Computing Conditional Probabilities in Large Domains by Maximizing Rényi's Quadratic Entropy

Charles Lawrence Zitnick III

CMU-RI-TR-03-20

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15217

May 23, 2003

Thesis Committee:

Takeo Kanade, Chair
Greg Cooper, University of Pittsburgh
Jeff Schneider, Carnegie Mellon University
Manuela Veloso, Carnegie Mellon University

Abstract

In this dissertation we discuss methods for efficiently approximating conditional probabilities in large domains by maximizing the entropy of the distribution given a set of constraints. The constraints are constructed from conditional probabilities, typically of low-order, that can be accurately computed from the training data. By appropriately choosing the constraints, maximum entropy methods can balance the tradeoffs in errors due to bias and variance.

Standard maximum entropy techniques are too computationally inefficient for use in large domains in which the set of variables that are being conditioned upon varies. Instead of using the standard measure of entropy first proposed by Shannon, we use a measure that lies within the family of Rényi's entropies. If we allow our probability estimates to occasionally lie outside the range from 0 to 1, we can efficiently maximize Rényi's quadratic entropy relative to the constraints using a set of linear equations.

We develop two algorithms, the inverse probability method and recurrent linear network, for maximizing Rényi's quadratic entropy without bounds. The algorithms produce identical results. However, depending on the type of problem, one method may be more computationally efficient than the other. We also propose an extension to the algorithms for partially enforcing the constraints based on our confidence in them.

Our algorithms are tested on several applications including: collaborative filtering, image retrieval and language modeling.

Acknowledgements

I would like to thank Takeo Kanade for his continued support throughout my years as a graduate student. He very patiently critiqued my research approach and results and accepted my changes in research topics. Without his trust and guidance this thesis would not be possible.

For his part in introducing me to research and nurturing my ideas I would like to thank Jon Webb. His enthusiasm in my early research gave me the confidence to pursue my ideas further.

In addition, I would like to sincerely thank my committee members Jeff Schneider, Manuela Veloso and Greg Cooper for their insights and time, and Jim Gemmell and Jim Gray for their great conversations and advice during my time spent at BARC.

My family, including my parents Sue and Chuck Zitnick along with my sister Karen and brother Dave, have given me complete support and encouragement throughout these years.

Finally, I would like to thank Krista for her love, support and her ability to listen patiently to my many ramblings about isolation problems and recurrent networks.

I'd like to acknowledge Microsoft Corporation and Hewlett-Packard Company for use of their databases in my experiments.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

My thesis is conditional probabilities between variables in large domains can be modeled efficiently and accurately using low-order interactions. The size of a problem's domain is measured by the number of variables associated with the problem. We consider a large domain as consisting of at least several hundred variables. Each variable represents some feature of the world, such as: the grass is wet, a customer purchased book A, the word "cat" was spoken.

A probability is a value ranging from zero to one corresponding to the likelihood of a feature occurring. A conditional probability is the probability of some variables given information about other known or evidence variables. The probability of it raining regardless of location may be $P(raining) = 0.1$. If we condition the probability on the knowledge that we are in Pittsburgh we may find $P(raining|Pittsburgh) = 0.3$.

Sometimes the knowledge of an evidence variable provides no additional information for computing the probability of a hidden variable. For example, when computing the probability of it raining the knowledge that its Tuesday provides no additional information. In this example we would say "raining" is independent of "Tuesday". Two variables may also be conditionally independent, that is two variables may be independent given the knowledge of the other evidence variables. If we are computing the probability of the grass being wet, the knowledge of whether its cloudy or not can help us refine our estimate. If it's cloudy, it is more likely that it is raining and thus the grass being wet. However, if we know whether it has rained, the knowledge of "cloudy" doesn't provide any additional information. Then the probability of "wet grass" is conditionally independent of "cloudy" given the knowledge of "rain", $P(wetgrass|cloudy, rain) = P(wetgrass|rain)$.

Typically, in problems with large domains many variables will be independent and an even larger number are conditionally independent. A variable is considered to be of high-order if it is

directly dependent on many variables and low-order if it is not. We will demonstrate that for many real world problems, dependencies between variables can be modeled accurately using low-order interactions. More over, the training data will rarely exist in large enough quantities to support or learn high-order interactions.

To demonstrate the amount of data that would be needed to learn a high-order model let us consider the following case: We have a database of 10,000 books and we'd like to compute the probability of a customer purchasing each book given five books they've previously purchased. The probabilities are computed using data from previous customers who have purchased the same five books. Given that there are 10,000 books total, there are approximately $10,000^5 = 10^{20}$ different sets of five books our current customer may have purchased. If each training case contained about ten books bought by a single previous customer, then we'd need approximately $\frac{10^{20}}{\binom{10}{5}} \approx 4 \times 10^{18}$ training cases to cover the entire set of possibilities. Even if we condition upon three books, the approximate number of training cases is still 83,000,000,000. Clearly, the number of high-order interactions supported by the data will be small fraction of the total possible.

Regardless of the computational costs of computing high-order interactions, it is impossible to accurately model them since there will never exist enough training data. This tradeoff between model complexity and data to support the model is well known in literature [22]. This tradeoff is commonly viewed as a balance between *bias* and *variance* errors. The bias refers to the error related to the representational power of the model. If a model has low bias, that means on average it will accurately model the training data. Variance refers to the change in the model as the data varies. A model with low variance will be robust with respect to noise in the training data.

1.1 Balancing Bias and Variance Errors Through Constraints

Our variables are related by some hidden underlying joint probability distribution P^*. Our goal is to model P^* from the training data. Given n binary variables, the size of the joint probability distribution P^* is 2^n. For $n = 100$ the number of entries in P^* is equal to $2^{100} > 10^{30}$, which outnumbers the size of any real-world training data sets. While we may not be able to directly compute each individual entry of P^* we can place constraints on sets of its entries. For example we may have enough data to accurately compute $P(rain)$ from the observed probability $\bar{P}(rain)$ that is computed directly from the training data. Thus we can impose a constraint on P such that $P(rain)$ is equal to $\bar{P}(rain)$.

The training data can be analyzed to find a set of constraints with high confidence. That is, a set of constraints that have corresponding values that can be accurately computed. Thus, the confi-

dence of a constraint is related to the variance of its value. In our above example, the constraint's value is $\bar{P}(rain)$, the observed probability. The variance of the observed probability is equal to $\frac{P(rain)(1.0-P(rain))}{m}$, where m is the number of times we observed the value of rain in the training data. Since the value of $P(rain)$ is unknown (it is the value we're trying to estimate) we may only be able place upper bounds on the variance of the constraint's value. Regardless, a set of constraints may be found with values having low variance. By basing our algorithm on these low variance constraints, the errors of the algorithm due to variance will be minimal.

The remaining problem is that the set of high confidence constraints will still not fully constrain our estimate of P^*. For all but the smallest problem domains, the constraints will not even come close to fully determining P^*.

If we make assumptions about the probability distribution that aren't supported by the constraints, then the results may have large errors due to bias. Ideally, we'd like to find a distribution that obeys the constraints while assuming nothing else. Imagine a case when we have no constraints other than the probabilities must sum to one, such as computing the probability of a roll of a die. What distribution should be given? Given the lack of information, most people would resort to assigning an equal probability to all sides of the die. This uniform distribution is viewed as the least committal. Similarly, the distribution that lies closest to the uniform distribution but obeys the constraints can be viewed as the least committal or as making the fewest assumptions given the constraints.

The distance between any distribution and the uniform distribution may be measured by their relative entropy. Entropy is a measure of the average amount of information needed to describe the variables at any particular time. For example, the amount of information needed to describe the outcome of a fair die will be larger than that required for a loaded die which yields "6" half the time. Since the uniform distribution is the distribution with highest entropy we may simplify the task to just maximizing the entropy of the constrained distribution [33, 44]. Why maximize entropy? It has been proposed [57] that maximum entropy (ME) is the only solution that obeys a set of "common sense" rules for probabilities. As a measure of information it is the only solution that is consistent with a set of fundamental rules developed by Shannon [73]. Jaynes [32] has stated that the ME solution "agrees with everything that is known, but carefully avoids anything that is unknown." One consequence of the ME solution finding the smoothest distribution, i. e. closest to the uniform, is that variables will be assumed to be independent until proven otherwise by the set of constraints.

Let us provide an example for how maximum entropy can be used. Consider the problem of trying to figure out if your neighbor's dog is outside. You've noticed that when its raining the dog

Dog	¬Rain	Rain
¬Mailman	p_1	p_2
Mailman	p_3	p_4

$p_2 + p_4 = 0.03$
$p_3 + p_4 = 0.08$
$p_1 + p_2 + p_3 + p_4 = 0.3$

Figure 1.1: Probability diagram for dog example: Each box in the diagram corresponds to a different combination of values for "rain" and "mailman" while the dog is out.

is rarely outside, $P(dog|rain) = 0.1$. You've also noticed that the dog likes to run out of the house to chase the mailman, $P(dog|mailman) = 0.8$. The probability of rain and mailman are independent with values 0.3 and 0.1 respectively. We also know from our experience that the dog is outside about a third of the time in general, $P(dog) = 0.3$. Suppose one day its raining and your friend stops by and tells you the mailman is there. What is the probably of the dog being outside? The problem is when its raining you rarely go outside, and you've never seen the mailman in the rain. Thus you have no firsthand data to compute $P(dog|rain, mailman)$. Regardless of this fact, most people would still offer a guess to the probability.

Figure 1.1 illustrates the problem. The probability p_1 corresponds to $P(dog, \neg rain, \neg mailman)$. That is, the probability of the dog being out, the mailman isn't around and it isn't raining. p_2, p_3 and p_4 represent the other combinations of rain and mailman while the dog is out. From our knowledge above we can construct the constraint $p_2 + p_4 = 0.03$ since $P(dog|rain) = 0.1$ and $p_3 + p_4 = 0.08$ since $P(dog|mailman) = 0.8$. We also know the probability of the dog being outside is 0.3, which implies $p_1 + p_2 + p_3 + p_4 = 0.3$. Thus we have three constraints with four variables. In a strict Bayesian sense the problem is not solvable and the real answer could lie anywhere from 0 to 1. However, we can find a solution by maximizing the entropy of the distribution while enforcing our constraint above. We will use the measure of entropy proposed by Shannon [73]:

$$H(x) = -\sum_{x \in [1,4]} p_x \log(p_x) \tag{1.1}$$

The solution with maximum entropy is shown in figure 1.2.

Using:

$$P(dog|rain, mailman) = \frac{P(dog, rain, mailman)}{P(rain, mailman)} \tag{1.2}$$

we find our ME estimate of $P(dog|rain, mailman)$ to be 0.267. This is a reasonable guess, given

Dog	¬Rain	Rain
¬Mailman	0.198	0.022
Mailman	0.072	0.008

$0.022 + 0.008 = 0.03$
$0.072 + 0.008 = 0.08$
$0.198 + 0.022 + 0.072 + 0.008 = 0.3$

Figure 1.2: Distribution with maximum entropy for dog example.

we'd expect the dog to be out more likely than if we just knew it was raining but less likely than if we just knew the mailman was outside. Throughout our everyday lives we're asked to make these type of calculations even though the amount of data we have is lacking. ME is one possible method to find reasonable guesses under these circumstances.

Unfortunately, the computational complexity of computing the ME distribution given a set of constraints is exponential with respect to the number of variables. This is due to the fact that the ME algorithm requires summing over the entire joint probability distribution [13, 14, 63]. In previous works this has been simplified to only include entries that occur in the training data set [68]. Even with this approximation, ME still requires a large amount of computation.

Approximately ten years after Shannon first introduced his measure of entropy 1.1, a mathematician named Rényi [64, 65, 66] developed a generalization of Shannon's measure. Within the family of entropies found by Rényi lies the following measure H_2:

$$H_2(x) = -\log(\sum_x p_x^2) \tag{1.3}$$

Rényi's family of entropies are found by relaxing one of Shannon's three properties for measures of entropy. As a result, some of the results for Shannon's entropy, such as $H(X, Y) = H(X) + H(Y|X)$ don't hold for H_2. For most applications this is will not be an issue.

Since we're maximizing the entropy we can drop the log from the equation leaving us with the task of minimizing the squared probabilities. If we ignore the fact that all probabilities must lie within the range of 0 to 1, minimizing the squared probabilities with respect to the constraints reduces to a set of linear equations that can be efficiently solved. The resulting values cannot be directly interpreted as probabilities since their values may lie below 0 or above 1, but we can view them as approximations of the true probabilities. As we will explore in later chapters, the results of minimizing the squared probabilities to find approximations of the entire joint distribution

can have varying results. However, if our goal is to compute the conditional probability of each variable individually given the set of evidence variables, the results can be as accurate as those found using Shannon's measure. Therefore for this special case, we can view minimizing the squared probabilities given the constraints as an efficient approximation of the ME method.

1.2 Applications of Maximum Entropy

Do applications exist in which we are only concerned with computing the conditional probability of each variable individually? There are many applications that fall under this category: collaborative filtering, language modeling, and image retrieval to name a few.

Collaborative filtering is the task of predicting a user's actions based on their and others' previous actions. A typical example is predicting what books a user might buy given their past buying history and the purchasing history of others. That is, the probability of a user purchasing each book conditioned upon past purchases. When deciding whether to recommend a specific book A to a user, we're only concerned with the probability of a user liking that particular book. The probability of the user liking book A along with some other book B is of little relevance.

For the task of image retrieval, we're only concerned with the probability of a user finding each particular image desirable, and not a set of images. In language modeling the task is to predict the next word in the sentence given some set of previous words. Once again the probability of each individual word is what we're concerned with.

Language modeling in many ways is an ideal task for ME [1, 46, 47, 68]. Given vocabularies of 20,000 words, the amount of data needed to cover every possible combination of just four previous words ranges well over a trillion. Given the substantially smaller subset of constraints that can be supported by the data, ME can find accurate estimations.

There is a fundamental difference between language modeling, for which ME has been previously applied, and other tasks. The nature of language modeling is that the features are either strictly inputs or outputs. In contrast, the role of each feature changes for the tasks of collaborative filtering and image retrieval. In the book example, if we know a user purchased book A then the variable corresponding to book A will be an input, otherwise if a user has not purchased the book it will be an output. In standard ME approaches the parameters for the model would need to be recomputed for every combination of inputs and outputs. Given the increased efficiency of our approach based on Rényi's entropy instead of Shannon's entropy, these tasks with varying inputs become computationally feasible. Therefore, the use of ME can be applying to a wider range of applications than those previously considered.

1.3 Outline of Work

In the following chapter, we will examine the relationship between Shannon's entropy and Rényi's entropy for computing both the joint distribution and conditional probabilities. We will also discuss the relation of Bayesian methods to maximum entropy.

In chapter 3, we'll describe two algorithms for computing Rényi's quadratic entropy without bounds. The methods produce identical results, but are computationally more efficient on different types of problems. We will propose an extension to the algorithms that allows us to partially enforce constraints based on their confidence values.

Chapter 4 discusses the issues related to using complex feature functions as constraint functions. A complex feature function is a function that has multiple variables as inputs. Several methods for finding complex functions that are useful to the algorithms will be discussed.

Concerns while implementing the algorithms in large domains are addressed in chapter 5. Several methods for increasing efficiency are proposed.

Results using Rényi's quadratic entropy without bounds are shown in chapter 6. We compare our method to Bayesian networks, naive Bayes and nearest neighbor approaches using synthetic data. Our method is compared against other collaborative filtering algorithms using two databases containing data on web browsing behavior and movie ratings. To demonstrate the methods in a large domain we show results for the task of image retrieval on a 10,000 image database. Finally, we examine the creation of complex feature functions in the language modeling task.

The main contributions of the dissertation are as follows:

An Efficient Approximation to Maximum Entropy - Rényi developed a family of entropy measures by generalizing the properties of entropy first proposed by Shannon. Within this family lies an entropy measure called Rényi's quadratic entropy. If we ignore the constraints that all probabilities must lie between 0 and 1, we may maximize this measure relative to our constraints using a set of linear functions that can be solved in polynomial time.

Computing Accurate Conditional Probabilities - Using Rény's quadratic entropy without bounds is largely inaccurate for computing estimates to the joint distribution. However, when computing the conditional probability of each variable given the set of evidence variables, Rényi's measure without bounds produces accurate results similar to those using Shannon's measure.

Recurrent Linear Network and Inverse Probability Method - We proposed two methods for finding estimates of conditional probabilities, the recurrent linear network and the inverse probability method. The recurrent linear network is an iterative approach that is most efficient when many variables have know values. The inverse probability method is a closed-form solution that is more

efficient when the number of evidence variables is small.

Learning Efficiency - The parameters for either method can be learned quickly using a matrix of pairwise probability values generated from the data. This matrix of probability values can be quickly updated given new data.

Constraint Confidence - Each constraint has a corresponding confidence value that controls the degree to which it affects the final outcome. The confidence values are based on the estimated variance of the constraint values. Thus new constraints can be added to the algorithm without risking an increase in error due to variance.

Experimental Results - We demonstrate the algorithms on several applications including: collaborative filtering, image retrieval and language modeling. We demonstrate the algorithm is capable of handling large domains, i.e. with over 10,000 variables, within the image retrieval and language modeling applications.

Chapter 2

Maximum Entropy

Our world is described by a set of variables $X = \{X_1, \ldots, X_a\}$ with a corresponding set of values $x = \{x_1, \ldots, x_a\}$. Each variable, X_i, represents an observable feature of the world. A variable is assigned the value of one if its corresponding world feature occurs, and a value of zero if it doesn't occur. We will abbreviate $P(X = x)$ as $P(x)$. The variables are related based on some hidden underlying world model represented by the joint distribution $P^* = \{p_1, \ldots, p_n\}$, with $n = 2^a$. Given a training set $T = \{t_1, \ldots, t_m\}$ of variable values, where $t_{j,i}$ is the value of variable X_i at time j, we can compute the observed probability distribution $\bar{P}$. Since the training set is of finite size, $\bar{P}$ is only a crude approximation of the underlying probability distribution P^*.

$$\bar{P}(X = x) = \frac{1}{m} \sum_{j \in m} \delta(x, t_j) \tag{2.1}$$

where

$$\delta(x, t_j) = \left\{ \begin{array}{ll} 1 & x = t_j \\ 0 & x \neq t_j \end{array} \right\} \tag{2.2}$$

At any particular time, some of the variables will be observed while others are not. The evidence variables representing the set of observed or known variables will form the set X_E. The hidden variables that are unobserved will form the set $X_H = X - X_E$. It is our goal to compute the value of the conditional probabilities $P(X_i \mid X_E)$ for all $X_i \in X_H$.

Before discussing the the more specific case of computing $P(X_i \mid X_E)$ we will address the more general case of computing $P(X)$.

2.1 Maximum Entropy Methods

We assume our variables are binary valued, thus the number of entries in the joint distribution $P(X)$ is $n = 2^a$. Clearly, for all but the smallest a there will not be enough data to accurately compute $P^*(X)$ directly from $\bar{P}(X)$. For a 25 variable problem we would need at a minimum 33,554,432 training examples to just observe each possible x once. Since we are concerned with problems potentially containing thousands of variables we need an alternative approach.

While we may not be able to accurately compute $P^*(x)$ for most x, there typically exists a set of feature functions $F = \{f_1, \ldots, f_c\}$ such that $P^*(f_i(X)) \approx \bar{P}(f_i(X))$ can be accurately computed. For example, a function f_i might have the following form:

$$f_i(x) = \left\{ \begin{array}{cc} 1 & x_1 = 1 \text{ and } x_3 = 0 \\ 0 & otherwise \end{array} \right\} \tag{2.3}$$

The set of functions F and their observed values $\bar{P}(f_i)$ form a set of constraints on our computed distribution P:

$$P(f_i) = \bar{P}(f_i) = \frac{1}{m} \sum_{j \in m} f_i(t_j) \tag{2.4}$$

The set of constraints will not fully constrain that values of the joint distribution, with $c \ll n$ typically. We will discuss the method for choosing the constraints later, but for now we will assume that each constraint corresponds to a marginal that can be accurately computed from the training set.

When computing our estimate of P^* we'd like to enforce the constraints while not otherwise biasing the probabilities. If we imagine a case when we have no constraints, the uniform distribution is typically regarded as the most unbiased distribution. For example, if we know nothing about a coin, it is usually assumed to be fair. Similarly, if we'd want to find the most unbiased distribution given the constraints, we should find the distribution that lies closest to the uniform distribution that also obeys the constraints.

The distance between any distribution and the uniform distribution may be measured by their relative entropy. Entropy is a measure of the average amount of information needed to describe the variables at any particular time. For example, the amount of information needed to describe the outcome of a fair die will be larger than that required for a loaded die which yields "6" half the time. Since the uniform distribution is the distribution with highest entropy, we may simplify our task to just maximizing the entropy of the constrained distribution [33, 44]. Entropy measures were first developed for measuring information [73]. They have also been applied to spectral analysis

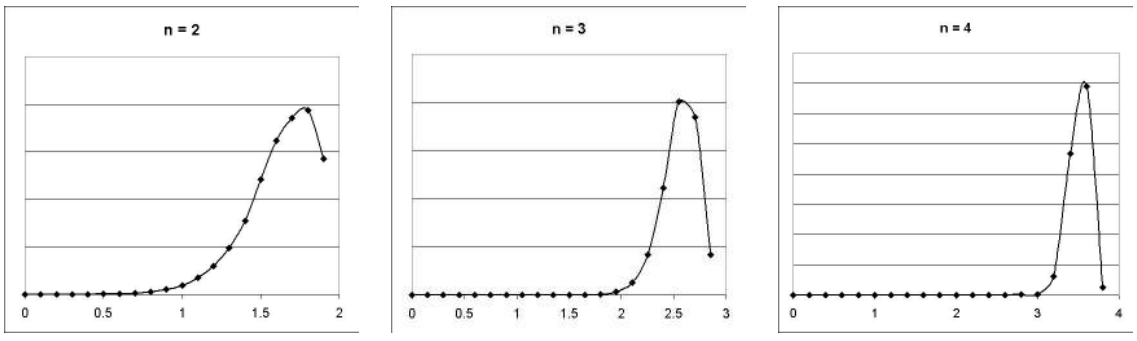
[8], reliability engineering [79], image reconstruction [28], language modeling [68] and economics [23].

Maximum entropy methods have the advantage that they choose the least committal solution to a problem given the constraints, i. e. assume independence until proven otherwise. Similarly, Jaynes [32] has said

> Maximum entropy agrees with everything that is known, but carefully avoids anything that is unknown.

Is the least committal solution the best solution? This point has been argued over the last 40 years with no clear consensus being reached [26, 35, 53, 54, 55, 56, 72, 81, 82]. Paris and Venkovska [57] argued for maximum entropy by showing that the approach is the only one that obeys a set of "common sense" rules for probabilities. An additional property of maximum entropy methods is they will assume two variables are independent until proven otherwise by the set of constraints.

Another argument in favor of maximum entropy is as follows [35]: If we consider all possible probability distributions given the constraints, they will typically be clustered around the distribution with maximum entropy. Figure 2.1, illustrates this point. We randomly sampled joint distributions with 2, 3 and 4 entries. We then created a histogram of the distribution's entropies. Clearly, the number of distributions with high entropy greatly outnumbers the distributions with low entropy, with the graphs becoming more skewed towards higher entropies as the number of variables increase. Therefore, if we consider all distributions equally likely, the true distribution will most likely have a high entropy.



(a) (b) (c)

Figure 2.1: Histogram of entropies for uniformly sampled distributions with 2, 3 and 4 entries.

2.1.1 Shannon's Entropy

One of the first measures of entropy and still most popular is that of Shannon [73]. Shannon's entropy measures the amount of information in a communication stream. Later research has applied his measure to a wide range of applications including but not limited to spectral analysis [8], language modeling [1, 62, 68] and economics [23]. Shannon constructed his measure H so that it satisfied the following properties for all p_i within the estimated joint probability distribution P_1:

1. H is a continuous positive function.
2. If all p_i are equal, $p_i = \frac{1}{n}$, then H should be a monotonic increasing function of n.
3. For all $n \geq 2$, $H(p_1, \ldots, p_n) = H(p_1 + p_2, p_3, \ldots, p_n) + (p_1 + p_2)H(\frac{p_1}{p_1+p_2}, \frac{p_2}{p_1+p_2})$.

Shannon showed the only function that satisfied these properties is:

$$H(P) = -\sum_i p_i \log(p_i) \tag{2.5}$$

Using Lagrangian multipliers the joint probability distribution P_1 that satisfies our constraints while maximizing the entropy function H has the following form:

$$P_1(x) = \prod_i \mu_i^{f_i(x)} \tag{2.6}$$

The set of unknown parameters μ_i can be computed using the "Generalized Iterative Scaling" algorithm (GIS) [14] or by using some variant of gradient descent. A short outline is as follows:

1. Initialize $\mu_j = 1$.
2. At iteration k, compute
$$P_1^k(f_i) = \sum_x f_i(x) \prod_j \mu_j^{f_j(x)} \tag{2.7}$$
3. Update the parameters μ_j by GIS or gradient descent using $\bar{P}(f_i)$ as the target value and $P_1^k(f_i)$ as the predicted value.
4. Repeat steps 2 and 3 until convergence.

The GIS or gradient descent algorithm is guaranteed to converge since the error space is convex [14]. Unfortunately, step 2 of the algorithm can be quite computationally expensive. Theoretically,

step 2 requires $O(n)$ summations, which is exponential in a. However, in practice only values that show up in the training set are generally used [14]. Even with this simplification learning in large domains can be prohibitively expensive.

2.1.2 Rényi's Entropy

About ten years after Shannon introduced his measure of entropy a mathematician from Hungary named Rényi [64, 65, 66] generalized his work. Rényi relaxed Shannon's third property for measures of entropy H_α as follows:

$$\text{For two independent distributions } P \text{ and } \acute{P} : H_\alpha(P\acute{P}) = H_\alpha(P) + H_\alpha(\acute{P}) \tag{2.8}$$

Rényi found that the following family of functions satisfies Shannon's first two properties and his generalization of the third (Rényi referred to his family of information measures as $I_\alpha = H_\alpha$):

$$H_\alpha(P) = \frac{1}{1-\alpha} \log\left(\sum_i p_i^\alpha\right) \text{ for } \alpha > 0 \tag{2.9}$$

As α approaches one, equation 2.9 reverts back to Shannon's equation 2.5, that is:

$$\lim_{\alpha \to 1} H_\alpha(P) = H(P) \tag{2.10}$$

Of particular interest to us is the case when α is equal to two. This measure has been called Rényi's Quadratic Entropy (RQE):

$$H_2 = -\log\left(\sum_i p_i^2\right) \tag{2.11}$$

Since we are only concerned with maximizing the entropy we can drop the log from the equation which results in:

$$-\sum_i p_i^2 \tag{2.12}$$

with

$$0 \leq p_i \leq 1 \text{ for all } p_i \tag{2.13}$$

Therefore, to maximize RQE we need to minimize the sum of the squared probabilities. In the past Jaynes [33] has stated the following concerning 2.12:

> In particular, the quantity $-\sum p_i^2$ has many of the qualitative properties of Shannon's

> information measure, and in many cases leads to substantially the same results. However, it is much more difficult to apply in practice. Conditional maxima of $-\sum p_i^2$ cannot be found by a stationary property involving Lagrangian multipliers, because the distribution which makes the quantity stationary subject to prescribed averages does not in general satisfy the condition $p_i \geq 0$.

It is true that if we use Lagrangian multipliers as we did for Shannon's measure, some probability estimates will be found to be less than zero. For this reason and others RQE has never achieved the popularity of Shannon's. Algorithms for minimizing functions given inequality constraints such as Zangwill's penalty function [85] or Fiacco and McCormick's barrier function [21] exist. However, these algorithms can be computationally expensive.

Despite this problem, Rényi's quadratic entropy has been successfully used in time-delay neural networks [19] and multi-layer perceptrons [20]. In econometrics, the use of Rényi's generalized family of entropies has been proposed for use in solving sets of linear equations [23, 24]

We will denote the probability distribution that maximizes the H_α measure of entropy as P_α. Thus P_1 is the probability distribution that maximizes Shannon's entropy given the constraints and P_2 is the distribution that maximizes RQE.

2.1.3 Unbounded Rényi Quadratic Entropy

If we ignore the bounding constraints for RQE we can use Lagrangian Multipliers to find a set of values Y that maximizes $\sum_x y_x^2$ given the constraints $\sum_x y_x f_i(x) = \sum_x \bar{p}(x) f_i(x) = \bar{P}(f_i)$. Since it is possible for $y_x < 0$ or $y_x > 1$ we cannot call the resultant Y a probability distribution. Thus the values y_x should be viewed as an approximation of a probability distribution and not as a probability distribution themselves. For a particular problem, if $0 \leq y_x \leq 1$ for all y_x then $y_x = P_2(x)$ for all x.

To solve for the Lagrangian Multipliers we need to maximize:

$$\Lambda(y, \kappa) = \sum_x y_x^2 - \sum_i \kappa_i \left(\sum_x y_x f_i(x) - \sum_x \bar{p}(x) f_i(x) \right) \tag{2.14}$$

Setting $\nabla \Lambda(y, \kappa) = 0$ we find:

$$0 = \frac{\partial \Lambda(y, \kappa)}{\partial y} = 2y_x - \sum_i \kappa_i f_i(x) \tag{2.15}$$

Thus, if $\lambda_i = \frac{\kappa_i}{2}$:

$$y_x = \sum_i \lambda_i f_i(x) \tag{2.16}$$

for some set of λs.

Using our constraints we can solve for the λs using the following set of equations:

$$\sum_x \sum_j \lambda_j f_j(x) f_i(x) = \sum_x \bar{p}(x) f_i(x) = \bar{P}(f_i) \tag{2.17}$$

Rearranging the summations we find:

$$\sum_j \lambda_j \sum_x f_j(x) f_i(x) = \bar{P}(f_i) \tag{2.18}$$

An alternative approach to achieve the same results is to use Least Squares (LS). We know from 2.16 that the maximum of $\sum_x y_x^2$ is a linear combination of the feature functions. Thus we can use LS to find the hyper-plane that minimizes the error between the hyper-plane and the training data:

$$\chi^2 = \sum_x \left(\bar{P}(x) - \sum_j \lambda_j f_j(x) \right)^2 \tag{2.19}$$

The minimum occurs when the derivative of χ^2 is zero. That is:

$$0 = \sum_x \left(\bar{P}(x) - \sum_j \lambda_j f_j(x) \right) f_i(x) \tag{2.20}$$

Rearranging the order of the summations we find:

$$0 = \sum_x \bar{P}(x) f_i(x) - \sum_j \lambda_j \sum_x f_j(x) f_i(x) \tag{2.21}$$

Since $\sum_x \bar{P}(x) f_i(x) = \bar{P}(f_i)$

$$0 = \bar{P}(f_i) - \sum_j \lambda_j \sum_x f_j(x) f_i(x) \tag{2.22}$$

Which is equivalent to 2.18.

Therefore we may view minimizing 2.12 without the bounds as doing least squares with the feature functions as axes and the parameters λ as regression coefficients. For the remaining of the paper we will refer to RQE as maximizing 2.12 with bounds and Unbounded Rényi Quadratic Entropy

(URQE) as maximizing 2.12 without bounds. The values they produce will form the sets P_2 and Y respectively. In previous work, Csiszár [13] has examined the relationship of least squares and maximum entropy in developing a rational for using the methods. Grünwald and Dawid [26, 27] have also examined minimizing the squared probabilities when considering alternative loss functions other than log loss when applied to game theory.

One advantage of using URQE is that it is computationally efficient. We can construct a matrix of pairwise frequencies $\sum_x f_i f_j$ for all $i, j \in c$ where c is the number of constraints:

$$\mathbf{F} = \begin{bmatrix} \sum f_0 & \sum f_1 f_0 & \cdots & \sum f_c f_0 \\ \vdots & \vdots & \ddots & \vdots \\ \sum f_0 f_c & \sum f_1 f_c & \cdots & \sum f_c \end{bmatrix} \tag{2.23}$$

Then we can directly compute the parameters from the following equation:

$$\mathbf{F} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_c \end{bmatrix} = \begin{bmatrix} \bar{P}(f_0) \\ \bar{P}(f_1) \\ \vdots \\ \bar{P}(f_c) \end{bmatrix} \tag{2.24}$$

Since inverting a matrix takes only $O(c^3)$ time, the amount of computation required to find the values Y can be greatly reduced over finding the distribution using Shannon's entropy.

2.1.4 Comparison of Shannon's and Rényi's Entropy Measures

As stated by Jaynes [33] both measures of entropy will result in similar predictions for many cases. To explore this statement let us consider the simple case when our joint distribution consists of three points. An example problem of this type would be determining if a person is currently in Mexico, U.S. or Canada. The person can only be in one country at a time so we know $P(Mexico) + P(U.S.) + P(Canada) = 1.0$. A common method for illustrating this type of distribution is to use a *simplex*, figure 2.2. A simplex is a triangle that represents the set of probability distributions. The three points of the triangle correspond to each outcome having a probability of one. If we impose a linear constraint on the probability distribution, such as C_1 that enforces $P(Canada) = 0.5$ in figure 2.2, the set of possible probability distributions will be limited to a line. The goal of maximum entropy methods is to find the point that satisfies the constraints, i. e. lies along the line, that has the greatest entropy.

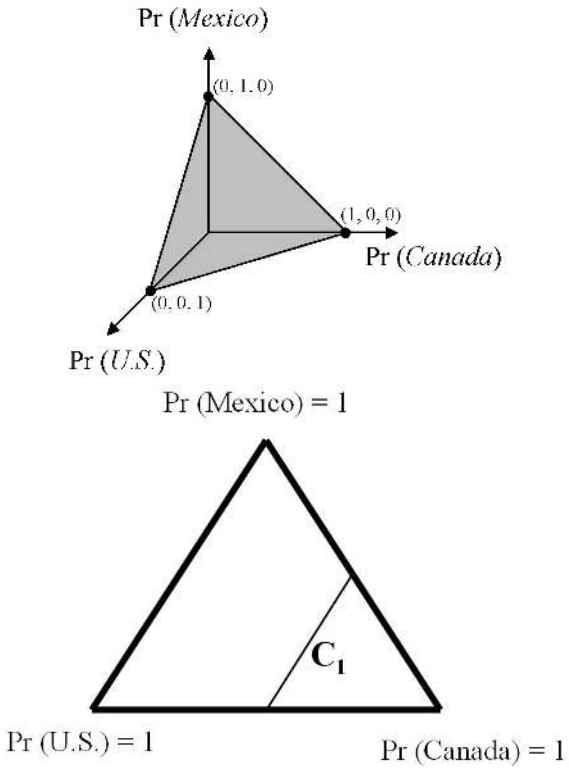


Figure 2.2: The set of probabilities is represented by a triangle with each corner corresponding to an outcome having a probability of one. If we impose a linear constraint such as C_1 the set of possible probability distributions is restricted to a line.

To illustrate how the entropy varies within the simplex we have created two topographic plots. The first shows Shannon's entropy, figure 2.3. The entropy is greatest at the center of the simplex and decreases as it approaches the edges. The second is a plot of RQE, figure 2.4. It is similar to that of Shannon's except around the edges or points of the simplex. With both measures, the entropy decreases in a mostly circular pattern near the center of the simplex. Towards the edges or points of the simplex the entropy measures differ. While Rényi's measure continues a circular pattern towards the edges, Shannon's begins to warp into a triangular shape. From these figures we might suspect that the entropy measures would produce similar results for problems in which the true underlying distribution lies close to the center of the simplex.

To test this hypothesis we tested both measures along with URQE on a set of pseudo-randomly selected joint distributions with five and ten binary valued variables. The marginals of all single and pair-wise probabilities were enforced using 15 constraints for the five variable case and 55 constraints for the ten variable case. The joint distributions had 32 and 1024 entries respectively.

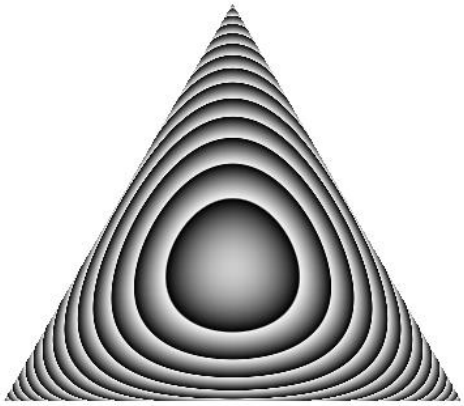


Figure 2.3: Shannon's entropy over the simplex.

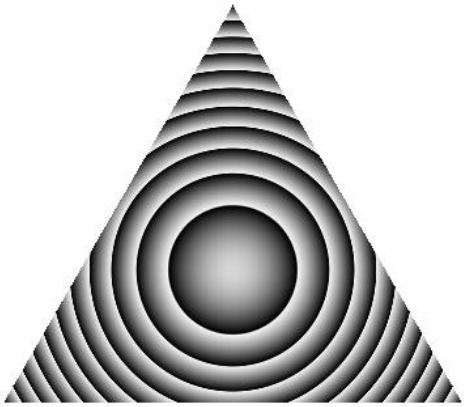
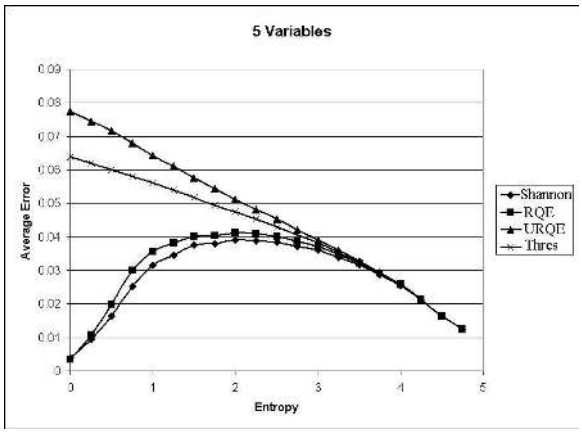


Figure 2.4: Rényi's quadratic entropy over the simplex.

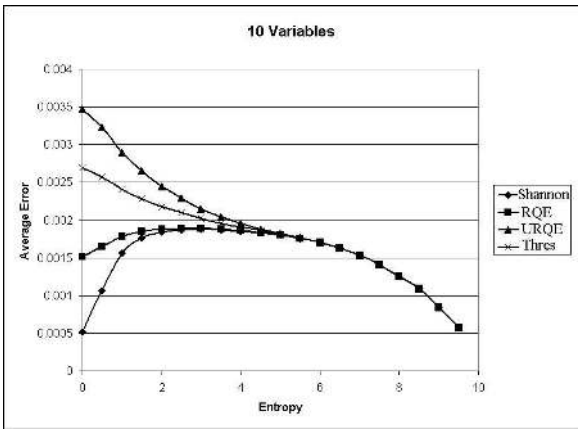
Errors were computed by finding the absolute difference between the true distribution P^* and that found by each method P_{found}:

$$error(P_{found}) = \frac{1}{n} \sum_{x} |P^*(x) - P_{found}(x)| \tag{2.25}$$

where $P_{found}(x)$ is equal to $P_1(x)$, $P_2(x)$ and y_x. Finally, we've plotted the error results against Shannon's entropy of the true distribution. The tests where run several thousand times for each data point.



(a)



(b)

Figure 2.5: Comparison of Shannon's entropy, RQE and URQE using 5(a) and 10(b) variables.

As shown in figure 2.5, Shannon's measure consistently out performs RQE for smaller entropies. As the true distribution's entropy increases, the results using RQE improve until there is no measurable difference between RQE and Shannon. This is in agreement with our expectations from studying figure 2.3 and 2.4. The greater the true entropy the more likely the two methods are to agree. URQE does much worse since the bounding constraints are not enforced. For comparison we've also included the URQE results with the values thresholded, labeled *Thres* in figure 2.5. Any value greater than one was set to one and any value less than zero was set to zero. After thresholding, the values of the *Thres* example do not sum to one. While this improved the results over URQE the errors were still not as good as RQE or Shannon's measure.

Figure 2.6 illustrates why the errors differ. The true distribution has low entropy and the constraint C_1 only allows distributions with low entropy. Shannon's measure and RQE both produce distributions in the tip of the simplex. Typically, Shannon's measure P_1 will find distributions closer to the center than RQE, P_2, for problems of this type. URQE does much worse since it finds values Y that don't represent a true probability distribution, i. e. its values lie outside the simplex. In this case, it would assign a negative value to y_{Canada}. The values from *Thres* don't lie on the same plane as the simplex since the values don't sum to one.

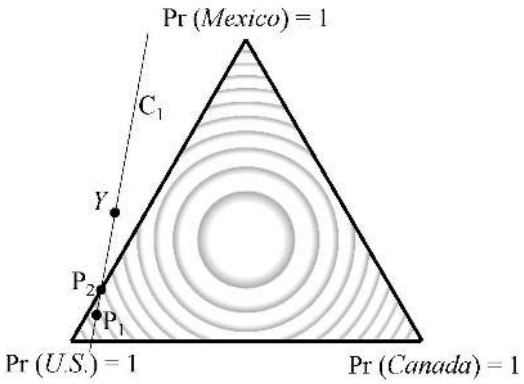


Figure 2.6: A constraint C_1 that has a minimum Y that lies outside the simplex.

As the entropy of the true joint distribution increases, the likelihood of the values Y forming a real distribution increases. Figure 2.7 displays the average percentage of values within Y that are negative as a function of entropy. If all values within Y range between 0 and 1 then $Y = P_2$. For reasons of round-off error we tested the values at $y_x < -0.001$, instead of $y_x < 0$.

As the number of variables increase, the results using URQE worsen. For any problem with a large number of variables, the use of URQE in predicting the joint distribution is highly inaccurate for low entropy distributions since the constraints $0 \leq y_x \leq 1$ aren't enforced. Unfortunately

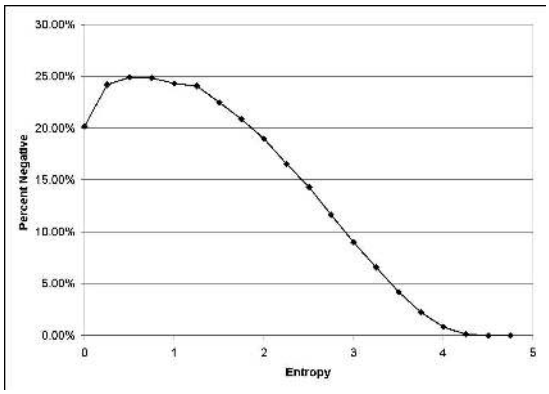


Figure 2.7: Average percentage of values within Y that are negative for the 5 variable example.

the enforcement of $0 \leq y_x \leq 1$ removes the computational advantages of using URQE. While using URQE may result in poor predictions for joint distributions we will show in the following section that URQE can be a good alternative to Shannon's entropy when predicting conditional probabilities.

2.2 Maximizing Conditional Entropy

In many real world problems we would like to compute conditional probabilities, that is the probability of x_H given x_E. A simple method for computing these probabilities is to compute the entire joint distribution as above and compute $P(x_H|x_E) = \frac{P(x_H,x_E)}{P(x_E)}$ [48]. Unfortunately, computing the joint distribution using Shannon's entropy or RQE is too computationally expensive and using URQE is too inaccurate. A vastly more efficient method for computing conditional probabilities is to maximize conditional entropy rather than the joint distribution's entropy. From the perspective of measuring information, the conditional entropy measures the amount of information needed to express X_H given that we know X_E.

2.2.1 Shannon's Conditional Entropy

The conditional form of Shannon's entropy is:

$$H(X_H|X_E) = -\sum_{x_H,x_E} P(x_E)P(x_H|x_E)\log(P(x_H|x_E)) \tag{2.26}$$

This is the expectation of Shannon's entropy conditioned upon X_E. For reasons of computational efficiency the following approximation of conditional entropy is typically used [68]:

$$H(X_H|X_E) \approx -\sum_{x_H,x_E} \bar{P}(x_E)P(x_H|x_E)\log(P(x_H|x_E)) \tag{2.27}$$

That is, only cases x_E that appear in the training set are used, for all other cases it is assumed $P(x_E) = 0$. This provides a large computational savings over maximizing the entropy over the entire joint distribution, but the maximization algorithm still has to sum over the entire training set during each iteration.

It is interesting to note the following:

$$H(X_H, X_E) = H(X_E) + H(X_H|X_E) \tag{2.28}$$

Proof:

$$\begin{aligned}
&H(X_H, X_E) = \\
&-\sum_{x_H,x_E} P(x_E)P(x_H|x_E)\log(P(x_E)P(x_H|x_E)) = \\
&-\sum_{x_H,x_E} P(x_E)P(x_H|x_E)\log(P(x_E)) - \sum_{x_H,x_E} P(x_E)P(x_H|x_E)\log(P(x_H|x_E)) =
\end{aligned}$$

Since $\sum_{X_H} P(x_H|x_E) = 1$:

$$\begin{aligned}
&-\sum_{X_E} P(x_E)\log(P(x_E)) - \sum_{X_H,X_E} P(x_E)P(x_H|x_E)\log(P(x_H|x_E)) = \\
&H(X_E) + H(X_H|X_E)
\end{aligned}$$

■

Resulting from equation 2.28, the conditional probabilities found by maximizing the conditional entropy or the joint's distributions entropy, with the proper constraints placed on X_E, will be equivalent.

To compute the distribution with maximal conditional entropy the same algorithm is used as above except $P_1^k(f_i)$ is computed by:

$$P_1^k(f_i) = \sum_{x_H,x_E} \bar{P}(x_E)P_1^k(x_H|x_E)f_i(x_H, x_E) \tag{2.29}$$

with

$$P_1^k(x_H|x_E) = \frac{1}{Z(x_E)} \prod_j \mu_j^{f_j(x_H,x_E)} \tag{2.30}$$

where

$$Z(x_E) = \sum_{x_H} \prod_j \mu_j^{f_j(x_H,x_E)} \tag{2.31}$$

to enforce $\sum_{x_H} P(x_H|x_E) = 1$.

2.2.2 Rényi's Conditional Entropy

The conditional form of RQE is:

$$H_2(X_H|X_E) = -\sum_{X_E} P(x_E) \log\left(\sum_{X_H} P(x_H|x_E)^2\right) \tag{2.32}$$

Since Rényi relaxed Shannon's third property for measures of entropy, $H_2(X_H, X_E)$ isn't equal to $H_2(X_E) + H_2(X_H|X_E)$. For this reason the conditional probabilities obtained by maximizing 2.32 will not in general be equal to the probabilities found by maximizing RQE over the entire joint distribution. As Shannon first stated, any measure of entropy other than H will be inconsistent in this manner.

As we did for Shannon's conditional entropy, the following approximation can be made:

$$H_2(X_H|X_E) \approx -\sum_{X_E} \bar{P}(x_E) \log\left(\sum_{X_H} P(x_H|x_E)^2\right) \tag{2.33}$$

An alternative to using the conditional form of Rényi's quadratic entropy is to first simplify equation 2.11 by dropping the log; resulting in equation 2.12. We can then find the expectation of 2.12 given the evidence variables X_E:

$$-\sum_{X_H,X_E} \bar{P}(x_E) P(x_H|x_E)^2 \tag{2.34}$$

with

$$0 \leq P(x_H|x_E) \leq 1 \tag{2.35}$$

Equation 2.34 is similar to equation 2.12 with conditional probabilities except it is weighted by $\bar{P}(x_E)$. As we will demonstrate the weighting of $\bar{P}(x_E)$ has a large effect on the errors found when using URQE.

Maximizing equation 2.33 will not result in exactly the same distribution as maximizing 2.34. However, the distributions will be nearly identical since 2.34 acts as a upper bound on 2.33 as shown in figure 2.8. Due to the ease of maximizing equation 2.34, we will use it instead of equation 2.33.

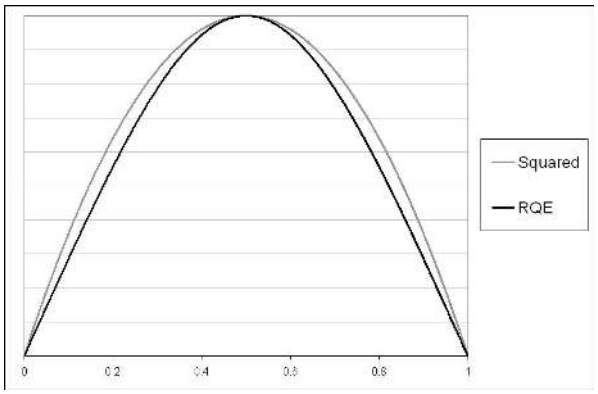


Figure 2.8: Plot of RQE and scaled squared probabilities for the case of one variable.

2.2.3 Independent Hidden Variables

We have already demonstrated through a couple simulations that using URQE to compute the joint distribution produces worse results than using Shannon's entropy or RQE when the true disribution has low entropy. Computing conditional probabilities is essentially the same problem except we're computing the joint distribution of X_H conditioned upon X_E. Once again the errors produced using URQE to approximate $P(X_H|X_E)$ will be larger than those produced by using either RQE or Shannon's entropy.

For many applications the desired result isn't to compute $P(X_H|X_E)$, but to compute $P(X_i|X_E)$ for all $X_i \in X_H$. For example, when computing which books a user will purchase given their previous buying history we aren't concerned with the probability of whether they will buy both book A and book B at the same time. We are only concerned with whether they will buy book A or buy book B. That is, we may consider the probability of buying book A independent of buying book B. Many applications fall under this category, such as collaborative filtering, image retrieval and language modeling. If the probability of some combination of variables is desired we can always create a "new" feature function that corresponds to this combination.

Previously, to compute the value $P_1(X_i|X_E)$ using Shannon's entropy we would need to compute:

$$P_1(X_i = 1|x_E) = \frac{1}{Z(x_E)} \sum_{x_H} x_i \prod_j \mu_j^{f_j(x_H, x_E)} \tag{2.36}$$

Since we need to sum over all x_H computing this value can be computationally expensive. When we are only concerned with $P_1(X_i|X_E)$, we can assume that the hidden variables are independent given X_E. Thus we can separately compute the value of $P_1(X_i|X_E)$ for all $X_i \in X_H$.

For computing conditional probabilities, the constraints will be based on pairs of feature functions instead of individual feature functions. Previously we wished to constrain the probability of each feature function f_i as $P(f_i) = \bar{P}(f_i)$. Now, we will constrain the probability of each feature function conditioned upon the evidence feature functions, i. e.:

$$P(X_i = 1|f_j) = \bar{P}(X_i = 1|f_j) \tag{2.37}$$

for all $X_i \in X_H$ and $f_j \in F_E$. A feature function f_i is an evidence or "known" function F_E if its value can be computed directly from X_E. All other feature functions will form the set of hidden functions $F_H = F - F_E$.

For each hidden variable $X_i \in X_H$ and evidence feature function f_j we compute a parameter $\mu_{i,j}$. Computing the conditional probabilities $P_1(X_i = 1|x_E)$ can then be done efficiently by:

$$P_1(X_i = 1|x_E) = \prod_j \mu_{i,j}^{f_j(x_E)} \tag{2.38}$$

Once again we can compute approximations of $P_2(X_i = 1|x_E)$ using URQE. We will denote the URQE approximation of $P_2(X_i = 1|x_E)$ as y_{i,x_E}. When computing y_{i,x_E} we need to minimize:

$$\sum_{x_i,x_E} \bar{P}(x_E) y_{i,x_E}^2 \tag{2.39}$$

Using Lagrangian Multipliers we find the set of values that minimizes the above equation and satisfies the constraints has the following form:

$$y_{i,x_E} = \sum_k \lambda_{i,k} f_k(x_E) \tag{2.40}$$

Our constraints state:

$$P(X_i = 1|f_j) = \frac{\sum_{x_E} \bar{P}(x_E) y_{i,x_E} f_j}{\bar{P}(f_j)} = \bar{P}(X_i = 1|f_j) \tag{2.41}$$

After rearranging and substituting in 2.40, we can find the λs by using the following set of equations:

$$\bar{P}(X_i = 1, f_j) = \sum_{x_E} \bar{P}(x_E) f_j(x_E) \sum_k \lambda_{i,k} f_k(x_E) \text{ for all } j \tag{2.42}$$

Rearranging the summations we find:

$$\bar{P}(X_i = 1, f_j) = \sum_k \lambda_{i,k} \sum_{x_E} \bar{P}(x_E) f_k(x_E) f_j(x_E) \text{ for all } j \tag{2.43}$$

By replacing $\bar{P}(f_j)$ for $\sum_{x_E} \bar{P}(x_E) f_j(x_E)$ we may compute the λs by our final equation:

$$\bar{P}(X_i = 1, f_j) = \sum_k \lambda_{i,k} \bar{P}(f_j, f_k) \text{ for all } j \tag{2.44}$$

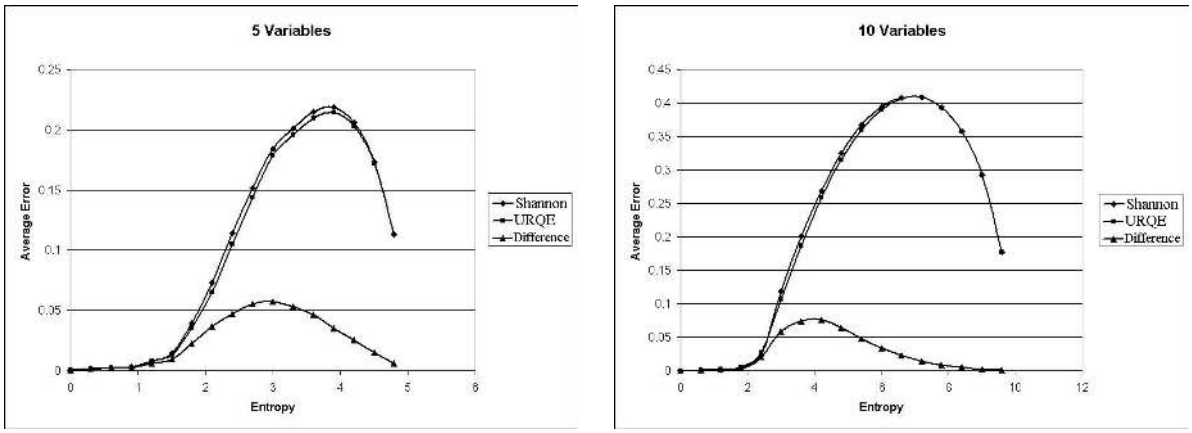
2.2.4 Comparison of Methods for Approximating Conditional Probabilities

Similar to our experiments for joint distributions, we've compared URQE to Shannon's entropy using a set of pseudo-random joint distributions with 5 and 10 binary valued variables. In each case we attempted to compute the conditional probability of the last variable X_i given the others X_E by maximizing the conditional entropy. The error values were computed from the following:

$$error(P_{found}) = \sum_x P^*(x_E) \left| P^*(X_i = 1 | x_E) - P_{found}(X_i = 1 | x_E) \right| \tag{2.45}$$

Where $P_{found}(x_H | x_E)$ is $P_1(X_i = 1 | x_E)$ and y_{i,x_E}. As shown in figure 2.9, the error results using URQE or Shannon's entropy are nearly identical. Unlike the results when computing the joint distribution, the errors don't diverge as the true entropy decreases. Even though the error results relative to the true distribution are very similar, the values computed using the two methods vary. The error between the two methods is displayed in figure 2.9 as the "difference" curve.

When using URQE, why do we get better results computing conditional probabilities than we do for joint distributions? When computing the joint distribution we're trying to estimate 2^{E+H} values given about $(E + H)^2$ parameters, where E is the number of evidence variables and H is the number of hidden variables. In contrast, when we are computing $P(X_i | X_E)$, we are trying to estimate $H2^E$ values from $(E + H)^2$ parameters. Clearly, the number of values we're trying to estimate is much smaller. As the entropy of the true distribution decreases, the number of values for X_E with high probability values $\bar{P}(x_E)$ decreases. Thus, the number of values we need to estimate will in essence decrease. For this reason the error results improve for smaller entropies.



(a) (b)

Figure 2.9: Comparison of maximizing Shannon's entropy and URQE measures using 5(a) and 10(b) variables.

In agreement with the error results, figure 2.10 shows the percentage of values for y_{i,x_E} that are negative. The percentage of negative values is much better than that for joint distributions.

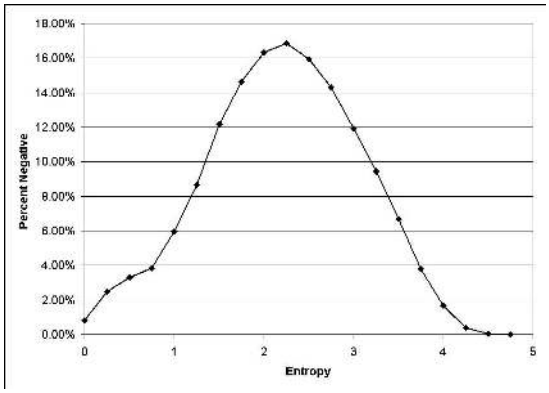


Figure 2.10: Average percentage of values within Y that are negative for the conditional 5 variable example.

2.3 The Relation of Bayesian Approaches to Maximum Entropy

Throughout the last 50 years there has been much debate between proponents of Bayesian vs. maximum entropy approaches [26, 35, 34, 54, 55, 56, 72]. There is good reason for this. Both approaches produce different results that cannot be duplicated using the other. We will not go into much detail

as to why this is the case, but we will give some simple examples to illustrate some differences.

Once again let us consider the simplex previously described. The Bayesian approach would assign a prior probability to each allowable joint distribution. Using the priors as weights the predicted joint distribution can be found. Let us consider the simple case when we have uniform priors. Is there an entropy measure H^* that we can maximize that would give us the same results as the Bayesian approach for any set of constraints?

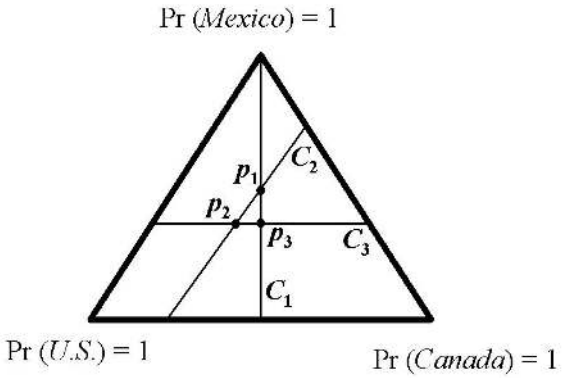


Figure 2.11: For each constraint C_i the Bayesian approach with uniform priors predicts a distribution P_i. If the same result were to be found by maximizing an entropy measure H^* then $H^*(P_1) > H^*(P_3) > H^*(P_2) > H^*(P_1)$ which is a contradiction.

If we examine figure 2.11 this is clearly not the case. The Bayesian approach predicts the joint distribution P_1 that lies at the midpoint when constraint C_1 is enforced. Thus $H^*(P_1)$ must have a higher value than any other point along C_1, i. e. $H^*(P_1) > H^*(P_3)$. Using this logic for constraints C_2 and C_3 we find that $H^*(P_1) > H^*(P_3) > H^*(P_2) > H^*(P_1)$ which is a contradiction. Therefore there exists no function H^* that would produce the same results as the Bayesian approach using uniform priors.

The converse is also true. Given a constraint C_1 and the probability distribution P_1 found by RQE, there may not exist a set of prior probabilities that produce the same results. The simplest example is that of figure 2.12. The probability distribution P_1 found by RQE lies on the edge of the simplex. Given any non-zero assignment of prior probabilities, the Bayesian approach will never find a probability distribution on the edge of the simplex.

These results shouldn't be surprising since the Bayesian and maximum entropy approaches make different assumptions and answer different questions. The Bayesian approach finds expectations while making some assumption about the prior probabilities of the distributions. The maximum entropy approach attempts to to find the distribution in which the variables are least dependent

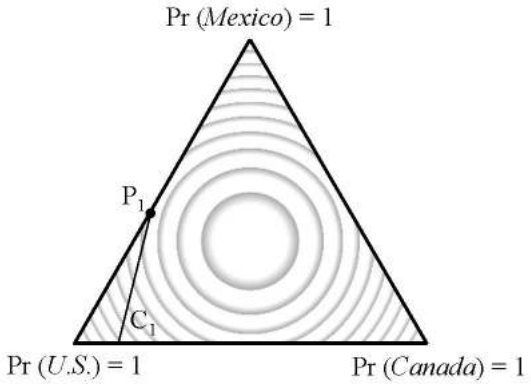


Figure 2.12: Using RQE, the probability distribution P_1 found with constraint C_1 lies on the edge of the simplex. A Bayesian approach with non-zero priors will never find a probability distribution on the edge of the simplex.

on each other, i. e. the distribution that is closest to the uniform distribution.

To illustrate these differences consider the following problem: A person lives in one of three countries, Canada, United States or Mexico. The only information we have is that the probability of the person living in Canada is equal to the probability of the person living in the United States.

A Bayesian approach, which assumes uniform priors, uses the fact that $P(\text{Canada}) = P(\text{United States})$ to reduce the set of possible distributions. The expected distribution is then computed as $\{\frac{1}{4}, \frac{1}{4}, \frac{1}{2}\}$.

Before any additional information is given, the distribution with maximum entropy is $\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$. When the new information that $P(\text{Canada}) = P(\text{United States})$ is given, we find that the same distribution $\{\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\}$ already satisfies this constraint.

For this simple example the Bayesian and maximum entropy approaches produce significantly different results. In general, if we have some information about the prior probabilities of the probability distributions the Bayesian approach can produce better results. For most real world problems however, no information about the priors in known and the question of priors becomes more philosophical. For problems of this type, we view maximum entropy as the best approach.

2.4 Naive Bayes

A problem related to finding probabilities conditioned on X_E is classification. The goal of classification is to find the variable $X_i \in X_H$ that has the highest conditional probability $P(X_i|x_E)$. A common algorithm for classification is Naive Bayes. Naive Bayes (NB) makes the simplifying assumption that all evidence variables X_E are independent given each hidden variable X_i. Thus the following is

assumed to be true:

$$P(x_E|X_i = 1) = \prod_j P(x_j|X_i = 1) \text{ for all } x_j \in x_E \tag{2.46}$$

Using Bayes rule we know the following:

$$P(X_i = 1|x_E) = \frac{P(x_E|X_i = 1)P(X_i = 1)}{P(x_E)} \tag{2.47}$$

Substituting in 2.46 we find:

$$P(X_i = 1|x_E) = \frac{\prod_j P(x_j|X_i = 1)P(X_i = 1)}{P(x_E)} \tag{2.48}$$

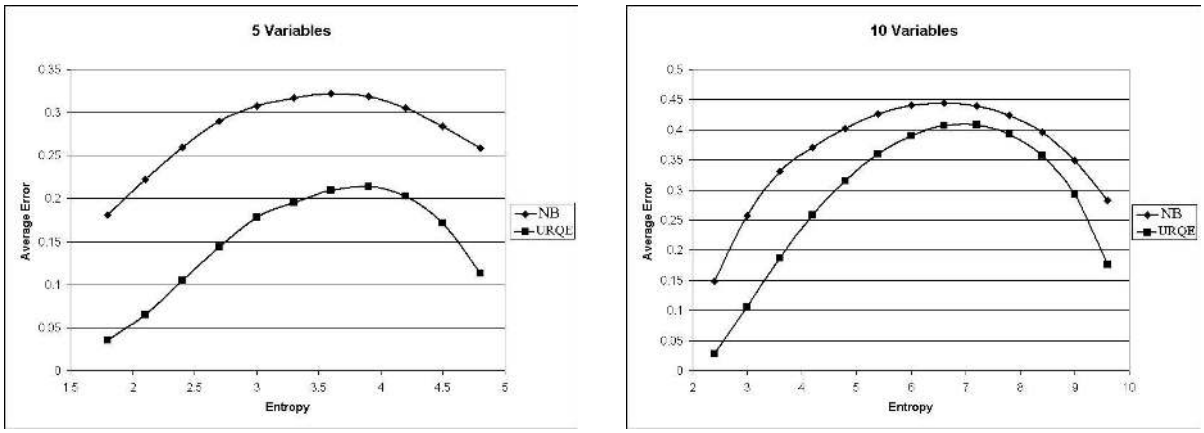
Due to the simplicity of computing 2.48, NB has attracted a lot of attention. Surprisingly, the results using NB have been shown to outperform many more sophisticated algorithms on classification tasks. Recently, it has been shown that even if the independent assumption for X_E is violated, NB classifiers can be optimal [16].

Why does NB do so well? While this question has been debated, it is understood that NB captures many of the low-order interactions between variables. In the real world data sets used to test NB, there is a significant trade off between bias and variance errors. NB classifiers are known to have low variance with high bias. Given relatively small data sets in large domains there is a lot of variance, thus algorithms such as NB will perform well. As we will show later, NB and our algorithm for URQE have the same number of real parameters. This leads us to the question of which algorithm is better?

Using the same simulated data as we used to compare URQE with Shannon's entropy we compared URQE with NB. Our first test was to measure the absolute difference in error between the true probability and that computed using URQE and NB. The results can be seen in figure 2.13. While the errors have similar shapes, URQE out performs NB across all distributions.

It is known that NB doesn't do as well for calibration tasks, i. e. problems in which the exact probability value must be computed. NB is typically used as a classification algorithm. To explore URQE vs. NB in this class of problem we have repeated our experiments except we introduce a probability threshold of 0.5. Any probability value greater than 0.5 is set to 1 while any value less than 0.5 is set to 0. We can view this as classifying the output variable into two states. The results for these tests can be seen in figure 2.14. Once again URQE out performs NB and in some cases produces almost twice as accurate results.

To understand these results let us first re-examine NB. In the classification task described above,



(a) (b)

Figure 2.13: Comparison of URQE and Naive Bayes using 5(a) and 10(b) variables.

the values for the variables are discrete, i. e. zero or one. Due to this, as shown in [17], we can devise a discriminant function $g(x_E)$ for a hidden variable X_i as:

$$g(x_E) = \log\left(\frac{P(x_E|X_i = 1)}{P(x_E|X_i = 0)}\right) + \log\left(\frac{P(X_i = 1)}{P(X_i = 0)}\right) \tag{2.49}$$

If $g(x_E) > 0$ then we assign a value of one to X_i and zero otherwise. By using the relation:

$$P(x_E|X_i = 1) = \prod_j P(x_j)^{x_j}(1 - P(x_j))^{1-x_j} \tag{2.50}$$

and rearranging, we find $g(x_E)$ has the following form:

$$g(x_E) = \sum_j w_j x_j + w_0 \tag{2.51}$$

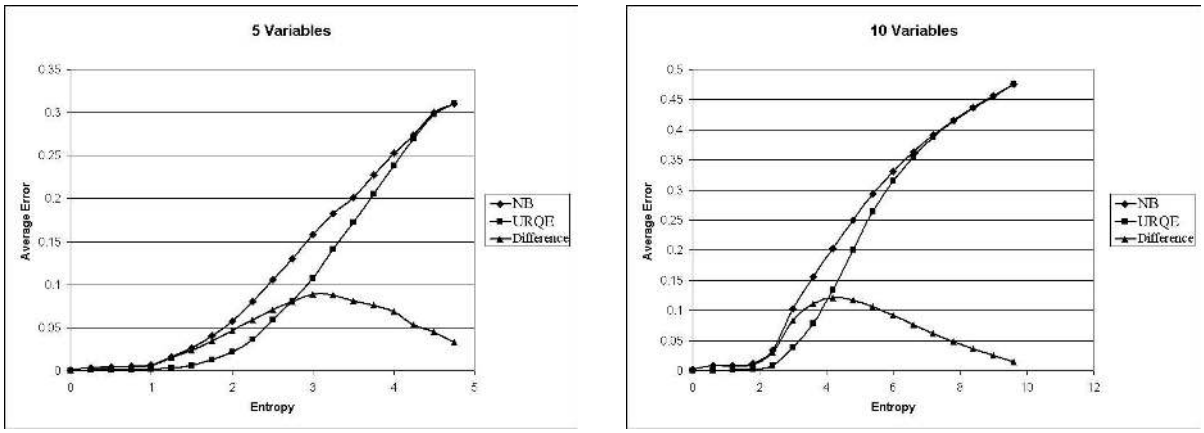
where

$$w_j = log\left(\frac{P(x_j|X_i = 1)\left(1 - P(x_j|X_i = 0)\right)}{P(x_j|X_i = 0)\left(1 - P(x_j|X_i = 1)\right)}\right) \tag{2.52}$$

and

$$w_0 = \sum_j \log\left(\frac{1 - P(x_j|X_i = 1)}{1 - P(x_j|X_i = 0)}\right) + log\left(\frac{P(X_i = 1)}{P(X_i = 0)}\right) \tag{2.53}$$

Therefore NB and URQE can each be viewed as computing a weighted linear sum of the ev-



(a) (b)

Figure 2.14: Comparison of URQE and Naive Bayes using 5(a) and 10(b) variables for classification.

idence variables using the weights w_j or λ_j respectively. Our tests on synthetic data indicate that using the λs produces better results. Later we will demonstrate both algorithms on real world data.

The simulated results shouldn't be surprising since both methods assume the output values are a weighted linear sum of the input values. This is the same as trying to fit a hyper-plane to a set of points corresponding to the training data set. It is well known that least squares minimizes the squared error between the hyper-plane and the data points. Thus, NB produces sub-optimal results with respect to squared error.

2.5 Alternative Explanations for RQE

Rényi's quadratic entropy has a close relationship to mean squared error [13], Brier's score [7] and Bregman's distance [6] for probability distributions. Entropy can be viewed as the distance between the proposed distribution $P(x)$ and the uniform distribution $U(x)$. Using mean squared error we could minimize:

$$\sum_x (P(x) - U(x))^2 \tag{2.54}$$

Since we know $\sum P(x) = 1$ this is the same as maximizing equation 2.12.

Similarly for two distributions P and U the Brier scoring rule [7, 27] is:

$$\sum_x (P(x) - U(x))^2 \tag{2.55}$$

The Bregman distance between two distributions P and U is:

$$B_g(P, U) = \sum_x g(P(x)) - g(U(x)) - g'(U(x))(P(x) - U(x)) \tag{2.56}$$

for some function g. If $g(x) = x^2$ then we obtain the mean squared distance [45]:

$$\sum_x (P(x) - U(x))^2 \tag{2.57}$$

Related to Rényi's family of entropies is the family of entropies proposed by Tsallis [80]:

$$H_\alpha^T = c\frac{\sum_i p_i^\alpha - 1}{1 - \alpha} \tag{2.58}$$

His family of entropy measures has different additive properties than those of Shannon and Rényi. Tsallis entropies don't have Rényi's additive property of:

$$\text{For two independent distributions } P \text{ and } \acute{P} : H_\alpha, (P\acute{P}) = H_\alpha(P) + H_\alpha(\acute{P}) \tag{2.59}$$

or the property of Shannon's entropy that the composite is equal to the conditional and marginal:

$$\text{For any two sets of variables } X \text{ and } Y : H(X, Y) = H(X) + H(Y|X) \tag{2.60}$$

Chapter 3

Maximizing the Unbounded Rényi Quadratic Entropy: Two Methods

Our goal is to compute the conditional probabilities $P(X_i|X_E)$ for all $X_i \in X_H$. That is, we want to compute the probability of each individual variable given the set of known variables. From the previous chapter we learned that maximizing the Unbounded Rényi Quadratic Entropy (URQE) produces similar error results as those produced using Shannon's entropy for problems of this type and outperforms those of naive Bayes. The values found using URQE take the following form:

$$y_{i,x_E} = \sum_j \lambda_{i,j} f_j(x_E) \tag{3.1}$$

Where f_j are feature functions with domains over X_E, i. e. they can be directly computed from the variables within X_E. The $\lambda_{i,j}$s can be computed using the follow set of equations for all i and j:

$$\bar{P}(X_i = 1, f_j) = \sum_k \lambda_{i,k} \bar{P}(f_j, f_k) \text{ for all } j \tag{3.2}$$

While it is possible to compute the conditional probabilities using the above equations, there are computationally more efficient methods depending on the characteristics of the problem being solved.

The main problem with solving for the conditional probabilities using the above method arises when the role of the variables within X_E varies. For many applications the variables may either be inputs X_E or outputs X_H depending on the current problem. For example, if we are trying to predict whether our current user will purchase a book A given they've purchased book B and C, the variable

corresponding to book A is an output while the variables corresponding to books B and C are the inputs. Our next user might have purchased book A and not book B, thus switching the roles of inputs and outputs for variables corresponding to book A and B. For the above example we would need to recompute the λ's for each problem whenever X_E changes. As we will show this is not always necessary. We offer two alternative but equivalent algorithms.

The first method we will discuss, called the Inverse Probability Method (IPM), will involve pre-computing the entire set of pair-wise probabilities for all the feature functions. The conditional probability estimates can then be computed from a weighted linear sum of these pair-wise probabilities. This method is typically most efficient when the size of X_E is small.

The second method we will describe is the Recurrent Linear Network (RLN). The RLN is a network of feature values that are iteratively updated using a weighted linear sum of the other features. After convergence, the values of the features correspond to the values Y found using URQE. This method is useful when the size of X_E is larger or when many features are conditionally independent.

Before we describe the two methods let us define the type and role of the feature functions. The feature function are split into three types: unit function, simple functions and complex functions.

$$F = \begin{bmatrix} f_0(x) = 1 & \\ f_i(x) = x_i & \forall i \in a \\ f_i(x) = B(x) & \forall i > a \end{bmatrix} \tag{3.3}$$

The unit function f_0 always has a value of one and acts as a bias. The simple functions, f_1 through f_a, are equivalence functions of a single variable. Finally, the complex functions comprise any binary function $B(x)$ with multiple variables as inputs, such as $x_1 \wedge x_4$ or $x_3 \vee x_6 \vee x_8$. The role and creation of the complex functions will be discussed in more detail in Chapter 4. For the examples within this chapter, we'll only use unit and simple functions.

Combining our three types of functions we form the set $F = f_0, f_1, \ldots, f_{b-1}$ of all feature functions over the entire set X. A feature function f_i is an evidence or "known" function F_E if its value can be computed directly from X_E. All other feature functions will form the set of hidden functions $F_H = F - F_E$. The number of evidence functions F_E will be denoted e and the number of hidden functions F_H as $h = b - e$. If $X_i \in X_E$ then $f_i \in F_E$. Since f_0 has a constant value of one, f_0 is always a member of F_E. We will denote the list of evidence functions as $f_{E,i}$ for all $i \in e$ and the list of hidden functions as $f_{H,j}$ for all $j \in h$. The same notation applies to $y_{E,i}$ and $y_{H,j}$.

The constraints that we place on our system will consist of conditional probabilities between

the feature functions:

$$\frac{\sum_{x_E} \bar{P}(x_E) y_{i,x_E} f_j(x_E)}{\bar{P}(f_j)} = \bar{P}(f_i|f_j) \text{ for all } i, j \tag{3.4}$$

3.1 Inverse Probability Method

The first method, called the Inverse Probability Method (IPM), for maximizing URQE involves initially computing a matrix. The pairwise conditional probability matrix $\mathbf{P}$, where $\mathbf{P}_{i,j} = \bar{P}(f_i|f_j)$. From the entries in $\mathbf{P}$ we can directly compute the weights $\lambda_{i,j}$ and the values y_i. We will abbreviate y_{i,x_E} as y_i.

Let $\mathbf{\Lambda}$ denote the $h \times e$ matrix of weights $\lambda_{i,j}$ with e equal to the number of evidence functions and h equal to the number of hidden functions:

$$\mathbf{\Lambda} = \begin{bmatrix} \lambda_{0,0} & \lambda_{0,1} & \cdots & \lambda_{0,e} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{h,0} & \lambda_{h,1} & \cdots & \lambda_{h,e} \end{bmatrix} \tag{3.5}$$

If $\mathbf{f}_E$ is a vector of values from the evidence functions:

$$\mathbf{f}_E = \begin{bmatrix} f_{E,0}(x_E) \\ f_{E,1}(x_E) \\ \vdots \\ f_{E,e}(x_E) \end{bmatrix} \tag{3.6}$$

and $\mathbf{f}_H$ is the vector of predicted values y_i for the hidden functions:

$$\mathbf{f}_H = \begin{bmatrix} y_{H,0} \\ y_{H,1} \\ \vdots \\ y_{H,h} \end{bmatrix} \tag{3.7}$$

then:

$$\mathbf{\Lambda} \mathbf{f}_E = \mathbf{f}_H \tag{3.8}$$

We learned earlier that the weights $\lambda_{i,j}$ can be computed directly from the set of equations 2.44. If $\mathbf{P}_E$ consists of all pairwise conditional probabilities between evidence functions and $\mathbf{P}_{H,E}$ consists

of all pairwise conditional probabilities between hidden and evidence functions:

$$\mathbf{P}_E = \begin{bmatrix} 1 & P(f_{E,0}|f_{E,1}) & \cdots & P(f_{E,0}|f_{E,e}) \\ \vdots & \vdots & \ddots & \vdots \\ P(f_{E,e}|f_{E,0}) & P(f_{E,e}|f_{E,1}) & \cdots & 1 \end{bmatrix} \tag{3.9}$$

$$\mathbf{P}_{H,E} = \begin{bmatrix} P(f_{H,0}|f_{E,0}) & P(f_{H,0}|f_{E,1}) & \cdots & P(f_{H,0}|f_{E,e}) \\ \vdots & \vdots & \ddots & \vdots \\ P(f_{H,h}|f_{E,0}) & P(f_{H,h}|f_{E,1}) & \cdots & P(f_{H,h}|f_{E,e}) \end{bmatrix} \tag{3.10}$$

then

$$\mathbf{P}_{H,E}\mathbf{P}_E^{-1} = \mathbf{\Lambda} \tag{3.11}$$

Combining 3.8 and 3.11 we find the values for the hidden functions $\mathbf{f}_H$ can be computed directly from:

$$\mathbf{f}_H = \mathbf{P}_{H,E}\mathbf{P}_E^{-1}\mathbf{f}_E \tag{3.12}$$

From which we can set the variables values x_i equal to y_i for all $i \leq a$. Since its possible for the matrix $\mathbf{P}_E$ to be singular, a method such as Singular Value Decomposition (SVD) should be used to invert $\mathbf{P}_E$.

Using IPM for computing the conditional probabilities is efficient as long as e is small. The running time for the algorithm is $O(e^3 + eh)$.

3.2 Recurrent Linear Network

The method described above, IPM, is useful for approximating conditional probabilities when the number of evidence functions is small, since the running time is $O(e^3 + eh)$. The Recurrent Linear Network (RLN), as we will describe it, has a running time of $O(h^2 + eh)$ and is more efficient when the number of evidence functions is larger.

3.2.1 Structure of the RLN

The RLN consists of a network where each node represents a feature function. The value at each node y_i corresponds to the approximated probability of f_i, given the set of evidence variables X_E.

The nodes are fully connected with the weights $\omega_{i,j}$ representing the relations between nodes. Critical to the working of the network is the following theorem:

Theorem 1:

There exists a single set of weights $\omega_{i,j}$, that for any X_E and for all $f_i \in F_H$:

$$y_i = \sum_k \lambda_{i,k} y_k = \sum_j \omega_{i,j} y_j \tag{3.13}$$

for all k such that $f_k \in F_E$ and for all j such that $f_j \in F - f_i$.

Proof:

It is known that for any three variables X_i, X_j and X_k:

$$P(x_i|x_k) = \sum_{x_j} P(x_i|x_j, x_k) P(x_j|x_k) \tag{3.14}$$

If f_{H-i} is the set of hidden feature functions without f_i then:

$$P(f_i|x_E) = \sum_{f_{H-i}} P(f_i|f_{H-i}, x_E) P(f_{H-i}|x_E) \tag{3.15}$$

If f_{-i} is the set of all feature functions without f_i and $\mathbf{P}_{-i}$ is the matrix of all pairwise conditionally probabilities between the functions within f_{-i} then using 3.12:

$$P(f_i|x_E) = \sum_{f_{H-i}} \mathbf{P}_{i,-i} \mathbf{P}_{-i}^{-1} f_{-i} P(f_{H-i}|x_E) \tag{3.16}$$

Rearranging the summation:

$$P(f_i|x_E) = \mathbf{P}_{i,-i} \mathbf{P}_{-i}^{-1} \sum_{f_{H-i}} f_{-i} P(f_{H-i}|x_E) \tag{3.17}$$

Since $\sum_{f_{H-i}} f_j P(f_{H-i}|x_E) = P(f_j|x_E) = y_j$:

$$P(f_i|x_E) = \mathbf{P}_{i,-i} \mathbf{P}_{-i}^{-1} y \tag{3.18}$$

Therefore the set of weights $\omega_{i,j}$ with:

$$\omega_i = \mathbf{P}_{i,-i} \mathbf{P}_{-i}^{-1} \tag{3.19}$$

will satisfy 3.13. ■

Theorem 1 states that a single set of weights ω can be used to compute the output values y for any set of evidence variables X_E. Our only problem is the value y_i is dependent on other y_js that

may not be known. The values that correspond to evidence functions, i. e. $f_i \in F_E$, are set equal to the values of the functions, however the y_i corresponding to hidden functions must be computed iteratively. A simple iterative method to solve for all y_i is as follows:

$$y_i^{j+1} = (1-\sigma)y_i^j + \sigma \sum_k \omega_{i,k} y_k^j \text{ for all } f_i \in F_H \tag{3.20}$$

and

$$y_i^{j+1} = f_i(x) \text{ for all } f_i \in F_E \tag{3.21}$$

The parameter σ controls the rate of convergence.

If ω^E is the matrix where:

$$\omega_{i,j}^E = \left\{ \begin{array}{cc} \omega_{i,j} & f_i \in F_H \\ 1 & f_i \in F_E, i = j \\ 0 & f_i \in F_E, i \neq j \end{array} \right\} \tag{3.22}$$

then y_i^∞ is an eigenvector of ω^E with an eigenvalue of one. The algorithm described above can be viewed as the power method for finding eigenvectors of a matrix. As long as the largest eigenvalue of ω^E is one, the algorithm will converge to the correct result. It is known that the convergence rate for the power method is relative to the ratio:

$$\frac{|\nu_1|}{|\nu_2|} \tag{3.23}$$

where ν_1 and ν_2 are the two largest eigenvalues of ω^E. In later sections we will discuss methods for increasing this ratio in relation to function confidence.

Since we know the vector y has an eigenvalue of one we may also compute y from the following:

$$(\omega^E - I)y = 0 \tag{3.24}$$

where I is the identity matrix. A common method for solving problems of this type is conjugate gradient.

3.2.2 Learning the Weights

In the previous section we learned that the weights can be directly computed using:

$$\omega_i = \mathbf{P}_{i,-i}\mathbf{P}_{-i}^{-1} \tag{3.25}$$

To compute the weights using this equation, the pairwise conditional probability matrix $\mathbf{P}$ must first be computed. Unfortunately, there may not be enough data to compute $\mathbf{P}$ directly. While we can restrict our selection of feature functions to those that occur frequently, it may still be the case that we never know the value of f_i given the value of some function f_j in the training set. In these cases there will be blank entries in the matrix $\mathbf{P}$.

One method for solving this problem is to compute the weights $\omega_{i,j}$ using a function that iterates through the training data.

We learned in the previous chapter that the weights λ may be computed from the following set of functions:

$$\bar{P}(X_i = 1, f_j) = \sum_{l \in e} \lambda_{i,l} \bar{P}(f_j, f_l) \text{ for all } j \tag{3.26}$$

Similarly the weights ω can be computed from:

$$\bar{P}(f_i, f_j) = \sum_{l \in c} \omega_{i,l} \bar{P}(f_j, f_l) \text{ for all } j \tag{3.27}$$

Since $\bar{P}(f_i) = \frac{1}{m} \sum_{j \in m} f_i(t_j)$:

$$\frac{1}{m} \sum_{k \in m} f_i(t_k) f_j(t_k) = \sum_{l \in c} \omega_{i,l} \frac{1}{m} \sum_{k \in m} f_j(t_k) f_l(t_k) \text{ for all } j \tag{3.28}$$

Rearranging the summations:

$$\sum_{k \in m} f_i(t_k) f_j(t_k) = \sum_{k \in m} f_j(t_k) \sum_{l \in c} \omega_{i,l} f_l(t_k) \text{ for all } j \tag{3.29}$$

Setting the function equal to zero:

$$\sum_{k \in m} \left(f_j(t_k) \left(f_i(t_k) - \sum_{l \in c} \omega_{i,l} f_l(t_k) \right) \right) = 0 \text{ for all } j \tag{3.30}$$

For each training example t_k, we can view $f_i(t_k) - \sum_{l \in c} \omega_{i,l} f_l(t_k)$ as the prediction error for feature function f_i. The error is multiplied by the amount of contribution $f_j(t_k)$ from each function. The

weights $\omega_{i,j}$ may then be updated using the following function:

$$\Delta\omega_{i,j} = \delta \sum_{k \in m} \left(f_j(t_k) \left(f_i(t_k) - \sum_{l \in c} \omega_{i,l} f_l(t_k) \right) \right) \tag{3.31}$$

This equation is essentially a gradient decent method, as is commonly implemented for many neural network algorithms. To ensure the combined magnitude of the weights is minimized a drag term can be added to the equation:

$$\Delta\omega_{i,j} = \delta \sum_{k \in m} \left(f_j(t_k) \left(f_i(t_k) - \sum_{l \in c} \omega_{i,l} f_l(t_k) \right) \right) - drag(\omega_{i,j}) \tag{3.32}$$

where:

$$drag(\omega_{i,j}) = \left\{ \begin{array}{cc} \tau\omega_{i,j} + \psi & \omega_{i,j} > 0 \\ 0 & \omega_{i,j} = 0 \\ \tau\omega_{i,j} - \psi & \omega_{i,j} < 0 \end{array} \right\} \tag{3.33}$$

for some τ and $\psi \ll \delta$. As before, the nodes are not allowed to feedback on themselves so we force:

$$\omega_{i,i} = 0 \text{ for all } i \tag{3.34}$$

In cases where the training data is fully observable, i. e. every variable is known in each training example, equation 3.32 can be efficiently implemented since the y_is can be computed directly from the training data. For cases in which data is missing, the values y_i need to be iteratively computed before the weights can be updated. If this needs to be done for each training example, it can require a large amount of computation. For these cases, it is typically faster to compute the weights directly from the pair-wise probability matrix **P** if possible.

3.2.3 Example

To demonstrate weight learning, let us consider a simple example problem. Our problem has four variables: X_1 = cloudy, X_2 = sprinkler, X_3 = rain and X_4 = wet grass. We will use five feature functions, the bias function f_0 and one for each variable. The network is trained using data created from the Bayesian network in figure 3.1:

The pairwise probability matrix **P** is shown in table 3.1. Table 3.2 shows the weights computed for the RLN. The same values would be found using either the matrix **P** directly or by computing the weights iteratively from the data with no drag.

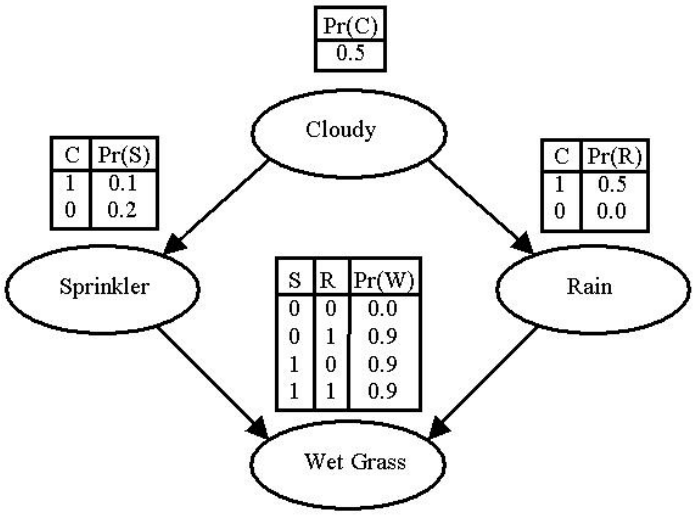


Figure 3.1: Wet Grass Bayesian network: C = cloudy, S = sprinkler, R = rain and W = wet grass.

$P(f_i\|f_j)$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=0$	1	1	1	1	1
$i=1$	0.500	1	0.333	1	0.735
$i=2$	0.150	0.100	1	0.100	0.400
$i=3$	0.250	0.500	0.167	1	0.667
$i=4$	0.333	0.495	0.900	0.900	1

Table 3.1: The pair-wise probability matrix **P** for the Wet Grass example.

$\omega_{i,j}$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	0.36	0	-0.09	0.70	-0.05
$i=2$	0.06	-0.03	0	-0.64	0.79
$i=3$	-0.03	0.20	-0.57	0	0.78
$i=4$	0.03	-0.02	0.74	0.81	0

Table 3.2: The weights $\omega_{i,j}$ computed for the RLN in the wet grass problem.

Test	Clouds	Sprinkler	Rain	Wet Grass
1	**1**	0.09, 0.10	**1**	0.89, 0.90
2	**1**	0.00, 0.02	0.17, 0.10	**0**
3	0.96, 1.00	**0**	0.94, 1.00	**1**
4	**0**	0.58, 1.00	0.42, 0.00	**1**

Table 3.3: Several example sets of evidence values for the wet grass problem. The value of the left is that found by the network and the value on the right is the true value. Evidence values are shown in bold.

We can easily test that these weights agree with the pair-wise probability table, for example:

$$P(f_2|f_1) = \omega_{2,0}P(f_0|f_1) + \omega_{2,1}P(f_1|f_1) + \omega_{2,3}P(f_3|f_1) + \omega_{2,4}P(f_4|f_1) = \tag{3.35}$$

$$0.06 * 1.0 - 0.03 * 1.0 - 0.64 * 0.5 + 0.79 * 0.5 = 0.1 \tag{3.36}$$

The columns of **P** form a set of stable states for the network. That is, $y_i^{j+1} = y_i^j$ for all y_i. Table 3.3 shows stable states for several different sets of evidence variables (evidence variable values are shown in bold.) The RLN produces fairly accurate results for most tests. For test 4, the errors are large, but it still considers the probability of the sprinkler being on as greater than it raining.

3.2.4 Weight Properties

An important feature of the weight matrix ω is that typically most weights will be close to or equal to zero. Theoretically, the amount of computation needed to compute the conditional probabilities is $O(h(h+e)I)$ where I is the number of iterations. Since for most real world problems, the weights ω will be a sparse matrix the running time can be much faster.

In general, the following can be said for the weights between conditionally independent functions:

Theorem 2:

$$\omega_{i,j} = 0 \text{ if } (f_i \amalg f_j|f_k) \text{ for some } f_k \tag{3.37}$$

That is, the weight $\omega_{i,j}$ will be equal to zero if there exists a feature function f_k such that f_i is conditionally independent of f_j given f_k.

Proof:

If f_i is conditionally independent of f_j given f_k then:

$$\bar{P}(f_i|f_j) = \bar{P}(f_i|f_k)\bar{P}(f_k|f_j) + \bar{P}(f_i|\neg f_k)\bar{P}(\neg f_k|f_j) \tag{3.38}$$

where $P(\neg f_k) = P(f_k = 0) = 1.0 - P(f_k)$.

We will assume the weights $\omega_{i,l}$, $l \neq j$ have converged. If $w_{i,j} = 0$ after the entire set of weights converges then the following must be true, we abbreviate $\sum_{l \in m} f_i(t_l)$ as $\sum_T f_i$:

$$\Delta\omega_{i,j} = \sum_T \left(f_j \left(f_i - \sum_{l \neq j} \omega_{i,l} f_l \right) \right) = 0 \tag{3.39}$$

Thus we need to prove:

$$\sum_T f_j f_i = \sum_T f_j \sum_{l \neq j} \omega_{i,l} f_l \tag{3.40}$$

Expanding the right hand side of 3.40 we find:

$$\sum_T f_j \sum_{l \neq j} \omega_{i,l} f_l = \sum_T f_j f_k \sum_{l \neq j} \omega_{i,l} f_l + \sum_T f_j (1 - f_k) \sum_{l \neq j} \omega_{i,l} f_l \tag{3.41}$$

Since f_i and f_j and conditional independent given f_k, we know $\bar{P}(f_i | f_j, f_k) = \bar{P}(f_i | f_k)$, or equivalently:

$$\frac{\sum\limits_T f_i f_j f_k}{\sum\limits_T f_j f_k} = \frac{\sum\limits_T f_i f_k}{\sum\limits_T f_k} \tag{3.42}$$

Using 3.42 and dividing 3.41 by $\sum_T f_j$ we find:

$$\frac{\sum\limits_T f_j \sum\limits_{l \neq j} \omega_{i,l} f_l}{\sum\limits_T f_j} = \left(\frac{\sum\limits_T f_k \sum\limits_{l \neq j} \omega_{i,l} f_l}{\sum\limits_T f_k} \right) \left(\frac{\sum\limits_T f_j f_k}{\sum\limits_T f_j} \right) + \left(\frac{\sum\limits_T (1 - f_k) \sum\limits_{l \neq j} \omega_{i,l} f_l}{\sum\limits_T (1 - f_k)} \right) \left(\frac{\sum\limits_T f_j (1 - f_k)}{\sum\limits_T f_j} \right) \tag{3.43}$$

Since the righthand side of 3.43 is equivalent to the righthand side of 3.38:

$$\frac{\sum\limits_T f_j \sum\limits_{l \neq j} \omega_{i,l} f_l}{\sum\limits_T f_j} = \bar{P}(f_i | f_k) \bar{P}(f_k | f_j) + \bar{P}(f_i | \neg f_k) \bar{P}(\neg f_k | f_j) = \bar{P}(f_i | f_j) = \frac{\sum\limits_T f_i f_j}{\sum\limits_T f_j} \tag{3.44}$$

Therefore:

$$\sum_T f_j f_i = \sum_T f_j \sum_{l \neq j} \omega_{i,l} f_l \tag{3.45}$$

and

$$\omega_{i,j} = 0 \text{ if } (f_i \amalg f_j | f_k) \text{ for some } f_k \tag{3.46}$$

■

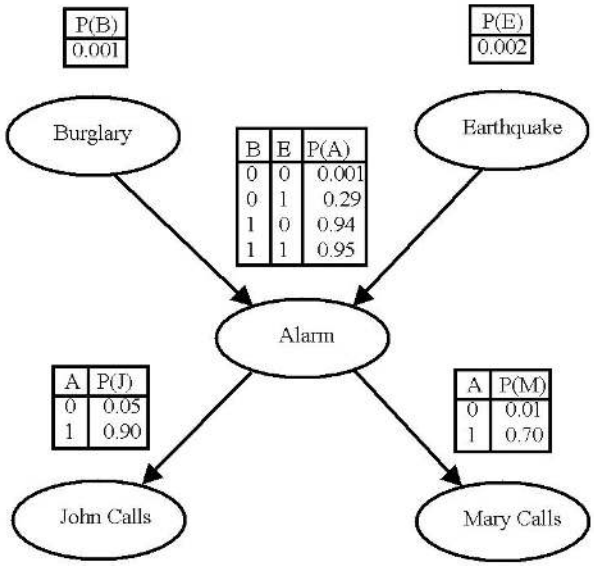


Figure 3.2: Alarm Bayesian network: B = burglary, E = earth quake, A = alarm, J = John calls and M = Mary calls.

It is important to note that the weight $\omega_{i,j}$ will only be guaranteed to converge to zero if f_i and f_j are conditionally independent given a *single* feature function f_k. If f_i and f_j are only conditionally independent given two or more functions then $\omega_{i,j}$ may not converge to zero. For example, the weights in table 3.2 between "cloudy" and "wet grass" are not zero since they are only conditionally independent given both "sprinkler" and "rain".

To help in understanding Theorem 2 let us consider the following Bayesian network first created by Judea Pearl [60, 42] representing the following situation:

> A new burglar alarm was installed in your home. It is supposed to sound an alarm whenever there is a burglary; unfortunately it also goes off occasionally during an earthquake. Whenever the alarm goes off you've instructed John and Mary to call you. Given that John or Mary called you'd like to compute the probably that a burglary or earthquake occurred. The assumption is made that burglary and earthquake are conditionally independent of John and Mary calling given alarm.

When we train our network with six feature functions, using data generated from the Bayesian network in figure 3.2, we get the weights in table 3.4.

Variables that were conditionally independent given a single variable in the Bayesian network have weights equal to zero in the RLN. A graphical representation can be seen in figure 3.3. Notice

$\omega_{i,j}$	$j = 0$	$j = 1$	$j = 2$	$j = 3$	$j = 4$	$j = 5$
$i = 1$	0	0	-0.120	0.403	0	0
$i = 2$	0.002	-0.335	0	0.355	0	0
$i = 3$	-0.001	0.864	0.271	0	0.025	0.092
$i = 4$	0.050	0	0	0.851	0	0
$i = 5$	0.010	0	0	0.685	0	0

Table 3.4: The weights $\omega_{i,j}$ computed for the RLN in the alarm problem .

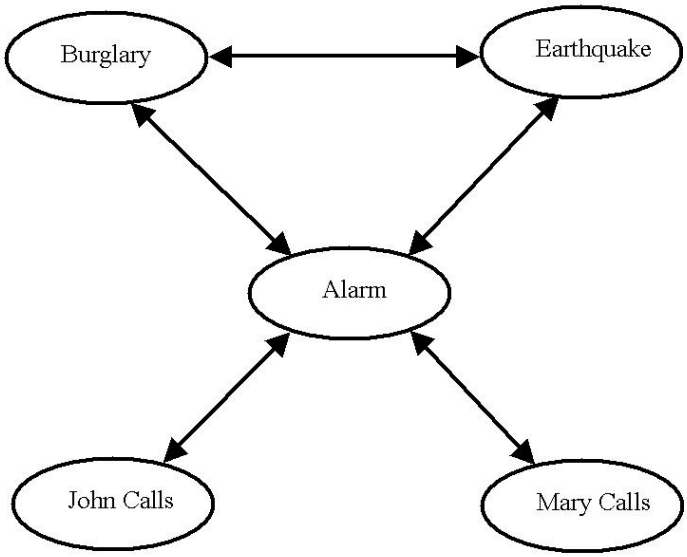


Figure 3.3: Alarm RLN network: Weights with values not equal to zero are shown.

that even though burglary and earthquake don't have a direct link between them in the Bayesian network they aren't conditionally independent given alarm. Variables sharing the same children are dependent in Bayesian networks, therefore their weights aren't equal to zero in our network.

3.2.5 Multiple Stable States

Up to this point, we have assumed that the RLN possesses a single stable state of values Y^{∞} for each set of evidence functions F_E. We learned that the weights ω may be computed directly from a matrix of pair-wise probability values **P**. Unfortunately, the matrix **P** may be singular. That is, **P** may have rank less than b. If the weights are learned from a singular matrix **P**, then the network may possess multiple stable states for some sets of evidence functions.

For example, consider the probabilities in table 3.5. Notice that the probability of f_1 occuring is equivalent to f_2, and f_3 is equivalent to f_4. The matrix **P** will only have rank 3 which is less than

$P(f_i\|f_j)$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=0$	1.0	1.0	1.0	1.0	1.0
$i=1$	0.5	1.0	1.0	0.3	0.3
$i=2$	0.5	1.0	1.0	0.3	0.3
$i=3$	0.5	0.5	0.5	1.0	1.0
$i=4$	0.5	0.5	0.5	1.0	1.0

Table 3.5: A pair-wise probability matrix **P** will rank less than b.

$\omega_{i,j}$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=1$	0.0	0.0	1.0	0.0	0.0
$i=2$	0.0	1.0	0.0	0.0	0.0
$i=3$	0.0	0.0	0.0	0.0	1.0
$i=4$	0.0	0.0	0.0	1.0	0.0

Table 3.6: The weights $\omega_{i,j}$ computed for the RLN.

$b = 5$. The weights found using the weight update rule are shown in table 3.6.

This set of weights only captures a fraction of the data given to us in the probability table above. The values of y_1 and y_2 are dependent only on each other. The network failed to learn that if $f_3 = 1$ then the probability of f_1 and f_2 is 0.3. If $F_E = \{f_3\}$, there are an infinite number of stable states for the network with $f_1 = f_2$. These multiple stable states will exist regardless of any additional feature functions that are added to the network.

The matrix **P** being singular is equivalent to there being a redundant feature function in the network. That is, there exists a feature function who's value can be exactly computed from the other feature functions. In our example above, f_1 is redundant given f_2 and f_3 is redundant given f_4. We can find the redundant functions by simply looking at the errors produced while training network. If there is zero absolute error when updating the function's corresponding weights then the function must be redundant.

We cannot just remove the redundant functions to solve the problem. For example, consider a network that has four variables labelled "Canada", "United States", "Mexico" and "Cold". Our task is to compute the probability of someone living in either Canada, United States or Mexico. The only additional information we're given is whether its cold where they live or not. They can only live in one and only one country at a time. In this example, the knowledge of two countries completely determines whether the person lives in the third country. If the person doesn't live in Canada or the United States they must certainly live in Mexico. Therefore one of the three countries is redundant. If we trained the network the weights from "cold" to any of the countries would be zero since

τ	y_1^∞	y_2^∞	y_3^∞	y_4^∞
0.0000	1.000	1.000	0.000	0.000
0.0001	1.000	0.977	0.499	0.499
0.0010	1.000	0.971	0.485	0.485
0.0100	1.000	0.769	0.385	0.385

Table 3.7: Converged values for different τ using equation 3.32 and table 3.5.

"cold" isn't needed to compute any of the countries probabilities. In this case, while removing one of the countries solves our redundancy problem we create another problem. If we removed Canada from our network and we discovered the person didn't live in Canada how would we convey this information to the network? Due to this problem, we've devised an alternative method for handling redundant functions without removing them.

One possible method for removing the incorrect extraneous stable states is to increase the drag on the weight values in equation 3.32. The conditional probabilities will not be learned exactly, but the extraneous stable states will be removed. For the above example, if $F_E = f_1$ and $f_1 = 1$, then Y^∞ will converge to the following values for different τ as shown in table 3.7.

Unfortunately, this solution has a couple problems. First, the rate of convergence for Y is inversely proportional to the drag coefficient τ. The greater the accuracy, the longer it takes to converge. A small value for τ creates multiple eigenvectors with eigenvalues close to 1 for the matrix ω^E. Since the ys are essentially computed using the power method for finding eigenvectors on ω^E, this will result in slow convergence. Second, as τ approaches 0, the changes in weight values are minimal, however variable values computed from the weights can change substantially. We would like the network to be fairly stable with respect to small changes in weight values. Despite these problems this is still the easiest and most effective method for handling multiple stable states that we've found.

3.2.6 Large Scale Domains

The naive approach to implementing the Recurrent Linear Network (RLN) may be too computationally inefficient for large scale databases. Many real world problems can have well over 10,000 variables. The standard approaches to implementing the RLN are $O(h^2 + eh)$. For problems within large domains, we need the algorithm to run in approximately $O(h + e)$ time to obtain timely results.

If the number of evidence variables is small the Inverse Probability Method (IPM) should be used. Its complexity is $O(eh + e^3)$, which in roughly linear when e is small. For large e, the RLN is more efficient, but still is not close to running in linear time. Fortunately, there are many

improvements that we can make on the RLN to increase efficiency.

When computing the output values for the RLN, we proposed a simple gradient descent algorithm. To increase the convergence rate various well known methods can be used such as momentum or conjugate gradient. We will discuss several additions that are more specific to the RLN below.

Sparse Weight Matrix

The following equation is used to update the function values using the RLN:

$$y_i^{j+1} = (1-\sigma)y_i^j + \sigma \sum_k \omega_{i,k} y_k^j \text{ for all } f_i \in F_H \tag{3.47}$$

If some weight $\omega_{i,j}$ is zero it will have no effect on the output of the network. Thus we may ignore all weights with zero value. If a significant number of the weights are zero we can create a list of all the non-zero weights and compute the outputs directly from them, without wasting time computing the contribution from the zero weights.

We have already discussed that a weight $\omega_{i,j}$ will equal zero if function f_i is conditionally independent of function f_j given some function f_k. The weight will also equal zero if the value of function f_i is never observed at the same time as function f_j.

While most functions will not be strictly conditionally independent, many will be nearly so. In such cases the weight $\omega_{i,j}$ between two functions f_i and f_j will be small. If we wish to increase computational efficiency, we can force all such small weights to zero with minimal loss of accuracy. The RLN is fairly robust with respect to weight removal, i.e forcing weights to zero that lie close to zero.

Push vs. Pull

In the previous section we discussed how to increase the efficiency of the network by taking advantage of zero weights. The efficiency of the network may also be increased by taking advantage of outputs y_k that have zero values.

The function update rule is:

$$y_i^{j+1} = (1-\sigma)y_i^j + \sigma \sum_k \omega_{i,k} y_k^j \text{ for all } f_i \in F_H \tag{3.48}$$

We will call this the ”pull” model. Using the above equation the new value y_i^{j+1} is found by computing the sum of the other outputs multiplied by their respective weights. That is, we pull the

value of each output y_k to find the value of y_i.

Another method for computing the output values is to add a small amount to each y_i for every other y_k. That is:

$$\Delta y_i^{j+1} = \sigma \omega_{i,k} y_k^j \text{ for all } i \tag{3.49}$$

We are "pushing" the value of each output onto the other outputs, essentially reversing the order in which the calculations are done.

If an output y_k has a value of zero it has no effect on the outputs of the network. If we "push" the weights instead of "pulling" them, we can check to see if y_k is equal to zero before doing the calculations. If $y_k = 0$ for a significant number of outputs we can reduce the number of calculations that need to be done.

Under certain circumstance, we may ignore all outputs that lie below some threshold. This can greatly decrease the computational requirements of the network. Unfortunately, the accuracy of the network is also degraded. The final values of the network will be less than those generated without using a threshold. If only the relative values of the outputs are needed the network will still produce reliable results. However, using a threshold can result in gross errors when exact values are needed.

Stability

As the size of the network increases it becomes more prone to instabilities. As we mentioned earlier, the RLN will have multiple stable states if the probability matrix **P** is singular. A nearly singular matrix can have a similar effect. While there may still be a single stable state, the convergence rate will become increasing slow as the matrix approaches one that is singular.

As the number of variables increase, the chances of finding a singular matrix increases. Trying to find the bottom of an error space when it is shallow can lead to instabilities. The instabilities may be caused by approximations made or numerical round-off.

Currently, the only reliable method we've found for reducing instabilities and ensuring a reasonable convergence rate is to increase the drag coefficient τ. While this may increase the error of the network, the gains in efficiency typically make this a reasonable tradeoff.

If the IPM is used instead of the RLN, then instabilities are less likely to occur. A matrix inversion technique such as SVD can handle singular matrices without problems while maintaining a reasonable amount of efficiency.

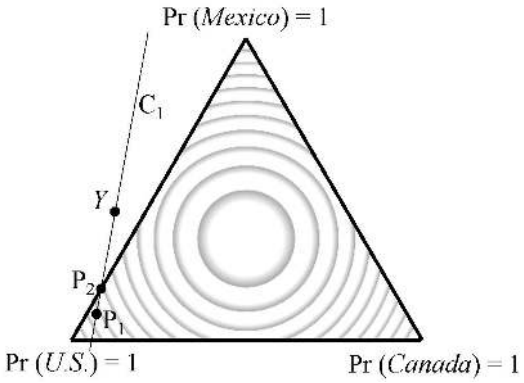


Figure 3.4: A constraint C_1 that has a minimum Y that lies outside the simplex.

3.3 Other Concerns

We have described the basic algorithm for the IPM and RLN. In this section we will discuss several issues involving problems with the algorithms.

First, we will discuss methods for restricting the output values to lie between 0 and 1 for the IPM and RLN. For sets of evidence values that don't frequently occur this can be a common problem. Our second problem deals with inconsistencies when computing probabilities estimates such as $P(X_i, X_j|x_E)$.

3.3.1 Bounding the Probability Values

Since we are not enforcing the bounding constraints, it is possible for some of the output values y to be greater than 1 or less than 0. To illustrate this point consider the simplex and constraint in figure 3.4. The value along the constraint C_1 that minimizes the squared values y actually lies outside of the simplex. Thus the algorithm as it is described to this point will produce probability values outside the range of 0 to 1. The naive approach to solving this problem is to compute the Y values and set any value greater than one to one and any value less than zero to zero. It is possible to do slightly better.

When using the RLN, we can bound the y_is during each iteration, instead of just bounding the final values. Thus during each iteration the following step would be added:

$$y_i^j = \left\{ \begin{array}{ll} 0 & y_i^j < 0 \\ y_i^j & 0 \leq y_i^j \leq 1 \\ 1 & y_i^j > 1 \end{array} \right\} \tag{3.50}$$

Using this addition, if one of the values y_i lies above 1 or below 0 its corresponding feature function is essentially added to the evidence functions F_E. This can produce better results since the network's values are updated appropriately given the new information, i. e. that values can only range from 0 to 1.

We may also find the same answer by computing the conditional probability using the IPM. We compute the values as before, except we check to see if any values lie outside the range from 0 to 1. If a y_i is greater than one, we set its corresponding feature function to one and add it to the evidence functions and recompute the values. The same is done for y_i less than zero.

3.3.2 Computing $P(X_i, X_j|x_E)$

The goal of our network was to compute $P(X_i|x_E)$ for all $X_i \in X_H$. For some applications we may want to approximate more complex probabilities such as $P(X_i = 1, X_j = 1|x_E)$ for some $X_i, X_j \in X_H$. One method for computing such values would be to add a feature function f_k that equaled one whenever both X_i and X_j equaled one. Depending on the size of the training set, we may not be able to compute reliable estimates for the new constraint values created by adding f_k. If enough data doesn't exist an alternative approach is needed.

Another method for finding $P(X_i = 1, X_j = 1|x_E)$ is to compute it using the following equality:

$$P(X_i = 1, X_j = 1|x_E) = P(X_i = 1|X_j = 1, x_E)P(X_j = 1|x_E) \tag{3.51}$$

The problem can then be broken into two steps. We compute $P(X_j = 1|x_E)$ and then add $X_j = 1$ to the evidence values x_E and compute $P(X_i = 1|X_j = 1, x_E)$. By symmetry we may also compute $P(X_i = 1, X_j = 1|x_E)$ using:

$$P(X_i = 1, X_j = 1|x_E) = P(X_j = 1|X_i = 1, x_E)P(X_i = 1|x_E) \tag{3.52}$$

That is, we switch the roles of X_i and X_j. Unfortunately, the results using the RLN or the IPM will differ depending on what order we compute the values.

For example, in our Wet Grass example we get the following values:

$$y_R = 0.500 \text{ with } X_E = \{C = 1\}$$

$$y_W = 0.495 \text{ with } X_E = \{C = 1\}$$

$$y_R = 0.755 \text{ with } X_E = \{C = 1, W = 1\}$$

$$y_W = 0.900 \text{ with } X_E = \{C = 1, R = 1\}$$

Then we find our estimates for $P(\mathrm{R} = 1, \mathrm{W} = 1|\mathrm{C} = 1)$ to be:

$$0.500 * 0.900 = 0.450$$

or

$$0.495 * 0.755 = 0.374$$

Thus our estimates of $P(\mathrm{R} = 1, \mathrm{W} = 1|\mathrm{C} = 1)$ vary depending on the order in which its computed. This problem arises from Rényi's generalization of Shannon's entropy. By relaxing the third rule for entropy, the distributions obtained may not be consistent as we have just seen.

3.4 Function Confidence

In the previous sections we assumed that each constraint was known exactly or with the same level of confidence. In real world problems this is not the case. For some constraints a large amount of data will be available and thus we can be confident of its value. However, many constraints will have less supporting data. We may not want to fully enforce a constraint that has a value for which we're not confident.

Previously, the value of the constraint on f_i given f_j was set equal to $\bar{P}(f_i|f_j)$. Depending on the amount of training data available, the approximation of $P^*(f_i|f_j)$ using $\bar{P}(f_i|f_j)$ may not be accurate. Instead of assigning constraint values equal to $\bar{P}(f_i|f_j)$, we will compute a new set of constraint values $\mathbf{C}_{i,j}$ that takes into account our confidence in the value of the observed probability. The matrix $\mathbf{C}$ of constraint values can be used in place of $\mathbf{P}$ in the equations used for the RLN and IPM.

The constraint values will be computed using a linear combination of $\bar{P}(f_i|f_j)$ and $\bar{P}(f_i)$:

$$\mathbf{C}_{i,j} = c_j \bar{P}(f_i|f_j) + (1.0 - c_j)\bar{P}(f_i) \tag{3.53}$$

The value $c_j \in [0, 1]$ is called the confidence value for the function f_j. If we have no confidence in the function f_j we set $c_j = 0$ and $\mathbf{C}_{i,j} = \bar{P}(f_i)$. A function with full confidence will have a confidence value of 1 resulting in $\mathbf{C}_{i,j} = \bar{P}(f_i|f_j)$. If $\mathbf{C}_{i,j} = \bar{P}(f_i)$, then the constraint values will be equivalent to those for $\mathbf{C}_{i,0}$ since $\mathbf{C}_{i,0} = \bar{P}(f_i|f_0) = \bar{P}(f_i)$. Thus the constraint $\mathbf{C}_{i,j}$ will add no additional information to the network and the output of the network will not depend on f_j. The closer the value of c_j is to one the greater the impact of f_j on the network.

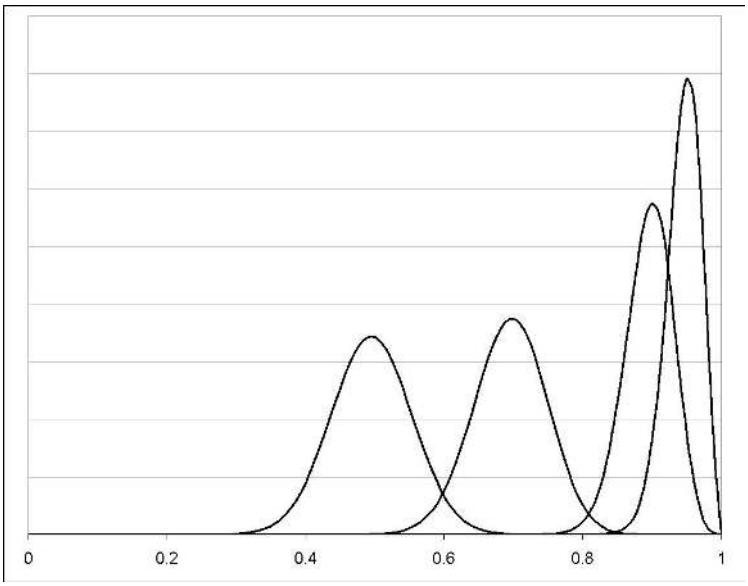


Figure 3.5: Four example distributions centered around $P(f_i) = 0.5, 0.7, 0.9, 0.95$. Notice the tail of the 0.95 case is truncated near 1.

How do we determine the confidence values c_j? We will present a method for computing c_j that assumes the observed probabilities $\bar{P}$ have a normal distribution. By computing the variances of the distributions we can determine c_j.

The observed probability $\bar{P}(f_i)$ has a binomial distribution. It is known that as the number of observations increase, the binomial distribution can be closely approximated using a normal distribution. The only exception is the tails are truncated, since $0 \leq \bar{P}(f_i) \leq 1$, see figure 3.5. To simply our analysis, we will assume $\bar{P}(f_i)$ has a normal distribution.

As the number of observations for the variable f_i increase, the variance of $\bar{P}(f_i)$ decreases. If m is the number of entries within the training set, the variance σ^2 will decrease relative to $\frac{1}{m}$, as shown in figure 3.6. If we know the value of $P(f_i)$ we can compute the value of the variance as $\frac{P(f_i)P(\neg f_i)}{m}$.

For the rest of the section we will assume that each function is observed with high enough frequency to allow accurate estimation of $\bar{P}(f_i)$. For example, if $P(f_i) = 0.5$ and $m = 1000$ then the variance of $\bar{P}(f_i)$ will equal $\frac{P(f_i)P(\neg f_i)}{m} = \frac{0.5*0.5}{1000} = 0.00025$ with a standard deviation of 0.0158.

We will assume the observed conditional probabilities $\bar{P}(f_i|f_j)$ also have normal distributions. Unlike $\bar{P}(f_i)$, the variance of $\bar{P}(f_i|f_j)$ decreases relative to $\frac{1}{mP(f_j)}$. Therefore, the error of the observed probability $\bar{P}(f_i|f_j)$ may be much larger than $\bar{P}(f_i)$ or $\bar{P}(f_j)$ if $\bar{P}(f_j)$ is small.

If $P(f_j)$ is close to zero then we may not be able to accurately compute $P(f_i|f_j)$ since we will not observe $f_j = 1$ with high enough frequency. The same holds for $P(\neg f_j)$ and $P(f_i|\neg f_j)$. Consider

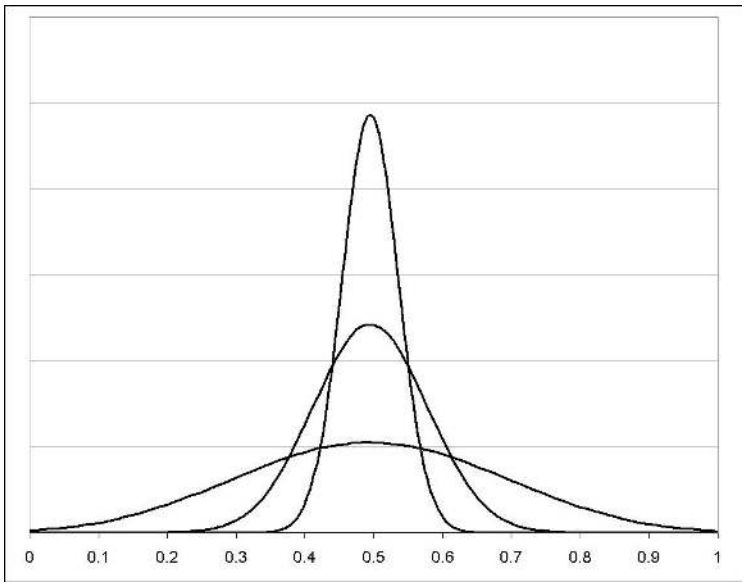


Figure 3.6: Three example distributions centered around $P(f_i) = 0.5$ for $m = 10, 50, 200$.

figure 3.7, if we assume the values $P(f_i)$ and $P(f_j)$ are known then point B is fixed. The values of $P(f_i|\neg f_j)$ and $P(f_i|f_j)$, corresponding to points A and C respectively, are related as follows:

$$\frac{P(f_i) - P(f_i|f_j)P(f_j)}{P(\neg f_j)} = P(f_i|\neg f_j) \tag{3.54}$$

That is, the three points A, B and C lie along a line. Thus point A is completely determined given B and C, and C is completely determined given A and B. If we constrain the value of $P(f_i|f_j)$ within our network we are also constraining the value of $P(f_i|\neg f_j)$. Therefore we must consider the error estimates of both $P(f_i|\neg f_j)$ and $P(f_i|f_j)$ when computing our confidence in f_j. Since point B is fixed, we are essentially trying to find the slope of the line in figure 3.7. If $P(f_j)$ is close to one as shown in figure 3.7(b) and 3.7(c) the variance of $P(f_i|f_j)$ decreases while the variance of $P(f_i|\neg f_j)$ increases. The variance of the slope varies relative to $\frac{1}{mP(f_j)P(\neg f_j)}$.

If we have no confidence in our constraint, we set $c_j = 0$ resulting in $P(f_i|f_j) = P(f_i|\neg f_j) = P(f_i)$. Thus there will be zero slope. We may interpret this as assuming a prior distribution around the zero slope. We will assume the prior distribution is a normal distribution with a variance of σ_0^2 and mean 0. We can convolve this with the distribution obtained from computing the slope of the observed probabilities. Figure 3.8 plots the the prior distribution, along with the estimated distribution from the observed slope $\bar{P}(f_i|f_j) - \bar{P}(f_i|\neg f_j)$. If we determine the variance of our observed slope to be

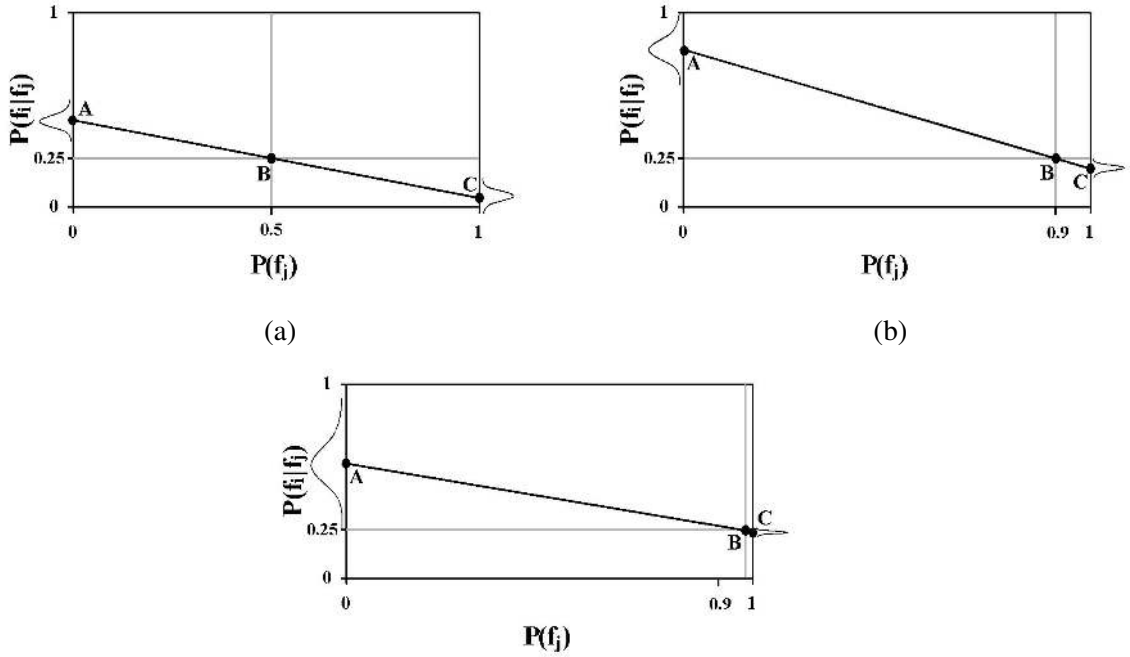


Figure 3.7: Various examples of relationship between $P(f_i|\neg f_j) \rightarrow A$, $P(f_i) \rightarrow B$ and $P(f_i|f_j) \rightarrow C$. As point B approaches point C, the variance of C decreases.

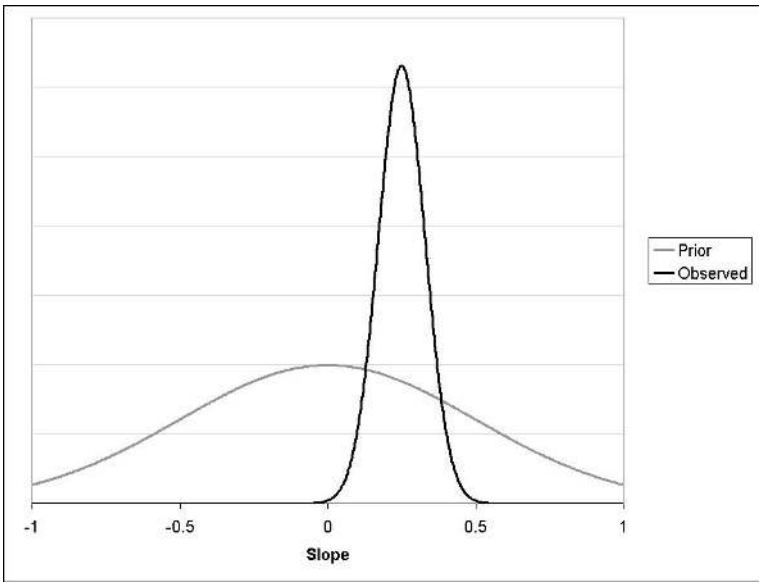


Figure 3.8: Example distributions for the slope having a value of zero (the wide distribution) and a value equal to the observed slope (the narrow distribution.)

$\sigma_j^2 = \frac{\varphi}{mP(f_j)P(\neg f_j)}$, then the mean or expected value of the two combined distributions will equal:

$$\frac{\sigma_0^2}{\frac{\varphi}{mP(f_j)P(\neg f_j)} + \sigma_0^2}\left(\bar{P}(f_i|f_j) - \bar{P}(f_i|\neg f_j)\right) \tag{3.55}$$

Using both 3.54 and 3.55 we find the confidence value c_j is equal to:

$$c_j = \frac{\sigma_0^2}{\frac{\varphi}{mP(f_j)P(\neg f_j)} + \sigma_0^2} \tag{3.56}$$

All that remains for computing c_j is setting the values of σ_0^2 and φ. Since φ is a function of unknown values, namely $P(f_i, f_j)$ and $P(f_i, \neg f_j)$, we will set its value to a ratio of σ_0^2. Using $\varphi = r\sigma_0^2$ equation 3.56 reduces to:

$$c_j = \frac{mP(f_j)P(\neg f_j)}{r + mP(f_j)P(\neg f_j)} \tag{3.57}$$

The value of r may be set to a wide range of values depending on how much we'd like to bias the network's results towards the prior probabilities. In practice, r is typically set between 2 and 10. A graph of confidence values for several values of r and increasing m is shown in figure 3.9.

Within this section we have discussed how to compute the confidence values c_j that are used to

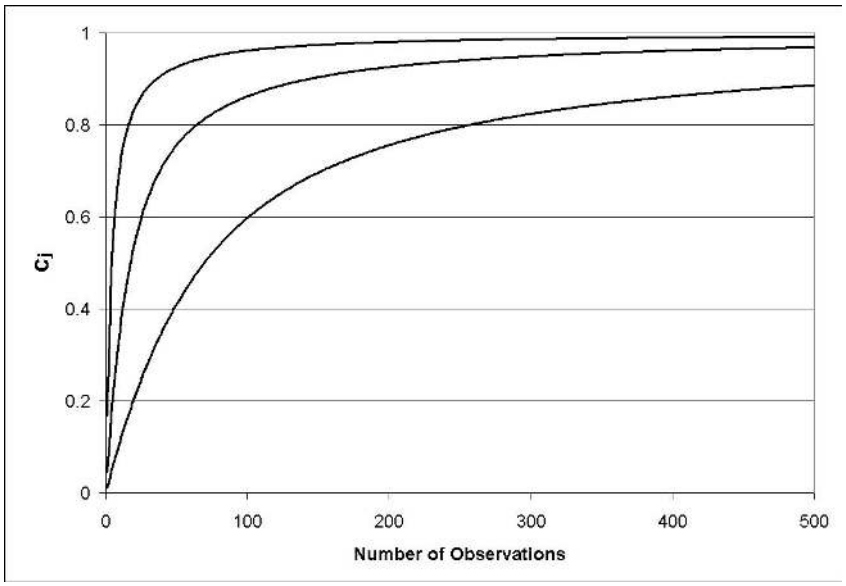


Figure 3.9: Example confidence values for varying values of r as the number of observations increase.

find the final constraint values $\mathbf{C}_{i,j} = c_j \bar{P}(f_i|f_j) + (1.0 - c_j)\bar{P}(f_i)$. If we compute the network weights directly from the probability matrix $\mathbf{P}$ we may substitute the values of $\bar{P}(f_i|f_j)$ with $\mathbf{C}_{i,j}$ to find the appropriate weights. We may also use confidence values if we wish to compute the weight values directly from the data using 3.32.

3.4.1 Computing the Weights Directly From Data

Achieving the same results using 3.32 is surprisingly simple using the drag coefficient τ. After the weight $\omega_{i,j}$ has converged $\Delta\omega_{i,j} = 0$:

$$\Delta\omega_{i,j} = 0 = \delta \sum_{k \in m} \left(f_j(t_k) \left(f_i(t_k) - \sum_{l \in c} \omega_{i,l} f_l(t_k) \right) \right) - \tau\omega_{i,j} \tag{3.58}$$

By adjusting the value of τ, we can achieve the same results as computing the weights directly from $\mathbf{C}$. We'll derive the equation for τ as a function of r and δ for the case when the network contains three constraint functions, f_0, f_i and f_j. The same result applies to networks with any number of constraints, but the mathematics are more complicated.

Expanding the weight update equation we find:

$$0 = \delta \sum_{k \in m} f_j f_i - \delta \sum_{k \in m} f_j \omega_{i,j} - \delta \sum_{k \in m} f_j \omega_{i,0} - \tau \omega_{i,j} \tag{3.59}$$

Dividing by m and using $\frac{\sum_m f_i}{m} = \bar{P}(f_i)$:

$$0 = \delta \bar{P}(f_j f_i) - \delta \bar{P}(f_j) \omega_{i,j} - \delta \bar{P}(f_j) \omega_{i,0} - \frac{\tau \omega_{i,j}}{m} \tag{3.60}$$

Since we assume there exists only three functions we know:

$$\bar{P}(f_j) \omega_{i,j} + \omega_{i,0} = \bar{P}(f_i) \Rightarrow \omega_{i,0} = \bar{P}(f_i) - \bar{P}(f_j) \omega_{i,j} \tag{3.61}$$

Substituting 3.61 in 3.60 and dividing by $\bar{P}(f_j)$:

$$0 = \delta \bar{P}(f_i | f_j) - \delta \omega_{i,j} - \delta \bar{P}(f_i) + \delta \bar{P}(f_j) \omega_{i,j} - \frac{\tau \omega_{i,j}}{m \bar{P}(f_j)} \tag{3.62}$$

Solving for $\omega_{i,j}$:

$$\omega_{i,j} = \frac{\delta \left(\frac{\bar{P}(f_i | f_j)}{\bar{P}(\neg f_j)} - \frac{\bar{P}(f_i)}{\bar{P}(\neg f_j)} \right)}{\delta + \frac{\tau}{m \bar{P}(\neg f_j) \bar{P}(f_j)}} \tag{3.63}$$

Rearranging we find:

$$\omega_{i,j} = \frac{\delta \left(\bar{P}(f_i | f_j) - \bar{P}(f_i | \neg f_j) \right)}{\delta + \frac{\tau}{m \bar{P}(\neg f_j) \bar{P}(f_j)}} \tag{3.64}$$

We will set c_j equal to:

$$c_j = \frac{\delta}{\delta + \frac{\tau}{m \bar{P}(\neg f_j) \bar{P}(f_j)}} \tag{3.65}$$

The value of the constraint $\mathbf{C}_{i,j}$ can be found using:

$$\mathbf{C}_{i,j} = \omega_{i,j} \mathbf{C}_{j,j} + \omega_{i,0} \mathbf{C}_{0,j} \tag{3.66}$$

The values of $\mathbf{C}_{j,j}$ and $\mathbf{C}_{0,j}$ are always equal to 1. Using equation 3.61 and 3.64 we find:

$$\mathbf{C}_{i,j} = c_j \left(\bar{P}(f_i | f_j) - \bar{P}(f_i | \neg f_j) \right) + \bar{P}(f_i) - c_j \bar{P}(f_j) \left(\bar{P}(f_i | f_j) - \bar{P}(f_i | \neg f_j) \right) \tag{3.67}$$

After rearranging the equation we find our final result:

$$\mathbf{C}_{i,j} = c_j \bar{P}(f_i|f_j) + (1 - c_j)\bar{P}(f_i) \tag{3.68}$$

Equation 3.68 is identical to equation 3.53. Therefore, by using the appropriate value of τ we may achieve the same results as above with:

$$c_j = \frac{\delta}{\delta + \frac{\tau}{m\bar{P}(\neg f_j)\bar{P}(f_j)}} = \frac{mP(f_j)P(\neg f_j)}{r + mP(f_j)P(\neg f_j)} \tag{3.69}$$

If we assume δ is fixed we may compute τ by:

$$\tau = r\delta \tag{3.70}$$

Chapter 4

Complex Feature Functions

The proper choice of complex feature functions can greatly increase the accuracy of the Recurrent Linear Network (RLN) or the Inverse Probability Method (IPM). A complex feature function is any binary function of multiple variables, such as:

$$\begin{aligned} f_j(x) &= x_2 \wedge x_5 \\ f_j(x) &= x_1 \vee x_4 \vee x_8 \\ f_j(x) &= (x_2 \vee x_5) \wedge (x_6 \vee x_8) \\ f_j(x) &= x_1 \oplus x_2 \oplus x_3 \end{aligned} \tag{4.1}$$

The role of the complex feature function is to account for higher order interactions between variables. The unit feature function imposes constraints on our estimates of the prior probabilities $P(x_i)$, while simple functions place constraints on probabilities conditioned on single variables $P(x_i|x_2)$ for $f_2(x) = x_2$. Complex feature functions can impose higher order constraints such as $P(x_i|x_2, x_5)$, if $f_j(x) = x_2 \wedge x_5$. By properly choosing the correct higher order constraints to enforce, we may greatly increase the accuracy of our methods.

When creating complex feature functions many aspects need to be considered. Is there enough data to accurately compute the constraint values? Will the feature function be mainly used as an evidence or hidden feature? Will other features be dependent on it, i. e. will it be useful to the network?

While our complex functions may be any binary function, traditionally only binary functions that can be represented using the *And* operator are usually considered. Previously, *And* functions have been used in naive Bayes networks [43, 59].

Within this chapter, we will describe several methods for adding complex feature functions. A

couple of these methods will only apply to the RLN since the magnitude of the weights between functions is used. Two methods, which use variable frequency and average error, can be applied to both the IPM and the RLN.

While complex functions may increase the accuracy of both methods, they can also increase the efficiency of the RLN. We will discuss the complex functions in relation to weight magnitude in the last section.

4.1 Adding Complex Functions for the IPM and the RLN

Within this section we will describe two methods for adding complex function when using the IPM and the RLN. Both methods will consider adding the traditional style of *And* functions.

4.1.1 High Frequency Pairs

For a function to be useful to our methods, it needs to have high discriminatory power. That is, we'd like a complex function to have a value of one as close to 50% of the time as possible. One method for finding potential complex functions is to consider all pairs of functions f_k and f_l such that $\bar{P}(f_k, f_l)(1.0 - \bar{P}(f_k, f_l))$ is high. For all pairs of functions f_k and f_l that are above some threshold we create a new complex function equal to:

$$f_j = f_k \wedge f_l \tag{4.2}$$

We may also consider the negatives $\neg f_k$ and $\neg f_l$ of f_k and f_l. Using this method we guarantee that the value of $P(f_j)$ will be as close to 0.5 as possible.

4.1.2 Analyzing Error Values

The real goal of adding complex functions to the network is to reduce error. If we know which variables have the most error associated with them, then we can add complex functions to help in computing these variables.

The average error for each variable X_i can be computed from the training data using:

$$error(x_i) = \frac{1}{m} \sum_{j \in m} \left| t_{j,i} - y_i \right| \tag{4.3}$$

For each variable X_i that has a large error or an error above some threshold we can search for

complex functions that may help in computing its value. If the functions f_k and f_l frequently have a value of $\bar{P}(f_k, f_l)(1.0 - \bar{P}(f_k, f_l))$ when X_i has a large error, we can create a new complex function equal to:

$$f_j = f_k \wedge f_l \tag{4.4}$$

4.2 Adding Complex Functions for the RLN

We can view creating complex functions for the RLN as a two step process. First, we analyze the training data, error rates and current network structure to find functions that are likely to be of high utility to the network. The second is to measure the utility of the complex functions once they are added to the network and the network is retrained. The complex functions with low utility are then pruned. We may be fairly liberal in adding complex functions in the first step since any functions that aren't useful to the network will be removed in the second step.

We will first discuss a method for computing the utility of a function using the weights of the RLN and issues related to pruning functions. Second we will discuss an additional method for adding complex functions to the RLN by analyzing weight values.

4.2.1 Utility of a Complex Function

A complex function is useful to the network if it reduces the overall error of the network. That is, after adding a new complex function the following should decrease:

$$error = \sum_{x_E} \bar{P}(x_E) \left| \bar{P}(X_i = 1 | x_E) - y_i \right| \tag{4.5}$$

Each complex function adds to the computational burden of the network. If it doesn't reduce the network error in proportion to the added computational costs, then the complex function shouldn't be added.

Unfortunately, computing the error reduction due to each complex function can itself be computationally expensive. Each complex function would need to be checked in conjunction with the other complex functions, since the reduction in error may not be independent of the other complex functions. Thus, all sets of complex functions under consideration would need to be added to the network in turn and the errors found. For large scale networks this method is computationally too inefficient.

An alternative or approximation of the above method is to view the magnitude of the weights

for a complex function f_j as a measure of its utility:

$$\sum_i \left|\omega_{i,j}\right| \tag{4.6}$$

While the summed magnitude of the weights can be a good predictor of utility, it is important to also consider the probability of f_j. If $P(f_j)$ is close to 0 the weights $\omega_{i,j}$ will have little effect on the final values of the network since $\omega_{i,j}y_i$ will be close to zero in almost all cases. Similarly if $P(f_j)$ is close to 1 the weights will have little overall effect, since $f_j \approx f_0$. For this reason we multiply the summed magnitude of the weights by $P(f_j)(1.0 - P(f_j))$ to obtain our final measure of utility:

$$P(f_j)(1.0 - P(f_j)) \sum_i \left|\omega_{i,j}\right| \tag{4.7}$$

4.2.2 Pruning Complex Functions

After the functions are added, the network is retrained and the utility of the complex functions can be computed using 4.7. Complex functions that have a utility below a certain threshold can be pruned from the network, after which the network must once again be trained.

When we are retraining the network, we must apply the appropriate drag coefficients to assure we minimize the weight values and create an efficient network. In the previous chapter, we discussed how the drag coefficient τ can be used to adjust the amount of confidence we have in each constraint. The drag coefficient ψ has a different purpose. By assigning a value greater than 0 to ψ, we can assure that the solution with the smallest number of non-zero weights is found.

For example, consider the case when $x_1 = x_2 \oplus x_3 \oplus x_4$; the parity function. Assume our current set of functions consists of those in table 4.1. Then the network will find the weights shown in table 4.1 when ψ is set to zero. The set of complex functions f_5, f_6, f_7 and f_8 can together exactly compute the parity function. Complex function f_9 computes the parity function directly. With ψ set to zero the weights are equally divided between them.

If we add a small amount of drag with $\psi = 0.0001$ we find the weights in table 4.2. Each weight gets the same amount of drag from ψ. Thus, the solution for X_1 given f_5, f_6, f_7 and f_8 gets four times the amount of drag as the solution using just f_9. This difference in drag causes the weights for f_5, f_6, f_7 and f_8 to converge to zero while the weight for f_9 converges to one.

The role of ψ in complex function pruning is very important. If all of the complex functions in table 4.1 were added to the network simultaneously and ψ was set to zero our measure of utility would be high for all complex functions, even though four of them are redundant. With ψ set to

f_j	$\omega_{1,j}$
$f_0 = 1$	0.0
$f_2 = x_2$	0.0
$f_3 = x_3$	0.0
$f_4 = x_4$	0.0
$f_5 = x_2 \wedge \neg x_3 \wedge \neg x_4$	0.2
$f_6 = \neg x_2 \wedge x_3 \wedge \neg x_4$	0.2
$f_7 = \neg x_2 \wedge \neg x_3 \wedge x_4$	0.2
$f_8 = x_2 \wedge x_3 \wedge x_4$	0.2
$f_9 = x_2 \oplus x_3 \oplus x_4$	0.8

Table 4.1: Weights for functions in the parity example when ψ is set to zero.

f_j	$\omega_{1,j}$
$f_0 = 1$	0.0
$f_2 = x_2$	0.0
$f_3 = x_3$	0.0
$f_4 = x_4$	0.0
$f_5 = x_2 \wedge \neg x_3 \wedge \neg x_4$	0.0
$f_6 = \neg x_2 \wedge x_3 \wedge \neg x_4$	0.0
$f_7 = \neg x_2 \wedge \neg x_3 \wedge x_4$	0.0
$f_8 = x_2 \wedge x_3 \wedge x_4$	0.0
$f_9 = x_2 \oplus x_3 \oplus x_4$	1.0

Table 4.2: Weights for functions in the parity example with ψ is set to 0.0001.

some value slightly greater than zero the network is able to discover the redundancies and remove the proper complex functions.

4.2.3 Analyzing Weight Values

Our final method for finding complex functions analyzes only the network weights. If we'd like to create a complex function f_j for which $\sum_i \left|\omega_{i,j}\right|$ is high, we may look at sets of functions that have similar weights. We will measure the similarity of the weights between two functions f_k and f_l by:

$$\frac{\sum_i \omega_{i,k} \cdot \omega_{i,l}}{\sqrt{\sum_i \omega_{i,k}^2} \sqrt{\sum_i \omega_{i,l}^2}} \tag{4.8}$$

Equation 4.8 computes the cosine of the angle between the weights for function f_k and f_l. The closer to one, the more similar the weights. If a pair of functions has similar weights, we can combine them to form a new complex function f_j equal to:

$$f_j = f_k \vee f_l \tag{4.9}$$

Notice that in this case we are using the *Or* function instead of the *And* function. If two functions have similar weights they are likely to represent similar features, thus using an *Or* function is more appropriate. For example, consider the language modeling task. We are trying to predict what word is coming next in a sentence given some set of previous words. If we analyze the weights we might find the words "North" and "South" are similar. This is not surprising since similar words typically follow the two words "North" and "South." By grouping the words "North" and "South" together we are forming a more general function that may better represent the underlying sentence structure. When forming groupings, it is typically most efficient to group many functions together at once rather than two at a time. This creates fewer complex functions and thus creates a more efficient network.

All methods for finding potential complex functions may be used in conjunction or separately. Typically the method that analyzes weight values is used first to form groupings. The previous two methods can then be used to handle *n*th order interactions within the data, or handle "special cases."

x_2	x_3	$P(X_1 = 1 \mid x_2, x_3)$
0	0	0.10
0	1	0.70
1	0	0.73
1	1	0.68

Table 4.3: Probability of $X_1 = 1$ conditioned upon X_2 and X_3 when $X_1 \approx x_2 \vee x_3$. $P(X_2 = 1) = P(X_3 = 1) = 0.5$

4.3 Increasing the Efficiency of the RLN

When it comes to constraining the probability distribution we can show that many binary functions will produce the same results. Any function that is the logical opposite of another function will produce identical results. For example, consider the function $f_j = x_2 \wedge x_3$ and $f'_j = \neg x_2 \vee \neg x_3$. The function f_j is the logical opposite of function f'_j with $f_j = 1 - f'_j$. If we are computing $P(X_1 = 1 \mid x_2, x_3)$, then f_j constrains the probability $P(X_1 = 1 \mid X_2 = 1, X_3 = 1) = P(X_1 = 1 \mid f_j)$ and f'_j constrains the probability $P(X_1 = 1 \mid X_2 = 0 \text{ or } X_3 = 0) = P(X_1 = 1 \mid f'_j)$. The constraints are equivalent since we know $P(X_1 = 1)$ and:

$$P(X_1 = 1 \mid f_j)P(f_j) = P(X_1 = 1) - P(X_1 = 1 \mid f'_j)P(f'_j) \tag{4.10}$$

It is possible for the network to produce the same results using different functions that aren't logical opposites. In the two evidence variable case if we are trying to compute $P(x_i \mid x_2, x_3)$ we need four constraints to fully represent all possible combinations of x_2 and x_3. The three functions f_0, f_2 and f_3 provide us with three constraints, so we only need one more. Adding a complex function, either $x_2 \wedge x_3$ or $x_2 \vee x_3$, will provide us with a fourth constraint. The final result produced by the network will be equivalent regardless of the complex function used, however the magnitude of the weights for the network my vary greatly. For instance, if we have three variables in our network and the value of the first variable is roughly a function of $x_2 \vee x_3$ as shown in table 4.3, the total magnitude of the weights will vary greatly depending on our choice of function.

Table 4.4 shows the combined magnitude of the weights using an *Or* function corresponding to $x_2 \vee x_3$ and an *And* function corresponding to $\neg x_2 \wedge \neg x_3$. The same may also be true for *And* functions. If the third variable is roughly a function of $x_2 \wedge x_3$, table 4.5, then we obtain the results in table 4.6.

In our two examples above we've demonstrated that the choice of using *And* or *Or* functions can affect the overall magnitude of the weights for the network. This effect can be very dramatic

	$\sum \lvert w_{i,j} \rvert$	$\# \lvert w_{i,j} \rvert > 0.01$	Prediction Error
No Complex	2.37	9	0.163
Or	5.23	14	0.008
And	8.73	16	0.008

Table 4.4: Weights of network when $X_1 \approx x_2 \vee x_3$: Given no complex functions, an *Or* function and an *And* function.

x_2	x_3	$P(X_1 = 1 \lvert x_2, x_3)$
0	0	0.10
0	1	0.11
1	0	0.09
1	1	0.68

Table 4.5: Probability of $X_1 = 1$ conditioned upon X_2 and X_3 when $X_1 \approx x_2 \wedge x_3$. $P(X_2 = 1) = P(X_3 = 1) = 0.5$

	$\sum \lvert w_{i,j} \rvert$	$\# \lvert w_{i,j} \rvert > 0.01$	Prediction Error
No Complex	2.61	9	0.160
Or	7.65	16	0.003
And	6.02	12	0.003

Table 4.6: Weights of network when $X_1 \approx x_2 \wedge x_3$: Given no complex functions, an *Or* function and an *And* function.

Set 1

f_i	$\omega_{1,i}$
1	0.0
x_2	1.0
x_3	1.0
x_4	1.0
$x_2 \wedge x_3$	-2.0
$x_2 \wedge x_4$	-2.0
$x_3 \wedge x_4$	-2.0
$x_2 \wedge x_3 \wedge x_4$	4.0

Set 2

f_i	$\omega_{1,i}$
1	0.0
x_2	0.0
x_3	0.0
x_4	0.0
$x_2 \wedge \neg x_3 \wedge \neg x_4$	1.0
$\neg x_2 \wedge x_3 \wedge \neg x_4$	1.0
$\neg x_2 \wedge \neg x_3 \wedge x_4$	1.0
$x_2 \wedge x_3 \wedge x_4$	1.0

Set 3

f_i	$\omega_{1,i}$
1	0.0
x_2	0.0
x_3	0.0
x_4	0.0
$x_2 \oplus x_3 \oplus x_4$	1.0

Table 4.7: Varying sets of functions with corresponding weights generated for the RLN.

depending on the properties of the training data. If a variable is dependent on other variables based on a function such as the parity function, the choice of complex functions can become very important. Let us assume we have four variables with the first variable x_1 dependent on the other three x_2, x_3 and x_4 based on the parity function. The parity function assigns a value of 1 to a function if the sum of the inputs is odd and a value of 0 otherwise. Figure 4.7 shows the weights for x_1 given three different sets of complex functions.

The magnitude and number of non-zero weights varies greatly for each set of complex functions, even though the network will produce identical results regardless of the complex function set used. In set one, seven weights are non-zero, and the magnitude of the weights grows exponentially. Set two does a better job since only 4 non-zero weights exist and the magnitude of the weights stays

constant at 1.0. The final set contains a complex function that exactly computes the parity function, and thus only a single complex function is needed.

Each set of complex functions has its own advantages and disadvantages. If we were to incrementally add complex functions to our network, starting with complex functions with 2 inputs and then 3, we would probably end up with a set of complex functions such as set one. While set one produces the correct output, it has more non-zero weights than the other sets and the magnitude of the weights is large. As the number of inputs for the parity function increase, the magnitude of the weights will increase exponentially. Large weights can lead to instability in the network and other problems.

Set two creates a complex function for each instance of x_2, x_3 and x_4 for which $x_1 = 1$. Using this method the magnitude of the weights stay fixed at 1.0. Unfortunately, the number of complex functions needed to correctly represent the output function is exponential with respect to the number of inputs for the parity function.

Set three is ideal since it contains a single complex function that exactly computes the parity function. However, in real world applications there will typically not exist enough data or it will be computationally too expense to find the exact function.

Chapter 5

Experimental Results

We will examine the results of using URQE within several applications and in comparison to other methods. Our applications include: collaborative filtering, image retrieval and language modeling. Along with these applications, we will examine URQE in relation to Bayesian networks on some synthetic data sets. We implement URQE using both IPM and RLN, depending on which is more efficient for the specific application.

The accuracy of the network will be tested in the collaborative filtering and language modeling data sets. The image retrieval application will mainly test the efficiency of the network. We will explore the results of adding complex functions in the collaborative filtering and language modeling applications.

5.1 Amount of Data vs. Accuracy

Within this section we will explore the results of the using URQE vs. other algorithms when there exists a varying amount of training data. Thus we can examine how each algorithm balances errors due to bias and variance. The algorithms used for comparison include: nearest neighbor, naive Bayes, and Bayesian networks.

Our data set is comprised of 7 variables from a hypothetical medical diagnosis problem first proposed in [49]. The problem is comprised of three diseases: tuberculosis, lung cancer and bronchitis, two causes: visiting Asia and smoking, and two symptoms: positive X-ray and Dyspnoea. Our task is to predict the probability of the diseases given the causes and symptoms.

The problem is described as follows:

> Shortness-of-breath (Dyspnoea) may be due to tuberculosis, lung cancer or bronchitis,

or none of them, or more than one of them. A recent visit to Asia increases the chances of tuberculosis, while smoking in known to be a risk factor to both lung cancer and bronchitis. The results of a single chest X-ray do not discriminate between lung cancer and tuberculosis, as neither does the presence or absence of Dyspnoea.

The nearest neighbor (NN) algorithm computes probabilities as follows:

$$\bar{P}(X_i = 1|x_E) = \frac{\sum_{j\in m} t_{j,i}\delta(j)}{\sum_{j\in m} \delta(j)} \tag{5.1}$$

where

$$\delta(j) = \left\{ \begin{array}{cc} 1 & t_{j,k} = x_k \text{ for all } X_k \in X_E \\ 0 & \text{otherwise} \end{array} \right\} \tag{5.2}$$

If there exist zero corresponding samples in the training data, then the probabilities are assigned a value equal to their prior probabilities $P(X_i|x_E) = P(X_i)$.

Our tests using Naive Bayes consists of three separate Bayesian networks, one for each disease as in figure 5.2. For the Bayesian network we will assume we know beforehand the actual relation of the variables to each other as shown in figure 5.1, i. e. which variables are independent of others. Thus the Bayesian network is given more knowledge than the other algorithms. We could attempt to compute the structure of the network directly from the data [30, 61, 75, 76], but there doesn't exist enough data in many cases to find it correctly. The parameters, i. e. the conditional probabilities for each node, are computed from the data. If a certain combination of evidence variables are found to be impossible given the networks computed parameters, then the conditional probabilities were assigned values equal to their prior probabilities $P(X_i|x_E) = P(X_i)$.

URQE was implemented using IPM. For our initial tests we used 11 complex functions corresponding to each combination of "Asia", "Smoking", "X-ray" and "Dyspnoea", table 5.1. The confidence coefficient r was set to 8.

Four experiments were run giving the algorithms 1, 2, 3 and 4 evidence values. Every combination of values was chosen from "Asia", "Smoking", "X-ray" and "Dyspnoea." Thus there were 8 combinations for the 1 evidence variable case, 24 for the 2 variable case, 32 for the 3 variable case and 16 for the 4 variable case. We tested each algorithm on data sets ranging from 20 to 1,000 training cases and 2,000 to 20,000 training cases. The error is computed from the average absolute deviation for the diseases "tuberculosis", "lung cancer" and "bronchitis." The results in 5.3 and 5.4 display the average from 5,000 tests. The results weren't weighted by the probability of the evidence values occurring.

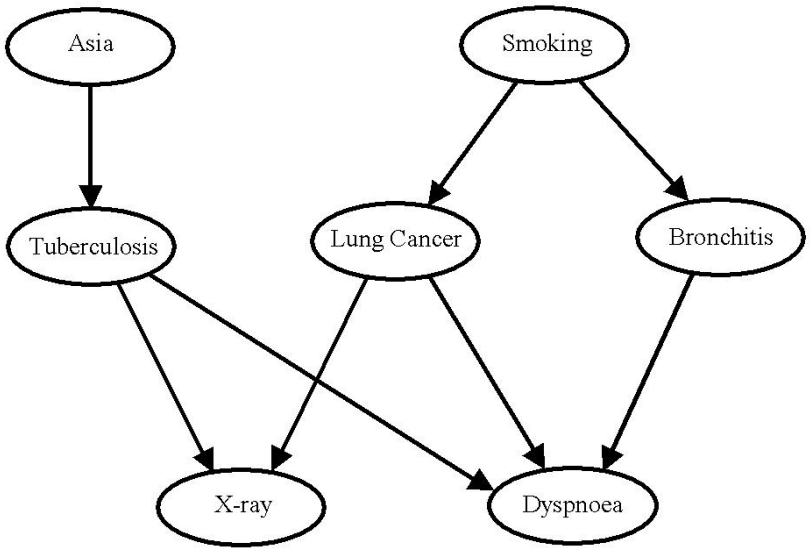


Figure 5.1: Structure of Bayesian network.

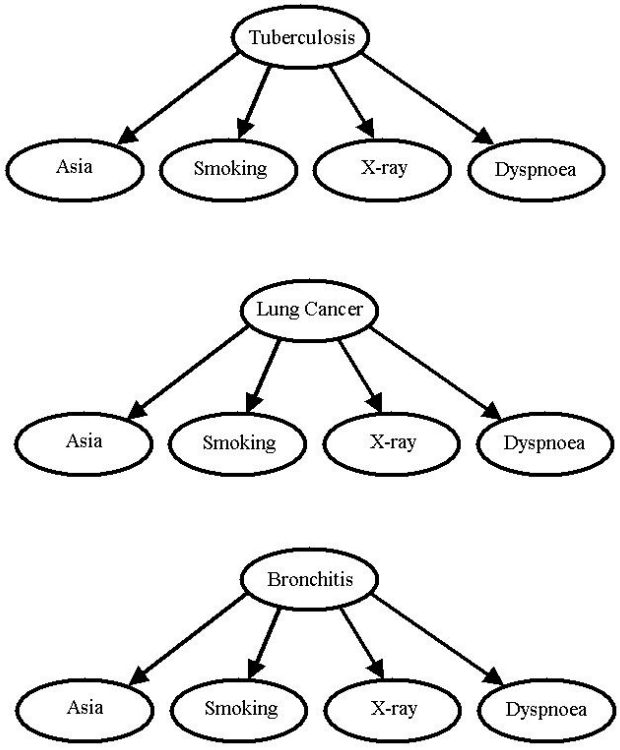


Figure 5.2: Structure of naive Bayes networks.

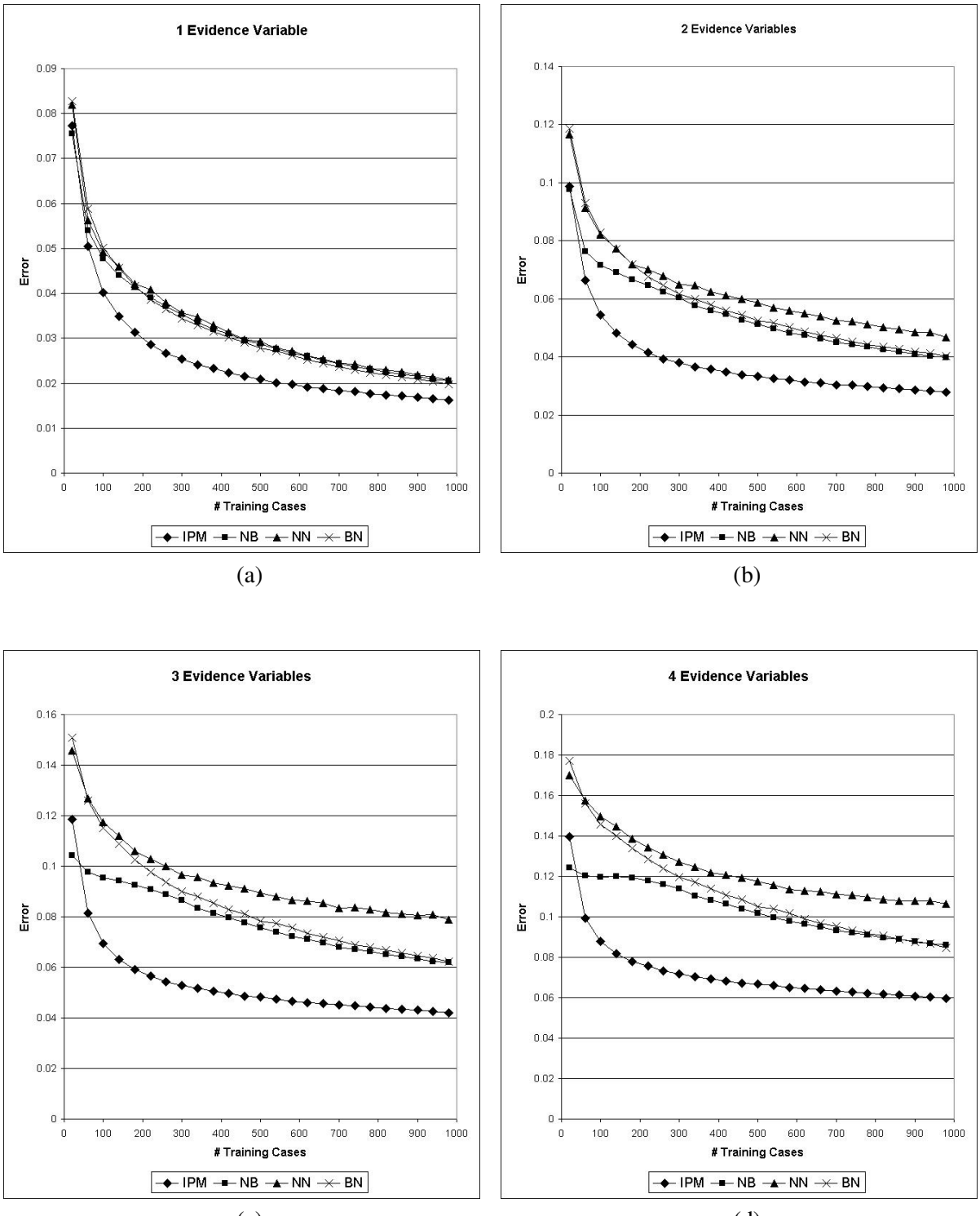


Figure 5.3: Results for training sizes of 20 to 1,000 for the Inverse Probability Method (IPM), Naive Bayes (NB), Nearest Neighbor (NN) and Bayesian Network (BN). Errors are measured using the absolute deviation between the predicted and actual values for the diseases. Four tests were run using 1 (a), 2 (b), 3 (c) and 4 (d) evidence variables. Confidence intervals are too small to be displayed.

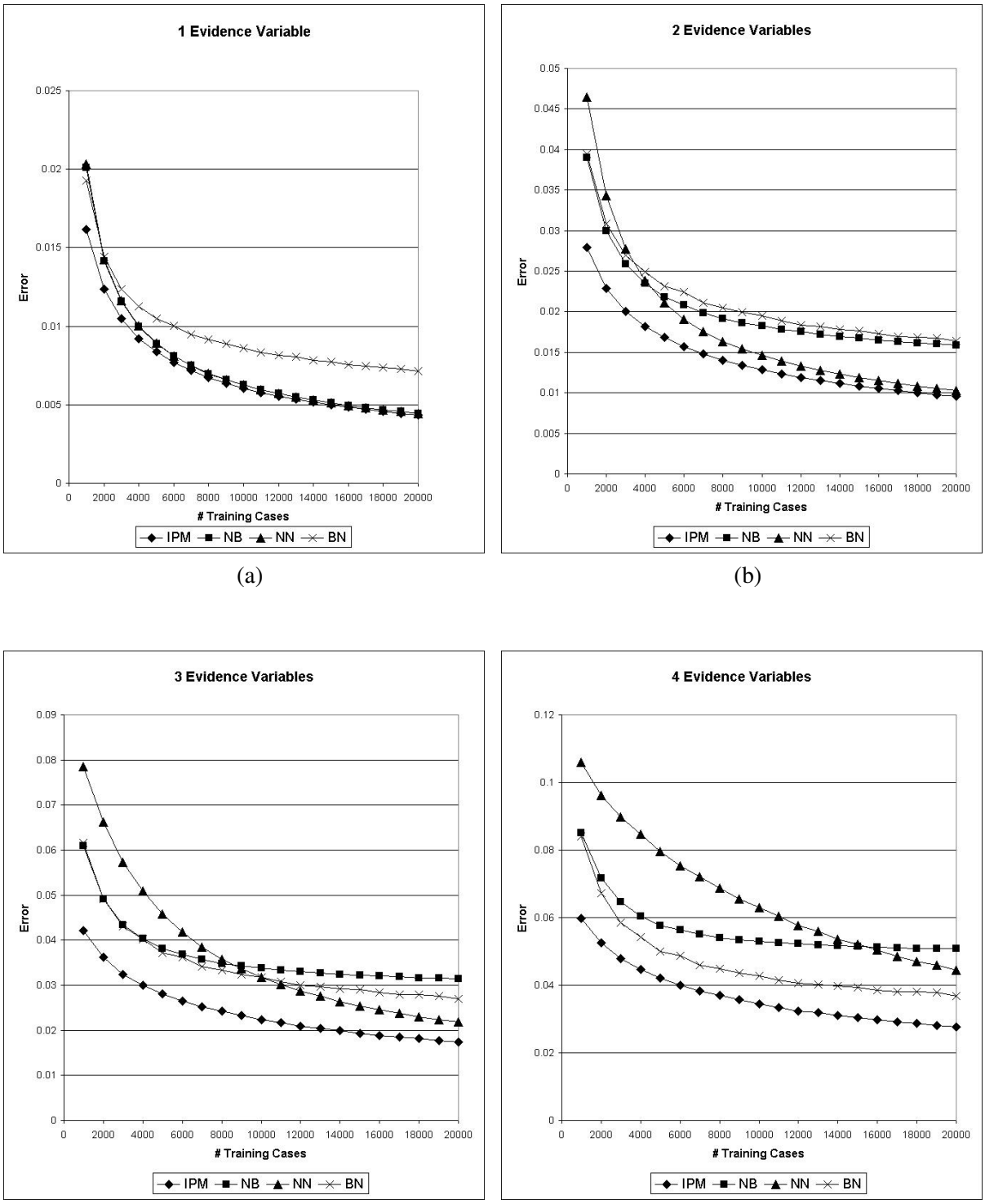


Figure 5.4: Results for training sizes of 2,000 to 20,000 for the Inverse Probability Method (IPM), Naive Bayes (NB), Nearest Neighbor (NN) and Bayesian Network (BN). Errors are measured using the absolute deviation between the predicted and actual values for the diseases. Four tests were run using 1 (a), 2 (b), 3 (c) and 4 (d) evidence variables.

Complex Functions
Asia ∧ Smoking
Asia ∧ X-ray
Asia ∧ Dyspnoea
Smoking ∧ X-ray
Smoking ∧ Dyspnoea
X-ray ∧ Dyspnoea
Asia ∧ Smoking ∧ X-ray
Asia ∧ Smoking ∧ Dyspnoea
Asia ∧ X-ray ∧ Dyspnoea
Smoking ∧ X-ray ∧ Dyspnoea
Asia ∧ Smoking ∧ X-ray ∧ Dyspnoea

Table 5.1: Complete set of complex functions used for the IPM.

The best algorithms vary depending on the number of evidence variables and the number of training cases provided. NB produces better results relative to the other algorithms when fewer training cases are used. Since NB isn't capable of exactly learning the distribution for 2 or more evidence variables, the errors plateau for larger training sizes. NN typically produces worse results than the other algorithms for smaller training sizes. However, as the number of training cases increase, NN is capable of learning the exact probabilities and can surpass the results given by NB. BN is more accurate when more evidence variables are known. For fewer evidence variables, the errors in predicting the network's parameters can compound to increase the overall error of the network.

The results for URQE implemented using the IPM out perform the other algorithms across all training sizes and experiments. Since 11 complex functions were used, the algorithm is capable of fully learning the probability values; resulting in reduced error due to bias. The constraint confidence values guaranteed that only constraint values with low variance were used. Thus minimizing the error due to variance. The other algorithms' parameters didn't have corresponding confidence values so their variance errors were larger. NB doesn't have as many parameters so it performed better in cases with small training sizes, but its inability to fully learn the distribution resulted in a large bias errors.

For larger problem domains it will typically be too computationally expensive to add complex functions corresponding to every combination of evidence values. Thus the IPM will exhibit some bias error. To explore the consequences of adding fewer complex functions we tested the IPM with 0, 3 and 6 complex functions, shown in tables 5.2 and 5.3, along with the complete set of 11. The

Complex Functions
Smoking ∧ X-ray
Smoking ∧ Dyspnoea
X-ray ∧ Dyspnoea

Table 5.2: Set of 3 complex functions used for the IPM.

Complex Functions
Asia ∧ Smoking
Asia ∧ X-ray
Asia ∧ Dyspnoea
Smoking ∧ X-ray
Smoking ∧ Dyspnoea
X-ray ∧ Dyspnoea

Table 5.3: Set of 6 complex functions used for the IPM.

sets of complex functions were chosen by selecting those with highest utility. The results for 2,000 to 50,000 training cases can be seen in figure 5.5. As expected the results degrade as the number of complex functions decrease. However, the results using 6 complex functions is nearly as accurate as those for the set of 11. For fewer than 5,000 training cases the results using only 3 complex functions are nearly identical to the complete set.

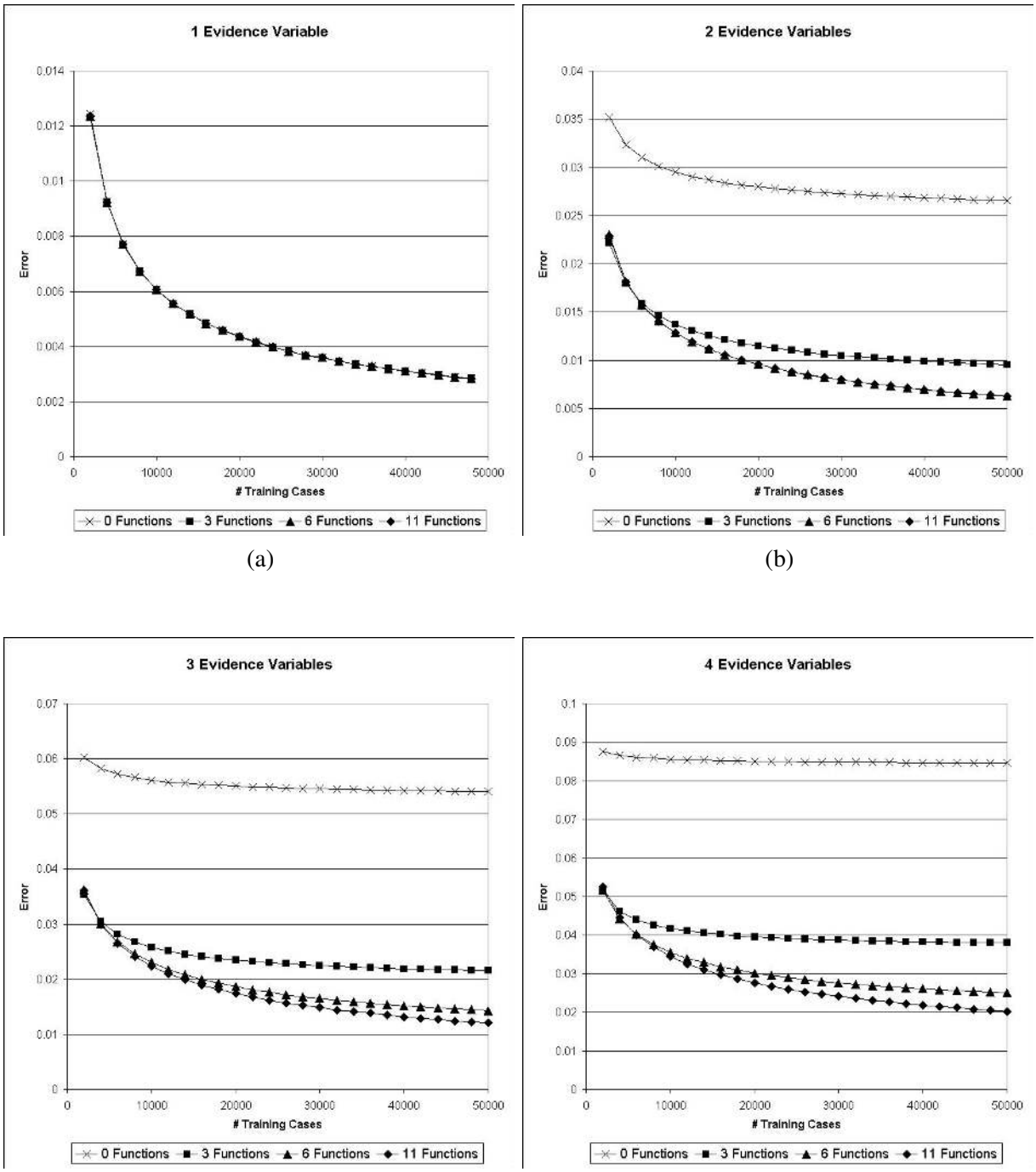


Figure 5.5: Results for training sizes of 2,000 to 50,000 for the Inverse Probability Method (IPM) using varying numbers of complex functions. Errors are measured using the absolute deviation between the predicted and actual values for the diseases. Four tests were run using 1 (a), 2 (b), 3 (c) and 4 (d) evidence variables.

5.2 Collaborative Filtering

Collaborative Filtering (CF) is the task of predicting a user's actions based on their and others' previous actions [3, 5, 31, 37, 50, 67]. A simple example is predicting what books a customer might purchase given their previous buying history and the buying histories of others.

We model the problem as predicting the probability of a user choosing item A given previous histories of choices. Each item has a corresponding variable and feature function. A value of one for a variable corresponds to the user choosing, buying or liking the item. Similarly, a value of zero corresponds to a user not choosing, not buying or disliking the item.

Since the number of evidence variables for CF is relatively small with respect to the total number of variables, we use the IPM which is more efficient for these types of problems instead of the RLN.

Related to collaborative filtering is the application of information retrieval. ME has been used for document retrieval by combining knowledge from keywords [12, 11, 25, 38, 39, 40]. Unlike our use of ME for collaborative filtering, ME is only used to relate keywords to documents rather than documents to documents. Even though the initial results are discouraging [40], we believe ME may be used effectively for information retrieval [25].

5.2.1 Web Browsing Behavior

Our first CF task is predicting which web pages a user will visit given their previous browsing history. Our database consists of 32,711 training cases, where each case is a list of web pages an individual user visited. The testing database consists of 5,000 cases. The data sets are supplied courtesy of Microsoft Corporation from users logs generated during one day in 1996.

To measure the accuracy of our results we will use the same error metric as described in [5, 31, 37]. For each test case the web pages visited will be split randomly into input and measurement sets. The input set will be given to the IPM as a set of evidence values to compute the probability of the user visiting the other web pages. The web pages are then ranked based on their computed probabilities. If K_i is the number of items in the measurement set, R_i is the number of items on the recommendation list and M is the total number of test cases, the accuracy over the entire set is computed as:

$$cfaccuracy = \frac{100}{M} \sum_{i}^{M} \frac{\sum_{k}^{R_i} \delta_{i,k} h(k)}{\sum_{k}^{K_i} h(k)} \tag{5.3}$$

If the kth item on the ith recommendation list is in the measurement set then $\delta_{i,k} = 1$, otherwise it is

Algorithm	Given 2	Given 5	Given 10	All But 1
IPM	61.07	60.20	55.58	64.61
BN	59.95	59.84	53.92	66.69
CR+	60.64	57.89	51.47	63.59
VSIM	59.22	56.13	49.33	61.70
BC	57.03	54.83	47.83	59.42
NB	60.16	46.44	30.43	59.66
Baseline	49.14	46.91	41.14	49.77
RD	0.91	1.82	4.49	0.93

Table 5.4: Results for the MS Web data set. The higher the score the better the results. *RD* is the required difference between scores to be deemed statistically significant at the 90% confidence level.

equal to 0. The function $h(k)$ is defined as:

$$h(k) = 2^{\frac{-k}{a}} \tag{5.4}$$

where a can be viewed as the "half-life" of $h(k)$, that is $h(k)$ will equal 0.5 when $k = a$. We use a value of 5 for a.

As in [5], we tested our algorithm on four experiments. For the first three experiments we gave the network 2, 5 and 10 web pages from each test case and asked it to predict the remainder. For the fourth we gave the network all but 1 web page visited by the user and asked it to predict the final web page. For the experiments given 2, 5 and 10 web pages if fewer than that many pages were in the test case the test case wasn't used. At least 2 web pages needed to be present in the test case for use in the all but 1 experiment. Thus each experiment used a different number of test cases.

For comparison we've supplied results from 5 other algorithms: Bayesian Networks (BN), Correlation technique (CR+) that uses inverse user frequency, default voting and case amplification extensions, Vector Similarity (VSIM) method with an inverse user frequency transformation, Bayesian Clustering (BC) and Naive Bayes (NB). The results for BN, CR+, VSIM and BC are supplied by [5]. Naive Bayes was implemented by creating a Bayesian network for each hidden variable in which the hidden variable was the parent of the set of evidence variables. The same set of conditional probabilities was used for NB as with the IPM. The confidence coefficient r is set to 5. The baseline results are found using the prior probabilities for each web page, i. e. whatever web pages are most visited overall are always chosen regardless of the data given the network. The value *RD* is the required difference between two values to be deemed significantly different at the 90% confidence level [5].

Given 2	Given 5	Given 10
9,090	5,495	2,857

Table 5.5: Recommendations per second for the MS Web data set, given 2, 5 and 10 ratings. All tests are done on a 1 GHz Pentium running Windows 2000.

Complex Functions	Threshold	Given 2	Given 5	Given 10	All But 1
0	None	61.07	60.20	55.58	64.61
28	1000	61.30	60.37	55.34	64.68
326	100	61.34	60.14	54.19	64.54
647	50	61.37	60.07	53.46	64.52

Table 5.6: Results for the MS Web data set using different sets of complex functions.

The results in table 5.4 show the IPM producing better results in the given 2, 5 and 10 experiments. However, the results are not significantly better. In the all but 1 experiment the IPM outperforms all methods except BN.

While producing some of the most accurate results, the IPM is efficient. The IPM is capable of producing between 2,000 and 9,000 queries per second while taking only 5 seconds for learning on a 1GHz Pentium PC, table 5.5.

In our tests above the IPM didn't contain any complex functions. Table 5.6 displays the results of the IPM with several different sets of complex functions. We tested three sets using pairs of visited web pages. We added a new complex function to the IPM if a pair of web pages were visited more than a threshold of 50, 100 and 1,000 times within the training data set. The results aren't conclusively better or worse. All of the tests lie well within the confidence interval of each other. We believe the reason for this is that there aren't many high order dependencies in predicting web browsing behavior. The likelihood of a user visiting a web page is generally related to each page the user visited individually and not groups of pages. It may also be possible that the IPM didn't capture the higher order dependencies, i. e. the wrong pairs were chosen or more than two web pages need to be grouped at a time.

5.2.2 Movie Ratings

Our second set of tests involves a database of movie ratings. The database is from the EachMovie collaborative filtering site run by Digital Equipment Research Center from 1995 to 1997. For more information visit *http://research.compaq.com/SRC/eachmovie/*. Each user was asked to rank movies on a 0 to 5 scale. Out of a total of 1,623 movies each user ranked on average 46.4 movies with a median at 26. There were 4,119 total users in the test set and 5,000 users in the training set. Once

Algorithm	Given 2	Given 5	Given 10	All But 1
IPM	1.059	1.014	0.982	0.928
CR	1.257	1.139	1.069	0.994
BC	1.127	1.144	1.138	1.103
BN	1.143	1.154	1.139	1.066
VSIM	2.113	2.177	2.235	2.136
Baseline	1.106	1.105	1.103	1.133
RD	0.022	0.023	0.025	0.043

Table 5.7: Results for the EachMovie data set. Absolute deviation from the true user ratings. Lower scores indicate better results. *RD* is the required difference to be deemed statistically significant.

again we tested the algorithm on four tasks. The first three gave 2, 5 and 10 movie ratings to the network and the network was asked to predict the remaining ratings given by the user. Our forth task provided the network with all the movie ratings for a user except one and was asked to predict the rating of the final movie. In all cases the movie ratings given to the network were chosen randomly from the list of rated movies given by each user. In the given 2, 5 and 10 tasks if a user provided fewer than the respective number of ratings the user was not included in testing. The all but 1 task only included users with at least 2 ratings.

We computed errors based on the absolute deviation between the predicted rating and that given by the user. Once again we provide results from several algorithms provided courtesy of [5]. More details on the implementation of CR, BC, BN and VSIM can be found in [5]. The baseline results use the average rating given to the movie by the users.

For the IPM we need to compute the conditional probability matrix **P**. This is made more difficult since the movie ratings are on a scale from 0 to 5 and not binary. To transfer the ratings to a 0 to 1 scale we used the following equation:

$$\bar{P}(f_i|f_j) = \frac{\sum_k \min(f_i(t_k), f_j(t_k))}{\sum_k f_j(t_k)} \tag{5.5}$$

If $t_{k,i}$ is unknown for some i then training case k was not used to compute $\bar{P}(f_i|f_j)$ or $\bar{P}(f_j|f_i)$ for any j. The final movie ratings are computed by multiplying the computed probability estimates by 5. The confidence coefficient r was set to 100 and no complex functions are used.

Surprisingly, the baseline results outperform all the algorithms in the given 2 and 5 experiments except the IPM, table 5.7. Correlation (CR) does outperform the baseline results for the given 10 and all but 1 experiments. The IPM produces significantly better results in all of the experiments.

For learning, the IPM took less than a minute on a 1 GHz Pentium running Windows 2000.

Given 2	Given 5	Given 10
1,852	1,010	581

Table 5.8: Recommendations per second for the EachMovie data set, given 2, 5 and 10 ratings. All tests are done on a 1 GHz Pentium running Windows 2000.

The learning times for the probabilistic models from [5] took up to 8 hours for learning on a 266 MHz Pentium. The correlation based method (CR) was capable of generating 3.2 recommendations per second while the Bayesian network (BN) can generate 12.9 recommendation per second on a 266 MHz Pentium II. In contrast the IPM is capable of generating between 581 and 1,852 recommendations per second, table 5.8, on a 1 GHz PC.

The EachMovie data set contains approximately 74,000 users. To maintain consistency with [5] we only used 5,000 users for training. The first 4,119 users with more than 2 ratings were used for the testing set and the following 5,000 users with more than 2 ratings were used for the training set. We tested the results of the network across several different subsets of the user data and found comparable results with those shown above.

5.2.3 Serendipity

When recommending items such as movies we might want to consider more than just how high a user will rate the movie. If a user gives a high rating to the movie ”Star Wars: Return of the Jedi” they will most likely also give a high rating to ”Star Wars: The Empire Strikes Back.” Should the second Star Wars movie be recommended if we know the user has given a high rating to the first? It is true that the user will most likely enjoy the second movie. However, it is also very likely that the user already knows about the second Star Wars movie. A recommendation the user is already aware of is not of much value to the user. Ideally, we’d like to recommend movies to the user that they aren’t aware of and that they’d find worth watching, i. e. we’d like to increase the serendipity [2] [4] of the recommendations.

One possible method to increase the serendipity of the recommendation is to rank them not only on their rating but also on the probability that a user has rated the movie. If a user has rated a movie then they must be aware of it and less likely to be aware of it otherwise. In our movie database each user has only rated a subset of movies from the database. Thus we can create two networks, one to compute the predicted ratings of the movies and one to predict the probability that a user has rated the movies. The movie recommendations may then be ranked using:

$$y_i = y_i^{rating} y_i^{rated} \tag{5.6}$$

Where y_i^{rating} is the predicted rating for movie i and y_i^{rated} is the estimated probability of the user rating movie i.

5.3 Image Retrieval

Content-based image retrieval has received a large amount of attention over the last 10 years [18] [74]. As this technology matures and is adopted by more users, another source of information for image retrieval becomes available. When a user enters a keyword, draws a sample image or selects a query image a new list of images, typically thumbnails, is generated with the hope that the user will find them useful. As the user scans the generated list they will select some images while ignoring others. Since the user is looking for a certain type of image during a specific query, the selected images must be related in some manner. This relation could have several forms; a user could be interested in images that contain a specific object, relate to a certain topic, have a specific kind of texture or have any other conceivable relation. By analyzing the user selections, relations between images within the database can be discovered. After enough user data is accumulated, an algorithm for computing image relevance could be completely content-free. That is, it is conceivable that an image retrieval system could rely completely on user feedback without explicit knowledge about the actual appearance of the images.

Several systems use relevance feedback from users to refine their searches [29, 71, 77, 78, 84]. Most of these systems only apply user feedback to the current query. Recently, some research has been done in long-term learning from user interactions [29].

To demonstrate the RLN on a large image database we obtained 9,900 images from [51, 83]. Along with the images, we selected 80 keywords such as ”Butterflies, Mountains, Autumn, Children, Planets, Bridge, Colorful Texture, etc.” Several users were asked to select images they thought were similar along with any keywords they thought were relevant. In total, approximately 1,600 entries were made. In a real world system, used by thousands of users, a much larger amount of input data would be available.

The keywords and images were treated identically with each assigned a variable. No complex feature functions were used, resulting in 9,981 feature functions. Training of the RLN takes approximately 10 minutes. All running times are generated on a 1GHz PC, Pentium 3. The computed weight matrix was sparse with only 1,147,680 non-zero weights out of a possible 99,800,100.

To decrease the computation time needed for each query we made the following optimization. The values of y_i^{j+1} were computed using only y_i^j that had values greater than a threshold. For our

experiments we set the threshold at 0.01. This optimization has a minimal effect on the recommendation ordering of the images.

On average, between 5 to 20 queries can be made per second using the RLN. The queries typically had between 5 and 50 user labelled images or keywords. Since the keywords were treated identically to the images, keywords are also suggested to the user given their preferences for images and other keywords. For example if image (a) from figure 5.7 was labelled as desired, the RLN would suggest keywords "Space, People and Space Ships."

We also tested the results of the IPM on the database. Given our training data there are 1,169,527 non-zero pairwise probabilities taking about 30 seconds to compute. 40 to 500 queries can be made per second. This is about one order of magnitude faster than computing y_i using the RLN.

Figures 5.6 and 5.7 show some sample queries using the system. In query 1, three images are in X_E with values equal to one. The images show sunsets next to the ocean with two of them containing people. The RLN returns images of ocean sunsets. In contrast, query 2 contains images with people and two of them are ocean scenes. The RLN in this case returns images with people in sunsets.

Query 3 contains a single image of an astronaut with the space shuttle. Varying images of astronauts and space shuttles are returned. In query 4 the value of an astronaut image is set to zero, i. e. undesirable. The RLN then returns space shuttle images.

It is important to remember that the results of the RLN are based heavily on the quality of the input training data. These results are merely meant to show the potential of the system.

Figure 5.6: Query 1 and Query 2: Top row contains evidence images followed by the six most relevant images as predicted by the RLN.

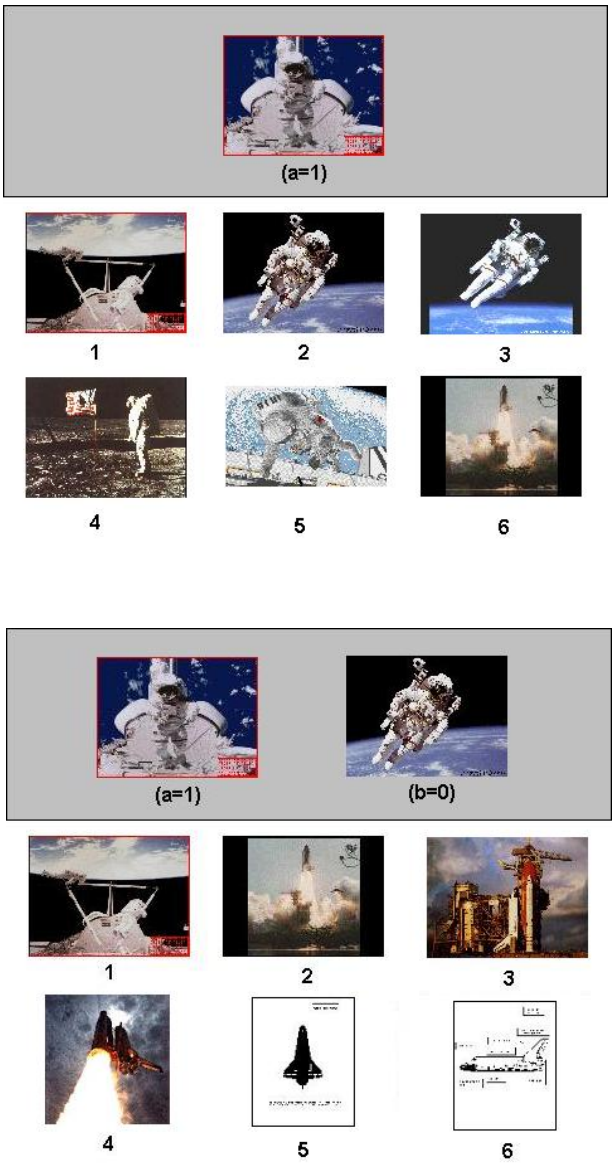
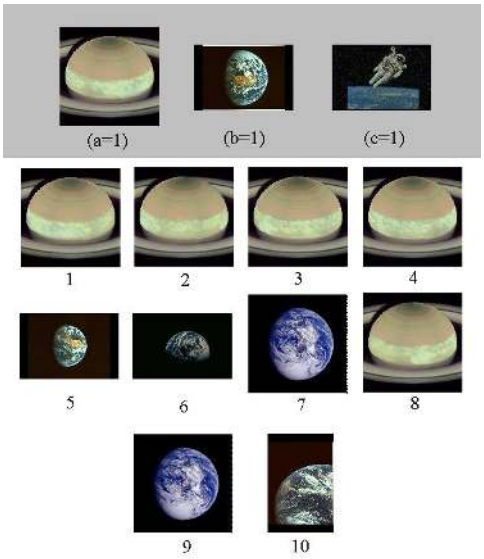


Figure 5.7: Query 3 and Query 4: Top row contains evidence images followed by the six most relevant images as predicted by the RLN.

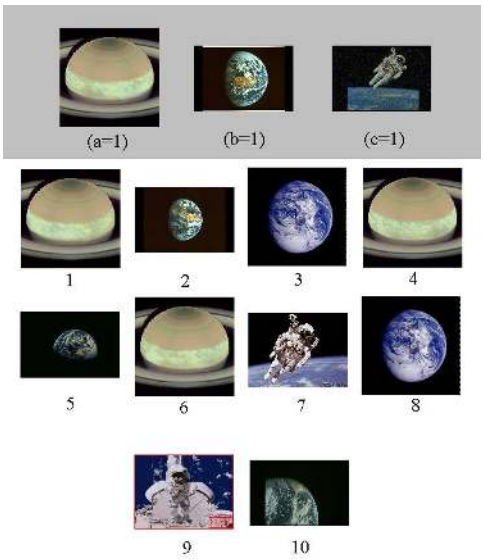
5.3.1 Providing Similar and Unique Images

Until this point, we've assumed that the images that are most similar to the query images will be of greatest interest to the user. At first this may appear to be true, but for many cases the user might not want the most similar images, but images that are just related. For example consider figure 5.8 (a). Three images are selected by the user; a picture of Saturn, the Earth and an astronaut. Within the database, there are several Saturn pictures that are almost exactly the same as the Saturn picture in the query. As a result, these Saturn pictures are returned as the top recommendations. Does the user really want images of Saturn that look almost exactly the same? Within the top ten recommendations there doesn't exist a single image of an astronaut, due to so many similar images of Saturn and the Earth.

Let us examine an alternative approach for recommending images. For our first recommendation we will return the most similar image, as we did before. For the second image, we will return the most similar image to those in the query while assuming that the user doesn't like the first recommended image. Thus, to find the second image we can add the first recommended image to the evidence images with a value of zero and then recompute the recommendations. Similarly, for each following recommendation we assume the user doesn't like the proceeding recommendations. Using this method we obtain the results in 5.8 (b). The results using this method still recommend similar images. However, more unique results are also returned. After the first Saturn image is recommended its value is set to zero, thus the values of the other Saturn images are reduced. This results in the Saturn images being pushed further down the recommendation list.



(a)



(b)

Figure 5.8: User recommendations using (a) the most similar images and (b) the most similar images assuming the user doesn't like the previous recommendations.

5.3.2 Overcoming the Cold Start Problem using CBIR

A common problem for collaborative filtering algorithms such as ours is the ”cold start” problem [52, 70]. That is, our algorithm needs a large amount of training data from users before it can start producing good results. Unfortunately, users won't use the system until it produces good results. We see two methods for solving this problem. First, content or keyword based image retrieval systems could find sets of similar images that are used as seeds to initially train the system. Second, results could be initially found using the content or keyword image retrieval system and as more user data becomes available the collaborative filtering algorithm could be used. In this way, we believe a hybrid system that combines the strengths of several systems will prove to be most beneficial.

5.4 Language Modeling

Language modeling is the task of predicting the next word in a sentence or utterance given the set of preceding words. Traditionally, probabilities are computed using a trigram model which only considers the previous two words. Within the trigram model probabilities are computed directly from training data. The training data sets are typically quite large with some having over 50 million words with vocabularies of around 20,000.

The main difficulty in implementing the trigram model is that all pairs of words that may exist in the testing data set may not appear in the training data set. A standard method for dealing with these cases is to use a bigram model, i.e. only use the previous word, when there doesn't exist enough data to compute the probabilities based on the trigram model. When their doesn't exist enough data for the bigram model then a unigram model is used, i.e. compute probabilities based solely on how often the words appear in the training data set. This method is referred to as the back-off method [41]. Other methods include combining the trigram, bigram and unigram models using linear interpolation [15, 36]. The trigram model performs surprisingly well, and is used in many speech recognition algorithms. Obtaining results that significantly outperform the model is difficult [10, 69].

Maximum entropy was first applied to language modeling by [1, 62] and later using word triggers by [46, 68]. Maximum entropy is well suited for language modeling since the complete probability distribution, i.e. the trigram model, can not be computed directly for the data. Maximum entropy can combine the information from the trigram, bigram and unigram models efficiently and intelligently. Rosenfeld [68] added another source of information to the model called triggers. Triggers are words that have occurred in the relatively recent past, such as in the last fifty words. For

unigram	$P(w_3)$
bigram	$P(w_3\|w_2)$
distance two bigram	$P(w_3\|w_1)$
trigram	$P(w_3\|w_1, w_2)$

Table 5.9: Different types of N-gram models. The words occur in the following order: w_1, w_2 and w_3.

example, if we see the word "stock" it is more likely that we'll see the word "shares" in the next several sentences. Using maximum entropy, Rosenfeld was able to easily incorporate this information into the model and obtain improved results.

The previous methods using maximum entropy for language modeling have all used Shannon's measure. We'll attempt to use URQE and explore some methods for adding additional complex functions.

Our experiments will use a combination of trigrams, bigrams, distance two bigrams and unigrams. A distance two bigram is computing the probability of the current word given the word that occurred two words before it. If we have two words w_1, w_2 that occur in order and we want to compute the probability of w_3, then our four sources of information are shown in table 5.9.

For each model and set of words, we can add a constraint to our algorithm. Each constraint will have a different amount of confidence based on how often its corresponding feature function occurs within the training data set.

Our first test uses only bigrams, distance two bigrams and unigrams. We train an RLN using a 5 million word training data base with a vocabulary of 19,980 words, see Appendix B. Thus our set of inputs consists of 39,961 feature functions. One for each bigram and distance two bigram, along with the bias feature function for the unigram model. Since the number of evidence feature functions is so large, we use an RLN to compute the probability estimates instead of the IPM. To minimize the size of the network, the weights $\omega_{j,i}$ are pruned based on the weight's utility:

$$P(f_i)(1.0 - P(f_i))\left|\omega_{j,i}\right| \tag{5.7}$$

If the utility of the weight was below the threshold of 10^{-7}, its value was set to zero. If the weights weren't pruned the total number of weights would be $39,961 \times 19,980 = 798,420,780$. After pruning, as shown in tables 5.10 and 5.12, between 0.3% and 0.6% of the weights remain. The value of the drag coefficient τ was set to 10^{-9} with $r = 10^{-5}$. The accuracy of the results are measured using perplexity. Perplexity measures the average branching factor for the language model and is

Algorithm	Perplexity	Number of Weights	Weight Magnitude	#Ands
Bigrams	252	2,586,537	12,878	0
Trigram	211	12,307,839	51,244	336,268

Table 5.10: Language modeling results for the RLN when using bigram and trigram models.

computed from [9]:

$$2^{\frac{1}{m_{test}} \sum \log_2(y_{correct})} \tag{5.8}$$

Where we sum over the entire testing data set and $y_{correct}$ is the computed probability estimate for the correct word. The number of words in the testing data set is equal to m_{test} = 325,196. The lower the perplexity of the language model, the more accurate it is. To ensure our probability estimates actually correspond to a true distribution we limit the minimum of any estimate to be 10^{-5} and then divide all of the estimates by their combined sum.

The results for the RLN bigrams model using only bigram, distance two bigram and unigram constraints is shown in table 5.10. The weight magnitude is the total absolute magnitude of all the weights in the network. The total number of weights corresponds to the number of non-zero weights. The perplexity of the bigrams model is 252.

Our second test uses trigram constraints along with the constraints used in the previous bigrams test. A new And function was created for each pair of words that occurred together at least 2 times in the training data for a total of 336,268 new And functions. Otherwise, the network was trained using the same parameters. The resulting perplexity for the RLN trigram was 211 as shown in table 5.10. The back-off method on the same data set is capable of obtaining a perplexity of 173. There are several optimizations used in the back-off model that we didn't use in our method. Another reason for the larger error may be the RLN minimizes the squared error. Thus the difference between 0.5 and 0.51 is the same as the difference between 0.001 and 0.011. Since the perplexity measure is log based, the difference between 0.001 and 0.011 is much greater than the difference between 0.5 and 0.51. Therefore, the RLN may be minimizing the wrong measure for this problem domain.

The learning times for the RLN were 16 hours for the bigrams network and 82 hours for the trigram network. We implemented a simple iterative method for weight learning. Learning times could be reduced by using a more complex algorithm, such as biconjugate gradient or methods that incorporate momentum.

5.4.1 Grouping Words

In the previous section we created new constraints for each pair of words that occurred at least twice in the training data set. As our vocabulary and size of the training data base increase, the number of constraints can get quite large. Many of these constraints can be redundant. For example the word pairs ”three dollars” and ”four dollars” will probably have similar words following them. The words ”three” and ”four” have similar meaning and even though they are different the words following them will be very similar.

Most words have other words that are very similar to them, such as: ”a” and ”the”, ”Harvard” and ”Yale”, ”dropped” and ”gained”, and ”may” and ”can.” If our network can group these words together before creating the word pairs, many fewer constraints can be used. For example, instead or having two constraints for ”three dollars” and ”four dollars”, we have one constraint corresponding to ”three” or ”four” followed by ”dollars.”

In section 4.2.3 we described a method for grouping similar feature functions together. This is done for two feature functions f_j and f_k by computing the dot product of their normalized weights:

$$\frac{\sum_l \omega_{l,j} \cdot \omega_{l,k}}{\sqrt{\sum_l \omega_{l,j}^2} \sqrt{\sum_l \omega_{l,k}^2}} \tag{5.9}$$

This equation computes the cosine of the angle between the weights of the feature functions. The closer the value is to one the more similar the weights. If we group words with weights less than 45 degree apart we find the groupings in table 5.11. All words with less than 10 non-zero weights were not considered.

THE	FROM	SAYS
A	POUNDS	CONSULTANT
AN	EARNED	TOWN
	TOTALED	MESSRS.
End of Sentence		EXPLAINS
THAT	MILLION	Q.
BUT	BILLION	COMPLAINS
WHILE	TRILLION	JEAN
		DECLARES
TO	HE	ACTOR
WILL	ANALYSTS	ERNEST
WOULD	SHE	GOODMAN
COULD	POLITICAL	PRONOUNCED
HELP	SOURCES	BOASTS
HELPED	WEREN'T	INGREDIENT
	COMING	EDGAR
ONE	QUITE	SHIPYARD
TWO	PEACE	HANS
FIVE	CONSULTING	CLIFFORD
THREE	SYMBOL	
SEVEN	SEASONAL	THOUSAND
SIX	PERSPECTIVE	TRANSPORTATION
FOUR	SHAREHOLDERS'	UTILITIES
EIGHT	FORECASTING	FRANCS
NINE	COPYRIGHT	POOR'S
ELEVEN	TERRIBLE	FORTUNE
THIRTEEN	UNILATERAL	LOSERS
	ET	EQUALING
IS	DISTANT	PROPOSITION
ARE	CHARITABLE	RECALLING
IT'S	POSTWAR	
ISN'T	PETROCHEMICAL	THEIR
TOO	ANYTIME	OWN
ENOUGH	GIBBS	MILITARY
ASKED	EVERYDAY	FACE
WE'RE	NAUTILUS	UNDISCLOSED
EXPECT		
ALLOW	BE	UP
REQUIRE	WAS	DOWN
URGED	BEEN	ROSE
ALLOWING	WERE	OFF
ALLOWS	BEING	FELL
APPLY	AREN'T	DAYS
ENCOURAGE	WASN'T	AVERAGING
REQUIRING		YIELDED
PERMIT	HAS	
I'D	HAVE	SOME
SUFFICIENT	HAD	MOST
GENEVA		MANY
ENABLE	COMPANY	
YOU'VE	INCORPORATED	PRESIDENT
AIM	CORPORATION	CHAIRMAN
DECIDING		COMMANDER
FIDUCIARY	TWENTY	
	THIRTY	MAY
AS	FIFTY	CAN
DESCRIBE	FORTY	SHOULD
REOPEN	SEVENTY	MIGHT
BUYBACK	SIXTY	
PORTRAYED	NINETY	YORK
		AMERICAN
MR.	ABOUT	SECURITIES
MS.	ROUGHLY	TORONTO
DOCTOR	APPROXIMATELY	LONDON'S
DON'T	MORE	SALES
DIDN'T	LESS	REVENUE
DOESN'T	FEWER	LOSS
WON'T		

Table 5.11: A sampling of the 111 word groups.

Algorithm	Perplexity	Number of Weights	Weight Magnitude	#Ands	#Ors
None	252	2,586,537	12,878	0	0
Or	253	2,840,565	12,726	0	222
And	222	3,653,641	21,320	10,832	0
And Or	220	3,771,004	20,521	10,242	222
And only Or	244	2,791,659	13,442	724	222

Table 5.12: Language modeling results for the RLN when adding And and Or complex functions.

If we only add complex Or functions corresponding to the groups found, the results of the network will not change since only a single word has a value of one at anytime for each position in time. To illustrate this point, consider three feature functions corresponding to ”three”, ”four” and ”three or four” at some particular time instance. If the weights from our feature functions to the output y_i are 0.3, 0.4 and 0.2 respectively, then we can achieve the same results with ”three” having a weight of 0.3 + 0.2 = 0.5 and ”four” having a weight of 0.4 + 0.2 = 0.6. Solely adding the Or complex functions may decrease the overall magnitude of the weights but it will not increase the accuracy of the network as shown in table 5.12. In general for other types of problems, adding Or functions will increase the accuracy. The only case when Or functions will not increase accuracy is when the variables that are being grouped together occur exclusively.

The accuracy of the network is increased by adding And complex functions. We have run three experiments using And functions. The first called ”And” doesn't use any Or functions. The second called ”And Or” adds And functions that combine single words and word groupings. The final test called ”And only Or” adds And functions that only combine word groupings. Each test combined words or groups if the pair occurred at least 50 times in the training data set. The ”And Or” test produced the best results, but only by a small amount over the ”And” results. The weight magnitude for the ”And Or” test was slightly less but the total number of weights was greater than the ”And” test. The ”And only Or” test has many fewer weights and feature functions but also produces worse results.

These results show little improvement when adding Or complex functions. However, we believe that Or complex functions will become a computational necessity when more than two words are used for prediction. If we increase the number of words used for prediction to just 4, the number of And complex functions needed may increase beyond what is computationally feasible to handle. Thus Or complex functions may provide a method for increasing the number of words used in prediction while still allowing the algorithm to be computationally feasible.

Chapter 6

Conclusion

When computing conditional probabilities in large domains, that is problems with hundreds if not thousands of variables, low-order interactions may not just be the only interactions supported by the data, but they can produce accurate results. The training data will typically support only a limited number of constraints, with most involving sets of just a few variables. Given these constraints, which don't fully constrain the distribution, we must compute estimates for the conditional probabilities. To make assumptions or constraints on the distribution other than those supported by the data will lead to results with high bias. If we impose too many constraints that are only partially supported by the data, the results may possess high variance.

To balance the tradeoffs between bias and variance we've described a method based on maximum entropy. Maximum entropy attempts to find the distribution that makes the least amount of assumptions other than those given by the constraints, resulting in low bias. Variance is reduced by assigning each constraint a confidence value. If a constraint has a high variance itself, then it will be assigned a low confidence value and have less effect on the final probabilities computed. This results in the estimated conditional probabilities having themselves lower variance.

6.1 Contributions

The main emphasis of this dissertation were two algorithms, the RLN and the IPM, for computing approximations to the maximum entropy distribution. While the results produced using these two algorithms are identical, the efficiency of the algorithms varies depending on the problem domain.

The main points of the dissertation include:

An Efficient Approximation to Maximum Entropy - Rényi developed a family of entropy

measures by generalizing the properties of entropy first proposed by Shannon. Within this family lies an entropy measure called Rényi's quadratic entropy. If we ignore the constraints that all probabilities must lie between 0 and 1, we may maximize this measure relative to our constraints using a set of linear functions. Maximizing this measure is equivalent to minimizing the squared probabilities.

Computing Conditional Probabilities using URQE - Using the unbounded Rényi quadratic entropy is largely inaccurate for computing estimates to the joint distribution. However, when computing the conditional probability of each variable given the set of evidence variables, URQE produces accurate results similar to those using Shannon's measure.

RLN and IPM - We proposed two methods for finding estimates of conditional probabilities, the recurrent linear network and the inverse probability method. The recurrent linear network is an iterative approach that is most efficient when many variables have known values. The inverse probability method is a closed-form solution that is more efficient when the number of evidence variables is small.

Learning Efficiency - The parameters for either method can be learned quickly using a matrix of pairwise probability values generated from the data. This matrix of probability values can be quickly updated given new data.

Constraint Confidence - Each constraint has a corresponding confidence value that controls the degree to which the constraint affects the final outcome. The confidence values are based on the estimated variance of the constraint values. Thus new constraints can be added to the algorithm without risking an increase in error due to variance.

Experimental Results - We demonstrated the algorithms on several applications including: collaborative filtering, image retrieval and language modeling. The results for collaborative filtering were at least as accurate if not more than the other algorithms tested, while being one or two orders of magnitude more efficient. We demonstrated the algorithm is capable of handling large domains, i.e. with over 10,000 variables, within the image retrieval and language modeling applications.

6.2 Future Work

While the experimental results for our approach appear promising, more testing needs to be done. Larger databases are needed to explore higher-order interactions between variables. Databases such as the web log from Microsoft and the movie ratings from Compaq only allow us to accurately compute a small number of constraint values for complex functions. The results using the synthetic data showed our method outperforms others across varying training data sizes, but we need large

real world databases to support this claim. Given these large databases, is it possible to find a set of complex functions that increases the accuracy of the network while not overly decreasing the efficiency?

Even if our results are better given the comparison criteria, this does not mean the user experience will be improved. We might be able to produce better movie ratings, but are we recommending movies for which the user is unaware and will like? For the image retrieval task, we may be able to compute the relationship between images, but does this really help the user find their desired image faster? Similarly, it has been shown for the language modeling task that reducing the perplexity does not always lead to a reduction in word error rate in speech recognizers [9]. To answer these questions more tests and user studies need to be done.

Within this dissertation we've discussed finding the conditional probability of each variable individually. For many applications the most likely state for several variables might be desired. For example we might like to find the maximum a posteriori or the maximum expected utility values for some set of variables. We have not investigated beyond simple greedy search algorithms how this may be done in relation to the RLN or IPM. Since estimated probabilities can be generated efficiently using our methods we believe there may also be promise in computing maximum a posteriori estimates.

Bibliography

[1] Adam L. Berger, Stephen A. Della Pietra, and Vincent J. Della Pietra. A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71, 1996.

[2] P. Bieganski. System, method and article of manufacture for increasing the user value of recommendations, July 1998. U.S. Patent Number 6,321,221.

[3] Daniel Billsus and Michael J. Pazzani. Learning collaborative information filters. In *Proc. 15th International Conf. on Machine Learning*, pages 46–54. Morgan Kaufmann, San Francisco, CA, 1998.

[4] J. Breese, D. Heckerman, E. Horvitz, C. Kadie, and K. Kanazawa. Methods and apparatus for retrieving and/or processing retrieved information as a function of a user's estimated knowledge, February 1997. U.S. Patent Number 6,006,218.

[5] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52, San Francisco, July 24–26 1998. Morgan Kaufmann.

[6] L. M. Bregman. The relaxation method to find the common point of convex sets and its applications to the solution of problems in convex programming. *USSR Computational Mathematics and Mathematical Physics*, 7:200–217, 1967.

[7] G. W. Brier. Verification of forecasts expressed in terms of probabilities. *Monthly Weather Review*, 78:1–3, 1950.

[8] J. P. Burg. Maximum Entropy Spectral Analysis, October 1967.

[9] S. Chen, D. Beeferman, and R. Rosenfeld. Evaluation metrics for language models. In *DARPA Broadcast News Transcription and Understanding Workshop*, 1998.

[10] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 310–318, San Francisco, 1996. Morgan Kaufmann Publishers.

[11] W.S. Cooper. Exploiting the maximum entropy principle to increase retrieval effectiveness. *Journal of the American Society for Information Science*, 34(1):31–39, 1983.

[12] W.S. Cooper and P. Huizinga. The maximum entropy principle and its application to the design of probabilistic retrieval systems. *Information Technology, Research and Development*, 1:99–112, 1982.

[13] I. Csiszár. A geometric interpretation of darroch and ratcliff's generalized interative scaling. *The Annals of Statistics*, 17(3):1409–1413, 1989.

[14] J. N. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Mathematical Statistics*, 43:1470–1480, 1972.

[15] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society series B*, 39:1–38, 1977.

[16] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[17] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley and Sons, New York, 1973.

[18] M. S. Lew (Ed.). *Principles of Visual Information Retrieval*. Springer-Verlag, New York, 2001.

[19] D. Erdogmus and J. C. Principe. Comparision of entropy and mean square error criteria in adaptive system training using higher order statistics. In P. Pajunen and J. Karhunen, editors, *Proceedings of the Second International Workshop on Independent Component Analysis and Blind Signal Separation*, pages 75–80, Helsinki, Finland, 2000.

[20] D. Erdogmus and J. C. Principe. Principe entropy minimization algorithm for multilayer perceptrons. In *Proceedings of the International Joint Conference on Neural Networks*, Washington, D.C., 2001.

[21] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming*. Wiley, 1968.

[22] Jerome H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *J. Data Mining and Knowledge Discovery*, 1(1):55–77, April 1997.

[23] A. Golan, G. Judge, and D. Miller. *Maximum Entropy Econometrics: Robust Estimation with Limited Data*. John, Wiley and Sons, New York, 1996.

[24] A. Golan and J. Perloff. Comparison of maximum entropy and higher-order entropy estimators. *J. Stat. Physics*, 107(1), 2002.

[25] Warren R. Greiff and Jay M. Ponte. The maximum entropy approach and probabilistic IR models. *ACM Trans. on Information Systems*, 18(3):246–287, 2000.

[26] P.D. Grnwald and A.P. Dawid. Game theory, maximum entropy, minimum discrepancy, and robust bayesian decision theory. Technical Report 223, University College London, London, 2002.

[27] P.D. Grnwald and A.P. Dawid. Game theory, maximum generalized entropy, minimum discrepancy, robust bayes and pythagoras. In *Proceedings of ITW*, Bangalore, India, October 2002.

[28] S. F. Gull and G. J. Daniell. Image reconstruction from incomplete and noisy data. *Nature*, 272:686–690, April 1978.

[29] X. He, W. Ma, O. King, M. Li, and H. Zhang. Learning and inferring a semantic space from users relevance feedback for image retrieval. Technical Report MSR-TR-2002-62, Microsoft Research, Redmond, Washington, 2002.

[30] D. Heckerman, D. Geiger, and M. Chickering. Learning Bayesian networks: The combination of knowledge and statistical data. Technical Report MSR-TR-94-09, Microsoft Research, Redmond, Washington, 1994.

[31] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and arid Carl Kadie. Dependency networks for collaborative filtering and data visualization. In Craig Boutilier and Moisés Goldszmidt, editors, *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-00)*, pages 264–273, SF, CA, June 2000. Morgan Kaufmann Publishers.

[32] E. Jaynes. Notes on present status and future prospects, 1990.

[33] E. T. Jaynes. Information theory and statistical mechanics. *Physical Review*, 106:620–30, 1957. continued in volume 108, pages = 171-190.

[34] E. T. Jaynes. Where do we stand on maximum entropy inference. In R. D. Levine and M. Tribus, editors, *The Maximum Entropy Formalism*. 1978.

[35] E. T. Jaynes. On the rationale of maximum-entropy methods. *Proc. IEEE*, 70(9):939–952, 1982.

[36] F. Jelinek and R. Mercer. *Interpolated estimation of Markov source parameters from sparse data*, pages 381–397. Amsterdam : North Holland Publishing Co., 1980.

[37] C. Kadie, C. Meek, and D. Heckerman. Cfw: A collaborative filtering systen using posteriors over weights of evidence. Technical Report MSR-TR-2002-46, Microsoft Research, Redmond, Washington, 2002.

[38] Paul B. Kantor. Maximum entropy and the optimal design of automated information retrieval systems. *Information Technology, Research and Development*, 3(2):88–94, 1984.

[39] Paul B. Kantor and Jung Jin Lee. The maximum entropy principle in information retrieval. In *Proceedings of the Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Probabilistic Retrieval, pages 269–274, 1986.

[40] Paul B. Kantor and Jung Jin Lee. Testing the maximum entropy principle for information retrieval. *Computational Complexity*, 49(6):557–566, 1998.

[41] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, March 1987.

[42] Jin H. Kim and Judea Pearl. A computational model for combined causal and diagnostic reasoning in inference systems. In *Proceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83)*, pages 190–193, Karlsruhe, Germany, August 1983. Morgan Kaufmann.

[43] I. Kononenko. Semi-naive bayesian classifier. In Y. Kodratoff, editor, *Proceedings of the European Working Session on Learning : Machine Learning (EWSL-91)*, volume 482 of *LNAI*, pages 206–219, Porto, Portugal, March 1991. Springer Verlag.

[44] S. Kullback. *Information Theory and Statistics.* Wiley, 1959.

[45] J. Lafferty, S. Della Pietra, and V. Della Pietra. Statistical learning algorithms based on bregman distances. In *Proceedings of the 1997 Canadian Workshop on Information Theory*, Toronto, Canada, 1997.

[46] R. Lau, R. Rosenfeld, and S. Roukos. Trigger-based language models: A maximum entropy approach. In *Proc. ICASSP '93*, pages II–45–II–48, Minneapolis, MN, April 1993.

[47] R. Lau, R. Rosenfeld, and S. Roukos. Adaptive language modeling using the maximum entropy principle. In *Proc. ARPA Human Language Technology Workshop '93*, pages 108–113, Princeton, NJ, March 1994.

[48] R. Lau, R. Rosenfeld, and S. Roukos. Adaptive language modeling using the maximum entropy principle. In *Proc. ARPA Human Language Technology Workshop '93*, pages 108–113, Princeton, NJ, March 1994. distributed as *Human Language Technology* by San Mateo, CA: Morgan Kaufmann Publishers.

[49] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–224, 1988.

[50] Wee Sun Lee. Collaborative learning for recommender systems. In *Proc. 18th International Conf. on Machine Learning*, pages 314–321. Morgan Kaufmann, San Francisco, CA, 2001.

[51] J. Li, J. Wang, and G. Wiederhold. Irm: Integrated region matching for image retrieval. In *Proc. ACM Int. Conf. on Multimedia*, pages 147–156, October 2000.

[52] P. Melville, R. Mooney, and R. Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Proc. Conf. on Artificial Intelligence (AAAI-2002)*, pages 187–192, July 2002.

[53] H. Nguyen, O. Kosheleva, and V. Kreinovich. Invariance-based justification of the maximum entropy method and of generalized maximum entropy methods in data processing, 2000.

[54] J. Paris. Common sense and maximum entropy. *Synthese*, 117 (1):75–93, 1999.

[55] J. Paris and A. Vencovshá. On the Applicability of Maximum Entropy to Inexact Reasoning. *International Journal of Approximate Reasoning*, 3:1–34, 1989.

[56] J. Paris and A. Vencovshá. In Defense of the Maximum Entropy Inference Process. *International Journal of Approximate Reasoning*, 17:77–103, 1997.

[57] J. B. Paris and A. Vencovská. A note on the inevitability of maximum entropy. *International Journal of Approximate Reasoning*, 4(3):183–224, 1990.

[58] D. Paul and J. Baker. The design for the wall street journal-based csr corpus. In *Proceedings of the DARPA SLS Workshop*, 1992.

[59] Michael Pazzani. Searching for attribute dependencies in Bayesian classifiers. In D. Fisher and H. Lenz, editors, *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 424–429, Ft.Lauderdale, FL., 1995.

[60] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In David Waltz, editor, *Proceedings of the National Conference on Artificial Intelligence*, pages 133–136, Pittsburgh, PA, August 1982. AAAI Press.

[61] Judea Pearl and Tom S. Verma. A theory of inferred causation. In Dag Prawitz, Brian Skyrms, and Dag Westerstahl, editors, *Logic, Methodology and Philosophy of Science IX*, pages 789–811, Amsterdam, 1994. Elsevier Science Publishers.

[62] S. Della Pietra, V. Della Pietra, R. Mercer, and S. Roukos. Adaptive language modeling based using minimum discriminant estimation. In *Proc. ICASSP '92*, pages 633–636, San Francisco, CA, March 1992.

[63] Stephen Della Pietra, Vincent Della Pietra, and John Lafferty. Inducing features of random fields. Technical Report CS-95-144, Carnegie Mellon University, School of Computer Science, May 1995.

[64] A. Rényi. On measures of entropy and information. *Selected Papers of Alfred Rényi*, 2(180):565–580, 1976.

[65] A. Rényi. On the foundations of information theory. *Selected Papers of Alfred Rényi*, 3(242):304–318, 1976.

[66] A. Rényi. Some fundamental questions of information theory. *Selected Papers of Alfred Rényi*, 2(174):526–552, 1976.

[67] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pages 175–186, Chapel Hill, North Carolina, 1994. ACM.

[68] R. Rosenfeld. *Adaptive Statistical Language Modeling: A Maximum Entropy Approach*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, April 1994. also appears as technical report CMU-CS-94-138.

[69] R. Rosenfeld. Two decades of statistical language modeling: Where do we go from here. *Proceedings of the IEEE*, 88(8), 2000.

[70] Andrew I. Schein, Alexandrin Popescul, and Lyle H. Ungar. Methods and metrics for cold-start recommendations. In *Proceedings of the 25'th annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2002.

[71] C. Schmid. Constructing models for content-based image retrieval. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[72] T. Seidenfeld. Why i am not an objective bayesian; some reflections prompted by rosenkrantz. *Theory and Decision*, 11:413–440, 1979.

[73] Claude E. Shannon and Warren Weaver. *The Mathematical Theory of Communication*. Univ of Illinois Pr., 1963. ISBN: 0–252–72548–4.

[74] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[75] D. Spiegelhalter, P. Dawid, S. Lauritzen, and R. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8:219–282, 1993.

[76] Bo Thiesson, Christopher Meek, David Maxwell Chickering, and David Heckerman. Learning mixtures of DAG models. In Gregory F. Cooper and Serafín Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 504–513, San Francisco, July 24–26 1998. Morgan Kaufmann.

[77] K. Tieu and P. Viola. Boosting image retrieval. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 228–235, 2000.

[78] S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *Proc. ACM Int. Conf. on Multimedia*, pages 107–118, October 2001.

[79] M. Tribus. *Rational Descriptions, Decisions and Designs*. Pergamon Press, New York, 1969.

[80] C. Tsallis. Possible generalization of boltzmann-gibbs statistics. *Statistics*, 52:479–487, 1988.

[81] J. Uffink. Can the maximum entropy principle be explained as a consistency requirement. *Studies in History and Philosophy of Modern Physics*, 26:223–261, 1995.

[82] J. Uffink. The constraint rule of the maximum entropy principle. *Studies in History and Philosophy of Modern Physics*, 27(1):47–79, 1996.

[83] J. Wang, J. Li, and G. Wiederhold. Simplicity: Semantics-sensitive integrated matching for picture libraries. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(9):947–963, 2001.

[84] Y. Wu, Q. Tian, and T. S. Huang. Discriminant-em algorithm with application to image retrieval. In *IEEE Conf. Computer Vision and Pattern Recognition*, pages 222–227, 2000.

[85] W. J. Zangwill. *Nonlinear Programming: A Unified Approach*. Prentice-Hall, 1969.

Appendix A

A.1 Notation

$X = \{X_1, \ldots, X_a\}$	Set of variables
X_E	Set of evidence variables
X_H	Set of hidden variables
$x = \{x_1, \ldots, x_a\}$	Set of variable values
x_E	Set of evidence variable values
x_H	Set of hidden variable values
a	Number of variables
$P(X_i)$	$P(X_i = 1)$
$P(x_i)$	$P(X_i = x_i)$
$P(f_i)$	$P(f_i = 1)$
$P^* = \{p_1, \ldots, p_n\}$	True underlying probability distribution
$\bar{P}$	Observed probability distribution
P_1	Probability distribution computed using Shannon's measure of entropy
P_2	Probability distribution computed using Rényi's quadratic entropy
n	Number of entries in the joint probability distribution
$T = \{t_1, \ldots, t_m\}$	Set of training data
$t_{j,i}$	The value of the ith variable in the jth entry of the training data
$F = \{f_0, \ldots, f_{b-1}\}$	Set of all feature functions
F_E	Set of all evidence feature functions
F_H	Set of all hidden feature functions
b	Number of feature functions
e	Number of evidence feature functions
h	Number of hidden feature functions

$H(P)$ or $H_1(P)$	Entropy of the probability distribution P using Shannon's measure
$H_\alpha(P)$	Entropy of the probability distribution P using Rényi's measure α
$H_2(P)$	Entropy of the probability distribution P using Rényi's quadratic entropy
μ	Weights used to compute P_1 for a specific set of evidence variables X_E
λ	Weights used to compute the URQE solution for a specific set of evidence variables X_E
ω	Weights used to compute the URQE solution for any set X_E
$\mathbf{P}_{i,j} = \bar{P}(f_i \mid f_j)$	Pairwise conditional probability matrix for the functions F

A.2 Abbreviations

BN	Bayesian Network
CF	Collaborative Filtering
IPM	Inverse Probability Method
LS	Least Squares
NB	Naive Bayes
NN	Nearest Neighbor
RLN	Recurrent Linear Network
RQE	Rényi's Quadratic Entropy
SVD	Singular Value Decomposition
URQE	Unbounded Rényi Quadratic Entropy

Appendix B

B.1 Language Modeling Database

Our language database consist of the ARPA CSR Wall Street Journal corpus. Articles within the database were published in the Wall Street Journal from December 1986 through November 1989. We used a non-verbalized-punctuation version of the corpus. That is, we assumed all punctuation was not verbalized and was removed from the data. The data was further processed by [58] to clean up the data and make it suitable for language modeling.

For training we used the following subsets of the data:

Number of words total: 4,949,093

Year 1987	Year 1988
W7_001	W8_001
W7_002	W8_002
W7_003	W8_003
W7_004	W8_004
W7_006	W8_006
W7_007	W8_007
W7_008	W8_008
W7_010	W8_010
W7_011	W8_011
W7_012	W8_012
W7_013	W8_013
W7_014	W8_015
W7_015	W8_016
W7_016	W8_017
W7_017	W8_018

For testing we used the following subsets of the data:

Number of words total: 325,196

Year 87	Year 88
W7_009	W8_009