost to users of asking 20 images per round is often negligible since users can pick out relevant images easily. Furthermore, there is virtually no additional computational cost in calculating the 20 images to query over the 10 images to query. Thus, for this particular task, we believe that it is worthwhile to display around 20 images per screen and limit the number of querying rounds, rather than display fewer images per screen and use many more querying rounds.

We also investigated how performance altered when various aspects of the algorithm were changed. Table 4.5 shows how all three of the texture resolutions are important. Also, the performance of the SVM appears to be greatest when all of the texture resolutions are included (although in this case the difference is not statistically significant). Table 4.6 indicates how other SVM kernel functions perform on the image retrieval task compared to the radial basis function kernel. It appears that the radial basis function kernel is the most



(a) (b)

Figure 4.10: (a) Active and regular passive learning on the fifteen-category dataset after three rounds of querying. (b) Active and regular passive learning on the fifteen-category dataset after five rounds of querying. Standard error bars are smaller than the curves' symbol size. Legend order reflects order of curves.



(a) (b) (c)

Figure 4.11: (a) Top-100 precision of the landscape topic in the four-category dataset as we vary the number of examples seen. (b) Top-100 precision of the landscape topic in the four-category dataset as we vary the number of querying rounds. (c) Comparison between asking ten images per pool-query round and twenty images per pool-querying round on the fifteen-category dataset. Legend order reflects order of curves.

Texture features	Top-50 Accuracy
None	80.6 ± 2.3
Fine	85.9 ± 1.7
Medium	84.7 ± 1.6
Coarse	85.8 ± 1.3
All	86.3 ± 1.8

Table 4.5: Average top-50 accuracy over the four-category data set using a regular SVM trained on 30 images. Texture spatial features were omitted.

	Top-50	Top-100	Top-150
Degree 2 Polynomial	95.9 ± 0.4	86.1 ± 0.5	72.8 ± 0.4
Degree 4 Polynomial	92.7 ± 0.6	82.8 ± 0.6	69.0 ± 0.5
Radial Basis	$\mathbf{96.8 \pm 0.3}$	$\mathbf{89.1 \pm 0.4}$	$\mathbf{76.0 \pm 0.4}$

Table 4.6: Accuracy on four-category data set after three querying rounds using various kernels. Bold type indicates statistically significant results.

suitable for this feature space.

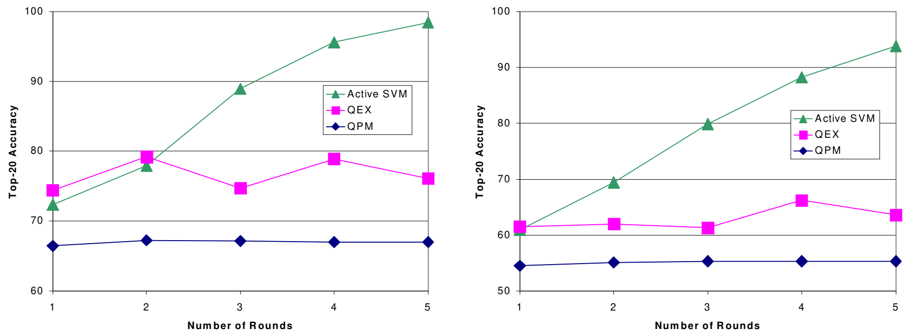
One other important aspect of any relevance feedback algorithm is the wall clock time that it takes to generate the next pool-queries. Relevance feedback is an interactive task, and if the algorithm takes too long then the user is likely to lose patience and be less satisfied with the experience. Table 4.7 shows that SVM$_{Active}$ averages about a second on a Sun Workstation to determine the 20 most informative images for the users to label. Retrieval of the 150 most relevant images takes an similar amount of time and computing the final SVM model never exceeds two seconds.

Scheme Comparison

Relevance feedback techniques proposed by the database and image retrieval communities also perform non-random sampling and are closely related to active learning. The study

Dataset	Dataset Size	round of 20 queries (secs)	Computing final SVM	Retrieving top 150 images
4 Cat	602	0.34 ± 0.00	0.5 ± 0.01	0.43 ± 0.02
10 Cat	1277	0.71 ± 0.01	1.03 ± 0.03	0.93 ± 0.03
15 Cat	1920	1.09 ± 0.02	1.74 ± 0.05	1.37 ± 0.04

Table 4.7: Average run times in seconds



(a) (b)

Figure 4.12: (a) Average top-k accuracy over the ten-category dataset. (b) Average top-k accuracy over the fifteen-category dataset.

of (Porkaew et al., 1999b) puts these relevance feedback approaches into two categories: *query reweighting/query point movement* and *query expansion*.

- *Query reweighting* and *query point movement* (QPM) (Ishikawa et al., 1998; Ortega et al., 1999; Porkaew et al., 1999a). Both query reweighting and query point movement use nearest-neighbor sampling: They return top ranked objects to be marked by the user and refine the query based on the feedback.

- *Query expansion* (QEX) (Porkaew et al., 1999b; Wu et al., 2000). The *query expansion* approach can be regarded as a multiple-instances sampling approach. The samples of the next round are selected from the neighborhood (not necessarily the nearest ones) of the positive-labeled instances of the previous round. The study of (Porkaew et al., 1999b) shows that query expansion achieves only a slim margin of improvement (about 10% in precision/recall) over query point movement.

We compared SVM_{Active} with these two traditional query refinement methods. In this experiment, each scheme returned the 20 most relevant images after up to five rounds of relevance feedback. To ensure that the comparison to SVM_{Active} was fair, we seeded both schemes with one randomly selected relevant image to generate the first round of images. On the ten-category image dataset, Figure 4.12(a) shows that SVM_{Active} achieves nearly 90% accuracy on the top-20 results after three rounds of relevance feedback, whereas the accuracies of both QPM and QEX never reach 80% and do not tend to improve significantly after just five querying rounds. On the fifteen-image category dataset, Figure 4.12(b) shows that SVM_{Active} outperforms the others by even wider margins. SVM_{Active} reaches 80% top-20 accuracy after three rounds and 94% after five rounds, whereas QPM and QEX cannot achieve 65% accuracy.

Traditional information retrieval schemes often require a large number of image instances to achieve any substantial refinement. By refining current relevant instances both QPM and QEX tend to be fairly localized in their exploration of the image space and hence rather slow in exploring the entire space. During the relevance feedback phase SVM_{Active} takes both the relevant and irrelevant images into account when choosing the next pool-queries. Furthermore, it chooses to ask the user to label images that it regards as most **informative** for learning the query concept, rather than those that have the most likelihood of being relevant. Thus it tends to explore the feature space more aggressively.

Figures 4.13 and 4.14 show an example run of the SVM_{Active} system. For this run, we are interested in obtaining architecture images. In Figure 4.13 we initialize the search by giving SVM_{Active} one relevant and one irrelevant image. We then have three feedback rounds. The images that SVM_{Active} asks us to label in these three feedback rounds are images that SVM_{Active} will find most informative to know about. For example, we see that it asks us to label a number of landscape images and other images with a blue or gray background with something in the foreground. The feedback rounds allow SVM_{Active} to narrow down the types of images that we like. When it comes to the retrieval phase (Figure 4.14) SVM_{Active} returns, with high precision, a large variety of different architecture images, ranging from old buildings to modern cityscapes.

4.3 Multiclass SVM Experiments

The previous two domains both involved binary classification: we were interested in distinguishing **relevant** instances from **irrelevant** ones. We now consider using the extension to the multiclass scenario discussed in Section 3.7.

Recall that, in the binary classification setting, our Simple method is essentinally the same as Lewis and Gale's uncertainty sampling method since we query the pool instance that is closest to the current SVM decision boundary; i.e., the instance that we are most uncertain about. In the multiclass case, however, the Simple method and uncertainty sampling differ. The Simple method attempts to approximately reduce the size of the version space and using the current SVMs as a guide via Eq. (3.5) and Eq. (3.6). Uncertainty sampling explicitly chooses points that are closest to all of the hyperplanes. For example, given the k current SVMs $f_1, \ldots, f_k$, uncertainty sampling will choose to query the pool instance $\mathbf{x}$ for which:

$$\prod_i f_i(\mathbf{x}) \tag{4.1}$$

is smallest.[8]

We compared the version space Simple active method with the uncertainty sampling active method and regular random sampling on a variety of multiclass data sets: the iris, vehicle and wine UCI Irvine datasets (Blake et al., 1998) and the four-class Corel photo CD image dataset (text domain experiments were not performed due to time constraints). We initialized each of the learners with one instance from each of the classes. Figures 4.15(a-e) show the test set accuracy for the different datasets. We see that our Simple method, which takes a version space reduction view of active learning, generally performs significantly better than uncertainty sampling and random sampling. Furthermore, although the uncertainty sampling criteria for choosing a pool instance (Eq. (4.1)) seems intuitively reasonable, it can sometimes perform significantly worse that random sampling. This observation suggests that designing effective active learning querying components is a subtle task. Furthermore, viewing the binary classification Simple method as a version space reduction method enables us to extend the Simple method to an effective querying algorithm for the multiclass case. In contrast, viewing the binary classification Simple method as uncertainly sampling produces a less effective extension to the multiclass case. This observation indicates that the version space reduction interpretation of the binary classification Simple method, rather than the uncertainty sampling interpretation, is the more consistent view.

[8]Rather than taking the product of f_is, we could instead look at the sum. Empirically, minimizing the product of f_is performs significantly better.

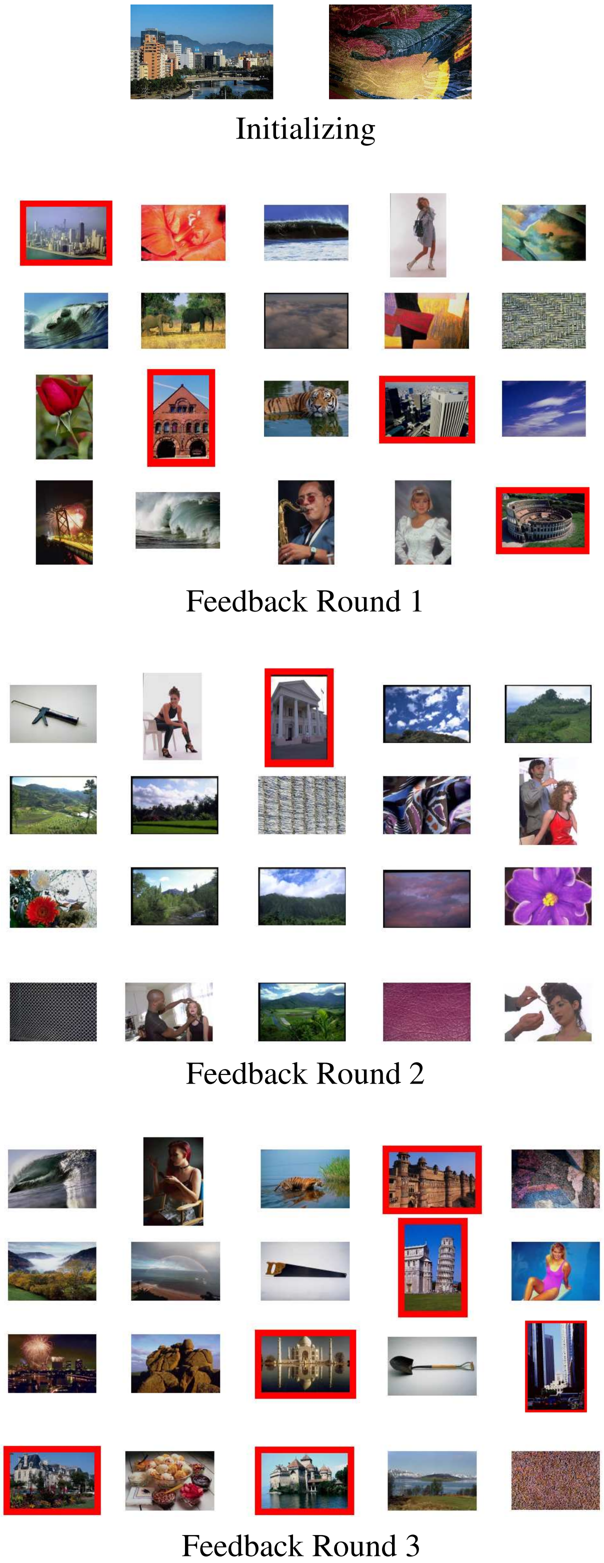
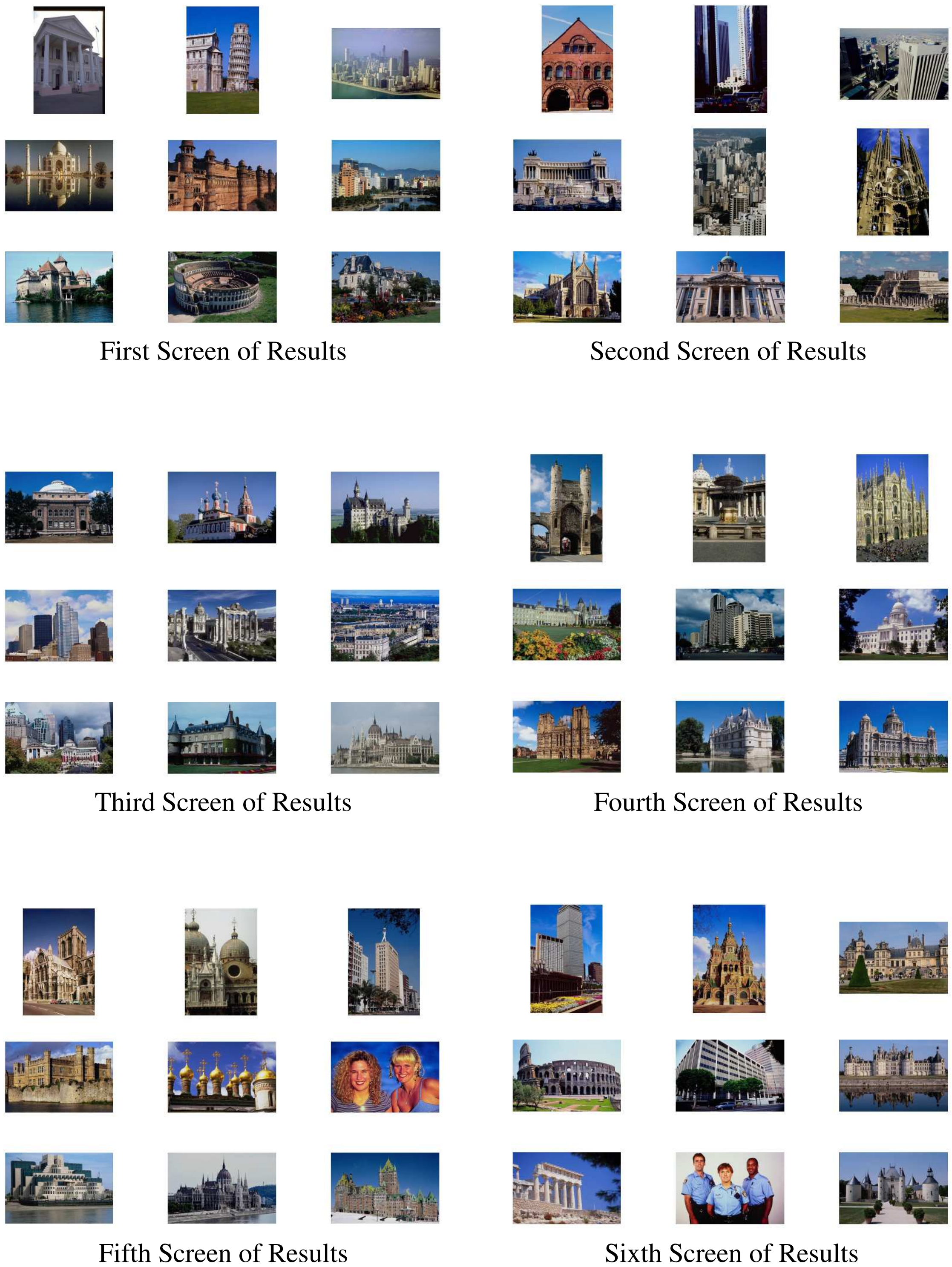


Figure 4.13: Searching for architecture images. SVM$_{Active}$ Feedback phase.



First Screen of Results

Second Screen of Results

Third Screen of Results

Fourth Screen of Results

Fifth Screen of Results

Sixth Screen of Results

Figure 4.14: Searching for architecture images. SVM_{Active} Retrieval phase.

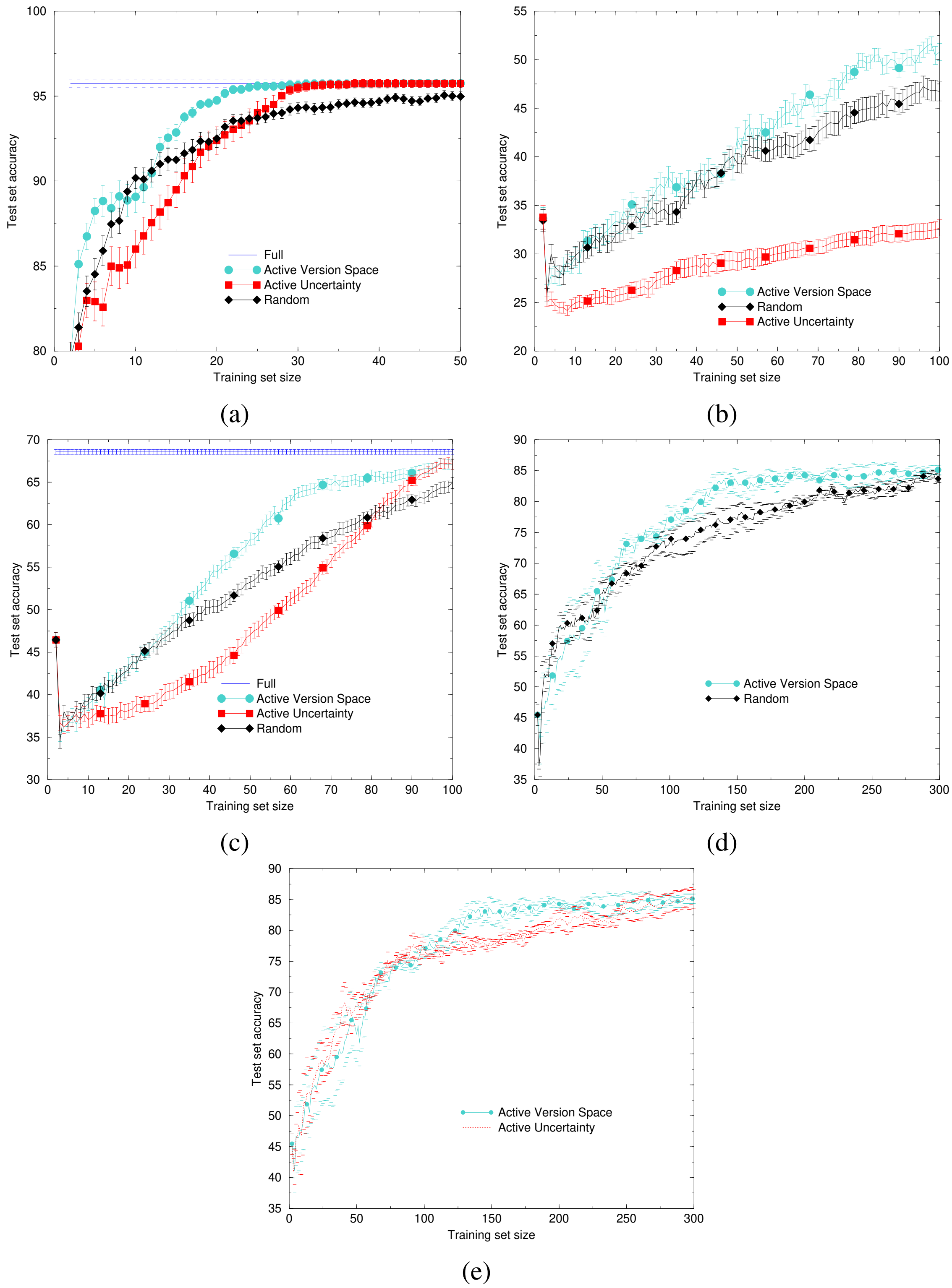


Figure 4.15: (a) Iris dataset. (b) Vehicle dataset. (c) Wine dataset. (d) Image dataset (Active version space vs. Random). (e) Image dataset (Active version space vs. uncertainty sampling). Axes are zoomed for resolution. Legend order reflects order of curves.

Part III

Bayesian Networks

Chapter 5

Bayesian Networks

5.1 Introduction

We often wish to build models that describe domains of interest. However, uncertainty is inherent in the world. In order to provide a realistic model, we would like to encode such non-determinism explicitly. Probability theory provides us with a sound, principled framework for describing and reasoning about uncertainty. In the field of Artificial Intelligence, *Bayesian networks (BNs)* have emerged as the representation of choice for multivariate probability distributions. In the next two chapters we review the main areas of Bayesian network representation, inference and learning which we shall then use in order to tackle active learning in Bayesian networks.

Bayesian networks are a compact graphical representation of joint probability distributions. They have been successfully used as models of a wide variety of complex systems. For example, medical diagnosis (Heckerman, 1988), troubleshooting in the Microsoft Windows operation system (Heckerman et al., 1994), monitoring electric generators (Morjaia et al., 1993), filtering junk email (Sahami et al., 1998), displaying information for time-critical decision making (Horvitz et al., 1992) and determining the needs of software users (Horvitz et al., 1998).

The key property of Bayesian networks is that they permit the explicit encoding of conditional independencies in a natural manner. Thus, Bayesian networks allow qualitative, structural aspects of a domain to be represented and harnessed.

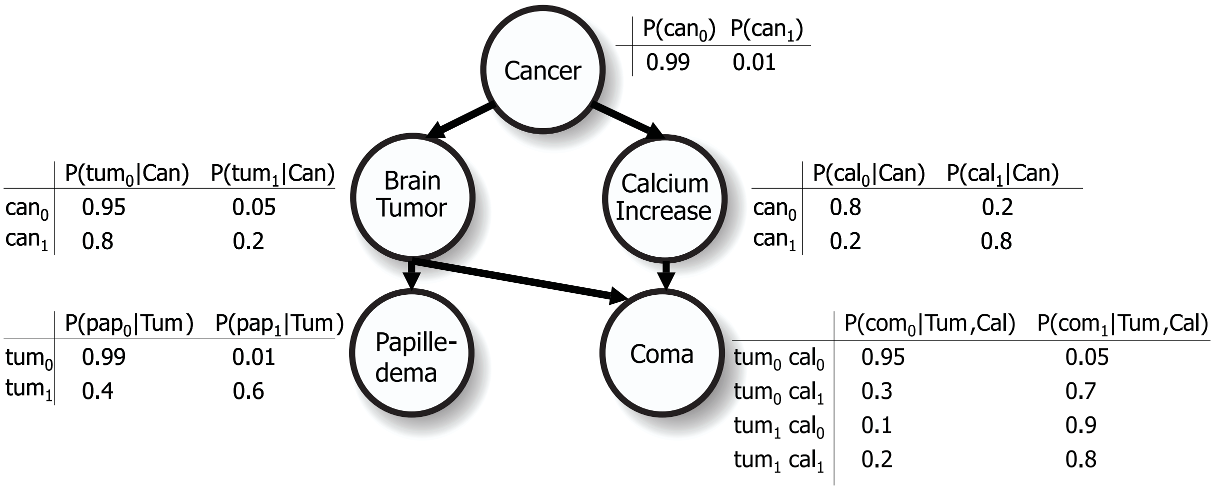


Figure 5.1: **Cancer** Bayesian network modeling a simple cancer domain. "Cancer" denotes whether the subject has secondary, or metastatic, cancer. "Calcium increase" denotes if there is an increase of calcium level in the blood. "Papilledema" is a swelling of the optical disc.

A Bayesian network consists of a graph structure together with local probability models for each node of the graph. See Fig. 5.1 for an example. The graph structure of a Bayesian network encodes conditional independencies of the distribution and the parameters at each node in the BN encode the local conditional distributions of each node given its parents. The network structure, together with the set of numerical parameters, specify a joint distribution over the domain variables. The graphical representation is both compact and natural. Furthermore, the factored representation via local conditional distributions enables a Bayesian network to support both efficient inference and learning from data.[1]

5.2 Notation

Before we proceed to the formal definition of a Bayesian network, it will be helpful to introduce a little notation. We shall be frequently talking about probability distributions

[1] The term Bayesian network is a bit of a misnomer. There is nothing inherently Bayesian about a Bayesian network – any form of statistical parameter estimation can be used to learn a Bayesian network.

over sets of random variables. We shall use the shorthand $P(X_1, \ldots, X_n)$ to denote:

$$\forall x_1, \ldots, x_n \quad P(X_1 = x_1, \ldots, X_n = x_n),$$

and we use $P(x_1, \ldots, x_n)$ to denote:

$$P(X_1 = x_1, \ldots, X_n = x_n).$$

For example, when we write $P(X_1, X_2) = P(X_1)P(X_2 \mid X_1)$ we mean:

$$\forall x_1 \forall x_2 \; P(X_1 = x_1, X_2 = x_2) = P(X_1 = x_1)P(X_2 = x_2 \mid X_1 = x_1),$$

and when we write $P(x_1, x_2) = 0.4$ we mean:

$$P(X_1 = x_1, X_2 = x_2) = 0.4.$$

We use boldface to denote a vector of variables $\mathbf{X} = (X_1, \ldots X_n)$, or instantiations $\mathbf{x} = (X_1 = x_1, \ldots, X_n = x_n)$.

Definition 5.2.1 *We say that* $\mathbf{X}$ *is* conditionally independent *of* $\mathbf{Y}$ *given* $\mathbf{Z}$ *if:*

$$P(\mathbf{X} \mid \mathbf{Y}, \mathbf{Z}) = P(\mathbf{X} \mid \mathbf{Z}),$$

and we denote this relationship by the statement: $I(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})$.

5.3 Definition of Bayesian Networks

The formal definition of a Bayesian network is:

Definition 5.3.1 *Let* $\mathcal{X} = \{X_1, \ldots, X_n\}$ *be a set of random variables. Let* $\mathcal{G}$ *be a directed acyclic graph over* $\mathcal{X}$. *Let* $\mathbf{U}_i$ *be the set of parents of* X_i. *Let* $\boldsymbol{\theta}$ *be a set of parameters which specify* conditional probability distributions (CPDs) $P_{\boldsymbol{\theta}}(X_i \mid \mathbf{U}_i)$. *Then a* Bayesian network *over* $\mathcal{X}$ *is a pair* $(\mathcal{G}, \boldsymbol{\theta})$.

The structure $\mathcal{G}$ of a Bayesian network asserts conditional independence statements given by the following definition:

Definition 5.3.2 *A Bayesian network structure $\mathcal{G}$ encodes the conditional independence statement "Every node is independent of its non-descendants given its parents":*

$$\forall X_i \;\; I(X_i; \textit{Non-descendants}(X_i) \mid \mathbf{U}_i).$$

Given the above definitions it is possible to show that any distribution P satisfying the conditional independencies in Definition 5.3.2 can be encoded as a BN with $\mathcal{G}$ as a structure and with CPDs corresponding to the corresponding local conditional distributions of P, and it can be shown that the joint distribution P can be expressed by the *chain rule for Bayesian networks*:

$$P(X_1, \ldots, X_n) = \prod_i P(X_i \mid \mathbf{U}_i). \tag{5.1}$$

When a distribution P satisfies the conditional independencies in Definition 5.3.2, we say that the distribution P is *consistent* with the structure $\mathcal{G}$, or that $\mathcal{G}$ is an *independency mapping (I-MAP)* of P. Finally, given a Bayesian network $(\mathcal{G}, \boldsymbol{\theta})$, we denote the distribution that it induces over the entire set of variables $\mathbf{X}$ in $\mathcal{G}$ by: $P(\mathbf{X} \mid \boldsymbol{\theta}, \mathcal{G})$.

5.4 D-Separation and Markov Equivalence

The graph structure of a Bayesian network asserts a set of conditional independencies that can be derived from Definition 5.3.2. For example, suppose we have a five node network $(U \leftarrow V \rightarrow X \leftarrow Y \rightarrow Z)$. Then, it is actually possible to prove, using the statements given in Definition 5.3.2, that U is independent of Z for every distribution that is consistent with $\mathcal{G}$.

Definition 5.4.1 *Given a Bayesian network graph structure $\mathcal{G}$, define the* entire *set of conditional independence statements $I(\mathcal{G})$ that $\mathcal{G}$ asserts as the set of conditional independence statements for which every distribution P consistent with $\mathcal{G}$ must satisfy.*

Now, given an arbitrary BN graph, we can deduce the set of conditional independence statements that it encodes by considering which nodes $\mathbf{X}$ are *d-separated* from other nodes $\mathbf{Y}$ given nodes $\mathbf{Z}$. Before we proceed with looking at d-separation, there is a graph substructure that is important to define first:

Definition 5.4.2 *A v-structure is a graph substructure of the form $A \rightarrow B \leftarrow C$. We also say that B is the* center *of the v-structure.*

We can now formally define d-separation:

Definition 5.4.3 *Given a Bayesian network graph structure $\mathcal{G}$, single node X, single node Y and set of nodes $\mathbf{Z}$, we say that X is* **not** d-separated *from Y given $\mathbf{Z}$ if there exists an (undirected) path P from X to Y such that:*

- *Whenever a node W in P is the center of a v-structure, either W or one of W's descendants is in $\mathbf{Z}$.*
- *Whenever a node W in P is not the center of a v-structure it is not in $\mathbf{Z}$.*

We say that X is d-separated *from Y given $\mathbf{Z}$ if no such path exists.*

The definition can be extended to accommodate sets of variables $\mathbf{X}$ and $\mathbf{Y}$: $\mathbf{X}$ is d-separated from $\mathbf{Y}$ given $\mathbf{Z}$ if every X in $\mathbf{X}$ is d-separated from every Y in $\mathbf{Y}$. It can be shown that a conditional independence statement $I(\mathbf{X}; \mathbf{Y} \mid \mathbf{Z})$ is in $I(\mathcal{G})$ if and only if $\mathbf{X}$ is d-separated from $\mathbf{Y}$ given $\mathbf{Z}$.

It is possible for two different network structures to encode identical sets of conditional independence statements. For example, the networks $X \rightarrow Y$ and $X \leftarrow Y$ both encode no conditional independence statements. When two networks encode precisely the same conditional independence statements we say that they are *Markov equivalent* (Pearl, 1988).

Definition 5.4.4 *Let $\mathcal{X}_{\mathcal{G}}$ denote the set of variables in graph $\mathcal{G}$. Then the Markov equivalence class of a Bayesian network structure $\mathcal{G}$ is:*

$$\{\mathcal{G}' \mid \mathcal{X}_{\mathcal{G}} = \mathcal{X}_{\mathcal{G}'}, I(\mathcal{G}) = I(\mathcal{G}')\}.$$

All networks in a Markov equivalence class have the same *skeleton* (the set connected (X, Y) pairs). For some of the pairs, the direction of the edge is fixed, while the other edges can be directed either way (Spirtes et al., 1993). See Fig. 5.2 for an example of networks in the same Markov equivalence class.

5.5 Types of CPDs

In much of our work we shall assume that the CPD of each node consists of a separate multinomial distribution over $\mathit{Dom}[X_i]$ for each instantiation $\mathbf{u}$ of the parents $\mathbf{U}_i$. The BN in Fig. 5.1 is of this form. We have a parameter $\theta_{x_{ij}|\mathbf{u}}$ for each $x_{ij} \in \mathit{Dom}[X_i]$; we use $\boldsymbol{\theta}_{X_i|\mathbf{u}}$ to represent the vector of parameters associated with the multinomial $P(X_i \mid \mathbf{u})$.

In general, any conditional distribution can be used as a CPD. Other common types of CPDs are: tree CPDs (Boutilier et al., 1996), Gaussian CPDs (Lauritzen, 1996) and Conditional Linear Gaussian CPDs (Lauritzen, 1996).

5.6 Bayesian Networks as Models of Causality

A Bayesian network represents a joint distribution over the set of variables $\mathcal{X}$. Viewed as a probabilistic model, it can answer any query of the form $P(\mathbf{Y} \mid \mathbf{Z} = \mathbf{z})$ where $\mathbf{Y}$ and $\mathbf{Z}$ are sets of variables and $\mathbf{z}$ an assignment of values to $\mathbf{Z}$. However, a BN can also be viewed as a *causal model* (Pearl, 2000). Under this perspective, the BN can also be used to answer *interventional queries*, which specify probabilities after we intervene in the model, forcibly setting one or more variables to take on particular values.

In Pearl's framework (Pearl, 2000), an intervention in a causal model that sets a single node $X := x$ replaces the standard causal mechanism of X with one where X is forced to take the value x. In graphical terms, this intervention corresponds to *mutilating* the model $\mathcal{G}$ by cutting the incoming edges to X. Intuitively, in the new model, X does not directly depend on its parents; whereas in the original model, the fact that $X = x$ would give us information (via evidential reasoning) about X's parents, in the intervention, the fact that $X = x$ tells us nothing about the values of X's parents. For example, in a fault diagnosis model for a car, if we observe that the car battery is not charged, we might

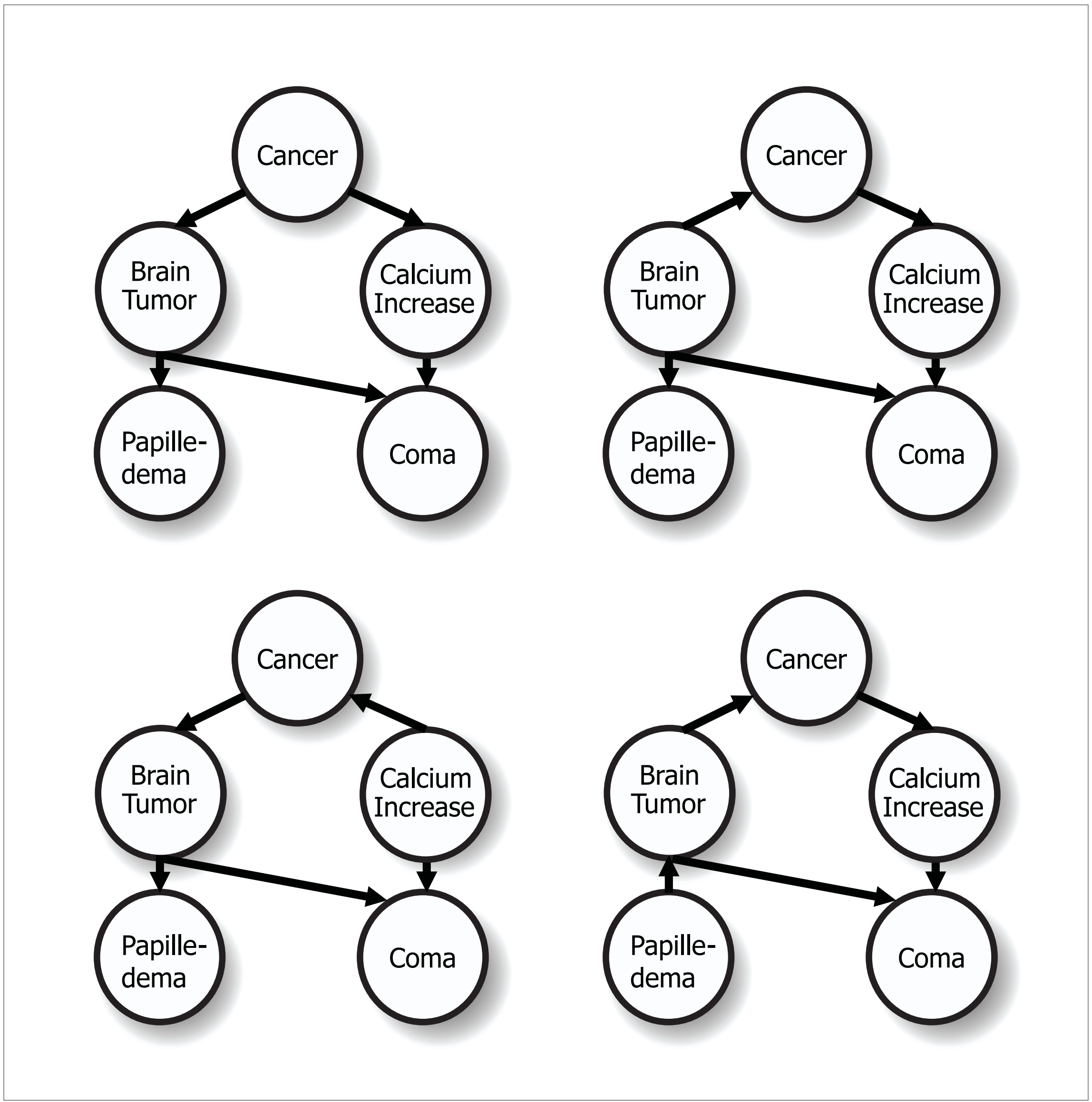


Figure 5.2: The entire Markov equivalence class for the **Cancer** network

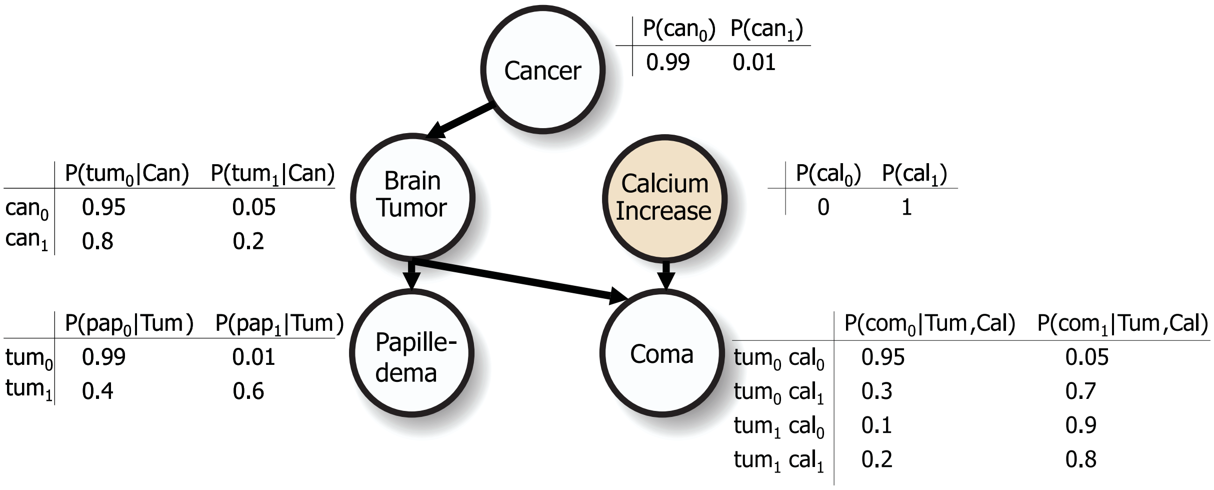


Figure 5.3: Mutilated **Cancer** Bayesian network after we have forced $\mathit{Cal} := \mathit{cal}_1$.

conclude evidentially that the alternator belt is possibly defective, but if we deliberately drain the battery, then the fact that it is empty obviously gives us no information about the alternator belt. Thus, if we set $\mathbf{X} := \mathbf{x}$, the resulting model is a distribution where we mutilate $\mathcal{G}$ to eliminate the incoming edges to nodes in $\mathbf{X}$, and set the CPDs of these nodes so that $\mathbf{X} = \mathbf{x}$ with probability 1.

Fig. 5.3 demonstrates what happens when we intervene in the **Cancer** network by forcing there to be a high calcium level in the blood, i.e., by forcing *Cal* to be cal_1. If we simply observe that there is a high blood calcium level, then the probability of the mouse subject having cancer can be computed to be $P(\mathit{Can} = \mathit{can}_1 \mid \mathit{Cal} = \mathit{cal}_1) = 0.0567$, but if we purposely inject the mouse subject with calcium solution, then the fact that it has a high blood calcium level gives us no information about whether it has cancer and so the probability that the mouse has cancer given that we have set $\mathit{Cal} := \mathit{cal}_1$ is just the prior probability: $P(\mathit{Can} = \mathit{can}_1 \mid \mathit{Cal} := \mathit{cal}_1) = P(\mathit{Can} = \mathit{can}_1) = 0.001$.

More formally, we use define a mutilated Bayesian network that results from performing an intervention as:

Definition 5.6.1 *Let $(\mathcal{G}, \boldsymbol{\theta})$ be a Bayesian network. Let $\mathbf{Y}$ be some set of nodes in $\mathcal{G}$. Define the* mutilated Bayesian network *resulting from the intervention $\mathbf{Y} := \mathbf{y}$ to be the pair $(\mathcal{G}_{\mathbf{Y}:=\mathbf{y}}, \boldsymbol{\theta}_{\mathbf{Y}:=\mathbf{y}})$ where:*

- $\mathcal{G}_{\mathbf{Y}:=\mathbf{y}}$ *is the same as* $\mathcal{G}$ *except any incoming edges to* $\mathbf{Y}$ *are removed.*

- $\boldsymbol{\theta}_{\mathbf{Y}:=\mathbf{y}}$ *is the same as* $\boldsymbol{\theta}$ *except* $\boldsymbol{\theta}_{\mathbf{Y}:=\mathbf{y}}$ *no longer contains parameters for* $P(Y \mid \mathbf{U})$ *for each* $Y \in \mathbf{Y}$*. Instead,* $\boldsymbol{\theta}_{\mathbf{Y}:=\mathbf{y}}$ *contains parameters that define:*

$$P_{\boldsymbol{\theta}_{\mathbf{Y}:=\mathbf{y}}}(Y = y) = \begin{cases} 1 & \textit{if } y_i \textit{ is consistent with } \mathbf{y} \\ 0 & \textit{otherwise} \end{cases}$$

for each $Y \in \mathbf{Y}$.

We now define a causal Bayesian network as follows:

Definition 5.6.2 *Let* $P^*(\mathbf{X})$ *be a probability distribution on a set of variables* $\mathbf{X}$*. Let* $P^*(\mathbf{X} \mid \mathbf{Y} := \mathbf{y})$ *denote the distribution resulting from intervening by forcing* $\mathbf{Y}$ *to have values* $\mathbf{y}$ *where* $\mathbf{Y}$ *is any subset of* $\mathbf{X}$*. The Bayesian network,* $(\mathcal{G}, \boldsymbol{\theta})$ *is a* causal Bayesian network *for the distribution* P^* *if:*

- $P^*(\mathbf{X}) = P(\mathbf{X} \mid \mathcal{G}, \boldsymbol{\theta})$,

- $\forall \mathbf{Y} \subseteq \mathbf{X}$, $P^*(\mathbf{X} \mid \mathbf{Y} := \mathbf{y}) = P(\mathbf{X} \mid \mathcal{G}_{\mathbf{Y}:=\mathbf{y}}, \boldsymbol{\theta}_{\mathbf{Y}:=\mathbf{y}})$.

5.7 Inference in Bayesian Networks

One of the main tasks of a probabilistic model is *inference*. The task of inference is to determine the value of a probabilistic expression involving the domain variables. For example, given the **Cancer** network (Fig. 5.1) we may wish to know the marginal distribution of coma: $P(\mathit{Com})$. Whilst in general the task of inference in BNs is NP-hard (Cooper, 1990), in a great many cases the factored representation of a Bayesian network allows us to compute this type of expression efficiently.

5.7.1 Variable Elimination Method

Let's consider computing the marginal distribution $P(\mathit{Com})$ in the **Cancer** network. By the Bayesian network chain rule we have:

$$
\begin{aligned}
&P(Com) && (5.2)\\
&\quad = \sum_{Tum,Cal,Pap,Can} P(Com, Tum, Cal, Pap, Can) && (5.3)\\
&\quad = \sum_{Tum}\sum_{Cal}\sum_{Pap}\sum_{Can} P(Com \mid Tum, Cal)P(Pap \mid Tum)P(Cal \mid Can)P(Tum \mid Can)P(Can). && (5.4)
\end{aligned}
$$

Notice that if we naively compute this expression we will be doing a lot of unnecessary work. For example, some of the terms do not involve the variable *Can*. Hence, it is unnecessary, and inefficient, to have the scope for the sum over *Can* to be all of the terms. In general, we can simplify the computation of this expression by pushing the summations in as far as they can go:

$$\sum_{Tum}\sum_{Cal} P(Com \mid Tum, Cal)\sum_{Pap} P(Pap \mid Tum)\sum_{Can} P(Cal \mid Can)P(Tum \mid Can)P(Can). \quad (5.5)$$

Now let us consider how to evaluate this expression efficiently. We first consider the innermost summation:

$$\sum_{Can} P(Cal \mid Can)P(Tum \mid Can)P(Can). \quad (5.6)$$

This expression is a sum over *Can* of an expression involving the variables Cal, Can and *Tum*. Note that $P(Cal \mid Can)$, $P(Tum \mid Can)$ and $P(Can)$ can be represented as tables, with each row of the table corresponding to a different instantiation of the variables. For example the table for $P(Cal \mid Can)$ is:

Cal	*Can*	$P(cal \mid can)$
cal_0	can_0	0.8
cal_0	can_1	0.2
cal_1	can_0	0.2
cal_1	can_1	0.8

We combine $P(Cal \mid Can)P(Tum \mid Can)P(Can)$ into one function, h_1, where h_1 is a function of *Cal*, *Can* and *Tum*. We call h_1 a *factor*. In fact we call all such functions involved in the inference computation, factors. Thus $P(Cal \mid Can)$, $P(Tum \mid Can)$ and $P(Can)$ are also called factors. We can represent the factor h_1 as a table where the (cal, can, tum) entry is equal to:

$$P(Cal = cal \mid Can = can)P(Tum = tum \mid Can = can)P(Can = can).$$

In other words, by multiplying the tables for $P(Cal \mid Can)$, $P(Tum \mid Can)$ and $P(Can)$ together we obtain the factor h_1.[2] The table for h_1 is then:

Cal	*Can*	*Tum*	$h_1(cal, can, tum)$
cal_0	can_0	tum_0	0.7524
cal_0	can_0	tum_1	0.0396
cal_0	can_1	tum_0	0.0016
cal_0	can_1	tum_1	0.0004
cal_1	can_0	tum_0	0.1881
cal_1	can_0	tum_1	0.0099
cal_1	can_1	tum_0	0.0064
cal_1	can_1	tum_1	0.0016

Now, returning to our inference computation, after summing out *Can* we end up with a function involving only *Tum* and *Cal*.

$$\begin{aligned} \sum_{Can} P(Cal \mid Can)P(Tum \mid Can)P(Can) &= \sum_{Can} h_1(Cal, Can, Tum) && (5.7) \\ &= g_1(Cal, Tum). && (5.8) \end{aligned}$$

The factor g_1 can be represented as a table, where we sum out *Can* in h_1:

[2]More formally $h_1(Cal, Can, Tum)$ is the *outer-product* of $P(Cal \mid Can)$, $P(Tum \mid Can)$ and $P(Can)$.

Cal	*Tum*	$g_1(cal, tum)$
cal_0	tum_0	0.754
cal_0	tum_1	0.04
cal_1	tum_0	0.1945
cal_1	tum_1	0.0115S

Substituting g_1 back into Eq. (5.5) we obtain:

$$\begin{aligned} &P(Com) \\ &= \sum_{Tum}\sum_{Cal} P(Com \mid Tum, Cal) \sum_{Pap} P(Pap \mid Tum) \sum_{Can} P(Cal \mid Can) P(Tum \mid Can) P(Can) \\ &= \sum_{Tum}\sum_{Cal} P(Com \mid Tum, Cal) \sum_{Pap} P(Pap \mid Tum) g_1(Cal, Tum). \end{aligned}$$

We have eliminated one of the summations, and hence we have eliminated one of the variables in the expression. Notice that to eliminate the variable *Can* we did *not* have to sum over all of the terms in the expression. We only had to sum over the terms $P(Cal \mid Can)$, $P(Tum \mid Can)$ and $P(Can)$. This is essentially where we gain computational efficiency over the naive evaluation of the expression.[3]

We can proceed similarly, multiplying and summing out factors, to eliminate the other variables:

$$\begin{aligned} &P(Com) && (5.9) \\ &= \sum_{Tum}\sum_{Cal} P(Com \mid Tum, Cal) \sum_{Pap} P(Pap \mid Tum) \sum_{Can} h_1(Cal, Can, Tum) && (5.10) \\ &= \sum_{Tum}\sum_{Cal} P(Com \mid Tum, Cal) \sum_{Pap} P(Pap \mid Tum) g_1(Cal, Tum) && (5.11) \\ &= \sum_{Tum}\sum_{Cal} P(Com \mid Tum, Cal) \left(\sum_{Pap} h_2(Pap, Tum) \right) g_1(Cal, Tum) && (5.12) \end{aligned}$$

[3]The general paradigm for the variable elimination algorithm is dynamic programming. We solve a large problem by solving subproblems, storing their results, and using them to solving other subproblems until we have solved the large problem.

$$= \sum_{Tum} \sum_{Cal} P(Com \mid Tum, Cal) g_2(Tum) g_1(Cal, Tum) \tag{5.13}$$
$$= \sum_{Tum} \sum_{Cal} h_3(Com, Tum, Cal) \tag{5.14}$$
$$= \sum_{Tum} g_3(Com, Tum) \tag{5.15}$$
$$= \sum_{Tum} h_4(Com, Tum) \tag{5.16}$$
$$= g_4(Com). \tag{5.17}$$

The final factor $g_4(Com)$ is equal to $P(Com)$:

Com	$g_4(com)$
com_0	0.781
com_1	0.219

We now consider the computational cost. All of our operations involve multiplying and summing out factors. Each of these operations is linear in the number of elements in the tables. Thus the computational complexity is determined by the size of the largest factor encountered in the computation. In our above example the largest factor involved 3 variables, and has size $2^3 = 8$. The total number of additions and multiplications is 52.

If we were to naively evaluate $P(Com)$ directly from Eq. (5.4) we would first need to create a factor over all 5 variables instead and so the largest factor will have size $2^5 = 32$. The total number of multiplications and additions required is 158.

The above computation was performed by first choosing an order in which to eliminate the variables (we chose the order *Can*, *Cal*, *Tum*, *Pap*). The complexity is dependent upon the choice of ordering. The optimal choice of ordering is an NP hard problem (Arnborg et al., 1987), but heuristics can be used to choose a reasonable ordering (Kjaerulff, 1990).

In general, the variable elimination algorithm (Zhang & Poole, 1994) for computing the marginal distribution $P(\mathbf{Y})$ is described in Fig. 5.4. The essential algorithm is to pick a variable to eliminate next, gather terms involving that variable, sum out the variable over those terms and then repeat.

VariableElimination(BN over $\mathcal{X}$, $\mathcal{Y}$)
  Check that $\mathcal{Y}$ is a subset of $\mathcal{X}$
  Initialize $\mathcal{F} := \{P(X_i \mid \mathbf{U}_i) \;\mid\; i = 1, \ldots, n\}$
  $\mathcal{Z} := \mathcal{X} - \mathcal{Y}$
  For Each $Z_i \in \mathcal{Z}$ // this is where the choice of ordering is important
    Extract and remove from $\mathcal{F}$ all factors $g_1, \ldots g_r$ involving Z_i
    $h := \prod_j g_j$
    $g := \sum_{Z_i} h$
    Insert g into $\mathcal{F}$
  End For
  Return $\left(\prod_{g \in \mathcal{F}} g\right)$

Figure 5.4: The variable elimination algorithm for computing marginal distributions.

Conditional Queries

The above algorithm is used to compute marginal distributions such as $P(\mathit{Can}, \mathit{Com})$ and $P(\mathit{Tum}, \mathit{Pap})$. We often wish to consider posterior distributions such as: $P(\mathit{Can} \mid \mathit{com}_0)$ and $P(\mathit{Tum} \mid \mathit{pap}_1)$. The variable elimination algorithm can be easily adapted to compute such distributions. Suppose we have evidence $\mathit{Com} = \mathit{com}_0$ and we want to compute $P(\mathit{Can} \mid \mathit{com}_0)$. By definition:

$$P(\mathit{Can} \mid \mathit{com}_0) = \frac{P(\mathit{Can}, \mathit{com}_0)}{P(\mathit{com}_0)}.$$

Thus, to compute $P(\mathit{Can} \mid \mathit{com}_0)$ we just need to compute $P(\mathit{Can}, \mathit{com}_0)$ and then renormalize. Computing $P(\mathit{Can}, \mathit{com}_0)$ is straightforward:

$$
\begin{aligned}
&P(\mathit{Can}, \mathit{com}_0) && (5.18)\\
&= \sum_{\mathit{Tum},\mathit{Cal},\mathit{Pap}} P(\mathit{com}_0, \mathit{Tum}, \mathit{Cal}, \mathit{Pap}, \mathit{Can}) && (5.19)\\
&= \sum_{\mathit{Tum}} \sum_{\mathit{Cal}} \sum_{\mathit{Pap}} P(\mathit{com}_0 \mid \mathit{Tum}, \mathit{Cal}) P(\mathit{Pap} \mid \mathit{Tum}) P(\mathit{Cal} \mid \mathit{Can}) P(\mathit{Tum} \mid \mathit{Can}) P(\mathit{Can}). && (5.20)
\end{aligned}
$$

Now, we can evaluate Eq. (5.20) just like we did before in the marginal distribution case (Eq. (5.4)). Like before, we have a sum of product of factors. The only difference is that we have slightly different factors than before.

Notice that, because we are conditioning on evidence $\mathit{Com} = \mathit{com}_0$, $P(\mathit{com}_0 \mid \mathit{Tum}, \mathit{Cal})$ is now not a function of *Com*. It is just a function of *Tum* and *Cal*. The table representing $P(\mathit{com}_0 \mid \mathit{Tum}, \mathit{Cal})$ can be obtained by *reducing* the table for $P(\mathit{Com} \mid \mathit{Tum}, \mathit{Cal})$. That is, the table for $P(\mathit{Com} \mid \mathit{Tum}, \mathit{Cal})$:

Com	*Tum*	*Cal*	$P(\mathit{com} \mid \mathit{tum}, \mathit{cal})$
com_0	tum_0	cal_0	0.95
com_0	tum_0	cal_1	0.3
com_0	tum_1	cal_0	0.1
com_0	tum_1	cal_1	0.2
com_1	tum_0	cal_0	0.05
com_1	tum_0	cal_1	0.7
com_1	tum_1	cal_0	0.9
com_1	tum_1	cal_1	0.8

reduces to:

Tum	*Cal*	$P(\mathit{com}_0 \mid \mathit{tum}, \mathit{cal})$
tum_0	cal_0	0.95
tum_0	cal_1	0.3
tum_1	cal_0	0.1
tum_1	cal_1	0.2

Definition 5.7.1 *Given a factor $g(X_1, \ldots, X_r)$, and instantiation $X_i = x_{ij}$ then the* reduced *factor $g|_{x_{ij}}$ is defined as:*

$$g|_{x_{ij}}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_r) = g(x_1, \ldots, x_{i-1}, x_{ij}, x_{i+1}, \ldots, x_r).$$

This definition can be extended in the obvious way to reducing factors by an instantiation that involves more that one variable.

CondVariableElimination(BN over $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{W}$, $\mathbf{w}$)
  Check that $\mathcal{Y}$, $\mathcal{W}$ and $\mathbf{w}$ only contain variables in $\mathcal{X}$
  Initialize $\mathcal{F} := \{P(X_i \mid \mathbf{U}_i) \quad | \quad i = 1, \ldots, n\}$
  Reduce each $g \in F$ to $g|_{\mathbf{w}}$
  $\mathcal{Z} := \mathcal{X} - \mathcal{Y} - \mathcal{W}$
  For Each $Z_i \in \mathcal{Z}$
    Extract and remove from $\mathcal{F}$ all factors $g_1, \ldots g_r$ involving Z_i
    $h := \prod_j g_j$
    $g := \sum_{Z_i} h$
    Insert g into $\mathcal{F}$
  End For
  Return renormalize$\Big(\prod_{g \in \mathcal{F}} g\Big)$

Figure 5.5: The Variable Elimination Algorithm.

Fig. 5.5 describes the general variable elimination algorithm for computing the distribution $P(Y_1, \ldots, Y_m \mid \mathbf{W} = \mathbf{w})$.

5.7.2 The Join Tree Algorithm

The join tree algorithm (Lauritzen & Spiegelhalter, 1988; Huang & Darwiche, 1996) is another method for computing marginal and posterior distributions. It is also known as the cluster tree, clique tree and junction tree algorithm. It is very similar to the variable elimination algorithm. It is slightly more complicated but allows one to simultaneously compute distributions over different sets of variables (for example, it can simultaneously derive $P(\textit{Can} \mid \textit{com}_0)$, $P(\textit{Tum}, \textit{Cal} \mid \textit{com}_0)$ and $P(\textit{Pap} \mid \textit{com}_0)$). This is in contrast to the variable elimination algorithm which can only compute one distribution at a time.

We shall describe the essence of this algorithm. A more detailed account can be found in (Huang & Darwiche, 1996). To gain an understanding of how this algorithm works, let us first take another look at the variable elimination algorithm for computing marginal distributions $P(\mathbf{Y})$. Consider the **Cancer** example that we presented in Equations 5.9 to 5.17. We can take a more graphical view of this computation. Given the elimination ordering *Can*, *Pap*, *Cal*, *Tum*, the variable elimination algorithm creates a series of intermediate factors $h_1, \ldots h_k$. Let us represent each h factor that we create as a node in an undirected

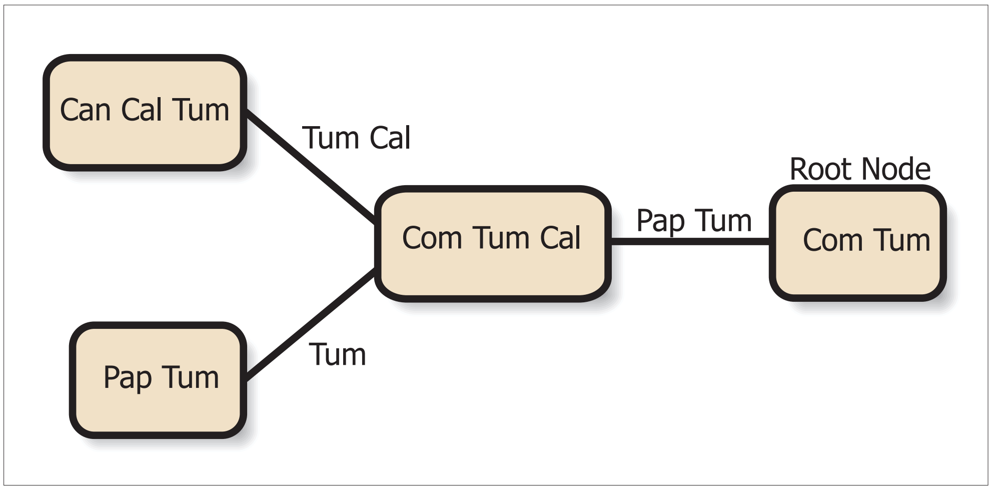


Figure 5.6: Initial join tree for the **Cancer** network constructed using the elimination ordering *Can*, *Pap*, *Cal*, *Tum*.

graph. We call the node corresponding to the last h created as the *root* node. We label the node with the variables that appear in h. Whenever we use a g (obtained by summing out a variable in h) to compute another h' we draw an edge between the node for h and the node for h' and we label the edge with the variables present in g. For each node, there is an edge that leads towards the root. We call such an edge the *outgoing* edge and we call all other edges the *incoming* edges. We call such a graph a *join tree*. See Fig. 5.6 for an example.

Given a join tree obtained from a given ordering, the variable elimination algorithm for computing marginal distributions $P(Y_1, \ldots, Y_m)$ can then be restated as follows:

- Insert each CPD $P(X_i \mid \mathbf{U}_i)$ into a node that is labeled with at least the variables $\{X_i\} \cup \mathbf{U}_i$.
- Starting from the leaf nodes, multiply incoming factors with any factors resident within the node. (This creates our h factors). Then sum out all variables not present in the variables on the outgoing edge (this creates our g factors) and pass this factor along the outgoing edge.

See Figures 5.7(a) and 5.7(b) for an example . By the end of this process we end up with a factor at the root node which is equal to the marginal distribution for the variables in the root node. Since the root node corresponded to the last h created in the variable elimination algorithm, it contains all of the variables $Y_1, \ldots, Y_m$. We then sum out the extraneous variables in the root node factor to obtain $P(Y_1, \ldots, Y_m)$.

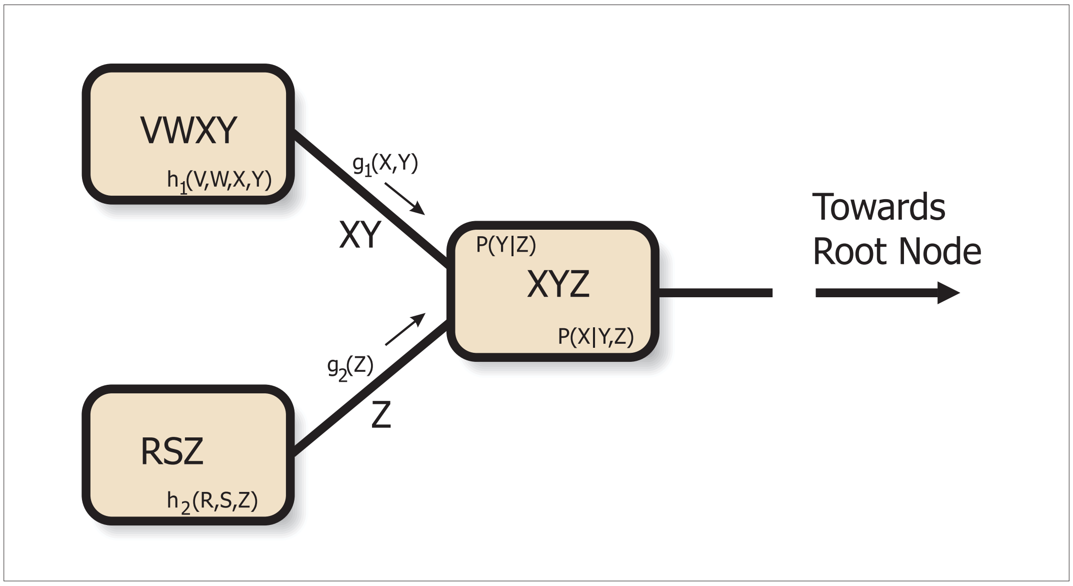
The join tree algorithm works in a very similar manner to this. In the terminology of the join tree algorithm, we have just performed the "upward" pass of the algorithm – we have passed factors from leaves to the root. After the upward pass the root node contains the marginal probability of the variables labeling that node. However, in general, for any other node, the h factor in that node is not equal to the marginal distribution of the variables labeling that node.

The join tree algorithm has a second phase called the "downward" pass. Here we pass factors from the root down towards the leaves. Figures 5.8(a) and 5.8(b) show how this process is performed. After the second phase, it can be shown that each node's h factor is the marginal distribution of the variables within that node (Lauritzen & Spiegelhalter, 1988). After performing these two passes we say that the tree has been *calibrated*.

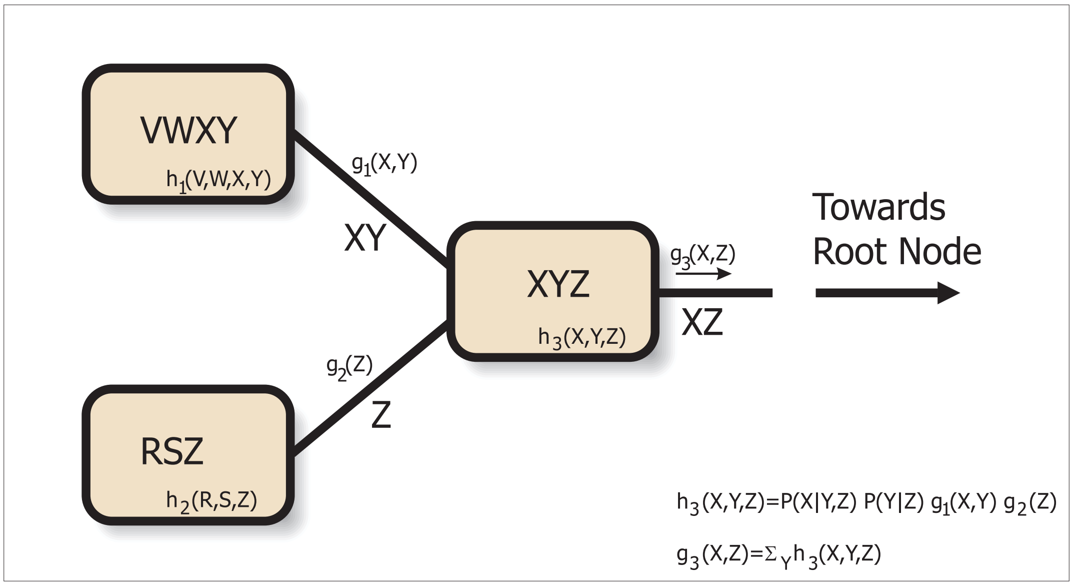
The cost of this algorithm is at most twice that of the variable elimination algorithm, but we have now managed to compute many different marginal distributions at once. We also note for future reference that each CPD factor belongs to a node, and so a calibrated join tree contains, in an accessible form, the marginals of $P(\mathbf{U}_i)$ for each node X_i.

Dealing with evidence $\mathbf{W} = \mathbf{w}$, is a straightforward modification, just as it was with the variable elimination algorithm. Before performing the upward and downward passes we reduce all the factors to be consistent with the evidence $\mathbf{w}$. After calibrating the tree with the upward and downward passes the factors at each node of the tree are distributions of the form: $P(Y_1, \ldots, Y_m, \mathbf{w})$. Renormalizing such factors will then yield: $P(Y_1, \ldots, Y_m \mid \mathbf{w})$ for each node.

We have briefly described the essence of the join tree algorithm. There are number of extensions and improvements to the algorithm that can be made. In reality, the structure of the join tree is constructed by a different method, rather than by using an ordering for variable elimination. It can be shown that, so long as the tree is constructed to obey certain properties, the upward and downward passes are guaranteed to produce the correct marginal distributions at each of the tree's nodes. Furthermore, it doesn't matter which join tree node we choose as the root node. See (Lauritzen & Spiegelhalter, 1988) for a more detailed explanation. The upward and downward passes need not be performed in series; they can

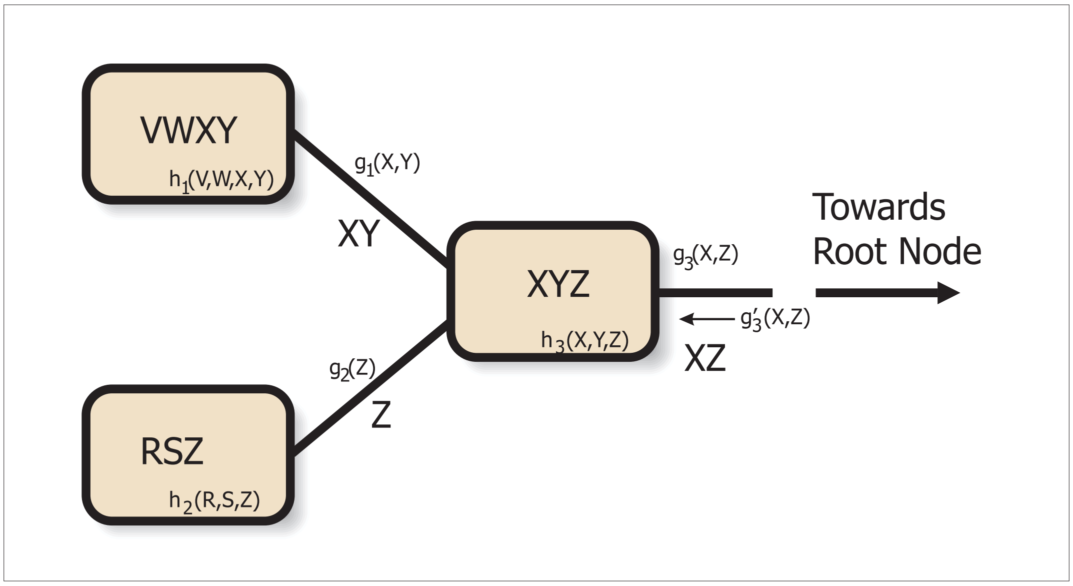


(a)

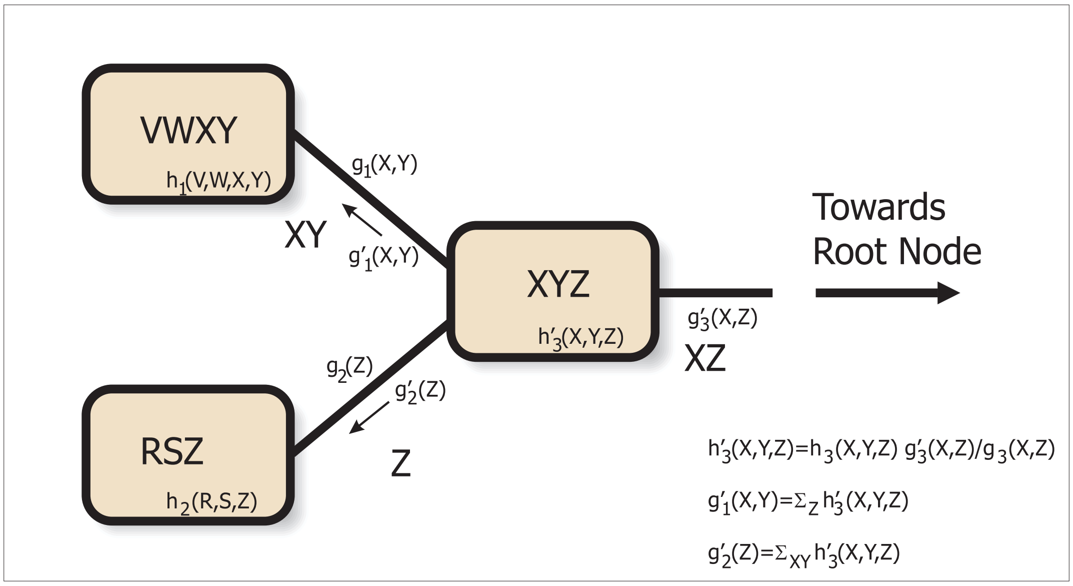


(b)

Figure 5.7: Processing the node XYZ during the upward pass. (a) Before processing the node. (b) After processing the node.



(a)



(b)

Figure 5.8: Processing the node XYZ during the downward pass. (a) Before processing the node. (b) After processing the node.

be interleaved to create a distributed algorithm. Also, given an calibrated tree, there are ways of efficiently recalibrating the tree in light of new evidence (Huang & Darwiche, 1996).

The variable elimination and join tree algorithms mentioned here are exact inference methods – they will always produce the exact answer. If exact inference is too costly for a given network there are a variety of approximate inference techniques that one can resort to. The most popular of these methods are likelihood sampling (Shachter & Peot, 1989), Markov Chain Monte Carlo techniques (Geman & Geman, 1987; Neal, 1993), variational approximations (Jordan et al., 1998) and loopy belief propagation (Murphy & Weiss, 1999; Yedidia et al., 2001).

Chapter 6

Parameter Estimation

6.1 Introduction

A Bayesian network consists of two components – the graph structure and the parameters. In a number of situations the graph structure is easier to obtain than the parameters, particularly when working with a domain expert. Human experts often have the ability to describe the qualitative correlations in a domain but, typically, they find it harder to pinpoint the exact parameter values.

Our first area of focus is the *parameter estimation* task. Suppose we are given data $D = \{\mathbf{d}[1], \mathbf{d}[2], \ldots, \mathbf{d}[M]\}$, where the instances are independent and identically distributed (i.i.d.). We are given the network structure $\mathcal{G}$, and our goal is to use the data to estimate the network parameters $\boldsymbol{\theta}$.

Much of the work on parameter estimation for BNs has focused on the case where we are presented with discrete data and wish to find the parameters of CPDs that take the form of tables of multinomials (with one multinomial for each possible assignment of the parents). We shall restrict our attention to this case. See Fig. 6.1 for an example of such a network. We let $\boldsymbol{\theta}_{X_i|\mathbf{u}}$ denote the set of parameters associated with the conditional distribution $P(X_i \mid \mathbf{u})$ and we denote the entire set of parameters for every possible parent instantiation $\mathbf{u}$ of a node X_i by $\boldsymbol{\theta}_{X_i|\mathbf{U}_i}$. We also introduce another piece of useful notation:

Definition 6.1.1 *Given discrete data D and let $\mathbf{y}$ be a partial instantiation. Denote the number of data instances in D that are consistent with the partial instantiation $\mathbf{y}$ by: $N(\mathbf{y})$.*

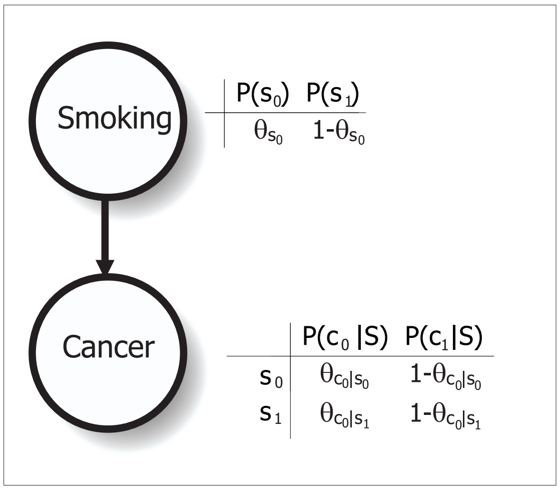


Figure 6.1: **Smoking** Bayesian network with its parameters.

6.2 Maximum Likelihood Parameter Estimation

A standard approach to parameter estimation in general is the *maximum likelihood* method. The intuitive idea is to choose the parameters $\boldsymbol{\theta}$ that best explain the observed data D. More formally:

Definition 6.2.1 *Given a family of distributions $P(.|\boldsymbol{\theta})$ parameterized by $\boldsymbol{\theta}$, and i.i.d. data D, the* maximum likelihood estimator (MLE) $\hat{\boldsymbol{\theta}}$ *is given by:*

$$\hat{\boldsymbol{\theta}} = \text{argmax}_{\boldsymbol{\theta}} P(D \mid \boldsymbol{\theta}). \tag{6.1}$$

As a simple example, suppose that we have a single binary node network: X. The CPD of X is parameterized by a single parameter θ_{x_0} that corresponds to the probability that $X = x_0$. Suppose that our data D consists of $N(x_0)$ occurrences of x_0 and $N(x_1)$ occurrences of x_1. Then the MLE is:

$$\begin{aligned}
\hat{\theta}_{x_0} &= \text{argmax}_{\theta_{x_0}} P(D \mid \theta_{x_0}) && (6.2)\\
&= \text{argmax}_{\theta_{x_0}} \theta_{x_0}^{N(x_0)} (1-\theta_{x_0})^{N(x_1)} && (6.3)\\
&= \frac{N(x_0)}{N(x_0)+N(x_1)}. && (6.4)
\end{aligned}$$

Smoker	Cancer	Frequency
No	No	80
No	Yes	0
Yes	No	19
Yes	Yes	1

Figure 6.2: An example data set for the **Smoking** network

In general the following theorem is a well known result (Heckerman, 1998):

Theorem 6.2.2 *The MLE for a discrete Bayesian network with graph structure G and multinomial table CPDs is given by:*

$$\forall i \forall j \forall \mathbf{u} \quad \hat{\theta}_{x_{ij}|\mathbf{u}} = \frac{N(x_{ij}, \mathbf{u})}{N(\mathbf{u})}. \tag{6.5}$$

where $\hat{\theta}_{x_{ij}|\mathbf{u}}$ is the parameter for the j-th value of the i-th node with parent instantiation $\mathbf{u}$.

MLE gives an intuitive, natural form of estimation. Furthermore, it is an objective estimator and does not rely on any subjectivity on the part of the human designer. However its main drawback is that it can tend to “overfit” the data. This phenomenon is particularly acute with discrete data and low probability events (a very common situation with medical diagnosis). For example, suppose we wish to fit parameters to the **Smoking** network in Fig. 6.1. Assume that we see a sample consisting of 100 people, 20 of whom smoke. Furthermore, suppose one smoker develops cancer and none of the 80 non-smokers develop cancer. This set of data is summarized in Fig. 6.2. Our maximum likelihood estimates for the network parameters are then:

$$\begin{aligned} \theta_{s_0} &= 0.8, \\ \theta_{c_0|s_0} &= 1, \\ \theta_{c_0|s_1} &= 0.95. \end{aligned}$$

Thus we are asserting that if someone does not smoke he or she will *never* develop cancer – an extremely strong statement. It asserts that it is *impossible* to develop cancer if we do not smoke. This claim is very different than saying that there is a very small, but non-zero, chance of a non-smoker developing cancer. Our estimate of $\theta_{c0|s0} = 1$ came about because we did not have enough data to provide sufficient resolution of the low probability event of developing cancer. There are a number of techniques one can use to address this issue (Lehmann, 1986; Lehmann & Casella, 1998). We next review the Bayesian methodology which has become one of the cornerstones of learning with Bayesian networks and it is a framework which tackles this issue of "overfitting" in an elegant manner.

6.3 Bayesian Parameter Estimation

6.3.1 Motivation

The reason why we find the estimate of $\theta_{c0|s0} = 1$ unsatisfying is that we think that there is some non-zero chance of developing cancer even if we do not smoke. Similarly, if we have a coin and we wish to estimate the probability of heads, and if we toss the coin only once and it lands as heads, then we find it unreasonable to use the ML estimate of $\theta_h = 1$. Intuitively, we find it unreasonable because we have some prior knowledge that coins generally do not land only on one side. If, however, our coin lands heads after a hundred tosses then we may be more willing to accept that the coin is biased; in other words our prior knowledge becomes less important the more data we receive. The Bayesian framework formalizes these intuitions as well as allowing us to incorporate other types of prior knowledge.

6.3.2 Approach

In the coin example, rather that ascribing a single number to θ_h we maintain a density over its possible values. The initial density in the absence of data, $p(\theta_h)$, is called the *prior*, and it encodes our prior beliefs of the parameter. If we have just a little prior knowledge and are unsure of θ_h's value then this density will be fairly flat. On the other hand, if we have a fairly strong prior belief of the value of θ_h then this prior density will be more peaked. As we gather more data D, we update this density, $p(\theta_h)$ to get the *posterior* density, $p(\theta_h \mid D)$.

Bayesian parameter estimation in Bayesian networks works in the same way. We keep a density over possible parameter values. We will make the common assumption of *parameter independence* (Heckerman et al., 1995): $p(\boldsymbol{\theta}) = \prod_i \prod_{\mathbf{u}} p(\boldsymbol{\theta}_{X_i|\mathbf{u}})$. This assumption allows us to represent the prior distribution $p(\boldsymbol{\theta})$ as a set of independent distributions, one for each multinomial $\boldsymbol{\theta}_{X_i|\mathbf{u}}$.

We now have to choose the functional form of our densities over parameters $\boldsymbol{\theta}_{X_i|\mathbf{u}}$. One desirable property of a parameter density $p(\boldsymbol{\theta}_{X_i|\mathbf{u}})$ is that, when we gather data and obtain the posterior $p(\boldsymbol{\theta}_{X_i|\mathbf{u}} \mid D)$, then the posterior density has the same functional form as the prior. In other words, when we update our prior parameter densities, we would like the posterior densities to remain in the same family. We call such families of densities *conjugate priors*.

For multinomials, the conjugate prior is a *Dirichlet* distribution (DeGroot, 1970), given by:

$$p(\theta_1, \ldots, \theta_r) = \textit{Dirichlet}(\alpha_1, \ldots, \alpha_r) = \frac{\Gamma(\alpha_*)}{\prod_{i=1}^{r} \Gamma(\alpha_i)} \prod_{j=1}^{r} \theta_j^{\alpha_j - 1}, \tag{6.6}$$

which is parameterized by *hyperparameters* $\alpha_j \in \mathbb{R}^+$, with $\alpha_* = \sum_j \alpha_j$, and Γ is the gamma function. Also, $\sum \theta_i = 1$, $\theta_i \geq 0$.

The Dirichlet family of distributions are conjugate priors for multinomials: if we obtain a new instance $X = x_j$ sampled from this distribution, then our posterior distribution $p'(\boldsymbol{\theta})$ is also distributed Dirichlet with hyperparameters $(\alpha_1, \ldots, \alpha_j + 1, \ldots, \alpha_r)$. In general the following is a standard result (DeGroot, 1970):

Theorem 6.3.1 *Let X be distributed multinomial with parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_r)$ and let $p(\boldsymbol{\theta}) = \textit{Dirichlet}(\alpha_1, \ldots, \alpha_r)$. Given i.i.d data D we have that:*

$$p(\boldsymbol{\theta} \mid D) = \textit{Dirichlet}(\alpha_1 + N(x_1), \ldots, \alpha_r + N(x_r)). \tag{6.7}$$

Intuitively, α_j represents the number of "imaginary x_j instances" observed prior to observing any actual data. In particular the follow well known result holds (DeGroot, 1970):

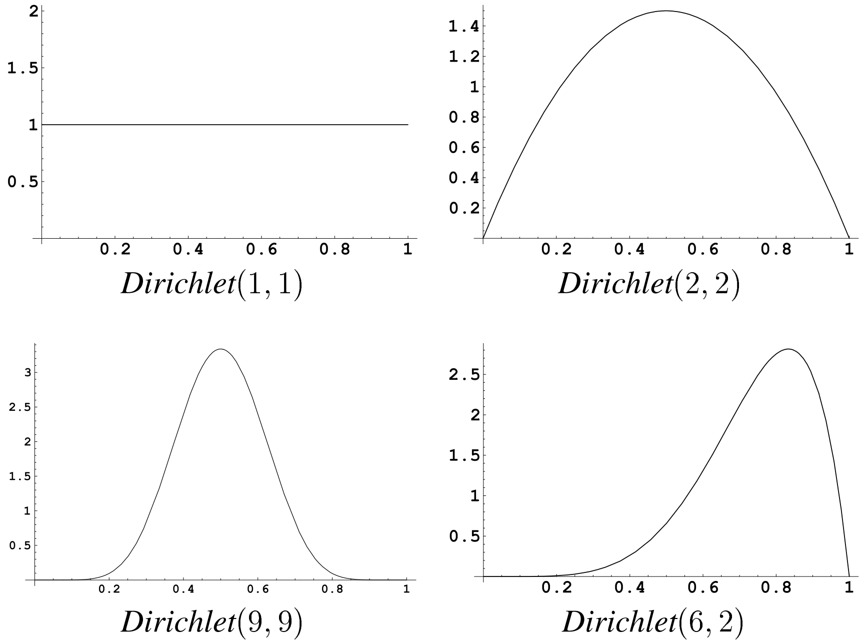


Figure 6.3: Examples of the Dirichlet distribution. θ is on the horizontal axis, and $p(\theta)$ is on the vertical axis.

Theorem 6.3.2 *Let X be distributed multinomial with parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_r)$ and let $p(\boldsymbol{\theta}) = \text{Dirichlet}(\alpha_1, \ldots, \alpha_r)$. The probability that our next observation is x_j is:*

$$P(x_j) = \frac{\alpha_j}{\alpha_*}.$$

Thus, the relative sizes of the different α_j's determine our prior beliefs in the probabilities of the different outcomes for X. The absolute sizes of the α_j's determine our confidence in the estimate; the higher α_* is, the longer it will take for our posterior distribution to be influenced by new data. See Fig. 6.3 for examples of some Dirichlet densities.

In a BN with the parameter independence assumption, we have a Dirichlet distribution for every multinomial distribution $\boldsymbol{\theta}_{X_i|\mathbf{u}}$. Given a distribution $p(\boldsymbol{\theta})$, we use $\alpha_{x_{ij}|\mathbf{u}}$ to denote the hyperparameter corresponding to the parameter $\theta_{x_{ij}|\mathbf{u}}$.

6.3.3 Bayesian One-Step Prediction

In most cases we are not necessarily interested in the densities over the parameters $p(\boldsymbol{\theta})$ in a BN. We are often more interested in distribution over the domain variables $X_1, \ldots, X_n$. For example, given complete data D, we may wish to know the probability of next seeing a data instance $\mathbf{x}$. Using Theorem 6.3.2, the following corollary can be shown to hold:

Corollary 6.3.3 *Let $(G, \boldsymbol{\theta})$ be a Bayesian network over $\mathbf{X} = (X_1, \ldots, X_n)$. Let D be i.i.d complete data, and let the prior density over parameters $p(\boldsymbol{\theta}) = \prod_i \prod_{\mathbf{u}} p(\boldsymbol{\theta}_{X_i|\mathbf{u}})$ be a product of Dirichlet densities. Then the Bayesian one-step prediction of next observing data instance $\mathbf{x}$ is given by:*

$$\begin{aligned} P(\mathbf{x} \mid D) &= E_{\boldsymbol{\Theta} \sim p(\boldsymbol{\theta}|D)} P(\mathbf{x} \mid \boldsymbol{\Theta}) && (6.8) \\ &= \int_{\boldsymbol{\theta}} P(\mathbf{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid D)\, d\boldsymbol{\theta} && (6.9) \\ &= \prod_i \frac{\alpha_{x_{ij}|\mathbf{u}} + N(x_{ij}, \mathbf{u})}{\sum_j \left(\alpha_{x_{ij}|\mathbf{u}} + N(x_{ij}, \mathbf{u})\right)}, && (6.10) \end{aligned}$$

where x_{ij} is the value of X_i in $\mathbf{x}$, and $\mathbf{u}$ is the value of $\mathbf{U}_i$ in $\mathbf{x}$.

As an example, let us suppose we are given the **Smoking** network and we have a prior distribution as follows:[1]

$$\begin{aligned} p(\theta_{s_0}) &= \textit{Dirichlet}(5, 5), && (6.11) \\ p(\theta_{c_0|s_0}) &= \textit{Dirichlet}(2.5, 2.5), && (6.12) \\ p(\theta_{c_0|s_1}) &= \textit{Dirichlet}(2.5, 2.5). && (6.13) \end{aligned}$$

Now suppose we observe the data in Fig. 6.2. Then the posterior densities over parameters will be:

$$p(\theta_{s_0}) = \textit{Dirichlet}(85, 25),$$

[1] Such a prior is called a *BDe uniform prior* with equivalent sample size of 10. It is as if we have imagined 10 instances that are uniformly distributed. The reason for a *Dirichlet*$(2.5, 2.5)$ density for $\theta_{c_0|s_0}$ is that only half of the 10 imaginary uniform instances would be non-smokers.

$$\begin{aligned}p(\theta_{c_0|s_0}) &= \textit{Dirichlet}(82.5, 2.5),\\ p(\theta_{c_0|s_1}) &= \textit{Dirichlet}(21.5, 3.5).\end{aligned}$$

The probability of observing a non-smoker without cancer next is:

$$P(c_0, s_0 \mid D) = E_{\mathbf{\Theta}\sim p(\boldsymbol{\theta}|D)} P(c_0, s_0 \mid \mathbf{\Theta}) = \frac{85}{85+25} \cdot \frac{82.5}{82.5+2.5} = \frac{3}{4}. \tag{6.14}$$

The probability of observing a non-smoker next is:

$$P(s_0 \mid D) = E_{\mathbf{\Theta}\sim p(\boldsymbol{\theta}|D)} P(s_0 \mid \mathbf{\Theta}) = \frac{85}{85+25} = \frac{17}{22}. \tag{6.15}$$

And the probability of observing a person without cancer next given that he or she is a non-smoker is:

$$P(c_0 \mid s_0, D) = \frac{P(c_0, s_0 \mid D)}{P(s_0 \mid D)} = \frac{82.5}{82.5+2.5} = \frac{33}{34}. \tag{6.16}$$

This is in contrast to the ML estimate, which asserted that:

$$P_{\hat{\boldsymbol{\theta}}}(c_0 \mid s_0) = 1. \tag{6.17}$$

Similarly we have that:

$$P(c_0 \mid s_1, D) = \frac{P(c_0, s_1 \mid D)}{P(s_1 \mid D)} = \frac{21.5}{21.5.5+3.5} = \frac{43}{50}. \tag{6.18}$$

Notice that these Bayesian one-step predictions are equivalent to setting:

$$\theta_{s_0} = \frac{17}{22}, \tag{6.19}$$

$$\theta_{c_0|s_0} = \frac{33}{34}, \tag{6.20}$$

$$\theta_{c_0|s_1} = \frac{43}{50}. \tag{6.21}$$

in the **Smoking** network in Fig. 6.1 and then performing inference in this network with these parameters.

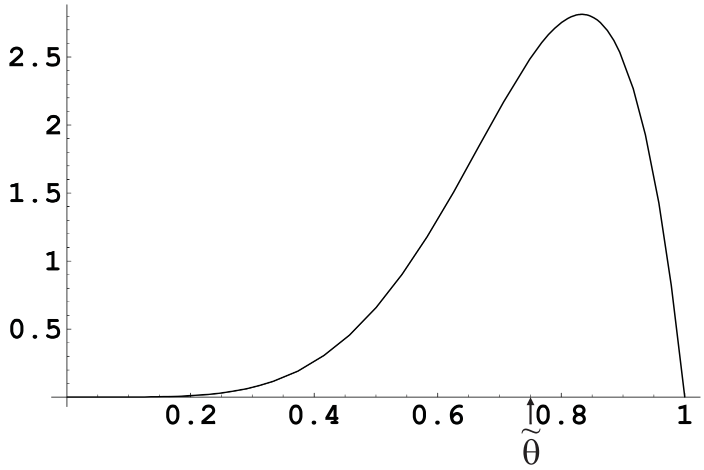


Figure 6.4: Bayesian point estimate for a *Dirichlet*$(6, 2)$ parameter density using KL divergence loss: $\tilde{\theta} = 0.75$.

6.3.4 Bayesian Point Estimation

In the Bayesian learning framework, we maintain a distribution $p(\boldsymbol{\theta})$ over all of the parameters. However, when we are asked to reason using the model, we often wish to "collapse" this distribution over parameters, generate a single representative vector of parameters $\tilde{\boldsymbol{\theta}}$, and answer questions relative to that. If we choose to use $\tilde{\boldsymbol{\theta}}$, whereas the "true" parameters are $\boldsymbol{\theta}^*$, we incur some loss $\mathsf{Loss}(\tilde{\boldsymbol{\theta}} \parallel \boldsymbol{\theta}^*)$[2]. In Bayesian point estimation, our goal is to pick a single vector of parameters that minimize this parameter loss. Of course, we do not have access to $\boldsymbol{\theta}^*$. However, our posterior distribution $p(\boldsymbol{\theta})$ represents our "optimal" beliefs about the different possible values of $\boldsymbol{\theta}^*$, given our prior knowledge and the evidence. Therefore, we can define the *risk* of a particular $\tilde{\boldsymbol{\theta}}$ with respect to p as:

$$E_{\Theta \sim p(\boldsymbol{\theta})} \mathsf{Loss}(\Theta \parallel \tilde{\boldsymbol{\theta}}) = \int_{\boldsymbol{\theta}} \mathsf{Loss}(\boldsymbol{\theta} \parallel \tilde{\boldsymbol{\theta}}) p(\boldsymbol{\theta})\, d\boldsymbol{\theta}. \tag{6.22}$$

We then define the *Bayesian point estimate* to be the value of $\tilde{\boldsymbol{\theta}}$ that minimizes the risk. See Fig. 6.4 for an example.

There are many possible choices of parameter loss functions, but perhaps one of the best

[2]Note that this parameter loss is between two sets of parameters $\tilde{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}^*$. This function should not be confused the active learning **model quality** or **model loss** of our **model**.

justified is the *relative entropy* or *Kullback-Leibler divergence (KL-divergence)* (Kullback & Leibler, 1951; Cover & Thomas, 1991):

$$\mathsf{KL}(\boldsymbol{\theta} \parallel \tilde{\boldsymbol{\theta}}) = \sum_{\mathbf{x}} P_{\boldsymbol{\theta}}(\mathbf{x}) \ln \frac{P_{\boldsymbol{\theta}}(\mathbf{x})}{P_{\tilde{\boldsymbol{\theta}}}(\mathbf{x})}. \tag{6.23}$$

The KL-divergence has several independent justifications, and a variety of properties that make it particularly suitable as a measure of distance between distributions.

Another very common parameter loss function is *Log loss*:

$$\mathsf{LL}(\boldsymbol{\theta} \parallel \tilde{\boldsymbol{\theta}}) = -\sum_{\mathbf{x}} P_{\boldsymbol{\theta}}(\mathbf{x}) \ln P_{\tilde{\boldsymbol{\theta}}}(\mathbf{x}). \tag{6.24}$$

The squared error loss is also commonly used in statistics:

$$\mathsf{L2}(\boldsymbol{\theta} \parallel \tilde{\boldsymbol{\theta}}) = \sum_{k} (\theta_k - \tilde{\theta}_k)^2. \tag{6.25}$$

However, squared error loss is not frequently used with Bayesian networks because it does not possess useful factorization properties that the other two loss functions have.

For all of these parameter loss functions, one can show that the Bayesian point estimate (the value $\tilde{\boldsymbol{\theta}}$ that minimizes the risk relative to p) is the mean value of the parameters:

$$\tilde{\boldsymbol{\theta}} \quad = \quad E_{\Theta \sim p(\boldsymbol{\theta})} \boldsymbol{\theta}. \tag{6.26}$$

Notice that, if we insert this Bayesian point estimate back into the expression for the risk (Eq. (6.22)) this risk expression now just depends upon the parameter density p:

Definition 6.3.4 *The risk of a density is given by:*

$$\mathsf{Risk}(p(\boldsymbol{\theta})) = E_{\Theta \sim p(\boldsymbol{\theta})} \mathsf{Loss}(\Theta \parallel \tilde{\boldsymbol{\theta}}), \tag{6.27}$$

where $\tilde{\boldsymbol{\theta}}$ is the Bayesian point estimate.

We can then observe the following:

- If we are using KL-divergence, then $\mathsf{Risk}(p(\boldsymbol{\theta}))$ is the expected KL-divergence of $\tilde{\boldsymbol{\theta}}$ from the "true" $\boldsymbol{\theta}$.
- If we are using log loss, then $\mathsf{Risk}(p(\boldsymbol{\theta}))$ is the negative expected log likelihood of a future data instance.
- If we are using squared error loss, then $\mathsf{Risk}(p(\boldsymbol{\theta}))$ is the variance of $\boldsymbol{\theta}$.

Finally, notice from Eq. (6.26) that, for all of these parameter loss functions, the Bayesian point estimates are equivalent to the Bayesian one-step predictions. For example, given the **Cancer** network with the same prior (Equations 6.11 to 6.13) and data set (Fig. 6.2) the Bayesian point estimates will be:

$$\tilde{\theta}_{s_0} = E_{\Theta\sim p(\boldsymbol{\theta})}\theta_{s_0} = \frac{17}{22}, \tag{6.28}$$

$$\tilde{\theta}_{c_0|s_0} = E_{\Theta\sim p(\boldsymbol{\theta})}\theta_{c_0|s_0} = \frac{33}{34}, \tag{6.29}$$

$$\tilde{\theta}_{c_0|s_1} = E_{\Theta\sim p(\boldsymbol{\theta})}\theta_{c_0|s_1} = \frac{43}{50}. \tag{6.30}$$

which are identical to Equations 6.19, 6.20 and 6.21.

Chapter 7

Active Learning for Parameter Estimation

"A prudent question is one-half of wisdom."

— Francis Bacon, (1561 - 1626).
English philosopher, statesman, essayist.

7.1 Introduction

The possibility of active learning in Bayesian networks can arise naturally in a variety of ways. In *selective* active learning, we have the ability of explicitly asking for an example of a certain "type"; i.e., we can ask for a full instance where some of the attributes take on requested values. For example, if our domain involves webpages, the learner might be able to ask a human teacher for examples of homepages of graduate students in a Computer Science department. The *pool-based* variant of active learning also arises in many cases. For example, one could redesign the U.S. census to have everyone fill out only the short form; the active learner could then select among the respondents for those that should fill out the more detailed long form. Another example is a cancer study in which we have a list of people's ages and whether they smoke, and we can ask a subset of these people to undergo a thorough examination.

A very different form of active learning arises with Bayesian networks when the learner can ask for experiments involving interventions to be performed. This type of active learning arises naturally in several domains, for example medical diagnosis, microbiology and manufacturing.

In such active learning settings, where we have the ability to actively choose instances on which to train, we need a mechanism that tells us which instances to select. We shall use the general approach that was outlined in Section 1.2 to arrive at a formal framework for active learning of parameters in Bayesian networks. We will assume that the graphical structure of the BN is fixed, and focus on the task of parameter estimation. We will define a notion of a **model** and **model quality**, and provide an algorithm that selects queries in a greedy way, designed to improve model quality as much as possible.

At first sight, the applicability of active learning to density estimation is unclear. After all, if we are trying to estimate a distribution, then random samples from that distribution would seem the best source. Surprisingly, we provide empirical evidence showing that, in a range of interesting circumstances, our approach learns from significantly fewer instances than random sampling.

7.2 Active Learning for Parameter Estimation

Assume that we start out with a network structure $\mathcal{G}$ and a prior distribution $p(\boldsymbol{\theta})$ over the parameters of $\mathcal{G}$. The distribution $p(\boldsymbol{\theta})$ is our **model**. In a standard machine learning framework, data instances are independently, randomly sampled from some underlying distribution. In an active learning setting, we have the ability to request certain types of instances. We formalize this idea by assuming that some subset $\mathcal{C}$ of the variables are *controllable*. The learner can select a subset of variables $\mathbf{Q} \subset \mathcal{C}$ and a particular instantiation $\mathbf{q}$ to $\mathbf{Q}$.

The request $\mathbf{Q} := \mathbf{q}$ is called a *query*. The result of such a query is called the *response* and it is a randomly sampled instance $\mathbf{x}$ of all the *non-query* variables, conditioned on $\mathbf{Q} := \mathbf{q}$. Thus $(\mathbf{q}, \mathbf{x})$ is a *complete* data instance.

The interpretation of such a request depends on our active learning setting. In a *selective* query, we assume that $(\mathbf{q}, \mathbf{x})$ is selected at random from instances satisfying the query.

Hence, $(\mathbf{q}, \mathbf{x})$ is a random instance from $P^*(\mathbf{X} \mid \mathbf{Q} = \mathbf{q})$. (The same statement holds for the pool-based variant of selective active learning.) In an *interventional* query, we assume that our graph $\mathcal{G}$ is a causal model and that $\mathbf{x}$ is the result of an experiment where we intervene in the model and explicitly set the variables in $\mathbf{Q}$ to take the values $\mathbf{q}$.

In the Bayesian network parameter estimation task, an active learner has a querying function that takes $\mathcal{G}$ and $p(\boldsymbol{\theta})$, and selects a query $\mathbf{Q} := \mathbf{q}$. It takes the resulting complete instance $(\mathbf{q}, \mathbf{x})$, and uses it to update its distribution $p(\boldsymbol{\theta})$ to obtain a posterior $p'(\boldsymbol{\theta})$. It then repeats the process, using p' for p. We note that the parameter distribution $p(\boldsymbol{\theta})$ summarizes all the relevant aspects of the data seen so far, so that we do not need to maintain the history of previous instances. To fully specify the algorithm, we need to address two issues: we need to describe how our parameter distribution is updated given that $(\mathbf{q}, \mathbf{x})$ is not a random sample, and we need to construct a mechanism for selecting the next query based on p.

7.2.1 Updating Using an Actively Sampled Instance

Clearly, the answer to the first of these questions depends on the active learning mechanism since the sampling distribution for $(\mathbf{q}, \mathbf{x})$ is different in the two cases.

Let us first consider the case of selective active learning. Assume for simplicity that our query is $Q = q$ for a single node Q. First, it is clear that we cannot use the resulting instance $(\mathbf{q}, \mathbf{x})$ to update the parameters of the node Q itself: the fact that we deliberately sought and found an instance where $Q = q$ does not tell us anything about the overall probability of such instances within the population.

However, we also have a more subtle problem. Consider a parent U of Q. Although $(\mathbf{q}, \mathbf{x})$ does give us information about the distribution of U, it is not information that we can conveniently use. Intuitively, $P(U \mid Q = q)$ is sampled from a distribution specified by a complex formula involving multiple parameters. For example, consider the **Smoking** network: *Cancer* $\rightarrow$ *Smoking* where we must *choose* the value of *Smoking* in advance. It then hard to get a coherent idea of the prior probability of cancer. We sidestep this problem simply by ignoring the information provided by $(\mathbf{q}, \mathbf{x})$ on nodes that are "upstream" of Q.

Definition 7.2.1 *A variable Y is* updateable in the context of a selective query $\mathbf{Q}$ *if it is not in* $\mathbf{Q}$ *or an ancestor of a node in* $\mathbf{Q}$.

Update(p, $\mathbf{Q} := \mathbf{q}$, $\mathbf{x}$)
  For each variable X_i updateable relative to $\mathbf{Q} := \mathbf{q}$
    Let $\mathbf{u}$ be the instantiation of $\mathbf{U}_i$ in $(\mathbf{q}, \mathbf{x})$
    Let x_{ij} be the instantiation of X_i in $\mathbf{x}$
    Set $\alpha'_{x_{ij}|\mathbf{u}} := \alpha_{x_{ij}|\mathbf{u}} + 1$
  Define p' according to $\boldsymbol{\alpha}'$

Figure 7.1: Algorithm for updating p' based on query $\mathbf{Q} := \mathbf{q}$ and response $\mathbf{x}$.

The case of interventional queries is much simpler. Here, each node in the query is forced to have no ancestors, as all of its incoming edges were cut. Thus, for example, the parent U of X in an interventional query $X := x$ is sampled from the original distribution P^*, and hence we can easily use the information about U in $(\mathbf{q}, \mathbf{x})$.

Definition 7.2.2 *A variable Y is* updateable in the context of an interventional query $\mathbf{Q}$ *if it is not in* $\mathbf{Q}$.

For Bayesian parameter estimation, our update rule is now very simple. Given a prior distribution $p(\boldsymbol{\theta})$ and an instance $(\mathbf{q}, \mathbf{x})$ from a query $\mathbf{Q} := \mathbf{q}$, we do standard Bayesian updating, as in the case of randomly sampled instances, but we update only the Dirichlet distributions of updateable nodes. See Fig. 7.1 for the algorithm. We use $p(\boldsymbol{\theta} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ to denote the distribution $p'(\boldsymbol{\theta})$ obtained from this algorithm; this expression can be read as "the density of $\boldsymbol{\theta}$ after performing query $\mathbf{q}$ and obtaining the complete response $\mathbf{x}$". Note that this expression is quite different from the density $p(\boldsymbol{\theta} \mid \mathbf{q}, \mathbf{x})$ which denotes standard Bayesian conditioning.

7.2.2 Applying the General Framework for Active Learning

Our second task is to construct an algorithm for deciding on our next query given our current distribution p. From Section 1.2 our general approach is to define a measure for the **model quality** of our learned **model** $p(\boldsymbol{\theta})$. We can then evaluate the extent to which various instances would improve the quality of our model, thereby providing us with a way to select the next query to perform.

Our formulation for the quality of the model is based on the framework of Bayesian point estimation. We are maintaining a distribution p over our parameters and hence p is our model.

Recall from section 6.3.4 that the Bayesian point estimate is the value of $\tilde{\boldsymbol{\theta}}$ that minimizes the risk and that the risk of a density, $\mathsf{Risk}(p(\boldsymbol{\theta}))$, is the risk using $\tilde{\boldsymbol{\theta}}$ as the estimate.

The risk of our density $p(\boldsymbol{\theta})$ is our measure for the **model quality** of our current state of knowledge, as represented by $p(\boldsymbol{\theta})$.[1] In a greedy scheme, our goal is to obtain an instance $(\mathbf{q}, \mathbf{x})$ such that the risk of the p' obtained by updating p with $(\mathbf{q}, \mathbf{x})$ is lowest. Of course, we do not know exactly which response $\mathbf{x}$ we are going to get. We know only that it will be sampled from a distribution induced by our query. We can, however, consider the expected quality or risk of asking a query. Our *expected posterior risk* is given by:

$$\mathsf{ExPRisk}(p(\boldsymbol{\theta}) \mid \mathbf{Q} := \mathbf{q}) = E_{\Theta \sim p(\boldsymbol{\theta})} E_{\mathbf{x} \sim P_{\boldsymbol{\Theta}}(\mathbf{X} \mid \mathbf{Q} := \mathbf{q})} \mathsf{Risk}(p(\boldsymbol{\theta} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})). \tag{7.1}$$

We have now defined the **model** and **model quality**. When we consider a query $\mathbf{Q} := \mathbf{q}$ we look at the *expected* quality of the posterior model. Using our general template for active learning (Section 1.2) leads immediately to the following simple algorithm: For each candidate query $\mathbf{Q} := \mathbf{q}$, we evaluate the expected posterior risk, and then select the query for which it is lowest.

7.3 Active Learning Algorithm

To obtain a concrete algorithm from the active learning framework shown in the previous section, we must pick a parameter loss function. As we mentioned in Section 6.3.4, although there are many possible choices, perhaps one of the best justified is the KL-divergence (Cover & Thomas, 1991). We therefore proceed using KL-divergence as our

[1] The notions of **model quality** and **model loss** are identical, however to avoid confusion with the *parameter* loss we will only use the term **model quality** in this chapter. To clarify, the parameter loss (e.g., KL-divergence) is used to define the risk of our density, and the risk of our density is our measure of model quality.

parameter loss function. An analogous analysis can be carried through for another very natural parameter loss function: log loss (which corresponds to the negative log-likelihood of future data). In the case of multinomial CPDs with Dirichlet densities over the parameters this alternative parameter loss results in an identical final algorithm. See appendix A.2.2 for details.

7.3.1 The Risk Function for KL-Divergence

We now want to find an efficient approach to computing the model quality which, in the case of parameter estimation, is the risk. Two properties of KL-divergence turn out to be crucial. The first is that the value $\tilde{\boldsymbol{\theta}}$ that minimizes the risk relative to p is the mean value of the parameters, $E_{\Theta \sim p(\boldsymbol{\theta})}\boldsymbol{\theta}$. The second observation is that, for BNs, KL-divergence decomposes with the graphical structure of the network (Heckerman et al., 1995):

$$\mathsf{KL}(\boldsymbol{\theta} \parallel \boldsymbol{\theta}') = \sum_i \mathsf{KL}(P_{\boldsymbol{\theta}}(X_i \mid \mathbf{U}_i) \parallel P_{\boldsymbol{\theta}'}(X_i \mid \mathbf{U}_i)), \tag{7.2}$$

where $\mathsf{KL}(P(X_i \mid \mathbf{U}_i) \parallel P'(X_i \mid \mathbf{U}_i))$ is the *conditional KL-divergence* and is given by $\sum_{\mathbf{u}} P(\mathbf{u})\mathsf{KL}(P(X_i \mid \mathbf{u}) \parallel P'(X_i \mid \mathbf{u}))$. With these two facts, we can prove the following:

Theorem 7.3.1 *Let $\Gamma(\alpha)$ be the Gamma function, $\Psi(\alpha)$ be the* digamma function $\Gamma'(\alpha)/\Gamma(\alpha)$, *and H be the entropy function. Define:*

$$\delta(\alpha_1, \ldots, \alpha_r) = \sum_{j=1}^{r} \left[\frac{\alpha_j}{\alpha_*} \left(\Psi(\alpha_j + 1) - \Psi(\alpha_* + 1)\right) + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*}\right) \right].$$

Then the risk decomposes as:

$$\mathsf{Risk}(p(\boldsymbol{\theta})) = \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \delta(\alpha_{x_{i1}|\mathbf{u}}, \ldots, \alpha_{x_{ir_i}|\mathbf{u}}). \tag{7.3}$$

Proof. See Appendix A.2.

□

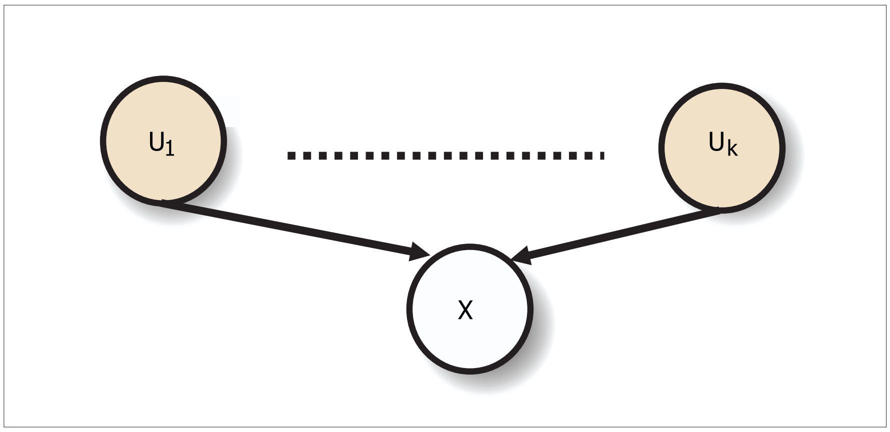


Figure 7.2: Single family. $U_1, \ldots, U_k$ are query nodes.

7.3.2 Analysis for Single CPDs

Eq. (7.3) gives us a concrete expression for evaluating the risk of $p(\boldsymbol{\theta})$. However, to evaluate a potential query, we also need its expected posterior risk. Recall that the expected posterior risk (Eq. (7.1)) is the expectation, over all possible answers to the query, of the risk of the posterior distribution p'. In other words, it is an average over an exponentially large set of possibilities.

To understand how we can evaluate this expression efficiently, we first consider a much simpler case. Consider a BN where we have only one child node X and its parents $\mathbf{U}$, i.e., the only edges are from the nodes $\mathbf{U}$ to X. We also restrict attention to queries where we control all and only the parents $\mathbf{U}$. In this case, a query $\mathbf{q}$ is an instantiation to $\mathbf{U}$, and the possible outcomes to the query are the possible values of the variable X. See Fig. 7.2.

The expected posterior risk contains a term for each variable X_i and each instantiation to its parents. In particular, it contains a term for each of the parent variables U. However, as these variables are not updateable, their hyperparameters remain the same following any query $\mathbf{q}$. Hence, their contribution to the risk is the same in every $p(\boldsymbol{\theta} \mid \mathbf{U} := \mathbf{q}, x)$, and in our prior $p(\boldsymbol{\theta})$. Thus, we can ignore the terms corresponding to the parents, and focus on the terms associated with the conditional distribution $P(X \mid \mathbf{U})$. Hence, we define:

Definition 7.3.2

$$\begin{aligned}\mathsf{Risk}_X(p(\boldsymbol{\theta})) &= \sum_{\mathbf{u}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u})\delta(\alpha_{x_1|\mathbf{u}},\ldots,\alpha_{x_r|\mathbf{u}}), \quad (7.4)\\ \mathsf{ExPRisk}_X(p(\boldsymbol{\theta}) \mid \mathbf{U} := \mathbf{q}) &= \sum_j P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) \sum_{\mathbf{u}} P_{\tilde{\boldsymbol{\theta}}'}(\mathbf{u})\delta(\alpha'_{x_1|\mathbf{u}},\ldots,\alpha'_{x_r|\mathbf{u}}), \quad (7.5)\end{aligned}$$

where $\alpha'_{x_j|\mathbf{u}}$ *is the hyperparameter in* $p(\boldsymbol{\theta} \mid \mathbf{U} := \mathbf{q}, x_j)$ *and* $\tilde{\boldsymbol{\theta}}'$ *is the Bayesian point estimates for the posterior* $p(\boldsymbol{\theta} \mid \mathbf{U} := \mathbf{q}, x_j)$.

Rather than evaluating the expected posterior risk directly, we will evaluate the reduction in risk obtained by asking a query $\mathbf{U} := \mathbf{q}$:

$$\Delta(X \mid \mathbf{q}) = \mathsf{Risk}(p(\boldsymbol{\theta})) - \mathsf{ExPRisk}(p(\boldsymbol{\theta}) \mid \mathbf{q}) = \mathsf{Risk}_X(p(\boldsymbol{\theta})) - \mathsf{ExPRisk}_X(p(\boldsymbol{\theta}) \mid \mathbf{q}).$$

Our next key observation relies on the fact that, since the variables $\mathbf{U}$ are not updateable for this query, their hyperparameters do not change and so $P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u})$ and $P_{\tilde{\boldsymbol{\theta}}'}(\mathbf{u})$ are the same. The final observation is that the hyperparameters for the CPD at node X corresponding to an instantiation $\mathbf{u}$ are the same in p and p' except for $\mathbf{u} = \mathbf{q}$. Hence, terms cancel and the expression simplifies to:

$$P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left(\delta(\alpha_{x_1|\mathbf{q}},\ldots,\alpha_{x_r|\mathbf{q}}) - \sum_j P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q})\delta(\alpha'_{x_1|\mathbf{q}},\ldots,\alpha'_{x_r|\mathbf{q}}) \right).$$

By taking advantage of certain functional properties of Ψ, we obtain the following theorem:

Theorem 7.3.3 *Consider a simple network in which* X *has parents* $\mathbf{Q}$*. Then:*

$$\Delta(X \mid \mathbf{q}) = P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left(H\left(\frac{\alpha_{x_1|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}},\ldots,\frac{\alpha_{x_r|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}\right) - \sum_j P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) H\left(\frac{\alpha'_{x_1|\mathbf{q}}}{\alpha'_{x_*|\mathbf{q}}},\ldots,\frac{\alpha'_{x_r|\mathbf{q}}}{\alpha'_{x_*|\mathbf{q}}}\right)\right), \quad (7.6)$$

where $\alpha_{x_*|\mathbf{q}} = \sum_i \alpha_{x_i|\mathbf{q}}$*. Also,* $\alpha'_{x_i|\mathbf{q}} = (\alpha_{x_i|\mathbf{q}} + 1)$ *if* $i = j$ *and* $\alpha'_{x_i|\mathbf{q}} = \alpha_{x_i|\mathbf{q}}$ *otherwise. So,* $\alpha'_{x_*|\mathbf{q}} = \alpha_{x_*|\mathbf{q}} + 1$.

Proof. See Appendix A.2.

□

If we now select our query $\mathbf{q}$ so as to maximize the difference between our current risk and the expected posterior risk, we get a very natural behavior: We will choose the query $\mathbf{q}$ that leads to the greatest reduction in the entropy (or, alternatively, greatest increase in information) of X given its parents.[2]

It is also here that we can gain an insight as to where active learning may have an edge over random sampling. Consider one situation in which $\mathbf{q}_1$ is 100 times less likely than $\mathbf{q}_2$. Let us suppose that we have previously observed 202 randomly sampled instances, 2 of which are consistent with $\mathbf{q}_1$ and 200 of which are consistent with $\mathbf{q}_2$. If we proceed with random sampling, we are most likely to observe a data instance which is consistent with $\mathbf{q}_2$. Notice that if we were to set $\mathbf{q}_1$, it will lead us to update a parameter whose current density is *Dirichlet*$(1, 1)$, whereas setting $\mathbf{q}_2$ will lead us to update a parameter whose current density is *Dirichlet*$(100, 100)$. According to Δ, updating the former is worth *more* than the latter. Thus, we should be gathering a data instance that is consistent with $\mathbf{q}_1$ rather than $\mathbf{q}_2$. In other words, if we are confident about commonly occurring situations, it is worth more to ask about the rare cases.

7.3.3 Analysis for General BNs

We now generalize this derivation to the case of an arbitrary BN and an arbitrary query. Here, our average over possible query answers encompasses exponentially many terms. Fortunately, we can utilize the structure of the BN to avoid an exhaustive enumeration.

Unfortunately, in the case of general BNs, we can no longer exploit one of our main simplifying assumptions. Recall that, in the expression for the risk (Eq. (7.4)), the term involving X_i and $\mathbf{u}$ is weighted by $P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u})$. In the expected posterior risk, the weight is $P_{\tilde{\boldsymbol{\theta}}'}(\mathbf{u})$. In the case of a single node and a full parent query, the hyperparameters of the parents could not change, so these two weights were necessarily the same. In the more general setting, an instantiation $(\mathbf{q}, \mathbf{x})$ can change hyperparameters all through the network, leading to different weights.

[2]We also note that this rule is very similar to the decision tree splitting rule used in the decision tree learners *ID3* and *C4.5* (Quinlin, 1986). In the decision tree splitting rule we wish to choose an attribute to split a subset of the data on that will provide us with the greatest gain in information of the class label given the splits.

However, we believe that a single data instance will not usually lead to a dramatic change in the distributions. Hence, these weights are likely to be quite close. To simplify the formula (and the associated computation), we therefore choose to approximate the posterior probability $P_{\tilde{\boldsymbol{\theta}}'}(\mathbf{u})$ using the prior probability $P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u})$. Under this assumption, we can use the same simplification as we did in the single node case.

Assuming that this approximation is a good one, we have the following theorem:

Theorem 7.3.4 *The change in risk of a Bayesian network over variables $\mathcal{X}$ when asking query $\mathbf{Q} := \mathbf{q}$ is given by:*

$$\begin{aligned} \Delta(\mathcal{X} \mid \mathbf{q}) &= \mathsf{Risk}(p(\boldsymbol{\theta})) - \mathsf{ExPRisk}(p(\boldsymbol{\theta}) \mid \mathbf{q}) & (7.7) \\ &\approx \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid \mathbf{Q} := \mathbf{q}) \Delta(X_i \mid \mathbf{u}), & (7.8) \end{aligned}$$

where $\Delta(X_i \mid \mathbf{u})$ is as defined in Eq. (7.6). Notice that we actually only need to sum over the updateable X_is since $\Delta(X_i \mid \mathbf{u})$ will be zero for all non-updateable X_is.

Proof. See Appendix A.2.

□

7.4 Algorithm Summary and Properties

The above analysis provides us with an efficient implementation of our general active learning scheme. We simply choose a set of variables in the Bayesian network that we are able to control, and for each instantiation of the controllable variables we compute the expected change in risk given by Eq. (7.7). We then ask the query with the greatest expected change and update the parameters of the updateable nodes. See Fig. 7.3 for the general algorithm.

We now consider the computational complexity of the algorithm. For each potential query $\mathbf{Q} := \mathbf{q}$ we need to compute the expected change in risk. The most expensive computation in evaluating the expected change is risk is computing $P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid \mathbf{Q} := \mathbf{q})$ and $P(\mathbf{u})$ for each instantiation $\mathbf{u}$ of each set of parents $\mathbf{U}_i$. However, as discussed in Section 5.7.2, all of the $P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid \mathbf{Q} := \mathbf{q})$ terms can be found in just one regular join tree inference and all

ActiveLearn(p)
  For each candidate query $\mathbf{Q} := \mathbf{q}$
    Compute the expected change in risk:
      $\sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid \mathbf{Q} := \mathbf{q}) \Delta(X_i \mid \mathbf{u})$
  Ask query $\mathbf{Q} := \mathbf{q}$ with greatest expected change
  Receive complete response $\mathbf{x}$
  $p :=$ **Update**(p, $\mathbf{Q} := \mathbf{q}$, $\mathbf{x}$)
  Repeat

Figure 7.3: Active learning algorithm for parameter estimation in Bayesian networks.

of the $P(\mathbf{u})$ terms can be found in another standard join tree inference. Thus, the run time complexity of the algorithm is: $\mathcal{O}(|\mathcal{Q}| \cdot$ cost of BN join tree inference), where $\mathcal{Q}$ is the set of candidate queries.[3].

Our algorithm (approximately) finds the query that reduces the expected risk the most. Given that we are not simply sampling from the underlying distribution, it is initially unclear that our active learning algorithm learns the correct density. In fact, we can show that our specific querying scheme (including the approximation) is *consistent* – in other words each parameter will tend towards the true θ^* that is generating the data (i.e., the long term relative frequencies of the relevant quantities).

Theorem 7.4.1 *Let $\mathcal{U}$ be the set of nodes which are updateable for at least one candidate query at each querying step. Assuming that the underlying true distribution has the same graphical structure as our network and is not deterministic, then our querying algorithm produces consistent estimates for the CPD parameters of every member of $\mathcal{U}$.*

Proof. See Appendix A.2.

□

The restriction to consistency of updateable nodes in this theorem is quite reasonable. If we have the network $Y \to X$ and we are forced to *always* choose a value for Y then it is impossible to get a consistent estimate for $P(y)$ and it is impossible even if we do

[3]In fact, in some cases we could possibly do even better by modifying the join tree algorithm and evaluating all queries together rather than in separate join tree passes.

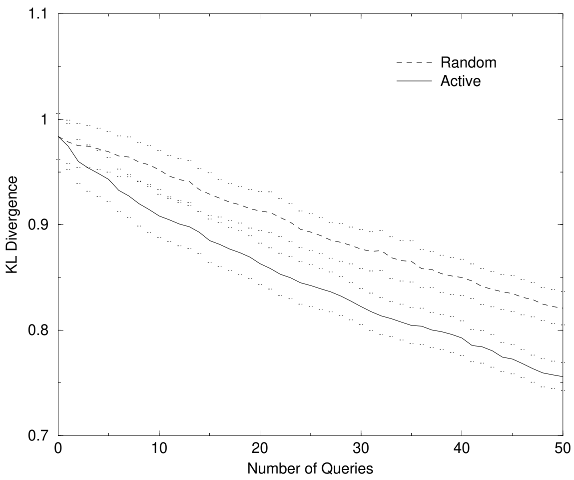
not do active learning and resort to some form of random sampling instead, since we are forced to pick a value for Y. The restriction to considering true distributions that encode at least the same conditional independencies as our graphical structure is also reasonable. If our graph structure consists of just two separate nodes X and Q, it asserts that X and Q are independent. Thus, no matter what we select Q to be, we should be able to use the X instantiations in the resulting data cases to find the marginal distribution of X. This is only a valid step to take if X and Q are independent in the true distribution. Without resorting to maintaining a distribution over possible graph structures, the only general way to consistently find the parameters of the CPD of a node X is to assert no control over the query node Q whatsoever and just sample randomly – and if we are always forced to choose a value for the query node there exists no way to consistently find the parameters for node X since we can't even do random sampling.

7.5 Active Parameter Experiments

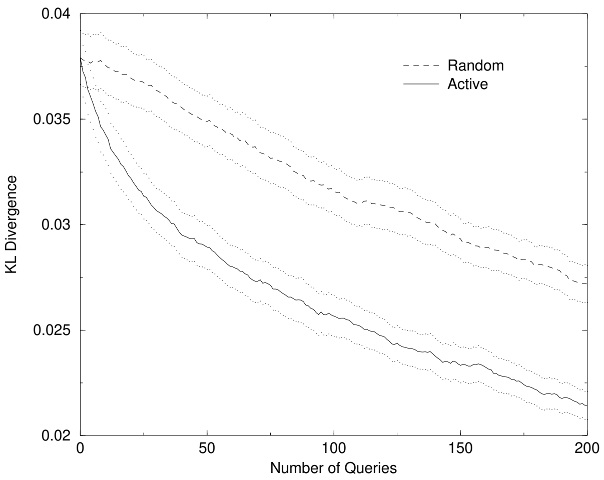
We performed experiments on three commonly used networks: [4] **Alarm**, **Asia** and **Cancer**. **Alarm** has 37 nodes and 518 independent parameters, **Asia** has eight nodes and 18 independent parameters, and **Cancer** has five nodes and 11 independent parameters.

We first needed to set the priors for each network. We use the standard approach (Heckerman et al., 1995) of eliciting a network and an equivalent sample size. In our experiments, we assume that we have fairly good background knowledge of the domain. To simulate this setup, we obtained our prior by sampling a few hundred instances from the true network and used the counts (together with smoothing from a uniform prior) as our prior. This arrangement is akin to asking for a prior network from a domain expert, or using an existing set of complete data to find initial settings of the parameters. We then compared refining the parameters either by using active learning or by random sampling. We permitted the active learner to abstain from choosing a value for a controlled node if it did not wish to; that node is then sampled as usual. For each network we chose some root nodes to be controllable. Controlling the root nodes can be done via *selective* or *interventional* active learning – there is no difference in this case. We also considered a situation where we

[4] e.g., obtainable from www.norsys.com/networklibrary.html



(a)



(b)

Figure 7.4: (a) **Alarm** network with three controllable root nodes. (b) **Asia** network with two controllable root nodes. The axes are zoomed for resolution.

controlled non-root nodes via selective active learning.

We used the true BN to simulate responses to queries asked by the learners. The **random** query method would sample randomly from the entire joint distribution and the **active** method would ask queries.

Figures 7.4 and 7.5 present the results for the three networks. The graphs compare the KL-divergence between the learned networks and the true network that is generating the data.

We see that active learning provides a substantial improvement in all three networks. The improvement in the **Alarm** network is particularly striking given that we had control of just three of the 37 nodes. The extent of the improvement depends on the extent to which queries allow us to reach rare events. For example, *Smoking* is one of the controllable variables in the **Asia** network. In the original network, $P(\mathit{Smoking}) = 0.5$. Although there was a significant gain by using active learning in this network, we found that there was a greater increase in performance if we altered the generating network to have $P(\mathit{Smoking}) = 0.1$; this is the graph that is shown. This increase reinforces our intuition that active learning boosts performance more when there are more pronounced "rare" and "common" cases. We found a similar situation with the **Cancer** network.

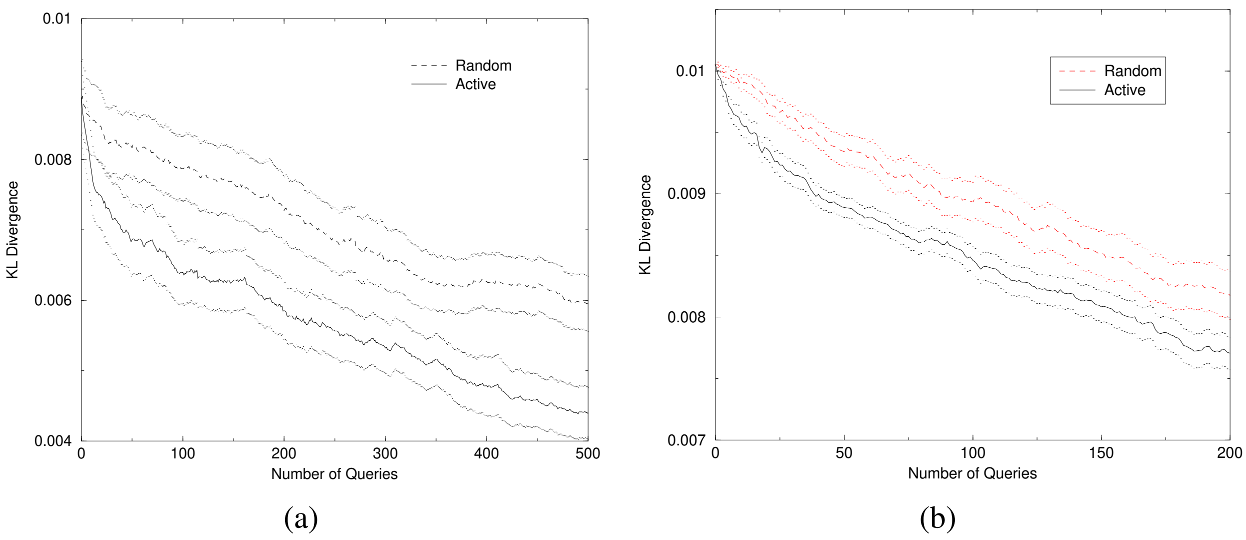


Figure 7.5: (a) **Cancer** network with one controllable root node. (b) **Cancer** network with two controllable non-root nodes using selective querying. The axes are zoomed for resolution.

We also experimented with specifying uniform priors with a small equivalent sample size. Here, we obtained significant benefit in the **Asia** network, and some marginal improvement in the other two. One possible reason is that the improvement is "washed out" by randomness, as the active learner and standard learner are learning from different instances. Another explanation is that the approximation in Eq. (7.7) may not hold as well when the prior $p(\boldsymbol{\theta})$ is uninformed and thereby easily perturbed even by a single instance. This observation indicates that our algorithm may perform best when refining an existing domain model.

We investigated how altering the extent to which a query node influences the rest of the network can affect the performance of our active learning algorithm. Intuitively, if our query nodes $\mathbf{Q}$ have a large influence on the distribution, than there should be more advantage in controling them with active learning. In the *Asia* network, the CPDs for $P(\mathit{Cancer} \mid \mathit{Smoking})$, $P(\mathit{Bronchitis} \mid \mathit{Smoking})$ and $P(\mathit{Tuberculosis} \mid \mathit{VisitAsia})$ are given by:

Smoking	$P(\mathit{cancer}_0 \mid \mathit{Smoking})$	$P(\mathit{cancer}_1 \mid \mathit{Smoking})$
$\mathit{smoking}_0$	0.1	0.9
$\mathit{smoking}_1$	0.01	0.99

Smoking	$P(\mathit{bronchitis}_0 \mid \mathit{Smoking})$	$P(\mathit{bronchitis}_1 \mid \mathit{Smoking})$
$\mathit{smoking}_0$	0.05	0.95
$\mathit{smoking}_1$	0.01	0.99

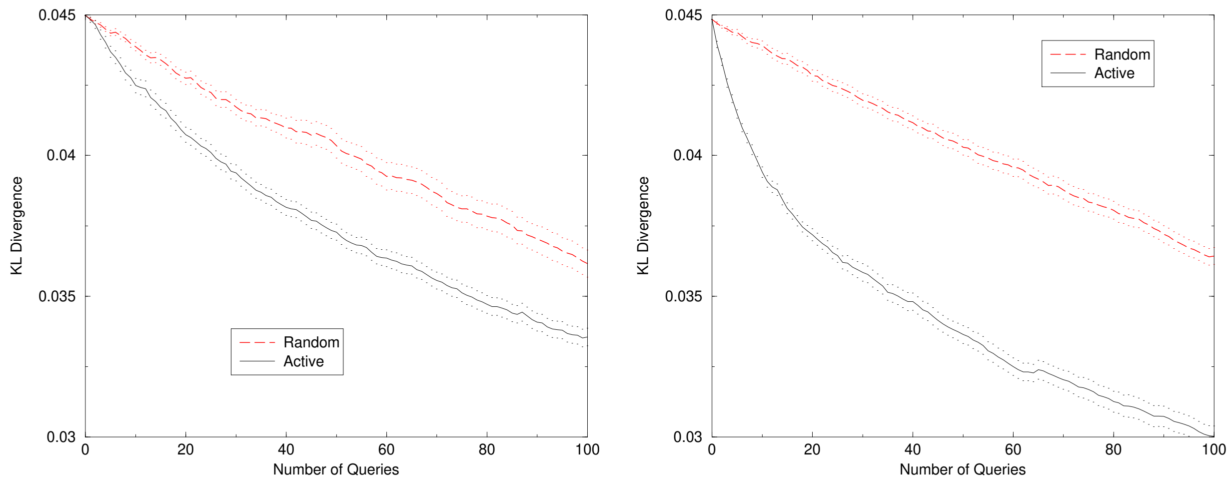
VisitAsia	$P(\mathit{tuberculosis}_0 \mid \mathit{VisitAsia})$	$P(\mathit{tuberculosis}_1 \mid \mathit{VisitAsia})$
$\mathit{visitAsia}_0$	0.6	0.4
$\mathit{visitAsia}_1$	0.3	0.7

We altered the extent to which *VisitAsia* and *Smoking* influence the distribution by modifying these CPDs. We created a new CPD for $P(\mathit{Cancer} \mid \mathit{Smoking})$ by having a mixture of the original CPD and:

Smoking	$P(\mathit{cancer}_0 \mid \mathit{Smoking})$	$P(\mathit{cancer}_1 \mid \mathit{Smoking})$
$\mathit{smoking}_0$	1	0
$\mathit{smoking}_1$	0	1

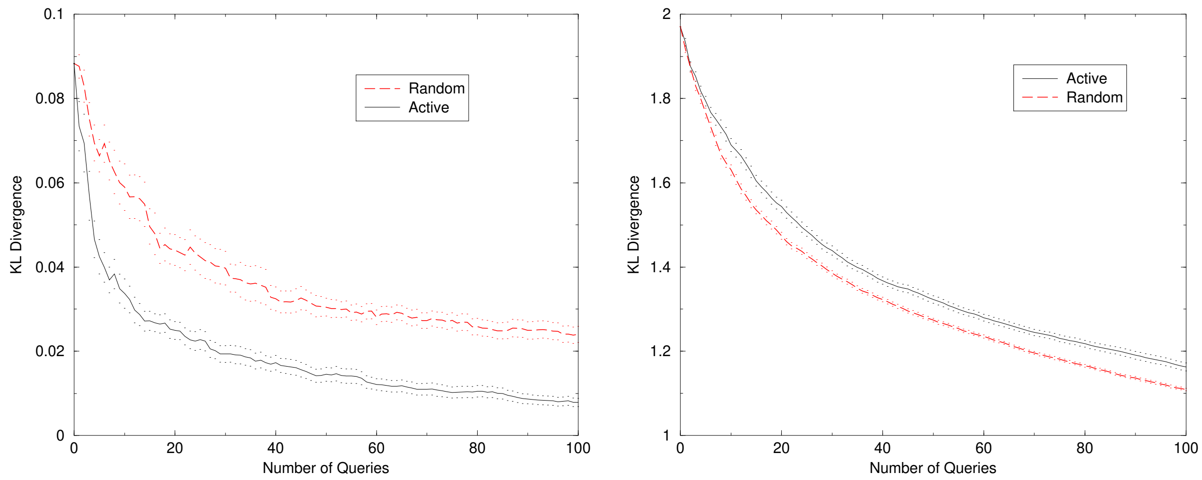
So, for example, $P^{(\text{new})}(\mathit{cancer}_0 \mid \mathit{smoking}_0) = (1 - \lambda) \times 0.1 + \lambda \times 1$. Thus, the closer the mixture component λ is to 1, the greater the difference between the distrubutions $P(\mathit{Cancer} \mid \mathit{smoking}_0)$ and $P(\mathit{Cancer} \mid \mathit{smoking}_1)$. We did a similar transformation for the other two CPDs. The parameter λ can be regarded as the "degree of control" for the *Smoking* and *VisitAsia* nodes. Fig. 7.6 shows the effect that changing λ has on the performance of the active learning algorithm. It shows that the more control the active learning algorithm has, the greater the gain in performance over random sampling.

Overall, we found that, in almost all situations, active learning performed as well as



(a) (b)

Figure 7.6: (a) **Asia** network with $\lambda = 0.3$. (b) **Asia** network with $\lambda = 0.9$. The axes are zoomed for resolution.



(a) (b)

Figure 7.7: (a) **Cancer** network with a "good" prior. (b) **Cancer** network with a "bad" prior. The axes are zoomed for resolution.

or better than random sampling. The situations where active learning produced most benefit were, unsurprisingly, those in which the prior was confident and correct about the commonly occurring cases and uncertain and incorrect about the rare ones (Fig. 7.7(a)). Clearly, this is the precisely the scenario we are most likely to encounter in practice when the prior is elicited from an expert or obtained from randomly sampled data. By experimenting with forcing different priors we found that active learning was worse in one type of situation: where the prior was confident yet incorrect about the commonly occurring cases and uncertain but actually correct about the rare ones (Fig. 7.7(b)). This type of scenario is unlikely to occur in practice.

Chapter 8

Structure Learning

8.1 Introduction

The task of causal structure discovery from empirical data is a fundamental problem in many areas. In Section 5.6, we saw that Bayesian networks could be used provide a causal model of a domain. If we assume that the graphical structure of some BN represents the causal structure of the domain, we can formalize the problem of discovering the causal structure of the domain as the task of learning the BN structure from data. This chapter reviews the standard techniques used for structure learning.

Over the last few years, there has been substantial work on discovering BN structure from purely observational data (Heckerman, 1998). However, there are inherent limitations on our ability to discover the structure based on randomly sampled data. Randomly sampled data will only enable us, in the limit, to recover the Markov equivalence class (see Section 5.4) of the underlying structure. Experimental data, where we intervene in the model, is vital for a full determination of the causal structure. By observing the results of these experiments, we can determine the direction of causal influence in cases where purely observational data is inadequate. The problem of uncovering the causal structure from observational and experimental data has been tackled in a non-active learning setting by Heckerman (1995) and Cooper and Yoo (1999). Furthermore, although Cooper and Yoo derived a closed-form scoring metric for full networks,[1] they only apply their technique to learn the relationship between single pairs of variables which they further assume are not confounded (do not have a common cause)[2].

8.2 Structure Learning in Bayesian Networks

Our goal is to learn the causal structure from data. In order to do this, we need to make a number of standard assumptions. We assume that there are no hidden variables and we make two further assumptions:

- **Causal Markov assumption:** The data is generated from an underlying causal Bayesian network $(\mathcal{G}^*, \boldsymbol{\theta}^*)$ over $\mathcal{X}$.
- **Faithfulness assumption**: The distribution P^* over $\mathcal{X}$ induced by $(\mathcal{G}^*, \boldsymbol{\theta}^*)$ satisfies no independencies beyond those implied by the structure of $\mathcal{G}^*$.

Our goal is to reconstruct $\mathcal{G}^*$ from the data. Clearly, given enough data, we can reconstruct P^*. However, in general, P^* does not uniquely determine $\mathcal{G}^*$. For example, if our network $\mathcal{G}^*$ has the form $X \rightarrow Y$, then $Y \rightarrow X$ is equally consistent with P^*. Given only samples from P^*, the best we can hope for is to identify the Markov equivalence class of $\mathcal{G}$: a set of network structures that induce precisely the same independence assumptions (see Section 5.4).

If we are given experimental as well as observational data, our ability to identify the structure is much larger (Cooper & Yoo, 1999). Intuitively, assume we are trying to determine the direction of an edge between X and Y. If we are provided with experimental data that intervenes at X, and we see that the distribution over Y does not change, while intervening at Y does change the distribution over X, we can conclude (based on the assumptions above) that the edge is $Y \rightarrow X$.

[1]They derived a closed for expression for the probability of a structure given the experimental and observational data.

[2]In the next chapter we show that, even setting aside the active learning aspects of our work, our framework permits combining observational and experimental data for learning the structure over *all* variables in our domain, allowing us to distinguish the structure at a much finer level, taking into consideration both indirect causation and confounding influences.

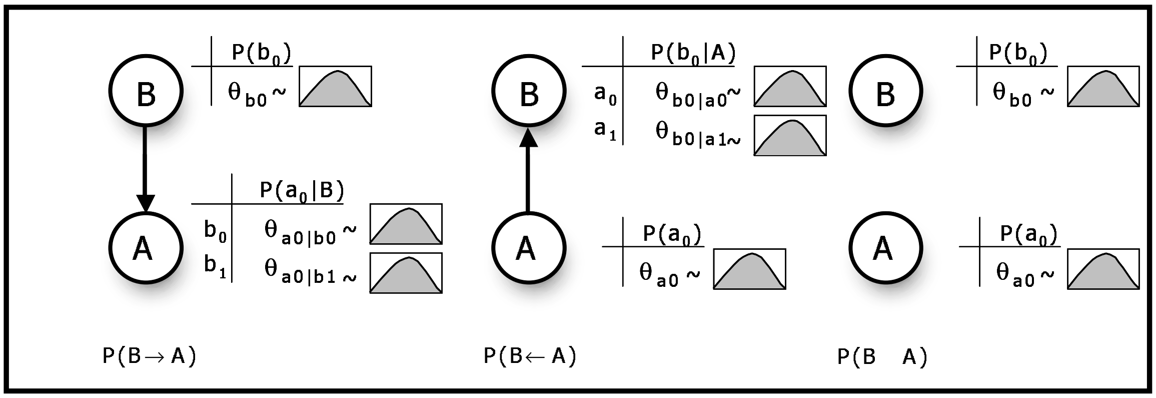


Figure 8.1: A distribution over networks and parameters.

8.3 Bayesian approach to Structure Learning

There are a number of techniques for performing structure learning. One common approach is to perform *model selection*, where we search for a good single representative structure and then perform all subsequent analyses and inferences with respect to that single structure. There are a number of criteria for selecting a good structure (Heckerman, 1998), for example: maximum likelihood, minimum description length and Bayesian marginal likelihood. An alternative approach, and one which we use to guide our active learning algorithm, is the full Bayesian framework. Rather than committing to a single structure, in the full Bayesian framework we keep a distribution and perform all inferences with respect to the entire distribution.

We now describe how to represent and maintain a distribution for structure learning. We maintain a distribution over the set of structures and their associated parameters (see Fig. 8.1 for a simple illustration on a two variable domain). We begin with a prior over structures and parameters, and use Bayesian conditioning to update it as new data is obtained. Following (Heckerman et al., 1995), we make several standard assumptions about the prior:

- **Structure Modularity**: The prior $P(\mathcal{G})$ can be written in the form:

$$P(\mathcal{G}) = \prod_i P(Pa(X_i) = \mathbf{U}_i^{\mathcal{G}}). \tag{8.1}$$

Thus, the *a priori* choices of the families for the different nodes are independent.

- **Parameter Independence**:

$$p(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathcal{G}) = \prod_i \prod_{\mathbf{u}} p(\boldsymbol{\theta}_{X_i|\mathbf{u}} \mid \mathcal{G}). \tag{8.2}$$

In other words, as in Chapter 6, we can decompose the joint density over the vector of network CPD parameters as a product of localized densities.

- **Parameter Modularity**: For two graphs $\mathcal{G}$ and $\mathcal{G}'$, if $\mathbf{U}_i^{\mathcal{G}} = \mathbf{U}_i^{\mathcal{G}'}$ then:

$$p(\boldsymbol{\theta}_{X_i|\mathbf{U}_i^{\mathcal{G}}} \mid \mathcal{G}) = p(\boldsymbol{\theta}_{X_i|\mathbf{U}_i^{\mathcal{G}'}} \mid \mathcal{G}'). \tag{8.3}$$

Thus, the density for each parameter only depends upon the network structure that is local to the variable that the parameter is over.

We also assume that the CPD parameters are multinomials and that the associated parameter distributions are the conjugate Dirichlet distributions where we denote the Dirichlet hyperparameter for the parameter corresponding to the j-th value of X_i given parents $\mathbf{u}$, $\theta_{x_{ij}|\mathbf{u}}$, by $\alpha_{x_{ij}|\mathbf{u}}$. However, some of our analysis holds for any distribution satisfying the parameter independence and parameter modularity assumptions. There are a number of different ways to choose the structure prior (Buntine, 1991; Heckerman, 1998), although a common choice for the structure prior is a uniform prior over structures.

Given these assumptions, we can represent a distribution over structures and parameters $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ by maintaining a structure prior component $P(\mathit{Pa}(X_i) = \mathbf{U})$ for each valid (node, parents) pair $(X_i, \mathbf{U})$ and a Dirichlet distribution for each node X_i and each instantiation of each the possible sets of parents. Now, given a particular prior distribution over Bayesian network structures and parameters, when we receive a new data instance (either observational or experimental) we update our distribution P to obtain the posterior distribution P'. We then use P' as our new distribution over structures and parameters. We next show how to update the prior distribution with a data instance so as to obtain the posterior distribution.

8.3.1 Updating using Observational Data

Given a complete, randomly sampled single instance $\mathbf{d}$ over $\mathcal{X}$, we define how to update the distribution $P(\mathcal{G}, \boldsymbol{\theta}_\mathcal{G})$. We break this distribution into two problems by using the identity:

$$P(\mathcal{G}, \boldsymbol{\theta}_\mathcal{G} \mid \mathbf{d}) = p(\boldsymbol{\theta}_\mathcal{G} \mid \mathbf{d}, \mathcal{G}) \cdot P(\mathcal{G} \mid \mathbf{d}).$$

Thus we need to determine how to update the parameter density of a structure and also how to update the distribution over structures themselves.

For the first term in this expression, consider a particular network structure $\mathcal{G}$ and a prior distribution $p(\boldsymbol{\theta}_\mathcal{G})$ over the parameters of $\mathcal{G}$. To obtain the posterior distribution over the parameters, $P(\boldsymbol{\theta}_\mathcal{G} \mid \mathbf{d}, \mathcal{G})$, we simply use standard Bayesian updating of each of the Dirichlet parameter distributions associated with this graph as described in Section 6.3. Note that this updating still preserves parameter modularity and parameter independence.

Now consider the distribution over structures $P(\mathcal{G})$. We need to compute the posterior distribution over structures $P(\mathcal{G} \mid \mathbf{d})$. We first introduce the following definition:

Definition 8.3.1 *Let X_i be a node and $\mathbf{U}$ be its parents. We define the score of a family as:*

$$\mathit{Score}(X_i, \mathbf{U} \mid \mathbf{d}) = \int P(x_i \mid \mathbf{u}, \boldsymbol{\theta}_{X_i|\mathbf{u}}) p(\boldsymbol{\theta}_{X_i|\mathbf{u}})\, d\boldsymbol{\theta}_{X_i|\mathbf{u}} = P(x_i \mid \mathbf{u}).$$

The following well-known theorem (Heckerman, 1998) tells us how we can compute $P(\mathcal{G} \mid \mathbf{d})$:

Theorem 8.3.2 *Given a complete data instance $\mathbf{d}$, if $P(\mathcal{G}, \boldsymbol{\theta}_\mathcal{G})$ satisfies* structure modularity*,* parameter independence *and* parameter modularity*, then:*

$$P(\mathcal{G} \mid \mathbf{d}) = \frac{1}{P(\mathbf{d})} \prod_i P(\mathit{Pa}(X_i) = \mathbf{U}_i^\mathcal{G}) \mathit{Score}(X_i, \mathbf{U}_i^\mathcal{G} \mid \mathbf{d}).$$

Notice that $P(\mathbf{d})$ is just a normalizing factor which is independent of the graph structure and parameters that we are considering, and so it can be ignored. Also, notice that, just like the prior, the posterior distribution over structures also obeys structure modularity, i.e., it is also a product of terms, one for each family. To obtain the posterior, we essential just need to multiple the term in the prior corresponding to each family by the score for that family.

We are using multinomial CPDs with Dirichlet distributions over the parameters. The following standard result (Heckerman, 1998) shows us how to compute the score in this case:

Theorem 8.3.3 *Let* $\mathbf{d}$ *be a complete data instance. For multinomial CPDs with Dirichlet distributions over the parameters we have:*

$$\mathit{Score}(X_i, \mathbf{U} \mid \mathbf{d}) = \frac{\Gamma(\alpha_{x_{i*}|\mathbf{u}})}{\Gamma(\alpha_{x_{i*}|\mathbf{u}} + 1)} \cdot \frac{\Gamma(\alpha_{x_{ij}|\mathbf{u}})}{\Gamma(\alpha_{x_{ij}|\mathbf{u}} + 1)}, \tag{8.4}$$

where $\mathbf{u}$ *and* x_{ij} *are the values of* $\mathbf{U}$ *and* X_i *in* $\mathbf{d}$ *and* $\alpha_{x_{i*}|\mathbf{u}} = \sum_j \alpha_{x_{ij}|\mathbf{u}}$.

Thus, given a data instance $\mathbf{d}$, to update the prior distribution $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ to obtain the posterior $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{d})$, we need to update the hyperparameters of all of the Dirichlet distributions by using Eq. (6.7) and we need to update the $P(\mathit{Pa}(X_i) = \mathbf{U}_i)$ components of the modular structure prior. The updated components of the structure prior are computed by:

$$P(\mathit{Pa}(X_i) = \mathbf{U}_i \mid \mathbf{d}) = C \cdot P(\mathit{Pa}(X_i) = \mathbf{U}_i) \cdot \mathit{Score}(X_i, \mathbf{U}_i \mid \mathbf{d}), \tag{8.5}$$

where C is a normalizing constant. In all of the computations that we perform in this thesis, we can ignore the normalizing constant since the constant will either cancel out, or will just require us to perform a simple re-normalization step at the end.

8.3.2 Updating using Experimental Data

Now, instead of having a complete random instance, suppose that we have an experiment, or query, that sets $\mathbf{Q} := \mathbf{q}$, and are given the resulting response $\mathbf{x}$. We need to define how to update the distribution $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ given this query and response. As before, we decompose this problem into two subproblems by using the identity:

$$P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}) = p(\boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \mathcal{G}) \cdot P(\mathcal{G} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}).$$

For the first term in this expression, given structure $\mathcal{G}$ and a prior distribution $p(\boldsymbol{\theta}_{\mathcal{G}})$ over the parameters of $\mathcal{G}$, our update rule for the parameter density is identical to the procedure for interventional queries described in Section 7.2.1. In other words, we perform regular Bayesian updating for all of the parameter densities associated with the non-query nodes. We also note that performing such an interventional update to the parameters still preserves parameter modularity.[3]

Now consider the distribution over structures. We use $P(\mathcal{G} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ to denote the posterior distribution over structures after performing the query and obtaining the response. The following theorem tells us how we can easily update the posterior over $\mathcal{G}$ given an interventional query:

Theorem 8.3.4 (Cooper and Yoo, 1999) *Given a query* $\mathbf{Q} := \mathbf{q}$ *and complete response* $\mathbf{x}$*, if* $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ *satisfies* parameter independence *and* parameter modularity*, then:*

$$P(\mathcal{G} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}) = \frac{1}{P(\mathbf{x} \mid \mathbf{Q} := \mathbf{q})} \prod_{i : X_i \notin \mathbf{Q}} P(\mathit{Pa}(X_i) = \mathbf{U}_i^{\mathcal{G}}) \mathit{Score}(X_i, \mathbf{U}_i^{\mathcal{G}} \mid \mathbf{x}, \mathbf{q}).$$

After we have seen a query $\mathbf{Q} := \mathbf{q}$ and response $\mathbf{x}$, we can use the updated distribution $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ as our new "prior" distribution. As in the observational case, notice that the posterior distribution over structures maintains the structure modularity condition, and that updating the Dirichlet parameter distributions preserves parameter independence and parameter modularity.

To summarize, to update our distribution $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$, we update the hyperparameters of the set of Dirichlet distributions that we are maintaining. The updated components of the structure prior for $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ are computed by:

$$P(\mathit{Pa}(X_i) = \mathbf{U}_i \mid \mathbf{Q} := \mathbf{q}) = C \cdot P(\mathit{Pa}(X_i) = \mathbf{U}_i) \cdot \mathit{Score}(X_i, \mathbf{U}_i \mid \mathbf{x}, \mathbf{q}), \qquad (8.6)$$

[3] Notice that we are assuming *interventional* queries. If we were to use *selective* queries then parameter modularity no longer holds. Recall that, given a selective query $\mathbf{Q} := \mathbf{q}$ and response, the variable Y in a graph is updateable if it is not an ancestor of $\mathbf{Q}$. But this ancestor-of-$\mathbf{Q}$ property is dependent upon the graph. So for some graphs, Y will be updateable and for others it will not. Thus, the parameter modularity assumption is violated when we observe selective data which, therefore, makes the task of representing and updating the distribution $P(\mathcal{G}, \theta_{\mathcal{G}})$ extremely hard.

where C is a normalizing constant and $\mathit{Score}(X_i, \mathbf{U}_i^{\mathcal{G}} \mid \mathbf{x}, \mathbf{q}) = 1$ if X_i is a query variable. As we mentioned in the previous section, in all of the computations that we perform in this thesis, we can ignore the normalizing constant.

8.4 Computational Issues

To represent a distribution over structures and parameters, we need to maintain a Dirichlet distribution for each node X_i and each instantiation of each the possible sets of parents. Similarly, we need to maintain the structure prior components $P(\mathit{Pa}(X_i) = \mathbf{U}_i)$ for each node and parent set. In practice this is often infeasible. The number of Dirichlet distributions and structure prior components grow exponentially with the number of variables in our domain. Instead, we can implicitly maintain these Dirichlet and structure component distributions by storing the data $\mathbf{D}$ that we have collected, and then only reconstruct the desired quantities (such as $P(\mathit{Pa}(X_i) = \mathbf{U}_i \mid \mathbf{D})$) when required by applying the update formulae mentioned in the previous two sections. Furthermore, all of the update formulae generalize to take into account multiple observations (Heckerman, 1998; Cooper & Yoo, 1999). Hence, if we are implicitly maintaining the distribution over graphs and parameters and have seen, say, five past data instances, then rather than using the single-instance update formula five times to reconstruct $P(\mathit{Pa}(X_i) = \mathbf{U}_i \mid \mathbf{D})$, we need only perform one generalized update step.

Chapter 9

Active Learning for Structure Learning

"The art of discovering the causes of phenomena,
or true hypothesis, is like the art of deciphering,
in which an ingenious conjecture greatly shortens the road."

— Gottfried Whilhem Leibniz, (1646-1716).
New Essays Concerning Human Understanding, IV, XII.

9.1 Introduction

Experimental data is crucial for determining the underlying causal structure of a domain. However, obtaining experimental data is often time consuming and costly. Thus the experiments must be chosen with care. Our goal is not merely to update the distribution over causal Bayesian networks based on experimental data. We want to *actively* choose instances that will allow us to learn the structure better.

We provide an active learning algorithm that selects interventional experiments that are most informative towards revealing the causal structure. We present a formal framework for active learning of causal structure in Bayesian networks, based on the principles of Bayesian learning. Our **model** is a distribution over Bayesian network structures, which is updated based on our data. We define a notion of **quality** of our **model**, and provide an algorithm that selects queries in a greedy way, designed to improve model quality as much as possible. We provide experimental results on a variety of domains, showing that our active learning algorithm can provide substantially more accurate estimates of the BN structure using the same amount of data. Interestingly, our active learning algorithm provides significant improvements even in cases where it cannot intervene in the model, but only select instances of certain types. Thus, it is applicable even to the problem of learning structure in a non-causal setting.

9.2 General Framework

Our goal is to use active learning to learn the BN structure – learning from data where we are allowed to control certain variables by intervening at their values. As in the active parameter case, we have some subset $\mathcal{C}$ of the variables that are controllable. The learner can select a subset of variables $\mathbf{Q} \subset \mathcal{C}$ and a particular instantiation $\mathbf{q}$ to $\mathbf{Q}$. We use the same notion of an *interventional query* as before, and the result of a query $\mathbf{Q} := \mathbf{q}$ is the response $\mathbf{x}$ which is a randomly sampled instance of all the non-query variables, conditioned on $\mathbf{Q} := \mathbf{q}$. We do not consider selective queries for structure estimation. In addition to the computational complications mentioned in Section 8.3.2, the value of selective queries is far less than those of interventional queries. As with randomly sampled data, they do not intervene in the domain and, hence, they only permit us to resolve up to the Markov equivalence class, rather than determine the full causal structure of the network.

For the case of causal structure learning, the querying function in an active learner selects an interventional query $\mathbf{Q} := \mathbf{q}$ based upon its current distribution over $\mathcal{G}$ and $\boldsymbol{\theta}_{\mathcal{G}}$. It takes the resulting response $\mathbf{x}$, and uses it to update its distribution over $\mathcal{G}$ and $\boldsymbol{\theta}_{\mathcal{G}}$. It then repeats the process. We described the update process in the previous chapter. Our task now is to construct an algorithm for deciding on our next query given our current distribution over structures and paramters $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$.

Our distribution over graphs and parameters will be our **model**. We shall need to define its **model quality**. To this end, we will define a *model loss function* as the notion of model quality. We can then use this measure of quality to evaluate the extent to which various instances would improve the quality of our distribution, thereby providing us with an approach for selecting the next query to perform.

More formally, given a distribution over graphs and parameters $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ we have a *model loss function* $\mathit{Loss}(P)$ that measures the **model quality** of our distribution over the graphs and parameters. Given a query $\mathbf{Q} := \mathbf{q}$ we define the *expected posterior loss* of the query as:

$$\begin{aligned} & \mathit{ExPLoss}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}) \mid \mathbf{Q} := \mathbf{q}) \\ = \quad & E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q})} \mathit{Loss}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})). \end{aligned} \tag{9.1}$$

Applying our general approach (Section 1.2) for active learning we have the following algorithm: for each candidate query $\mathbf{Q} := \mathbf{q}$, we evaluate the expected posterior loss, and then select the query for which it is lowest.

Although this idea seems good in principle, note that the expected loss appears to be very computationally expensive to evaluate. We need to maintain a distribution over the set of structures, and the size of this set is super-exponential in the number of nodes. Furthermore, given a query, to compute the expected posterior loss we have to perform a computation over the set of structures for each of the exponential number of possible responses to the query.

One possibility is to approximate the distribution over structures by sampling a number of them. We could then compute the expected posterior loss of asking a query with respect to this representative set of structures by sampling possible responses to the query. However, this method has serious shortcomings. First, the distribution over structures is often very "uneven", requiring a great many sample structures to approximate it to a reasonable degree (Friedman & Koller, 2000). Second, the effect of any single query and response on the distribution over structures is very small (since it is merely a single data instance) and so it is very likely that the small difference in effect of asking different queries will be overwhelmed by the variance introduced by the sampling of structures and completions.

Ideally, we would like to have a close form, yet efficiently computable expression for the expected posterior loss of asking a query. We show that, to some degree, this is possible to achieve.

9.3 Loss Function

To make the high-level active learning framework concrete, we must first pick a model loss function. We wish the model loss function to reflect our goal of determining the causal structure, and we also wish it to decompose in such a way that we can evaluate it efficiently. Our goal is to be as certain about the network structure as possible. Thus, one natural choice of model loss functions for a model $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$ is the entropy of the marginal distribution over graphs: $H(P(\mathcal{G}))$. It can be shown (Chaloner & Verdinelli, 1995) that the expected posterior loss using this model loss criterion corresponds to the *D-optimality* criterion used in optimal experimental design for linear regression. Unfortunately, $H(P(\mathcal{G}))$ does not have useful decomposition properties (for example, it does not break down into a sum or product of localized terms) and so computing it in closed form for each and every structure and query response is intractable.

However, a reasonable alternative can be found that is computationally tractable. Recall that our goal is to learn the correct structure; hence, we are interested in the presence and direction of the edges in the graph. For two nodes X_i and X_j, there are three possible edge relationships between them: either $X_i \rightarrow X_j$, or $X_i \leftarrow X_j$ or $X_i \quad X_j$. Our distribution P over graphs and parameters induces a distribution over these three possible edge relationships. We can measure the extent to which we are sure about this relationship using the entropy of this induced distribution:

$$\begin{aligned} H(X_i \leftrightarrow X_j) = & -P(X_i \rightarrow X_j) \log P(X_i \rightarrow X_j) \\ & -P(X_i \leftarrow X_j) \log P(X_i \leftarrow X_j) \\ & -P(X_i \quad X_j) \log P(X_i \quad X_j). \end{aligned} \tag{9.2}$$

The larger this entropy, the less sure we are about the relationship between X_i and X_j. This expression forms the basis for our *edge entropy* model loss function:

$$\mathit{Loss}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})) = \sum_{i,j} H(X_i \leftrightarrow X_j). \tag{9.3}$$

In certain domains we may be especially interested in determining the relationship between particular pairs of nodes. We can reflect this desire in our model loss function by introducing scaling factors in front of different $H(X_i \leftrightarrow X_j)$ terms. In Section 9.5, the fact that this loss function decomposes as a sum of local terms permits us to efficiently evaluate the expected posterior loss of a query.

Now that we have defined the **quality** for a **model** $P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}})$, our task is to find an efficient algorithm for computing the expected posterior loss of a given query $\mathbf{Q} := \mathbf{q}$ relative to P. We note that P is our current distribution, conditioned on all the data obtained so far. Initially, it is the prior; as we get more data, we use Bayesian conditioning (as described in Chapter 8) to update P, and then apply the same algorithm to the posterior.

Our approach to obtaining a tractable algorithm is based on the ideas of Friedman *et al.* (1999) and Friedman and Koller (2000). First, we restrict the set of possible parents of a node during each querying round. Second, we consider the simpler problem of restricting attention to network structures consistent with some total ordering, $\prec$; then, we relax this restriction by introducing a distribution over the orderings.

9.4 Candidate Parents

Following Friedman *et al.* (1999), we assume that each node X_i has a set $\mathbf{W}_i$ of at most m possible *candidate* parents that is fixed before each query round. In certain domains, we can use prior knowledge to construct $\mathbf{W}_i$. In other domains we can use a technique discussed by Friedman *et al.* (1999) where we can use randomly sampled observational data to point out nodes that are more likely to be directly related to X_i: one way to do this is to choose the m variables which have the highest individual *mutual information* with X. The mutual information (Cover & Thomas, 1991) between two variables X_i and X_j is given by the following expression:

$$MI(X_i; X_j) = \sum_{x_i, x_j} P(x_i, x_j) \ln \frac{P(x_i, x_j)}{P(x_i)P(x_j)}. \tag{9.4}$$

For this computation it is reasonable to use the maximum likelihood estimates for $P(x_i, x_j)$ since we are just estimating the distribution over two variables and only wish to determine the pairs of nodes that have the highest mutual information. However, we may choose to use any other form of estimator if we desire.[1]

9.5 Analysis for a Fixed Ordering

Let $\prec$ be a total ordering of $\mathcal{X}$. We restrict attention to network structures that are consistent with $\prec$, i.e., if there is an edge $X \to Y$, then $X \prec Y$. We also assume that, given $\prec$, the structure prior $P(\mathcal{G} \mid\prec)$ is modular. We note that, from Chapter 8, given data $\mathbf{Q} := \mathbf{q}, \mathbf{x}$, the posterior $P(\mathcal{G} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec)$ will also then be modular.

Recall that each node X_i has a set of at most m candidate parents $\mathbf{W}_i$. We define the set of candidate parents for a node X_i that are consistent with our ordering as:

$$\mathcal{U}_{i,\prec} = \{\mathbf{U} : \mathbf{U} \prec X_i, \mathbf{U} \subseteq \mathbf{W}_i\},$$

where $\mathbf{U} \prec X_i$ is defined to hold when all nodes in $\mathbf{U}$ precede X_i in $\prec$.

We note that the number of structures induced by $\prec$ and by having a set of candidate parents $\mathbf{W}_i$ for each node X_i is still exponential in the number of variables in $\mathcal{X}$, even when we hold the maximum number of candidate parents constant. The key impact of the restriction to a fixed ordering is that the choice of parents for one node is independent of the choice of parents for another node (Buntine, 1991; Friedman & Koller, 2000). Three important consequences are the following two theorems and corollary, which give us closed form, efficiently computable expressions for key quantities:

[1] Unfortunately, we cannot use *interventional* data to estimate the mutual information between two nodes. If we are using interventional data, we may produce inconsistent estimates for the mutual information. This inconsistency is not just because we are forcing the values of some nodes thus affecting the correlation between a forced node and other variables. The problem is more subtle, and even affects the estimate of the mutual information between two nodes that are non-query nodes. For example, suppose the true network is $X \to Y \leftarrow Z$ and the CPD of Y is such that if $Z = z_0$ then X and Y are independent, and if $Z = z_1$ then X and Y are totally dependent (i.e., deterministic). Also, suppose that $P(z_0) = 0.01$. Now suppose that we always intervene at Z, and we tend to set $Z := z_0$ much more than $Z := z_1$. It will then appear to us that the two non-query nodes X and Y are only slightly correlated when in fact they are very heavily correlated.

Theorem 9.5.1 *Given a query* $\mathbf{Q} := \mathbf{q}$*, the probability of a response* $\mathbf{x}$ *to our query is:*

$$
\begin{aligned}
& P(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}, \prec) \\
= & \sum_{\mathcal{G} \in \prec} \prod_{i} P(Pa(X_i) = \mathbf{U}_i^{\mathcal{G}} \mid \prec) \prod_{j : X_j \notin \mathbf{Q}} Score(X_j, \mathbf{U}_j^{\mathcal{G}} \mid \mathbf{x}, \mathbf{q}) \\
= & \lambda_{\mathbf{Q}} \prod_{i : X_i \notin \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec) Score(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q}),
\end{aligned}
$$

where $\lambda_{\mathbf{Q}} = \prod_{i : X_i \in \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec)$.

Proof. See Appendix A.3.

□

Theorem 9.5.2 (Friedman and Koller, 2000) *We can write the probability of an edge* $X_j \to X_i$ *as:*

$$
P(X_j \to X_i \mid \prec) = \frac{\sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}, \mathbf{U} \ni X_j} P(Pa(X_i) = \mathbf{U} \mid \prec)}{\sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec)}.
$$

Intuitively, we are dividing the probability mass for structures that have $X_i \to X_j$ by the total mass for all of the structures. Most of the terms for each expression cancel out leaving just the terms involving the families for X_i. Notice that since we are performing Bayesian averaging over multiple graphs the probability of an edge $X_i \to X_j$ will generally only be high if X_i is a *direct* cause of X_j rather than if X_i merely has some indirect causal influence on X_j. A simple corollary of the previous theorem is:

Corollary 9.5.3 *Given query* $\mathbf{Q} := \mathbf{q}$ *and completion* $\mathbf{x}$ *we can write the probability of an edge* $X_j \to X_i$ *as:*

$$
P(X_j \to X_i \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec) = \frac{\sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}, \mathbf{U} \ni X_j} P(Pa(X_i) = \mathbf{U} \mid \prec) Score(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q})}{\sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec) Score(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q})}.
$$

where we define $Score(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q}) = 1$ *if* $X_i \in \mathbf{Q}$.

Now, consider the expected posterior loss (Eq. (9.1)) given $\prec$:

$$\begin{aligned} & \mathit{ExPLoss}_{\prec}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}) \mid \mathbf{Q} := \mathbf{q}) \\ = \;& E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q}, \prec)} \sum_{i,j} H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec). \end{aligned} \tag{9.5}$$

We can compute the distribution for $P(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec)$ by using Corollary 9.5.3. The formula for computing the probability of an edge $X_j \rightarrow X_i$ depends on *Score*$(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q})$ for each $\mathbf{U} \in \mathcal{U}_{i,\prec}$. Recall from Theorem 8.3.4 that:

$$\mathit{Score}(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q}) = P(x_i \mid \mathbf{u}),$$

where x_i and $\mathbf{u}$ are the values of X_i and $\mathbf{U}$ in the data instance $(\mathbf{q}, \mathbf{x})$. For all $\mathbf{U} \in \mathcal{U}_{i,\prec}$, we have $\mathbf{U} \subseteq \mathbf{W}_i$ and hence *Score*$(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q})$ only depends upon the values that $\mathbf{q}$ and $\mathbf{x}$ give to X_i and $\mathbf{W}_i$. Therefore, the expression $P(X_j \rightarrow X_i \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec)$ only depends upon the values that $\mathbf{q}$ and $\mathbf{x}$ give to X_i and $\mathbf{W}_i$. Similarly, the expression for the probabilty of an edge from X_i to X_j, $P(X_j \leftarrow X_i \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec)$ only depends upon the values that $\mathbf{q}$ and $\mathbf{x}$ give to X_j and $\mathbf{W}_j$. Thus, $H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec)$ depends only on the values that $\mathbf{q}$ and $\mathbf{x}$ give to X_i, X_j, $\mathbf{W}_i$ and $\mathbf{W}_j$.

Using this fact and then applying Theorem 9.5.1, we can rewrite the expected posterior loss as described in the follow theorem:

Theorem 9.5.4 *Given a query* $\mathbf{Q} := \mathbf{q}$*, the expected posterior loss can be written as:*

$$\begin{aligned} & \mathit{ExPLoss}_{\prec}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}) \mid \mathbf{Q} := \mathbf{q}) \\ = \;& \lambda_{\mathbf{Q}} \sum_{i,j} \sum_{\mathbf{x}} \psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) \prod_{k : X_k \notin \mathbf{Q}} \phi(x_k, \mathbf{w}_k), \end{aligned} \tag{9.6}$$

where

$$\psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) = H(X_i \leftrightarrow X_j \mid x_i, x_j, \mathbf{w}_i, \mathbf{w}_j, \prec),$$

$$\phi(x_k, \mathbf{w}_k) = \sum_{\mathbf{U} \in \mathcal{U}_{k,\prec}} P(\mathit{Pa}(X_k) = \mathbf{U} \mid \prec) \mathit{Score}(X_k, \mathbf{U} \mid x_k, \mathbf{w}_k).$$

Proof. See Appendix A.3.

□

Notice that we have successfully decomposed the loss so that we no longer encounter the computational blow-up from the exponential number of structures. However, this expression still involves summations over the exponential number of possible completions of a query. We can deal with this second form of exponential intractability by taking another look at Eq. (9.6). Notice that, for each i and j in Eq. (9.6), the summation over completions $\mathbf{x}$ resembles the expression for computing a marginal probability in Bayesian network inference where we are marginalizing out $\mathbf{x}$. In other words:

$$\sum_{\mathbf{x}} \psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) \prod_{k: X_k \notin \mathbf{Q}} \phi(x_k, \mathbf{w}_k), \tag{9.7}$$

is similar to Eq. (5.4). It is a sum of product of factors each of which is dependent on only a small number of variables. Regarding ψ and each ϕ as factors, we can then use the variable elimination algorithm presented in Section 5.7.1 to evaluate this expression effectively. The restriction to a candidate set of parents for each node ensures that each factor ϕ is over at most $(m+1)$ variables, and each factor ψ over at most $2m+1$ variables. After applying the variable elimination algorithm we end up with a factor over the variables $\mathbf{Q}$ where for each possible query $\mathbf{q}$ we have the value of the expression in Eq. (9.7).

We need to perform such an inference for each i, j pair. However, since we restricted to at most m candidate parents, the number of possible edges is at most mn. Thus, the computational cost of computing the expected posterior loss for all possible queries is the cost of mn applications of Bayesian network inference.

9.6 Analysis for Unrestricted Orderings

In the previous section, we obtained a closed form expression for computing the expected posterior loss of a query for a given ordering. We now generalize this derivation by removing the restriction of a fixed ordering.

Primarily for computational reasons, we start with a uniform prior over structures given an ordering $P(\mathcal{G} \mid \prec)$ and a uniform prior over orderings $P(\prec)$. As discussed by Friedman and Koller (2000), a uniform prior over structures given an ordering together with a uniform distribution over orderings does not correspond to a uniform prior over structures. This is because simpler structures (e.g., $(X \quad Y)$) are consistent with more orderings than more complex structures (e.g., $(X \rightarrow Y)$). On the other hand, structures that make more assumptions about the ordering of the nodes are making more assumptions about the causal ordering or the domain variables. Our prior makes these types of structures less likely *a priori*, which is arguably a reasonable prior to start off with.

We also note that our structure prior $P(\mathcal{G})$ does not satisfy structure modularity. However, for any given ordering $\prec$, the structure prior given that ordering $P(\mathcal{G} \mid \prec)$ does statisfy structure modularity. This is all that we require in our analysis since we shall first condition on a fixed ordering and then perform computations with respect to that ordering.

The expression for the expected posterior loss can be rewritten as:

$$\begin{aligned}
& \mathit{ExPLoss}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}) \mid \mathbf{Q} := \mathbf{q}) && (9.8)\\
= \; & E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q})} \mathit{Loss}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})) && (9.9)\\
= \; & E_{\prec} E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q}, \prec)} \mathit{Loss}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})) && (9.10)\\
= \; & E_{\prec} E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q}, \prec)} \sum_{i,j} H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}). && (9.11)
\end{aligned}$$

The expectation over orderings can be approximated by sampling possible orderings from our current distribution over graphs and parameters. As shown by Friedman and Koller (2000), sampling from orderings can be done very effectively using Markov chain Monte Carlo (MCMC) techniques.

The expression inside the expectation over orderings is very similar to the expected posterior loss of the query with a fixed ordering (Eq. (9.5)). The only difference is that we now must compute the entropy terms $H(X_i \leftrightarrow X_j \mid \mathbf{x}, \mathbf{Q} := \mathbf{q})$ without restricting ourselves to a single ordering. This entropy term is based on probability expressions for relationships between nodes:

$$\begin{aligned}
& P(X_i \rightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})\\
= \; & E_{\prec \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}} P(X_i \rightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec). && (9.12)
\end{aligned}$$

Each of the terms inside the expectation can be computed using Theorem 9.5.3.

Naively, we can compute the expectation for each query $\mathbf{Q} := \mathbf{q}$ and completion $\mathbf{x}$ by sampling orderings from $P(\prec \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ and then computing $P(X_i \rightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec)$. Clearly, this approach is impractical. However, we can use a simple approximation that substantially reduces the computational cost. Our general MCMC algorithm generates a set of orderings sampled from $P(\prec)$. In many cases, a single data instance will only have a small effect on the distribution over orderings; hence, we can often use our samples from $P(\prec)$ to be a reasonably good approximation to samples from the distribution $P(\prec \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$. Thus, we approximate Eq. (9.12) by:

$$
\begin{aligned}
& P(X_i \rightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}) \\
&= E_{\prec \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}} P(X_i \rightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec) \qquad (9.13) \\
&\approx E_{\prec} P(X_i \rightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}, \prec), \qquad (9.14)
\end{aligned}
$$

where the expectation over orderings is computed with our current set of MCMC sampled orderings. Note that this small approximation error will not accumulate since we are not using the approximation to update any of the parameters of our model, but merely to predict the value of candidate queries in this current round.

With the above approximation, we can compute $H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ efficiently. We note that, as in the fixed ordering case, the entropy term $H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ depends only on the values given to the variables $X_i, X_j, \mathbf{W}_i$ and $\mathbf{W}_j$. Thus, we can use the same variable elimination method to compute the expression:

$$
E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q}, \prec)} \sum_{i,j} H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}). \qquad (9.15)
$$

In other words, we evaluate the above expression for a particular order $\prec$ by computing:

$$
\lambda_{\mathbf{Q}} \sum_{i,j} \sum_{\mathbf{x}} \psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) \prod_{k : X_k \notin \mathbf{Q}} \phi(x_k, \mathbf{w}_k), \qquad (9.16)
$$

where,

$$\begin{aligned}
\lambda_{\mathbf{Q}} &= \prod_{i:X_i \in \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec), \\
\psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) &= H(X_i \leftrightarrow X_j \mid x_i, x_j, \mathbf{w}_i, \mathbf{w}_j), \\
\phi(x_k, \mathbf{w}_k) &= \sum_{\mathbf{U} \in \mathcal{U}_{k,\prec}} P(Pa(X_k) = \mathbf{U} \mid \prec) \mathit{Score}(X_k, \mathbf{U} \mid x_k, \mathbf{w}_k).
\end{aligned}$$

This expression is the same as equation Eq. (9.6), except that now we average over the set of sampled orderings when computing the ψ and ϕ factors. Notice that ψ does not depend upon the particular ordering that we are currently considering, and so we need only compute this expression once.

We can now compute expression (9.15) efficiently for a given ordering. We compute the expectation over orderings in Eq. (9.11) by computing then averaging these expressions for each of the sampled orderings.

We made two approximations to enable us to relax the restriction of a fixed ordering, each of which introduces some small amount of error. First, and most significantly, we sample over orderings. As noted by Friedman and Koller (2000), unlike the distribution over structures, the distribution over orderings appears to be far more amenable to sampling methods. Secondly, we made a small approximation to enable us to evaluate the expected posterior loss component given a fixed ordering (Eq. (9.15)). In Section 9.9, we show empirically that our (almost) exact closed form expected posterior loss computation for all of the structures consistent with a fixed ordering together with the effectiveness of sampling over orderings is accurate enough to determine the most useful experiments to perform.

9.7 Algorithm Summary and Properties

To summarize the algorithm, we first sample a set of orderings from the current distribution over graphs and parameters. We then use this set of orderings to compute and cache the ψ term present in Eq. (9.16). Next, for each ordering, we compute Eq. (9.16) by using the variable algorithm to obtain a factor $h_{\prec}(\mathbf{Q})$ over all possible queries. This factor gives

ActiveLearn(P)
  Sample orderings using MCMC
  Compute and cache ψ functions for each X_i, X_j pair
  For each set of candidate query variables $\mathbf{Q}$
    For each ordering
      Compute the loss factor $h_{\prec}(\mathbf{Q})$ associated with the
        ordering by using variable elimination with Eq. (9.16)
    End For
    Average the loss factors $h_{\prec}(\mathbf{Q})$ obtained from each ordering
      to obtain the expected posterior loss factor $h(\mathbf{Q})$
  End For
  Scan expected posterior loss factor $h(\mathbf{Q})$ for query $\mathbf{q}$ with lowest value.
  Ask query $\mathbf{Q} := \mathbf{q}$
  Receive complete response $\mathbf{x}$
  Update P
  Repeat

Figure 9.1: Active learning algorithm for structure learning in Bayesian networks.

us the value of $E_{\mathbf{x}\sim P(\mathbf{X}|\mathbf{Q}:=\mathbf{q},\prec)}H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ for each query $\mathbf{Q} := \mathbf{q}$. We then average all of these query factors obtained from each ordering. For example, if we maintain three orderings, we obtain three factors $h_{\prec_1}(\mathbf{Q})$, $h_{\prec_2}(\mathbf{Q})$, $h_{\prec_3}(\mathbf{Q})$. We then create a new factor $h(\mathbf{Q})$ in which each $h(\mathbf{q})$ entry is the average of the $\mathbf{q}$ entries of the three original factors. This process of averaging factors computes the expectation over orderings in Eq. (9.11). The final result is a query factor $h(\mathbf{Q})$ that, for each possible query $\mathbf{q}$ over variables $\mathbf{Q}$, gives the expected posterior loss of asking that query.

We then choose to ask the query that gives the lowest expected posterior loss. After weq ask a query $\mathbf{Q} := \mathbf{q}$ and receive the response $\mathbf{x}$ we then update our model P to get the posterior P' and we use P' in place of P to find our subsequent query. The algorithm is summarized in Fig. 9.1.

We now consider the computational complexity of the algorithm. For each ordering we need to compute $E_{\mathbf{x}\sim P(\mathbf{X}|\mathbf{Q}:=\mathbf{q},\prec)}\sum_{i,j} H(X_i \leftrightarrow X_j \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$. This involves at most mn Bayesian network inferences. Each inference returns a factor over all possible queries involving $\mathbf{Q}$ and so the inference will take time exponential in the number of query variables in $\mathbf{Q}$.

The above computation was for one particular set of query variables $\mathbf{Q}$. We may also have the ability to choose different subsets $\mathbf{Q}$ of queries variables from some set of controllable variables $\mathcal{C}$. Thus we would do the above mn inference computations for each of those subsets $\mathbf{Q}$. Hence, the time complexity of our algorithm to generate the next query is:

$$\mathcal{O}(\text{\# of sampled orderings} \cdot mn \cdot \text{\# of query subsets of } \mathcal{C} \cdot \text{ cost of BN inference}). \quad (9.17)$$

In addition, we need to generate the sampled orderings themselves. Friedman and Koller (2000) provide techniques, such as caching of statistics and commonly used expressions, that greatly reduce the cost of this process. They also show that the Markov chain mixes fairly rapidly, thereby reducing the number of steps in the chain required to generate a random sample. In our setting, we can reduce the number of steps required even further. Initially, we start with a uniform prior over orderings, from which it is easy to generate random orderings. Each of these is now the starting point for a Markov chain. As we do a single query and get a response, the new posterior distribution over orderings is likely to be very similar to the previous one. Hence, our old set of orderings is likely to be be fairly close to the new stationary distribution. Thus, a very small number of MCMC steps from each of the current orderings will give us a new set of orderings which is very close to being sampled from the new posterior.

9.8 Comment on Consistency

We now comment on the issue of consistency. Ideally we would like to have a guarentee that our algorithm always finds the correct underlying structure. There exist a number of matters to address. First, unlike the active parameter estimation algorithm in Chapter 7, it is possible that our active strucutre algorithm will end up concentrating its efforts to learn about one part of the domain, while ignoring certain important other queries. In practice, this does not tend to happen to a great extent. Nevertheless, to remedy this shortcoming, we could simply modify our algorithm so that it chooses an experiment uniformly at random after every, say, twenty queries.

The second matter we need to address is more subtle, and far less easy to resolve. It has less to do with the active learning aspect of our algorithm, and more to do with how we treat experimental data. Recall from Section 7.4 that our parameter updates for a given graph $\mathcal{G}$ may not be consistent if the data is really being generated from a different graph $\mathcal{G}^*$. For example, suppose $\mathcal{G}$ consists of just two separate nodes: X and Q. We update the Dirichlet distribution for $\boldsymbol{\theta}_X$ no matter what we set the query node Q to be, since we assume that the data values for X are coming from the true marginal distribution $P^*(X)$. However, if the true graph $\mathcal{G}^*$ is really $Q \to X$ then the X value of a new data instance is not distributed according to $P^*(X)$, but instead $P^*(X \mid \mathbf{Q} := \mathbf{q})$ for whatever query $\mathbf{Q} := \mathbf{q}$ we have chosen. Hence, our parameter for $\boldsymbol{\theta}_X$ will not converge to the true parameter in the limit. We also note that this (inconsistent) assumption of always being able to update a non-query node no matter how we set the query nodes appears to be implicitly assumed in Cooper and Yoo's (1999) proof of Theorem 8.3.4.

One way to view the inconsistent parameter estimates is the following. Suppose that our domain consists of two variables, X and Q. Also, rather than perform active learning, suppose we set queries $Q := q$ according to some distribution $\tilde{P}(Q)$. Then the X values of the data instances we receive are distributed according to $\sum_q P^*(X \mid Q := q)\tilde{P}(q)$, and this quantity is what our $\boldsymbol{\theta}_X$ parameters will converge to in the limit.

Despite this drawback, we still believe that our algorithm does find the correct underlying structure, and in practice this seems to be the case. Notice, that the inconsistent parameters will only be those that correspond to the families whose parents are not supersets of a family present in the true structure. In particular, the parameters of the CPDs for families that are present in the true structure will be consistent. It is plausible that, as we gather more data, the probability mass associated with the true structure will dominate our distribution so much so that the inconsistent parameter estimates for the other structures will have an inconsequential effect on any inferences that we perform. The true structure should dominate since it will explain the data far better than any other structure.

Based on the ideas outlined in this section, we believe that it should be possible to show that our algorithm determines the correct strucutre and parameters in the limit, however such a proof would be a serious undertaking, and some assumptions may have to be made about way we perform the active queries.

9.9 Structure Experiments

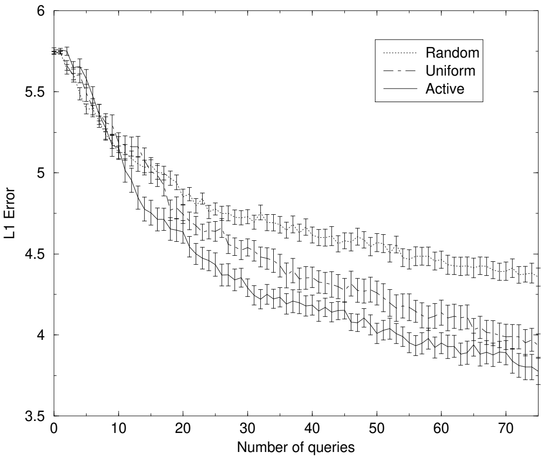
We experimented with three commonly used networks: **Cancer**, with five nodes; **Asia**, with eight nodes; and **Car Troubleshooter**, with twelve nodes. We evaluated the ability of our algorithm to reconstruct a network structure by using data generated from that network and then measuring how close the algorithm's estimate of the structure was to the true network. For each test network, we maintained 50–75 orderings, as described above. We restricted the set of candidate parents to have size $m = 5$.

We compared our active learning method with both random sampling and uniform querying, where we choose a setting for the query nodes from a uniform distribution. Each method produces estimates for the probabilities of edges between each pair of variables in our domain. Our goal is to learn the correct causal structure of a domain. Thus we would like all of the edges in our method's estimate to match those of the true network $\mathcal{G}^*$. We compared each method's estimate with the true network $\mathcal{G}^*$ by using the L_1 *edge error* of the estimate:

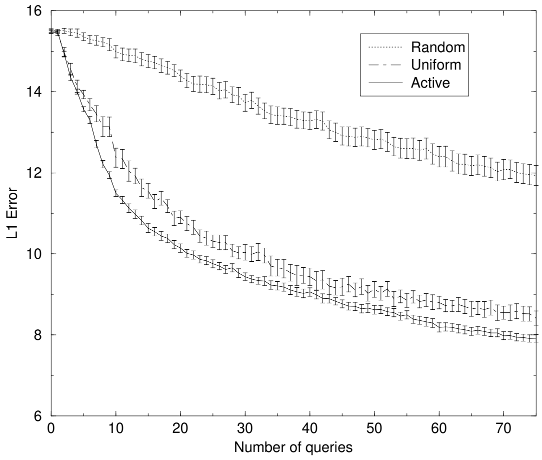
$$\begin{aligned} \mathit{Error}(P) = \sum_{i,j>i} & I_{\mathcal{G}^*}(X_i \rightarrow X_j)(1 - P(X_i \rightarrow X_j)) \\ & + I_{\mathcal{G}^*}(X_i \leftarrow X_j)(1 - P(X_i \leftarrow X_j)) \\ & + I_{\mathcal{G}^*}(X_i \quad X_j)(1 - P(X_i \quad X_j)), \end{aligned} \tag{9.18}$$

where $I_{\mathcal{G}^*}(A) = 1$ if A holds in $\mathcal{G}^*$ and is zero otherwise.

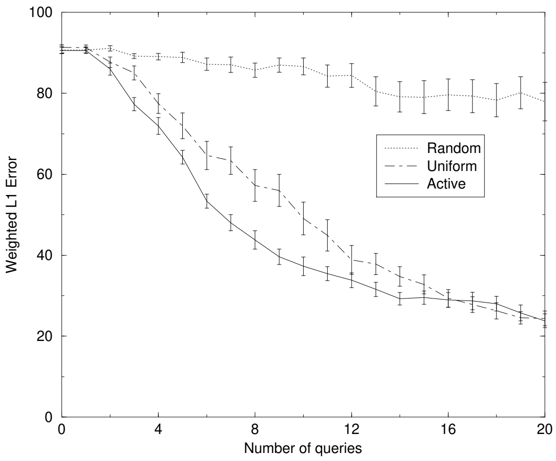
We first considered whether the active method provides any benefit over random sampling other than the obvious additional power of having access to queries that intervene in the model. Thus, for the first set of experiments, we eliminated this advantage by restricting the active learning algorithm to query only roots of $\mathcal{G}^*$. When the query is a root, a causal query is equivalent to simply selecting a data instance that matches the query (e.g., "Give me a 40-year-old male"); hence, there is no need for a causal intervention to create the response. Situations where we can only query root nodes arise in many domains; in medical domains, for example, we often have the ability to select subjects of a certain age, gender,



(a)



(b)



(c)

Figure 9.2: (a) **Cancer** with one root query node. (b) **Car** with four root query nodes. (c) **Car** with three root query nodes and weighted edge importance. Legends reflect order in which curves appear. The axes are zoomed for resolution.

or ethnicity, variables which are often assumed to be root nodes. All algorithms were informed that these nodes were roots by setting their candidate parent sets to be empty. In this batch of experiments, the candidate parents for the other nodes were selected at random, except that the node's true parents in the generating network were always in its candidate parent set. It typically took a few minutes for the active method to generate the next query.

Figures 9.2(a) and 9.2(b) show the learning curves for the **Cancer** and **Car** networks. We used a uniform prior over the structures and experimented with using uniform Dirichlet (BDe) priors and also more informed priors (simulated by sampling 20 data instances from the true network[2]). The type of prior made little qualitative difference in the comparative performance between the learning methods (the graphs shown are with uniform priors). In both graphs, we see that the active method performs significantly better than random sampling and uniform querying.

In some domains, determining the existence and direction of causal influence between two particular nodes may be of special importance. We experimented with this possibility in the **Car** network. We modified the L1 edge error function Eq. (9.18) and the edge entropy Eq. (9.3) used by the active method to make determining the relationship between two particular nodes (the *FuelSubsystem* and *EngineStart* nodes) 100 times more important than a regular pair of nodes. We used three other nodes in the network as query nodes. The results are shown in Fig. 9.2(c). Again, the active learning method performs substantially better.

Note that, without true causal interventions, all methods have the same limited power to identify the model: asymptotically, they will identify the skeleton and the edges whose direction is forced in the Markov equivalence class (rather than identifying all edge directions in the true causal network). However, even in this setting, the active learning algorithm allows us to derive this information significantly faster.

Finally, we considered the ability of the active learning algorithm to exploit its ability to perform interventional queries. We permitted our active algorithm to choose to set any pair of nodes or any single node or no nodes at all. We compared this approach to random sampling and also uniformly choosing one of our possible queries (setting a single node,

[2]In general, information from observational data can easily be incorporated into our model simply by setting $\mathbf{Q}$ to be the empty set for each of the observational data instances. By Theorem 8.3.4, the update rule for these instances is equivalent to standard Bayesian updating of the model.

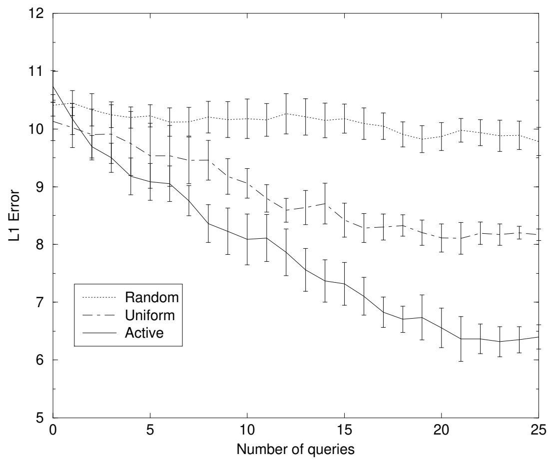


Figure 9.3: **Asia** with any pairs or single or no nodes as queries. Legends reflect order in which curves appear. The axes are zoomed for resolution.

pair of nodes, or no nodes). Experiments were performed on the **Asia**, **Cancer**, and **Car** networks with an informed prior of 20 random observations. In this batch of experiments, we also experimented with different methods for choosing the candidate parents for a node X. As an alternative to using random nodes together with the true parents, we chose the $m = 5$ variables which had the highest individual mutual information with X.[3] Empirically, both methods of choosing the candidate parents gave very similar results, despite the fact that for one node in the **Car** network, a true parent of a node happened not to be chosen as a candidate parent with the mutual information method. We present the results using the mutual information criterion for choosing parents.

Figures 9.3, 9.4(a) and 9.4(c) show that in all networks our active method significantly outperforms the other methods. We also see, in Figures 9.4(b) and 9.4(d), that the prediction error graphs are very similar to the graphs of the edge entropy (Eq. (9.3)) based on our distribution over structures. Recall that the edge entropy is our model's internal measure of **quality** – the model doesn't have access to the true causal network structure that it is trying to find and so cannot use the $L1$ edge error as its measure of quality. Ideally we would like the internal measure of quality to match closely with how near we really are to the true network structure. These graphs show that the edge entropy is, indeed, a reasonable surrogate for predictive accuracy.

[3] As we mentioned in Section 9.5, in practice this information can be obtained from observational data.

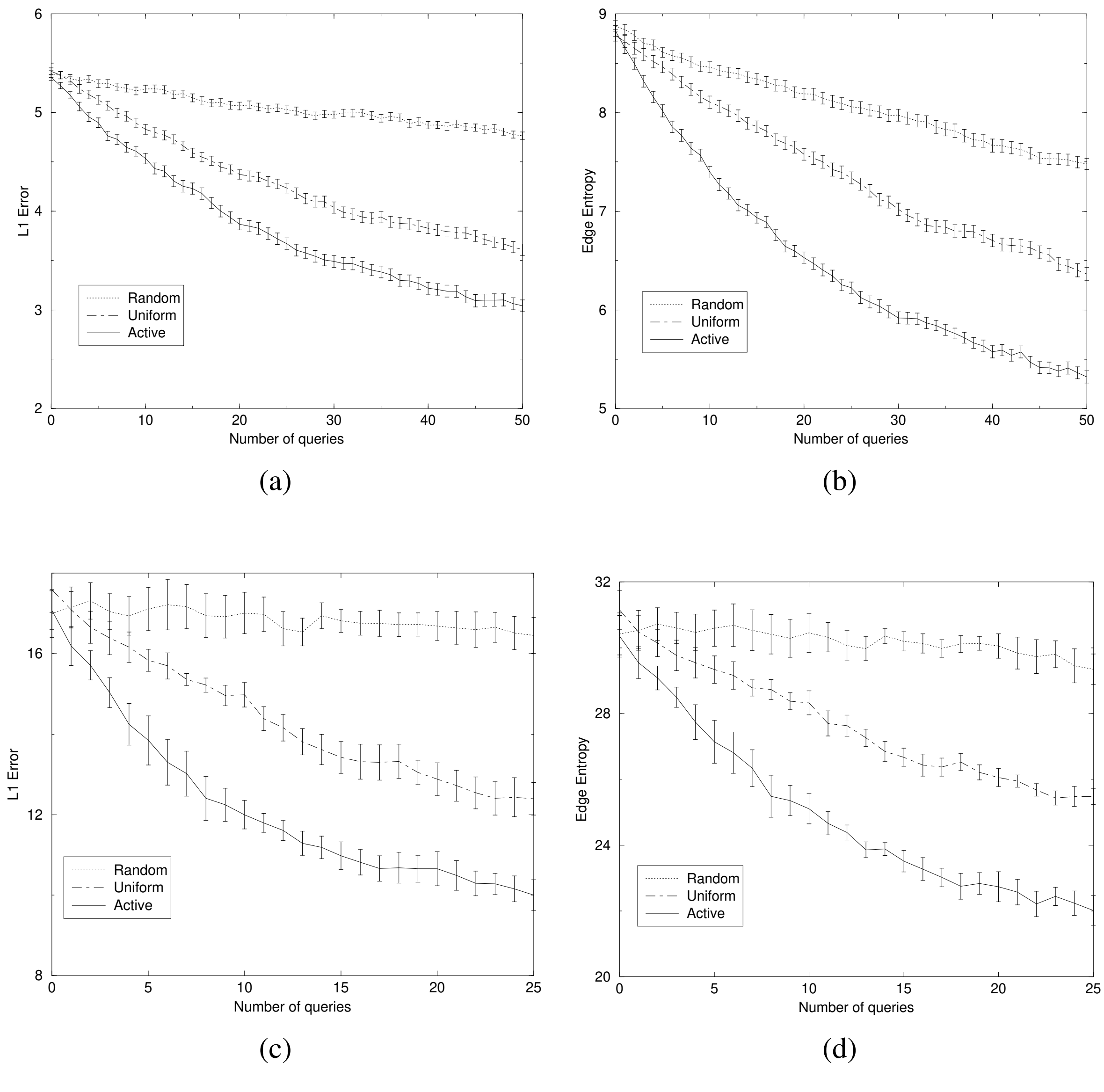


Figure 9.4: (a) **Cancer** with any pairs or single or no nodes as queries. (b) **Cancer** edge entropy. (c) **Car** with any pairs or single or no nodes as queries. (d) **Car** edge entropy. Legends reflect order in which curves appear. The axes are zoomed for resolution.

Figures 9.5(b), 9.5(c) and 9.5(d) show typical estimated causal edge probabilities in these experiments for random sampling, uniform querying and active querying respectively for the **Cancer** network (Fig. 9.5(a)). Figure 9.5(b) demonstrates that one requires more that just random observational data to learn the directions of many of the edges, and Fig. 9.5(d) shows that our active learning method creates better estimates of the causal interactions between variables than uniform querying. In fact, in some of the trials our active method recovered the edges and direction perfectly (when discarding low probability edges) and was the only method that was able to do so given the limitation of just 50 queries. Also, our active method tends to be much better at not placing edges between variables that are only indirectly causally related; for instance in the network distribution learned by the active method (summarized in Fig. 9.5(d)), the probability of an edge from *Cancer* to *Papilledema* is only 4% as opposed to 49% for uniform querying and 22% for random sampling.

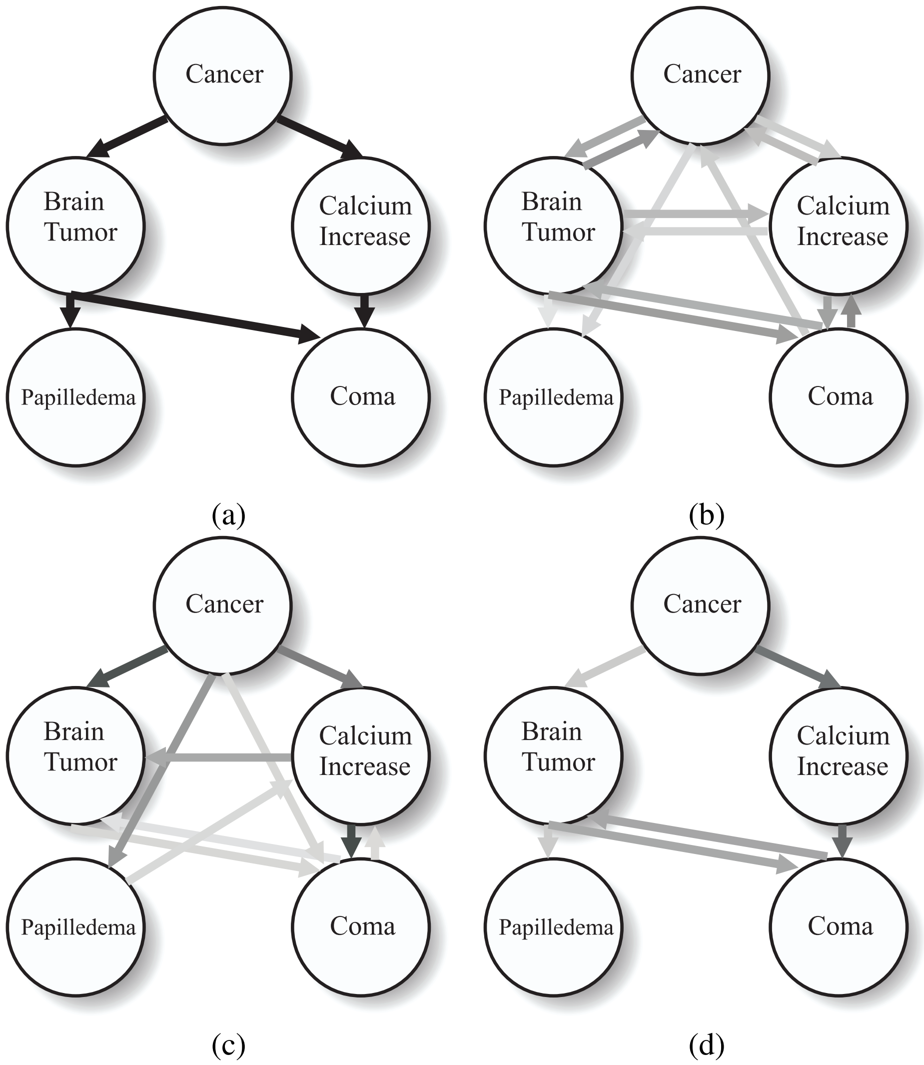


Figure 9.5: (a) Original **Cancer** network. (b) **Cancer** network after 70 observations. (c) **Cancer** network after 20 observations and 50 uniform experiments. (d) **Cancer** network after 20 observations and 50 active experiments. The darker the edges the higher the probability of edges existing. Edges with less than 15% probability are omitted to reduce clutter.

Part IV

Conclusions and Future Work

Chapter 10

Contributions and Discussion

"Questions are the creative acts of intelligence."

— Frank Kingdon, (1885-1958)

British botanist.

The goal of machine learning is to extract patterns from the world which can then be used to forward scientific understanding, create automated processes, assist with labor intensive tasks, and much more besides. However, much of machine learning relies on data, and gathering data is typically expensive and time consuming. We have demonstrated that, in a variety of widely applicable scenarios, active learning can be used to ask targeted, poignant and informative questions thereby vastly reducing the amount of data that needs to be gathered while, at the same time, increasing the quality of the resulting models, classifiers and conclusions.

We have tackled active learning by first creating a general approach whereby we define a **model** and its **quality**. We then myopically choose the next query that most improves the expected or minimax **quality**. We then have applied this general decision theoretic approach to the task at hand. In particular, we have addressed three different tasks: classification using support vector machines, parameter estimation and causal discovery using Bayesian networks.

10.1 Classification with Support Vector Machines

In the first part of this thesis, we introduced techniques for performing active learning with SVMs. We used the notion of a version space as our **model** and its size as the **quality**. By taking advantage of the duality between parameter space and feature space, we arrived at three algorithms that approximately reduce the version space as much as possible at each query round.

Empirically, these techniques can provide considerable gains in both the inductive and transductive settings for text classification – in some cases reducing the need for labeled instances by over an order of magnitude, and in almost all cases reaching the performance achievable on the entire pool having seen only a fraction of the data. Furthermore, larger pools of unlabeled data improve the quality of the resulting classifier by providing a wider range of potential queries for the active learner to choose from. Support vector machines are already one of the most effective classifiers for text classification, and our active learning methods improve their performance even further.

We have also demonstrated that active learning with support vector machines can provide a powerful tool for searching image databases, outperforming a number of traditional query refinement schemes. Our image retrieval algorithm, SVM_{Active}, not only achieves consistently high accuracy on a wide variety of user queries, but also does it quickly and maintains high precision when asked to deliver large quantities of images. Also, unlike recent systems such as SIMPLIcity (Wang et al., 2000), it does not require an explicit semantic layer to perform well.

Of the three main methods presented, the Simple method is computationally the fastest. However, the Simple method would seem to be a rougher and more unstable approximation, as we witnessed when it performed poorly on two of the five Newsgroup topics. If asking each query is expensive relative to computing time then using either the MaxMin or MaxRatio may be preferable. However, if the cost of asking each query is relatively cheap and more emphasis is placed upon fast feedback, as in the image retrieval domain, then the Simple method may be more suitable. In either case, we have shown that the use of these methods for learning can substantially outperform standard passive learning. Furthermore, experiments with the Hybrid method indicate that it is possible to combine the benefits of the MaxRatio and Simple methods.

The work presented on support vector machines leads us to many directions of interest. Several studies have noted that gains in computational speed can be obtained at the expense of generalization performance by querying multiple instances at a time (Lewis & Gale, 1994; McCallum & Nigam, 1998). Viewing SVMs in terms of the version space gives an insight as to where the approximations are being made, and may provide a guide as to which multiple instances are better to query. For instance, it is suboptimal to query two instances whose version space hyperplanes are fairly parallel to each other. There may exist a reasonable tradeoff between how well an instance bisects the version space and how mutually perpendicular it is to the other instances that we will be asking as queries.

Bayes Point Machines (Herbrich et al., 1999) also take advantage of the version space framework. They approximately find the center of mass of the version space. Using the Simple method with this point rather than the SVM point in version space may produce an improvement in performance and stability. The use of Monte Carlo methods (Applegate & Kannan, 1991; Herbrich & Graepel, 2001) to estimate version space areas may also give improvements.

Monte Carlo methods may also permit us to maintain a distribution over the version space. One way of viewing the strategy of always choosing to halve the version space is that we have essentially placed a uniform distribution over the current space of consistent hypotheses and we wish to reduce the expected size of version space as fast as possible. Rather than maintaining a uniform distribution over consistent hypotheses, it is plausible that the addition of prior knowledge over our hypothesis space may allow us to modify our query algorithm and provided us with an even better strategy. Furthermore, the PAC-Bayesian framework introduced by McAllester (1999) considers the effect of prior knowledge on generalization bounds and this approach may lead to theoretical guarantees for the modified querying algorithms.

For the image retrieval task, the running time of our algorithm scales linearly with the size of the image database both for the relevance feedback phase and for the retrieval of the top-k images. This linear scaling is because, for each querying round, we have to scan through the database for the twenty images that are closest to the current SVM boundary, and in the retrieval phase we have to scan the entire database for the top k most relevant images with respect to the learned concept. SVM_{Active} is practical for image databases that contain a few thousand images; however, we would like to find ways for it to scale to larger sized databases.

For the relevance feedback phase, one possible way of coping with a large image database is, rather than using the entire database as the pool, to sample a few thousand images from the database and use these as the pool of potential images with which to query the user. The technique of subsampling databases is often used effectively when performing data mining with large databases (e.g., (Chaudhuri et al., 1998)). It is plausible that this technique will have a negligible effect on overall accuracy, while significantly speeding up the running time of the SVM_{Active} algorithm on large databases. Retrieval speed of relevant images in large databases can perhaps be sped up significantly by using intelligent clustering and indexing schemes (Moore, 1991; Li et al., 2001). An online version of the SVM_{Active} system is available at: http://www.robotics.stanford.edu/~stong/svmActive.html. It already incorporates some of these clustering techniques.

Another direction we wish to pursue is an issue that faces many relevance feedback algorithms: that of designing methods to seed the algorithm effectively. At the moment we assume that we are presented with one relevant data instance and one irrelevant instance. It would be beneficial to modify SVM_{Active} so that it is not dependent on having a relevant starting instance. We are currently investigating ways of using SVM_{Active}'s output to explore the feature space effectively until a single relevant image is found.

Finally, the MaxRatio and MaxMin methods are computationally expensive since they have to step through each of the unlabeled data instances and learn an SVM for each possible labeling. This limits their use for interactive relevance feedback tasks in particular, and for active learning with large datasets in general. However, the temporarily modified data sets will only differ by one instance from the original labeled data set and so one can envisage learning an SVM on the original data set and then computing the "incremental" updates to obtain the new SVMs (Cauwenberghs & Poggio, 2001) for each of the possible labelings of each of the unlabeled instances. Thus, one would hopefully be able to obtain a much more efficient implementation of the MaxRatio and MaxMin methods and hence allow these active learning algorithms to scale up to larger machine learning problems and, in interactive relevance feedback tasks, to provide sufficiently fast responses.

10.2 Parameter Estimation and Causal Discovery

We have also explored active learning for Bayesian networks. To our knowledge, this study is one of the first applications of active learning in an unsupervised context.

We have demonstrated that active learning can have significant advantages for the task of parameter estimation in BNs, particularly in the case where our parameter prior is of the type that a human expert is likely to provide. We used the distribution over parameters as our **model** and the expected KL-divergence to the "true" parameters (or alternatively, the expected log likelihood of future data) as our notion of model **quality**. Intuitively, the benefit of active learning comes from estimating the parameters associated with rare events. Although it is less important to estimate the probabilities of rare events accurately, the number of instances obtained if we randomly sample from the distribution is still not enough. We note that this advantage arises even though use a loss function that considers only the accuracy of the distribution. In many practical settings such as medical or fault diagnosis, the rare cases are even more important, as they are often the ones that it is critical for the system to deal with correctly.

We have also considered the fundamental task of causal structure discovery. Here we used a distribution of graphs and parameters. Unlike the related non-active work of Cooper and Yoo (1999), our framework permits us to efficiently combine observational and experimental data for learning the structure over *all* variables in our domain, rather that just non-confounded pairs of variables. Thus we can take a much more global view of causal structure learning by taking into account indirect causation and confounding influences.

We demonstrated that active learning can provide significant benefits for causal structure discovery. We used the distribution over structures and parameters as our **model** and the entropy of the existence of edges between variables as our model **quality**. Our active method provides substantially better predictions regarding structure than both random sampling, and a process by which interventional queries are selected at random. Somewhat surprisingly, our algorithm achieves significant improvements over these other approaches even when it is restricted to querying roots in the network, and therefore cannot exploit the advantage of intervening in the model.

10.2.1 Augmentations

There are many interesting directions in which our work with Bayesian networks can be extended. For example, a treatment of continuous variables would be worthwhile. Two key issue to address are how to choose an query if the query variables are continuous, and whether the terms involving the continuous variables in the expected quality expression have a closed form and are decomposable.

In many domains there are missing data values (for example, partial experimental results) and hidden variables (variables that we never measure or observe) and it would be useful to explore how our algorithms could be extended to cope with such situations. Maintaining a distribution over graphs and parameters in the presence of missing data or hidden variables quickly becomes intractable (Heckerman, 1998). Among other things, the distribution over parameters becomes heavily multi-modal (thus prohibiting an efficient, closed form representation of the individual parameter distributions) and the parameters become dependent (thus preventing prohibiting us from factorizing the joint density over parameters into individual, smaller terms). Thus it remains a challenging research problem to extend Bayesian network active learning to cope with these scenarios.

Active learning can be regarded as being part of the large field of decision theory (Howard, 1970). Decision theory tackles the problem of decide how to act (in our case, which queries to ask) so as to maximum some utility function. The general field of decision theory tackles a great number of issues such as multiple decision making, computing the value of extra information, modeling people's utility functions and using decision theory as a framework for rationality.

Markov decision processes (MDPs) (Puterman, 1994) are a framework for representing the type of sequential decision making problems most related to active learning. They can potentially be used to relax the myopia approximation and enable us to introduce more advanced aspects of decision theory. For example, we may like to compute the next best query given that we can perform, say, twenty queries in total, or that we have, say, $\$10,000$ in total and each different type of query costs a certain amount. Such a setup also enables us to determine optimal stopping rules when performing queries – the point at which the expected future information gleaned from queries is outweighed by the expected cost. Unfortunately, even using the simplest networks, expressing our active learning problems as full MDPs becomes intractable. We would have a special type of MDP called a *belief state MDP*. The state space of our MDP would be huge: it would be the set of possible models, and each model is a distribution over parameters (and structures in the causal structure learning case). Approximate algorithms for dealing with massive state space sizes as well as algorithms for tackling belief state MDPs do exist (Bertsekas & Tsitsiklis, 1996; Kaelbling et al., 1998; Boutilier et al., 1999; Koller & Parr, 1999; Guestrin et al., 2001) although their applicability to active learning for Bayesian networks is unclear. The use of MDPs for augmenting the power of active learning in Bayesian networks remains an open issue.

Some of the benefits of the full decision theoretic framework could, perhaps, be approximately obtained without resorting to an MDP. For example, our active learning algorithms maintain an internal notion of model quality and thus we can plot the curve of model quality versus number of queries that we've asked so far. We can then extrapolate this learning curve and use the curve to decide whether to stop asking queries.

10.2.2 Scaling Up

Handling larger domains and larger data sets is an important area of research for most machine learning techniques. We would like to explore ways in which our active learning algorithms can be scaled up to cope with complex domains. There are a number of issue to tackle here. In our active learning for structure, we use MCMC methods to sample node orderings. MCMC methods often become infeasible when faced with a large data set size or a large dimensional problem. With a large amount of data the posterior distribution landscape often becomes much more "peaked", which causes MCMC methods great difficulty with slow convergence. Friedman and Koller (Friedman & Koller, 2000) note that this landscape is often much smoother when we sample over orderings as opposed to graph structures, but nevertheless, with enough data, even the posterior over orderings will become sharply peaked. Fortunately, this difficultly is slightly assuaged in the case of active learning because we typically wish to use active learning to reduce the amount of data we wish to collect. With very high dimensional problems containing several thousands of variables, the posterior distribution is often concentrated on a lower-dimensional subspace, which again can lead MCMC methods to suffer from slow convergence (Breiman, 1997). One can envisage scenarios in which we have combinations of a large quantity of data, a high dimensional domain and active learning. In the case where we have large data set sizes, we may be able to take advantage of the possibility that there will only be a few graphs (and hence orderings) that fit the data well. Thus, perhaps maintaining a small set of key orderings would be enough to account for most of the probability mass of the distribution over orderings. If we are faced with a very high dimensional problem the problem of convergence using MCMC will only be one of several issues that need to be addressed.

Our Bayesian network algorithms currently evaluate the expected posterior quality for *every* possible query. The number of possible queries grows exponentially with the number of query variables that we can control at once. There are a number of approaches one could explore to reduce the number of queries that are evaluated for each round of querying. For example, we could make use of the observation that if the expected quality of a query $\mathbf{Q} := \mathbf{q}$ is high last querying round, then, because the model does not changed much in response to a single query, it is likely that $\mathbf{Q} := \mathbf{q}$ will produce a large increase in expected quality in the next querying round as well. Thus, if we can only afford to evaluate, say, 100 candidate queries, we could perform some form of sampling in which the most promising queries (the queries that gave a large expected increase in quality in the previous few querying rounds) are sampled with higher likelihood than the less promising ones.

10.2.3 Temporal Domains

Discrete time-step temporal processes can be represented as *dynamic* Bayesian networks (Dean & Kanazawa, 1989) (see Fig. 10.1 for an example). The temporal aspect of a domain defines a natural partial causal ordering on the nodes in the network: nodes in the past cannot be causally dependent upon those in the future. If we assume that we know the edges present within each discrete time-slice (but that we don't necessarily know the edges between time-slices), then this constraint enforces a total ordering on the nodes. Thus, there is no need to sample node orderings to compute the expected loss. Furthermore, given that we have just one ordering, we may be able to use a wider variety of loss functions to

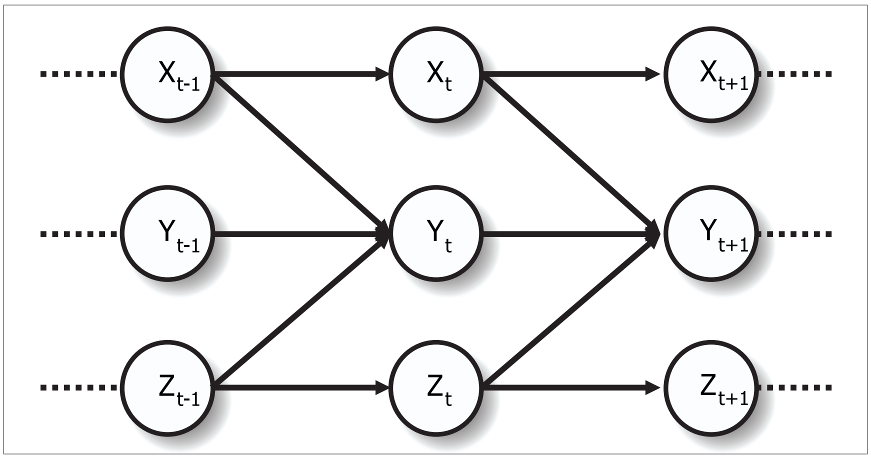


Figure 10.1: Three time-slices of a Dynamic Bayesian network.

measure the model quality – for example the entropy of the distribution over structures.

Using active learning to uncover the parameters or underlying structure of a DBN could be extended to the problem of active learning for optimal control (Boyan, 1995). This problem is closely related to that of reinforcement learning. In the optimal control problem, at each time-step, one observes some variables and then is permitted to perform some actions. The goal is to find the best actions to perform given current and past observations so as to maximize some utility. Such a task can be represented by a Markov decision process which can be regarded as a DBN augmented with nodes that represent actions and nodes that represent utilities.

10.2.4 Other Tasks and Domains

There exist many other problems related to Bayesian networks an related representations that we would like to explore. Relating active learning to the value of information, we might be able to use active learning to decide which extra variable to observe or which extra piece of missing data we should try to obtain in order to best learn the model. In practice, data instances are not always complete, or are partial on purpose. For example, doctors may always take a patient's temperature, but may not give every patient a X-ray. It may be useful to suggest which extra readings will be most promising to take.

Another exciting direction is the potential of using active learning in order to try to

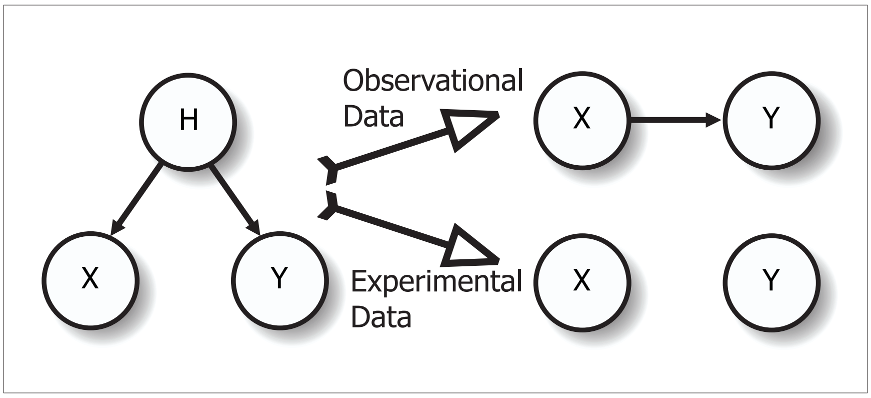


Figure 10.2: A hidden variable H makes X and Y appear correlated in observational data, but independent in experimental data.

uncover the existence of a hidden variable in our domain. As we noted in Section 10.2.1, representing a distribution over structures and parameters in the presence of hidden variables can be very difficult. Intuitively, the task of searching for hidden variables should not have to involve such a complex setup. If we believe that there is a hidden variable H between X and Y that is making X and Y appear to be causally dependent then one easy way to ascertain whether H exists is first to set X and observe Y and then set Y and observe X. If X and Y appear independent then it is likely that there is a hidden variable (see Fig. 10.2). One possible direction to explore in formalizing this intuition is to gather observational data and consider the distribution of graph structures given the data. If there is a high probability of an edge between nodes X and Y, then, if there are no hidden variables, it should be due to a direct causal influence from X to Y or from Y to X. If there were a hidden variable, then when we just look at experimental data where we intervene at X or Y, there will be a much lower probability of an edge between X and Y. Thus, we could attempt to choose queries so as to maximize some form of discrepancy between the distribution over graphs obtained by using observational data, and the distribution over graphs obtained using experimental data.

Object oriented Bayesian networks (OOBNs) (Koller & Pfeffer, 1997) and, more generally, probabilistic relational models (PRMs) (Pfeffer, 2000) are effective frameworks for enabling Bayesian networks to scale-up to very large domains. PRMs extend the standard attribute-based Bayesian network representation to incorporate a richer relational structure. These models allow the specification of a probability model for classes of objects rather than simple attributes; they also allow properties of an object to depend probabilistically on properties of other related objects. PRMs augment the representational power of Bayesian networks – for example they enable one to model structural uncertainty over the very existence or number of objects in our domain. A possibly fruitful avenue to pursue would be to investigate how the methods and techniques presented here carry over to these new representations. Potential research issues are how to represent queries (in a relational database the notion of a set of data instances no longer exists), whether the parameter sharing nature of these models can be exploited efficiently in the querying algorithm, and whether one can actively choose queries that uncover or reveal the new types of structural uncertainty.

10.3 Epilogue

We hope that the work presented here will provide motivation for further work into exploring the uses of active learning within machine learning and statistics. There are numerous applications of active learning to real-world domains, a number of which have been demonstrated, and many of which have been alluded to in this text. Active learning provides clear productivity and financial benefits in industrial settings by reducing the expensive task of gathering data and performing experiments. In addition, the investigation of active learning can provide a useful insight into how automated devices can be designed so as to ask meaningful and apparently intelligent questions in order to learn about a domain. We have also outlined an number of open issues that now present themselves to us with respect to improving and extending the current work. In the words of a famous American economist, social commentator and former Stanford professor:

"The outcome of any serious research can only be
to make two questions grow where only one grew before."

— Thorstein Veblen, (1857-1929).
The Place of Science in
Modern Civilization.

Appendix A

Proofs

A.1 Preliminaries

We shall frequently use the following identity:

$$\forall z \;\; z\Gamma(z) = \Gamma(z+1). \tag{A.1}$$

We shall also use the following equivalence frequently. For a Bayesian network paremterized by multinomial table CPDs with independent Dirichlet distributions over the CPD parameters:

$$\begin{aligned}
\tilde{\theta}_{x_{ij}|\mathbf{u}} &= \int \theta_{x_{ij}|\mathbf{u}} p(\theta_{x_{ij}|\mathbf{u}})\, d\theta_{x_{ij}|\mathbf{u}} & \text{(A.2)} \\
&= \frac{\alpha_{x_{ij}|\mathbf{u}}}{\alpha_{x_{i*}|\mathbf{u}}} & \text{(A.3)} \\
&= P(x_{ij} \mid \mathbf{u}), & \text{(A.4)}
\end{aligned}$$

which is equivalent to the standard (Bayesian) approach used for collapsing a distribution over BN parameters into a single parameter vector for one-step prediction. We shall also make use of the following well know result (DeGroot, 1970):

Lemma A.1.1 *Suppose* $p(\theta_1, \ldots, \theta_r) = \textit{Dirichlet}(\alpha_1, \ldots, \alpha_r)$. *Then,*

$$p(\theta_i) = \textit{Beta}(\alpha_i, \sum_{k \neq i} \alpha_k).$$

Lemma A.1.2 *Suppose* $p(\theta) = \textit{Beta}(a, b)$. *Then,*

$$\int_0^1 (\theta \ln \theta)\, p(\theta)\, d\theta = \frac{a}{a+b}(\Psi(a+1) - \Psi(a+b+1)), \tag{A.5}$$

where Ψ *is the* digamma function $\frac{\Gamma'(\alpha)}{\Gamma(\alpha)}$.

Proof.

$$\begin{aligned} & \int_0^1 (\theta \ln \theta)\, p(\theta)\, d\theta & \text{(A.6)} \\ = \; & \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \int_0^1 \theta^a (1-\theta)^{(b-1)} \ln \theta \, d\theta. & \text{(A.7)} \end{aligned}$$

Using a standard table of integrals, the above expression can be re-written as:

$$\begin{aligned} & \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} \cdot \frac{\Gamma(a+1)\Gamma(b)}{\Gamma(a+b+1)} (\Psi(a+1) - \Psi(a+b+1)) & \text{(A.8)} \\ = \; & \frac{a}{a+b}(\Psi(a+1) - \Psi(a+b+1)). & \text{(A.9)} \end{aligned}$$

□

A.2 Parameter Estimation Proofs

A.2.1 Using KL Divergence Parameter Loss

Theorem A.2.1 *Let* $\Gamma(\alpha)$ *be the Gamma function,* $\Psi(\alpha)$ *be the* digamma function *and* H *be the entropy function. Define:*

$$\delta(\alpha_1, \ldots, \alpha_r) = \sum_{j=1}^{r} \left[\frac{\alpha_j}{\alpha_*} (\Psi(\alpha_j + 1) - \Psi(\alpha_* + 1)) + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*}\right) \right].$$

Then the risk decomposes as:

$$\mathsf{Risk}(p(\boldsymbol{\theta})) = \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \delta(\alpha_{x_{i1}|\mathbf{u}}, \ldots, \alpha_{x_{ir_i}|\mathbf{u}}). \tag{A.10}$$

Proof.

$$\begin{aligned}
\mathsf{Risk}(p(\boldsymbol{\theta})) &= E_{\boldsymbol{\Theta} \sim p(\boldsymbol{\theta})} \mathsf{KL}(\Theta \parallel \tilde{\boldsymbol{\theta}}) && \text{(A.11)} \\
&= \int \mathsf{KL}(\boldsymbol{\theta} \parallel \tilde{\boldsymbol{\theta}}) p(\boldsymbol{\theta})\, d\boldsymbol{\theta} && \text{(A.12)} \\
&= \int \sum_i \sum_{\mathbf{u}} P_{\boldsymbol{\theta}}(\mathbf{u}) \mathsf{KL}(P_{\boldsymbol{\theta}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}}(X_i \mid \mathbf{u})) p(\boldsymbol{\theta})\, d\boldsymbol{\theta}. && \text{(A.13)}
\end{aligned}$$

Now, using parameter independence, which allows us to separately integrate $P_{\boldsymbol{\theta}}(\mathbf{u})$, and noticing $\int P_{\boldsymbol{\theta}}(\mathbf{u}) p(\boldsymbol{\theta})\, d\boldsymbol{\theta} = P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u})$, expression (A.13) becomes:

$$\sum_i \sum_{\mathbf{u}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \sum_j \int_0^1 \theta_{x_{ij}|\mathbf{u}} \ln \frac{\theta_{x_{ij}|\mathbf{u}}}{\tilde{\theta}_{x_{ij}|\mathbf{u}}} p(\theta_{x_{ij}|\mathbf{u}})\, d\theta_{x_{ij}|\mathbf{u}}. \tag{A.14}$$

Using that $\tilde{\theta}_{x_{ij}|\mathbf{u}} = \int_0^1 \theta_{x_{ij}|\mathbf{u}} p(\theta_{x_{ij}|\mathbf{u}})\, d\theta_{x_{ij}|\mathbf{u}} = P(x_{ij} \mid \mathbf{u})$ we have that this expression is equal to:

$$\begin{aligned}
&\sum_i \sum_{\mathbf{u}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \left(\sum_j \int_0^1 \left(\theta_{x_{ij}|\mathbf{u}} \ln \theta_{x_{ij}|\mathbf{u}}\right) p(\theta_{x_{ij}|\mathbf{u}})\, d\theta_{x_{ij}|\mathbf{u}} - \sum_j P(x_{ij} \mid \mathbf{u}) \ln P(x_{ij} \mid \mathbf{u}) \right) \\
&= \sum_i \sum_{\mathbf{u}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \left(\sum_j \int_0^1 \left(\theta_{x_{ij}|\mathbf{u}} \ln \theta_{x_{ij}|\mathbf{u}}\right) p(\theta_{x_{ij}|\mathbf{u}})\, d\theta_{x_{ij}|\mathbf{u}} + H(P(X_i \mid \mathbf{u})) \right). \qquad \text{(A.15)}
\end{aligned}$$

Applying Lemma A.1.1 and Lemma A.1.2 we finally obtain:

$$\sum_i \sum_{\mathbf{u}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \sum_j \left[\frac{\alpha_{x_{ij}|\mathbf{u}}}{\alpha_{x_{i*}|\mathbf{u}}} \left(\Psi(\alpha_{x_{ij}|\mathbf{u}} + 1) - \Psi(\alpha_{x_{i*}|\mathbf{u}} + 1) \right) + H(P(X_i \mid \mathbf{u})) \right]. \tag{A.16}$$

□

Theorem A.2.2 *Consider a simple network in which X has parents $\mathbf{Q}$. Then:*

$$\Delta(X \mid \mathbf{q}) = P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left(H\left(\frac{\alpha_{x_1|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}, \ldots, \frac{\alpha_{x_r|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}\right) - \sum_j P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) H\left(\frac{\alpha'_{x_1|\mathbf{q}}}{\alpha'_{x_*|\mathbf{q}}}, \ldots, \frac{\alpha'_{x_r|\mathbf{q}}}{\alpha'_{x_*|\mathbf{q}}}\right)\right), \tag{A.17}$$

where $\alpha_{x_*|\mathbf{q}} = \sum_i \alpha_{x_i|\mathbf{q}}$. Also, $\alpha'_{x_i|\mathbf{q}} = (\alpha_{x_i|\mathbf{q}} + 1)$ if $i = j$ and $\alpha'_{x_i|\mathbf{q}} = \alpha_{x_i|\mathbf{q}}$ otherwise. Thus $\alpha'_{x_*|\mathbf{q}} = \alpha_{x_*|\mathbf{q}} + 1$.

Proof. To ease notation, let $\alpha_{x_i|\mathbf{q}} = \alpha_i$ for all $i = 1, \ldots, r$ and $\alpha_* = \sum_{i=1}^r \alpha_i$.

By the discussion in Section 7.3.2,

$$\begin{aligned} &\Delta(X \mid \mathbf{q}) && \text{(A.18)}\\ &= P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left[\delta(\alpha_1, \ldots, \alpha_r) - \sum_{j=1}^r P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) \delta(\alpha_1, \ldots, \alpha_j + 1, \ldots, \alpha_r)\right]. && \text{(A.19)} \end{aligned}$$

Let

$$K(\alpha_1, \ldots, \alpha_r) = \sum_j \frac{\alpha_j}{\alpha_*} \Psi(\alpha_j + 1) + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*}\right),$$

and

$$H_j = H\left(\frac{\alpha_1}{\alpha_* + 1}, \ldots, \frac{\alpha_j + 1}{\alpha_* + 1}, \ldots, \frac{\alpha_r}{\alpha_* + 1}\right).$$

Also, using the fact that, $\forall z \;\; \Psi(z+1) = \Psi(z) + \frac{1}{z}$ and $P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) = \frac{\alpha_j}{\alpha_*}$, after some algebraic manipulation we obtain:

$$\begin{aligned} &\Delta(X \mid \mathbf{q}) && \text{(A.20)}\\ &= P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left[\frac{1}{\alpha_* + 1} + K(\alpha_1, \ldots, \alpha_r) - \sum_{j=1}^r P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) K(\alpha_1, \ldots, \alpha_j + 1, \ldots, \alpha_r)\right] && \text{(A.21)}\\ &= P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left[\frac{1}{\alpha_* + 1} + \sum_{j=1}^r \frac{\alpha_j}{\alpha_*} \Psi(\alpha_j + 1) + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*}\right)\right.\\ &\qquad \left. - \sum_{j=1}^r P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) \left(\sum_{k \neq j} \frac{\alpha_k}{\alpha_* + 1} \Psi(\alpha_k + 1) + \frac{\alpha_j + 1}{\alpha_* + 1} \Psi(\alpha_j + 2) + H_j\right)\right] && \text{(A.22)}\\ &= P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left[\frac{1}{\alpha_* + 1} + \sum_{j=1}^r \frac{\alpha_j}{\alpha_*} \Psi(\alpha_j + 1) + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*}\right)\right. \end{aligned}$$

$$-\sum_{j=1}^{r} P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) \left(\sum_{k=1}^{r} \frac{\alpha_k}{\alpha_* + 1} \Psi(\alpha_k + 1) + \frac{\alpha_j}{(\alpha_* + 1)(\alpha_j + 1)} + \frac{1}{\alpha_* + 1} \Psi(\alpha_j + 2) + H_j \right) \Bigg] . \tag{A.23}$$

Gathering $\Psi(\alpha_j + 1)$ terms in Eq. (A.23) and then expanding $\Psi(\alpha_j + 2) = \Psi(\alpha_j + 1) + \frac{1}{\alpha_j + 1}$ we obtain:

$$\begin{aligned}
& P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \Bigg[\frac{1}{\alpha_* + 1} + \sum_{j=1}^{r} \frac{\alpha_j}{(\alpha_*)(\alpha_* + 1)} \Psi(\alpha_j + 1) + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*} \right) \\
& \quad - \sum_{j=1}^{r} \frac{\alpha_j}{\alpha_*} \left(\frac{\alpha_j}{(\alpha_* + 1)(\alpha_j + 1)} + \frac{1}{\alpha_* + 1} \left(\Psi(\alpha_j + 1) + \frac{1}{\alpha_j + 1} \right) + H_j \right) \Bigg] \\
= \; & P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left[\frac{1}{\alpha_* + 1} + H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*} \right) - \sum_{j=1}^{r} \frac{\alpha_j}{(\alpha_* + 1)(\alpha_*)} - \sum_{j=1}^{r} P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) H_j \right] \qquad \text{(A.24)} \\
= \; & P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left[H\left(\frac{\alpha_1}{\alpha_*}, \ldots, \frac{\alpha_r}{\alpha_*} \right) - \sum_{j=1}^{r} P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) H_j \right] . \qquad \text{(A.25)}
\end{aligned}$$

□

Theorem A.2.3 *The change in risk of a Bayesian network over variables $\mathcal{X}$ when asking query $\mathbf{Q} := \mathbf{q}$ is given by:*

$$\begin{aligned}
\Delta(\mathcal{X} \mid \mathbf{q}) &= \mathsf{Risk}(p(\boldsymbol{\theta})) - \mathsf{ExPRisk}(p(\boldsymbol{\theta}) \mid \mathbf{q}) \qquad \text{(A.26)} \\
&\approx \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid \mathbf{Q} := \mathbf{q}) \Delta(X_i \mid \mathbf{u}), \qquad \text{(A.27)}
\end{aligned}$$

where $\Delta(X_i \mid \mathbf{u})$ is as defined in Eq. (A.17). Notice that we actually only need to sum over the updateable X_is since $\Delta(X_i \mid \mathbf{u})$ will be zero for all non-updateable X_is.

Proof.

$$\begin{aligned}
& \mathsf{ExPRisk}(p(\boldsymbol{\theta}) \mid \mathbf{Q} := \mathbf{q}) \\
& = E_{\Theta \sim p(\boldsymbol{\theta})} E_{\mathbf{x} \sim P_{\Theta}(\mathbf{X} \mid \mathbf{Q} := \mathbf{q})} \mathsf{Risk}(p(\boldsymbol{\theta} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})) \\
& = E_{\mathbf{x} \sim P_{\tilde{\boldsymbol{\theta}}}(\mathbf{X} \mid \mathbf{Q} := \mathbf{q})} \mathsf{Risk}(p(\boldsymbol{\theta} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})).
\end{aligned}$$

Let $\tilde{\boldsymbol{\theta}}'$ be the point estimate for $p(\theta \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$. Then using the fact that the KL divergence decomposes (Eq. (7.2)) we have that this expression is equal to:

$$\begin{aligned}
& E_{\mathbf{x}\sim P_{\tilde{\boldsymbol{\theta}}}(\mathbf{X}\mid\mathbf{Q}:=\mathbf{q})} E_{\boldsymbol{\Theta}'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} \mathsf{KL}(\boldsymbol{\Theta}' \parallel \tilde{\boldsymbol{\theta}}') \\
&= E_{\mathbf{x}\sim P_{\tilde{\boldsymbol{\theta}}}(\mathbf{X}\mid\mathbf{Q}:=\mathbf{q})} E_{\boldsymbol{\Theta}'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} \sum_i \sum_{\mathbf{u}\in Dom[\mathbf{U}_i]} P_{\boldsymbol{\Theta}'}(\mathbf{u}) \mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u})) \\
&= \sum_i E_{\mathbf{x}\sim P_{\tilde{\boldsymbol{\theta}}}(\mathbf{X}\mid\mathbf{Q}:=\mathbf{q})} \sum_{\mathbf{u}\in Dom[\mathbf{U}_i]} E_{\boldsymbol{\Theta}'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} P_{\boldsymbol{\Theta}'}(\mathbf{u}) \mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u})) \\
&= \sum_i \sum_{\mathbf{x}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}) \sum_{\mathbf{u}\in Dom[\mathbf{U}_i]} E_{\boldsymbol{\Theta}'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} P_{\boldsymbol{\Theta}'}(\mathbf{u}) \mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u})).
\end{aligned}$$

First using parameter independence and then supposing that $P_{\tilde{\boldsymbol{\theta}}'}(\mathbf{u}) \approx P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u})$ we have that this expression becomes:

$$\begin{aligned}
& \sum_i \sum_{\mathbf{x}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}) \sum_{\mathbf{u}\in Dom[\mathbf{U}_i]} \Big(E_{\Theta'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} P_{\boldsymbol{\Theta}'}(\mathbf{u}) \times \\
& \qquad\qquad E_{\boldsymbol{\Theta}'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} \mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u})) \Big) \\
&\approx \sum_i \sum_{\mathbf{x}} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}) \sum_{\mathbf{u}\in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) E_{\boldsymbol{\Theta}'\sim p(\boldsymbol{\theta}\mid\mathbf{Q}:=\mathbf{q},\mathbf{x})} \mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u})).
\end{aligned}$$

Notice that $\mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u}))$ is just dependent upon the parameters $\boldsymbol{\theta}'_{X_i\mid\mathbf{u}}$ (i.e., $\theta'_{x_{ij}\mid\mathbf{u}}$ for all j). Now, $p(\boldsymbol{\theta}_{X_i\mid\mathbf{u}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x})$ is only dependent upon the values of X_i and $\mathbf{U}_i$ within the instantiation $\mathbf{Q} := \mathbf{q}, \mathbf{x}$.

Also, notice that if X_i is not updateable, then $\mathsf{KL}(P_{\boldsymbol{\Theta}'}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'}(X_i \mid \mathbf{u})) = \mathsf{KL}(P_{\boldsymbol{\Theta}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}}(X_i \mid \mathbf{u}))$ and so the loss does not depend upon the completion $\mathbf{x}$ that we are summing over. Furthermore, if X_i is an updateable node, then the nodes in $\mathbf{Q}$ are not descendents of X_i (by definition of updateable in the selectional query case, and because of mutilation in the interventional query case). Thus X_i is independent of $\mathbf{Q}$ given the value of its parents $\mathbf{U}_i$. Hence, $p(\boldsymbol{\theta}_{X_i\mid\mathbf{u}} \mid \mathbf{Q} := \mathbf{q}, \mathbf{x}) = p(\boldsymbol{\theta}_{X_i\mid\mathbf{u}} \mid \mathbf{u}, x_i)$. We now have:

$$\begin{aligned}
& \sum_i \sum_{x_i,\mathbf{u}'} P_{\tilde{\boldsymbol{\theta}}}(x_i, \mathbf{U}_i = \mathbf{u}' \mid \mathbf{Q} := \mathbf{q}) \sum_{\mathbf{u}\in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \times \\
& \qquad E_{\boldsymbol{\Theta}'_{X_i\mid\mathbf{u}}\sim p(\boldsymbol{\theta}'_{X_i\mid\mathbf{u}}\mid\mathbf{u},x_i)} \mathsf{KL}(P_{\boldsymbol{\Theta}'_{X_i\mid\mathbf{u}}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'_{X_i\mid\mathbf{u}}}(X_i \mid \mathbf{u}))
\end{aligned} \quad \text{(A.28)}$$

$$= \sum_i \sum_{\mathbf{u}' \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}' \mid \mathbf{Q} := \mathbf{q}) \sum_{x_i} P_{\tilde{\boldsymbol{\theta}}}(x_i \mid \mathbf{U}_i = \mathbf{u}') \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) \times E_{\boldsymbol{\Theta}'_{X_i|\mathbf{u}} \sim p(\boldsymbol{\theta}'_{X_i|\mathbf{u}} \mid \mathbf{u}, x_i)} \mathsf{KL}(P_{\boldsymbol{\Theta}'_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u})). \tag{A.29}$$

Let us take a look at the regular risk:

$$\begin{aligned} \mathsf{Risk}(p(\boldsymbol{\theta})) &= E_{\boldsymbol{\Theta} \sim p(\boldsymbol{\theta})} \mathsf{KL}(\Theta \parallel \tilde{\boldsymbol{\theta}}) \\ &= E_{\boldsymbol{\Theta} \sim p(\boldsymbol{\theta})} \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\boldsymbol{\theta}}(\mathbf{u}) \mathsf{KL}(P_{\boldsymbol{\theta}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}}(X_i \mid \mathbf{u})) \\ &= \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) E_{\boldsymbol{\Theta}_{X_i|\mathbf{u}} \sim p(\boldsymbol{\theta}_{X_i|\mathbf{u}})} \mathsf{KL}(P_{\boldsymbol{\Theta}_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u})). \end{aligned} \tag{A.30}$$

When we take the difference of Eq. (A.30) and Eq. (A.29) we obtain:

$$\mathsf{Risk}(p(\boldsymbol{\theta})) - \mathsf{ExPRisk}(p(\boldsymbol{\theta}) \mid \mathbf{q}) \tag{A.31}$$

$$\begin{aligned} \approx & \sum_i \sum_{\mathbf{u}' \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}' \mid \mathbf{Q} := \mathbf{q}) \times \\ & \left(\sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) E_{\boldsymbol{\Theta}_{X_i|\mathbf{u}} \sim p(\boldsymbol{\theta}_{X_i|\mathbf{u}})} \mathsf{KL}(P_{\boldsymbol{\Theta}_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u})) \right. \\ & \left. - \sum_{x_i} P_{\tilde{\boldsymbol{\theta}}}(x_i \mid \mathbf{U}_i = \mathbf{u}') \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}) E_{\boldsymbol{\Theta}'_{X_i|\mathbf{u}} \sim p(\boldsymbol{\theta}'_{X_i|\mathbf{u}} \mid \mathbf{u}, x_i)} \mathsf{KL}(P_{\boldsymbol{\Theta}'_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}'_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u})) \right). \end{aligned} \tag{A.32}$$

From the proof of Theorem A.2.1 we have that:

$$E_{\boldsymbol{\Theta}_{X_i|\mathbf{u}} \sim p(\boldsymbol{\theta}_{X_i|\mathbf{u}})} \mathsf{KL}(P_{\boldsymbol{\Theta}_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u}) \parallel P_{\tilde{\boldsymbol{\theta}}_{X_i|\mathbf{u}}}(X_i \mid \mathbf{u})) = \delta(\alpha_{x_{i1}|\mathbf{u}}, \ldots, \alpha_{x_{ir_i}|\mathbf{u}}).$$

Using this, together with Eq. (A.19), the expression (A.32) becomes:

$$\sum_i \sum_{\mathbf{u}' \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u}' \mid \mathbf{Q} := \mathbf{q}) \Delta(X_i \mid \mathbf{u}'),$$

where $\Delta(X_i \mid \mathbf{u}')$ is defined as in Eq. (A.17).

□

Theorem A.2.4 *Let $\mathcal{U}$ be the set of nodes which are updateable for at least one candidate query at each querying step. Assuming that the underlying true distribution has the same graphical structure as our network and is not deterministic, then our querying algorithm produces consistent estimates for the CPD parameters of every member of $\mathcal{U}$.*

Proof. Let P^* by the underlying true distribution that is generating the data. Notice that no query node is a descendent of X_i in the interventional case (because we sever the edges from incoming edges to query nodes) or in the selective case (because of the definition of updateable node, and because P^* has the same network structure as our network).

Furthermore, from the definition of a Bayesian network, every node is conditionally independent of its non-decendents given its parents. Thus, when we perform a selective or interventional query $\mathbf{Q} := \mathbf{q}$, and have that the parents of and updateable node X_i take values $\mathbf{u}$, we have that X_i is sampled from the distribution:

$$P^*(X_i \mid \mathbf{Q} := \mathbf{q}, \mathbf{u}) = P^*(X_i \mid \mathbf{u}).$$

So, whenever we update a parameter $\theta_{x_{ij}|\mathbf{u}}$ from data instance $\mathbf{d}$, the value x_{ij} present in $\mathbf{d}$ is generated from $P^*(X_i \mid \mathbf{u})$. Thus, since Bayesian point estimate updating is known to be consistent, the parameter $\theta_{x_{ij}|\mathbf{u}}$ will converge to the true limiting probability $P^*(X_i = x_{ij} \mid \mathbf{u})$.

Thus, each of our point estimate parameters will converge to the correct quantities. We only need to show that we will update each parameter in $\mathcal{U}$ an infinite number of times. Since the true distribution is not deterministic, the only parameters that could possibly not be updated infinitely many times are $\theta_{x_{ij}|\mathbf{u}}$ where $\mathbf{U}$ contains a query node.

In Eq. (A.17), we can use standard results from information theory (e.g., from (Cover & Thomas, 1991)) to show that $\Delta(X \mid \mathbf{u}) \to 0$ as $\alpha_{x_*} \to \infty$ and that $\Delta(X \mid \mathbf{u}) > 0$, where $\mathbf{u}$ is a complete instantiation of X's parents.

Now, suppose we have a domain where we set or select the value of a single node Q. Let us consider a candidate query $Q := q$ and let X_k be a child of Q. We wish to show that this query is asked infinitely often. Our algorithm uses a measure of **model quality** to evaluate the benefit of asking $Q := q$, and this quantity is given by Eq. (A.26):

$$\begin{aligned}
& \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid Q := q) \Delta(X_i \mid \mathbf{u}) && \text{(A.33)} \\
> & \sum_{\mathbf{u} \in Dom[\mathbf{U}_k]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid Q := q) \Delta(X_k \mid \mathbf{u}) && \text{(A.34)} \\
> & \sum_{\mathbf{u} \in Dom[\mathbf{U}_k]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid Q := q) \min_{\mathbf{v} \in Dom[\mathbf{U}_k], \mathbf{v} \text{ consistent with } q} \Delta(X_k \mid \mathbf{v}) && \text{(A.35)} \\
= & \Delta(X_k \mid \mathbf{v}) && \text{(A.36)} \\
= & \epsilon > 0, && \text{(A.37)}
\end{aligned}$$

where the instantiation $\mathbf{v}$ is consistent with q. Now, asking any other query $Q := q'$ causes that query's quality to tend to zero:

$$\sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid Q := q') \Delta(X_i \mid \mathbf{u}) \xrightarrow{+} 0. \tag{A.38}$$

Furthermore, asking $Q := q'$ does not alter any of the parameters $\theta_{X_k \mid \mathbf{v}}$ since it always sets Q to some other value. Thus, ϵ remains constant. Thus, eventually, ϵ will be greater than the score for any other query and so we shall eventually ask the query $Q := q$.

By using a similar argument, we can extend the proof to accomodate sets of candidate queries.

□

A.2.2 Using Log Loss

The theorems in this subsection show that when we use log loss (rather than KL divergence) as our parameter loss function we get an identical algorithm. The upcoming series of theorems follow the same progression as the KL divergence derivation. We first show that the risk decomposes. We then analyze the case for a single family network and then we generalize to general Bayesian networks.

Theorem A.2.5 *The risk when using log loss as the loss function decomposes as:*

$$\mathsf{Risk}_{LL}(p(\boldsymbol{\theta})) = \sum_i H(X_i \mid \mathbf{U}_i). \tag{A.39}$$

Proof.

$$\mathsf{Risk}_{LL}(p(\boldsymbol{\theta})) = E_{\boldsymbol{\Theta}\sim p(\boldsymbol{\theta})}\mathsf{LL}(\Theta \parallel \tilde{\boldsymbol{\theta}}) = E_{\boldsymbol{\Theta}\sim p(\boldsymbol{\theta})}E_{\mathbf{X}\sim P_{\boldsymbol{\Theta}}(\mathbf{X})} - \ln P(\mathbf{X} \mid \tilde{\boldsymbol{\theta}}), \tag{A.40}$$

which is the negative expected loglikelihood of future data and is equal to:

$$\begin{aligned}
&= \int p(\boldsymbol{\theta}) \sum_{\mathbf{x}} -P(\mathbf{x} \mid \boldsymbol{\theta}) \ln P(\mathbf{x} \mid \tilde{\boldsymbol{\theta}})\, d\boldsymbol{\theta} && \text{(A.41)}\\
&= -\sum_{\mathbf{x}} \ln P(\mathbf{x} \mid \tilde{\boldsymbol{\theta}}) \int p(\boldsymbol{\theta}) P(\mathbf{x} \mid \boldsymbol{\theta})\, d\boldsymbol{\theta} && \text{(A.42)}\\
&= -\sum_{\mathbf{x}} P(\mathbf{x}) \ln P(\mathbf{x} \mid \tilde{\boldsymbol{\theta}}) && \text{(A.43)}\\
&= -\sum_{\mathbf{x}} P(\mathbf{x}) \ln P(\mathbf{x}) && \text{(A.44)}\\
&= -\sum_{\mathbf{x}} P(\mathbf{x}) \ln \prod_i P(x_i \mid \mathbf{u}_i) && \text{(A.45)}\\
&= -\sum_i \sum_{x_i} \sum_{\mathbf{u}_i} P(x_i, \mathbf{u}_i) \ln P(x_i \mid \mathbf{u}_i) && \text{(A.46)}\\
&= -\sum_i \sum_{\mathbf{u}_i} P(\mathbf{u}_i) \sum_{x_i} P(x_i \mid \mathbf{u}_i) \ln P(x_i \mid \mathbf{u}_i) && \text{(A.47)}\\
&= -\sum_i H(X_i \mid \mathbf{U}_i). && \text{(A.48)}
\end{aligned}$$

□

Theorem A.2.6 *Consider a simple Bayesian network in which X has parents $\mathbf{Q}$. Define $\Delta_{LL}(X \mid \mathbf{q}) = \mathsf{Risk}_{LL}(X) - \mathsf{ExPRisk}_{LL}(X \mid \mathbf{Q} := \mathbf{q})$. Then:*

$$\Delta_{LL}(X \mid \mathbf{q}) = P_{\tilde{\boldsymbol{\theta}}}(\mathbf{q}) \left(H\left(\frac{\alpha_{x_1|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}, \ldots, \frac{\alpha_{x_r|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}\right) - \sum_j P_{\tilde{\boldsymbol{\theta}}}(x_j \mid \mathbf{q}) H\left(\frac{\alpha'_{x_1|\mathbf{q}}}{\alpha'_{x_*|\mathbf{q}}}, \ldots, \frac{\alpha'_{x_r|\mathbf{q}}}{\alpha'_{x_*|\mathbf{q}}}\right)\right), \tag{A.49}$$

where $\alpha_{x_*|\mathbf{q}} = \sum_i \alpha_{x_i|\mathbf{q}}$. Also, $\alpha'_{x_i|\mathbf{q}} = (\alpha_{x_i|\mathbf{q}} + 1)$ if $i = j$ and $\alpha'_{x_i|\mathbf{q}} = \alpha_{x_i|\mathbf{q}}$ otherwise. Thus $\alpha'_{x_*|\mathbf{q}} = \alpha_{x_*|\mathbf{q}} + 1$.

Proof. This is immediate from Theorem A.2.5 and the fact that:

$$H(X \mid \mathbf{q}) = H\left(\frac{\alpha_{x_1|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}, \ldots, \frac{\alpha_{x_r|\mathbf{q}}}{\alpha_{x_*|\mathbf{q}}}\right). \tag{A.50}$$

□

Now, notice that $\Delta_{LL}(X \mid \mathbf{q})$ is identical to $\Delta(X \mid \mathbf{q})$ from Eq. (A.17). In other words, for this simple network, the difference in expected posterior loss when using log loss is the same as when using KL divergence. Thus, the proof for Theorem A.2.3 can be used to prove the analogous theorem:

Theorem A.2.7 *The change in risk of a Bayesian network over variables $\mathcal{X}$ when asking query $\mathbf{Q} := \mathbf{q}$ is given by:*

$$\begin{aligned} \Delta_{LL}(\mathcal{X} \mid \mathbf{q}) &= \mathsf{Risk}_{LL}(p(\boldsymbol{\theta})) - \mathsf{ExPRisk}_{LL}(p(\boldsymbol{\theta}) \mid \mathbf{q}) &\text{(A.51)}\\ &\approx \sum_i \sum_{\mathbf{u} \in Dom[\mathbf{U}_i]} P_{\tilde{\boldsymbol{\theta}}}(\mathbf{u} \mid \mathbf{Q} := \mathbf{q}) \Delta_{LL}(X_i \mid \mathbf{u}), &\text{(A.52)} \end{aligned}$$

where $\Delta_{LL}(X_i \mid \mathbf{u})$ is as defined in Eq. (A.49). Notice that we actually only need to sum over the updateable X_is since $\Delta_{LL}(X_i \mid \mathbf{u})$ will be zero for all non-updateable X_is.

Thus, we have exactly the same algorithm as before, and so the proof for consistency also holds.

A.3 Structure Estimation Proofs

Theorem A.3.1 *Given a query $\mathbf{Q} := \mathbf{q}$, we can write the probability of a response $\mathbf{x}$ to our query as:*

$$\begin{aligned} &P(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}, \prec) \\ &= \lambda_{\mathbf{Q}} \prod_{i: X_i \notin \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec) Score(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q}), \end{aligned}$$

where $\lambda_{\mathbf{Q}} = \prod_{i: X_i \in \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(Pa(X_i) = \mathbf{U} \mid \prec)$.

Proof. Applying Theorem 8.3.4 and parameter modularity we have:

$$
\begin{aligned}
&P(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}, \prec) \\
&= \sum_{\mathcal{G} \in \prec} P(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}, \mathcal{G}) P(\mathcal{G} \mid \prec) \\
&= \sum_{\mathcal{G} \in \prec} \prod_i P(\mathit{Pa}(X_i) = \mathbf{U}_i^{\mathcal{G}} \mid \prec) \prod_{j : X_j \notin \mathbf{Q}} \mathit{Score}(X_j, \mathbf{U}_j^{\mathcal{G}} \mid \mathbf{x}, \mathbf{q}) \\
&= \sum_{\mathcal{G} \in \prec} \left(\prod_{j : X_j \in \mathbf{Q}} P(\mathit{Pa}(X_j) = \mathbf{U}_j^{\mathcal{G}} \mid \prec) \right) \left(\prod_{i : X_i \notin \mathbf{Q}} P(\mathit{Pa}(X_i) = \mathbf{U}_i^{\mathcal{G}} \mid \prec) \mathit{Score}(X_i, \mathbf{U}_i^{\mathcal{G}} \mid \mathbf{x}, \mathbf{q}) \right) \\
&= \left(\prod_{j : X_j \in \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{j,\prec}} P(\mathit{Pa}(X_j) = \mathbf{U} \mid \prec) \right) \times \\
&\qquad \left(\prod_{i : X_i \notin \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} P(\mathit{Pa}(X_i) = \mathbf{U} \mid \prec) \mathit{Score}(X_i, \mathbf{U} \mid \mathbf{x}, \mathbf{q}) \right).
\end{aligned}
$$

The last step relies on parameter modularity and the observation that:

$$
\sum_{\mathcal{G} \in \prec} \prod_i f(X_i, \mathbf{U}) = \prod_i \sum_{\mathbf{U} \in \mathcal{U}_{i,\prec}} f(X_i, \mathbf{U}).
$$

□

Theorem A.3.2 *Given a query* $\mathbf{Q} := \mathbf{q}$*, the expected posterior loss can be written as:*

$$
\begin{aligned}
&\mathit{ExPLoss}_{\prec}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}) \mid \mathbf{Q} := \mathbf{q}) \\
&\quad = \lambda_{\mathbf{Q}} \sum_{i,j} \sum_{\mathbf{x}} \psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) \prod_{k : X_k \notin \mathbf{Q}} \phi(x_k, \mathbf{w}_k),
\end{aligned}
\tag{A.53}
$$

where,

$$
\psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) = H(X_i \leftrightarrow X_j \mid x_i, x_j, \mathbf{w}_i, \mathbf{w}_j, \prec)
$$

$$
\phi(x_k, \mathbf{w}_k) = \sum_{\mathbf{U} \in \mathcal{U}_{k,\prec}} P(\mathit{Pa}(X_k) = \mathbf{U} \mid \prec) \mathit{Score}(X_k, \mathbf{U} \mid x_k, \mathbf{w}_k).
$$

Proof.

$$
\mathit{ExPLoss}_{\prec}(P(\mathcal{G}, \boldsymbol{\theta}_{\mathcal{G}}) \mid \mathbf{Q} := \mathbf{q}) \tag{A.54}
$$

$$
\begin{aligned}
&= E_{\mathbf{x} \sim P(\mathbf{X} \mid \mathbf{Q} := \mathbf{q}, \prec)} \sum_{i,j} H(X_i \leftrightarrow X_j \mid x_i, x_j, \mathbf{w}_i, \mathbf{w}_j, \prec) && \text{(A.55)} \\
&= \sum_{i,j} \sum_{\mathbf{x}} P(\mathbf{x} \mid \mathbf{Q} := \mathbf{q}, \prec) H(X_i \leftrightarrow X_j \mid x_i, x_j, \mathbf{w}_i, \mathbf{w}_j, \prec) && \text{(A.56)} \\
&= \sum_{i,j} \sum_{\mathbf{x}} H(X_i \leftrightarrow X_j \mid x_i, x_j, \mathbf{w}_i, \mathbf{w}_j, \prec) \times \\
&\qquad \lambda_{\mathbf{Q}} \prod_{k : X_k \notin \mathbf{Q}} \sum_{\mathbf{U} \in \mathcal{U}_{k,\prec}} P(\mathit{Pa}(X_k) = \mathbf{U} \mid \prec) \mathit{Score}(X_k, \mathbf{U} \mid \mathbf{x}, \mathbf{q}) && \text{(A.57)} \\
&= \lambda_{\mathbf{Q}} \sum_{i,j} \sum_{\mathbf{x}} \psi(x_i, x_j, \mathbf{w}_i, \mathbf{w}_j) \prod_{k : X_k \notin \mathbf{Q}} \phi(x_k, \mathbf{w}_k). && \text{(A.58)}
\end{aligned}
$$

□

Bibliography

Applegate, D., & Kannan, R. (1991). Sampling and integration of near log-concave functions. *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing* (pp. 156–163).

Arnborg, S., Corneil, D., & Proskurowski, A. (1987). Complexity of finding embeddings in a k-tree. *SIAM Journal of Algebraic and Discrete Methods*, *8*, 277–284.

Atkinson, A. C., & Bailey, R. A. (2001). One hundred years of the design of experiments on and off the pages of "Biometrika". *Biometrika*. In press.

Bertsekas, D., & Tsitsiklis, J. (1996). *Neuro-dynamic programming*. Athena.

Blake, C., Keogh, E., & Merz, C. (1998). UCI repository of machine learning databases.

Boutilier, C., Dean, T., & Hanks, S. (1999). Decision theoretic planning: Structural assumptions and computational leverage. *Journal of Artificial Intelligence Research*, *10*.

Boutilier, C., Friedman, N., Goldszmidt, M., & Koller, D. (1996). Context-specific independence in Bayesian networks. *Proceedings of Uncertainty in Artificial Intelligence*.

Box, G. E. P., & Draper, N. R. (1987). *Empirical model-building and response surfaces*. Wiley.

Boyan, J. A. (1995). Active learning for optimal control in acyclic domains. *Proceedings of AAAI Symposium on Active Learning*.

Breiman, L. (1997). No Bayesians in foxholes. *IEEE Expert November/December issue, Trends and Controversies*.

Bryant, C. H., Muggleton, S. H., Page, C. D., & Sternberg, M. J. E. (1999). Combining active learning with inductive logic programming to close the loop in machine learning. *Proceedings of AISB'99 Symposium on AI and Scientific Creativity* (pp. 59–64).

Buntine, W. (1991). Theory refinement on Bayesian Networks. *Proceedings of Uncertainty in Artificial Intelligence*.

Burges, C. J. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167.

Campbell, C., Cristianini, N., & Smola, A. (2000). Query learning with large margin classifiers. *Proceedings of the Seventeenth International Conference on Machine Learning*.

Cauwenberghs, G., & Poggio, T. (2001). Incremental and decremental support vector machine learning. *Advances in Neural Information Processing Systems*.

Chaloner, K., & Verdinelli, I. (1995). Bayesian experimental design: a review. *Statistical Science*, *10*, 273–304.

Chan, P. K., & Stolfo, S. J. (1998). Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. *In KDD98*.

Chaudhuri, S., Narasayya, V., & Motwani, R. (1998). Random sampling for histogram construction: How much is enough? *ACM Sigmod*.

Cohn, D. (1997). Minimizing statistical bias with queries. *Advances in Neural Information Processing Systems*.

Cohn, D., Ghahramani, Z., & Jordan, M. (1996). Active learning with statistical models. *Journal of Artificial Intelligence Research*, *4*.

Cooper, G. (1990). Probabilistic inference using belief networks is NP-hard. *Artificial Intelligence*, *42*, 393–405.

Cooper, G. F., & Yoo, C. (1999). Causal discovery from a mixture of experimental and observational data. *Proceedings of Uncertainty in Artificial Intelligence*.

Cortes, C., & Vapnik, V. (1995). Support vector networks. *Machine Learning*, *20*, 1–25.

Cover, T., & Thomas, J. (1991). *Information theory*. Wiley.

Dagan, I., & Engelson, S. (1995). Committee-based sampling for training probabilistic classifiers. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 150–157). Morgan Kaufmann.

Dean, T., & Kanazawa, K. (1989). A model for reasoning about persistence and causation. *Computational Intelligence*, *5*.

DeGroot, M. H. (1970). *Optimal statistical decisions*. New York: McGraw-Hill.

Dempster, A., Laird, N., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*.

Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis*. Wiley, New York.

Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. *Proceedings of the Seventh International Conference on Information and Knowledge Management*. ACM Press.

Freund, Y., Seung, H., Shamir, E., & Tishby, N. (1997). Selective sampling using the Query by Committee algorithm. *Machine Learning*, *28*, 133–168.

Friedman, J. (1996). *Another approach to polychotomous classification* (Technical Report). Department of Statistics, Stanford University.

Friedman, N., & Koller, D. (2000). Being Bayesian about network structure. *Proceedings of Uncertainty in Artificial Intelligence*.

Friedman, N., Nachman, I., & Pe'er, D. (1999). Learning Bayesian network structure from massive datasets: The "sparse candidate" algorithm. *Proceedings of Uncertainty in Artificial Intelligence*.

Geman, S., & Geman, D. (1987). Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *Readings in Computer Vision: Issues, Problems, Principles and Paradigms*.

Goldstein, E. B. (1999). *Sensation and perception (5^{th} edition)*. Brooks/Cole.

Guestrin, C., Koller, D., & Parr, R. (2001). Max-norm projections for factored MDPs. *Proceedings of the International Joint Conference on Artificial Intelligence*.

Hastie, T., & Tibshirani, R. (1998). Classification by pairwise coupling. *Advances in Neural Information Processing Systems 10*.

Heckerman, D. (1988). An empirical comparison of three inference methods. *Proceedings of the Fourth on Uncertainty in Artificial Intelligence*.

Heckerman, D. (1995). *A Bayesian approach to learning causal networks* (Technical Report MSR-TR-95-04). Microsoft Research.

Heckerman, D. (1998). A tutorial on learning with Bayesian networks. In M. I. Jordan (Ed.), *Learning in graphical models*. Kluwer Academic Publishers.

Heckerman, D., Breese, J., & Rommelse, K. (1994). *Troubleshooting Under Uncertainty* (Technical Report MSR-TR-94-07). Microsoft Research.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, *20*, 197–243.

Herbrich, R., & Graepel, T. (2001). Large scale Bayes Point Machines. *Advances in Neural Information Processing Systems 13*.

Herbrich, R., Graepel, T., & Campbell, C. (1999). Bayes point machines: Estimating the Bayes point in kernel space. *International Joint Conference on Artificial Intelligence Workshop on Support Vector Machines* (pp. 23–27).

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., & Rommelse, K. (1998). The Lumiere project: Bayesian user modeling for inferring the goals and needs of software users. *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence* (pp. 256–265).

Horvitz, E., Ruokangas, E., Srinivas, C., & Barry, S. (1992). A decision-theoretic approach to the display of information for time-critical decisions: The Vista project. *Proceedings of SOAR-92*.

Horvitz, E., & Rutledge, G. (1991). Time dependent utility and action under uncertainty. *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann.

Howard, R. (1970). Decision analysis: Perspectives on inference, decision, and experimentation. *Proceedings of the IEEE*, *58*, 632–643.

Hua, K. A., Vu, K., & Oh, J.-H. (1999). Sammatch: A flexible and efficient sampling-based image retrieval technique for image databases. *Proceedings of ACM Multimedia*.

Huang, C., & Darwiche, A. (1996). Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, *15*, 225–263.

Ishikawa, Y., Subramanya, R., & Faloutsos, C. (1998). Mindreader: Querying databases through multiple examples. *VLDB*.

Joachims, T. (1998). Text categorization with support vector machines. *Proceedings of the European Conference on Machine Learning*. Springer-Verlag.

Joachims, T. (1999). Transductive inference for text classification using support vector machines. *Proceedings of the Sixteenth International Conference on Machine Learning* (pp. 200–209). Morgan Kaufmann.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., & Saul, L. K. (1998). An introduction to variational methods for graphical models. In M. I. Jordan (Ed.), *Learning in graphical models*. Kluwer Academic Publishers.

Kaebling, L. P., Littman, M. L., & Moore, A. (1996). Reinforcement learning: a survey. *Journal of AI Research*, *4*, 237–285.

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, *101*, 99–134.

Kearns, M., & Koller, D. (1999). Efficient reinforcement learning in factored MDPs. *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence* (pp. 740–747).

Kearns, M., & Singh, S. (1998). Near-optimal reinforcement learning in polynomial time. *Proceedings of the Fifteenth International Conference on Machine Learning* (pp. 260–268). Morgan Kaufmann, San Francisco, CA.

Kjaerulff, U. (1990). *Triangulation of graphs – algorithms giving small total state space* (Technical Report TR R 90-09). Department of Mathematics and Computer Science, Strandvejen, Aalborg, Denmark.

Koller, D., & Parr, R. (1999). Computing factored value functions for policies in structured MDPs. *Proceedings of the International Joint Conference on Artificial Intelligence* (pp. 1332–1339).

Koller, D., & Pfeffer, A. (1997). Object-oriented Bayesian networks. *Proceedings of the 13th Annual Conference on Uncertainty in AI (UAI)* (pp. 302–313).

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *Annals of Mathematical Statistics*, *22*, 76–86.

Latombe, J.-C. (1991). *Robot motion planning*. Kluwer Academic Publishers.

Lauritzen, S. L. (1996). *Graphical models*. Oxford: Clarendon Press.

Lauritzen, S. L., & Spiegelhalter, D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society*, *B 50*.

LeCun, Y., Haffner, P., Bottou, L., & Bengio, Y. (1999). Gradient-based learning for object detection, segmentation and recognition. *Feature Grouping*.

LeCun, Y., Jackel, L. D., Bottou, L., Brunot, A., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P., & Vapnik, V. (1995). Comparison of learning algorithms for handwritten digit recognition. *International Conference on Artificial Neural Networks* (pp. 53–60). Paris.

Lehmann, E. L. (1986). *Testing statistical hypotheses*. Springer-Verlag.

Lehmann, E. L., & Casella, G. (1998). *Theory of point estimation*. Springer-Verlag.

Leu, J.-G. (1991). Computing a shape's moments from its boundary. *Pattern Recognition*, Vol.24, No.10,pp.949–957.

Lewis, D. (1995). A sequential algorithm for training text classifiers: Corrigendum and additional data. *Special Interest Group on Information Retrieval Forum*.

Lewis, D., & Catlett, J. (1994). Heterogeneous uncertainty sampling for supervised learning. *Proceedings of the Eleventh International Conference on Machine Learning* (pp. 148–156). Morgan Kaufmann.

Lewis, D., & Gale, W. (1994). A sequential algorithm for training text classifiers. *Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval* (pp. 3–12). Springer-Verlag.

Li, C., Chang, E., Garcia-Molina, H., & Wiederhold, G. (2001). Clustering for approximate similarity queries in high-dimensional spaces. *IEEE Transaction on Knowledge and Data Engineering (to appear)*.

Liere, R. (2000). *Active learning with committees: An approach to efficient learning in text categorization using linear threshold algorithms*. Oregon State University Ph.D Thesis.

Liere, R., & Tadepalli, P. (1997). Active learning with committees for text categorization. *Proceedings of AAAI* (pp. 591–596).

Ma, W. Y., & Zhang, H. (1998). Benchmarking of image features for content-based retrieval. *Proceedings of Asilomar Conference on Signal, Systems* & *Computers*.

MacKay, D. (1992). Information-based objective functions for active data selection. *Neural Computation*, *4*, 590–604.

Manjunath, B., Wu, P., Newsam, S., & Shin, H. (2001). A texture descriptor for browsing and similarity retrieval. *Signal Processing Image Communication*.

Manning, C., & Schütze, H. (1999). *Foundations of statistical natural language processing*. The MIT Press.

McAllester, D. (1999). PAC-Bayesian model averaging. *Computational Learning Theory*.

McCallum, A., & Nigam, K. (1998). Employing EM in pool-based active learning for text classification. *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*, *28*, 203–226.

Moore, A. (1991). *An introductory tutorial on kd-trees* (Technical Report No. 209). Computer Laboratory, University of Cambridge, Cambridge, UK.

Moore, A. W., Schneider, J. G., Boyan, J. A., & Lee, M. S. (1998). Q2: Memory-based active learning for optimizing noisy continuous functions. *Proceedings of the Fifteenth International Conference on Machine Learning*. Morgan Kaufmann.

Morjaia, M., Rink, F., Smith, W., Klempner, J., Burns, C., & Stein, J. (1993). Commercialization of EPRI's generator expert monitoring system. *Expert System Application for the Electric Power Industry, EPRI, 1993.*.

Murphy, K., & Weiss, Y. (1999). Loopy belief propagation for approximate inference: an empirical study. *Proceedings of Uncertainty in Artificial Intelligence*.

Nakajima, C., Norihiko, I., Pontil, M., & Poggio, T. (2000). Object recognition and detection by a combination of support vector machine and rotation invariant phase only correlation. *Proceedings of International Conference on Pattern Recognition*.

Neal, R. (1993). *Probabilistic inference using Markov Chain Monte Carlo methods.* (Technical Report CRG-TR-93-1). Department of Computer Science, University of Toronto.

Odewahn, S., Stockwell, E., Pennington, R., Humphreys, R., & Zumach, W. (1992). Automated star/galaxy discrimination with neural networks. *Astronomical Journal*, *103*, 318–331.

Ortega, M., Rui, Y., Chakrabarti, K., Warshavsky, A., Mehrotra, S., & Huang, T. S. (1999). Supporting ranked boolean similarity queries in mars. *IEEE Transaction on Knowledge and Data Engineering*, *10*, 905–925.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann.

Pearl, J. (2000). *Causality: Models, reasoning, and inference*. Cambridge University Press.

Pfeffer, A. J. (2000). *Probabilistic reasoning for complex systems*. Stanford University Ph.D Thesis.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. *Advances in Large Margin Classifiers*.

Platt, J., Cristianini, N., & Shawe-Taylor, J. (2000). Large margin DAGS for multiclass classification. *Advances in Neural Information Processing Systems, 12*.

Porkaew, K., Chakrabarti, K., & Mehrotra, S. (1999a). Query refinement for multimedia similarity retrieval in mars. *Proceedings of ACM Multimedia*.

Porkaew, K., Mehrota, S., & Ortega, M. (1999b). Query reformulation for content based multimedia retrieval in MARS. *ICMCS*, 747–751.

Porter, M. (1980). An algorithm for suffix stripping. *Automated Library and Information Systems* (pp. 130–137).

Puterman, M. L. (1994). *Markov decision processes: Discrete stochastic dynamic programming*. New York: Wiley.

Quinlin, R. (1986). Induction of decision trees. *Machine Learning*, *1*, 81–106.

Raab, G. M., & Elton, R. A. (1993). Bayesian-analysis of binary data from an audit of cervical smears. *Statistics Medicine*, *12*, 2179–2189.

Rocchio, J. (1971). Relevance feedback in information retrieval. *The SMART retrieval system: Experiments in automatic document processing*. Prentice-Hall.

Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). A Bayesian approach to filtering junk e-mail. *AAAI-98 Workshop on Learning for Text Categorization*.

Salton, G., & Buckley, C. (1988). Term weighting approaches in automatic text retrieval. *Information Processing and Management* (pp. 513–523).

Schohn, G., & Cohn, D. (2000). Less is more: Active learning with support vector machines. *Proceedings of the Seventeenth International Conference on Machine Learning*.

Seung, H., Opper, M., & Sompolinsky, H. (1992). Query by committee. *Proceedings of Computational Learning Theory* (pp. 287–294).

Shachter, R., & Peot, M. (1989). Simulation approaches to general probabilistic inference on belief networks. *Fifth Workshop on Uncertainty in Artificial Intelligence*.

Smith, J., & Chang, S.-F. (1996). Automated image retrieval using color and texture. *IEEE Transaction on Pattern Analysis and Machine Intelligence*.

Sollich, P. (1999). Probabilistic interpretation and Bayesian methods for support vector machines. *International Conference on Artificial Neural Networks 99*.

Spirtes, P., Glymour, C., & Scheines, R. (1993). *Causation, prediction and search*. MIT Press.

Tamura, H., Mori, S., & Yamawaki, T. (1978). Texture features corresponding to visual perception. *IEEE Transaction on Systems Man Cybernet (SMC)*.

Tong, S., & Chang, E. (2001). Support vector machine active learning for image retrieval. *ACM Multimedia*.

Tong, S., & Koller, D. (2001a). Active learning for parameter estimation in Bayesian networks. *Advances in Neural Information Processing Systems 13* (pp. 647–653).

Tong, S., & Koller, D. (2001b). Active learning for structure in Bayesian networks. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence* (pp. 863–869).

Tong, S., & Koller, D. (2001c). Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, To appear.

Vapnik, V. (1982). *Estimation of dependences based on empirical data*. Springer Verlag.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer, New York.

Vapnik, V. (1998). *Statistical learning theory*. Wiley.

Wald, A. (1950). *Statistical decision functions*. Wiley, New York.

Wang, J., Li, J., & Wiederhold, G. (2000). Simplicity: Semantics-sensitive integrated matching for picture libraries. *ACM Multimedia Conference*.

Wu, L., Faloutsos, C., Sycara, K., & Payne, T. R. (2000). Falcon: Feedback adaptive loop for content-based retrieval. *The* 26^{th} *VLDB Conference*.

Yang, Y., & Pedersen, J. (1997). A comparative study on feature selection in text categorization. *Proceedings of the Fourteenth International Conference on Machine Learning*. Morgan Kaufmann.

Yedidia, J., Freeman, W., & Weiss, Y. (2001). Generalized belief propagation. *Advances in Neural Information Processing Systems 13*.

Zhang, N. L., & Poole, D. (1994). A simple approach to Bayesian network computations. *Proceedings of the Tenth Canadian Conference on Artificial Intelligence* (pp. 171–178).

Index